***2021***

****

**计算机组成原理 ·实验报告·**

|  |  |
| --- | --- |
| 学 院： | 计算机与信息学院 |
| 班 级： | 物联网工程19-2班 |
| 学 号： | 2019217769 |
| 姓 名： | 袁焕发 |
| 电 话： | 13012039276 |
| 邮 件： | [2635825108@qq.com](mailto:2635825108@qq.com) |
| 完成日期： | 2021-06-05 |
| 指导教师： | 阙 夏 |

目 录

[1 CPU部件实现之ALU和寄存器堆实验 2](#_Toc31600)

[1.1 设计要求 2](#_Toc3465)

[1.2 方案设计 2](#_Toc18832)

[1.3 实验步骤 8](#_Toc7672)

[1.4 故障与调试 13](#_Toc10489)

[1.5 仿真及分析 14](#_Toc19001)

[2 CPU部件实现之ALU和寄存器堆实验 16](#_Toc20195)

[2.1 设计要求 16](#_Toc14871)

[2.2 方案设计 16](#_Toc13714)

[2.3 实验步骤 24](#_Toc20935)

[2.4 故障与调试 26](#_Toc16231)

[2.5 仿真及分析 27](#_Toc9532)

[3 总结与心得 29](#_Toc18258)

[3.1 实验总结 29](#_Toc1861)

[3.2 实验心得 29](#_Toc12795)

[参考文献 30](#_Toc6812)

# CPU部件实现之ALU和寄存器堆实验

## 设计要求

### 使用 Verilog 完成 ALU 的设计,并编写测试仿真文件验证其正确性。

ALU 支持 16 位的加、减、与、或以及移位运算。

### 使用 Verilog 完成通用寄存器堆的设计，并编写测试仿真文件验证其正确性。

寄存器堆包含 8 个 16 位的寄存器；寄存器堆有两个读端口和一个写端。

## 方案设计

ALU模块设计如图 12.1所示

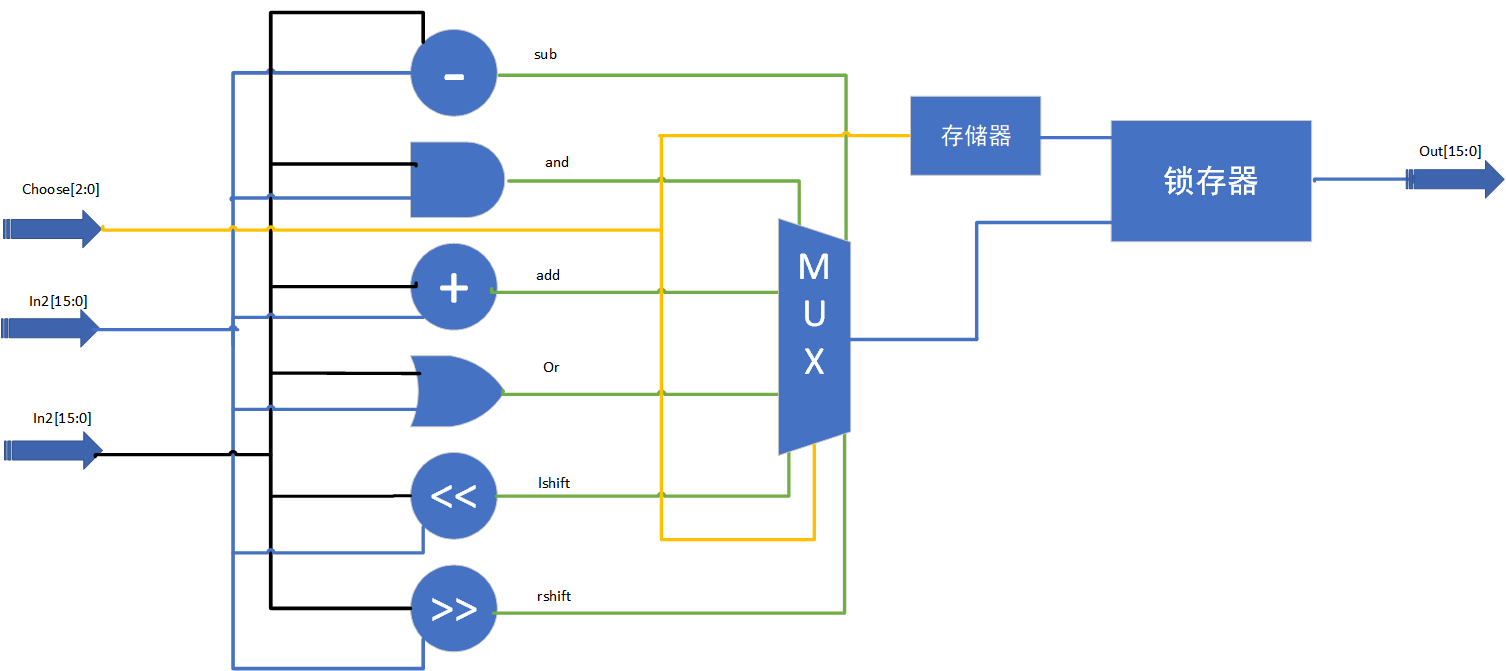


图 12.1 ALU模块设计图

寄存器堆设计如图 12.2所示

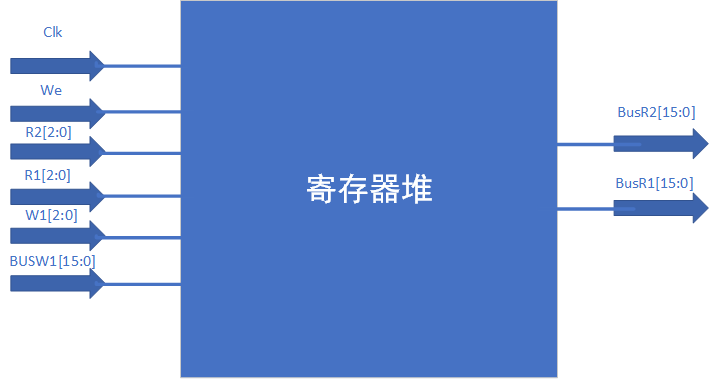


图 12.2寄存器堆模块设计图

### Verilog 关键代码实现

表 12.1 ALU模块功能描述

|  |  |
| --- | --- |
| 输入 | 16位输入信号in1、in2和3位选择信号choose |
| 输出 | 16位输出信号out |
| 功能 | 根据选择信号选择运算方式，根据输入数据，输出相应的运算值 |

ALU模块的功能描述如表 12.1 ALU模块功能描述所示;

ALU模块verilog代码如下：

module alu(

input wire [15:0] in1, in2,

input wire [2:0] choose,

output reg [15:0] out

);

always@\* begin

case(choose)

3'b000: out = in1+in2;//0

3'b001: out = in1-in2;//1

3'b010: out = in1&in2;//2

3'b011: out = in1|in2;//3

3'b100: out= in1<<in2;//4

3'b101: out= in1>>in2;//5

endcase

end

endmodule

表 12.2寄存器堆模块功能描述

|  |  |
| --- | --- |
| 输入 | 写信号WE,时钟信号clk，3位输入地址R1、R2、W1,16位输入总线busW1 |
| 输出 | 16位输出busR1, busR2 |
| 功能 | 设计8个16位寄存器组成寄存器堆 |

寄存器堆模块的功能描述如表 12.2寄存器堆模块功能描述所示;

寄存器堆verilog代码如下：

module register(

input wire WE, clk,//写，时钟信号

input wire [2:0] R1, R2, W1,//地址，用来决定读或写的寄存器

input wire [15:0] busW1,//写总线要写入寄存器堆的内容

output wire [15:0] busR1, busR2//读总线从寄存器堆读出的内容

);

reg [15:0] regfile[7:0];//寄存器16位，8单元寄存器堆

integer i;

initial begin // 初始化寄存器堆

for(i=0;i<8;i=i+1) //数值0,1,4,9,16，，，，49

regfile[i] = i\*i;

end

always@(posedge clk) begin//上升沿

if(WE == 1)//可写

regfile[W1] = busW1;//将总线内容写入寄存器

end

assign busR1 = regfile[R1];//分配总线到寄存器

assign busR2 = regfile[R2];

endmodule

### 测试文件(TestBench)关键代码描述

ALU模块测试文件关键代码如下：

module alutest();

wire [15:0] out;

reg [2:0] opcode;

reg [15:0] data\_a\_in, data\_b\_in;

initial begin

opcode = 0;//a+b

data\_a\_in = 0;

data\_b\_in = 0;

#10

data\_a\_in = 10;

data\_b\_in = 5;

//a+b

#10

opcode = 1;//a-b

data\_a\_in = 10;

data\_b\_in = 5;

#10

opcode = 2; // 0010&0001

data\_a\_in = 16;

data\_b\_in = 1;

#10

opcode = 3; // 0010or0001

data\_a\_in = 16;

data\_b\_in = 1;

#10

opcode = 4; // 0020左移2位

data\_a\_in = 32;

data\_b\_in = 2;

#10

opcode = 5;//0100右移2位

data\_a\_in =256;

data\_b\_in =2;

#100 $stop;

end

alu test(.in1(data\_a\_in),.in2(data\_b\_in),.choose(opcode), .out(out));

endmodule

寄存器模块测试文件关键代码如下：

module registertest();

reg clk,WE;

reg[2:0] RA,RB,RW;//地址，用来决定读或写的寄存器

reg[15:0] busW;//写总线，要写入寄存器堆的内容

wire[15:0] busA,busB;//读总线，从寄存器堆读出的内容

register test(

.clk(clk),

.WE(WE),

.R1(RA),

.R2(RB),

.W1(RW),

.busW1(busW),

.busR1(busA),

.busR2(busB)

);

initial begin

clk=0;

WE=0;

RA=3'd2;//002

RB=3'd4;//004

RW=3'd0;//000

busW = 16'd24;//0000，0000，0000，0018

#1000 $stop;

end

always #5 clk = ~clk;//5ns，反转

always begin

#20 WE = 1;//高电平读写

#20 WE = 0;

RW = RW + 1;//寄存器地址+1

busW = busW + 23;//数值+23

end

always begin

#10

RA = RA + 1;//读地址从2开始，+1，依次3,4,5....

RB = RB + 3;//读地址从4开始，+3，依次7,10.....

end

endmodule

## 实验步骤

1. Vivado创建工程

由于vivado安装步骤截图丢失，故从使用vivado创建工程开始演示。

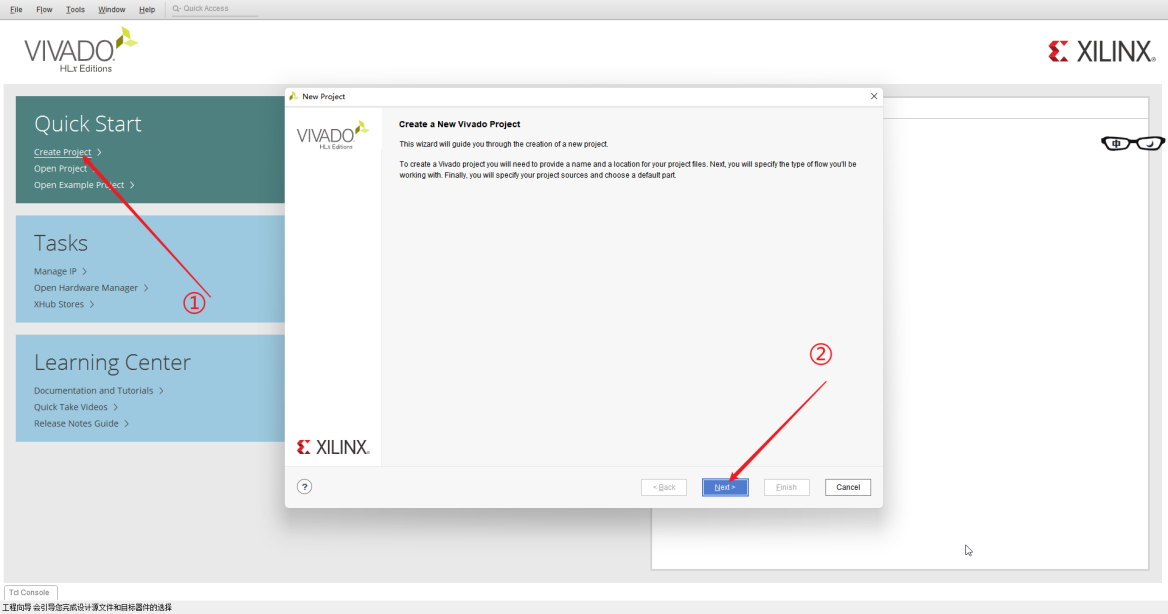


图 13.1

如图 13.1所示，按照图中编号顺序操作，接下来一直选择next直到看到图 4

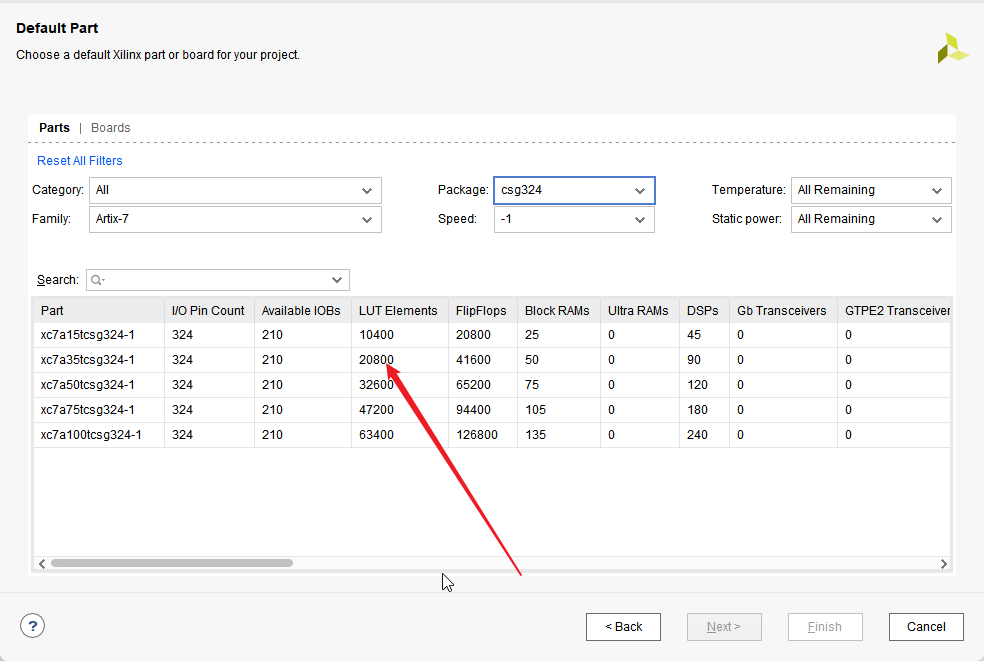


图 13.2

如图 13.2所示选择相应选项。

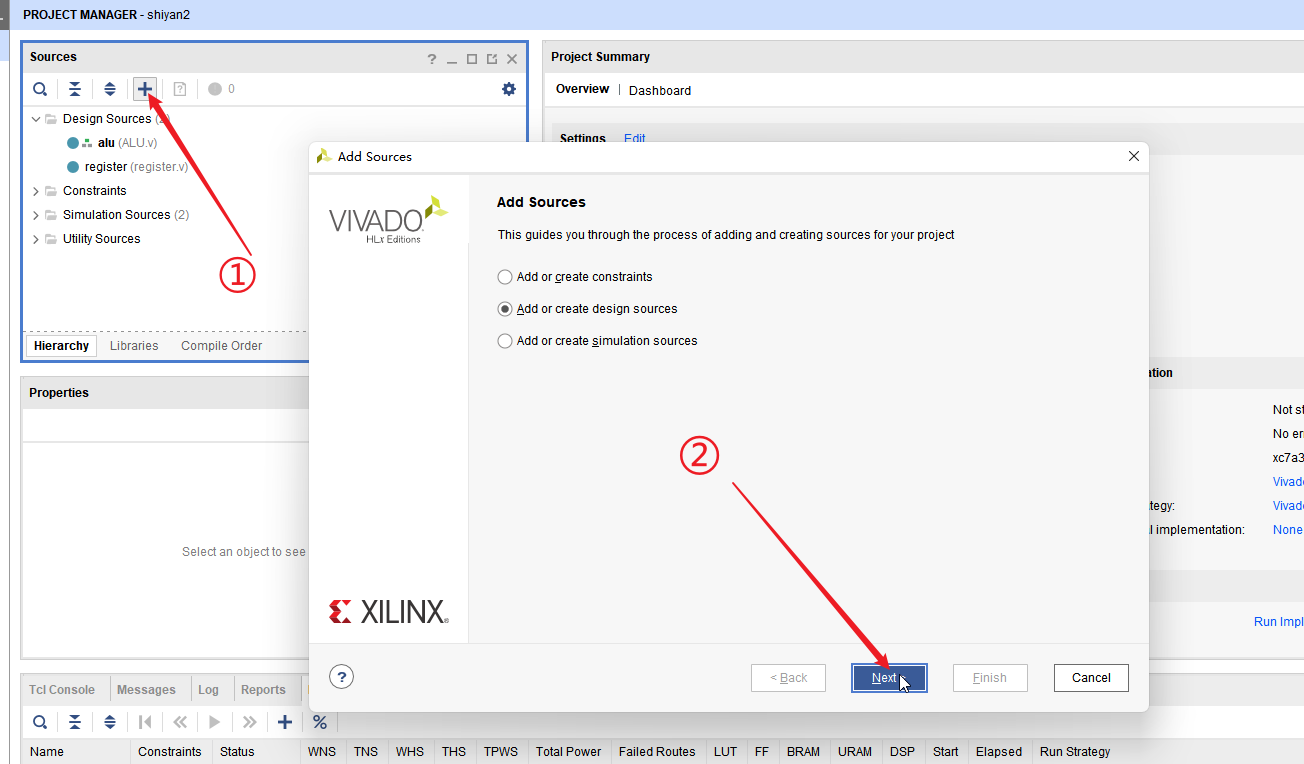


图 13.3

进入工程界面之后如图 13.3进行操作。

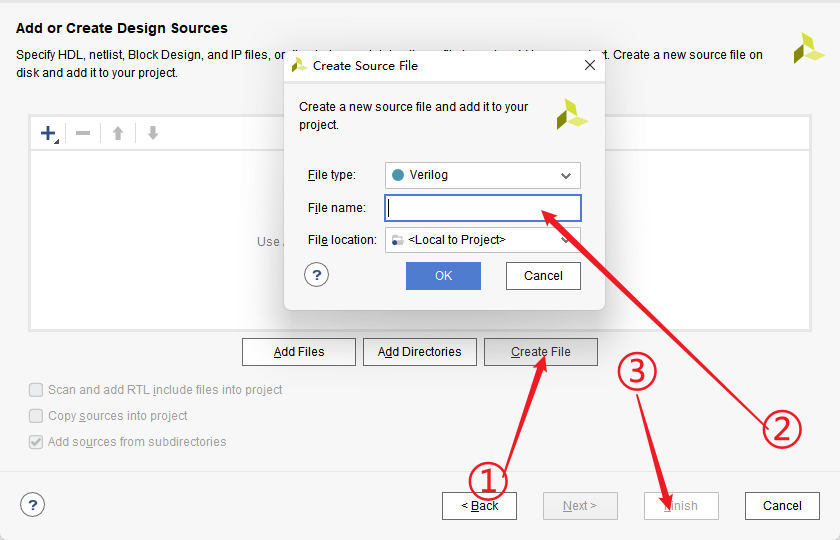


图 13.4

接下来如图 13.4所示。

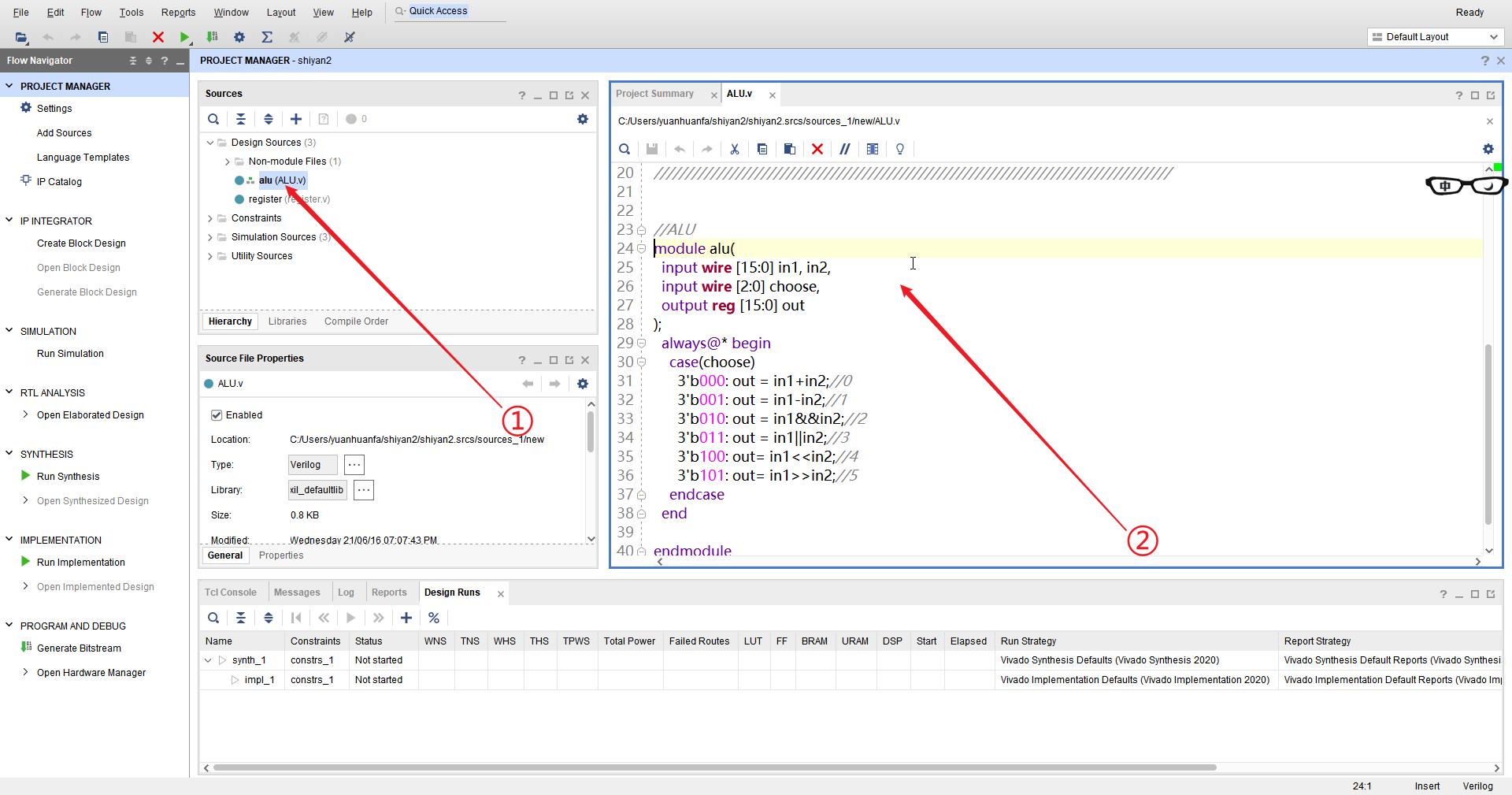


图 13.5

如图 13.5所示，进行verilog代码编辑。

1. 模拟运行

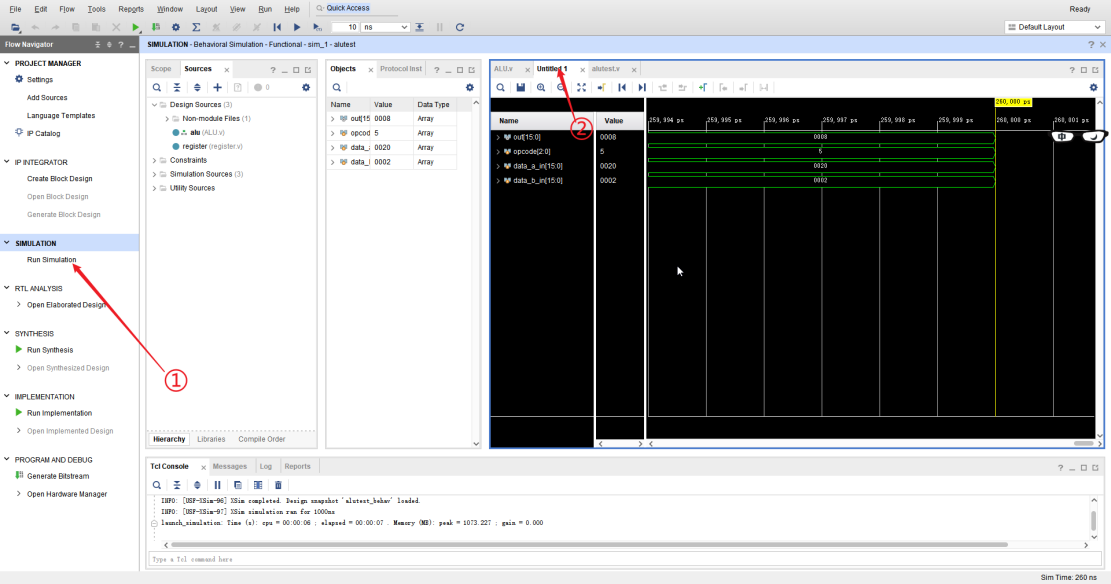


图 13.6

代码编辑完成之后，点击模拟，如图 13.6即可查看模拟情况，具体分析见仿真与分析部分。

1. 数据通路分析

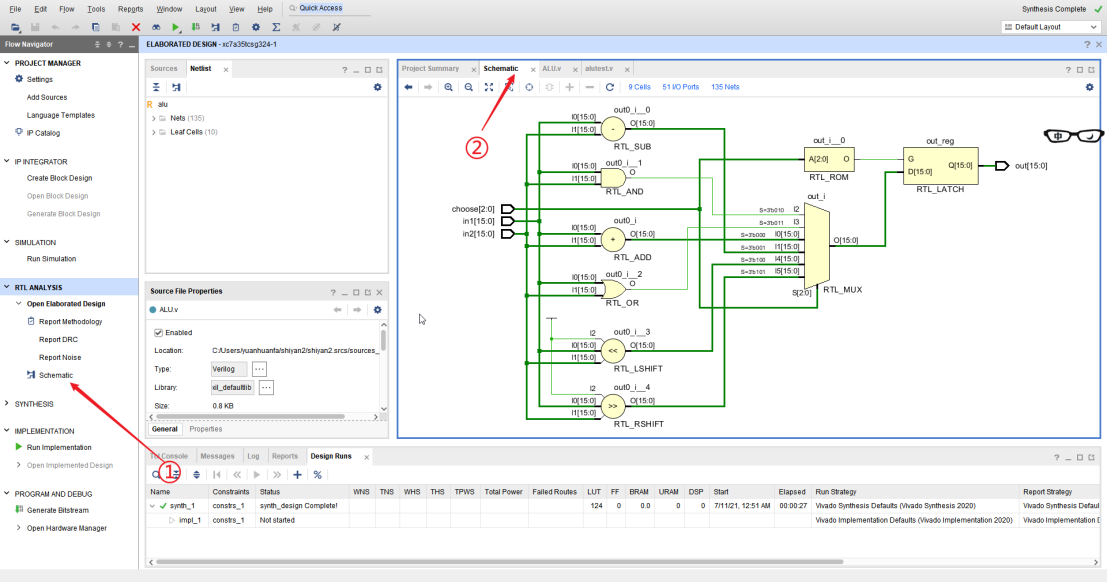


图 13.7

如图 13.7所示，可查看模块电路图，分析数据的流向和运算处理。

## 故障与调试

### 测试文件故障1

**故障现象：**

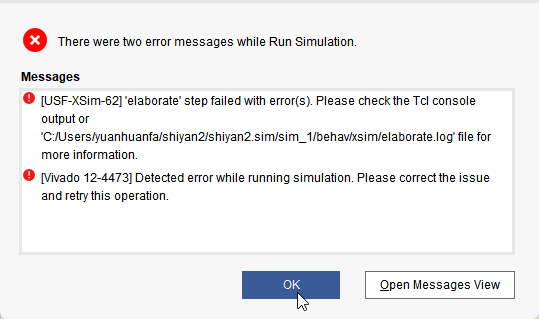


图 14.1

如图 14.1

**原因分析：**

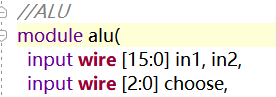
****

图 14.2

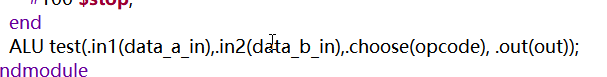


图 14.3

在测试文件中调用设计文件中的模块名时，名称与设计文件中模块名不一致，如图 14.2图 14.3。

**解决方案：**

将测试文件中 ALU test(.in1(data\_a\_in),.in2(data\_b\_in),.choose(opcode), .out(out));

改为 alu test(.in1(data\_a\_in),.in2(data\_b\_in),.choose(opcode), .out(out));

## 仿真及分析

### ALU仿真及分析

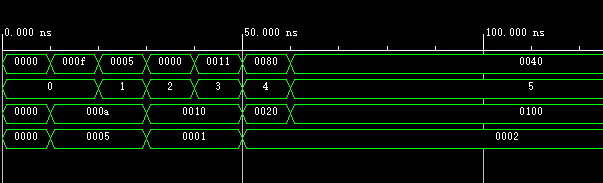


图 15.1

仿真图如图 15.1所示。

**分析：**

第一行为输出值，第二行为操作数，第三行和第四行为输入值，输入输出均为16进制。

操作数为000，为加法运算，两个输入值初始为0，延迟10ms后，in1=10，in2=5；

输出值由初始0变为000f=15；

操作数为001，为减法运算，in1=10，in2=5；输出值由000f变为0005；

操作数为010，为与运算，in1=0010，in2=5=0001，与运算之后为0000；

操作数为011，为或运算，in1=0010，in2=5=0001，或运算后为0011；

操作数为100，为左移位运算，32=0020，左移2位，相当于乘以4，为128=0080；

操作数为101，为右移位运算，256=0100，右移2位，相当于除以4，为64=0040。

### 寄存器堆仿真及分析

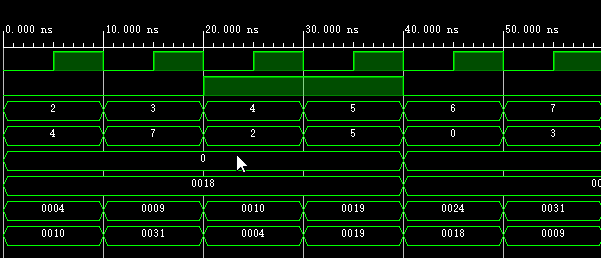


图 15.2

仿真图如图 15.2所示。

**分析：**

第一行为时钟信号，第二行为写信号，第三行、第四行为寄存器地址，第五行为写入地址，第六行为写入总线，第七行、第八行为寄存器输出总线。

寄存器地址RA初始值为2，RB初始值为4，写入地址RW初始值为0。

RA = RA + 1；RB = RB + 3，RW = RW + 1，busW = busW + 23;

寄存器初始化的值为0,1,4,9,···7\*7。

RA地址规律为2，3，4，5，6，7，0，1，······。读取数据为4、9、16、······

RB地址规律为4，7，2，5，0，3，6，1，4······。读取数据为16、49、4······

busW值依次递增23。

RW为0, 1，2，3，4，5，6，7时，且WE=1时，寄存器reg[i]分别写入busW数据（1+23），（24+23），······（1+23\*i），

BusA根据RA地址读入数据，4、9、16、25、36、49、24（由24替换原本数据0获得）、、4（还未被替换，之后数据也是）、

BusB根据RB地址读入数据，16、49、4、25、24（由24替换原本数据0获得）、9、36、47（由47替换原来数据1获得）、16、49。

# CPU部件实现之ALU和寄存器堆实验

## 设计要求

设计和实现一个支持加法指令的单周期 CPU。要求该加法指令（表示为 add r1，r2，r3）格式约定如下：

采用寄存器寻址，r1，r2，r3 为寄存器编号，r1 和 r2 存放两个源操作数，r3为目标寄存器，其功能为[r1] + [r2] -> r3；

指令字长 16 位，操作码和地址码字段分配如表 21.1所示：

表 21.1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 15 9 | 8 6 | | 5 3 | 2 0 |
| OpCode | r1 | r2 | | r3 |

## 方案设计

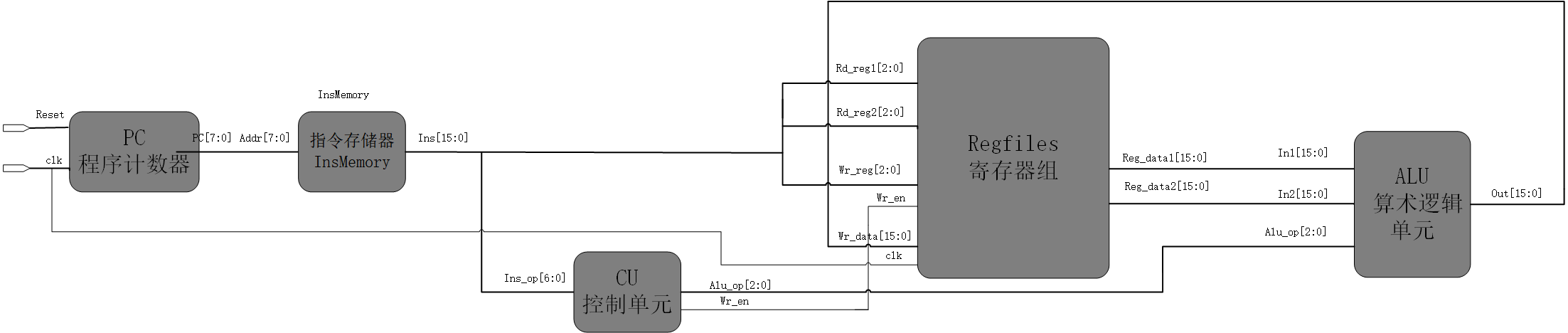


图 22.1

如图 22.1所示，相关模块之间的电路图。

### 设计思路

（1）Instruction Memory指令存储器，用作CPU指令的存储与输出

（2）ALU算术逻辑单元，用作CPU内部各项逻辑运算

（3）PC程序计数器，负责管理CPU指令地址，并保存当前指令地址

（4）Control Uint控制单元，用作控制各个指令相对应的控制信号

（5）Data Memory数据存储器，用作存储ALU的计算结果或寄存器组中的数据，或者输出其所存储的数据

（6）Register File 寄存器组，根据指令存储和读取相对应地址中的数据

### Verilog 关键代码实现

1. PC 模块

表 22.1

|  |  |
| --- | --- |
| 输入 | 时钟信号 clk、重置信号 rst |
| 输出 | 指令地址 pc（8 位） |
| 功能 | 每个时钟上升沿 PC 的值自动加 1，并输出 |

Verilog 关键代码：

module PC(

input wire clk, rst,

output reg [7:0] PC

);

always@(posedge clk) begin

if(rst == 1)

PC = 0;

else

PC =PC+ 1;

end

endmodule

1. 指令存储器模块

表 22.2

|  |  |
| --- | --- |
| 输入 | 8 位指令地址 Addr |
| 输出 | 16 位指令 Ins |
| 功能 | 存放待执行的指令（初始化），并根据地址输出指令 |

Verilog 关键代码：

module InsMem(

input wire [7:0] Addr,

output reg [15:0] Ins

);

reg[15:0] instructions[255:0];

//八个16位寄存器

integer i;

initial begin

for (i = 0; i < 256; i = i + 1) begin

instructions[i][2:0] = (i + 2) % 8;//结果寄存器

instructions[i][5:3] = (i + 1) % 8;//加数寄存器1

instructions[i][8:6] = i % 8;//加数寄存器2

instructions[i][15:9] = 0;

end

end

always@\* begin

Ins = instructions[Addr];//根据指令地址读取指令

end

endmodule

1. 寄存器堆

表 22.3

|  |  |
| --- | --- |
| 输入 | 时钟信号 clk、读写控制线 wr\_en、读寄存器编号 read\_reg1 和read\_reg2、写寄存器编号 write\_reg、写入数据 write\_data |
| 输出 | 对应两个读寄存器编号的寄存器值 reg1 和 reg2 |
| 功能 | 根据读寄存器编号给出对应寄存器的值；在写允许情况下，把写入端的数据在 clk 下降沿写到写寄存器编号对应的寄存器 |

Verilog 关键代码：

module RegFiles(

input wire clk,

input wire Wr\_en,

input wire [2:0]Rd\_reg1,Rd\_reg2,

input wire [2:0] Wr\_reg,

input wire [15:0] Wr\_data,

output reg [15:0] Reg\_data1,Reg\_data2

);

reg [15:0] regfile[7:0];

integer i;

initial begin

for(i = 0; i < 8; i = i + 1) begin//寄存器初始化

regfile[i] = i;

end

end

always@\* begin

Reg\_data1 <= regfile[Rd\_reg1];//非阻塞值并行执行

Reg\_data2 <= regfile[Rd\_reg2];

end

always@(negedge clk) begin

if(Wr\_en == 1)

regfile[Wr\_reg] = Wr\_data;//向寄存器写入数据

end

endmodule

1. ALU

表 22.4

|  |  |
| --- | --- |
| 输入 | 操作数 in1 和 in2、操作选择信号 alu\_op |
| 输出 | ALU 运算结果Out |
| 功能 | 根据操作选择信号计算 in1 和 in2 的运算结果Out |

Verilog 关键代码：

module ALU(

input wire [15:0] In1,In2,

input wire [2:0] Alu\_op,

output reg [15:0] Out

);

always@\*begin

case(Alu\_op)

3'b000:Out = In1+In2;//0

3'b001:Out = In1-In2;//1

3'b010:Out = In1&In2;//2

3'b011:Out = In1|In2;//3

3'b100:Out = In1<<In2;//4

3'b101:Out = In1>>In2;//5

endcase

end

endmodule

1. 控制单元

表 22.5

|  |  |
| --- | --- |
| 输入 | 指令（操作码） |
| 输出 | 寄存器堆的读写控制线 wr\_en、ALU 的操作选择信号alu\_op |
| 功能 | 根据当前指令功能对 wr\_en 和 alu\_op 赋值 |

Verilog 关键代码：

module CU(

input wire[6:0] Ins\_op,

output reg Wr\_en,

output reg[2:0] Alu\_op

);

always@\* begin

if (Ins\_op == 0) begin//控制写和操作数的选择

Wr\_en = 1;

Alu\_op = 0;

end

end

endmodule

1. CPU 顶层文件封装实现

Verilog 关键代码：

module CPU(

input wire clk, rst

);

wire wr;

wire [7:0] addr;

wire[15:0] ins;

wire [2:0] alu\_op;

wire [15:0] alu\_in1,alu\_in2,alu\_out;

PC pc(

.clk(clk), .rst(rst), .PC(addr)

);

InsMem insmem(

.Addr(addr), .Ins(ins)

);

CU cu(

.Ins\_op(ins[15:9]),

.Wr\_en(wr),

.Alu\_op(alu\_op)

);

RegFiles regfiles(

.clk(clk),

.Wr\_en(wr),

.Rd\_reg1(ins[8:6]),

.Rd\_reg2(ins[5:3]),

.Wr\_reg(ins[2:0]),

.Reg\_data1(alu\_in1),

.Reg\_data2(alu\_in2),

.Wr\_data(alu\_out)

);

ALU alu(

.In1(alu\_in1),

.In2(alu\_in2),

.Alu\_op(alu\_op),

.Out(alu\_out)

);

endmodule

### 测试模块关键代码

module CPU\_tb;

reg clk, rst;

always #1 clk = ~clk;

initial begin

clk = 1;

rst = 1;

#10 rst = 0;

#100 $stop;

end

CPU u\_cpu(

.clk(clk),

.rst(rst)

);

endmodule

## 实验步骤

创建工程步骤同实验一，故跳过。

1. 仿真模拟运行

由于本次实验文件过多，要检查文件顺序，如图 23.1，CPU以及它的测试文件必须位于它们所在组的最上方。

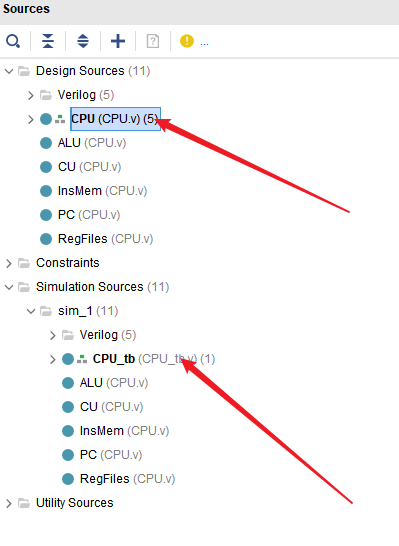


图 23.1

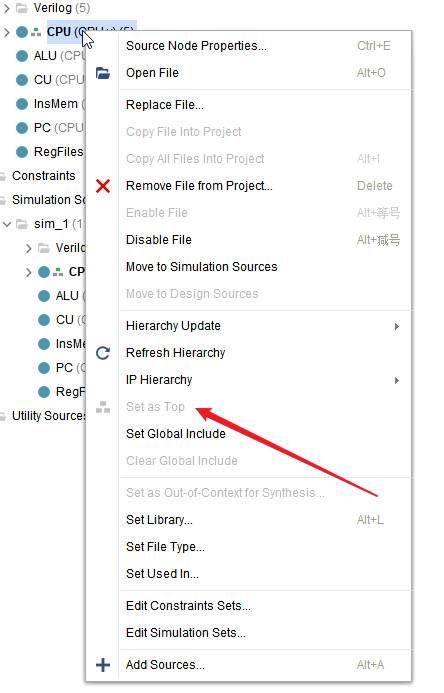


图 23.2

如果没有位于左上方，需要按照图 23.2进行设置

1. 模拟结果查看

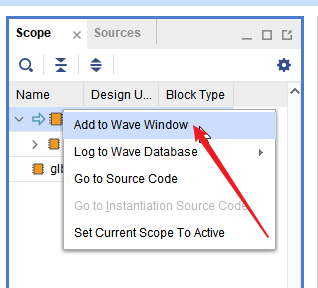


图 23.3

如果波形中没有想要的结果显示，如图 23.3进行操作，将波形添加到窗口查看。

1. 波形查看窗口调整

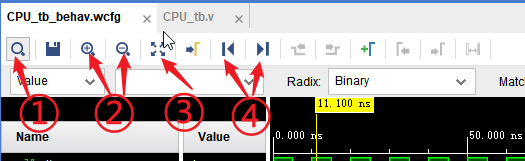


图 23.4

具体操作如图 23.4

①处可以根据名称查找想要查看的数值

②处可以放大缩小比例尺

③处可以填充满波形窗口

④处可以快速到达波形起始和结束

## 故障与调试

### XXX故障1

**故障现象：**

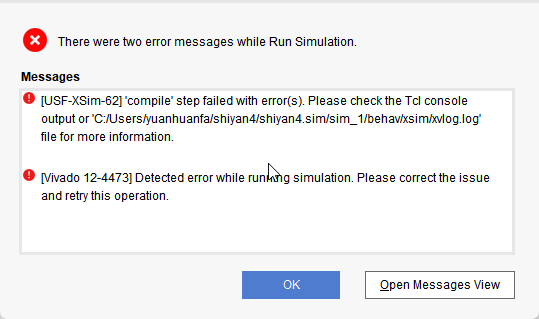


图 24.1

如图 24.1所示。

**原因分析：**

在删除某个模块，出现该错误，原因可能是磁盘文件没有清理干净。

**解决方案：**

最终查询解决方案无果，只能重新创建工程，将代码复制到新工程中，新工程再无报错

## 仿真及分析

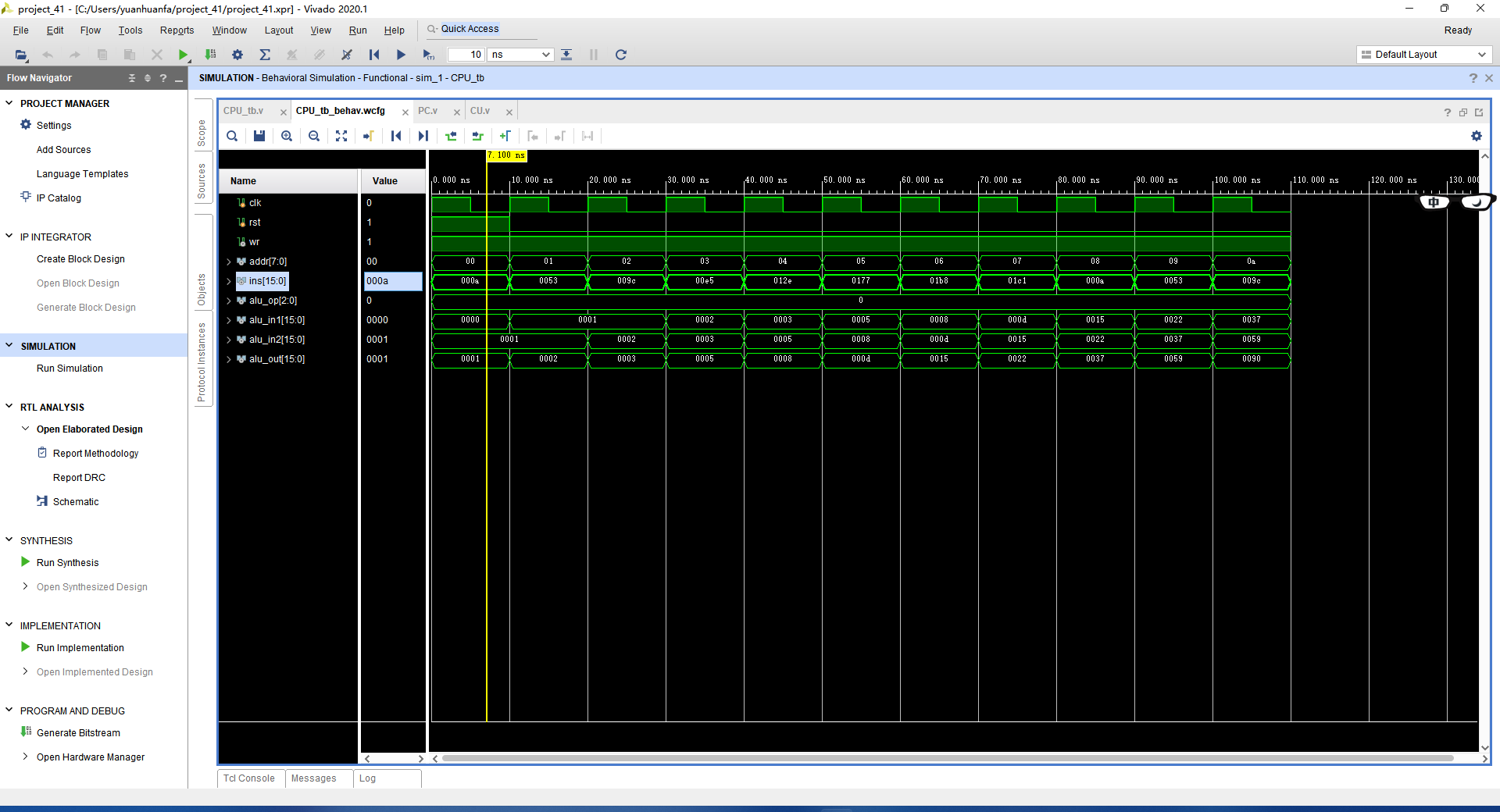


图 25.1

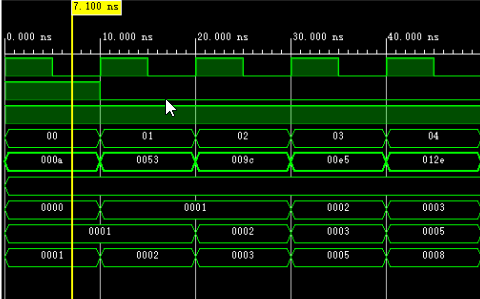


图 25.2

**分析:**

第一行为时钟信号，第二行为重置信号，第三行为可写信号，第四行为地址，第五行为存储指令，第六行为操作数，第七行和第八行为输入，第九行为输出。

地址每次增加1；

写入信号一直为1；

指令存储器，初始值i=0，之后i++；

[2:0] :(i + 2) % 8为结果寄存器，初始值为2=010，

[5:3] : i % 8为加数寄存器2，初始值为0=000，

[8:6] :(i + 2) % 8为结果寄存器，初始值为1=001，

最终指令存储器初始值为0000,0000,0000,1010（二进制）=000a（十六进制），之后的指令存储器结果以此类推；

第七行0000和第八行0001相加等于第九行0001；

寄存器堆初始值,0，1，2，3，4，5，6，7，用于ALU的输入。每次执行之后将前两次的输入值相加，获得第三个输入值，也就是将每次ALU 运算值写入到寄存器中，用作下一次ALU运算的一个输入值。

# 总结与心得

## 实验总结

本次实验主要完成了如下几点工作：

1. 完成方案总结

在实验一中，根据实验指导书给出的要求以及查阅资料，设计出了ALU和寄存器模块，实现了ALU 的简单架构，完成了两个输入值的加、减、与、或，左右位移运算。

在实验二中，在现有已经完成的实验基础上，将完成过的各个模块，组合设计出了一个单指令、单周期CPU，实现了模块的连接和CPU封装，完成了CPU单个指令的CPU运行过程。

1. 功能总结

实验一中，ALU具有的主要功能有接收两组输入信号，一组选择信号，并将ALU运算结果输出。

实验二中，CPU具有的主要功能有接收写入信号，重置信号，自动进行时钟信号翻转，调用ALU模块进行加法运算，使用PC进行程序计数，使用指令存储器存储运算指令，使用CU控制写信号和ALU运算选择信号，使用寄存器存储运算所需数据。

## 实验心得

通过这几次实验，我学会了使用Vivado软件进行模拟实验、查看模拟电路图以及分析模拟实验的运行结果，循序渐进的了解了CPU中每部分模块的结构组成以及工作原理，并通过代码方式将结构进行复现，以及为它们编写相应的测试程序。

实验过程中最大的收获就是在不断试错以及Debug中慢慢理清思路，寻找模块代码之间的关系；在和软件的各种报错中斗智斗勇，能找到解决办法就解决，解决不了的就重新创建工程；在模拟结果与预想结果不统一时，不断的删改代码，预设数据，寻找到思路错误的部分，学到了由大到小，按顺序进行分析解决问题的方法。

# 参考文献

1. DAVID A.PATTERSON(美).计算机组成与设计硬件/软件接口(原书第5版).北京:机械工业出版社.
2. David Money Harris(美).数字设计和计算机体系结构（第二版）. 机械工业出版社
3. 谭志虎,秦磊华,胡迪青.计算机组成原理实践教程.北京:清华大学出版社，2018年.
4. 秦磊华，吴非，莫正坤.计算机组成原理. 北京:清华大学出版社，2011年.
5. 袁春风编著. 计算机组成与系统结构. 北京:清华大学出版社，2011年.
6. 张晨曦，王志英. 计算机系统结构. 高等教育出版社，2008年.

|  |
| --- |
| 一、原创性声明 |
| 本人郑重声明本报告内容，是由作者本人独立完成的。有关观点、方法、数据和文献等的引用已在文中指出。除文中已注明引用的内容外，本报告不包含任何其他个人或集体已经公开发表的作品成果，不存在剽窃、抄袭行为。  特此声明！  **作者签字: 1I0K8X97}TVA}E0{VEP%110** |
| 二、对课程实验的学术评语（教师填写） |
|  |
| 三、对课程实验的评分（教师填写） |
| |  |  |  |  | | --- | --- | --- | --- | | 评分项目  （分值） | 报告撰写  （30分） | 课设过程  （70分） | 最终评定  （100分） | | 得分 |  |  |  | |
| **指导教师签字:** |