# 模块实验报告

|  |  |  |  |
| --- | --- | --- | --- |
| 姓名 | 叶福伦 | 学号 | 21051331 |
| 专业 | 计算机科学与技术 | 班级 | 21052313 |
| 指导教师 | 髙志刚 | 课程名称 | 计算机组成原理课程设计（甲） |
| 实验序号 | 04 | 实验名称 | 寄存器堆设计实验 |
| 实验时间 | 05.09 | 实验地点 | 1教223 |

# 成绩表

|  |  |  |
| --- | --- | --- |
| **考核项目** | **考查分项** | **得分** |
| **一、实验方案设计（30分）** | **实验目的与要求（5分）** |  |
| **模块构成与连接（15分）** |  |
| **板级验证方案（10分）** |  |
| **二、FPGA程序设计（20分）** | **程序源代码（15分）** |  |
| **管脚约束程序源代码（5分）** |  |
| **三、仿真实验（15分）** | **仿真代码（5分）** |  |
| **仿真波形及结果分析（10分）** |  |
| **四、板级实验（15分）** | **实验操作（5分）** |  |
| **结果记录（5分）** |  |
| **结果分析（5分）** |  |
| **五、思考与探索（10分）** | **拓展提升（10分）** |  |
| **格式（10分）** | **书写规范、图表清晰、表达清楚** |  |
| **合计** | |  |

# 一、实验方案设计

1. 实验目的与要求（5分）

学习寄存器堆的结构和数据传送原理，掌握三端口寄存器堆的设计方法；

掌握运算器的结构与工作原理，能将寄存器堆与暂存器及ALU进行正确连接，构成运算器

设计一个寄存器堆模块，要求符合RV32I的寄存器堆特征，仿真验证其功能。

构造一个运算器模块，引用实验2的带暂存器的多功能ALU模块、引用上述寄存器堆模块，并将它们连接起来。

针对使用的实验板卡，设计运算器模块的板级验证实验方案，编写顶层测试模块。

选择寄存器号和运算功能，验证你的运算器是否能正常工作，将实验结果记录到表格中，要求寄存器的读写操作和ALU的运算功能都被有效测试。

撰写实验报告，格式见附录，重点内容包括：

对仿真结果进行分析；

描述你设计的板级验证实验方案、模块结构与连接；

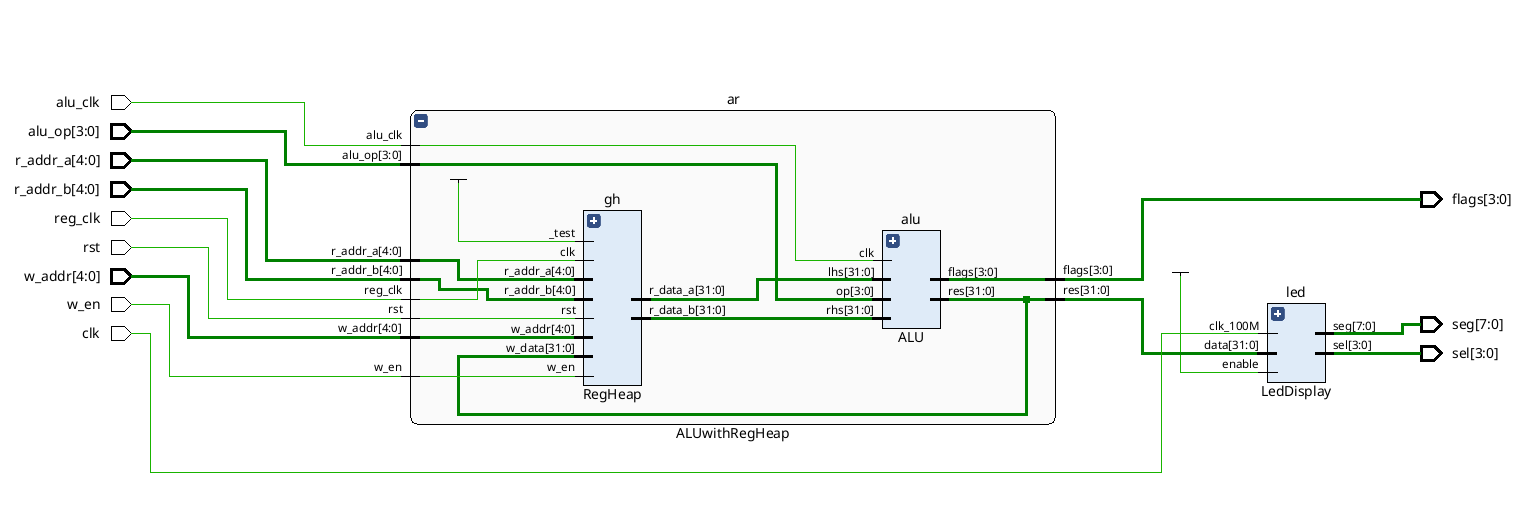
说明你的板级操作过程；

分析记录下来的板级实验结果、得到有效结论。

请力所能及回答或实践本实验的“思考与探索”部分。

1. 模块构成与连接（15分）

（请附图，并说明各模块的功能，以及模块间的连接关系。也可附开发工具中显示的电路模块连接图，加以说明。）



分为两个大模块。LedDisplay模块用于数码管输出，另外一个大模块是寄存器堆和ALU。RegHeap模块的r\_data\_a, r\_data\_b与ALU连接为ALU提供左右操作数。ALU的res端口数据返回w\_data，从而能够回写寄存器数据。三端口寄存器堆的三个地址由r\_addr\_a, r\_addr\_b, w\_addr输入提供。alu\_clk是ALU时钟，reg\_clk是寄存器堆时钟。rst是全局重置信号，w\_en是寄存器写允许信号，alu\_op指定运算

1. 板级验证方案（10分）

（说明顶层测试模块的输入/输出信号，及对应使用的I/O设备；可附图说明）

|  |  |  |  |
| --- | --- | --- | --- |
| 类型 | 信号名 | 使用的IO设备 | 说明 |
| input | alu\_clk | 按键 | alu时钟 |
| reg\_clk | 按键 | 寄存器堆时钟 |
| rst | 按键 | 全局重置信号 |
| clk | 全局时钟 | 驱动数码管 |
| alu\_op[3:0] | 一组逻辑开关 | 指定alu运算 |
| w\_en | 逻辑开关 | 控制寄存器写 |
| r\_data\_a[4:0] | 一组逻辑开关 | 指定读寄存器地址 |
| r\_data\_b[4:0] | 一组逻辑开关 | 指定读寄存器地址 |
| w\_data[4:0] | 一组逻辑开关 | 指定写寄存器地址 |
| output | flags[3:0] | 一组LED灯 | 标志寄存器 |
| sel[3:0] | 数码管 | 显示运算结果 |
| seg[7:0] |

# 二、FPGA程序设计

1. 实验程序源代码（25分）

（实验各个模块的代码，包含功能注释）

//alu\_op.v

//alu\_op编码

`define ALU\_ADD\_OP 4'b0000

`define ALU\_SLL\_OP 4'b0001

`define ALU\_SLT\_OP 4'b0010

`define ALU\_SLTU\_OP 4'b0011

`define ALU\_XOR\_OP 4'b0100

`define ALU\_SRL\_OP 4'b0101

`define ALU\_OR\_OP 4'b0110

`define ALU\_AND\_OP 4'b0111

`define ALU\_SUB\_OP 4'b1000

`define ALU\_SRA\_OP 4'b1001

`define ALU\_ADDU\_OP 4'b1010 //扩展不是标准实现,与add区别在标志寄存器的溢出和进位判断上

`define ALU\_SUBU\_OP 4'b1011 //扩展不是标准实现,与sub区别在标志寄存器的溢出和进位判断上

`define ALU\_UNVALID\_OP 4'b1111

//ALU.v

`include "../define/alu\_op.v"

module ALU (

input [31:0] lhs,

input [31:0] rhs,

input clk,

input [3:0] op,

output reg [31:0] res,

output reg [3:0] flags // [is\_zero, is\_positive, carry, overflow]

);

reg \_c;

always @(negedge clk) begin

case(op)

`ALU\_ADD\_OP :begin { \_c, res } = lhs + rhs; flags[0] = \_c ^ res[31] ^ lhs[31] ^ rhs[31]; end

`ALU\_SLL\_OP :begin res = lhs << rhs; end

`ALU\_SLT\_OP :begin res = $signed(lhs) < $signed(rhs); end

`ALU\_SLTU\_OP:begin res = lhs < rhs; end

`ALU\_XOR\_OP :begin res = lhs ^ rhs; end

`ALU\_SRL\_OP :begin res = lhs >> rhs; end

`ALU\_OR\_OP :begin res = lhs | rhs; end

`ALU\_AND\_OP :begin res = lhs & rhs; end

`ALU\_SUB\_OP :begin { \_c, res } = lhs - rhs; flags[0] = \_c ^ res[31] ^ lhs[31] ^ rhs[31]; end

`ALU\_SRA\_OP :begin res = $signed(lhs) >>> rhs; end

`ALU\_ADDU\_OP:begin { flags[1], res } = lhs + rhs; end

`ALU\_SUBU\_OP:begin { flags[1], res } = lhs - rhs; end

default: begin res = 0; end

endcase

flags[3] <= res == 32'h0000\_0000 ? 1 : 0;

flags[2] <= res[31];

end

endmodule

//ReagHeap.v

module RegHeap(

input clk,

input w\_en,

input rst,

input \_test,

input [4:0] r\_addr\_a,

output reg [31:0] r\_data\_a,

input [4:0] r\_addr\_b,

output reg [31:0] r\_data\_b,

input [4:0] w\_addr,

input [31:0] w\_data

);

reg [31:0]regs[31:0];

integer i;

always@(negedge clk or posedge rst) begin

if(rst) begin

if(\_test) begin

regs[0] <= 0;

for(i=1;i<32;i=i+1) regs[i] <= 1 << i;

end else begin

for(i=0;i<32;i=i+1) regs[i] <= 0;

end

end else begin

if(w\_en == 1 && w\_addr != 0) begin

regs[w\_addr] <= w\_data;

end

end

end

always@(negedge clk)begin

r\_data\_a <= regs[r\_addr\_a];

r\_data\_b <= regs[r\_addr\_b];

end

endmodule

//ALUwithRegHeap.v

`include "../03-ALU/ALU.v"

module ALUwithRegHeap(

input rst,

input w\_en,

input [4:0] r\_addr\_a, r\_addr\_b, w\_addr,

input [3:0] alu\_op,

input reg\_clk, alu\_clk,

output [31:0] res,

output [3:0] flags

);

wire [31:0] r\_data\_a, r\_data\_b;

wire [31:0] w\_data;

assign w\_data = res;

RegHeap gh(

reg\_clk,

w\_en,

rst,

1'b1,

r\_addr\_a,

r\_data\_a,

r\_addr\_b,

r\_data\_b,

w\_addr,

w\_data

);

ALU alu(

r\_data\_a,

r\_data\_b,

alu\_clk,

alu\_op,

res,

flags

);

endmodule

//ALUwithRegHeap.top.v

`include "../include/LedDisplay.v"

module ALUwithRegHeapToppper(

input [4:0] r\_addr\_a, r\_addr\_b, w\_addr, //逻辑开关

input [3:0] alu\_op, //逻辑开关

input w\_en,

input reg\_clk, alu\_clk, //用按键模拟

input rst, //用按键模拟

input clk,

output [3:0] flags,

output [3:0] sel,

output [7:0] seg

);

wire [31:0] res;

ALUwithRegHeap ar(

rst,

w\_en,

r\_addr\_a,

r\_addr\_b,

w\_addr,

alu\_op,

reg\_clk,

alu\_clk,

res,

flags

);

LedDisplay led(

clk,

res,

1'b1,

sel,

seg

);

endmodule

1. FPGA管脚约束代码（5分）

（引脚约束文件的内容，备注使用的具体设备）

r\_addr\_a[4:0], r\_addr\_b[4:0], w\_addr[4:0]使用逻辑开关

alu\_op[3:0]使用逻辑开关

w\_en使用逻辑开关

reg\_clk, alu\_clk使用按键模拟

rst使用按键

clk使用全局时钟

sel[3:0]， seg[7:0]共同驱动数码管

set\_property BITSTREAM.GENERAL.COMPRESS TRUE [current\_design]

set\_property -dict {IOSTANDARD LVCMOS18 PACKAGE\_PIN H4} [get\_ports clk]

set\_property IOSTANDARD LVCMOS18 [get\_ports {seg[7]}]

set\_property IOSTANDARD LVCMOS18 [get\_ports {seg[6]}]

set\_property IOSTANDARD LVCMOS18 [get\_ports {seg[5]}]

set\_property IOSTANDARD LVCMOS18 [get\_ports {seg[4]}]

set\_property IOSTANDARD LVCMOS18 [get\_ports {seg[3]}]

set\_property IOSTANDARD LVCMOS18 [get\_ports {seg[2]}]

set\_property IOSTANDARD LVCMOS18 [get\_ports {seg[1]}]

set\_property IOSTANDARD LVCMOS18 [get\_ports {seg[0]}]

set\_property IOSTANDARD LVCMOS18 [get\_ports {sel[3]}]

set\_property IOSTANDARD LVCMOS18 [get\_ports {sel[2]}]

set\_property IOSTANDARD LVCMOS18 [get\_ports {sel[1]}]

set\_property IOSTANDARD LVCMOS18 [get\_ports {sel[0]}]

set\_property PACKAGE\_PIN H19 [get\_ports {seg[7]}]

set\_property PACKAGE\_PIN G20 [get\_ports {seg[6]}]

set\_property PACKAGE\_PIN J22 [get\_ports {seg[5]}]

set\_property PACKAGE\_PIN K22 [get\_ports {seg[4]}]

set\_property PACKAGE\_PIN K21 [get\_ports {seg[3]}]

set\_property PACKAGE\_PIN H20 [get\_ports {seg[2]}]

set\_property PACKAGE\_PIN H22 [get\_ports {seg[1]}]

set\_property PACKAGE\_PIN J21 [get\_ports {seg[0]}]

set\_property PACKAGE\_PIN L21 [get\_ports {sel[3]}]

set\_property PACKAGE\_PIN M22 [get\_ports {sel[2]}]

set\_property PACKAGE\_PIN M21 [get\_ports {sel[1]}]

set\_property PACKAGE\_PIN N22 [get\_ports {sel[0]}]

set\_property IOSTANDARD LVCMOS18 [get\_ports alu\_clk]

set\_property IOSTANDARD LVCMOS18 [get\_ports reg\_clk]

set\_property IOSTANDARD LVCMOS18 [get\_ports {alu\_op[3]}]

set\_property IOSTANDARD LVCMOS18 [get\_ports {alu\_op[2]}]

set\_property IOSTANDARD LVCMOS18 [get\_ports {alu\_op[1]}]

set\_property IOSTANDARD LVCMOS18 [get\_ports {alu\_op[0]}]

set\_property IOSTANDARD LVCMOS18 [get\_ports {r\_addr\_a[4]}]

set\_property IOSTANDARD LVCMOS18 [get\_ports {r\_addr\_a[3]}]

set\_property IOSTANDARD LVCMOS18 [get\_ports {r\_addr\_a[2]}]

set\_property IOSTANDARD LVCMOS18 [get\_ports {r\_addr\_a[1]}]

set\_property IOSTANDARD LVCMOS18 [get\_ports {r\_addr\_a[0]}]

set\_property IOSTANDARD LVCMOS18 [get\_ports {r\_addr\_b[4]}]

set\_property IOSTANDARD LVCMOS18 [get\_ports {r\_addr\_b[3]}]

set\_property IOSTANDARD LVCMOS18 [get\_ports {r\_addr\_b[2]}]

set\_property IOSTANDARD LVCMOS18 [get\_ports {r\_addr\_b[1]}]

set\_property IOSTANDARD LVCMOS18 [get\_ports {r\_addr\_b[0]}]

set\_property IOSTANDARD LVCMOS18 [get\_ports {w\_addr[4]}]

set\_property IOSTANDARD LVCMOS18 [get\_ports {w\_addr[3]}]

set\_property IOSTANDARD LVCMOS18 [get\_ports {w\_addr[2]}]

set\_property IOSTANDARD LVCMOS18 [get\_ports {w\_addr[1]}]

set\_property IOSTANDARD LVCMOS18 [get\_ports {w\_addr[0]}]

set\_property PULLDOWN true [get\_ports {w\_addr[4]}]

set\_property PULLDOWN true [get\_ports {w\_addr[3]}]

set\_property PULLDOWN true [get\_ports {w\_addr[2]}]

set\_property PULLDOWN true [get\_ports {w\_addr[1]}]

set\_property PULLDOWN true [get\_ports {w\_addr[0]}]

set\_property PACKAGE\_PIN R4 [get\_ports reg\_clk]

set\_property PACKAGE\_PIN AA4 [get\_ports alu\_clk]

set\_property PACKAGE\_PIN T3 [get\_ports {r\_addr\_a[4]}]

set\_property PACKAGE\_PIN U3 [get\_ports {r\_addr\_a[3]}]

set\_property PACKAGE\_PIN T4 [get\_ports {r\_addr\_a[2]}]

set\_property PACKAGE\_PIN V3 [get\_ports {r\_addr\_a[1]}]

set\_property PACKAGE\_PIN V4 [get\_ports {r\_addr\_a[0]}]

set\_property PACKAGE\_PIN W4 [get\_ports {r\_addr\_b[4]}]

set\_property PACKAGE\_PIN Y4 [get\_ports {r\_addr\_b[3]}]

set\_property PACKAGE\_PIN Y6 [get\_ports {r\_addr\_b[2]}]

set\_property PACKAGE\_PIN W7 [get\_ports {r\_addr\_b[1]}]

set\_property PACKAGE\_PIN Y8 [get\_ports {r\_addr\_b[0]}]

set\_property PACKAGE\_PIN Y7 [get\_ports {alu\_op[3]}]

set\_property PACKAGE\_PIN T1 [get\_ports {alu\_op[2]}]

set\_property PACKAGE\_PIN U1 [get\_ports {alu\_op[1]}]

set\_property PACKAGE\_PIN U2 [get\_ports {alu\_op[0]}]

set\_property PACKAGE\_PIN W1 [get\_ports {w\_addr[4]}]

set\_property PACKAGE\_PIN W2 [get\_ports {w\_addr[3]}]

set\_property PACKAGE\_PIN Y1 [get\_ports {w\_addr[2]}]

set\_property PACKAGE\_PIN AA1 [get\_ports {w\_addr[1]}]

set\_property PACKAGE\_PIN V2 [get\_ports {w\_addr[0]}]

set\_property PULLDOWN true [get\_ports {alu\_op[3]}]

set\_property PULLDOWN true [get\_ports {alu\_op[2]}]

set\_property PULLDOWN true [get\_ports {alu\_op[1]}]

set\_property PULLDOWN true [get\_ports {alu\_op[0]}]

set\_property PULLDOWN true [get\_ports {r\_addr\_a[3]}]

set\_property PULLDOWN true [get\_ports {r\_addr\_a[4]}]

set\_property PULLDOWN true [get\_ports {r\_addr\_a[2]}]

set\_property PULLDOWN true [get\_ports {r\_addr\_a[1]}]

set\_property PULLDOWN true [get\_ports {r\_addr\_a[0]}]

set\_property PULLDOWN true [get\_ports {r\_addr\_b[4]}]

set\_property PULLDOWN true [get\_ports {r\_addr\_b[3]}]

set\_property PULLDOWN true [get\_ports {r\_addr\_b[2]}]

set\_property PULLDOWN true [get\_ports {r\_addr\_b[1]}]

set\_property PULLDOWN true [get\_ports {r\_addr\_b[0]}]

set\_property CLOCK\_DEDICATED\_ROUTE FALSE [get\_nets alu\_clk\_IBUF]

set\_property IOSTANDARD LVCMOS18 [get\_ports {flags[3]}]

set\_property IOSTANDARD LVCMOS18 [get\_ports {flags[2]}]

set\_property IOSTANDARD LVCMOS18 [get\_ports {flags[1]}]

set\_property IOSTANDARD LVCMOS18 [get\_ports {flags[0]}]

set\_property PACKAGE\_PIN R1 [get\_ports {flags[3]}]

set\_property PACKAGE\_PIN P2 [get\_ports {flags[2]}]

set\_property PACKAGE\_PIN P1 [get\_ports {flags[1]}]

set\_property PACKAGE\_PIN N2 [get\_ports {flags[0]}]

set\_property PACKAGE\_PIN AB6 [get\_ports rst]

set\_property PACKAGE\_PIN Y2 [get\_ports w\_en]

set\_property IOSTANDARD LVCMOS18 [get\_ports w\_en]

set\_property IOSTANDARD LVCMOS18 [get\_ports rst]

set\_property PULLDOWN true [get\_ports w\_en]

# 三、仿真实验

（只对CPU模块仿真，无需对板级验证的顶层测试模块仿真）

1. 仿真代码（5分）

（含仿真源代码、仿真验证方案）

`timescale 1ns/1ps

module RegHeapTester();

reg clk;

reg w\_en;

reg rst;

reg [4:0] r\_addr\_a;

wire [31:0] r\_data\_a;

reg [4:0] r\_addr\_b;

wire [31:0] r\_data\_b;

reg [4:0] w\_addr;

reg [31:0] w\_data;

RegHeap gh(

clk,

w\_en,

rst,

1'b1,

r\_addr\_a,

r\_data\_a,

r\_addr\_b,

r\_data\_b,

w\_addr,

w\_data

);

parameter PERIOD = 10;

initial begin

clk = 0;

forever begin

#(PERIOD/2) clk = ~clk;

end

end

initial begin

rst = 0;

w\_en = 0;

r\_addr\_a = 5'b0;

r\_addr\_b = 5'b0;

w\_addr = 5'b0;

w\_data = 5'b0;

#10

rst = 1;

#10

rst = 0;

#10

r\_addr\_a = 5'b00000;

r\_addr\_b = 5'b00001;

#70

w\_en = 1;

w\_addr = 5'b00000;

w\_data = 32'hffff\_ffff;

#50

w\_addr = 5'b00001;

w\_data = 32'hffff\_ffff;

#50

w\_en = 0;

r\_addr\_a = 5'b00010;

w\_addr = 5'b00010;

w\_data = 32'hffff\_ffff;

#50

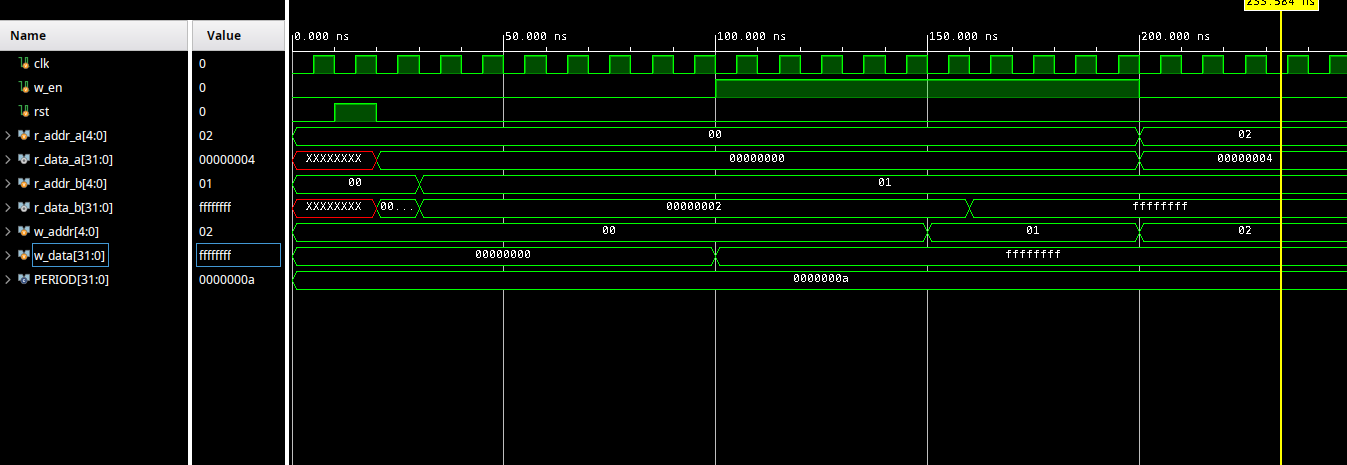
$finish;

end

endmodule

1. 仿真波形及结果分析（15分）

（仿真波形截图，并对仿真波形进行分析）

如图，前30ns给寄存器复位，下一个阶段让r\_addr\_a指向x0，r\_addr\_b指向x1, 让w\_en置为1，w\_addr指向x0，写入0xFFFF\_FFFF,输出r\_data\_a = 0x0, r\_data\_b = 1 << 1 = 0x2说明rst复位是有效的且证明了x0的只读性

下个阶段让w\_addr指向x1，写入0xFFFF\_FFFF，r\_data\_b在下一个时钟下降沿变为0xFFFF\_FFFF，说明写入有效。

下一个阶段让w\_en=0, 让r\_addr\_a 指向x2，w\_addr指向x2，写入0xFFFF\_FFFF发现r\_addr\_a并没有变化，说明写允许信号发挥了作用阻止了数据写入

# 由仿真结果可知，该代码大致符合预期结果

# 四、板级实验

1. 板级实验操作说明（5分）

1）按rst\_n键复位；

2）拨动开关输入A口和B口地址，按下时钟键clk\_RR，将读出的A口数据和B口数据分别打入暂存器A和B；

3）拨动开关选择运算功能ALU\_OP，按下时钟键clk\_F，保存结果到F；

4）拨动开关输入写端口的地址，拨Reg\_Write开关=1，按下时钟键clk\_WB，将暂存器F中的运算结果写入指定寄存器；

5）拨动开关选择输出的数据，观察LED灯或者数码管，记录实验结果到表1格中，并分析实验结果是否正确。

1. 板级实验结果记录（5分）

（以表格形式展现，具体见教材上实验结果记录表）

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| r\_addr\_a | r\_addr\_b | w\_addr | ALU\_OP | 功能 | w\_en | F | FR | 目的 |
| 00000 | 00001 | 00000 | 0000 | 做加法运算 | 1 | 0000\_0002 | 0000 | 1. 尝试写入X0  2. 验证加法有效性 |
| 00000 | 00000 |  | 0000 | 做加法运算 | 0 | 0000\_0000 | 1000 | 查看X0是否有被写入 |
| 00001 | 00010 | 00011 | 0110 | 或运算 | 1 | 0000\_0006 | 0000 | 尝试写入X3 |
| 00011 | 00000 |  | 0000 | 加法 | 0 | 0000\_0006 | 0000 | 查看X3是否又被写入 |
| 00011 | 00001 | 00011 | 0000 | 加法 | 1 | 0000\_0008 | 0000 | 尝试给X3加上2 |
| 00011 | 00000 |  | 0000 | 加法 | 0 | 0000\_0008 | 0000 | 读取X3的值判断是否加2 |

1. 结果分析与结论（5分）

（分析实验结果，给出实验结论）

由上可知，寄存器的X0的不可写入性，写控制信号的有效性，寄存器的读写功能都是正常的

# 五、思考与探索

（10分）

1. 问题与解决方案

（整个实验过程中发生了什么问题？你是如何解决的。）

报错显示端口类型不匹配：实例化模块时的变量顺序排错了，根据声明修改后把问题解决

1. 思考题

（力所能及，尝试实践或回答教材上的思考与探索题目，至少完成1道）

1. 让一个读地址和写地址指向同一个寄存器，查看对应的读数据在下一个时钟下降沿是否变为对应的值

2. r\_addr\_a[4:0]一组，r\_addr\_b[4:0]一组，w\_addr[4:0]一组，reg\_write和alu\_op[3:0]一组共四组，用两个逻辑开关加以区分，并另外要一个按键来输入打入脉冲。

因此总共要7个逻辑开关和额外一个按钮

4. 新增一个复用器，让alu的右操作数端口从imm32和B寄存器中选择一个输入数据。alu根据复用信号从imm32和B寄存器中选择一个作为右操作数参与和左操作数的运算然后输出到F寄存器。

# 六、实验心得体会、意见建议

（如果是小组合作，请给出具体分工、分别给出心得体会）

在完成RISC-V寄存器堆实验的过程中，我学习了如何在RISC-V处理器中实现寄存器堆，并且掌握了一些基本的操作，如读取寄存器的值、向寄存器中写入数据等。通过这个实验，我深入了解了寄存器堆的作用和实现原理，同时也更好地理解了计算机组成原理中的一些基本概念和操作。

在实验中，我遇到了一些问题，例如在实现读取寄存器的值时，需要使用一个5位的二进制数来表示寄存器编号。这让我有些困惑，但最终我通过仔细阅读实验文档和参考资料，成功地解决了这个问题。

另外，为了测试我的寄存器堆实现是否正确，我编写了一些测试代码，包括读取寄存器的值、向寄存器中写入数据、在寄存器之间进行数据传输等操作。这些测试代码帮助我验证了我的寄存器堆实现的正确性，并且让我对RISC-V处理器的操作更加熟悉。

总的来说，这个实验是一次非常有意义的学习经历，它让我更好地理解了计算机组成原理中的一些重要概念和操作，并且让我更加熟悉了RISC-V处理器的结构和运行方式。希望这个实验对我的学习和成长有所帮助！