实验报告

2021 年 5 月 12 日 成绩：

姓名	刘合	学号		班级	
专业	计算机科学与技术		课程名称	计算机组成原理课程设计	
任课老师		指导老师		机位号	
实验序号	3	实验名称	寄存器堆与运算器设计		
实验时间	2021.5.12	实验地点		实验设备号	

一、实验目的与要求

1.实验目的：

1. 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；

2. 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。

2.实验要求：

1. 设计一个寄存器堆模块(如图要求符合 RV32I 的寄存器堆特征,仿真验证其功能 。

2. 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，如图所示。

3. 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。

对于本实验，如表 11.7 所示，可将三个端口的寄存器地址及 ALU 的功能配置到逻辑开关上，可以根据需要将运算结果 F 配置到显示灯或者数码管上，复位及时钟信号可以配置到按键上。

表11.7 实验 3 信号配置表

	信号名称	配置设备管脚
输入	R_Addr_A[4:0]	5 个逻辑开关
	R_Addr_B[4:0]	5 个逻辑开关
	W_Addr[4:0]	5 个逻辑开关
	ALU_OP[3:0]	4 个逻辑开关
	Reg_Write	1 个逻辑开关
时钟	rst_n	按键
	clk_RR	按键
	clk_F	按键
	clk_WB	按键
输出	F[31:0]	8 个数码管
	FR[3:0]	4 个 LED 灯

4. 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，要求寄存器的读写操作和 ALU 的运算功能都被有效测试 。

5. 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。

3. 实验步骤

1. 新建一个工程新建一个寄存器堆模块，如图所示 注意 x0 为零寄存器 。

2. 编写激励代码，仿真验证寄存器模块功能；分析仿真结果，确保读写操作正确。

3. 将实验 2 的带暂存器的多功能 ALU 模块的.v 文件（包括子模块的.v 文件）拷贝到本工程的源代码目录下，并将它们加入工程。

4. 新建一个运算器模块，引用实验 2 的带暂存器的多功能 ALU 模块，及上述寄存器堆模块，各定义一个实例，将它们连接起来，如图所示。如果有必要，可进行仿真验证。为方便进行有效的测试，可以修改寄存器模块的初始化操作，不再全部清零而是初始化相关寄存器为典型的测试数据。

5. 设计运算器模块板级验证的实验方案，然后据此编写一个顶层测试模块。

6. 可以依据实际需要，对顶层测试模块进行仿真测试，或者直接进入管脚约束环节。

7. 新建管脚约束文件，依据 表 11.7 的提示和 设计 的 板级验证实验方案，进行相应的引脚配置。

8. 生成 *.bit 文件，下载到实验设备的 FPGA 芯片中。

9. 板级实验：按照你所设计的实验方案，操作输入设备、观察输出设备，一般过程为：

（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 11.5 中，并分析实验结果是否正确。

注意：在板级实验中，要保证能观察或者经分析后确认：运算结果已经更新了目的寄存器的值。

二、实验设计与程序代码

1、 模块设计说明

程序包含了 3 个模块：

1.运算器顶层模块：有序调用其他模块的程序；

2. ALU 顶层模块：实现 10 个计算功能逻辑的模块；

3.寄存器堆模块：实现地址读写寄存器堆数据，数据打入寄存器堆。

（2）板级验证：

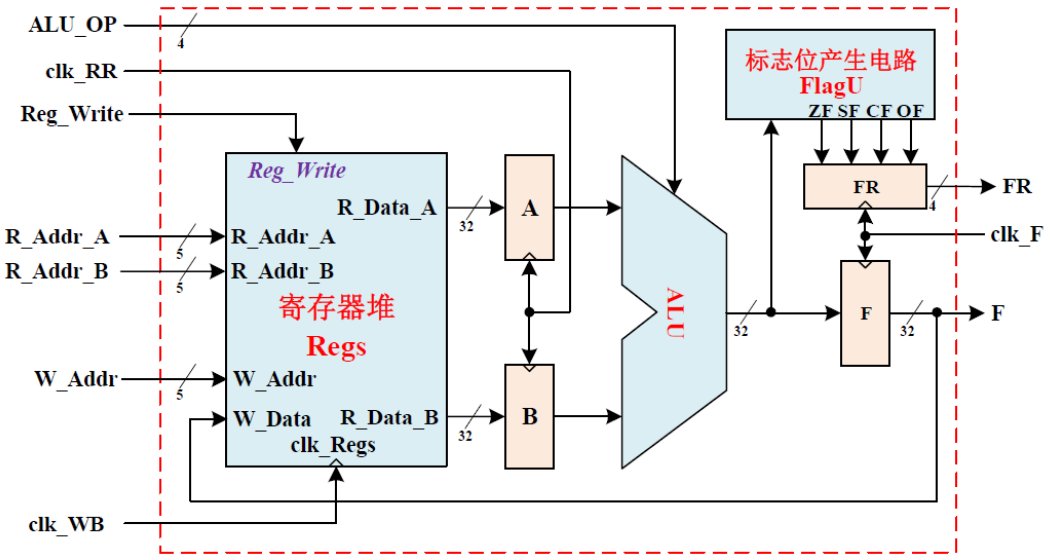


图11.17　运算器逻辑框图

①给出寄存器堆的 A 口地址 R_Addr_A=i：读出源寄存器xi内容到 A 口数据端 R_Data_A；同时给出寄存器堆的 B 口地址 R_Addr_B=j：读出源寄存器xj内容到 B 口数据端 R_Data_B；

②提供 clk_RR 时钟上跳沿：将 A 口数据 R_Data_A 打入暂存器 A，同时将 B 口数据 R_Data_A 打入暂存器 B；

③给出 ALU_OP：指定 ALU 运算功能θ；

④提供 clk_F 时钟上跳沿：将 ALU 运算结果$(xi)\theta(xj)$打入暂存器 F；

⑤给出寄存器堆的写入口地址 W_Addr=k，且置 Reg_Write=1；

⑥提供寄存器堆的写回时钟 clk_WB：将运算结果打入了寄存器xk。

2.实验程序源代码及注释等

```
module dcen(rst_n,
            R_Addr_A,R_Addr_B,W_Addr,Reg_Write,clk_WB,
            clk_1M,CLR,clk_RR,clk_F,ALU_OP,choose,FR,AN,seg);
    input rst_n;
    //寄存器堆
    input [4:0]R_Addr_A;
    input [4:0]R_Addr_B;
    input [4:0]W_Addr;
    input Reg_Write;
    input clk_WB;
```

```
    wire [31:0]W_Data;
    wire [31:0]R_Data_A;
    wire [31:0]R_Data_B;
  //ALU
  input clk_1M;
  input CLR;
  input clk_RR;
  input clk_F;
  input [3:0]ALU_OP;
  input [1:0]choose;
  output [3:0]FR;
  output [7:0]AN;
  output [7:0]seg;
Reg hh(rst_n,R_Addr_A,R_Addr_B,W_Addr,Reg_Write,
          W_Data,clk_WB,R_Data_A,R_Data_B);
dingceng kk(R_Data_A,R_Data_B,clk_1M,CLR,rst_n,
clk_RR,clk_F,ALU_OP,choose,W_Data,FR,AN,seg);
endmodule

module Reg(rst_n,R_Addr_A,R_Addr_B,W_Addr,//寄存器模块
Reg_Write,W_Data,clk_Regs,R_Data_A,R_Data_B);
    input rst_n;
    input [4:0]R_Addr_A;
    input [4:0]R_Addr_B;
    input [4:0]W_Addr;
    input Reg_Write;
    input [31:0]W_Data;
    input clk_Regs;
    output [31:0]R_Data_A;
    output [31:0]R_Data_B;
```

```
    integer i;
    wire clk;
    reg[31:0]REG_Files[0:31];//寄存器阵列
    assign R_Data_A=REG_Files[R_Addr_A];//读 A 操作
    assign R_Data_B=REG_Files[R_Addr_B];//读 B 操作
    always@(posedge clk_Regs or negedge rst_n)
    begin
        if(!rst_n)
                begin
                    REG_Files[0]<=32'h0000_0000;
                    REG_Files[1]<=32'h0000_0001;
                    REG_Files[2]<=32'h0000_0004;
                    REG_Files[3]<=32'hFFF8_0f0f;
                    REG_Files[4]<=32'hFFF8_0001;
                    REG_Files[5]<=32'hFFF8_0002;
                  for(i=6;i<31;i=i+1) REG_Files[i]<=32'h0000_0000;//初始化 32 寄存器
                end
        else
            begin
                if(Reg_Write&&W_Addr!=0)//上跳沿
                    REG_Files[W_Addr]<=W_Data;//写入寄存器
            end
    end
endmodule

module dingceng(R_Data_A,R_Data_B,clk_1M,CLR,rst_n,
clk_RR,clk_F,ALU_OP,choose,F,FR,AN,seg);//顶层模块
   input [31:0]R_Data_A;
   input [31:0]R_Data_B;
   input clk_1M;
```

```
    input CLR;
    input rst_n;
    input clk_RR;
    input clk_F;
    input [3:0]ALU_OP;
    input [1:0]choose;
    output [3:0]FR;
    output [7:0]AN;
    output [7:0]seg;
    wire CLK;
    wire [31:0]ALU_A;
    wire [31:0]ALU_B;
    wire [31:0]ALU_F;
    output [31:0]F;
    wire [31:0]Choice;
    wire ZF,SF,CF,OF;
    zcA aa(rst_n,R_Data_A,clk_RR,ALU_A);
    zcB bb(rst_n,R_Data_B,clk_RR,ALU_B);
    ALU cc(ALU_OP,ALU_A,ALU_B,ALU_F,ZF,SF,CF,OF);
    zcF dd(rst_n,ALU_F,clk_F,F);
    jcqFR ee(clk_F,rst_n,ZF,SF,CF,OF,FR);
    fpq gg(CLR,clk_1M,CLK);
    threeone kk(choose,ALU_A,ALU_B,F,Choice);
    smgsm ff(CLK,CLR,Choice,AN,seg);
endmodule

module threeone(Out_Sel,ALU_A,ALU_B,F,Choice);//三选一 A,B,F 输出到数码管显示
      input [1:0]Out_Sel;
      input [31:0]ALU_A;
      input [31:0]ALU_B;
```

```
    input [31:0]F;
    output reg [31:0]Choice;
    always@(*)
    begin
        case(Out_Sel)
                2'b00:begin Choice=F; end
                2'b01:begin Choice=ALU_A; end
                2'b10:begin Choice=ALU_B; end
                default:begin Choice=F; end
        endcase
    end
endmodule

module zcA(rst_n,Data_A,clk_A,ALU_A);//A 暂存器
    input rst_n;
    input [31:0]Data_A;
    input clk_A;
    output reg [31:0]ALU_A;
    always@(negedge rst_n or posedge clk_A)
    begin
        if(!rst_n)
            ALU_A<=32'b0;
        else
            ALU_A<=Data_A;
    end
endmodule

module zcB(rst_n,Data_B,clk_B,ALU_B);//B 暂存器
    input rst_n;
    input [31:0]Data_B;
```

```
    input clk_B;
    output reg [31:0]ALU_B;
    always@(negedge rst_n or posedge clk_B)
    begin
        if(!rst_n)
            ALU_B<=32'b0;
        else
            ALU_B<=Data_B;
    end
endmodule

module zcF(rst_n,ALU_F,clk_F,F);//F 暂存器
    input rst_n;
    input [31:0]ALU_F;
    input clk_F;
    output reg [31:0]F;
    always@(negedge rst_n or posedge clk_F)
    begin
        if(!rst_n)
            F<=32'b0;
        else
            F<=ALU_F;
    end
endmodule

module jcqFR(clk_F,rst_n,ZF,SF,CF,OF,FR);//FR 标志寄存器
    input clk_F;
    input rst_n;
    input ZF,SF,CF,OF;
    output reg [3:0]FR;
```

```
    always@(negedge rst_n or posedge clk_F)
    begin
        if(!rst_n)
            FR<=4'b0;
        else
            begin
                FR[3]=ZF;
                FR[2]=SF;
                FR[1]=CF;
                FR[0]=OF;
            end
    end
endmodule

module ALU(ALU_OP,ALU_A,ALU_B,ALU_F,ZF,SF,CF,OF); //ALU 功能模块
    input [3:0]ALU_OP;
    input [31:0]ALU_A;
    input [31:0]ALU_B;
    output [31:0]ALU_F;
    output   ZF;//是否为 0，结果为 0 时，ZF=1;否则，ZF=0
    output   SF;//符号标志位，与运算结果的最高位相同，正数时，SF=0
    output   CF;//进位/借位，最高位产生的进位 C32，加法：C32=1 时，CF=1;减法：C32=0
时，CF=1   无符号
    output   OF;//溢出标志位，有溢出，OF=1；带符号
    reg [31:0]ALU_F;
    reg C32;//最高进位
    reg ZF,SF,CF,OF;
    always@(*)
    begin
        C32=0;
```

```
        case(ALU_OP)
          4'b0000:begin {C32,ALU_F}=ALU_A+ALU_B; end                    //加法
          4'b0001:begin ALU_F=ALU_A<<ALU_B; end                         //左移
          4'b0010:begin  ALU_F=($signed(ALU_A)<$signed(ALU_B))?1:0;  end      //有符
号数比较
          4'b0011:begin ALU_F=(ALU_A<ALU_B)?1:0; end                 //无符号数比较
          4'b0100:begin ALU_F=ALU_A^ALU_B; end                          //异或
          4'b0101:begin ALU_F=ALU_A>>ALU_B; end              //逻辑右移，高位补零
          4'b0110:begin ALU_F=ALU_A|ALU_B; end                          //按位或
          4'b0111:begin ALU_F=ALU_B&ALU_A; end                          //按位与
          4'b1000:begin {C32,ALU_F}=ALU_A-ALU_B; end                    //减法
          4'b1101:begin  ALU_F=($signed(ALU_A))>>>ALU_B;  end          //算术右移，
高位补 ALU_A[31]
        endcase
          ZF = ALU_F==0;//F 全为 0，则 ZF=1
          SF = ALU_F[31];//符号标志,取 F 的最高位
          CF = C32; //进位借位标志
          OF = ALU_A[31]^ALU_B[31]^ALU_F[31]^C32;//溢出标志公式
    end
endmodule

module fpq(CLR,clk_1M,CLK);              //分频器模块
  input CLR;
  input clk_1M;
  output reg CLK;
  reg [8:0]counter;
  always@(negedge CLR or posedge clk_1M)
    begin
      if(!CLR)
        begin
```

```
            counter<=9'd0;
            CLK<=1'b0;
        end
    else if(counter==9'd2000)
            begin
                CLK<=~CLK;
                counter<=9'd0;
            end
    else
            counter<=counter+1'b1;
end
endmodule

module smgsm(CLK,CLR,Choice,AN,seg);//数码管扫描显示模块
    input CLK; //扫描的间隔始终，控制数码管轮流点亮的频率
    input CLR; //复位信号，用于初始化状态
    input[31:0]Choice;
    output reg [7:0]AN;//位选
    output reg [7:0]seg; //段选
    reg [3:0] data;
    reg [2:0] bit_sel;   //数码管计数器指示，000~111  最左到最右
    always@(negedge CLR or posedge CLK)
    begin
        if(!CLR)
                bit_sel<=3'b000;
        else
            bit_sel<=bit_sel+1'b1;
    end
    always@(*)
        begin
```

```
        case(bit_sel)
            3'b000:data<=Choice[3:0];
            3'b001:data<=Choice[7:4];
            3'b010:data<=Choice[11:8];
            3'b011:data<=Choice[15:12];
            3'b100:data<=Choice[19:16];
            3'b101:data<=Choice[23:20];
            3'b110:data<=Choice[27:24];
            3'b111:data<=Choice[31:28];
            default:data<=data;
        endcase
    end
always@(*)
    begin
        case(bit_sel)
            3'b000:AN=8'b11111110;
            3'b001:AN=8'b11111101;
            3'b010:AN=8'b11111011;
            3'b011:AN=8'b11110111;
            3'b100:AN=8'b11101111;
            3'b101:AN=8'b11011111;
            3'b110:AN=8'b10111111;
            3'b111:AN=8'b01111111;
            default:AN=8'b11111111;
        endcase
    end
always@(*)
    begin
        case(data)
                0:seg<=8'b00000011;
```

```
                1:seg<=8'b10011111;
                2:seg<=8'b00100101;
                3:seg<=8'b00001101;
                4:seg<=8'b10011001;
                5:seg<=8'b01001001;
                6:seg<=8'b01000001;
                7:seg<=8'b00011111;
                8:seg<=8'b00000001;
                9:seg<=8'b00001001;
                10:seg<=8'b00010001;
                11:seg<=8'b11000001;
                12:seg<=8'b01100011;
                13:seg<=8'b10000101;
                14:seg<=8'b01100001;
                15:seg<=8'b01110001;
            default:seg<=8'b11111111;
        endcase
    end
endmodule
```

三、实验仿真

1、 仿真代码

这里只是寄存器堆模块的仿真激励代码，验证寄存器地址读写功能

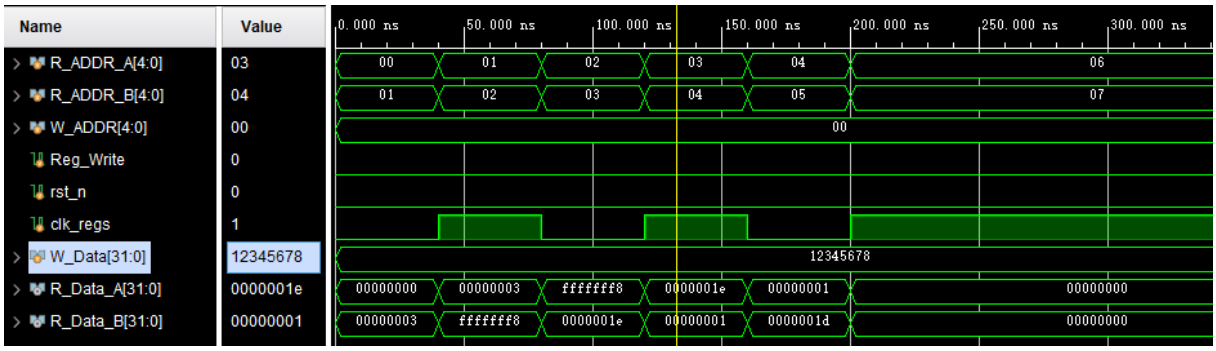
```
module jcqdfz();
//INPUT
reg rst_n;
reg [4:0]R_Addr_A;
reg [4:0]R_Addr_B;
reg [4:0]W_Addr;
reg Reg_Write;
```

```
reg [31:0]W_Data;
reg clk_Regs;

//OUTPUT
wire [31:0]R_Data_A;
wire [31:0]R_Data_B;
initial
      begin
          rst_n=0;clk_regs<=0;rst_n<=1;
      end
initial
   begin
      R_Addr_A<=0;R_Addr_B<=1;W_Addr<=0;
      Reg_Write<=0;rst_n<=0;W_Data<=32'h12345678;
          #40 clk_Regs<=~clk_Regs;R_Addr_A<=1;R_Addr_B<=2;
          #40 clk_Regs<=~clk_Regs;R_Addr_A<=2;R_Addr_B<=3;
          #40 clk_Regs<=~clk_Regs;R_Addr_A<=3;R_Addr_B<=4;
          #40 clk_Regs<=~clk_Regs;R_Addr_A<=4;R_Addr_B<=5;
          #40 clk_Regs<=~clk_Regs;R_Addr_A<=6;R_Addr_B<=7;
      End
Reg hh(rst_n,R_Addr_A,R_Addr_B,W_Addr,
Reg_Write,W_Data,clk_Regs,R_Data_A,R_Data_B);
endmodule
```

2、 仿真波形

（运行仿真时，波形截图）



3、 仿真结果分析

寄存器堆模块的读写功能都符合预期的效果。逻辑成功。

四、电路图

五、引脚配置

（引脚约束文件的内容，描述主要配置情况）

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 {AN[7]}]

set_property IOSTANDARD LVCMOS18 [get_ports {AN[6]}]

set_property IOSTANDARD LVCMOS18 [get_ports {AN[5]}]

set_property IOSTANDARD LVCMOS18 [get_ports {AN[4]}]

set_property IOSTANDARD LVCMOS18 [get_ports {AN[3]}]

set_property IOSTANDARD LVCMOS18 [get_ports {AN[2]}]

set_property IOSTANDARD LVCMOS18 [get_ports {AN[1]}]

set_property IOSTANDARD LVCMOS18 [get_ports {AN[0]}]

set_property IOSTANDARD LVCMOS18 [get_ports {choose[1]}]

set_property IOSTANDARD LVCMOS18 [get_ports {choose[0]}]

set_property IOSTANDARD LVCMOS18 [get_ports {FR[3]}]

set_property IOSTANDARD LVCMOS18 [get_ports {FR[2]}]

set_property IOSTANDARD LVCMOS18 [get_ports {FR[1]}]

set_property IOSTANDARD LVCMOS18 [get_ports {FR[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 {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 {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 IOSTANDARD LVCMOS18 [get_ports clk_1M]
set_property IOSTANDARD LVCMOS18 [get_ports clk_F]
set_property IOSTANDARD LVCMOS18 [get_ports clk_RR]
set_property IOSTANDARD LVCMOS18 [get_ports clk_WB]
set_property IOSTANDARD LVCMOS18 [get_ports CLR]
set_property IOSTANDARD LVCMOS18 [get_ports Reg_Write]
set_property IOSTANDARD LVCMOS18 [get_ports rst_n]
```

```
set_property PULLDOWN true [get_ports clk_1M]
set_property PULLDOWN true [get_ports clk_F]
set_property PULLDOWN true [get_ports clk_RR]
set_property PULLDOWN true [get_ports clk_WB]
set_property PULLDOWN true [get_ports CLR]
set_property PULLDOWN true [get_ports Reg_Write]
set_property PULLDOWN true [get_ports rst_n]
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 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 PULLDOWN true [get_ports {R_Addr_A[4]}]
set_property PULLDOWN true [get_ports {R_Addr_A[3]}]
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 {choose[1]}]
set_property PULLDOWN true [get_ports {choose[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 PACKAGE_PIN T14 [get_ports {ALU_OP[3]}]
```

```
set_property PACKAGE_PIN V15 [get_ports {ALU_OP[2]}]
set_property PACKAGE_PIN R15 [get_ports {ALU_OP[1]}]
set_property PACKAGE_PIN U16 [get_ports {ALU_OP[0]}]

set_property PACKAGE_PIN C9 [get_ports {AN[7]}]
set_property PACKAGE_PIN C10 [get_ports {AN[6]}]
set_property PACKAGE_PIN D10 [get_ports {AN[5]}]
set_property PACKAGE_PIN C11 [get_ports {AN[4]}]
set_property PACKAGE_PIN M17 [get_ports {AN[3]}]
set_property PACKAGE_PIN J14 [get_ports {AN[2]}]
set_property PACKAGE_PIN K13 [get_ports {AN[1]}]
set_property PACKAGE_PIN P14 [get_ports {AN[0]}]

set_property PACKAGE_PIN R17 [get_ports {choose[1]}]
set_property PACKAGE_PIN T18 [get_ports {choose[0]}]

set_property PACKAGE_PIN V5 [get_ports {R_Addr_A[4]}]
set_property PACKAGE_PIN T4 [get_ports {R_Addr_A[3]}]
set_property PACKAGE_PIN V6 [get_ports {R_Addr_A[2]}]
set_property PACKAGE_PIN T5 [get_ports {R_Addr_A[1]}]
set_property PACKAGE_PIN T6 [get_ports {R_Addr_A[0]}]

set_property PACKAGE_PIN U6 [get_ports {FR[3]}]
set_property PACKAGE_PIN R5 [get_ports {FR[2]}]
set_property PACKAGE_PIN U7 [get_ports {FR[1]}]
set_property PACKAGE_PIN R6 [get_ports {FR[0]}]

set_property PACKAGE_PIN V7 [get_ports {R_Addr_B[4]}]
set_property PACKAGE_PIN R8 [get_ports {R_Addr_B[3]}]
set_property PACKAGE_PIN U9 [get_ports {R_Addr_B[2]}]
```

```
set_property PACKAGE_PIN T9 [get_ports {R_Addr_B[1]}]
set_property PACKAGE_PIN V10 [get_ports {R_Addr_B[0]}]

set_property PACKAGE_PIN F14 [get_ports {seg[7]}]
set_property PACKAGE_PIN N14 [get_ports {seg[6]}]
set_property PACKAGE_PIN J13 [get_ports {seg[5]}]
set_property PACKAGE_PIN G13 [get_ports {seg[4]}]
set_property PACKAGE_PIN F13 [get_ports {seg[3]}]
set_property PACKAGE_PIN G14 [get_ports {seg[2]}]
set_property PACKAGE_PIN M13 [get_ports {seg[1]}]
set_property PACKAGE_PIN H14 [get_ports {seg[0]}]

set_property PACKAGE_PIN R10 [get_ports {W_Addr[4]}]
set_property PACKAGE_PIN U11 [get_ports {W_Addr[3]}]
set_property PACKAGE_PIN R11 [get_ports {W_Addr[2]}]
set_property PACKAGE_PIN U12 [get_ports {W_Addr[1]}]
set_property PACKAGE_PIN T13 [get_ports {W_Addr[0]}]

set_property PACKAGE_PIN E3 [get_ports clk_1M]
set_property PACKAGE_PIN P18 [get_ports clk_F]
set_property PACKAGE_PIN N17 [get_ports clk_RR]
set_property PACKAGE_PIN P17 [get_ports clk_WB]
set_property PACKAGE_PIN U17 [get_ports CLR]
set_property PACKAGE_PIN V14 [get_ports Reg_Write]
set_property PACKAGE_PIN U18 [get_ports rst_n]

set_property CLOCK_DEDICATED_ROUTE FALSE [get_nets clk_1M]
set_property CLOCK_DEDICATED_ROUTE FALSE [get_nets clk_F]
set_property CLOCK_DEDICATED_ROUTE FALSE [get_nets clk_RR]
set_property CLOCK_DEDICATED_ROUTE FALSE [get_nets clk_WB]
```

set_property CLOCK_DEDICATED_ROUTE FALSE [get_nets CLR]

set_property CLOCK_DEDICATED_ROUTE FALSE [get_nets Reg_Write]

set_property CLOCK_DEDICATED_ROUTE FALSE [get_nets rst_n]

六、思考与探索

1、 实验结果记录：

REG_Files[0]<=32'h0000_0000;

REG_Files[1]<=32'h0000_0001;

REG_Files[2]<=32'h0000_0004;

REG_Files[3]<=32'hFFF8_0f0f;

REG_Files[4]<=32'hFFF8_0001;

REG_Files[5]<=32'hFFF8_0002;

for(i=6;i<31;i=i+1) REG_Files[i]<=32'h0000_0000;//初始化 32 寄存器

（实验操作的过程及结果记录）

A[4:0]	B[4:0]	ALU_OP	功能	F[31:0]	FR(ZF SF CF OF)	W_Addr[4:0]
00000	00001	0000	加法	0000_0001	0000	00000
00000	00001	0000	加法	0000_0001	0000	01001
01001	00001	0000	加法	1FFF_FFFF	0000	01001
01001	00001	0000	加法	2000_0000	0000	01010
00011	00010	0101	逻辑右移	0FFF_80F0	0001	01100
00011	00010	1101	算数右移	FFFF_80F0	0100	01110

2、 实验结论：

实验结果符合预期结果，逻辑正确，实验成功。

3、 问题与解决方案：

仿真总是直接仿真整个顶层模块，无法只仿真寄存器模块；

在反复试了几次后，把除了仿真要的模块外的内容都注释掉，设为顶层文件。

4、 思考题：

1）. 在仿真测试和板级测试中，你是如何确认寄存器堆被写入成功的请具体说明。

答：反复做加法。例如选择 1 号寄存器，原本里面的数据是 FFFF_0000H，再和

2 号寄存器中的数据 0000_0001H 做加法，然后写入 1 号寄存器中，若 1 号寄存器读出的数据依次为 FFFF_0001H,FFFF_0002H,则寄存器堆被写入成功。

2）. 按照 表 11.7 输入信号需要的开关总计 20 个，请分析板级实验的过程，说明哪些开关可以分时复用？为什么？按照这样的思路，请计算最少需要多少个开关？

答：R_Addr_A[4:0]和 R_Addr_B[4:0]和 W_Addr[4:0]都可以分时复用；最少需要 10 个开关。

3）. 图 11.17 中，暂存器 A 和 B 的打入脉冲是 clk_RR，暂存器 F 和标志寄存器 FR 的打入脉冲是 clk_F，目的寄存器的打入脉冲是 clk_WB，请问它们能合并为一个吗？如果能，请给出计算的过程；如果不能，请说明理由。

答：不能，因为会有延迟，合并成一个会导致输出错误结果。

4）. 图 11.17 所示的运算器 只能实现寄存器间的运算 $(xi)\theta(xj)\rightarrow xk$，如果要想同时完成寄存器$xi$内容与一个立即数$m$进行运算，结果送目的寄存器$xk$，即 ：$(xi)\ \theta\ m\ \rightarrow xk$，你觉得应该如何改造图 11.17？说明你的设计方案，以及运算器工作的过程。

答：从外界输入一个立即数，并从寄存器xi读出内容，输入到在一个运算模块中进行运算，然后把结果送到目的寄存器xk。

5）. 谈谈你在实验中碰到了哪些问题？又是如何解决的？

答：仿真总是直接仿真整个顶层模块，无法只仿真寄存器模块；在反复试了几次后，把除了仿真要的模块外的内容都注释掉，设为顶层文件。