通用寄存器堆实验报告

2021 年 4 月 26 日　　成绩：_____

姓名	周俊佐	学号	19071337	班级	19185312
姓名	颜伟鹏	学号	19071334	班级	19185312
姓名	高炜哲	学号	18081811	班级	18181411
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	章复嘉	指导老师	章复嘉	机位号	14
实验序号	3	实验名称	通用寄存器堆实验		

一、实验目的和实验要求

1:掌握寄存器堆的数据传送与读写工作原理;

2:掌握 ARM v7 CPU 多工作模式·多端口寄存器堆的设计方法;

3:编程实现 16×32 位的 User/Sys 模式下通用寄存器堆模块，并通过仿真验证;

4:在 16×32 位寄存器堆基础上扩展，编程实现 32×32 位的 9 种工作模式下通用寄存器堆模块，通过仿真验证和板级调试;

5:撰写实验报告·(备注：本组将测试代码分成两组，一组正常测试，一组测试所有异常情况，如读写 hyp 模式下的 r14 寄存器)

二、实验程序源代码

Board.v

```
`timescale 1ns / 1ps

module Board(sw, swb, led, clk, which, seg, enable);

    input [1:32] sw;
    input [1:6] swb;
    reg [4:1]  R_Addr_A, R_Addr_B, R_Addr_C;
    reg [4:1] W_Addr;
    reg [5:1] M;

    reg Write_Reg,Write_PC;
    reg [32:1] W_Data,PC_New;
    reg [8:1] Shift_Num;
    reg [32:0] data;

    reg [2:1]cnt = 0;
    reg [3:1]cnt2 = 0;

    wire [32:1] R_Data_A, R_Data_B, R_Data_C;
    wire [32:1] R_Data_PC;

    output [1:32] led;
    input clk;
    output [2:0] which;
    output [7:0] seg;

    output reg enable = 1;

    always @(posedge swb[1])
    begin
        if (swb[2])
        begin
            case(cnt)
            0:begin {R_Addr_A, R_Addr_B, R_Addr_C} = sw[1:12]; M=sw[17:21]; W_Addr=sw[25:28]; Wri
te_Reg=sw[31];Write_PC=sw[32]; end
            1:begin W_Data=sw; end
            2:begin PC_New=sw; end
            endcase
            cnt <= (cnt+1)%3;
        end
    end

    always @(posedge swb[6])
```

```
    begin
        case(cnt2)
        0:begin data <= {R_Data_A,1'b1}; end
        1:begin data <= {R_Data_B,1'b1}; end
        2:begin data <= {R_Data_C,1'b1}; end
        3:begin data <= {R_Data_PC,1'b1}; end
        default:begin data <= {32'h88888888,1'b0}; end
        endcase
        cnt2 <= (cnt2+1)%5;
    end
    assign led[31:32] = cnt[2:1];
    assign led[17]=Write_Reg;
    assign led[18]=Write_PC;
    assign led[12:16]=M;
    assign led[1:3] = cnt2[3:1];
```

```
    regFile regFile_Instance(
        .clk(swb[3]),
        .Rst(swb[4]),
        .M(M),
        .PC_New(PC_New),
        .Write_PC(Write_PC),
        .Write_Reg(Write_Reg),
        .R_Addr_A(R_Addr_A),
        .R_Addr_B(R_Addr_B),
        .R_Addr_C(R_Addr_C),
        .W_Addr(W_Addr),
        .W_Data(W_Data),
        .R_Data_A(R_Data_A),
        .R_Data_B(R_Data_B),
        .R_Data_C(R_Data_C),
        .R_Data_PC(R_Data_PC)
    );
```

```
    Display Display_Instance(
        .clk(clk),
        .data(data),
        .which(which),
        .seg(seg)
    );

endmodule
```

Display.v

```
`timescale 1ns / 1ps
```

```
module Display(clk, data, which, seg, count, digit);
    input clk;
    input [32:1] data;
    output reg [2:0] which = 0;
    output reg [7:0] seg;

    output reg [10:0] count = 0;
    always @(posedge clk) count <= count + 1'b1;
    always @(negedge clk) if (&count) which <= which + 1'b1;

    output reg [3:0] digit;
    always @* case (which)
        0: digit <= data[32:29];
        1: digit <= data[28:25];
        2: digit <= data[24:21];
        3: digit <= data[20:17];
        4: digit <= data[16:13];
        5: digit <= data[12:09];
        6: digit <= data[08:05];
        7: digit <= data[04:01];
    endcase

    always @* case (digit)
        4'h0: seg <= 8'b0000_0011;
        4'h1: seg <= 8'b1001_1111;
        4'h2: seg <= 8'b0010_0101;
        4'h3: seg <= 8'b0000_1101;
        4'h4: seg <= 8'b1001_1001;
        4'h5: seg <= 8'b0100_1001;
        4'h6: seg <= 8'b0100_0001;
        4'h7: seg <= 8'b0001_1111;
        4'h8: seg <= 8'b0000_0001;
        4'h9: seg <= 8'b0000_1001;
        4'hA: seg <= 8'b0001_0001;
        4'hB: seg <= 8'b1100_0001;
        4'hC: seg <= 8'b0110_0011;
        4'hD: seg <= 8'b1000_0101;
        4'hE: seg <= 8'b0110_0001;
        4'hF: seg <= 8'b0111_0001;
    endcase

endmodule // Display
```

regfile_Usr_Sys_State.v

```
`timescale 1ns / 1ps

module regfile_User_Sys_State( // User/Sys Mode 16*32 regfile
//  INPUT
    input clk,
    input Rst,
//  input [4:0] M, // from CPSR
//  Read Addr
    input [3:0] R_Addr_A,
    input [3:0] R_Addr_B,
    input [3:0] R_Addr_C,
//  Write Addr
    input [3:0] W_Addr,
//  Write Data
    input [31:0] W_Data,
//    input [31:0] PC_New,
//    input Write_PC,
    input Write_Reg,

//  OUTPUT
    output [31:0] R_Data_A,
    output [31:0] R_Data_B,
    output [31:0] R_Data_C
    );

    reg [31:0] regfile [0:15]; // regfile
    integer i; // traverse all regs

    // initialize
    initial begin
        for(i=0; i<16; i=i+1) regfile[i] <= 0;
    end
```

```
    // Write or Clear Operation
    always@(posedge ~clk or posedge Rst) begin
        // Write
        if(Write_Reg) regfile[W_Addr] <= W_Data;
        // Clear
        if (Rst) for(i=0; i<16; i=i+1) regfile[i] <= 0;
    end
```

```
    // Read Operation
    assign R_Data_A = regfile[R_Addr_A];
    assign R_Data_B = regfile[R_Addr_B];
    assign R_Data_C = regfile[R_Addr_C];

endmodule
```

测试数据：

tb_regFile.v

```
`timescale 1ns / 1ps
////////////////////////////////////////////////////////////////////////////////

module tb_regFile();
//  INPUT
    reg clk = 0;
    reg Rst;
    reg [4:0] M; // from CPSR
//  Read Addr
    reg [3:0] R_Addr_A, R_Addr_B, R_Addr_C;
//  Write Addr
    reg [3:0] W_Addr;
//  Write Data
    reg [31:0] W_Data, PC_New;
//  Enable
    reg Write_Reg, Write_PC;
//  OUTPUT
    wire [31:0] R_Data_A, R_Data_B, R_Data_C, R_Data_PC;
    wire err1, err2;

    always #1 clk = ~clk;
    initial begin
```

```
// ############################
    // WRITE
    M = 5'b10000;   //usr
    Write_Reg = 1;
    W_Addr = 0;
    W_Data = 32'h11110000;
    #25
    W_Addr = 13;
```

```
W_Data = 32'hffff0000;
#25
```

```
M = 5'b10001;   //fiq
Write_Reg = 1;
W_Addr = 1;
W_Data = 32'h11110001;
#25
W_Addr = 14;
W_Data = 32'h99999999;
```

```
M = 5'b10010;   //irq
Write_Reg = 1;
W_Addr = 2;
W_Data = 32'h11110010;
#25
W_Addr = 13;
W_Data = 32'haaaaaaaa;
```

```
M = 5'b10011;   //svc
Write_Reg = 1;
W_Addr = 3;
W_Data = 32'h11110011;
#25
W_Addr = 14;
W_Data = 32'hbbbbbbbb;
```

```
M = 5'b10110;   //mon
Write_Reg = 1;
W_Addr = 4;
W_Data = 32'h11110100;
#25
W_Addr = 13;
W_Data = 32'hcccccccc;
```

```
M = 5'b10111;   //abt
Write_Reg = 1;
W_Addr = 5;
W_Data = 32'h11110101;
#25
W_Addr = 14;
W_Data = 32'hdddddddd;
```

```
M = 5'b11010;   //hyp
Write_Reg = 1;
W_Addr = 6;
W_Data = 32'h11110110;
#25
W_Addr = 13;
W_Data = 32'heeeeeeee;
```

```
M = 5'b11011;   //und
Write_Reg = 1;
W_Addr = 7;
W_Data = 32'h11110111;
#25
W_Addr = 14;
W_Data = 32'hffffffff;
```

```
M = 5'b11111;   //sys
Write_Reg = 1;
W_Addr = 8;
W_Data = 32'h11111000;
#25
W_Addr = 14;
W_Data = 32'hffff0000;
#25
```

```
// unenabled
M = 5'b11111;
Write_Reg = 0;
W_Addr = 13;
W_Data = 32'h00000000;
#25

// READ
M = 5'b10000;   //usr
R_Addr_A = 13;
#25
M = 5'b10001;   //fiq
R_Addr_B = 0;
#25
M = 5'b10010;   //irq
R_Addr_C = 1;
#25
```

```
M = 5'b10011;   //svc
R_Addr_A = 2;
#25
M = 5'b10110;   //mon
R_Addr_B = 3;
#25
M = 5'b10111;   //abt
R_Addr_C = 4;
#25
M = 5'b11010;   //hyp
R_Addr_A = 5;
#25
M = 5'b11011;   //und
R_Addr_B = 6;
#25
M = 5'b11111;   //sys
R_Addr_C = 7;
#25
```

```
// Test PC
M = 5'b10111;   //abt
Write_PC = 1;
PC_New = 32'hffffffff;
#25
```

```
M = 5'b10001;   //fiq
R_Addr_A = 15;
#25
```

```
M = 5'b10000;   //usr
Write_PC = 0;
PC_New = 32'h00000000;
#25
```

```
M = 5'b10111;   //sys
R_Addr_A = 15;
```

```
M = 5'b10001;   //usr
Write_PC = 1;
PC_New = 32'h11111111;
#25
```

```
M = 5'b10001;   //sys
R_Addr_A = 15;
```

```
    end

    regFile regFile_Instance(
    //  INPUT
        .clk(clk),
        .Rst(Rst),
        .M(M), // from CPSR
    //  Read Addr
        .R_Addr_A(R_Addr_A),
        .R_Addr_B(R_Addr_B),
        .R_Addr_C(R_Addr_C),
    //  Write Addr
        .W_Addr(W_Addr),
    //  Write Data
        .W_Data(W_Data),
        .PC_New(PC_New),
        .Write_PC(Write_PC),
        .Write_Reg(Write_Reg),
    //  OUTPUT
        .R_Data_A(R_Data_A),
        .R_Data_B(R_Data_B),
        .R_Data_C(R_Data_C),
        .R_Data_PC(R_Data_PC),
        .err1(err1),
        .err2(err2)
    );
endmodule
```

tb_regFile_implementation.v

```
`timescale 1ns / 1ps

module tb_regFile_implementation();
//  INPUT
    reg clk = 0;
    reg Rst;
    reg [4:0] M; // from CPSR
//  Read Addr
    reg [3:0] R_Addr_A, R_Addr_B, R_Addr_C;
//  Write Addr
    reg [3:0] W_Addr;
//  Write Data
    reg [31:0] W_Data, PC_New;
```

```
//  Enable
    reg Write_Reg, Write_PC;
//  OUTPUT
    wire [31:0] R_Data_A, R_Data_B, R_Data_C, R_Data_PC;
    wire err1, err2;

    always #1 clk = ~clk;
    initial begin
```

```
    M = 5'b10000; // right
    Write_Reg = 1;
    W_Addr = 14;
    W_Data = 32'h11110000;
    #25
```

```
    M = 5'b11010;   //hyp write -> R14
    Write_Reg = 1;
    W_Addr = 14;
    W_Data = 32'h11110110;
    #25
```

```
    M = 5'b11010;   // right
    Write_Reg = 1;
    W_Addr = 13;
    W_Data = 32'h11110110;
    #25
```

```
    M = 5'b11010;   //hyp read -> R14
    R_Addr_A = 14;
    #25
```

```
    M = 5'b11010;   //right
    R_Addr_A = 13;
    #25
```

```
    M = 5'b01010; // wrong state write -> R14
    Write_Reg = 1;
    W_Addr = 14;
    W_Data = 32'h11110000;
    #25
```

```
    M = 5'b10000; // right
    Write_Reg = 1;
    W_Addr = 14;
```

```
    W_Data = 32'h11110000;
    #25

    M = 5'b01010; // wrong state write -> PC
    Write_PC = 1;
    W_Data = 32'h11110000;
    #25

    M = 5'b10000; // right
    Write_Reg = 1;
    W_Addr = 14;
    W_Data = 32'h11110000;
    #25

    M = 5'b10000; // usr write -> 15 without Write_PC
    Write_PC = 0;
    W_Addr = 15;
    W_Data = 32'h11110000;

    end

    regFile regFile_Instance(
    //  INPUT
        .clk(clk),
        .Rst(Rst),
        .M(M), // from CPSR
    //  Read Addr
        .R_Addr_A(R_Addr_A),
        .R_Addr_B(R_Addr_B),
        .R_Addr_C(R_Addr_C),
    //  Write Addr
        .W_Addr(W_Addr),
    //  Write Data
        .W_Data(W_Data),
        .PC_New(PC_New),
        .Write_PC(Write_PC),
        .Write_Reg(Write_Reg),
    //  OUTPUT
        .R_Data_A(R_Data_A),
        .R_Data_B(R_Data_B),
        .R_Data_C(R_Data_C),
        .R_Data_PC(R_Data_PC),
        .err1(err1),
        .err2(err2)
```

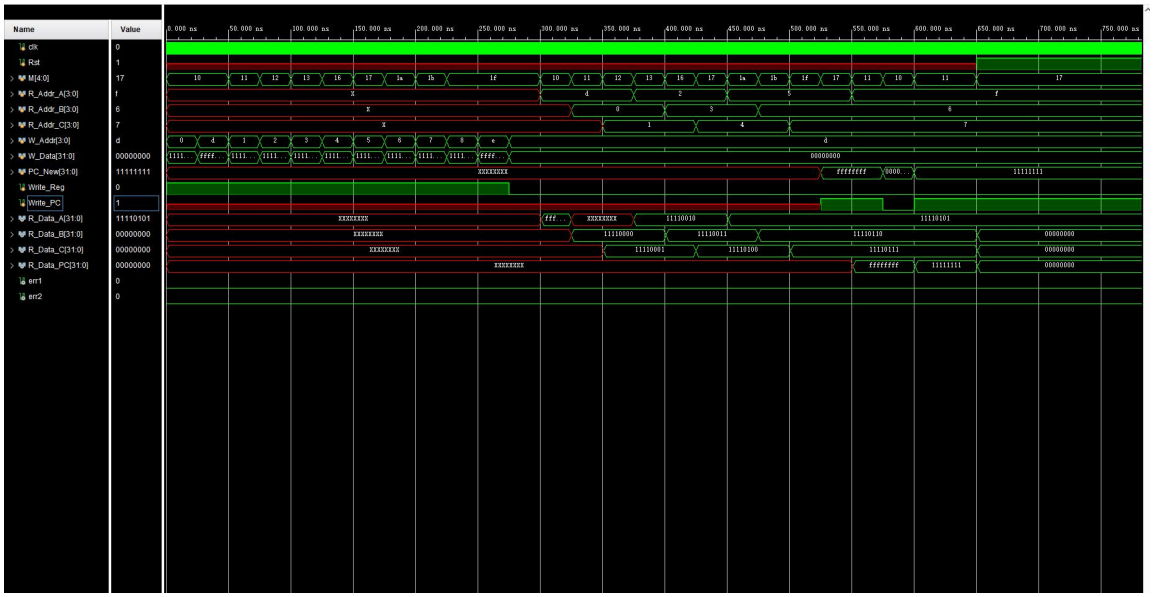
```
    );
endmodule
```

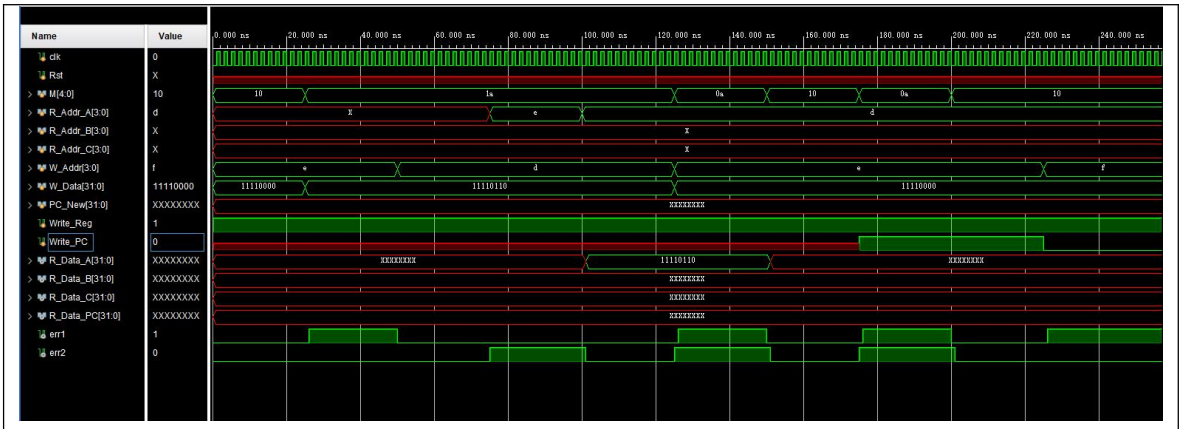
tb_regfile_User_Sys_State.v

```
`timescale 1ns / 1ps

module tb_regfile_User_Sys_State();
    reg clk = 0;
    reg Rst;
    reg [3:0] R_Addr_A, R_Addr_B, R_Addr_C;
    reg [3:0] W_Addr;
    reg [31:0] W_Data;
    reg Write_Reg;
    wire [31:0] R_Data_A, R_Data_B, R_Data_C;

    always #10 clk = ~clk;
    initial begin
    // write
    Write_Reg = 1;
    W_Addr = 1;
    W_Data = 32'hac963a55;
    #100

    Write_Reg = 1;
    W_Addr = 2;
    W_Data = 32'h11111111;
    #100

    Write_Reg = 1;
    W_Addr = 3;
    W_Data = 32'hffffffff;
    #100

    // read
    R_Addr_A = 1;
    R_Addr_B = 2;
    R_Addr_C = 3;
    #100

    // claer
    Rst = 1;
    R_Addr_C = 3;
```

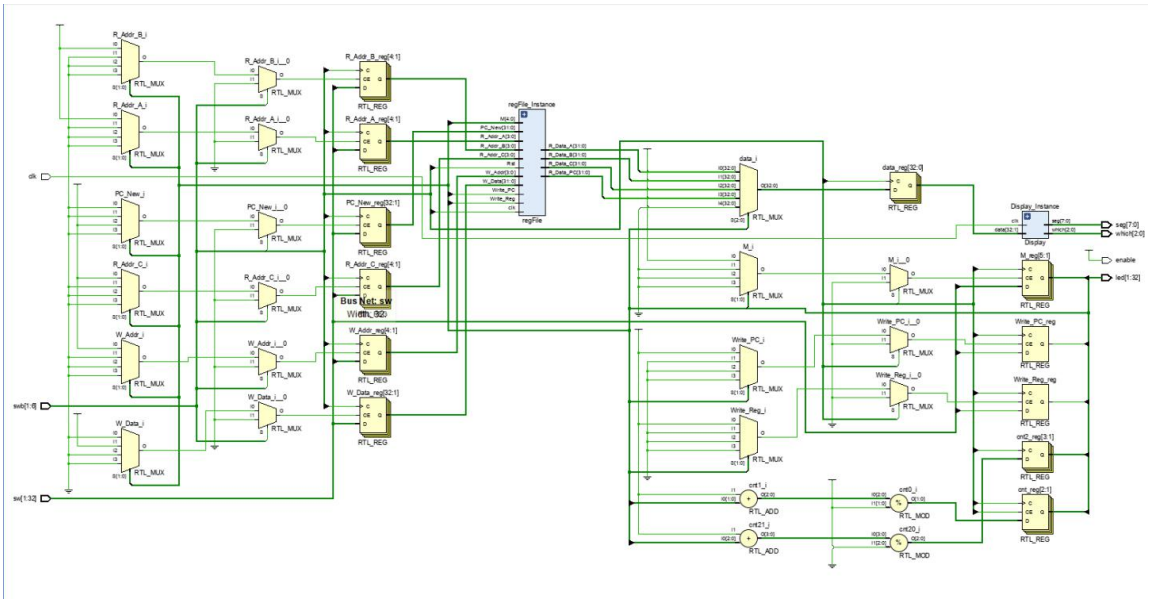
```
end


regfile_User_Sys_State regfile(
    clk, Rst,
    R_Addr_A, R_Addr_B, R_Addr_C,
    W_Addr, W_Data, Write_Reg,
    R_Data_A, R_Data_B, R_Data_C
);
endmodule
```

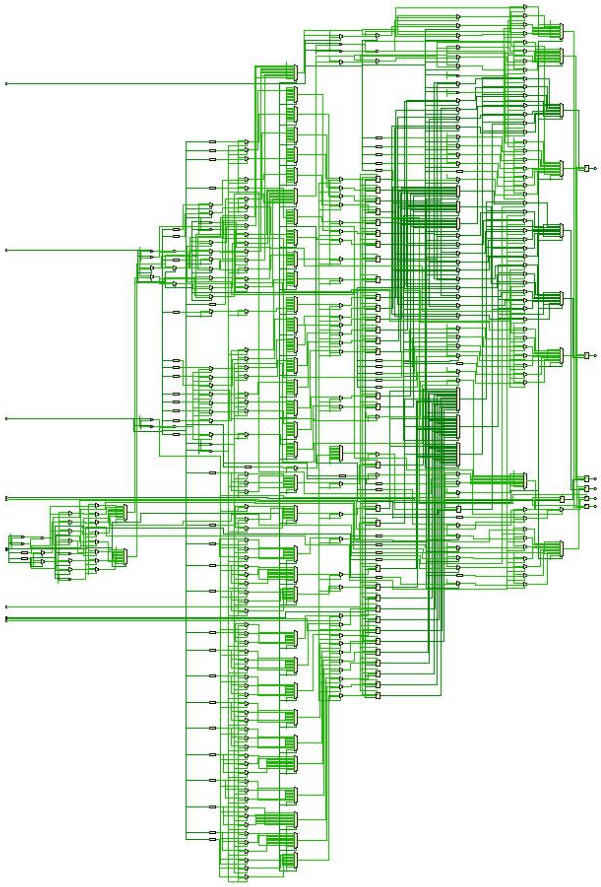
三、仿真文件和波形图（有就填，没有就不填）





四、电路图（有就填，没有就不填）





五、引脚配置（约束文件，有就填，没有就不填））

```
# .bit
set_property BITSTREAM.GENERAL.COMPRESS TRUE [current_design]

# Switch
set_property PULLDOWN true [get_ports sw]
set_property IOSTANDARD LVCMOS18 [get_ports sw]
set_property PACKAGE_PIN T3  [get_ports {sw[1]}]
set_property PACKAGE_PIN U3  [get_ports {sw[2]}]
set_property PACKAGE_PIN T4  [get_ports {sw[3]}]
set_property PACKAGE_PIN V3  [get_ports {sw[4]}]
set_property PACKAGE_PIN V4  [get_ports {sw[5]}]
set_property PACKAGE_PIN W4  [get_ports {sw[6]}]
set_property PACKAGE_PIN Y4  [get_ports {sw[7]}]
set_property PACKAGE_PIN Y6  [get_ports {sw[8]}]
set_property PACKAGE_PIN W7  [get_ports {sw[9]}]
set_property PACKAGE_PIN Y8  [get_ports {sw[10]}]
set_property PACKAGE_PIN Y7  [get_ports {sw[11]}]
set_property PACKAGE_PIN T1  [get_ports {sw[12]}]
```

```
set_property PACKAGE_PIN U1  [get_ports {sw[13]}]
set_property PACKAGE_PIN U2  [get_ports {sw[14]}]
set_property PACKAGE_PIN W1  [get_ports {sw[15]}]
set_property PACKAGE_PIN W2  [get_ports {sw[16]}]
set_property PACKAGE_PIN Y1  [get_ports {sw[17]}]
set_property PACKAGE_PIN AA1 [get_ports {sw[18]}]
set_property PACKAGE_PIN V2  [get_ports {sw[19]}]
set_property PACKAGE_PIN Y2  [get_ports {sw[20]}]
set_property PACKAGE_PIN AB1 [get_ports {sw[21]}]
set_property PACKAGE_PIN AB2 [get_ports {sw[22]}]
set_property PACKAGE_PIN AB3 [get_ports {sw[23]}]
set_property PACKAGE_PIN AB5 [get_ports {sw[24]}]
set_property PACKAGE_PIN AA6 [get_ports {sw[25]}]
set_property PACKAGE_PIN R2  [get_ports {sw[26]}]
set_property PACKAGE_PIN R3  [get_ports {sw[27]}]
set_property PACKAGE_PIN T6  [get_ports {sw[28]}]
set_property PACKAGE_PIN R6  [get_ports {sw[29]}]
set_property PACKAGE_PIN U7  [get_ports {sw[30]}]
set_property PACKAGE_PIN AB7 [get_ports {sw[31]}]
set_property PACKAGE_PIN AB8 [get_ports {sw[32]}]
```

```
# Switch Buttonť
set_property IOSTANDARD LVCMOS18 [get_ports swb]
set_property PACKAGE_PIN R4  [get_ports {swb[1]}]
set_property PACKAGE_PIN AA4 [get_ports {swb[2]}]
set_property PACKAGE_PIN AB6 [get_ports {swb[3]}]
set_property PACKAGE_PIN T5  [get_ports {swb[4]}]
set_property PACKAGE_PIN V8  [get_ports {swb[5]}]
set_property PACKAGE_PIN AA8 [get_ports {swb[6]}]
```

```
# LED
set_property PULLDOWN true [get_ports led]
set_property IOSTANDARD LVCMOS18 [get_ports led]
set_property PACKAGE_PIN R1 [get_ports {led[1]}]
set_property PACKAGE_PIN P2 [get_ports {led[2]}]
set_property PACKAGE_PIN P1 [get_ports {led[3]}]
set_property PACKAGE_PIN N2 [get_ports {led[4]}]
set_property PACKAGE_PIN M1 [get_ports {led[5]}]
set_property PACKAGE_PIN M2 [get_ports {led[6]}]
set_property PACKAGE_PIN L1 [get_ports {led[7]}]
set_property PACKAGE_PIN J2 [get_ports {led[8]}]
set_property PACKAGE_PIN G1 [get_ports {led[9]}]
set_property PACKAGE_PIN E1 [get_ports {led[10]}]
set_property PACKAGE_PIN D2 [get_ports {led[11]}]
```

```
set_property PACKAGE_PIN A1 [get_ports {led[12]}]
set_property PACKAGE_PIN L3 [get_ports {led[13]}]
set_property PACKAGE_PIN G3 [get_ports {led[14]}]
set_property PACKAGE_PIN K4 [get_ports {led[15]}]
set_property PACKAGE_PIN G4 [get_ports {led[16]}]
set_property PACKAGE_PIN K1 [get_ports {led[17]}]
set_property PACKAGE_PIN J1 [get_ports {led[18]}]
set_property PACKAGE_PIN H2 [get_ports {led[19]}]
set_property PACKAGE_PIN G2 [get_ports {led[20]}]
set_property PACKAGE_PIN F1 [get_ports {led[21]}]
set_property PACKAGE_PIN E2 [get_ports {led[22]}]
set_property PACKAGE_PIN D1 [get_ports {led[23]}]
set_property PACKAGE_PIN B1 [get_ports {led[24]}]
set_property PACKAGE_PIN B2 [get_ports {led[25]}]
set_property PACKAGE_PIN N3 [get_ports {led[26]}]
set_property PACKAGE_PIN M3 [get_ports {led[27]}]
set_property PACKAGE_PIN K3 [get_ports {led[28]}]
set_property PACKAGE_PIN H3 [get_ports {led[29]}]
set_property PACKAGE_PIN N4 [get_ports {led[30]}]
set_property PACKAGE_PIN L4 [get_ports {led[31]}]
set_property PACKAGE_PIN J4 [get_ports {led[32]}]
```

```
set_property IOSTANDARD LVCMOS18 [get_ports seg]
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 IOSTANDARD LVCMOS18 [get_ports which]
set_property PACKAGE_PIN N22 [get_ports {which[0]}]
set_property PACKAGE_PIN M21 [get_ports {which[1]}]
set_property PACKAGE_PIN M22 [get_ports {which[2]}]
set_property -dict {IOSTANDARD LVCMOS18 PACKAGE_PIN L21} [get_ports enable]
set_property -dict {IOSTANDARD LVCMOS18 PACKAGE_PIN H4} [get_ports clk]
```

```
# [Place 30-574] Poor placement for routing between an IO pin and BUFG.If this
# sub optimal condition is acceptable for this design, you may use the
# CLOCK_DEDICATED_ROUTE constraint in the .xdc file to demote this message to a
# WARNING. However, the use of this override is highly discouraged.
set_property CLOCK_DEDICATED_ROUTE FALSE [get_nets clk_IBUF]
set_property CLOCK_DEDICATED_ROUTE FALSE [get_nets swb_IBUF[1]]
```

```
set_property CLOCK_DEDICATED_ROUTE FALSE [get_nets swb_IBUF[2]]
set_property CLOCK_DEDICATED_ROUTE FALSE [get_nets swb_IBUF[3]]
set_property CLOCK_DEDICATED_ROUTE FALSE [get_nets swb_IBUF[4]]
set_property CLOCK_DEDICATED_ROUTE FALSE [get_nets swb_IBUF[5]]
set_property CLOCK_DEDICATED_ROUTE FALSE [get_nets swb_IBUF[6]]
```

六、实验分工（需写清楚每个人负责的工作，以及个人贡献在小组总工作量中的占比）

总计：高炜哲负责 40%的实验，周俊佐和颜伟鹏各负责 30%的实验。
具体分工如下：
高炜哲：代码编写，代码调试，代码仿真测试，报告修改
周俊佐：实验报告，测试数据，资料分析
颜伟鹏：错误分析，板卡调试。

七、实验收获（心得体会等，请以个人为单位分别写）

思考题：

1：在输入寄存器 CLK 端的逻辑中，与门的作用是什么?为什么写操作要引入系统时钟 clk 的下降沿，而读操作不需要时钟?

写端口控制信号是 Write_Reg 且结合边沿触发时才有效；
所有写入操作的输入信号:W_Addr、W_Data、Write_Reg 必须在时钟边沿来临时已经有效；
写控制信号 Write_Reg 与写地址 W_Addr 的译码信号(高电平有效)、系统时钟 clk 取反，三者进行“与”操作后，与门输出作为寄存器的打入时钟信号；
保证与门输出为 1 时可以将 W_Data 写入。

D 触发器只有在高电平才能写入；
写操作会改变寄存器的内容，必须加以限制，否则数据就不确定，无法使用；而读操作对数据

没有影响，随时都可以进行。

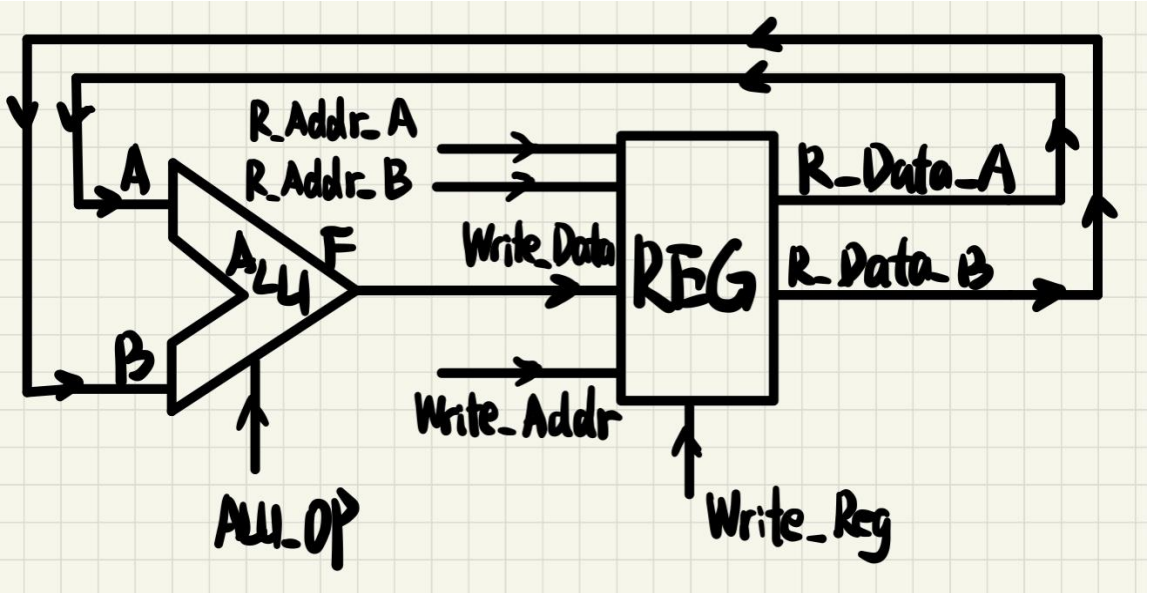
2：思考当控制信号和数据信号的数量不是 n:2"的关系时，如何改造多路选择器和数据分配器？用 Verilog HDL 如何实现设计？

对于 R13，他只有 7 个，大于 4=2^2，添加到成为 8=2^3；
然后正常进行前面 7 个控制信号的分配，最后的信号报错。
也可以采用拼接进行操作。

管脚约束的时候，用按键分别控制输入或输出，用按键次数来控制开关设备轮流输入 32 位数据，非 32 位的输入数据可以凑在一起，由第 n 次按键控制输入。也可用按键次数来控制 LED 灯/数码管轮流输出 32 位数据。

3：本实验设计的通用寄存器堆模块与实验二完成的带桶形移位器的多功能 ALU 运算器连接时应注意什么？请设计对 2 个寄存器操作数进行或运算并将结果存回寄存器的数据通路。

（1）应注意移位器输出的接口和 alu 输入的接口是否连接正确；在连接正确的前提下，保证移位器输出接口和 alu 输入接口的端口位数是都正确一致；最后通过测试，验证数据通路的正确性。



（2）

高炜哲 感悟

1. 我们采用了 PC 值单独输出的方式来输出 PC 值
优点：不会和其他数据混淆；

缺点：需要专门在确定一个数据口专供使用，会增加一点 I/0 功耗。

2. 我们组的代码架构和其他组略有不同，其他组是先确定地址，再根据不同模式分类，而我们的代码先根据模式，再根据地址。其缺点是代码量大大增加，几乎是前者的 2 倍，但优点是设计出来的电路的能耗比更好，具有更好的性能。

3. 本次代码量比起上次有显著增加，在编写代码的过程中，要先仔细观察电路图的输入和输出，确定输入输出的变量，再确定数字电路内部的逻辑，最后进行构思并撰写代码，这样代码的错误率会显著降低。

数据表见附件