**decode.v**

`timescale 1ns / 1ps

module decode (

input [3:0] a,

input en,

output reg [15:0] d

);

always @ (a or en) begin

if (en == 1'b0) // 使能被禁用

case (a)

4'b0000: d = 16'h0001;

4'b0001: d = 16'h0002;

4'b0010: d = 16'h0004;

4'b0011: d = 16'h0008;

4'b0100: d = 16'h0010;

4'b0101: d = 16'h0020;

4'b0110: d = 16'h0040;

4'b0111: d = 16'h0080;

4'b1000: d = 16'h0100;

4'b1001: d = 16'h0200;

4'b1010: d = 16'h0400;

4'b1011: d = 16'h0800;

4'b1100: d = 16'h1000;

4'b1101: d = 16'h2000;

4'b1110: d = 16'h4000;

4'b1111: d = 16'h8000;

default: d = 16'hFFFF; // 全部禁用

endcase

else

d = 16'h0000; // 禁用全部输出

end

endmodule

**regfile.v**

`timescale 1ns / 1ps

module regfile (

input clk,

input rst,

input enb,

input [3:0] din,

output reg [3:0] dout

);

always @(posedge clk) begin

if (rst) begin // 在复位信号为高电平时，将寄存器值清零

dout = 4'b0;

end else if (enb) begin // 当复位信号未激活且使能信号为高电平时，将输入数据保存到寄存器中

dout = din;

end

end

// 在时钟上升沿后，将寄存器值传递给输出端口

//assign dout = (enb) ? reg\_data : 4'bZ; // 只有在使能信号为高电平时才会向外传递当前的寄存器值

endmodule

**select.v**

`timescale 1ns / 1ps

module select (

input [3:0] D\_0, // 数据输入 0

input [3:0] D\_1, // 数据输入 1

input [3:0] D\_2, // 数据输入 2

input [3:0] D\_3, // 数据输入 3

input [3:0] D\_4, // 数据输入 4

input [3:0] D\_5, // 数据输入 5

input [3:0] D\_6, // 数据输入 6

input [3:0] D\_7, // 数据输入 7

input [3:0] D\_8, // 数据输入 8

input [3:0] D\_9, // 数据输入 9

input [3:0] D\_10, // 数据输入 10

input [3:0] D\_11, // 数据输入 11

input [3:0] D\_12, // 数据输入 12

input [3:0] D\_13, // 数据输入 13

input [3:0] D\_14, // 数据输入 14

input [3:0] D\_15, // 数据输入 15

input [3:0] S, // 输入地址线

input en, // 使能信号

output reg [3:0] SOUT// 输出选中数据

);

always @ (S or D\_0 or D\_1 or D\_2 or D\_3 or D\_4 or D\_5 or D\_6 or D\_7 or D\_8 or D\_9 or D\_10 or D\_11 or D\_12 or D\_13 or D\_14 or D\_15 or en) begin

if(en == 1'b0) // 确保使能信号为低电平有效

case(S)

4'b0000: SOUT = D\_0;

4'b0001: SOUT = D\_1;

4'b0010: SOUT = D\_2;

4'b0011: SOUT = D\_3;

4'b0100: SOUT = D\_4;

4'b0101: SOUT = D\_5;

4'b0110: SOUT = D\_6;

4'b0111: SOUT = D\_7;

4'b1000: SOUT = D\_8;

4'b1001: SOUT = D\_9;

4'b1010: SOUT = D\_10;

4'b1011: SOUT = D\_11;

4'b1100: SOUT = D\_12;

4'b1101: SOUT = D\_13;

4'b1110: SOUT = D\_14;

4'b1111: SOUT = D\_15;

default: SOUT = 16'h0000; // 全部禁用

endcase

else

SOUT = 16'h0000; // 禁用全部输出

end

endmodule

**top.v**

`timescale 1ns / 1ps

module top (

input clk, //寄存器组时钟信号，下降沿写入数据

input rst, //异步复位信号,高电平时全部寄存器置零

input we, //寄存器读写有效信号,低电平时允许寄存器写入数据，

//高电平时允许寄存器读出数据

input[3:0] raddr1, //所需读取的寄存器的地址

input[3:0] raddr2, //所需读取的寄存器的地址

input[3:0] waddr, //写寄存器的地址

input[3: 0] wdata, //写寄存器数据,数据在clk下降沿时被写入

output[3: 0] rdata1, //raddr1所对应寄存器的输出数据

output[3: 0] rdata2 //raddr2所对应寄存器的输出数据

);

// 实例化decode模块

wire [15:0] addr\_dec;

decode dec(.a(waddr), .en(we), .d(addr\_dec));

wire [3:0] reg\_outs [15:0];//定义一个4x4的二维 wire 数组

// 实例化4个寄存器模块

regfile reg1 (.clk(clk), .rst(rst), .enb(addr\_dec[0]), .din(wdata), .dout(reg\_outs[0]));

regfile reg2 (.clk(clk), .rst(rst), .enb(addr\_dec[1]), .din(wdata), .dout(reg\_outs[1]));

regfile reg3 (.clk(clk), .rst(rst), .enb(addr\_dec[2]), .din(wdata), .dout(reg\_outs[2]));

regfile reg4 (.clk(clk), .rst(rst), .enb(addr\_dec[3]), .din(wdata), .dout(reg\_outs[3]));

regfile reg5 (.clk(clk), .rst(rst), .enb(addr\_dec[4]), .din(wdata), .dout(reg\_outs[4]));

regfile reg6 (.clk(clk), .rst(rst), .enb(addr\_dec[5]), .din(wdata), .dout(reg\_outs[5]));

regfile reg7 (.clk(clk), .rst(rst), .enb(addr\_dec[6]), .din(wdata), .dout(reg\_outs[6]));

regfile reg8 (.clk(clk), .rst(rst), .enb(addr\_dec[7]), .din(wdata), .dout(reg\_outs[7]));

regfile reg9 (.clk(clk), .rst(rst), .enb(addr\_dec[8]), .din(wdata), .dout(reg\_outs[8]));

regfile reg10 (.clk(clk), .rst(rst), .enb(addr\_dec[9]), .din(wdata), .dout(reg\_outs[9]));

regfile reg11 (.clk(clk), .rst(rst), .enb(addr\_dec[10]), .din(wdata), .dout(reg\_outs[10]));

regfile reg12 (.clk(clk), .rst(rst), .enb(addr\_dec[11]), .din(wdata), .dout(reg\_outs[11]));

regfile reg13 (.clk(clk), .rst(rst), .enb(addr\_dec[12]), .din(wdata), .dout(reg\_outs[12]));

regfile reg14 (.clk(clk), .rst(rst), .enb(addr\_dec[13]), .din(wdata), .dout(reg\_outs[13]));

regfile reg15 (.clk(clk), .rst(rst), .enb(addr\_dec[14]), .din(wdata), .dout(reg\_outs[14]));

regfile reg16 (.clk(clk), .rst(rst), .enb(addr\_dec[15]), .din(wdata), .dout(reg\_outs[15]));

// 实例化选择器模块并连接输入

//4选1数据选择器

select sel1(

.D\_0(reg\_outs[0]),

.D\_1(reg\_outs[1]),

.D\_2(reg\_outs[2]),

.D\_3(reg\_outs[3]),

.D\_4(reg\_outs[4]),

.D\_5(reg\_outs[5]),

.D\_6(reg\_outs[6]),

.D\_7(reg\_outs[7]),

.D\_8(reg\_outs[8]),

.D\_9(reg\_outs[9]),

.D\_10(reg\_outs[10]),

.D\_11(reg\_outs[11]),

.D\_12(reg\_outs[12]),

.D\_13(reg\_outs[13]),

.D\_14(reg\_outs[14]),

.D\_15(reg\_outs[15]),

.S(raddr1),

.en(~we),

.SOUT(rdata1)

);

select sel2(

.D\_0(reg\_outs[0]),

.D\_1(reg\_outs[1]),

.D\_2(reg\_outs[2]),

.D\_3(reg\_outs[3]),

.D\_4(reg\_outs[4]),

.D\_5(reg\_outs[5]),

.D\_6(reg\_outs[6]),

.D\_7(reg\_outs[7]),

.D\_8(reg\_outs[8]),

.D\_9(reg\_outs[9]),

.D\_10(reg\_outs[10]),

.D\_11(reg\_outs[11]),

.D\_12(reg\_outs[12]),

.D\_13(reg\_outs[13]),

.D\_14(reg\_outs[14]),

.D\_15(reg\_outs[15]),

.S(raddr2),

.en(~we),

.SOUT(rdata2)

);

endmodule

**toptest.v**

`timescale 1ns / 1ps

module toptest;

reg clk;

reg rst;

reg we;

reg[3:0] raddr1;

reg[3:0] raddr2;

reg[3:0] waddr;

reg[3:0] wdata;

wire[3: 0] rdata1;

wire[3: 0] rdata2;

top top(

.clk(clk),

.rst(rst),

.we(we),

.raddr1(raddr1),

.raddr2(raddr2),

.waddr(waddr),

.wdata(wdata),

.rdata1(rdata1),

.rdata2(rdata2)

);

initial begin

clk = 1'b0;

we = 1'b0;

rst = 1'b0;

waddr = 4'b0000;

wdata = 4'b1111;

#100

waddr = 4'b0001;

wdata = 4'b1001;

#100

waddr = 4'b0010;

wdata = 4'b0010;

#100

waddr = 4'b0011;

wdata = 4'b0010;

#100

waddr = 4'b0100;

wdata = 4'b0100;

#100

waddr = 4'b0101;

wdata = 4'b0110;

#100

waddr = 4'b0110;

wdata = 4'b0010;

#100

waddr = 4'b0111;

wdata = 4'b0010;

#100

waddr = 4'b1000;

wdata = 4'b0100;

#100

waddr = 4'b1001;

wdata = 4'b0110;

#100

waddr = 4'b1010;

wdata = 4'b0110;

#100

waddr = 4'b1011;

wdata = 4'b1010;

#100

waddr = 4'b1100;

wdata = 4'b0010;

#100

waddr = 4'b1101;

wdata = 4'b0110;

#100

waddr = 4'b1110;

wdata = 4'b0000;

#100

waddr = 4'b1111;

wdata = 4'b1110;

#100

we = 1'b1;

raddr1 = 4'b0000;

#100

raddr2 = 4'b0001;

#100

raddr1 = 4'b0010;

#100

raddr2 = 4'b0011;

#100

raddr1 = 4'b0100;

#100

raddr2 = 4'b0101;

we = 1'b1;

raddr1 = 4'b0110;

#100

raddr2 = 4'b0111;

#100

raddr1 = 4'b1000;

#100

raddr2 = 4'b1001;

#100

raddr1 = 4'b1010;

#100

raddr2 = 4'b1011;

we = 1'b1;

raddr1 = 4'b1100;

#100

raddr2 = 4'b1101;

#100

raddr1 = 4'b1110;

#100

raddr2 = 4'b1111;

#100

rst = 1'b1;

end

always #25 clk <= ~clk;

endmodule