一、实验内容

使用 Verilog 实现 RAM 的设计，并用 Logicsim 绘制 16 个 4 位寄存器组成的 RAM，验证逻辑。进行 verilog 建模并进行 Modelsim 波形仿真。本实验写入线与输出线不相同。

二、硬件逻辑图

该实验未要求

三、模块建模

利用行为描述的方式进行建模，通过监测时钟上升沿和 ena 下降沿进行输入输出控制，通过 wena 区分写入与读取行为，并进行相应赋值

```
module ram(
    input clk,
    input ena,
    input wena,
    input [4:0] addr,
    input [31:0] data_in,
    output reg [31:0]    data_out
    );
     reg [31:0] ram [31:0];
    always @(posedge clk or negedge ena)
        begin
            if(ena)
            begin
                if(wena)
                    ram[addr] <= data_in;      //wena:1, input data
                else data_out <= ram[addr];    //wena:0, output data
            end
            else
                data_out <= 32'bz;                  //ena:0 reset RAM to z
        end
endmodule
```

四、测试模块建模

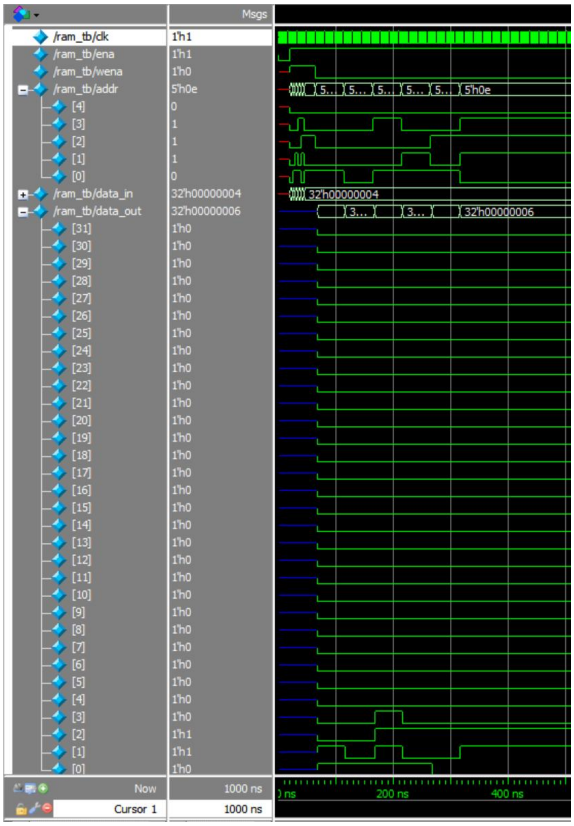
先进行 ena 端口的测试，确保 ena 无效时输出为高阻抗。之后进行写操作，对六个不同且较为随机的地址进行写入操作，之后统一进行输出操作，在输出时输出顺序打乱，从而检验是否 RAM 将值真正写入。

```
module ram_tb();
    reg clk;
    reg ena;
    reg wena;
    reg [4:0] addr;
    reg [31:0] data_in;
    wire [31:0] data_out;

    ram
    RAM(
        .clk(clk),
        .ena(ena),
        .wena(wena),
        .addr(addr),
        .data_in(data_in),
        .data_out(data_out)
    );

    //clk
    initial
    begin
        clk = 0;
        #1 clk = 1;
    end
    always #2 clk = ~clk;

    initial
    begin
        ena = 0;
        #20 ena = 1;

        wena = 1;
        addr = 5'b0000;
        data_in = 32'b1;
        #5
        addr = 5'b0001;
        data_in = 32'b0011;

        #5
```

```
        addr = 5'b0011;
        data_in = 32'b0101;

        #5
        addr = 5'b1001;
        data_in = 32'b1111;

         #5
        addr = 5'b1110;
        data_in = 32'b0110;

        #5
        addr = 5'b0101;
        data_in = 32'b0100;




        #20
        wena = 0;
        addr = 5'b0001;
        #50
        addr = 5'b0000;
        #50
        addr = 5'b1001;
        #50
        addr = 5'b0011;
        #50
        addr = 5'b0101;
        #50
        addr = 5'b1110;


    end


endmodule
```

五、实验结果

1 Modelsim 波形仿真



首先可以观察到当 ena 无效时，所得到的输出值为高阻抗

其次在写入之后，随机抽取已经写入位置进行读取操作，均按照相应的存储值进行输出，说明 RAM 将值正确存入，逻辑正确。

一、实验内容

使用 Verilog 实现 RAM 的设计，并用 Logicsim 绘制 16 个 4 位寄存器组成的 RAM，验证逻辑。进行 verilog 建模并进行 Modelsim 波形仿真。本实验写入线与输出线不相同，实现复用。

二、硬件逻辑图

该实验未要求

三、模块建模

与上一实验不同之处就在于输入输出线变成了一个，此时需要注意这时的 inout 类型在过程中不能直接赋值，可以用一个临时寄存器类型变量进行存储，最后用 assign 进行赋值给 wire 类型的 data 端口

```
module ram2(
    input clk,
    input ena,
    input wena,
    input [4:0] addr,
    inout [31:0] data
    );

    reg [31:0] ram [31:0];
    reg [31:0] temp;     //inout type cannot be assigned in procedural process, register needed

    always   @(posedge clk or negedge ena)
    begin
        if(ena)
        begin
            if(wena) ram[addr] <= data;
            else temp <= ram[addr];
        end
        else temp <= 32'bz;
    end

    assign data = temp;
endmodule
```

四、测试模块建模

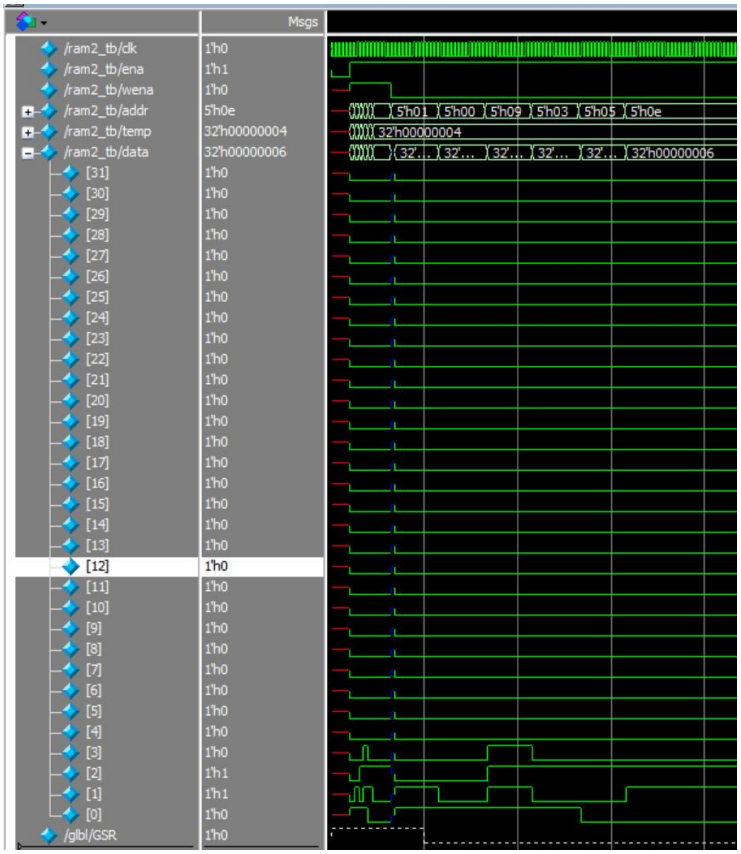
与上一实验大致相同，可以进行端口修改后重复使用

```
module ram2_tb();
reg clk;
	reg ena;
	reg wena;
	reg [4:0] addr;
	reg [31:0] temp;
	wire [31:0] data;

	ram2
	RAM(
		.clk(clk),
		.ena(ena),
		.wena(wena),
		.addr(addr),
		.data(data)
	);

	//clk
	initial
	begin
		clk = 0;
		#1 clk = 1;
	end
	always #2 clk = ~clk;

      assign data = wena?temp:32'bz;

	initial
	begin
		ena = 0;
		#20 ena = 1;

		wena = 1;
		addr = 5'b0000;
		temp = 32'b1;
		#5
		addr = 5'b0001;
		temp = 32'b0011;

		#5
		addr = 5'b0011;
		temp = 32'b0101;

		#5
```

```
        addr = 5'b1001;
        temp = 32'b1111;

          #5
        addr = 5'b1110;
        temp = 32'b0110;

        #5
        addr = 5'b0101;
        temp = 32'b0100;


        #20
        wena = 0;
        addr = 5'b0001;
        #50
        addr = 5'b0000;
        #50
        addr = 5'b1001;
        #50
        addr = 5'b0011;
        #50
        addr = 5'b0101;
        #50
        addr = 5'b1110;


    end
Endmodule
```

五、实验结果

1.Modelsim 仿真



可以看到输出部分正常进行输出，对应相应的存储量，说明逻辑正确

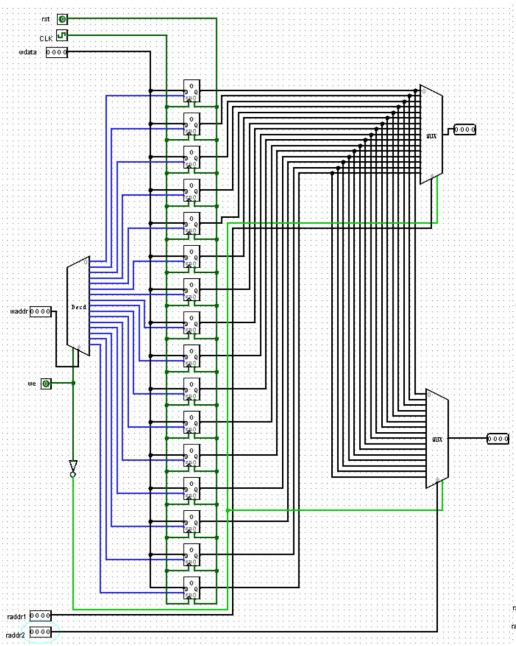
一、实验内容

利用 Verlog 实现寄存器堆的仿真建模并进行波形验证。该实验要求使用之前已经建立好的模块：数据分配器、数据选择器、pc 寄存器的实例化进行建模。

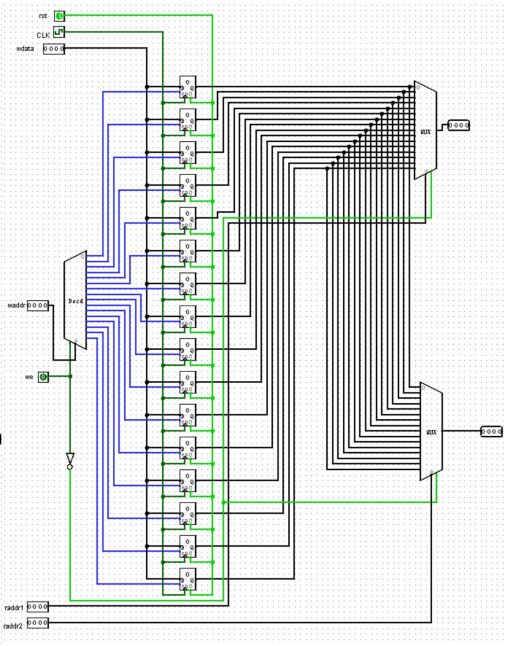
二、硬件逻辑图

1. Logicsim 逻辑验证

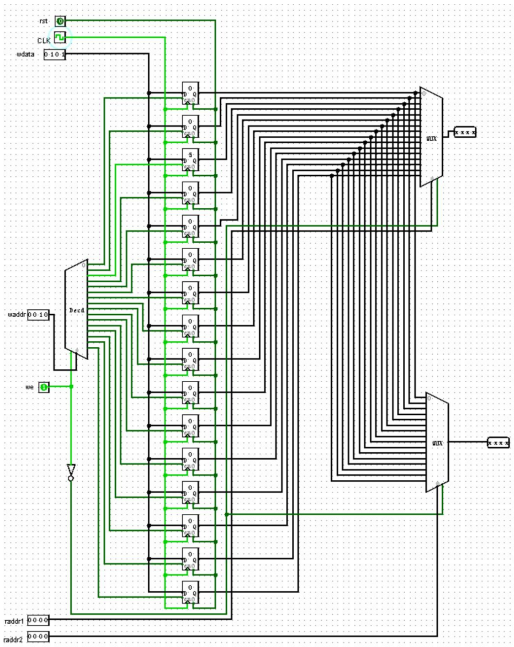
初始状态：



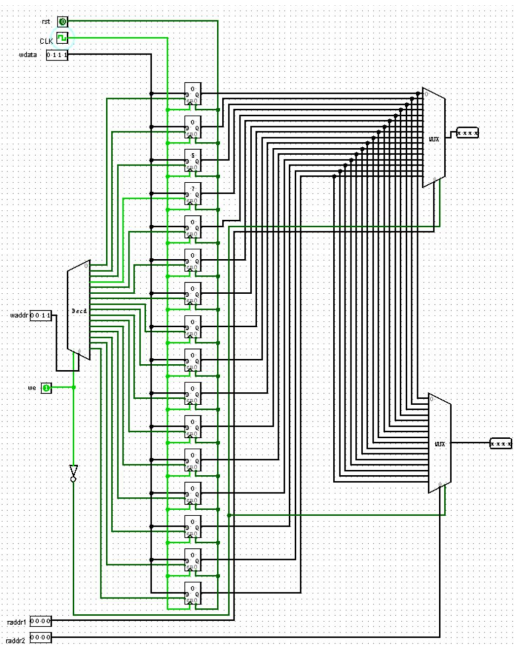
进行清零操作



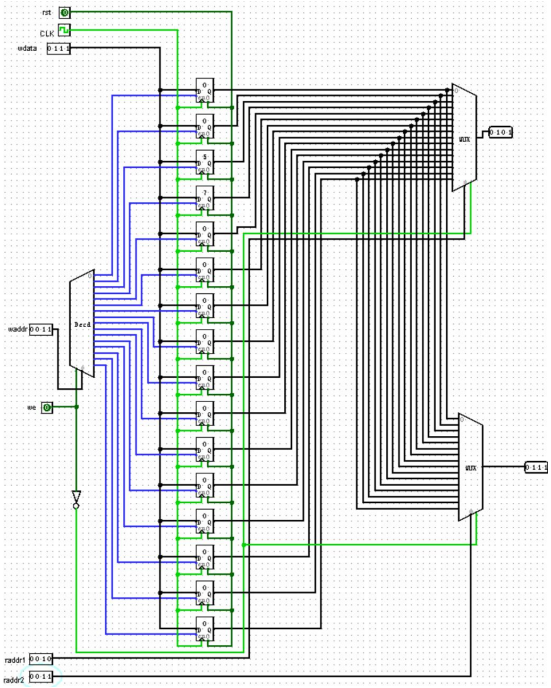
进行打开写使能端，向 2 号寄存器写入 5



向 3 号寄存器写入 7

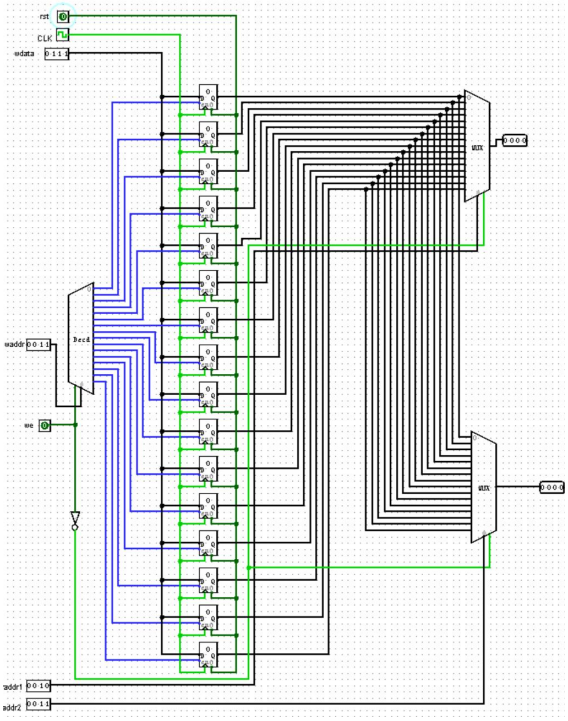


关闭写使能端，选择一号输出端输出 2 号寄存器内容，二号输出端输出 3 号寄存器内容



可以看到，2 号寄存器存储的 5 正常输出到一号输出端，3 号寄存器存储的 7 正常输出到二号输出端

进行清零操作，RAM 存储内容内容清零



三、模块建模

通过实例化数据分配器、数据选择器、pc 寄存器模块，通过连接模块间的接口实现寄存器堆的建模。注意这里时当时钟下降沿写入数据，清零数据时清零信号高电平进行信号得到清零。

```
module decoder5to32(
    input ena,
    input [4:0] iData,
    output [31:0] oData
     );
     assign oData[0] = ~iData[4] & ~iData[3] & ~iData[2] & ~iData[1] & ~iData[0] & ena;
     assign oData[1] = ~iData[4] & ~iData[3] & ~iData[2] & ~iData[1] & iData[0] & ena;
     assign oData[2] = ~iData[4] & ~iData[3] & ~iData[2] & iData[1] & ~iData[0] & ena;
     assign oData[3] = ~iData[4] & ~iData[3] & ~iData[2] & iData[1] & iData[0] & ena;
     assign oData[4] = ~iData[4] & ~iData[3] & iData[2] & ~iData[1] & ~iData[0] & ena;
     assign oData[5] = ~iData[4] & ~iData[3] & iData[2] & ~iData[1] & iData[0] & ena;
     assign oData[6] = ~iData[4] & ~iData[3] & iData[2] & iData[1] & ~iData[0] & ena;
     assign oData[7] = ~iData[4] & ~iData[3] & iData[2] & iData[1] & iData[0] & ena;
     assign oData[8] = ~iData[4] & iData[3] & ~iData[2] & ~iData[1] & ~iData[0] & ena;
     assign oData[9] = ~iData[4] & iData[3] & ~iData[2] & ~iData[1] & iData[0] & ena;
     assign oData[10] = ~iData[4] & iData[3] & ~iData[2] & iData[1] & ~iData[0] & ena;
     assign oData[11] = ~iData[4] & iData[3] & ~iData[2] & iData[1] & iData[0] & ena;
     assign oData[12] = ~iData[4] & iData[3] & iData[2] & ~iData[1] & ~iData[0] & ena;
     assign oData[13] = ~iData[4] & iData[3] & iData[2] & ~iData[1] & iData[0] & ena;
     assign oData[14] = ~iData[4] & iData[3] & iData[2] & iData[1] & ~iData[0] & ena;
     assign oData[15] = ~iData[4] & iData[3] & iData[2] & iData[1] & iData[0] & ena;
     assign oData[16] = iData[4] & ~iData[3] & ~iData[2] & ~iData[1] & ~iData[0] & ena;
     assign oData[17] = iData[4] & ~iData[3] & ~iData[2] & ~iData[1] & iData[0] & ena;
     assign oData[18] = iData[4] & ~iData[3] & ~iData[2] & iData[1] & ~iData[0] & ena;
     assign oData[19] = iData[4] & ~iData[3] & ~iData[2] & iData[1] & iData[0] & ena;
     assign oData[20] = iData[4] & ~iData[3] & iData[2] & ~iData[1] & ~iData[0] & ena;
     assign oData[21] = iData[4] & ~iData[3] & iData[2] & ~iData[1] & iData[0] & ena;
     assign oData[22] = iData[4] & ~iData[3] & iData[2] & iData[1] & ~iData[0] & ena;
     assign oData[23] = iData[4] & ~iData[3] & iData[2] & iData[1] & iData[0] & ena;
     assign oData[24] = iData[4] & iData[3] & ~iData[2] & ~iData[1] & ~iData[0] & ena;
     assign oData[25] = iData[4] & iData[3] & ~iData[2] & ~iData[1] & iData[0] & ena;
     assign oData[26] = iData[4] & iData[3] & ~iData[2] & iData[1] & ~iData[0] & ena;
     assign oData[27] = iData[4] & iData[3] & ~iData[2] & iData[1] & iData[0] & ena;
     assign oData[28] = iData[4] & iData[3] & iData[2] & ~iData[1] & ~iData[0] & ena;
     assign oData[29] = iData[4] & iData[3] & iData[2] & ~iData[1] & iData[0] & ena;
     assign oData[30] = iData[4] & iData[3] & iData[2] & iData[1] & ~iData[0] & ena;
     assign oData[31] = iData[4] & iData[3] & iData[2] & iData[1] & iData[0] & ena;
endmodule
```

```
module selector32to1(
	input [31:0] iC0,
	input [31:0] iC1,
	input [31:0] iC2,
	input [31:0] iC3,
	input [31:0] iC4,
	input [31:0] iC5,
	input [31:0] iC6,
	input [31:0] iC7,
	input [31:0] iC8,
	input [31:0] iC9,
	input [31:0] iC10,
	input [31:0] iC11,
	input [31:0] iC12,
	input [31:0] iC13,
	input [31:0] iC14,
	input [31:0] iC15,
	input [31:0] iC16,
	input [31:0] iC17,
	input [31:0] iC18,
	input [31:0] iC19,
	input [31:0] iC20,
	input [31:0] iC21,
	input [31:0] iC22,
	input [31:0] iC23,
	input [31:0] iC24,
	input [31:0] iC25,
	input [31:0] iC26,
	input [31:0] iC27,
	input [31:0] iC28,
	input [31:0] iC29,
	input [31:0] iC30,
	input [31:0] iC31,
	input [4:0] option,
	input ena,
	output reg [31:0] oZ
	);
	always @(*)
	begin
	if(ena)
	begin
		if (option == 0) oZ = iC0;
		else if (option == 1) oZ = iC1;
		else if (option == 2) oZ = iC2;
```

```
            else if (option == 3) oZ = iC3;
            else if (option == 4) oZ = iC4;
            else if (option == 5) oZ = iC5;
            else if (option == 6) oZ = iC6;
            else if (option == 7) oZ = iC7;
            else if (option == 8) oZ = iC8;
            else if (option == 9) oZ = iC9;
            else if (option == 10) oZ = iC10;
            else if (option == 11) oZ = iC11;
            else if (option == 12) oZ = iC12;
            else if (option == 13) oZ = iC13;
            else if (option == 14) oZ = iC14;
            else if (option == 15) oZ = iC15;
            else if (option == 16) oZ = iC16;
            else if (option == 17) oZ = iC17;
            else if (option == 18) oZ = iC18;
            else if (option == 19) oZ = iC19;
            else if (option == 20) oZ = iC20;
            else if (option == 21) oZ = iC21;
            else if (option == 22) oZ = iC22;
            else if (option == 23) oZ = iC23;
            else if (option == 24) oZ = iC24;
            else if (option == 25) oZ = iC25;
            else if (option == 26) oZ = iC26;
            else if (option == 27) oZ = iC27;
            else if (option == 28) oZ = iC28;
            else if (option == 29) oZ = iC29;
            else if (option == 30) oZ = iC30;
            else if (option == 31) oZ = iC31;
        end
    end

endmodule


module ADFF(
    input clk,
    input d,
    input rst,
    input ena,
    output reg Q1
);
    always @(negedge clk or posedge rst)
    begin
```

```
        if (rst)
        begin
            Q1 = 0;
        end

        else
        begin
            if (ena)
            begin
                Q1 = d;
            end
        end
    end
endmodule




module pcreg(
    input clk,
    input rst,
    input ena,
    input [31:0] data_in,
    output [31:0] data_out
    );
        ADFF p0(.clk(clk), .d(data_in[0]), .rst(rst), .ena(ena), .Q1(data_out[0]));
        ADFF p1(.clk(clk), .d(data_in[1]), .rst(rst), .ena(ena), .Q1(data_out[1]));
        ADFF p2(.clk(clk), .d(data_in[2]), .rst(rst), .ena(ena), .Q1(data_out[2]));
        ADFF p3(.clk(clk), .d(data_in[3]), .rst(rst), .ena(ena), .Q1(data_out[3]));
        ADFF p4(.clk(clk), .d(data_in[4]), .rst(rst), .ena(ena), .Q1(data_out[4]));
        ADFF p5(.clk(clk), .d(data_in[5]), .rst(rst), .ena(ena), .Q1(data_out[5]));
        ADFF p6(.clk(clk), .d(data_in[6]), .rst(rst), .ena(ena), .Q1(data_out[6]));
        ADFF p7(.clk(clk), .d(data_in[7]), .rst(rst), .ena(ena), .Q1(data_out[7]));
        ADFF p8(.clk(clk), .d(data_in[8]), .rst(rst), .ena(ena), .Q1(data_out[8]));
        ADFF p9(.clk(clk), .d(data_in[9]), .rst(rst), .ena(ena), .Q1(data_out[9]));
        ADFF p10(.clk(clk), .d(data_in[10]), .rst(rst), .ena(ena), .Q1(data_out[10]));
        ADFF p11(.clk(clk), .d(data_in[11]), .rst(rst), .ena(ena), .Q1(data_out[11]));
        ADFF p12(.clk(clk), .d(data_in[12]), .rst(rst), .ena(ena), .Q1(data_out[12]));
        ADFF p13(.clk(clk), .d(data_in[13]), .rst(rst), .ena(ena), .Q1(data_out[13]));
        ADFF p14(.clk(clk), .d(data_in[14]), .rst(rst), .ena(ena), .Q1(data_out[14]));
        ADFF p15(.clk(clk), .d(data_in[15]), .rst(rst), .ena(ena), .Q1(data_out[15]));
        ADFF p16(.clk(clk), .d(data_in[16]), .rst(rst), .ena(ena), .Q1(data_out[16]));
        ADFF p17(.clk(clk), .d(data_in[17]), .rst(rst), .ena(ena), .Q1(data_out[17]));
        ADFF p18(.clk(clk), .d(data_in[18]), .rst(rst), .ena(ena), .Q1(data_out[18]));
```

```
        ADFF p19(.clk(clk), .d(data_in[19]), .rst(rst), .ena(ena), .Q1(data_out[19]));
        ADFF p20(.clk(clk), .d(data_in[20]), .rst(rst), .ena(ena), .Q1(data_out[20]));
        ADFF p21(.clk(clk), .d(data_in[21]), .rst(rst), .ena(ena), .Q1(data_out[21]));
        ADFF p22(.clk(clk), .d(data_in[22]), .rst(rst), .ena(ena), .Q1(data_out[22]));
        ADFF p23(.clk(clk), .d(data_in[23]), .rst(rst), .ena(ena), .Q1(data_out[23]));
        ADFF p24(.clk(clk), .d(data_in[24]), .rst(rst), .ena(ena), .Q1(data_out[24]));
        ADFF p25(.clk(clk), .d(data_in[25]), .rst(rst), .ena(ena), .Q1(data_out[25]));
        ADFF p26(.clk(clk), .d(data_in[26]), .rst(rst), .ena(ena), .Q1(data_out[26]));
        ADFF p27(.clk(clk), .d(data_in[27]), .rst(rst), .ena(ena), .Q1(data_out[27]));
        ADFF p28(.clk(clk), .d(data_in[28]), .rst(rst), .ena(ena), .Q1(data_out[28]));
        ADFF p29(.clk(clk), .d(data_in[29]), .rst(rst), .ena(ena), .Q1(data_out[29]));
        ADFF p30(.clk(clk), .d(data_in[30]), .rst(rst), .ena(ena), .Q1(data_out[30]));
        ADFF p31(.clk(clk), .d(data_in[31]), .rst(rst), .ena(ena), .Q1(data_out[31]));


endmodule


module Regfiles(
    input clk,
    input rst,
    input we,
    input [4:0] raddr1,
    input [4:0] raddr2,
    input [4:0] waddr,
    input [31:0] wdata,
    output [31:0] rdata1,
    output [31:0] rdata2
    );

    wire [31:0] ram [31:0];
    wire [31:0] coding;

   decoder5to32 d(we,waddr,coding);
   pcreg p0(clk,rst,coding[0],wdata,ram[0]);
   pcreg p1(clk,rst,coding[1],wdata,ram[1]);
   pcreg p2(clk,rst,coding[2],wdata,ram[2]);
   pcreg p3(clk,rst,coding[3],wdata,ram[3]);
   pcreg p4(clk,rst,coding[4],wdata,ram[4]);
   pcreg p5(clk,rst,coding[5],wdata,ram[5]);
   pcreg p6(clk,rst,coding[6],wdata,ram[6]);
   pcreg p7(clk,rst,coding[7],wdata,ram[7]);
   pcreg p8(clk,rst,coding[8],wdata,ram[8]);
   pcreg p9(clk,rst,coding[9],wdata,ram[9]);
```

```
    pcreg p10(clk,rst,coding[10],wdata,ram[10]);
    pcreg p11(clk,rst,coding[11],wdata,ram[11]);
    pcreg p12(clk,rst,coding[12],wdata,ram[12]);
    pcreg p13(clk,rst,coding[13],wdata,ram[13]);
    pcreg p14(clk,rst,coding[14],wdata,ram[14]);
    pcreg p15(clk,rst,coding[15],wdata,ram[15]);
    pcreg p16(clk,rst,coding[16],wdata,ram[16]);
    pcreg p17(clk,rst,coding[17],wdata,ram[17]);
    pcreg p18(clk,rst,coding[18],wdata,ram[18]);
    pcreg p19(clk,rst,coding[19],wdata,ram[19]);
    pcreg p20(clk,rst,coding[20],wdata,ram[20]);
    pcreg p21(clk,rst,coding[21],wdata,ram[21]);
    pcreg p22(clk,rst,coding[22],wdata,ram[22]);
    pcreg p23(clk,rst,coding[23],wdata,ram[23]);
    pcreg p24(clk,rst,coding[24],wdata,ram[24]);
    pcreg p25(clk,rst,coding[25],wdata,ram[25]);
    pcreg p26(clk,rst,coding[26],wdata,ram[26]);
    pcreg p27(clk,rst,coding[27],wdata,ram[27]);
    pcreg p28(clk,rst,coding[28],wdata,ram[28]);
    pcreg p29(clk,rst,coding[29],wdata,ram[29]);
    pcreg p30(clk,rst,coding[30],wdata,ram[30]);
    pcreg p31(clk,rst,coding[31],wdata,ram[31]);
```

四、测试模块建模

首先将寄存器堆进行清零，之后向寄存器堆中写入数据，分别向对应编号的寄存器内写入 31-编号数的值，之后统一进行输出，输出时观察输出结果，输出结果应波形正好对称

```
module Regfiles_tb();
    reg clk;
    reg rst;
    reg we;
    reg [4:0] raddr1;
    reg [4:0] raddr2;
    reg [4:0] waddr;
    reg [31:0] wdata;
    wire [31:0] rdata1;
    wire [31:0] rdata2;

    Regfiles Rf(
            .clk(clk),
```

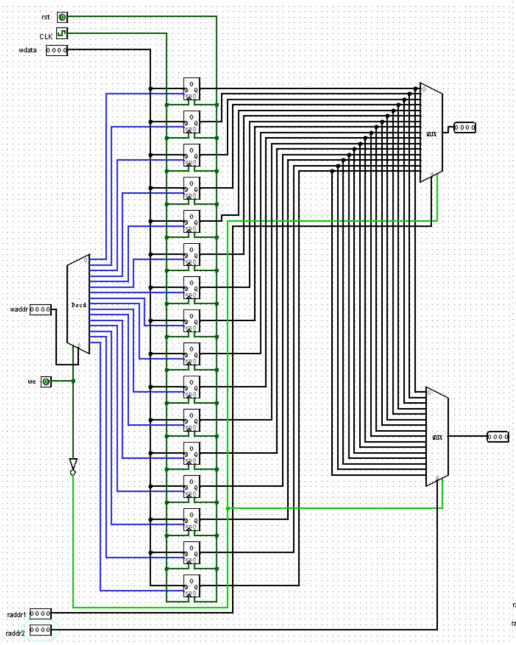
```
            .rst(rst),
            .we(we),
            .raddr1(raddr1),
            .raddr2(raddr2),
            .waddr(waddr),
            .wdata(wdata),
            .rdata1(rdata1),
            .rdata2(rdata2)
        );
integer i = 0;

initial
begin
  raddr1 = 5'b0;
raddr2 = 5'b0;
  //we = 1;
        clk = 0;
      rst = 0;
        #2 rst = 1;
        #2 rst =0;


end
always #2 clk = ~clk;

initial
begin
we = 0;
#20 we = 1;
        while(i <= 31)
        begin
            waddr = i;
            wdata = 31 - i;
            #10
            i = i + 1;
        end
        i = 0;
        we = 0;
        while(i <= 31)
        begin
            raddr1 = i;
            raddr2 = 31 - i;
            #20
            i = i + 1;
```

```
        end
    end
```

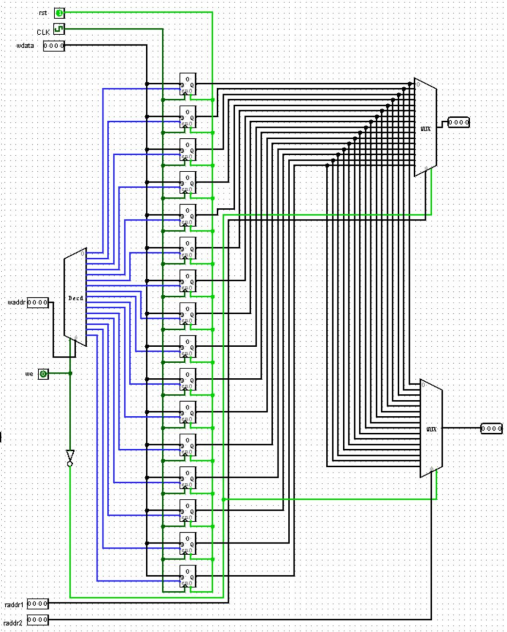
五、实验结果

1. Logicsim 逻辑验证

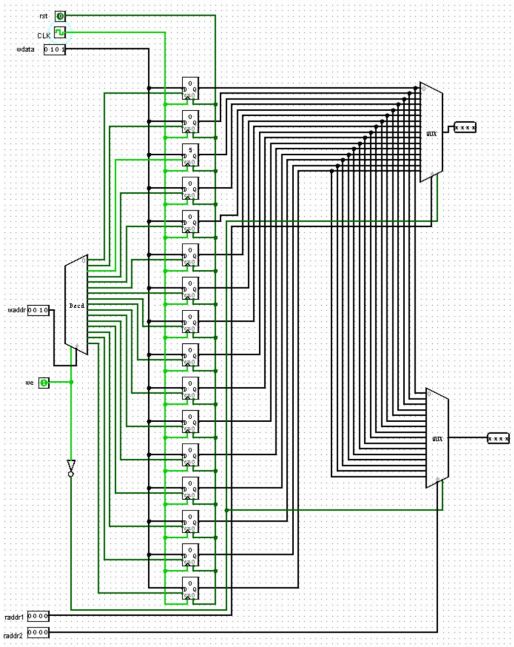
初始状态：



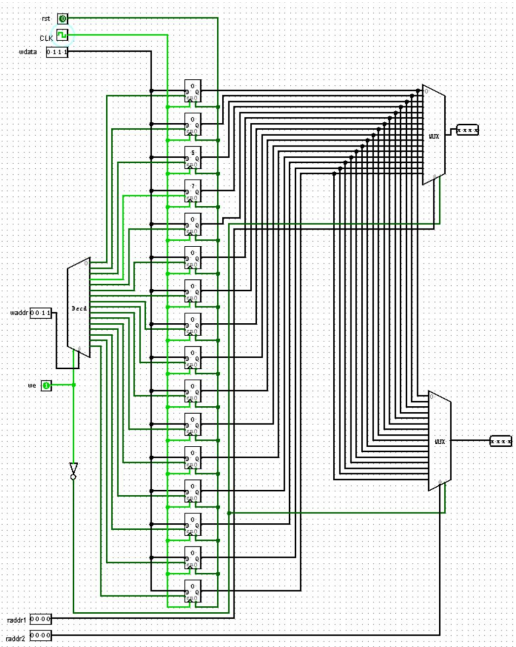
进行清零操作



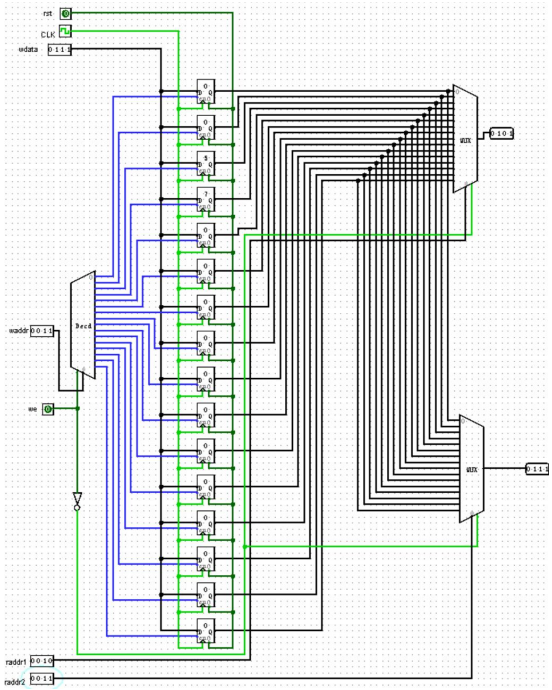
进行打开写使能端，向 2 号寄存器写入 5



向 3 号寄存器写入 7

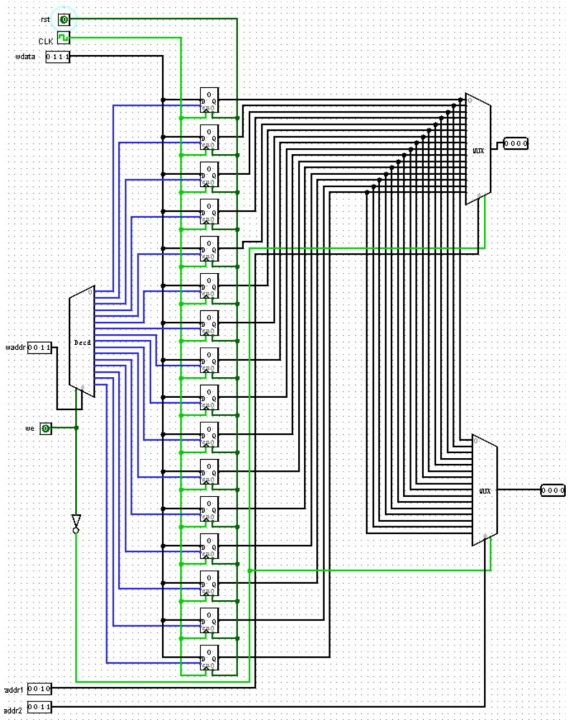


关闭写使能端，选择一号输出端输出 2 号寄存器内容，二号输出端输出 3 号寄存器内容

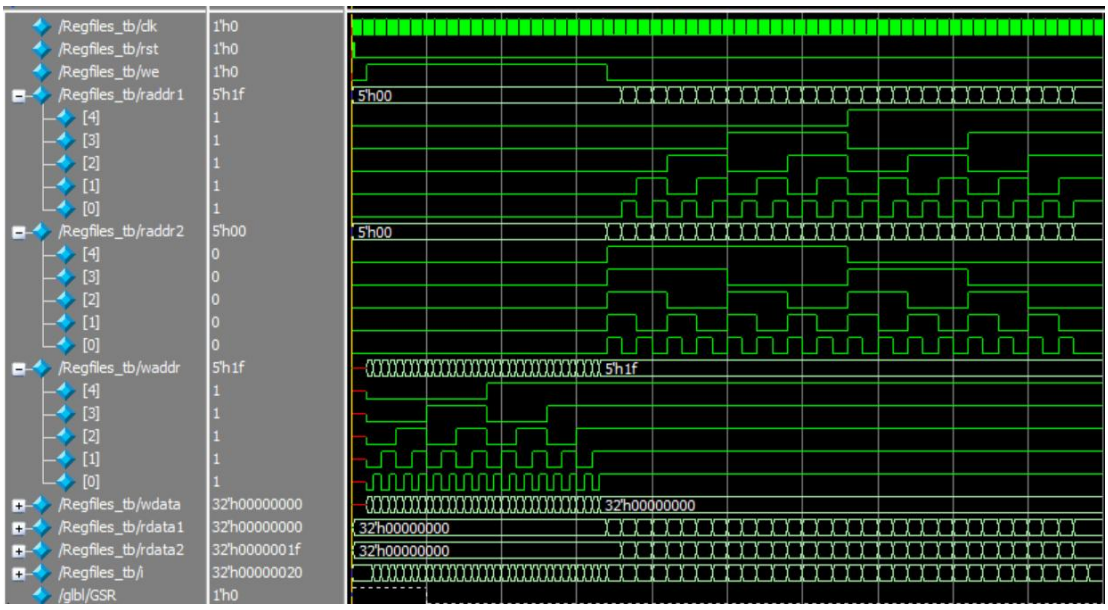


可以看到，2 号寄存器存储的 5 正常输出到一号输出端，3 号寄存器存储的 7 正常输出到二号输出端

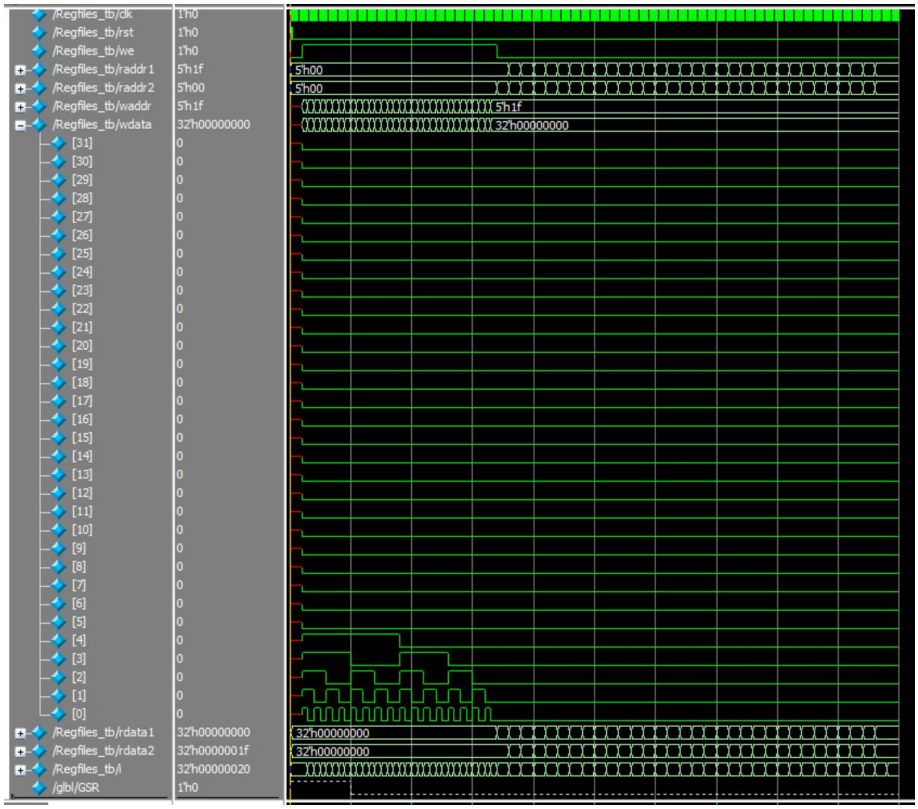
进行清零操作，RAM 存储内容内容清零



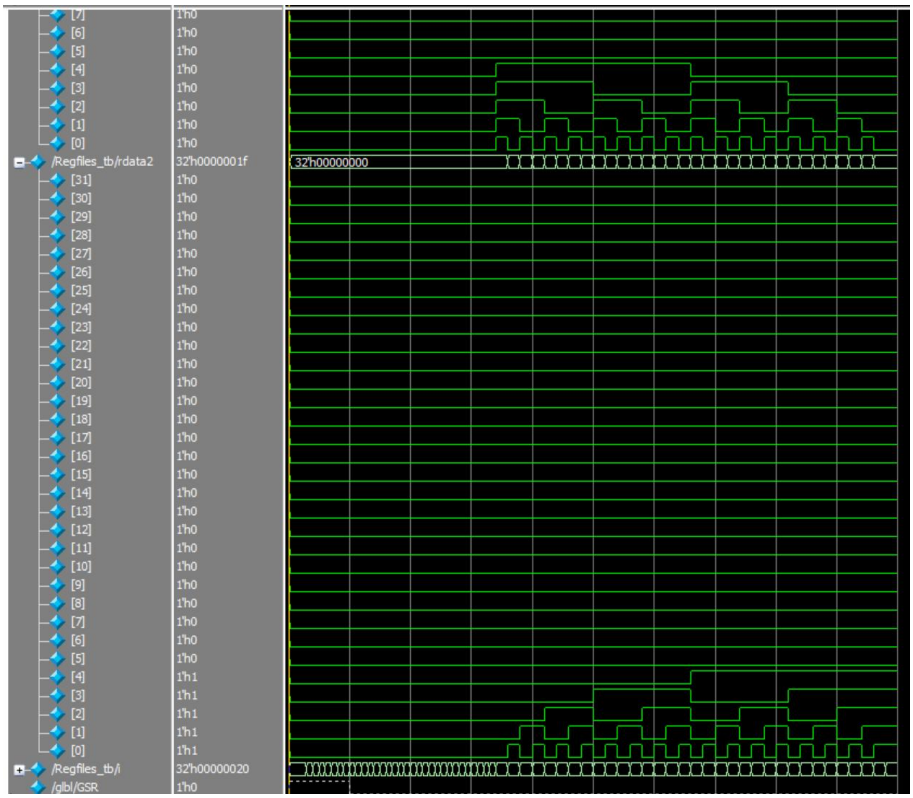
2. Modelsim 波形仿真



上图为读取地址以及写入地址的波形变化情况，实验中读取地址在以 15 为原点对称



该图为写入数据情况，实验中向相应序号单元寄存器中写入 31 减序号数的相应数据



观察波形图，前半部分出现过 rst 上升沿，寄存器堆中数据比清零，后可以看到两个读取地址读取的数据波形形状对称，读取相应值正确，波形验证正确。