同济大学计算机系

数字逻辑课程实验报告

实验八 **RAM** 与寄存器堆

学　　号 **2252941**

姓　　名 杨瑞灵

专　　业 计算机科学与技术

授课老师 张冬冬

一、实验内容

实验介绍

在本次实验中，我们将使用 Verilog HDL 语言实现 RAM 以及寄存器堆的设计和仿真。

实验目标

6.8_1

深入了解 RAM 与寄存器堆的原理。 λ 用 logicsim 画出一个包含 16 个寄存器的寄存器堆原理图。 λ 学习使用 Verilog HDL 语言设计实现 RAM 以及寄存器堆。 3.实验原理 1)RAM 半导体随机读写存储器，简称 RAM，它是数字计算机和其他数字系统的重要存储部件， 可存放大量的数据。图 6.8.1 所示为 RAM 的逻辑结构图，其主体是存储矩阵，另有地址译 码器和读写控制电路两大部分。读写控制电路中加有片选控制和输入输出缓冲器等，以便组 成双向 I/O 数据线

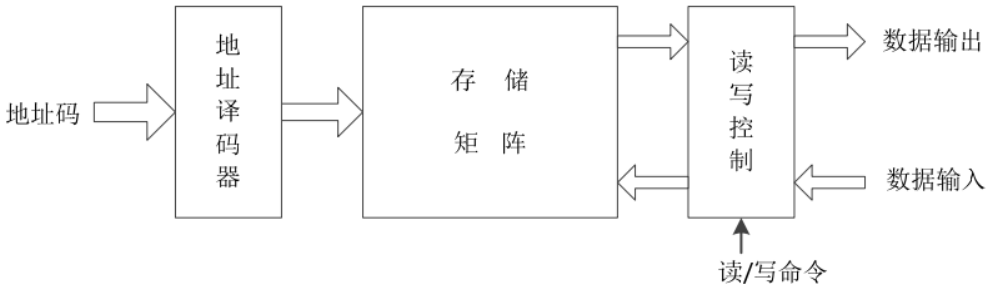


图 6.8.1 RAM 的逻辑结构图

RAM 有三组信号线：

地址线：单向，传送地址码（二进制数），以便按地址码访问存储单元；

数据线：双向，将数据码（二进制数）送入存储矩阵或从存储矩阵读出；

读/写命令线：单向控制线，分时发送这两个命令，要保证读时不写，写时不读。

如图 6.8.2 所示，为本实验所需要实现的 RAM 的示意图

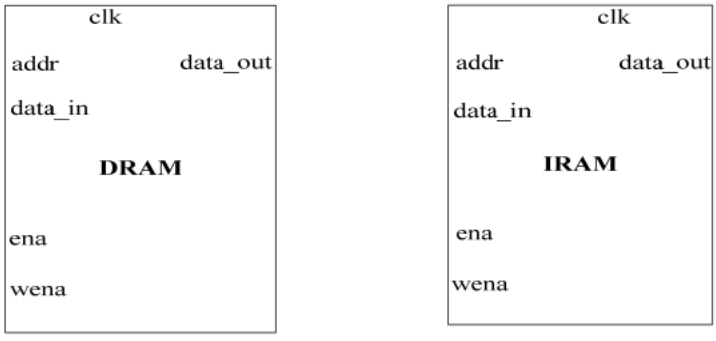


图 6.8.2 本实验的 RAM 示意图

● 接口定义：

```
module ram (
input clk,  //存储器时钟信号，上升沿时向 ram 内部写入数据
inputena, //存储器有效信号，高电平时存储器才运行，否则输出 z
input wena, //存储器读写有效信号，高电平为写有效，低电平为读有效，与 ena
                    同时有效时才可对存储器进行读写
input [4:0] addr, //输入地址，指定数据读写的地址
input [31:0] data_in, //存储器写入的数据，在 clk 上升沿时被写入
output [31:0] data_out //存储器读出的数据，
)
```

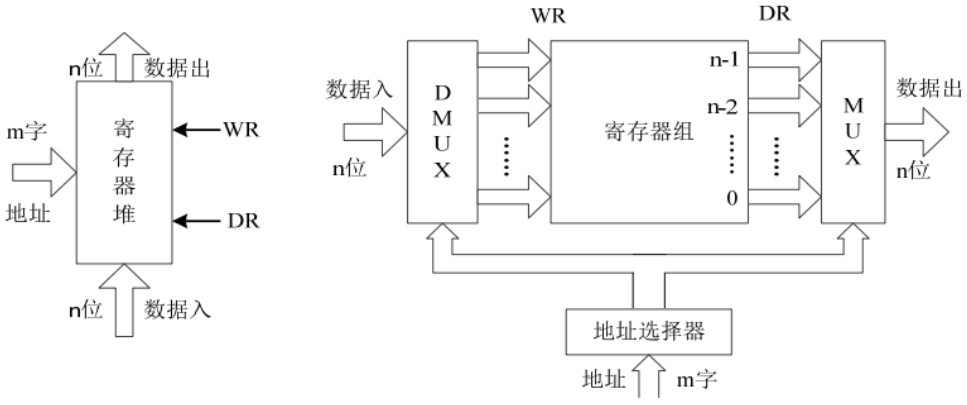
6.8_2

```
module ram2 (
    input clk,   //存储器时钟信号，上升沿时向 ram 内部写入数据
    input ena,   //存储器有效信号，高电平时存储器才运行，否则输出 z
    input wena,  //存储器读写有效信号，高电平为写有效，低电平为读有效，与
                   ena 同时有效时才可对存储器进行读写
    input [4:0] addr,   //输入地址，指定数据读写的地址
    inout [31:0] data,  //存储器数据线，可传输存储器读出或写入的数据。
                          写入的数据在 clk 上升沿时被写入
)
```

6.8_3 寄存器堆（regfiles）

一个寄存器是由 m 个触发器或锁存器按并行方式输入且并行方式输出连接而成。它只能记忆 1 个字，1 个字的长度等于 n 个比特。当需要记忆多个字时，一个寄存器就不够用了， 在这种情况下，这些集中使用的寄存器组的逻辑结构称为寄存器堆。

图 6.8.3 位寄存器堆的逻辑结构与原理示意图，它由寄存器组、地址译码器、多路选 择器 MUX 及多路分配器 DMUX 等部分组成。



（a）逻辑结构图（b）原理示意图

图 6.8.3 寄存器堆的逻辑结构

图 6.8.4 给出了由四个 4 位寄存器组成的寄存器堆原理图，其主要由 1 个 2-4 译码器、4 个 4 位寄存器和 2 个 4 位 4 选 1 选择器实现，其中

- we 信号连 2-4 译码器的使能端，使用 waddr 确定 we 信号被输入到哪个寄存器中；
- clk 信号连接入每个寄存器，控制寄存器的读写；
- rst 信号连接入每个寄存器，控制寄存器的复位；
- 所有寄存器的输出连接到两个 4 位 4 选 1 选择器，使用两个输入 raddr1，raddr2 控制输出的值

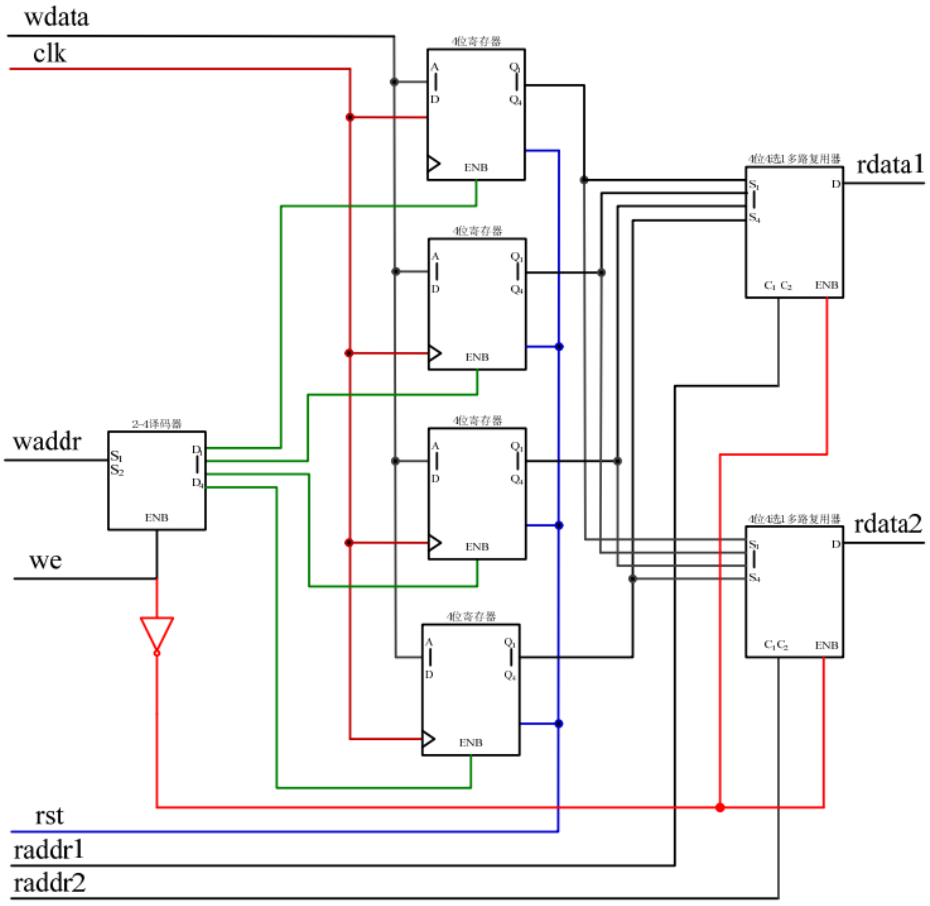


图 6.8.4 由 4 个寄存器组成的寄存器堆原理图

图 6.8.5 给出了本实验所要建模的寄存器堆的功能框图。

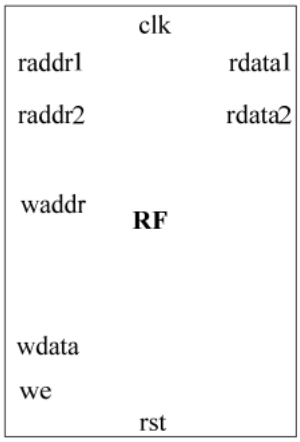


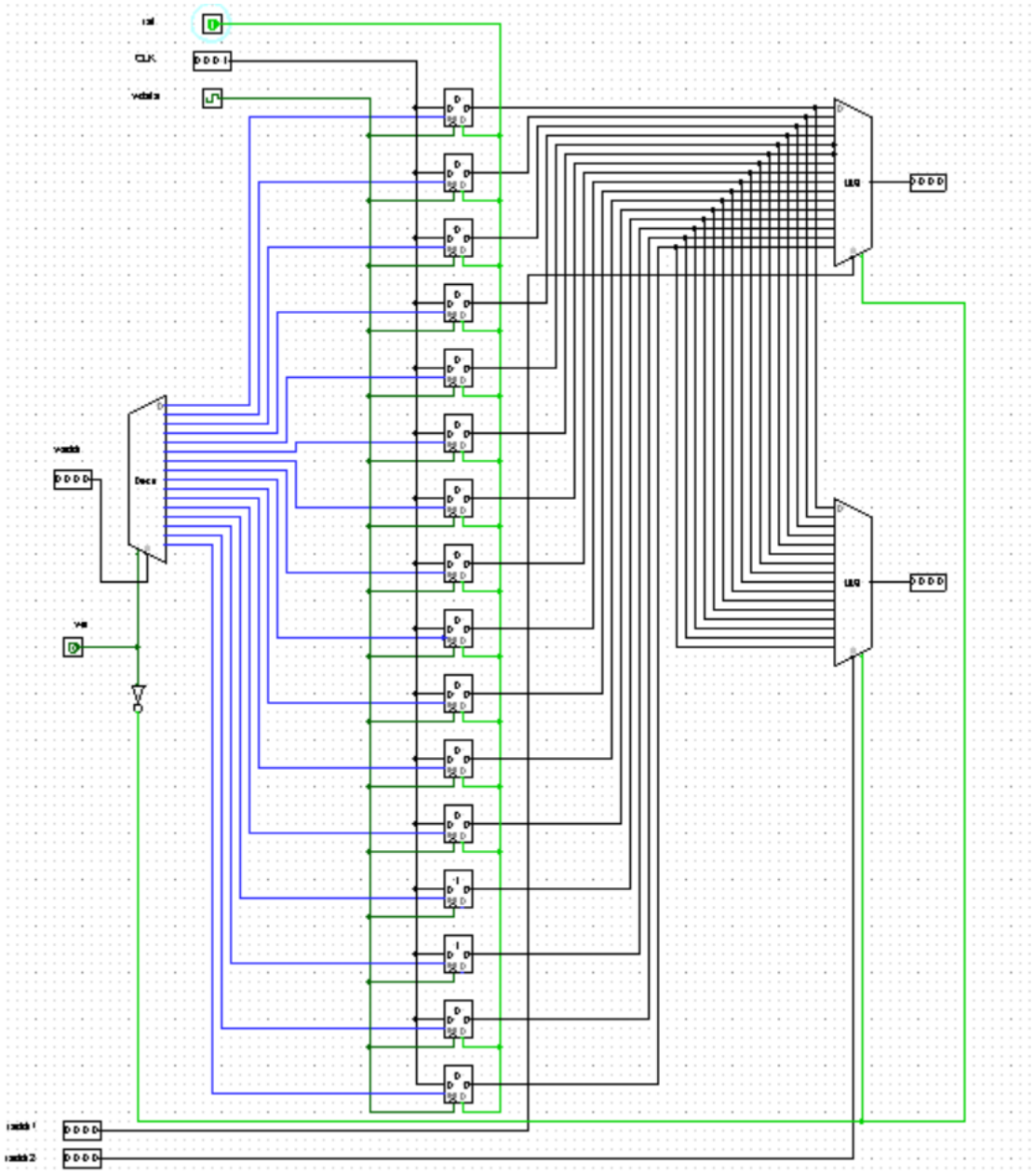
图 6.8.5 本实验所要建模的寄存器堆功能框图

```
module Regfiles(
input clk, //寄存器组时钟信号，下降沿写入数据
input rst, //reset 信号，异步复位，高电平时全部寄存器置零
 input we, //寄存器读写有效信号，高电平时允许寄存器写入数据，
                      低电平时允许寄存器读出数据
input [4:0] raddr1, //所需读取的寄存器的地址
input [4:0] raddr2, //所需读取的寄存器的地址
input [4:0] waddr, //写寄存器的地址
input [31:0] wdata, //写寄存器数据，数据在 clk 下降沿时被写入
    output [31:0] rdata1, //raddr1 所对应寄存器的输出数据
output [31:0] rdata2  //raddr2 所对应寄存器的输出数据
);
```

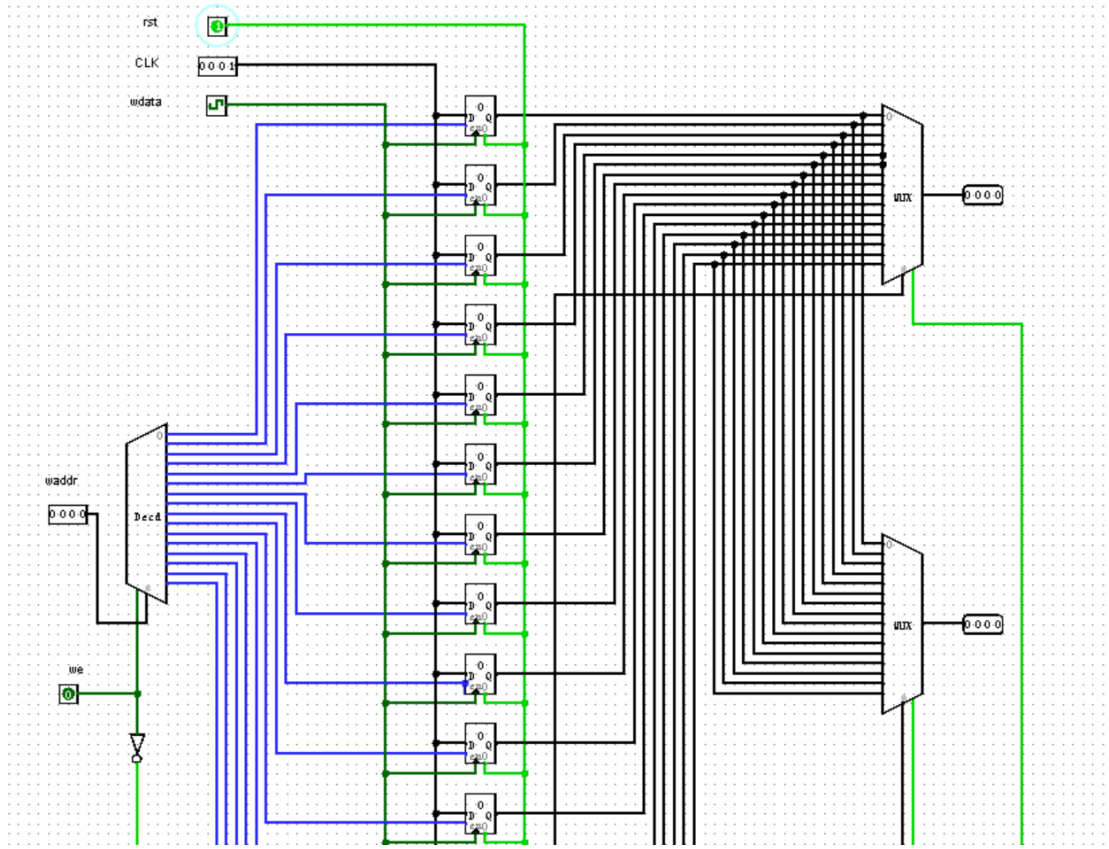
二、硬件逻辑图

6.8_3 寄存器堆（regfiles）

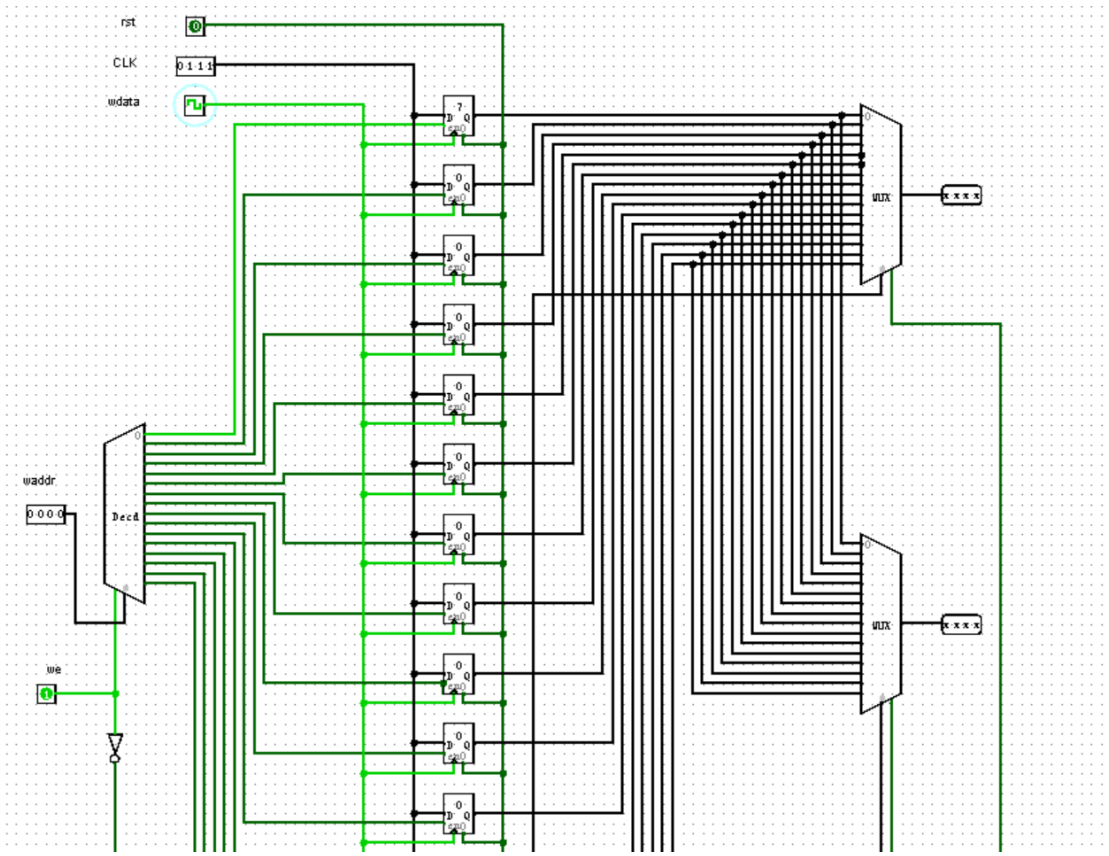
整体：


CLK
Decx
LEQ
LEQ

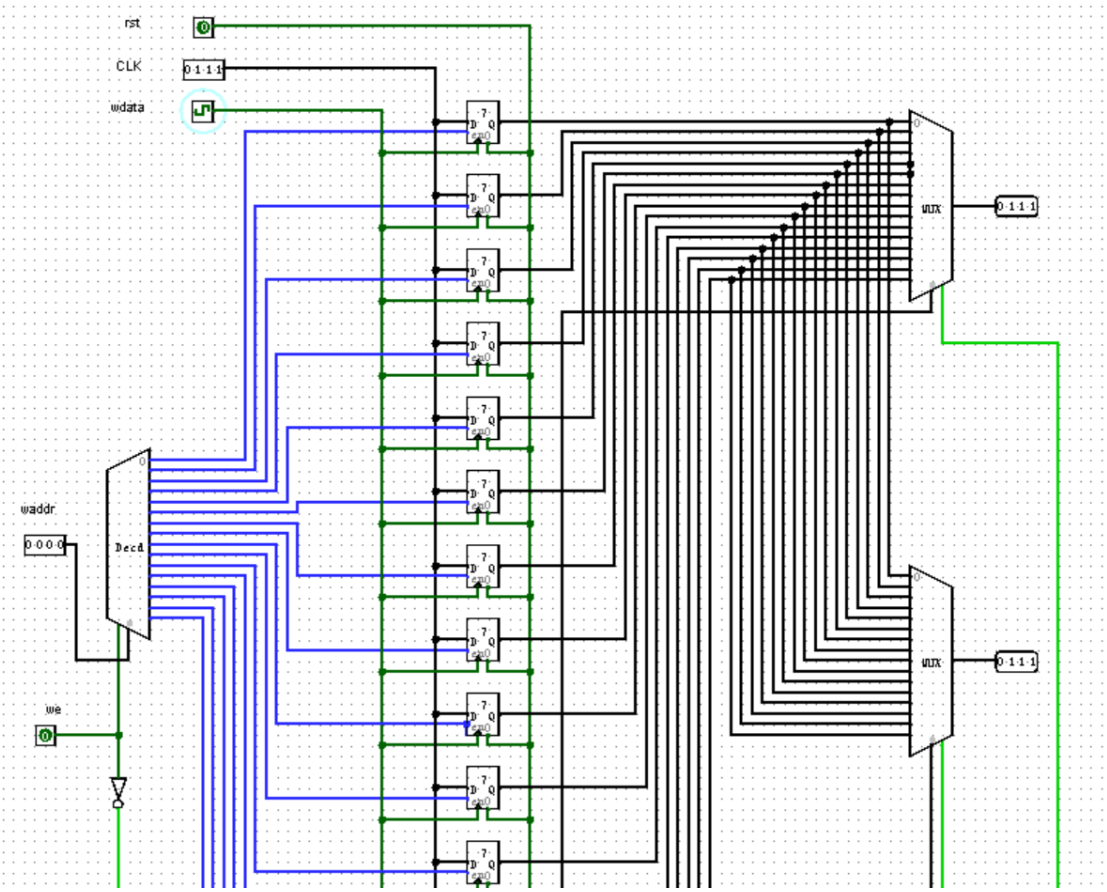
Rst=1 清零



Wdata=0111，we=1 写入



We=0 读出 0000 地址数据 0111



三、模块建模

6.8_1

```
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 [0:31];
        always @(posedge clk)
                begin
                if (ena)
                        begin
```

```
            if (wena)
                ram[addr] <= data_in;
            end
        end
    always @(*)
        begin
        if (ena&~wena)
            data_out <= ram[addr];
        else
            data_out<='bz;
    end
endmodule
```

6.8_2

```
module ram2(
    input clk,
    input ena,
    input wena,
    input [4:0] addr,
    inout [31:0] data
    );
    reg [31:0] ram [0:31];
    reg [31:0] temp;                    //inout type cannot be assigned in procedural process, register needed
  always @(posedge clk)
        begin
        if (ena)
            begin
            if (wena)
                ram[addr] <= data;
            else
                temp <= ram[addr];
            end
        else
            temp<='bz;
        end
    assign data = temp;
endmodule
```

6.8_3

```
module ram(
    input clk,//时钟
    input wena,// 1 是写入,0 是输出
    input [4:0] addr,//读出和写入的地址
    input [31:0] data_in,//写入的数据
    output reg [31:0] data_out,//读出的
    input rst//异步复位
    );
    //    ram ram1(clk,we,addr1,wdata,rdata1,rst);
    reg ena=1;
    reg [31:0] ram [0:31]; // 32 个寄存器  位宽是 32 位
    integer i;
    //  下降沿写入数据
    always @(negedge clk)
        begin
        if (ena)
            begin
            if (wena)
                ram[addr] <= data_in;
        end
    end
    always @(*) begin
        if (ena&~wena)
                data_out <= ram[addr];
        else
        data_out<='bz;
    end
    //异步复位，rst 等于 0 时全部寄存器清零
    always@(*)
        begin
        if(rst)
            begin
            for (i = 0; i < 32; i=i+1)
                ram[i] = 32'h00000000;
            end
        end
endmodule

module Regfiles(clk,rst,we,raddr1,raddr2,waddr,wdata,rdata1,rdata2);
    input clk;//时钟
    input rst;//置零，高电平时全部寄存器清零
    input we;//写入还是写出
    input [4:0] raddr1;//写出的地址 1
    input [4:0] raddr2;//写出的地址 2
```

```
    input [4:0] waddr;//准备写的寄存器的地址
    input [31:0] wdata;//写入的数据，下降沿写入
    output [31:0] rdata1;//寄存器对应的输出数据 1，由地址 raddr1 从寄存器 ram 中获取
    output [31:0] rdata2;//寄存器对应的输出数据 2，由地址 raddr2 从寄存器 ram 中获取
    reg [4:0] addr1;
    reg [4:0] addr2;
    reg [31:0] data_out;//接收，赋值给 rdata1 或者 radata2
    //相当于定义两个寄存器，两边同时进行操作，写入，但是输出 data_out 的不同
    ram ram1(clk,we,addr1,wdata,rdata1,rst);
    ram ram2(clk,we,addr2,wdata,rdata2,rst);
    always@(*)
        begin
        if(we)
            begin
            //写入
            addr1<=waddr;
            addr2<=waddr;
            end
        else
            begin
            //写出
            addr1<=raddr1;
            addr2<=raddr2;
            end
        end
endmodule
```

四、测试模块建模

6.8_1

```
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));
initial
```

```
begin
clk<=0;
ena<=0;
wena<=0;
end
always
#5 clk=~clk;

initial
begin
#8 ena<=1;
    wena<=1;
    addr<=4'b0000;
    data_in<=32'b1010_1010_1010_0000_0000_1111_0000_1111;
#10 wena<=0;

#10 wena<=1;
    addr<=4'b1010;
    data_in<=32'b0000_1010_1010_0000_0000_1111_0000_1111;
#10 wena<=0;

#10 wena<=1;
    addr<=4'b1111;
    data_in<=32'b1010_0000_0000_0000_0000_1111_0000_1111;
#10 wena<=0;

end
endmodule
```

6.8_2

```
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
```

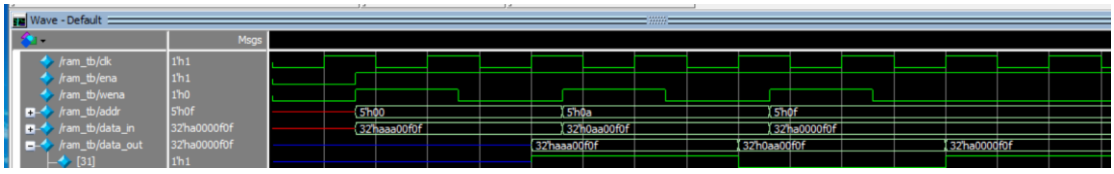
6.8_3

```
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;
```

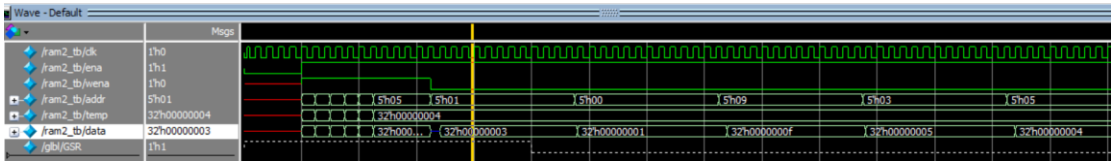
```
            raddr2 = 5'b0;
            clk = 0;
            rst = 0;
            #2 rst = 1;
            #2 rst =0;
        end
        always #2 clk = ~clk;

        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
endmodule
```

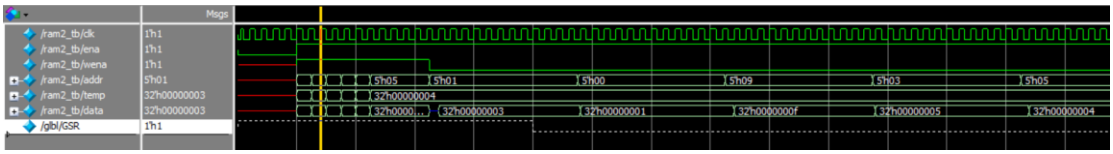
五、实验结果

6.8_1



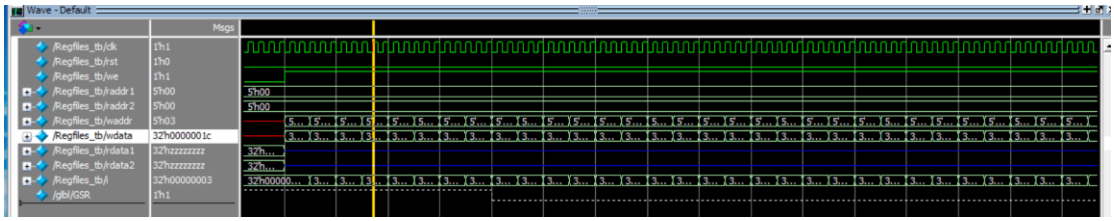
6.8_2


Wave - Default
Msgs
/ram2_tb/clk
/ram2_tb/ena
/ram2_tb/wena
/ram2_tb/addr
/ram2_tb/temp
/ram2_tb/data
/glbl/GSR

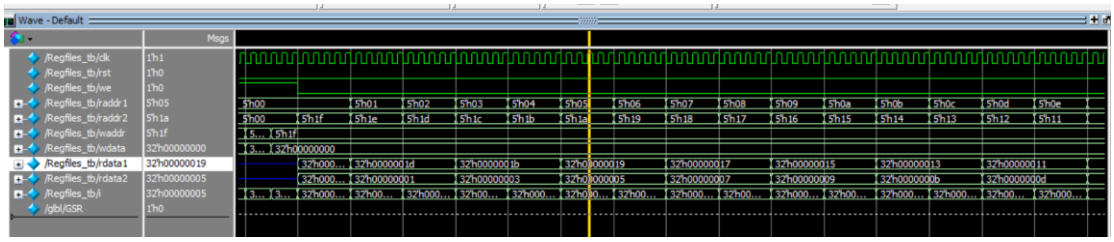

Msgs
/ram2_tb/clk
/ram2_tb/ena
/ram2_tb/wena
/ram2_tb/addr
/ram2_tb/temp
/ram2_tb/data
/glbl/GSR

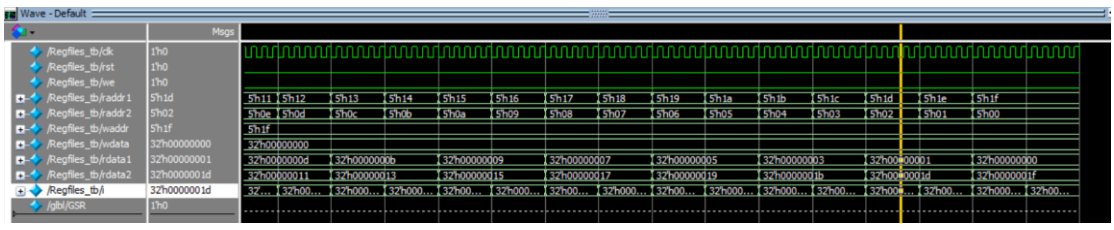
6.8_3

写入


Wave - Default
Msgs
/Regfiles_tb/clk
/Regfiles_tb/rst
/Regfiles_tb/we
/Regfiles_tb/raddr1
/Regfiles_tb/raddr2
/Regfiles_tb/waddr
/Regfiles_tb/wdata
/Regfiles_tb/rdata1
/Regfiles_tb/rdata2
/Regfiles_tb/i
/glbl/GSR

读出


Wave - Default
Msgs
/Regfiles_tb/clk
/Regfiles_tb/rst
/Regfiles_tb/we
/Regfiles_tb/raddr1
/Regfiles_tb/raddr2
/Regfiles_tb/waddr
/Regfiles_tb/wdata
/Regfiles_tb/rdata1
/Regfiles_tb/rdata2
/Regfiles_tb/i
/glbl/GSR


Wave - Default
Msgs
/Regfiles_tb/clk
/Regfiles_tb/rst
/Regfiles_tb/we
/Regfiles_tb/raddr1
/Regfiles_tb/raddr2
/Regfiles_tb/waddr
/Regfiles_tb/wdata
/Regfiles_tb/rdata1
/Regfiles_tb/rdata2
/Regfiles_tb/i
/glbl/GSR