同济大学计算机系

数字逻辑课程实验报告

学　　号 2154312

姓　　名 郑博远

专　　业 计算机科学与技术

授课老师 郭玉臣

一、实验内容

（1）RAM 实验：RAM，即半导体随机读写存储器，是数字计算机和其他数字系统的重要存储部件，可存放大量数据。RAM 的主体是存储矩阵，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。RAM 的信号线分为三组：地址线为单向，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/写命令线为单向，分时发送两个命令。本次实验中分别实现单端口与双端口的 RAM；

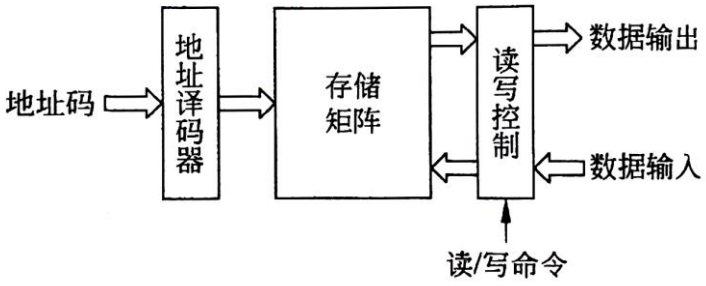
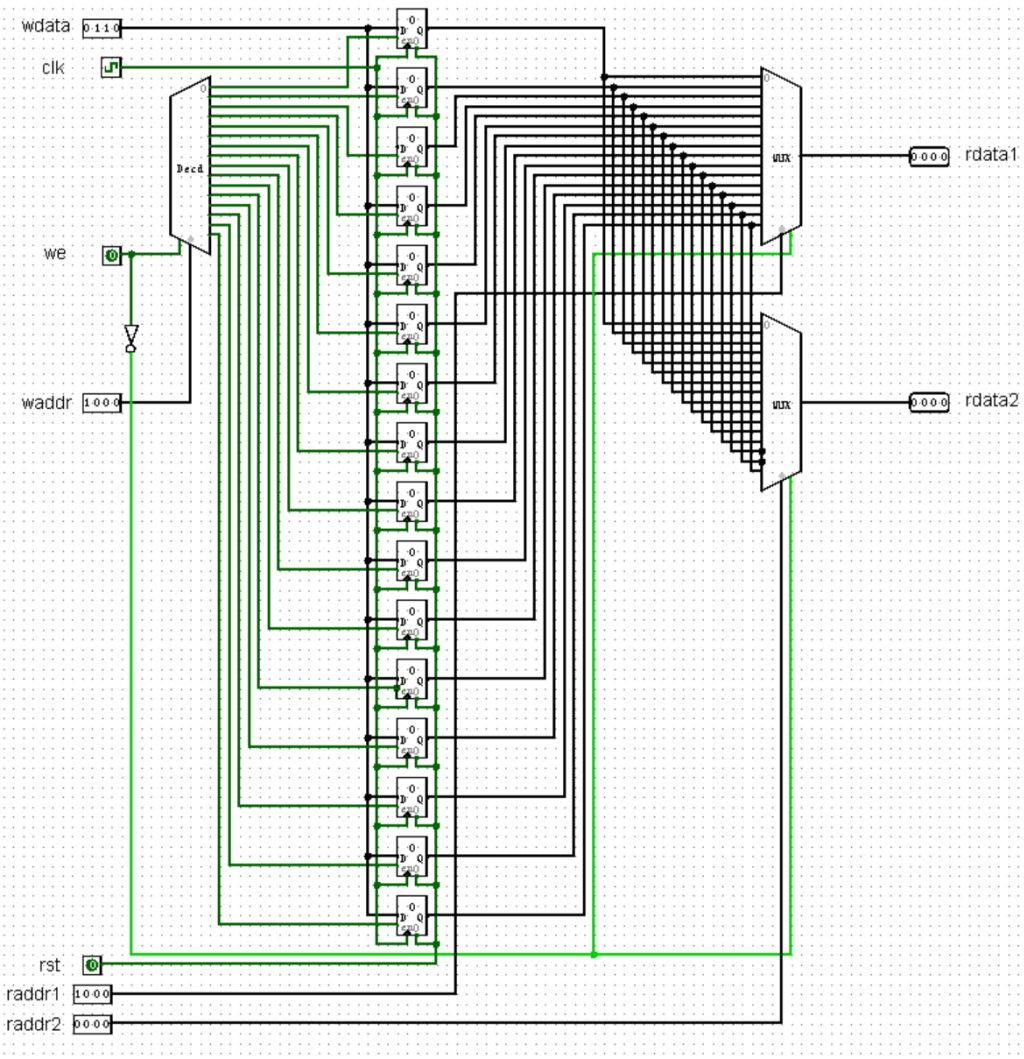


图 1　RAM 逻辑结构图

（2）寄存器堆实验（regfiles）： 一个寄存器是由 n 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。它只能记忆一个字，一个字的长度等于 n 个比特。当需要记忆生个字时，一个寄存器就不够用了。在这种情况下，需要使用由多个寄存器组成的寄存器堆。寄存器堆由寄存器组、地址译码器、多路选择器 MUX 及多路分配器 DMUX 等部分组成。向寄存器写数据或读数据，必须先给出寄存器的地址编号。写数据时,控制信号 WR 有效,待写入的数据经 DMUX 送到地址给定的某个寄存器；读数据时，控制信号 RD 有效,由地址给定的某个寄存器的数据内容经多路开关 MUX 送出。由于读写工作是分时进行的，所以寄存器组在逻辑上能满足写数据或读数据的需要。

二、硬件逻辑图

（1）寄存器堆实验：



三、模块建模

（1）RAM 实验（双端口）：

当时钟信号 CLK 在上升沿时，若 ena 为高电平有效，当 wena 为高电平写有效时，存储器从 data_in 中写入 addr 对应的输入地址；当 wena 为低电平写有效时，存储器从 addr 对应的地址读出到 data_out。若 ena 为低电平，则输出 z。

```
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)
    if(ena)
    begin
        if(wena)
            RAM[addr] <= data_in;
        else
            data_out <= RAM[addr];
    end
    else
    begin
        data_out = 32'bz;
    end

endmodule
```

（2）RAM 实验（单端口）：

当时钟信号 CLK 在上升沿时，若 ena 为高电平有效，当 wena 为高电平写有效时，存储器从 data 中写入 addr 对应的输入地址；当 wena 为低电平写有效时，存储器从 addr 对应的地址读出到 data。若 ena 为低电平，则输出 z。

```
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] inner_data;
    assign data = wena ? 32'bz : inner_data;

    always @(posedge clk)
```

```
    if(ena)
    begin
        if(wena)
            RAM[addr] <= data;
        else
            inner_data <= RAM[addr];
    end
    else
    begin
        inner_data = 32'bz;
    end

endmodule
```

（3）寄存器堆实验：

当时钟信号 CLK 在上升沿时，若 ena 为高电平有效，当 wena 为高电平写有效时，存储器从 data 中写入 addr 对应的输入地址；当 wena 为低电平写有效时，存储器从 addr 对应的地址读出到 data。若 ena 为低电平，则输出 z。

```
module decoder(
    input [4:0] iData,
    input iEna,
    output [31:0] oData
);

    assign oData = (iEna ? (1 << iData) : 32'b0);

endmodule
```

```
module selector32_1(
    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,
```

```
        begin
            case(iS)
            5'd0 : oZ = iC0;
            5'd1 : oZ = iC1;
            5'd2 : oZ = iC2;
            5'd3 : oZ = iC3;
            5'd4 : oZ = iC4;
            5'd5 : oZ = iC5;
            5'd6 : oZ = iC6;
            5'd7 : oZ = iC7;
            5'd8 : oZ = iC8;
```

```
    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] iS,
    output reg [31:0] oZ,
    input ena
    );
    always @(*)
    begin
        if(ena)
            5'd9 : oZ = iC9;
            5'd10: oZ = iC10;
            5'd11: oZ = iC11;
            5'd12: oZ = iC12;
            5'd13: oZ = iC13;
            5'd14: oZ = iC14;
            5'd15: oZ = iC15;
            5'd16: oZ = iC16;
            5'd17: oZ = iC17;
            5'd18: oZ = iC18;
            5'd19: oZ = iC19;
            5'd20: oZ = iC20;
            5'd21: oZ = iC21;
            5'd22: oZ = iC22;
            5'd23: oZ = iC23;
            5'd24: oZ = iC24;
            5'd25: oZ = iC25;
            5'd26: oZ = iC26;
            5'd27: oZ = iC27;
            5'd28: oZ = iC28;
            5'd29: oZ = iC29;
            5'd30: oZ = iC30;
            5'd31: oZ = iC31;
            endcase
        end
        else
            oZ = 32'bz;
    end
endmodule
```

```
module pcreg(
    input clk,
    input rst,
    input ena,
    input[31:0] data_in,
    output reg[31:0] data_out
    );
    always@ (posedge clk or posedge rst)
```

```
    begin
        if(rst)
            data_out <= 0;
        else if(ena)
            data_out <= data_in;
    end


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] decoder_out;
    wire [31:0] regfile_out [31:0];

    decoder decoder_inst(waddr, we, decoder_out);

    genvar i;
    generate
        for(i = 0; i < 32; i = i + 1)
            pcreg pcreg_inst(clk, rst, decoder_out[i], wdata,
regfile_out[i]);
    endgenerate

    selector32_1
selector32_1_inst(regfile_out[0],regfile_out[1],regfile_ou
t[2],regfile_out[3],
    regfile_out[4],regfile_out[5],regfile_out[6],regfile_ou
t[7],
```

```
    regfile_out[8],regfile_out[9],regfile_out[10],regfile_o
ut[11],
    regfile_out[12],regfile_out[13],regfile_out[14],regfile
_out[15],
    regfile_out[16],regfile_out[17],regfile_out[18],regfile
_out[19],
    regfile_out[20],regfile_out[21],regfile_out[22],regfile
_out[23],
    regfile_out[24],regfile_out[25],regfile_out[26],regfile
_out[27],
    regfile_out[28],regfile_out[29],regfile_out[30],regfile
_out[31], raddr1, rdata1, ~we);

    selector32_1
selector32_1_inst2(regfile_out[0],regfile_out[1],regfile_o
ut[2],regfile_out[3],
    regfile_out[4],regfile_out[5],regfile_out[6],regfile_ou
t[7],
    regfile_out[8],regfile_out[9],regfile_out[10],regfile_o
ut[11],
    regfile_out[12],regfile_out[13],regfile_out[14],regfile
_out[15],
    regfile_out[16],regfile_out[17],regfile_out[18],regfile
_out[19],
    regfile_out[20],regfile_out[21],regfile_out[22],regfile
_out[23],
    regfile_out[24],regfile_out[25],regfile_out[26],regfile
_out[27],
    regfile_out[28],regfile_out[29],regfile_out[30],regfile
_out[31], raddr2, rdata2, ~we);

endmodule
```

四、测试模块建模

（1）RAM 实验（单端口）：

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

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

    initial
    begin
    clk = 0;
        forever
        begin
            clk = #5 ~clk;
        end
    end

    initial
    begin
        ena = 0;
        #10;
        ena = 1;
        data_in = 0;
        wena = 1;
        for(addr = 0; addr < 31; addr = addr + 1)
        begin
            data_in = data_in + 1;
            #10;
        end

        wena = 0;
        for(addr = 0; addr < 31; addr = addr + 1)
        begin
            #10;
        end

        ena = 0;
    end

   ram ram_inst(clk, ena, wena, addr, data_in, data_out);

endmodule
```

（2）RAM 实验（双端口）：

```
`timescale 1ns / 1ps

module ram2_tb();
    reg clk, ena, wena;
    reg [4:0] addr;
    wire [31:0] data;
    reg [31:0] data_in;
    reg wr;

    initial
    begin
        clk = 0;
        forever
        begin
            clk = #5 ~clk;
        end
    end

    assign data = wr ? data_in : 'bz;

    initial
    begin
        ena = 1;
        wena = 1;
        wr = 1;
        data_in = 127;

        for(addr = 0; addr < 31; addr = addr + 1)
        begin
            data_in = data_in + 1;
            #10;
        end

        wena = 0;
        wr = 0;
        for(addr = 0; addr < 31; addr = addr + 1)
        begin
            #10;
        end

        ena = 0;
    end
```

```
    ram2 ram2_inst(clk, ena, wena, addr, data);
endmodule
```

（3）寄存器堆实验：

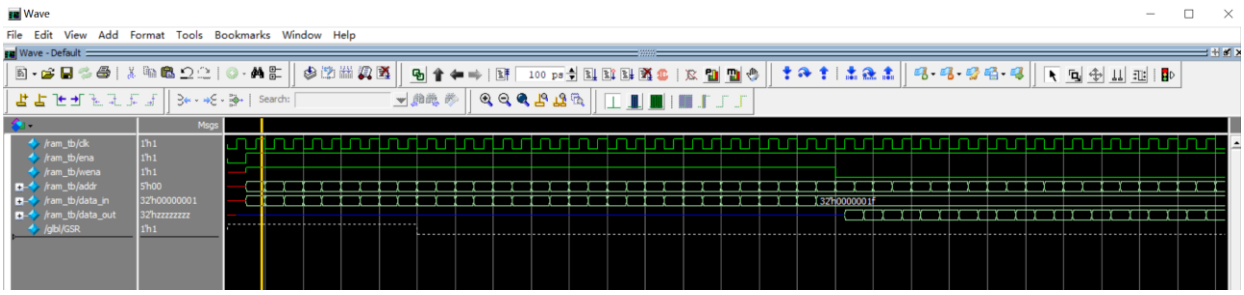
```
`timescale 1ns / 1ps

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;

    initial
    begin
        rst = 0;
        clk = 0;
        forever
        begin
            clk = #5 ~clk;
        end
    end

    initial
    begin
        we = 1;
        wdata = 32'b0;
        for(waddr = 0; waddr < 31; waddr = waddr + 1)
        begin
            wdata = wdata + 1;
            #10;
        end
        we = 0;
        for(raddr1 = 0; raddr1 < 31; raddr1 = raddr1 + 1)
        begin
```

```
            raddr2 = 30 - raddr1;
            #10;
        end
    end

    Regfiles Regfiles_inst(clk, rst, we, raddr1, raddr2,
waddr, wdata, rdata1, rdata2);

endmodule
```

五、实验结果

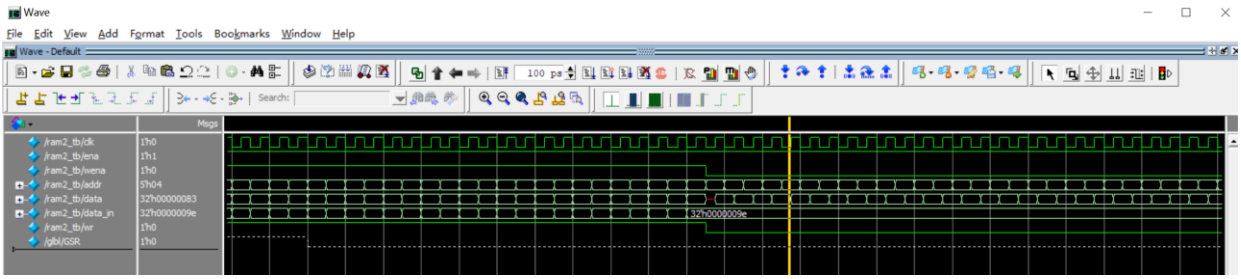
（1）RAM 实验（双端口）：

a) modelsim 仿真波形图



首先测试写入，ena 高电平有效，wena 高电平代表写有效：每当 CLK 上升沿到来时，从 data_in 读入数据到对应的 addr 地址处；然后测试读出，ena 高电平有效，wena 低电平代表读有效：每当 CLK 上升沿到来时，从对应的 addr 地址处读出数据到 data_out。

（2）RAM 实验（单端口）

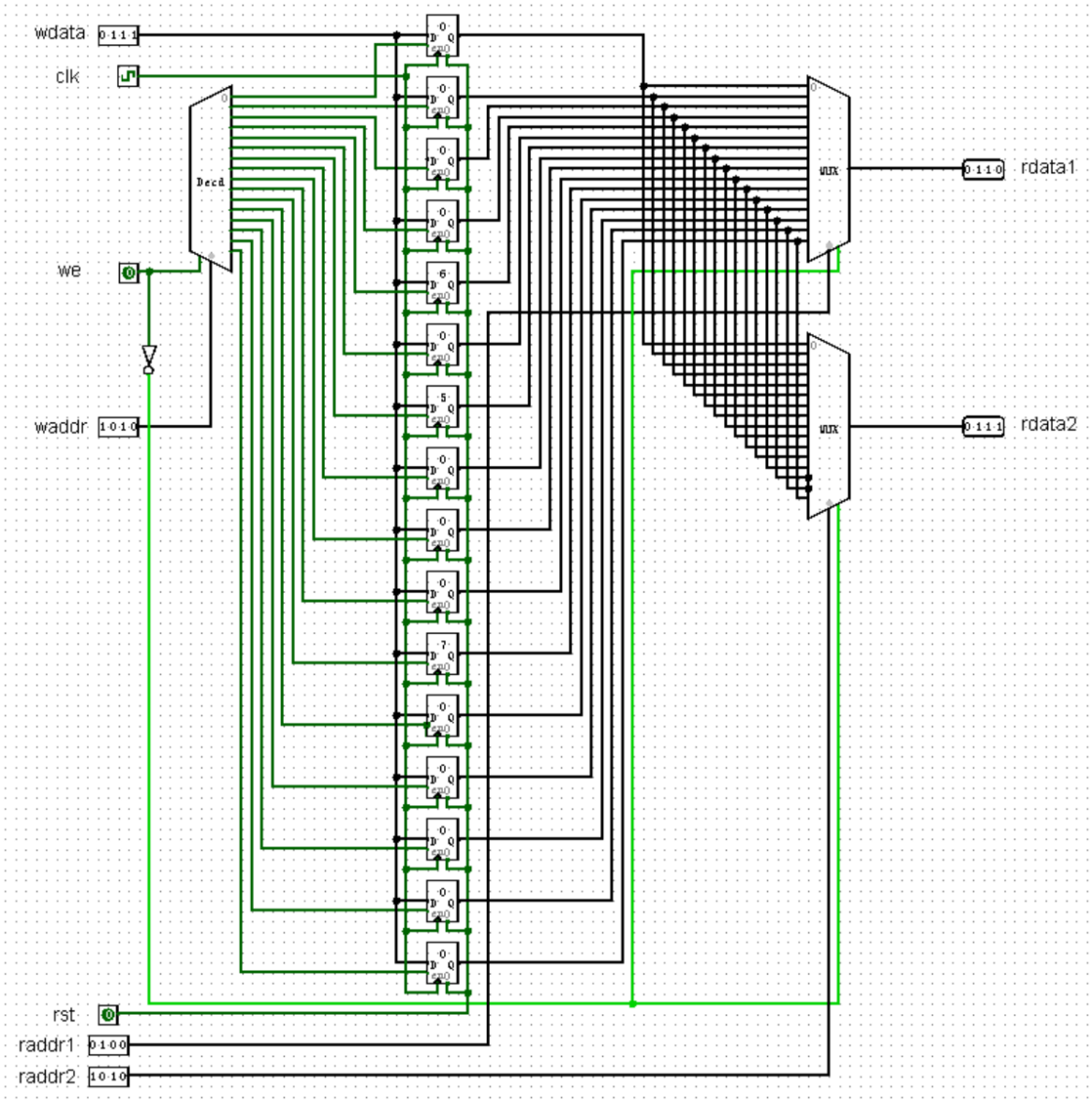


首先测试写入，ena 高电平有效，wena 高电平代表写有效：每当 CLK 上升沿到来时，从 data 读入数据到对应的 addr 地址处；然后测试读出，ena 高电平有

效，wena 低电平代表读有效：每当 CLK 上升沿到来时，从对应的 addr 地址处读出数据到 data。

（3）寄存器堆实验：

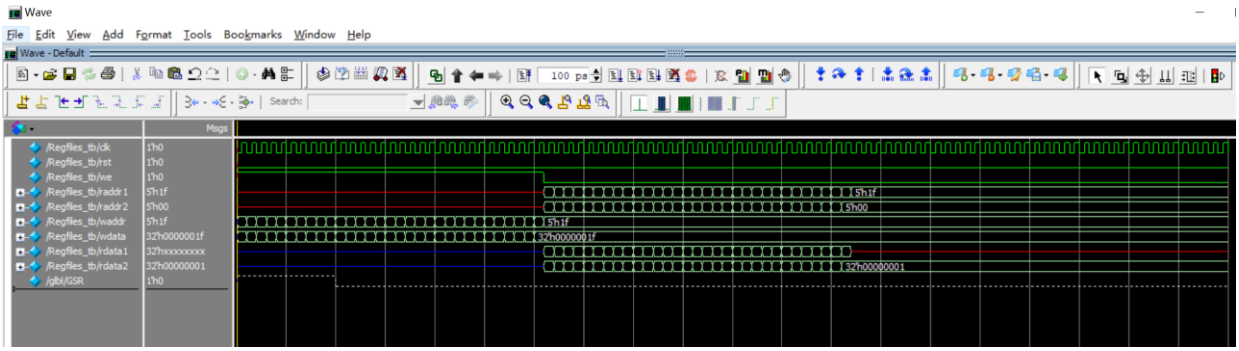
a) logisim 逻辑验证图



先将 rst 置 1，使得所有寄存器清零；然后将 rst 置 0 开始测试。先测试写入，将 we 置为高电平写有效。将 waddr 置为 0100，wdata 置为 0110，将 6 送入 4 号寄存器；将 waddr 置为 0110，wdata 置为 0101，将 5 送入 6 号寄存器；将 waddr 置为 1010，wdata 置为 0111，将 7 送入 10 号寄存器。然后测试读出，将 we 置

为低电平读有效。将 raddr1、raddr2 分别置为 0100 和 1010，分别读出 4 号、10 号寄存器中的内容，rdata1 与 rdata2 的值分别为 0110、0111。

b) modelsim 仿真波形图



首先测试写入，we 高电平代表写有效：每当 CLK 上升沿到来时，从 wdata 读入数据到对应的 waddr 地址处；然后测试读出，we 低电平代表读有效：从对应的 raddr1、raddr2 地址处读出数据到 rdata1、rdata2。