同济大学计算机系

数字逻辑课程实验报告

学　　号 2350752

姓　　名 田思宇

专　　业 计算机科学与技术

授课老师 张冬冬

一、实验内容

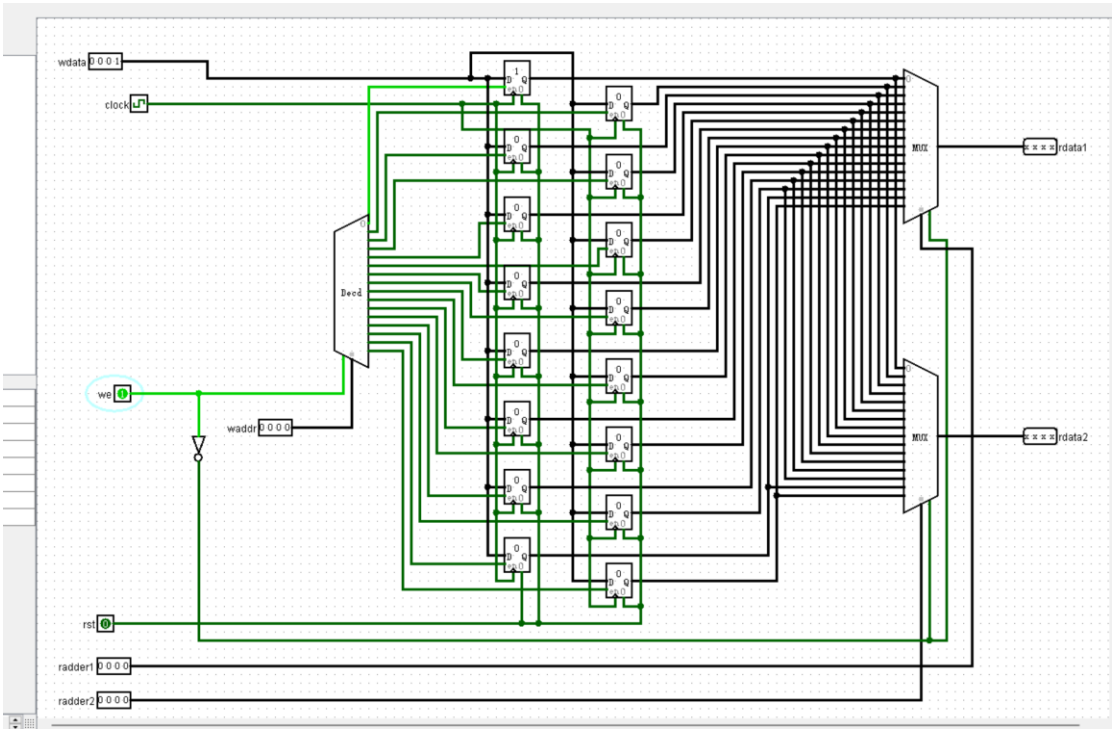
深入理解 RAM 和寄存器堆的底层电路逻辑，并且用 Verilog 语言实现这两个部件的描述，本次实验有三个模块，第一个模块是 RAM 输入和输出分两条线，第二个模块是 RAM 输入输出共用一条线，第三个模块是寄存器堆的实现。

1、 RAM：是计算机中重要的读写存储器，其底层电路是存储矩阵，然后通过行列地址来读取这些存储矩阵，这些存储矩阵可以是由寄存器构成（sram），也可以由电容构成（dram），所以要有译码器（翻译地址确定到底是哪一个行或列存储单元），同时还有选择器，选择到底是哪一个存储单元输出。

2、 寄存器堆：是 CPU 中非常重要的构成部分，由触发器构成，触发器再构成寄存器，寄存器再构成寄存器堆。可以记忆多个字，然后每个字的字长就是寄存器的位宽。

二、硬件逻辑图

下图为 16 个 4 位寄存器组成的寄存器堆的 logisim 原理图



三、模块建模

（该部分要求对实验中建模的所有模块进行功能描述，并列出各模块建模的 verilog 代码）

实验一：

1.功能描述：本模块实现了一个 RAM，端口有时钟、读写信号、使能信号、

读写的地址，32 位写入的数据，32 位输出。，并且读的操作不受时钟控制。首先定义一个 32*32 的寄存器数组，表示有 32 个 32bit 的存储单元。处理输入：5 位的地址正好可以一一对应 32 个存储单元，用 always 语句把输入限制在时钟下，只有 ena 高电平（ram 有效）、wena 高电平（写入）才可以写入。然后处理输出，输出首先判断 ena，如果低电平则输出不定态，如果 ena 高电平且 wena 低电平（读）那么输出就是对应的存储单元，其他情况不考虑。

2.代码：

```
module ram (
  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
input [31:0] data_in, //存储器写入的数据，在 clk 上升沿时被写入
output reg [31:0] data_out //存储器读出的数据
);
reg [31:0] memory[0:31];
always @(posedge clk)
begin
if(wena&&ena)
begin
memory[addr]=data_in;
end
end
always @(*)
begin
if(!wena&&ena)
begin
data_out=memory[addr];
end
else if(!ena)
data_out=32'bz;
end
endmodule
```

实验二：

1.功能描述：本模块仍然是实现一个 RAM，其功能和上面的几乎一样，唯一的区别是本模块的输入输出是共用一条线的，对应端口定义上是 inout，所以在行为及语言描述的时候，要定义一个 reg 的中间变量，首先定义一个 32*32 的寄存器数组，表示有 32 个 32bit 的存储单元。处理输入：5 位的地址正好可以一一对应 32 个存储单元，用 always 语句把输入限制在时钟下，只有 ena 高电平（ram 有效）、wena 高电平（写入）才可以写入。然后处理输出，输出首先判断 ena，

如果低电平则输出不定态，如果 ena 高电平且 wena 低电平（读）那么输出就是对应的存储单元。最后用 assign 语句对 data 赋值（只可以是三目运算符），如果 wena 为低电平那么就把输出赋值给 data，如果 wena 是高电平那么 data 就是自己本身。

2.代码：

```
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
);
reg [31:0] memory[0:31];
reg  [31:0]  data_reg;//为了可以使用行为级描述，存放数据的寄存器，最后再用 assign 赋值给 data

always @(posedge clk)
begin
if(wena&&ena)
memory[addr]=data;
end

always @(*)
begin
if(!wena&&ena)
data_reg=memory[addr];
else if(!ena)
data_reg=32'bz;
end

assign data = (!wena ) ? data_reg : 32'bz;//如果不是读取操作，那么就是未定态
endmodule
```

实验三：

1.功能描述：本模块实现的是一个寄存器堆，32 个 32 位的寄存器，端口有时钟、复位信号、读写信号、写的地址、两个读的地址、32 位写的数据、两个 32

位输出。本模块首先实例化一个译码器，负责译码五位写入地址，根据译码器的原理，采用 case 语句对译码器 32 位的输出赋值。首先如果 rst 为高电平，采用 always 语句把 32 个寄存器清零。当 we 为高电平（写），译码器译码，同时当时钟下降沿来临的时候，写入 wdata 的内容，此时输出是不定态。当 we 为低电平时（读），允许输出，根据两个选择器，选择对应的寄存器输出。

2.代码：

```
module decoder32(
input [4:0] iData,
output reg [31:0] oData,
input ena
);
always@(*)
begin
if(ena)
begin
case(iData)
5'd0:    oData=32'b11111111111111111111111111111110;
5'd1:    oData=32'b11111111111111111111111111111101;
5'd2:    oData=32'b11111111111111111111111111111011;
5'd3:    oData=32'b11111111111111111111111111110111;
5'd4:    oData=32'b11111111111111111111111111101111;
5'd5:    oData=32'b11111111111111111111111111011111;
5'd6:    oData=32'b11111111111111111111111110111111;
5'd7:    oData=32'b11111111111111111111111101111111;
5'd8:    oData=32'b11111111111111111111111011111111;
5'd9:    oData=32'b11111111111111111111110111111111;
5'd10: oData=32'b11111111111111111111101111111111;
5'd11: oData=32'b11111111111111111111011111111111;
5'd12: oData=32'b11111111111111111110111111111111;
5'd13: oData=32'b11111111111111111101111111111111;
5'd14: oData=32'b11111111111111111011111111111111;
5'd15: oData=32'b11111111111111110111111111111111;
5'd16: oData=32'b11111111111111101111111111111111;
5'd17: oData=32'b11111111111111011111111111111111;
5'd18: oData=32'b11111111111110111111111111111111;
5'd19: oData=32'b11111111111101111111111111111111;
5'd20: oData=32'b11111111111011111111111111111111;
5'd21: oData=32'b11111111110111111111111111111111;
5'd22: oData=32'b11111111101111111111111111111111;
5'd23: oData=32'b11111111011111111111111111111111;
5'd24: oData=32'b11111110111111111111111111111111;
5'd25: oData=32'b11111101111111111111111111111111;
5'd26: oData=32'b11111011111111111111111111111111;
```

```
5'd27: oData=32'b11110111111111111111111111111111;
5'd28: oData=32'b11101111111111111111111111111111;
5'd29: oData=32'b11011111111111111111111111111111;
5'd30: oData=32'b10111111111111111111111111111111;
5'd31: oData=32'b01111111111111111111111111111111;
endcase
end
end
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,
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]addr,
input ena,
```

```
output reg [31:0] oZ
);
always @(*)
begin
if(ena)
begin
case(addr)
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;
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;
default: oZ=32'bz;
endcase
end
else
begin
```

```
oZ=32'bz;
end
end
endmodule

module Regfiles(
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，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
);
wire [31:0]temp1;
reg [31:0]stk[0:31];
reg i;
decoder32 uut1(waddr,temp1,we);
selector32_1
uut2(stk[0],stk[1],stk[2],stk[3],stk[4],stk[5],stk[6],stk[7],stk[8],stk[9],stk[10],stk[11],stk[12],stk[13],stk[14],stk[15],stk[16],stk[17],stk[18],stk[19],stk[20],stk[21],stk[22],stk[23],stk[24],stk[25],stk[26],stk[27],stk[28],stk[29],stk[30],stk[31],raddr1,~we,rdata1);
selector32_1
uut3(stk[0],stk[1],stk[2],stk[3],stk[4],stk[5],stk[6],stk[7],stk[8],stk[9],stk[10],stk[11],stk[12],stk[13],stk[14],stk[15],stk[16],stk[17],stk[18],stk[19],stk[20],stk[21],stk[22],stk[23],stk[24],stk[25],stk[26],stk[27],stk[28],stk[29],stk[30],stk[31],raddr2,~we,rdata2);
always@(negedge clk or posedge rst)
begin
if(rst)
begin
stk[0]=32'b0;
stk[1]=32'b0;
stk[2]=32'b0;
stk[3]=32'b0;
stk[4]=32'b0;
stk[5]=32'b0;
stk[6]=32'b0;
stk[7]=32'b0;
stk[8]=32'b0;
stk[9]=32'b0;
stk[10]=32'b0;
```

```
stk[11]=32'b0;
stk[12]=32'b0;
stk[13]=32'b0;
stk[14]=32'b0;
stk[15]=32'b0;
stk[16]=32'b0;
stk[17]=32'b0;
stk[18]=32'b0;
stk[19]=32'b0;
stk[20]=32'b0;
stk[21]=32'b0;
stk[22]=32'b0;
stk[23]=32'b0;
stk[24]=32'b0;
stk[25]=32'b0;
stk[26]=32'b0;
stk[27]=32'b0;
stk[28]=32'b0;
stk[29]=32'b0;
stk[30]=32'b0;
stk[31]=32'b0;
end
if(we&&!rst)
begin
case(temp1)
32'b11111111111111111111111111111110:stk[0]=wdata;
32'b11111111111111111111111111111101:stk[1]=wdata;
32'b11111111111111111111111111111011:stk[2]=wdata;
32'b11111111111111111111111111110111:stk[3]=wdata;
32'b11111111111111111111111111101111:stk[4]=wdata;
32'b11111111111111111111111111011111:stk[5]=wdata;
32'b11111111111111111111111110111111:stk[6]=wdata;
32'b11111111111111111111111101111111:stk[7]=wdata;
32'b11111111111111111111111011111111:stk[8]=wdata;
32'b11111111111111111111110111111111:stk[9]=wdata;
32'b11111111111111111111101111111111:stk[10]=wdata;
32'b11111111111111111111011111111111:stk[11]=wdata;
32'b11111111111111111110111111111111:stk[12]=wdata;
32'b11111111111111111101111111111111:stk[13]=wdata;
32'b11111111111111111011111111111111:stk[14]=wdata;
32'b11111111111111110111111111111111:stk[15]=wdata;
32'b11111111111111101111111111111111:stk[16]=wdata;
32'b11111111111111011111111111111111:stk[17]=wdata;
32'b11111111111110111111111111111111:stk[18]=wdata;
```

```
32'b11111111111101111111111111111111:stk[19]=wdata;
32'b11111111111011111111111111111111:stk[20]=wdata;
32'b11111111110111111111111111111111:stk[21]=wdata;
32'b11111111101111111111111111111111:stk[22]=wdata;
32'b11111111011111111111111111111111:stk[23]=wdata;
32'b11111110111111111111111111111111:stk[24]=wdata;
32'b11111101111111111111111111111111:stk[25]=wdata;
32'b11111011111111111111111111111111:stk[26]=wdata;
32'b11110111111111111111111111111111:stk[27]=wdata;
32'b11101111111111111111111111111111:stk[28]=wdata;
32'b11011111111111111111111111111111:stk[29]=wdata;
32'b10111111111111111111111111111111:stk[30]=wdata;
32'b01111111111111111111111111111111:stk[31]=wdata;
endcase
end
end
endmodule
```

四、测试模块建模

（要求列写各建模模块的 test bench 模块代码）

实验一：

```
`timescale 1ns / 1ps
module ram_tb;
    reg clk;
    reg ena;
    reg wena;
    reg [4:0] addr;
    reg [31:0] data_in;
    wire [31:0] data_out;
    reg [31:0] memory[0:31];

    // 实例化 RAM 模块
    ram uut (clk,ena,wena,addr,data_in,data_out);
initial
begin
clk = 0;
forever #5 clk = ~clk; // 每 5 个时间单位反转一次时钟,周期为 100
end

initial
begin
```

```
$readmemh("ram1",memory);
ena = 0;
wena = 0;
addr = 0;
data_in = 0;

        // 等待一段时间
        #10;

        // 启用存储器
        ena = 1;

        // 写入数据
        wena = 1; // 写使能
        addr = 5'b00000; // 地址 0
        data_in = 32'hA5A5A5A5; // 写入数据
        #20; // 等待时钟上升沿

        wena = 1; // 写使能
        addr = 5'b00001; // 地址 1
        data_in = 32'h5A5A5A55; // 写入数据
        #20; // 等待时钟上升沿
        wena = 0; // 读使能
        addr = 5'b00000; // 地址 0
        #20; // 等待时钟上升沿
        wena = 0; // 读使能
        addr = 5'b00001; // 地址 1
        #20; // 等待时钟上升沿
        ena = 0; // 禁用存储器
        addr = 5'b00000; // 地址 0
end
endmodule
```

实验二：

```
`timescale 1ns / 1ps
module ram2_tb;
reg clk;
reg ena;
reg wena;
reg [4:0] addr;
wire [31:0] data;
reg [31:0] data_contorl;
reg [31:0] MEM[0:31];
```

```
assign data=data_contorl;

initial
begin
clk=0;
#40;
forever #20 clk=~clk;
end

initial
begin
$readmemh("ram2", MEM);
ena=0;
#80 ena=1;wena=1;addr=3;
data_contorl=32'h5f;
#30 wena=0;ena=0;
#30 ena=1;wena=1;addr=0;
data_contorl=32'hff;
#20 addr=3;
#20 addr=0;
end
ram2 uut(clk,ena,wena,addr,data);
endmodule
```

实验三：

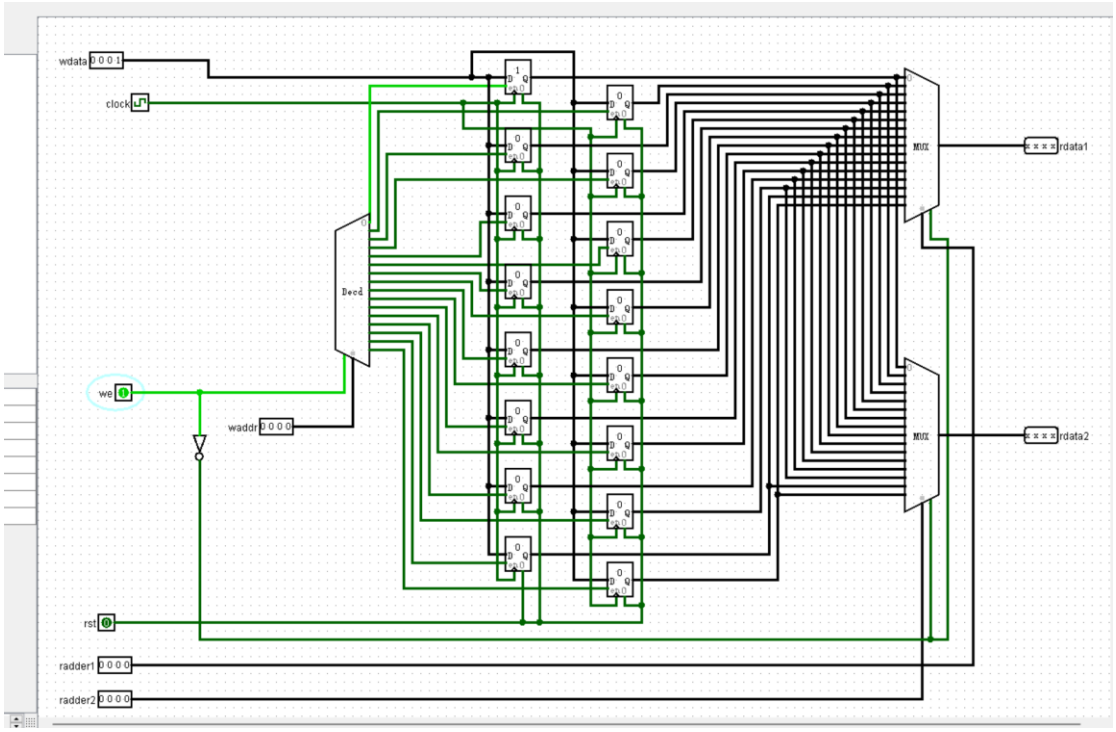
```
`timescale 1ns / 1ns
module Regfiles_tb;
reg clk;
reg rst;
reg we;
reg [4:0] raddr1;
reg [4:0] raddr2;
reg [4:0] waddr;
reg [31:0] wdata;
wire [31:0] rdata1;
wire [31:0] rdata2;
Regfiles uut8 (clk,rst,we,raddr1,raddr2,waddr,wdata,rdata1,rdata2);
initial
begin
clk = 0;
forever #5 clk = ~clk;
end
```

```
initial
begin
rst = 1;
we = 0;
raddr1 = 0;
raddr2 = 0;
waddr = 0;
wdata = 0;
#20
rst = 0;
#50
we = 1;
#20
waddr = 5'd1;
wdata = 32'h5A5A5A5A;
#20
we=0;
raddr1 = 5'd0;
raddr2 = 5'd1;
end
endmodule
```

五、实验结果

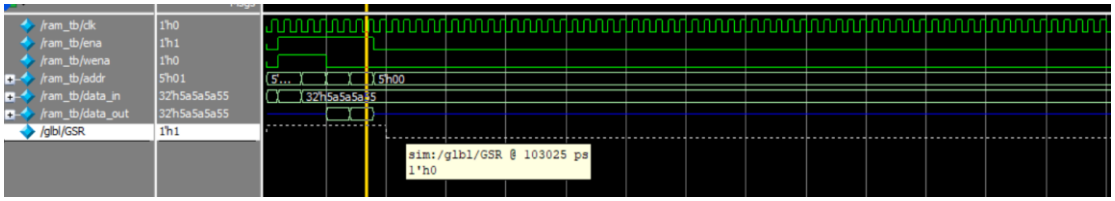
（该部分可截图说明，要求 logisim 逻辑验证图、modelsim 仿真波形图、以及下板后的实验结果贴图（实验步骤中没有下板要求的实验，不需要下板贴图））

Logisim 图：



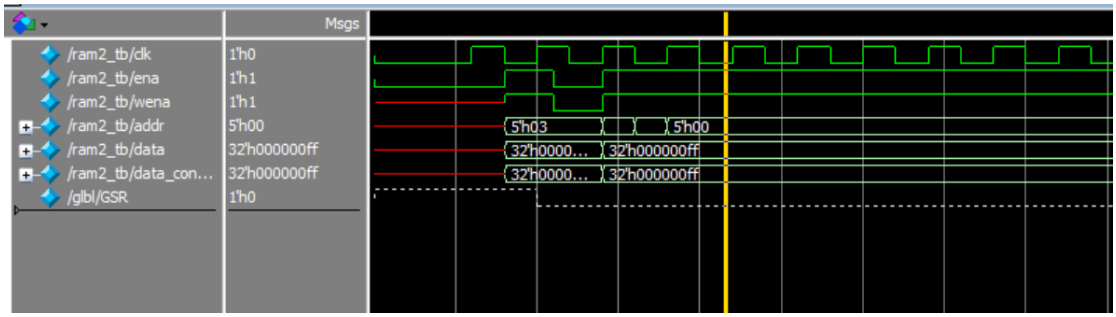
实验一：

Modelsim 图



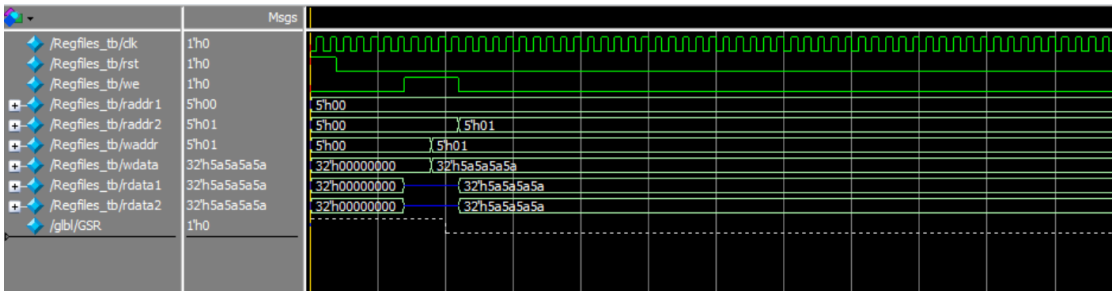
实验二：

Modelsim 图



实验三：

Modelsim 图


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
/glbl/GSR
1'h0
1'h0
1'h0
5'h00
5'h01
5'h00
5'h01
32'h00000000
32'h5a5a5a5a
32'h00000000
32'h5a5a5a5a
32'h5a5a5a5a
1'h0
5'h00
5'h01
5'h00
5'h01
32'h00000000
32'h5a5a5a5a
32'h00000000
32'h5a5a5a5a