# 同济大学计算机系

# 数字逻辑课程实验报告


学　　号 **2151140**

姓　　名 王谦

专　　业 信息安全

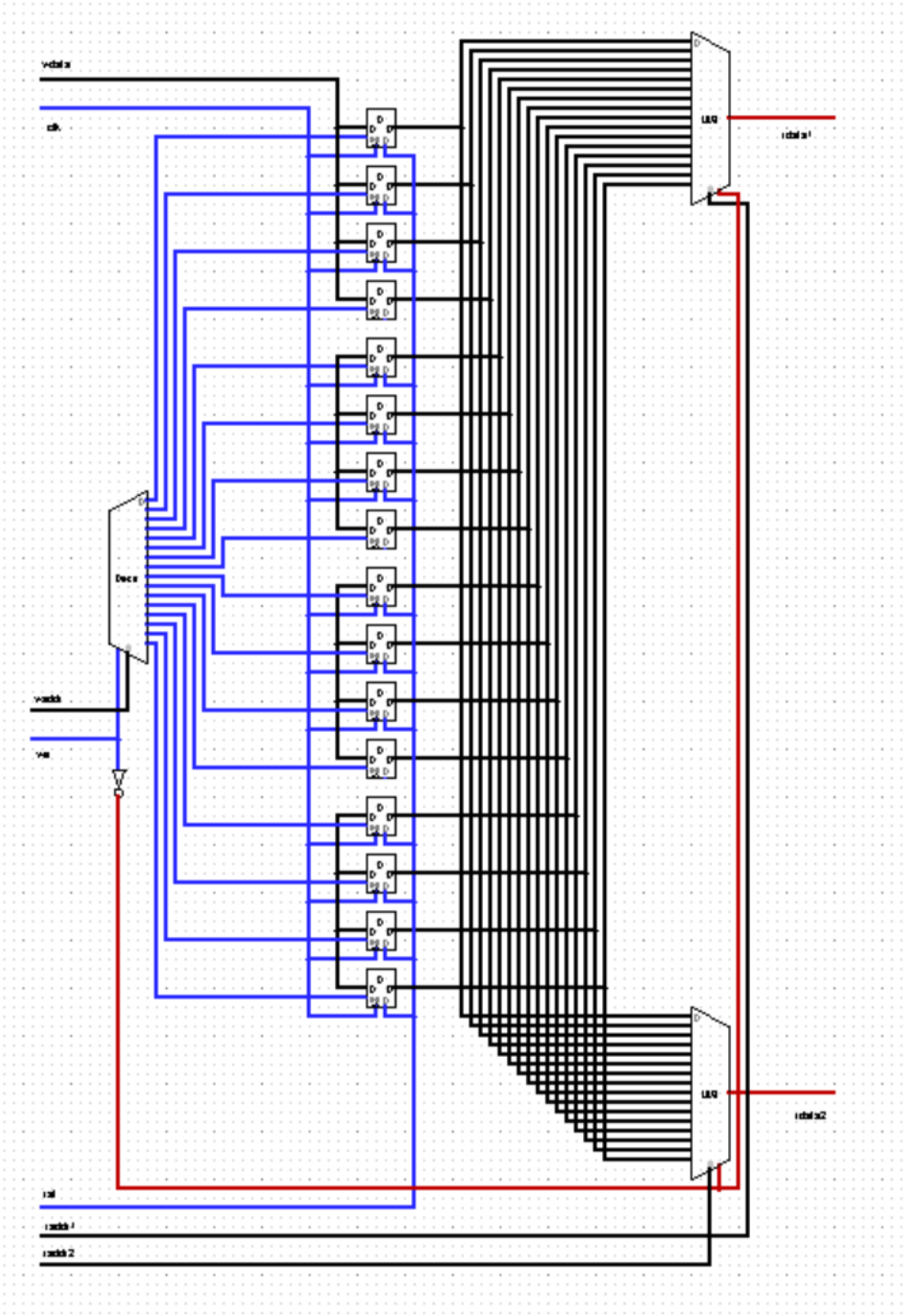
授课老师 郭玉臣

# 一、实验内容

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。

# 二、硬件逻辑图

（实验步骤中要求用 logisim 画图的实验，在该部分给出 logisim 原理图，否则该部分在实验报告中不用写）



# 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）

1.RAM

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件，可存放大量的数据。如图 6.25 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，以便组成双向 I/O 数据线。

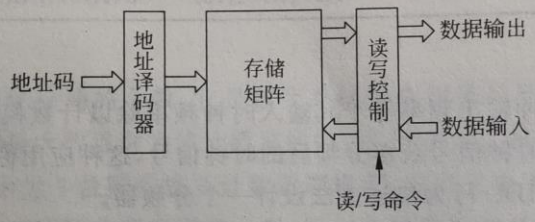



图 6.25　RAM 的逻辑结构图

RAM 有以下三组信号线。

（1）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。

（2）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。

（3）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。

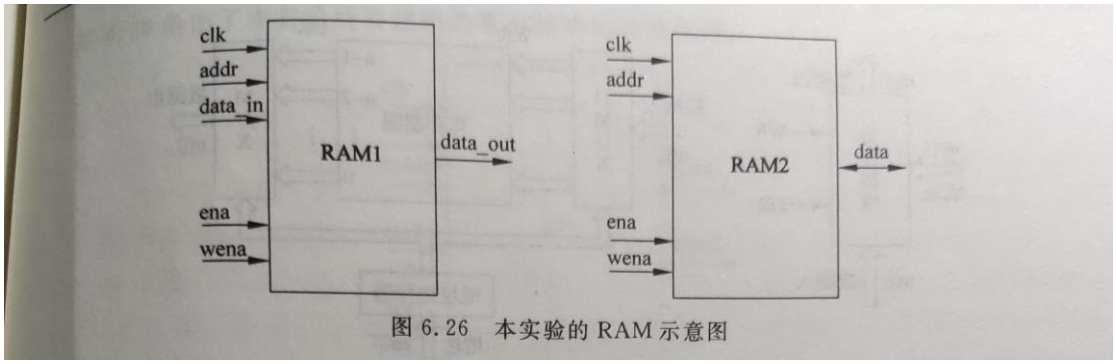



图 6.26　本实验的 RAM 示意图

```
module ram(clk,ena,wena,addr,data_in,data_out);
input clk, ena, wena;
input [4:0] addr;
input wire [31:0] data_in;
output reg [31:0] data_out;

reg [31:0] bram[31:0];
integer i;
reg [31:0] data;

    initial begin
    for(i=0;i<=31;i=i+1)
        bram[i] <= 32'b0;
    end
```

```
        always @(posedge clk)
        begin
          if (~ena)
             begin
                for(i=0;i<=31;i=i+1)
                data_out <= 32'bz;
             end
          else
             begin
             if (wena) begin
                     bram[addr] <= data_in;
             end
             else begin
                     data_out <= bram[addr];
             end
          end
        end
endmodule
```

2.RAM2

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

reg [31:0] bram[31:0];
reg [31:0] Data;

integer i;
        initial begin
        for(i=0;i<=31;i=i+1)
              bram[i] <= 32'b0;
        end

        always @(posedge clk)
        begin
          if (~ena)
             begin
                for(i=0;i<=31;i=i+1)
                Data <= 32'bz;
             end
          else
             begin
```

```
        if (wena) begin
            bram[addr] <= data;
        end
        else begin
            Data <= bram[addr];
        end
    end
end

    //assign data = (~wena) ? bram[addr] : 32'bz; //三态门实现
assign data =(~wena)? Data : 32'bz;
endmodule
```

3.寄存器堆

2）寄存器堆(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。当需要记忆多个字时，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。

图 6.27 所示为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。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。

图 6.28 给出了由 4 个 4 位寄存器组成的具有两个数据输出端口的寄存器堆原理图，它可以同时从寄存器堆中取出两个数据，和加法器一起构成一个简单的运算通路。其主要由 1 个 2-4 译码器(ENB 为使能端，$S_1$、$S_2$ 为两位编码输入端，$D_1$～$D_4$ 为译码输出端)、4 个 4 位寄存器(ENB 为使能端，A～D 为数据输入端，$Q_1$～$Q_4$ 为数据输出端)和 2 个 4 位 4 选 1 数据选择器(ENB 为使能端，$S_1$～$S_4$ 为 4 路数据输入端，$C_1$、$C_2$ 为选择控制端，D 为数据输

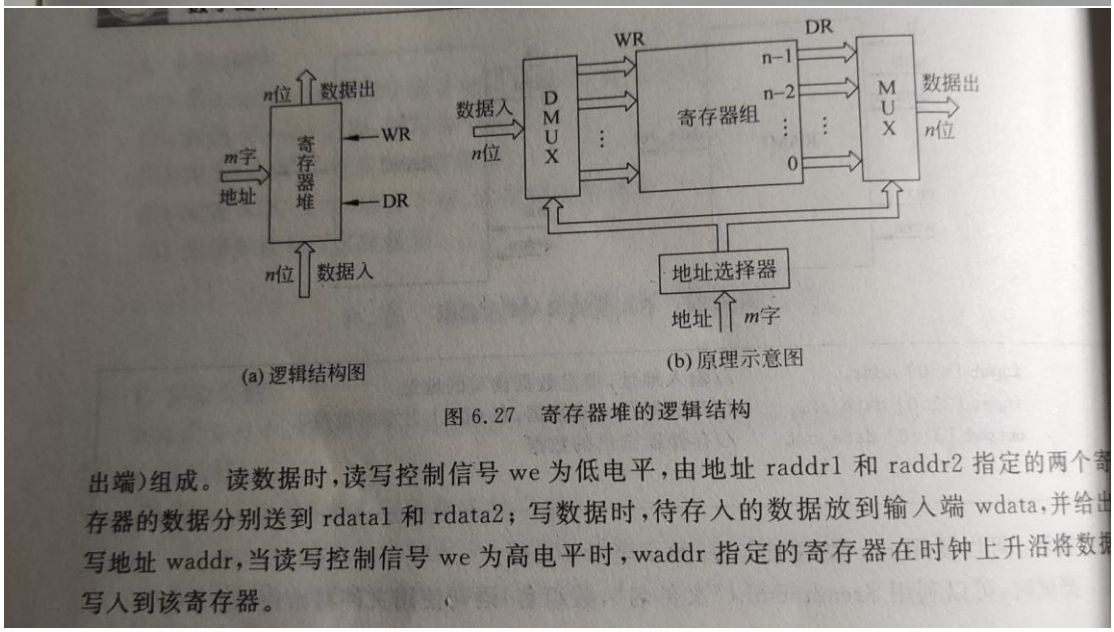
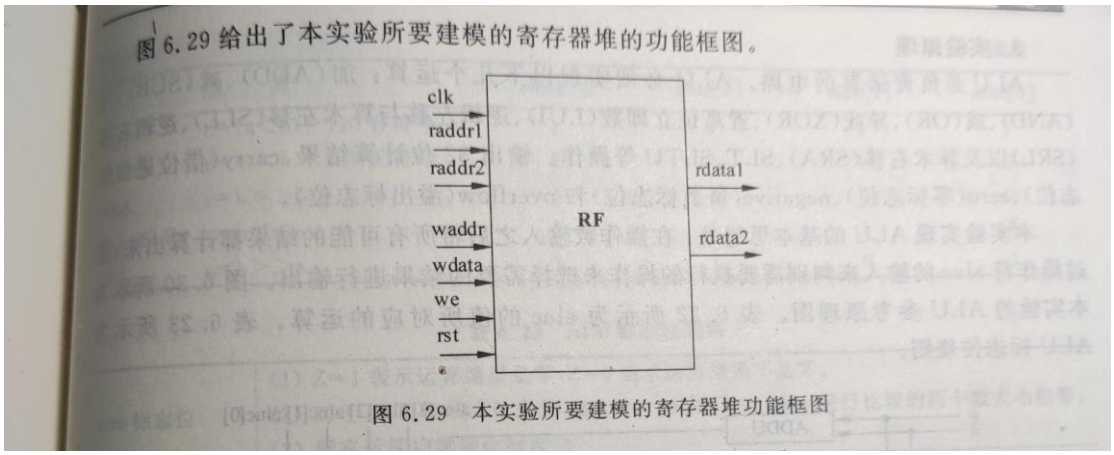



图 6.27　寄存器堆的逻辑结构

出端)组成。读数据时，读写控制信号 we 为低电平，由地址 raddr1 和 raddr2 指定的两个寄存器的数据分别送到 rdata1 和 rdata2；写数据时，待存入的数据放到输入端 wdata，并给出写地址 waddr，当读写控制信号 we 为高电平时，waddr 指定的寄存器在时钟上升沿将数据写入到该寄存器。




```
//寄存器堆************************************************************
module Regfiles(clk,rst,we,raddr1,raddr2,waddr,wdata,rdata1,rdata2);
input clk,rst,we;
input[4:0]raddr1,raddr2,waddr,wdata;
output[31:0]rdata1,rdata2;

wire [31:0]Q[31:0];
wire [31:0]code;
decoder deco(waddr,we,code);
pcreg r0(clk,rst,~code[0],wdata,Q[0]);
pcreg r1(clk,rst,~code[1],wdata,Q[1]);
pcreg r2(clk,rst,~code[2],wdata,Q[2]);
pcreg r3(clk,rst,~code[3],wdata,Q[3]);
pcreg r4(clk,rst,~code[4],wdata,Q[4]);
pcreg r5(clk,rst,~code[5],wdata,Q[5]);
pcreg r6(clk,rst,~code[6],wdata,Q[6]);
pcreg r7(clk,rst,~code[7],wdata,Q[7]);
pcreg r8(clk,rst,~code[8],wdata,Q[8]);
pcreg r9(clk,rst,~code[9],wdata,Q[9]);
pcreg r10(clk,rst,~code[10],wdata,Q[10]);
pcreg r11(clk,rst,~code[11],wdata,Q[11]);
pcreg r12(clk,rst,~code[12],wdata,Q[12]);
pcreg r13(clk,rst,~code[13],wdata,Q[13]);
pcreg r14(clk,rst,~code[14],wdata,Q[14]);
pcreg r15(clk,rst,~code[15],wdata,Q[15]);
pcreg r16(clk,rst,~code[16],wdata,Q[16]);
pcreg r17(clk,rst,~code[17],wdata,Q[17]);
pcreg r18(clk,rst,~code[18],wdata,Q[18]);
pcreg r19(clk,rst,~code[19],wdata,Q[19]);
pcreg r20(clk,rst,~code[20],wdata,Q[20]);
pcreg r21(clk,rst,~code[21],wdata,Q[21]);
pcreg r22(clk,rst,~code[22],wdata,Q[22]);
pcreg r23(clk,rst,~code[23],wdata,Q[23]);
```

```
pcreg r24(clk,rst,~code[24],wdata,Q[24]);
pcreg r25(clk,rst,~code[25],wdata,Q[25]);
pcreg r26(clk,rst,~code[26],wdata,Q[26]);
pcreg r27(clk,rst,~code[27],wdata,Q[27]);
pcreg r28(clk,rst,~code[28],wdata,Q[28]);
pcreg r29(clk,rst,~code[29],wdata,Q[29]);
pcreg r30(clk,rst,~code[30],wdata,Q[30]);
pcreg r31(clk,rst,~code[31],wdata,Q[31]);
selector32_1
s1(Q[0],Q[1],Q[2],Q[3],Q[4],Q[5],Q[6],Q[7],Q[8],Q[9],Q[10],Q[11],Q[12],Q[13],Q[14],Q[15],Q[
16],Q[17],Q[18],Q[19],Q[20],Q[21],Q[22],Q[23],Q[24],Q[25],Q[26],Q[27],Q[28],Q[29],Q[30],Q
[31],raddr1,~we,rdata1);
selector32_1
s2(Q[0],Q[1],Q[2],Q[3],Q[4],Q[5],Q[6],Q[7],Q[8],Q[9],Q[10],Q[11],Q[12],Q[13],Q[14],Q[15],Q[
16],Q[17],Q[18],Q[19],Q[20],Q[21],Q[22],Q[23],Q[24],Q[25],Q[26],Q[27],Q[28],Q[29],Q[30],Q
[31],raddr2,~we,rdata2);
endmodule
// ASynchronous_D_FF*****************************************
module ASynchronous_D_FF(
input CLK,
input RST_n,
input ena,
input D,
output reg Q1
    );
    always@(posedge CLK or RST_n or ena)
    if(RST_n==1'b1)
    Q1=1'b0;
    else
    if(ena==1'b1)
    Q1=D;
endmodule
// decoder*****************************************************
module decoder(iData,iEna,oData);
    input [4:0] iData; //三位输入  D2,D1,D0
    input iEna; //使能信号  G1,G2
    output reg [31:0] oData;//32 位译码输出?7 ~?0 ,低电平有效
always@(iData or iEna)
begin
    if(iEna==2'b1)
        begin
          oData=32'hFFFFFFFF;
        oData[iData[0]+iData[1]*2+iData[2]*4+iData[3]*8+iData[4]*16]=0;
        end
```

```
        else
                oData=32'hFFFFFFFF;
end
endmodule
// selector32_1************************************************
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] iS,
  input ena,
output [31:0] oZ);
        wire [31:0]iC_all[31:0];
assign iC_all[0] = iC0;
assign iC_all[1] = iC1;
```

```
assign iC_all[2] = iC2;
assign iC_all[3] = iC3;
assign iC_all[4] = iC4;
assign iC_all[5] = iC5;
assign iC_all[6] = iC6;
assign iC_all[7] = iC7;
assign iC_all[8] = iC8;
assign iC_all[9] = iC9;
assign iC_all[10] = iC10;
assign iC_all[11] = iC11;
assign iC_all[12] = iC12;
assign iC_all[13] = iC13;
assign iC_all[14] = iC14;
assign iC_all[15] = iC15;
assign iC_all[16] = iC16;
assign iC_all[17] = iC17;
assign iC_all[18] = iC18;
assign iC_all[19] = iC19;
assign iC_all[20] = iC20;
assign iC_all[21] = iC21;
assign iC_all[22] = iC22;
assign iC_all[23] = iC23;
assign iC_all[24] = iC24;
assign iC_all[25] = iC25;
assign iC_all[26] = iC26;
assign iC_all[27] = iC27;
assign iC_all[28] = iC28;
assign iC_all[29] = iC29;
assign iC_all[30] = iC30;
assign iC_all[31] = iC31;
	assign oZ=((ena==1'b1)?iC_all[iS]:32'bz);
endmodule
// pcreg************************************************************
module pcreg(
input clk,
input rst,
input ena,
input [31:0] data_in,
output [31:0] data_out
	);
	ASynchronous_D_FF T0(clk, rst, ena, data_in[0], data_out[0]);
	ASynchronous_D_FF T1(clk, rst, ena, data_in[1], data_out[1]);
	ASynchronous_D_FF T2(clk, rst, ena, data_in[2], data_out[2]);
	ASynchronous_D_FF T3(clk, rst, ena, data_in[3], data_out[3]);
```

```
    ASynchronous_D_FF T4(clk, rst, ena, data_in[4], data_out[4]);
    ASynchronous_D_FF T5(clk, rst, ena, data_in[5], data_out[5]);
    ASynchronous_D_FF T6(clk, rst, ena, data_in[6], data_out[6]);
    ASynchronous_D_FF T7(clk, rst, ena, data_in[7], data_out[7]);
    ASynchronous_D_FF T8(clk, rst, ena, data_in[8], data_out[8]);
    ASynchronous_D_FF T9(clk, rst, ena, data_in[9], data_out[9]);
    ASynchronous_D_FF T10(clk, rst, ena, data_in[10], data_out[10]);
    ASynchronous_D_FF T11(clk, rst, ena, data_in[11], data_out[11]);
    ASynchronous_D_FF T12(clk, rst, ena, data_in[12], data_out[12]);
    ASynchronous_D_FF T13(clk, rst, ena, data_in[13], data_out[13]);
    ASynchronous_D_FF T14(clk, rst, ena, data_in[14], data_out[14]);
    ASynchronous_D_FF T15(clk, rst, ena, data_in[15], data_out[15]);
    ASynchronous_D_FF T16(clk, rst, ena, data_in[16], data_out[16]);
    ASynchronous_D_FF T17(clk, rst, ena, data_in[17], data_out[17]);
    ASynchronous_D_FF T18(clk, rst, ena, data_in[18], data_out[18]);
    ASynchronous_D_FF T19(clk, rst, ena, data_in[19], data_out[19]);
    ASynchronous_D_FF T20(clk, rst, ena, data_in[20], data_out[20]);
    ASynchronous_D_FF T21(clk, rst, ena, data_in[21], data_out[21]);
    ASynchronous_D_FF T22(clk, rst, ena, data_in[22], data_out[22]);
    ASynchronous_D_FF T23(clk, rst, ena, data_in[23], data_out[23]);
    ASynchronous_D_FF T24(clk, rst, ena, data_in[24], data_out[24]);
    ASynchronous_D_FF T25(clk, rst, ena, data_in[25], data_out[25]);
    ASynchronous_D_FF T26(clk, rst, ena, data_in[26], data_out[26]);
    ASynchronous_D_FF T27(clk, rst, ena, data_in[27], data_out[27]);
    ASynchronous_D_FF T28(clk, rst, ena, data_in[28], data_out[28]);
    ASynchronous_D_FF T29(clk, rst, ena, data_in[29], data_out[29]);
    ASynchronous_D_FF T30(clk, rst, ena, data_in[30], data_out[30]);
    ASynchronous_D_FF T31(clk, rst, ena, data_in[31], data_out[31]);
endmodule
```

# 四、测试模块建模

（要求列写各建模模块的 test bench 模块代码）

1.

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

integer i;
```

```
initial clk = 0;
always #5 clk = ~clk;

initial
begin
        data_in = 32'b0;
        ena = 1;
        wena = 1;
        //********写
        for (i = 0; i < 32; i = i + 1)
        begin
       @(posedge clk)
        begin
         addr = i;
         data_in = data_in    + 1;
         end
        end
        //********读
        ena = 1;
        wena = 0;
        for (i = 0; i < 32; i = i + 1)
        begin
       @(posedge clk)
       addr = i;
        end
        //********写
        ena = 0;
        wena = 1;
        for (i = 0; i < 32; i = i + 1)
        begin
        @(posedge clk)
        begin
        addr = i;
        data_in = data_in    + 1;
        end
        end
        //********读
        ena = 0;
        wena = 0;
        for (i = 0; i < 32; i = i + 1)
        begin
        @(posedge clk)
        addr = i;
```

```
        end
end

ram
ram_init(
.clk(clk),
.ena(ena),
.wena(wena),
.addr(addr),
.data_in(data_in),
.data_out(data_out)
);
endmodule
```

2.

```
`timescale 1ns / 1ns
module ram2_tb;
reg clk, ena, wena;
reg [4:0] addr;
wire [31:0] data_wire;

reg [31:0] data_reg;

integer i;

initial clk = 0;
always #5 clk = ~clk;

initial
begin
data_reg = 32'b0;
ena = 1;
wena = 1;
//写
for (i = 0; i < 32; i = i + 1)
begin
@(posedge clk)
begin
addr = i;
data_reg = data_reg    + 1;
end
end

//读
```

```
ena = 1;
wena = 0;
for (i = 0; i < 32; i = i + 1)
begin
@(posedge clk)
addr = i;
end

ena = 0;
wena = 1;
//写
for (i = 0; i < 32; i = i + 1)
begin
@(posedge clk)
begin
addr = i;
data_reg = data_reg    + 1;
end
end

//读
ena = 0;
wena = 0;
for (i = 0; i < 32; i = i + 1)
begin
@(posedge clk)
addr = i;
end
end

assign data_wire = wena ? data_reg:32'bz;

ram2
ram2_init(
.clk(clk),
.ena(ena),
.wena(wena),
.addr(addr),
.data(data_wire)
);

endmodule
```

3.

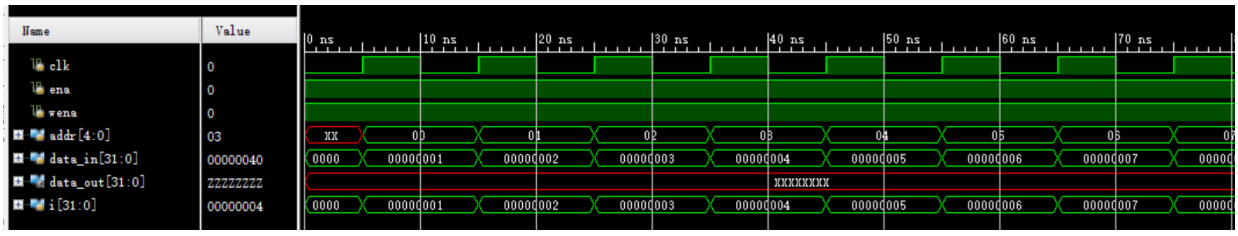
```
`timescale 1ns / 1ns
module Regfiles_tb();
reg clk;
reg rst;
reg we;
reg [4:0]raddr1;
reg [4:0]raddr2;
reg [4:0]waddr;
reg [31:0]wdata;
wire [31:0]rdata1;
wire [31:0]rdata2;
always
begin
#2 clk=~clk;
end
initial
begin
clk=0;we=1;rst=0;
raddr1=5'b00000;
raddr2=5'b11111;
waddr=5'b00000;
wdata=32'h180174FA;//向 00000 写入 180174FA
#4 wdata=32'hFA180174; waddr=5'b11111;//向 11111 写入 FA180174
#4 we=0; //把刚刚写的两位读出来
#10 rst=1;//复位测试
#20
$stop;
end
Regfiles rf(clk,rst,we,raddr1,raddr2,waddr,wdata,rdata1,rdata2);
endmodule
```

# 五、实验结果

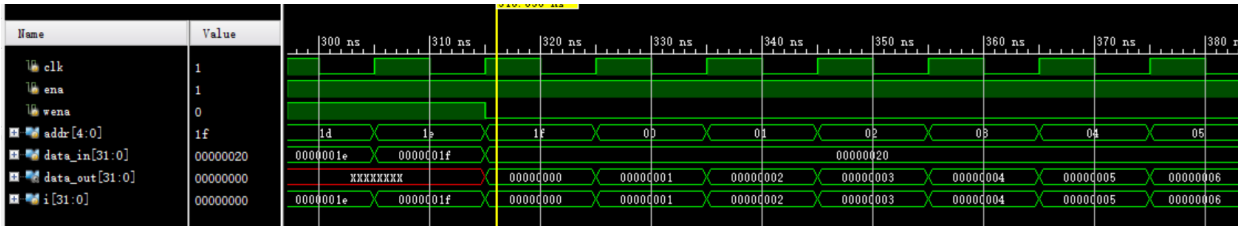
（该部分可截图说明，要求 logisim 逻辑验证图、modelsim 仿真波形图、以及下板后的实验结果贴图（实验步骤中没有下板要求的实验，不需要下板贴图））

1.

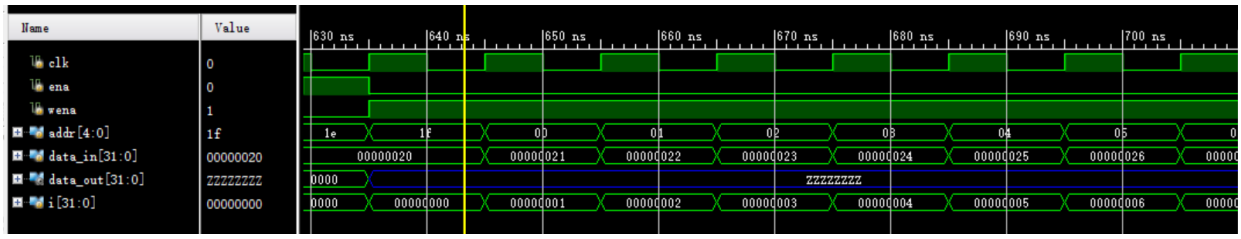
写时：

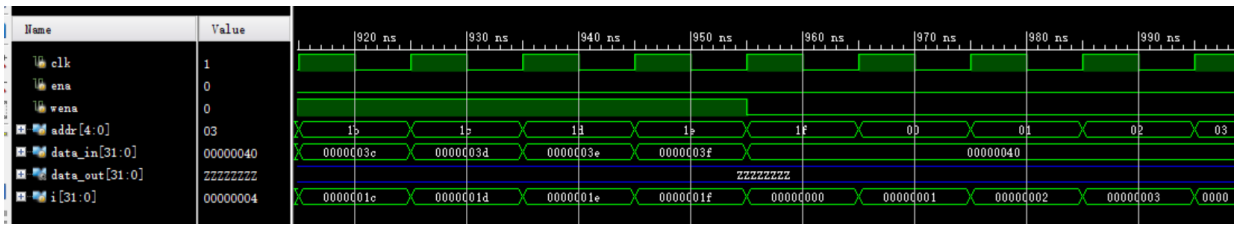



读时：




ena 为 0 时，读写输出均为 z：

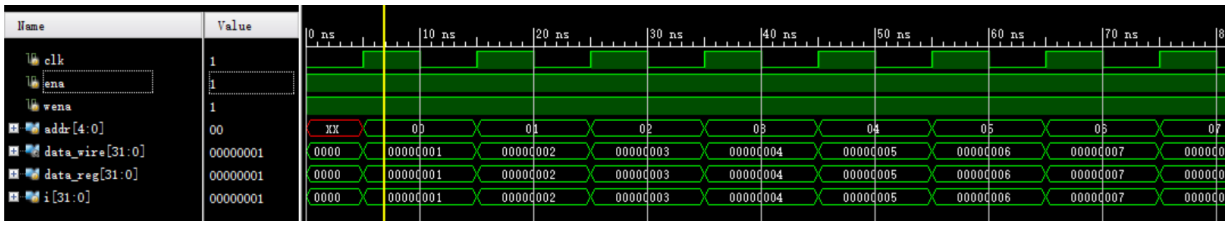


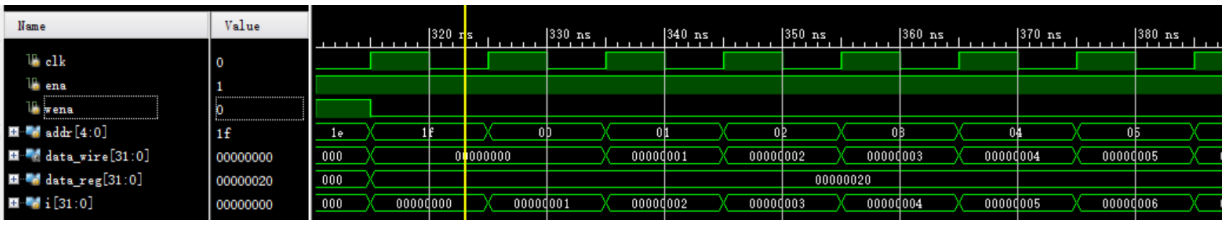

2.

data_wire 和 data_reg 物理上为同一接口

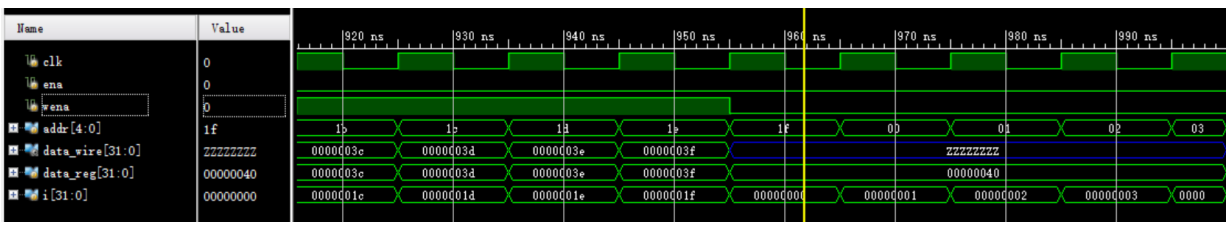
写时：




读时为 wire：

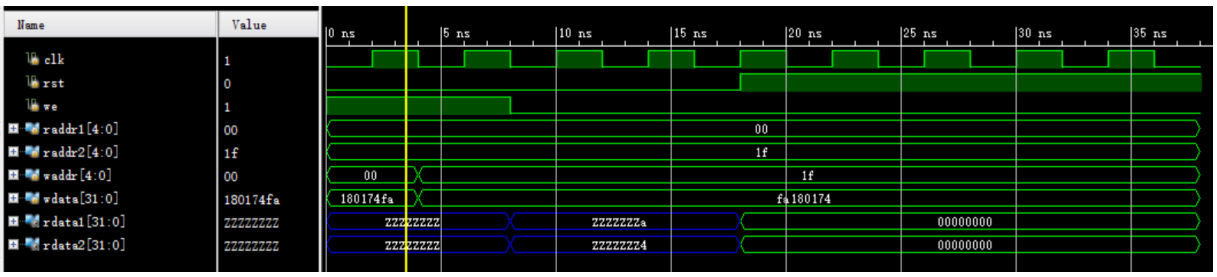



ena 为 0 时，读出为 z：

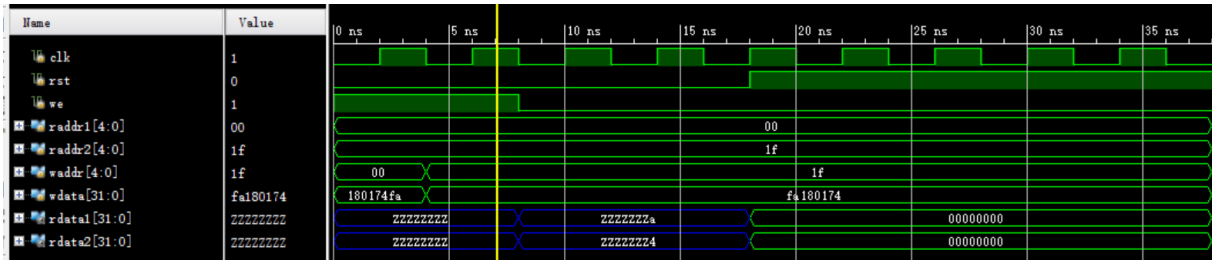


3.

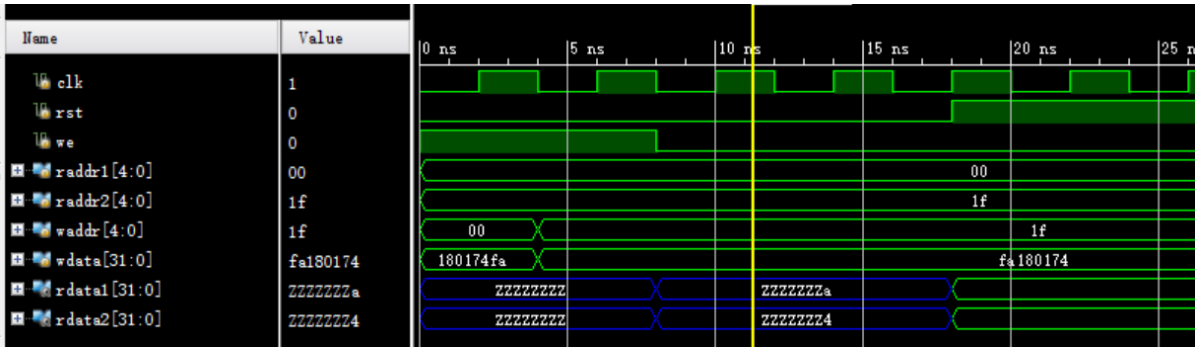
向 00000 写入 180174FA：




向 11111 写入 FA180174：




读出刚刚两个地址写入的数 A 和 4：




置 0：