同济大学计算机系

数字逻辑课程实验报告

学　　号 **2253156**

姓　　名 闫浩扬

专　　业 计算机科学与技术（精英班）

授课老师 郭玉臣

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

深入了解 RAM 与寄存器堆的原理。

用 logicsim 画出一个包含 16 个寄存器的寄存器堆原理图。

学习使用 Verilog HDL 语言设计实现 RAM 以及寄存器堆。

二、 硬件逻辑与原理

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。下图为 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。

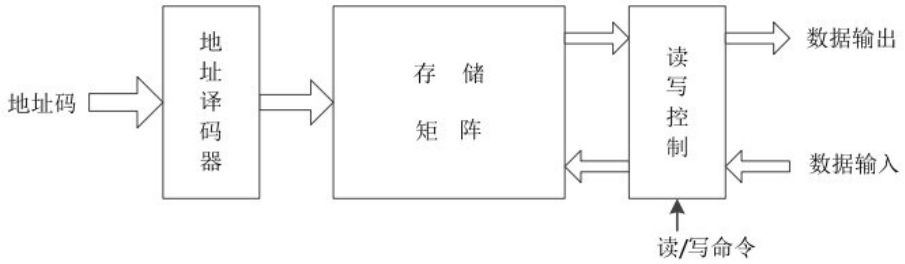
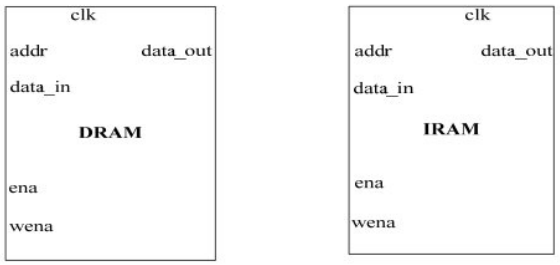


图 1 RAM的逻辑结构图

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。

下图为本实验所需要实现的 RAM 的示意图。

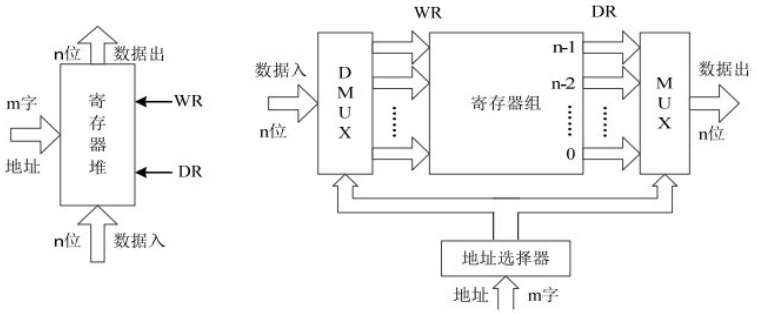


2) 寄存器堆(regfiles)

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。

下图二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。

下图三为使用logisim绘制的由16个四位寄存器组成的寄存器堆电路原理图，它由一个4-32译码器（DMUX）,16个四位寄存器以及两个32选1（MUX）组成。



（a）逻辑结构图（b）原理示意图

图 2

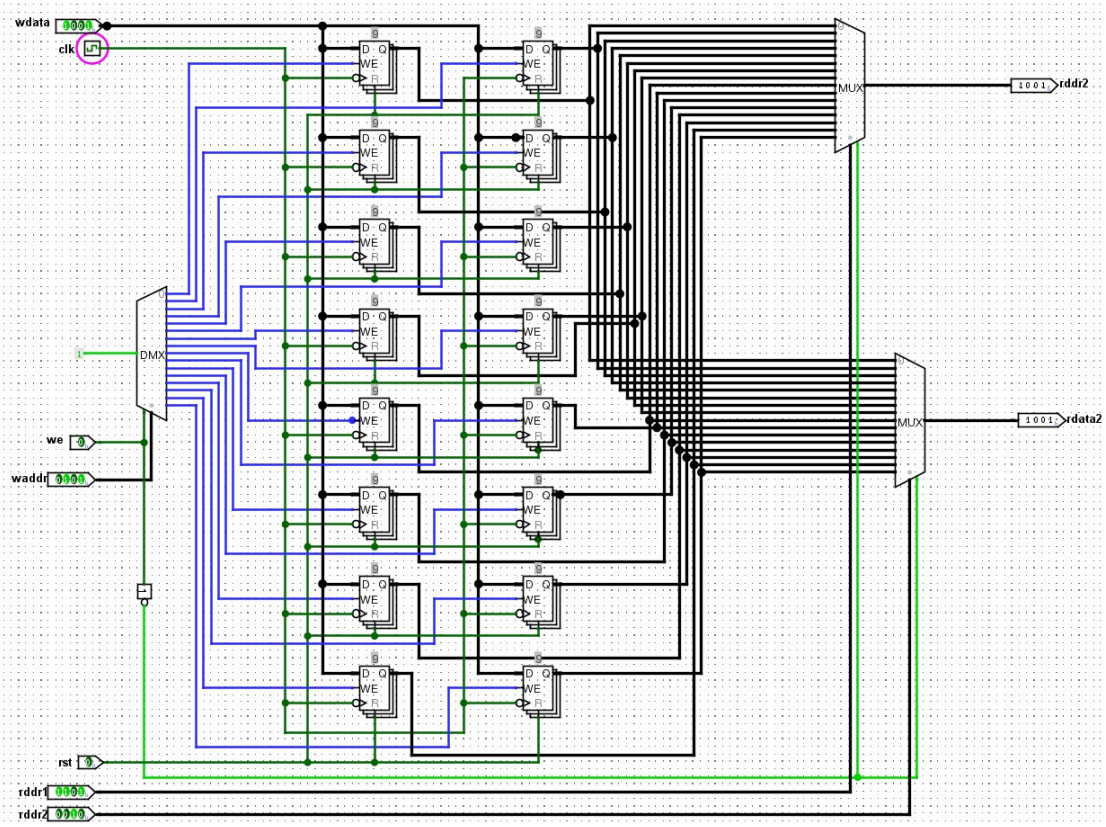
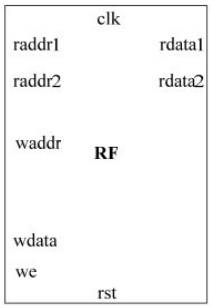


图 3

- 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 32 位选1 选择器，使用两个输入 raddr1，raddr2 控制输出的值。

下图为本实验所要建模的寄存器堆功能框架图：



三、 模块建模

1) ram

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
   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 [31:0] data_out //存储器读出的数据，
)
```

提示：可以使用 reg 数组来实现，大小至少 1024bit。 可以利用$readmemh("文件名", 数组名) 语句使用文件初始化 reg 数组。

Verilog代码描述

```
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] tmp [31:0];
    integer i;
    initial begin
        for ( i = 0;i <= 31;i = i+1 )
            tmp[i] <= 32'b0;
    end
    always @(posedge clk) begin
        if (ena) begin
            if (wena) begin
                tmp[addr] <= data_in;
            end
        end
        else begin
            data_out <= 32'bz;
        end
    end
    always @(*) begin //异步读取
        if (ena) begin
            if(~wena)
                data_out <= tmp[addr];
        end
        else
            data_out <= 32'bz;
    end
endmodule
```

- **功能描述**

ram模块实现了一个简单的异步RAM，采用了时序逻辑的方式，使用行为型描述，通过时钟信号clk和读写信号wena进行异步操作，通过使能信号ena控制模块启用，在上升沿时，根据wena判断进行读还是写，若是写则将data__in写入addr指定的地址中；同时，在使能的情况下，不论clk上升沿是否到来，根据wena进行读操作，将addr指定地址的数据读到data_out中。

2) ram2

接口定义

```
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据。
                      写入的数据在 clk 上升沿时被写入
)
```

Verilog代码描述

```
module ram2 (
   input clk,
   input ena,
   input wena,
   input [4:0] addr,
   inout [31:0] data);
    reg [31:0] ram_data [31:0];
    reg [31:0] tmp;
    integer i;
    initial begin
        for (i = 0;i <= 31;i = i+1)
            ram_data[i] <= 0;
    end
    always @(posedge clk) begin
        if (ena) begin
            if (wena) begin
                ram_data[addr] = data;
            end
        end
    end
    always @(*) begin
        if (~ena) begin
            tmp <= 32'bz;
        end
        else begin
            if (~wena)
                tmp = ram_data[addr];
        end
    end
    assign data = ~wena? tmp : 32'bz;
endmodule
```

功能描述

ram2模块同ram模块功能类似，也是一个异步RAM。ram2逻辑如下：在时钟上升沿，根据写使能信号wena将输入数据data写入指定地址（addr确定）的存储器；异步读取通过组合逻辑实现，可以在时钟之外的任何时间读取存储器的内容，如果未使能，inout端口data被设置为高阻态，但是未处理越界地址和其他错误条件，需要优化。

3) 寄存器堆 Regfiles

接口定义

```
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
);
```

要求：使用以前实验中的译码器、寄存器、以及选择器的模块实例化来实现。

Verilog代码描述

```
`timescale 1ns / 1ps
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] decAddr;
    wire[31:0] OutData[31:0];
    decoder dec(waddr,we,decAddr);
    pcreg reg0(clk, rst, decAddr[0], wdata, OutData[0]);
    pcreg reg1(clk, rst, decAddr[1], wdata, OutData[1]);
    ****省略****
    pcreg reg31(clk, rst, decAddr[31], wdata, OutData[31]);
    wire re;
    assign re=~we;
    selector321 sel1(
    OutData[0], OutData[1], OutData[2], OutData[3],
    OutData[4], OutData[5], OutData[6], OutData[7],
    OutData[8], OutData[9], OutData[10], OutData[11],
    OutData[12], OutData[13], OutData[14], OutData[15],
    OutData[16], OutData[17], OutData[18], OutData[19],
    OutData[20], OutData[21], OutData[22], OutData[23],
    OutData[24], OutData[25], OutData[26], OutData[27],
    OutData[28], OutData[29], OutData[30], OutData[31],
    raddr1, re, rdata1
  );
    selector321 sel2(
    OutData[0], OutData[1], OutData[2], OutData[3],
    OutData[4], OutData[5], OutData[6], OutData[7],
    OutData[8], OutData[9], OutData[10], OutData[11],
    OutData[12], OutData[13], OutData[14], OutData[15],
    OutData[16], OutData[17], OutData[18], OutData[19],
    OutData[20], OutData[21], OutData[22], OutData[23],
    OutData[24], OutData[25], OutData[26], OutData[27],
    OutData[28], OutData[29], OutData[30], OutData[31],
    raddr2, re, rdata2
  );
endmodule
```

```
`timescale 1ns / 1ps
module pcreg(
    input clk,
    input rst,
    input ena,
    input [31:0] data_in,
    output reg [31:0] data_out
    );
    always @ (negedge clk or posedge rst) begin
        if (rst)
            data_out = 32'h0000_0000;
        else
            if (ena)
                data_out = data_in;
    end
endmodule
```

```
module decoder(
    input [4:0] iData,
    input iEna,
    output reg [31:0] oData
    );
    always @(*) begin
        if(iEna) begin
            oData  = 32'b0;
            oData[iData] = 1'b1;
        end
        else begin
            oData=32'bz;
        end
    end
endmodule
```

```
module selector321(
    input [31:0] iC0,
    input [31:0] iC1,
    ****省略****
    input [31:0] iC31,
    input [4:0] addr,
    input ena,
    output reg [31:0] oZ);
    always @(*) begin
        if(ena == 1)
            case(addr)
                5'b00000: oZ = iC0;
                5'b00001: oZ = iC1;
                ****省略****
                5'b11111: oZ = iC31;
                default: oZ = {32{1'bz}};
            endcase
        else
            oZ = {32{1'bz}};
    end
```

功能描述

Regfiles模块实现了一个32个32位寄存器的寄存器堆。通过实例化之前编写的译码器（decoder），寄存器（pcreg）以及32选1选择器（selector321）模块实现，支持异步写入和并行异步读取两个地址的数据。复位信号rst用于将所有寄存器清零，写使能信号we用于选择性地写入数据，读取数据由两个选择器selector321实现，通过读raddr1和raddr2选择对应的数据输出。

整体而言，这个模块构成了一个灵活的寄存器文件，适用于处理器设计中的寄存器堆要求。

四、 测试模块建模

(一) ram_tb

```
`timescale 1ns / 1ps
module ram_tb;
  reg clk, ena, wena;
  reg [4:0] addr;
  reg [31:0] data_in;
  wire [31:0] data_out;
  reg [31:0] tmp[31:0];
  ram uut (
    .clk(clk),
    .ena(ena),
    .wena(wena),
    .addr(addr),
    .data_in(data_in),
    .data_out(data_out)
  );
  integer i = 0;
  initial  begin

$readmemb("E:\\NL\\projects\\homework\\ex08\\ram.srcs\\sources_1\\new\\data.txt", tmp);
```

```
clk = 1;
    addr=0;
    data_in=0;
  end
  always #10 clk = ~clk;
  initial begin
    ena <= 0;
    wena <= 1;
    #23 ena <= 1;
    for (i = 0; i < 32; i = i + 1) begin
      addr <= i;
      data_in <= tmp[i];
      #20;
    end
    wena <= 0;
    #11 addr = 5'b10101;
    #22 addr = 5'b01011;
    #33 addr = 5'b11111;
    #44 addr = 5'b00011;
    #110 ena = 0;
  end
endmodule
```

(二) Ram2_tb

```
`timescale 1ns / 1ps
module ram2_tb ();
    reg clk, ena, wena;
    reg [4:0] addr;
    reg [31:0] data_in;
    wire [31:0] data_out;
    wire [31:0] data;
    reg [31:0] ram_data[31:0];
    integer i = 0;
    ram2 uut(
        .clk(clk),
        .ena(ena),
        .wena(wena),
        .addr(addr),
        .data(data)
    );
    assign data = wena ? data_in : 32'bz;
    assign  data_out  =  ~wena  ?  data  :
32'bz;
    initial
$readmemb("data.txt",ram_data);
    initial begin
        clk = 0;
    end
```

```
    always #10 clk = ~clk;
    initial begin
       ena = 0;
       wena = 1;
       #25 ena = 1;

      for (i = 0; i < 32; i = i + 1) begin
           addr = i;
           data_in = ram_data[i];
           #10;
       end
       wena = 0;
       #10 addr = 5'b10100;
       #10 addr = 5'b10101;
       #10 addr = 5'b10110;
       #10 addr = 5'b11111;
       #10 addr = 5'b10001;
       #10 addr = 5'b00001;
       #10 addr = 5'b11001;
       #10 addr = 5'b10100;
       #50 ena = 0;
       #100 $finish;
    end
endmodule
```

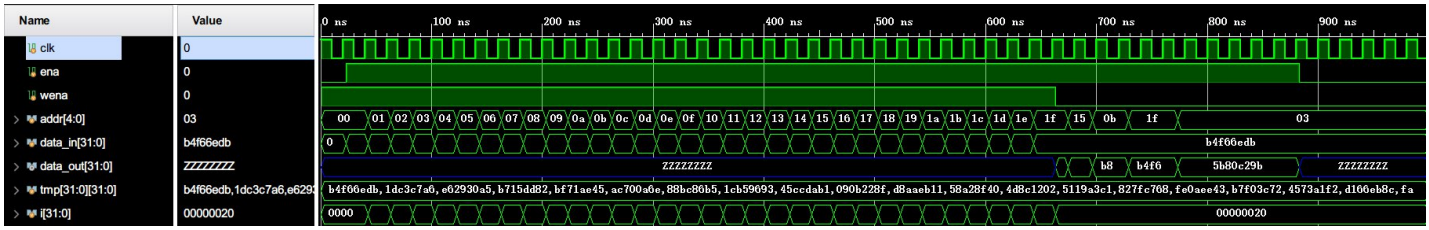
(三) Regfiles_tb

```
`timescale 1ns/1ns
module Regfiles_tb ();
    reg clk,rst,we;
    reg [4:0] raddr1,raddr2,waddr;
    reg [31:0] data;
    wire [31:0] dataO1,dataO2;
    Regfiles
uut(clk,rst,we,raddr1,raddr2,waddr,data,d
ataO1,dataO2);
    initial begin
        rst=1;
        #10
        rst=0;
        we=1;
        #300
        we=0;
    end
    initial begin
        clk=0;
        repeat(100) begin
            #10 clk=~clk;
        end
    end
    initial begin
        data=0;
        repeat(31) begin
            # 10 data=data+1;
        end
        #80
        data=0;
        repeat(31) begin
            # 10 data=data+1;
        end
    end
    initial begin
        waddr=0;
        repeat(31) begin
            # 10 waddr=waddr+1;
        end
        #80
        waddr=0;
        repeat(31) begin
            # 10 waddr=waddr+1;
        end
    end
    initial begin
        raddr1=0;
        raddr2=31;
        repeat(31) begin
            # 10 raddr1=raddr1+1;
            raddr2=raddr2-1;
        end
        #100
        raddr1=0;
        raddr2=31;
        repeat(31) begin
            # 10 raddr1=raddr1+1;
            raddr2=raddr2-1;
        end
    end
endmodule
```

五、 实验结果

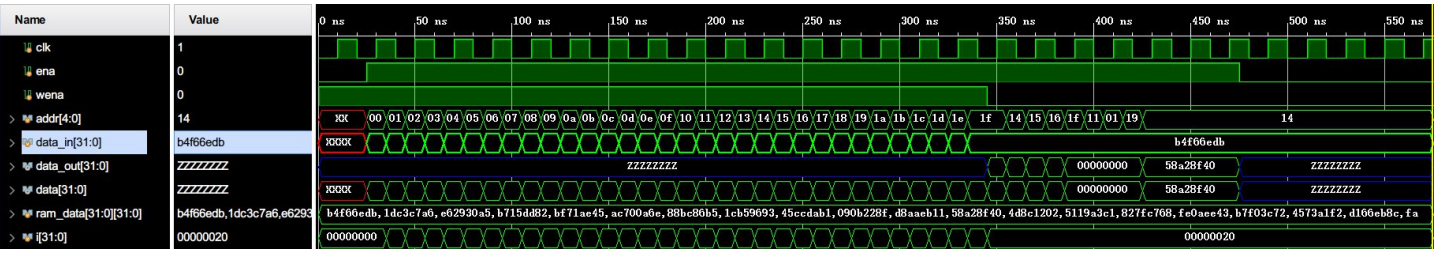
(一) ram

- 波形图仿真



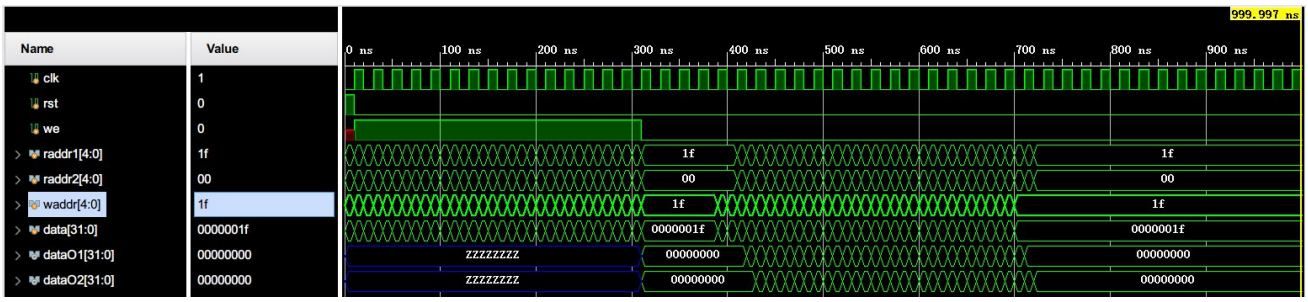
(二) ram2

- 波形图仿真

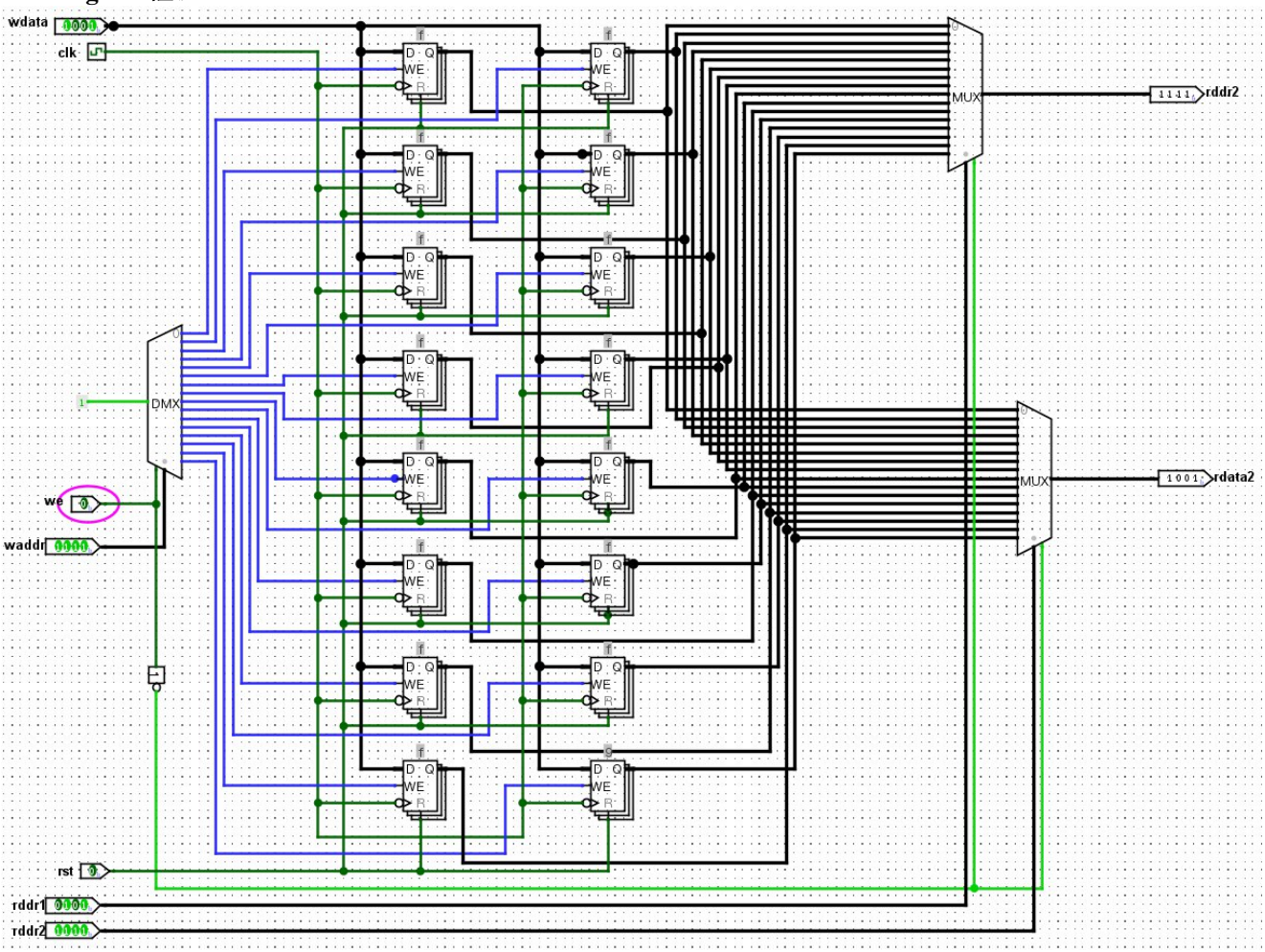


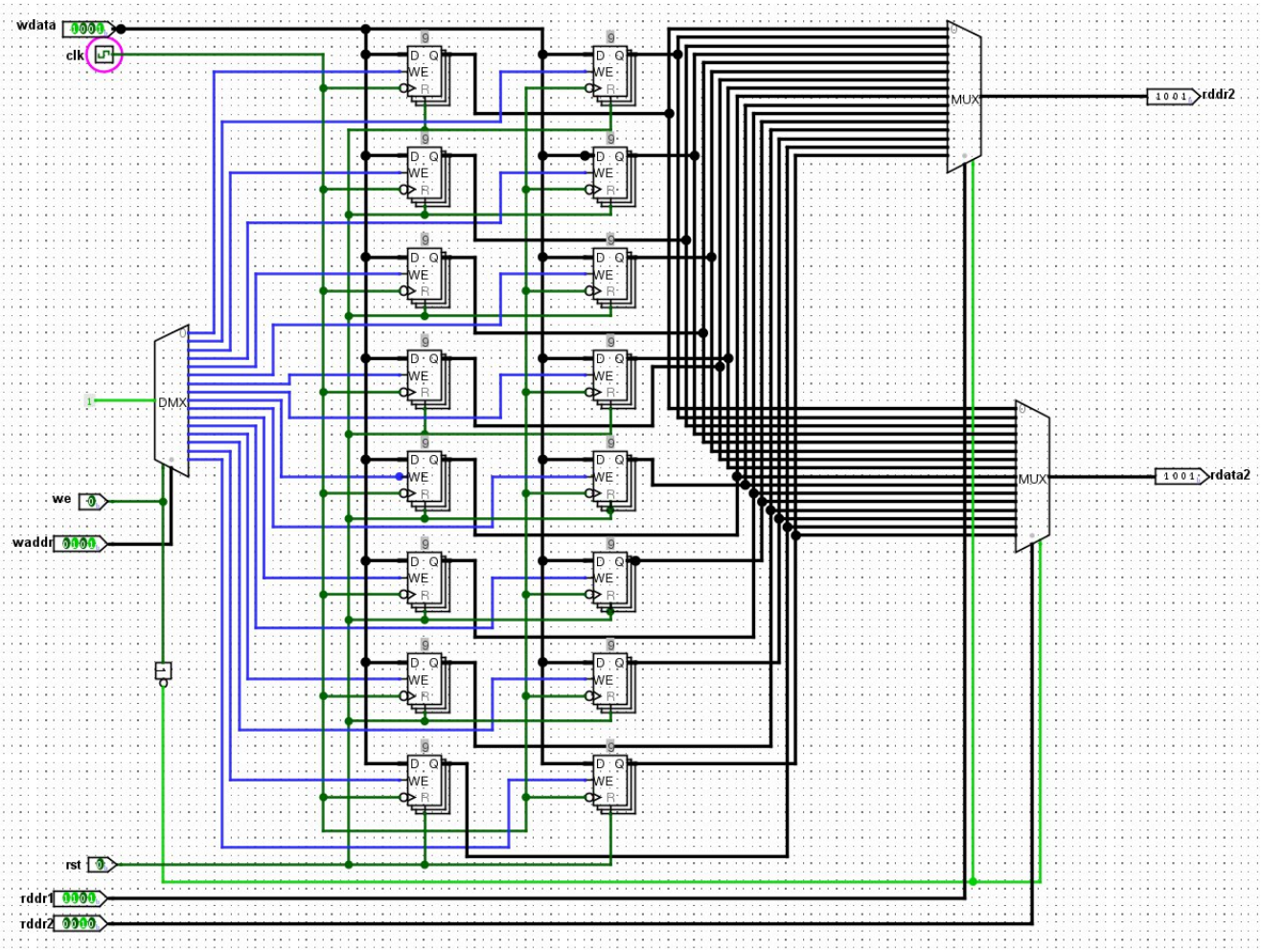
(三) Regfiles

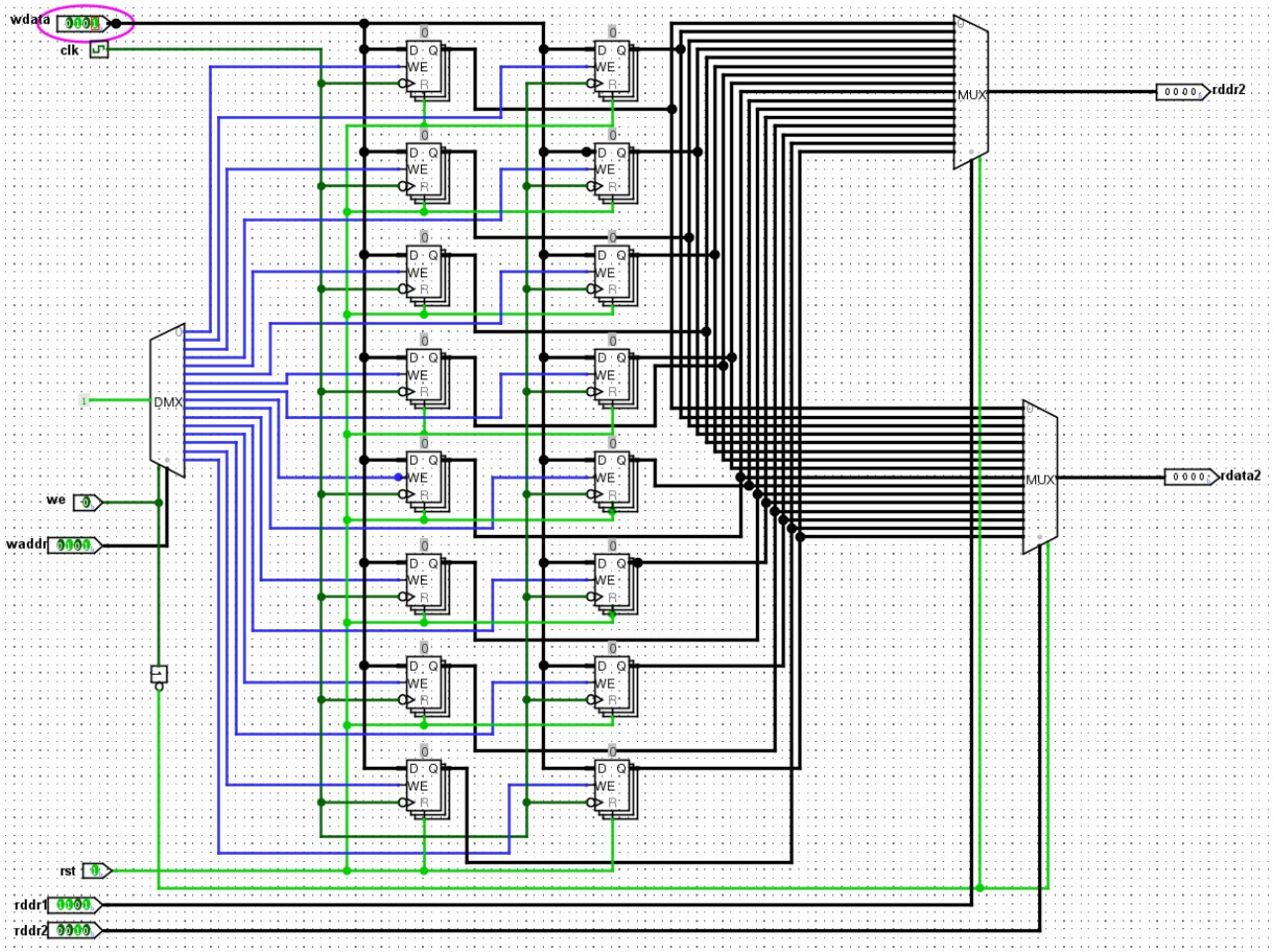
- 波形图仿真

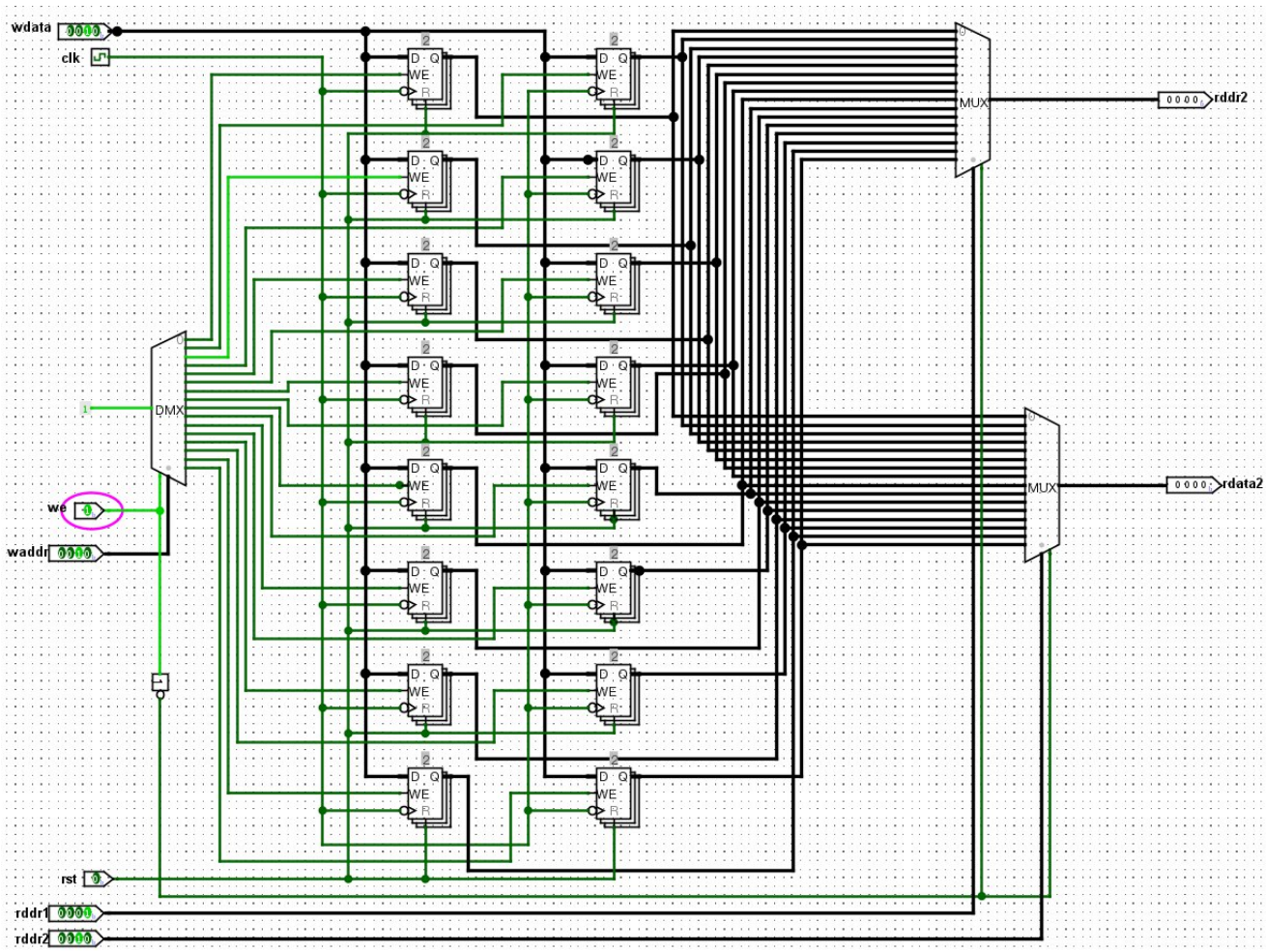


- logisim验证




wdata
clk
we
waddr
rst
rddr1
rddr2
DMX
MUX
MUX
rddr2
rdata2
D Q
WE
R


wdata
clk
we
waddr
rst
rddr1
rddr2
DMX
MUX
MUX
rddr2
rdata2
D
Q
WE
R


wdata
clk
we
waddr
rst
rddr1
rddr2
DMX
MUX
MUX
rddr2
rdata2
D Q
WE
R