实验四 存储器阵列设计

**一、实验目的**

1 掌握Verilog语言和Vivado、Logisim开发平台的使用；

2 掌握存储器和寄存器组的设计和测试方法。

**二、实验内容**

1 存储器设计与测试

2 寄存器组设计与测试

**三、实验要求**

1 掌握Vivado或Logisim开发工具的使用，掌握以上电路的设计和测试方法；

2 记录设计和调试过程（Verilog代码/电路图/表达式/真值表，Vivado仿真结果，Logisim验证结果等）；

3 分析Vivado仿真波形/Logism验证结果，注重输入输出之间的对应关系。

**四、实验过程及分析**

1、4Kx32位寄存器：

代码：

**1Kx16位寄存器**：

设计文件：

module RAM\_1Kx16\_inout(Data,Addr,Rst,R\_W,CS,CLK);

parameter Addr\_Width=10;

parameter Data\_Width=16;

parameter SIZE=2 \*\* Addr\_Width;

inout [Data\_Width-1:0]Data;

input [Addr\_Width-1:0]Addr;

input Rst;

input R\_W;

input CS;

input CLK;

integer i;

reg[Data\_Width-1:0]Data\_i;

reg[Data\_Width-1:0]RAM[SIZE-1:0];

assign Data=(R\_W)?Data\_i:16'bz;

always@(\*)begin

casex({CS,Rst,R\_W})

4'bx1x:for(i=0;i<=SIZE-1;i=i+1)RAM[i]=0;

//$readmemb("D:/ram\_data\_b.txt",RAM,0,1023);

//$readmemb("D:/ram\_data\_b.txt",RAM,0,1023);

4'b101:Data\_i=RAM[Addr];

4'b100:RAM[Addr]=Data;

default:Data\_i=16'bz;

endcase

end

endmodule

代码分析：

本模块的实现难度主要在于inout 类型Data，会随读写信号的不同改变为寄存器的写入端和读出端，直接声明为reg或者wire都是不合适的，我们选择引入临时寄存器Data\_i，并使用assign Data=(R\_W)?Data\_i:16'bz，此语句的作用是：当R\_W信号为1时，此时为读周期，Data数据线的值为内部寄存器中Data\_i中的值，即读出数据；当R\_W信号为0时，此时为写周期，为不与外部写入数据造成冲突，在内部置为高阻态16‘bz，以便外部数据写入。当{CS,Rst,R\_W}==x1x，即当Rst信号为1时，不管其他信号如何，都对寄存器当前地址的内容做置零操作。当{CS,Rst,R\_W}==101，处于读周期，将寄存器RAM地址为Addr的数据读入寄存器Data\_i内，以便数据线Data读出；当{CS,Rst,R\_W}=100时，此时为写周期，将Data数据线从外部连接的值写入寄存器RAM[Addr]内。

**24译码器：**

module Decoder24(Y, I);

input [1:0] I;

output reg [3:0] Y;

always @(\*) begin

case(I)

2'b00: Y = 4'b0001;

2'b01: Y = 4'b0010;

2'b10: Y = 4'b0100;

2'b11: Y = 4'b1000;

endcase

end

endmodule

代码分析：24译码器以多次实现，此处不多做分析。

**4Kx32位寄存器：**

module RAM\_4Kx32\_inout

#(parameter Addr\_Width = 12,

Data\_Width = 32)

(inout [Data\_Width-1:0] Data,

input [Addr\_Width-1:0] Addr,

input Rst,

input R\_W,

input CS,

input CLK);

wire [3:0] CS\_i;

Decoder24 Decoder24\_1(CS\_i, Addr[Addr\_Width-1:Addr\_Width-2]);

RAM\_1Kx16\_inout CSO\_H\_16bit(Data[Data\_Width-1:Data\_Width/2], Addr[Addr\_Width-3:0], Rst, R\_W, CS\_i[0]&CS, CLK), CSO\_L\_16bit(Data[Data\_Width/2-1:0], Addr[Addr\_Width-3:0], Rst, R\_W, CS\_i[0]&CS, CLK);

RAM\_1Kx16\_inout CS1\_H\_16bit(Data[Data\_Width-1:Data\_Width/2], Addr[Addr\_Width-3:0], Rst, R\_W, CS\_i[1]&CS, CLK);

RAM\_1Kx16\_inout CS1\_L\_16bit(Data[Data\_Width/2-1:0], Addr[Addr\_Width-3:0], Rst, R\_W, CS\_i[1], CLK);

RAM\_1Kx16\_inout CS2\_H\_16bit(Data[Data\_Width-1:Data\_Width/2], Addr[Addr\_Width-3:0], Rst, R\_W, CS\_i[2]&CS, CLK);

RAM\_1Kx16\_inout CS2\_L\_16bit(Data[Data\_Width/2-1:0], Addr[Addr\_Width-3:0], Rst, R\_W, CS\_i[2]&CS, CLK);

RAM\_1Kx16\_inout CS3\_H\_16bit(Data[Data\_Width-1:Data\_Width/2], Addr[Addr\_Width-3:0], Rst, R\_W, CS\_i[3]&CS, CLK);

RAM\_1Kx16\_inout CS3\_L\_16bit(Data[Data\_Width/2-1:0], Addr[Addr\_Width-3:0], Rst, R\_W, CS\_i[3]&CS, CLK);

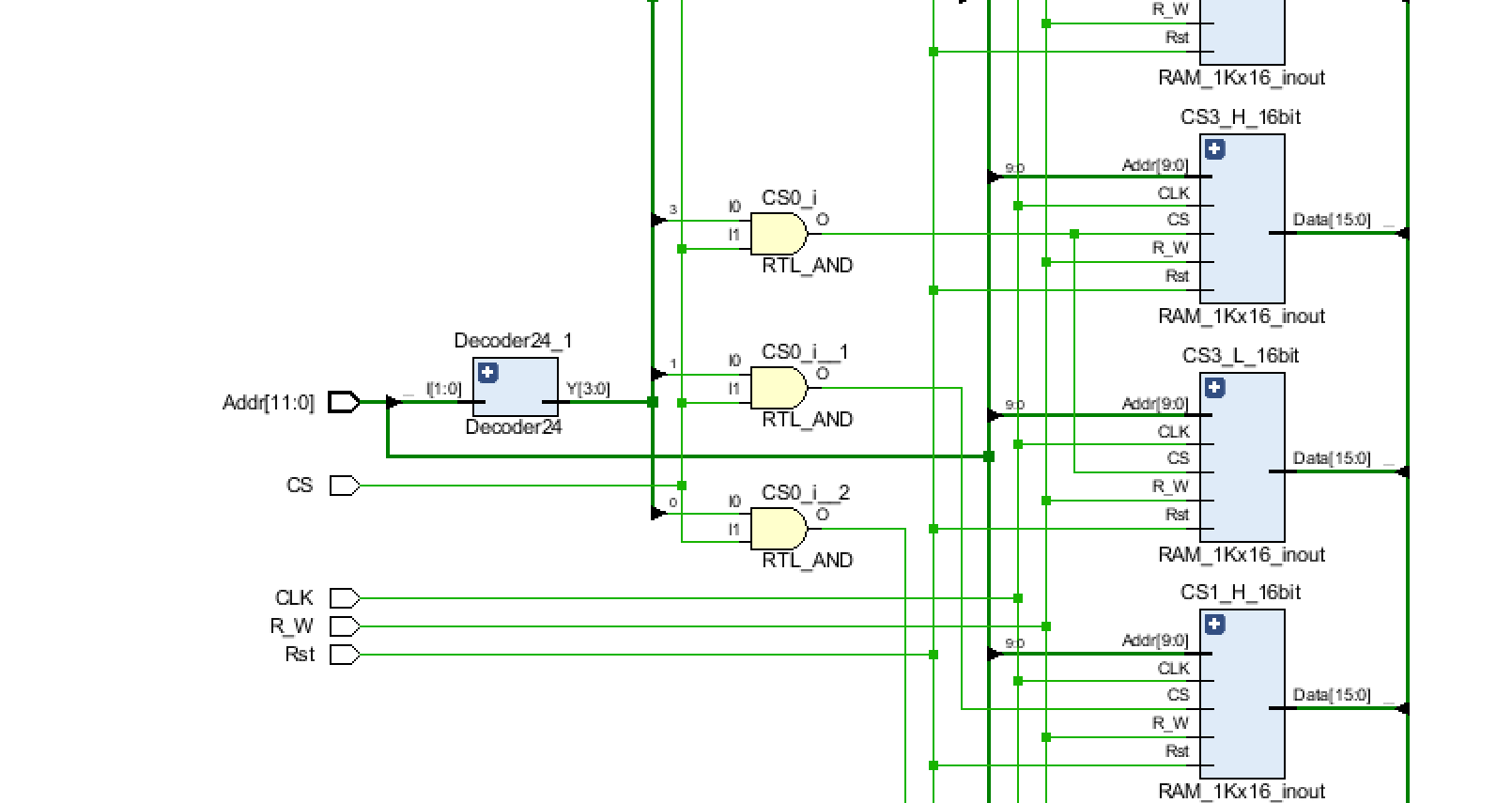
endmodule

代码分析：

本实验中4Kx32位寄存器主要通过调用子模块1Kx16位寄存器实现。要通过1Kx16实现4Kx32，既需要字扩展也需要位扩展。

字扩展：字扩展实现较为简单，要用16位寄存器实现32位寄存器，只需要直接将相同的地址赋给两个16位寄存器，且32位寄存器数据线宽度为16位寄存器数据线宽度2倍，两个寄存器的数据线分别作封装后的32位寄存器的高位和地位即可。

位扩展：位扩展相比字扩展稍复杂。要从1K（地址线为10位）扩展为4K（地址线为12位），多出的2位地址通过24译码器译码后作为片选信号，将译码得到的4位片选信号与封装后的4K寄存器的使能信号CS作逻辑与后的每一位分别作为4片寄存器组的使能信号，从而达到片选的目的。

逻辑仿真图：  
 

仿真文件：

module sim\_RAM\_4Kx32\_inout;

parameter Addr\_Width=12;

parameter Data\_Width=32;

wire [Data\_Width-1:0]Data;

reg [Addr\_Width-1:0]Addr;

reg Rst;

reg R\_W;

reg CS;

reg CLK;

reg [Data\_Width-1:0] temp;

assign Data=(R\_W)?32'bz:temp;

RAM\_4Kx32\_inout ram\_4Kx32(Data,Addr,Rst,R\_W,CS,CLK);

initial

begin

Addr=12'h0000\_0000;Rst=0;R\_W=0;CS=1;CLK=0;temp=32'h1111\_1111;

fork

repeat(100)#5 CLK=~CLK;

repeat(100)#100 Rst=~Rst;

repeat(100)#30 R\_W=~R\_W;

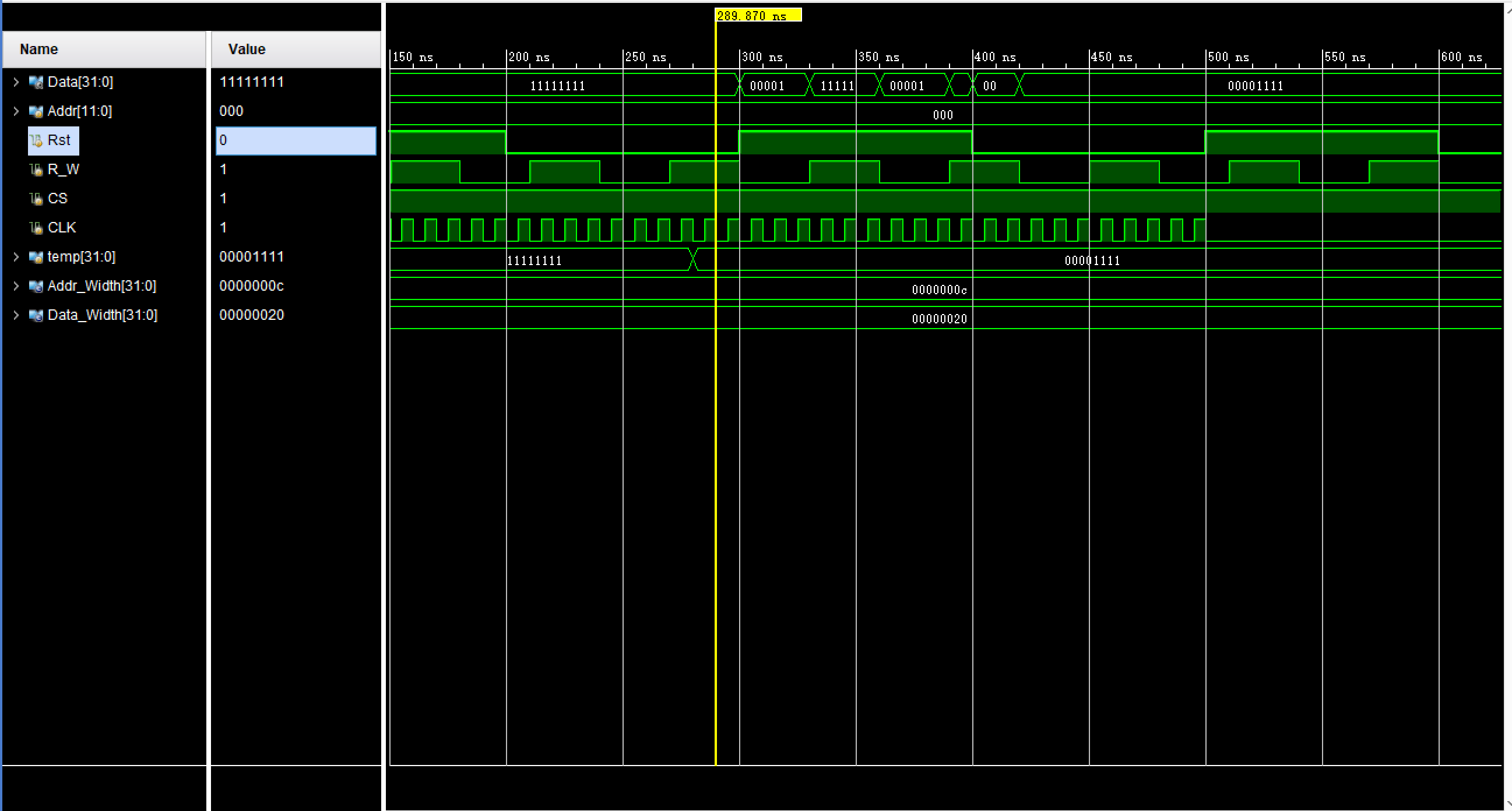
#280 temp=32'h0000\_1111;

join

end

endmodule

仿真时序图：



分析时序图：

分析时序图可以得到：

（1）、当处于写周期R\_W=0时，数据线Data的值为从外部写入的数据11111111（temp寄存器中的值）。

（2）、在黄线处，当处于读周期R\_W=1时，数据线Data的值为从寄存器RAM[Adrr]中读取的值，通过时序图可以看到，黄线处此时temp寄存器的值已经发生改变，但Data的值并不发生改变，说明此时确实已经是在读寄存器内部值而不是继续使用外部寄存器temp的值。

（3）RST复位信号正常工作，当RST信号为高时，进行对内部寄存器目标地址数据清零操作，之所以在RST为高时，Data的值没有立刻变为零，原因是，此时虽然内部寄存器RAM[Addr]已经清零，但是RST仍为1 ，只有在{CS,RST,R\_W}=101,才会进行Data\_i=RAM[Addr]，也就是说此时寄存器Data\_i没有更新，所以Data的值也仍为1111\_1111，。从后续RST信号从高变为低后的信号可以验证此说法：RST从高变为低后，此时R\_W=1，处于读周期，Data\_i寄存器的值可以正常更新RAM[Addr]的值，Data连接到Data\_i，Data变为0，可以说明RST确实将RAM[Addr]处的值置零了。

综上，经过仿真文件验证，设计的4Kx32位寄存器正常工作。

2、IMem寄存器：

设计文件：

define DATA\_WIDTH 32

module IMem(

input [5:0] A,

output [`DATA\_WIDTH-1:0] RD);

parameter IMEM\_SIZE = 64;

reg[`DATA\_WIDTH-1:0] RAM[IMEM\_SIZE-1:0];

initial

$readmemh("C:/Users/LeBlanc/Desktop/11.txt", RAM);

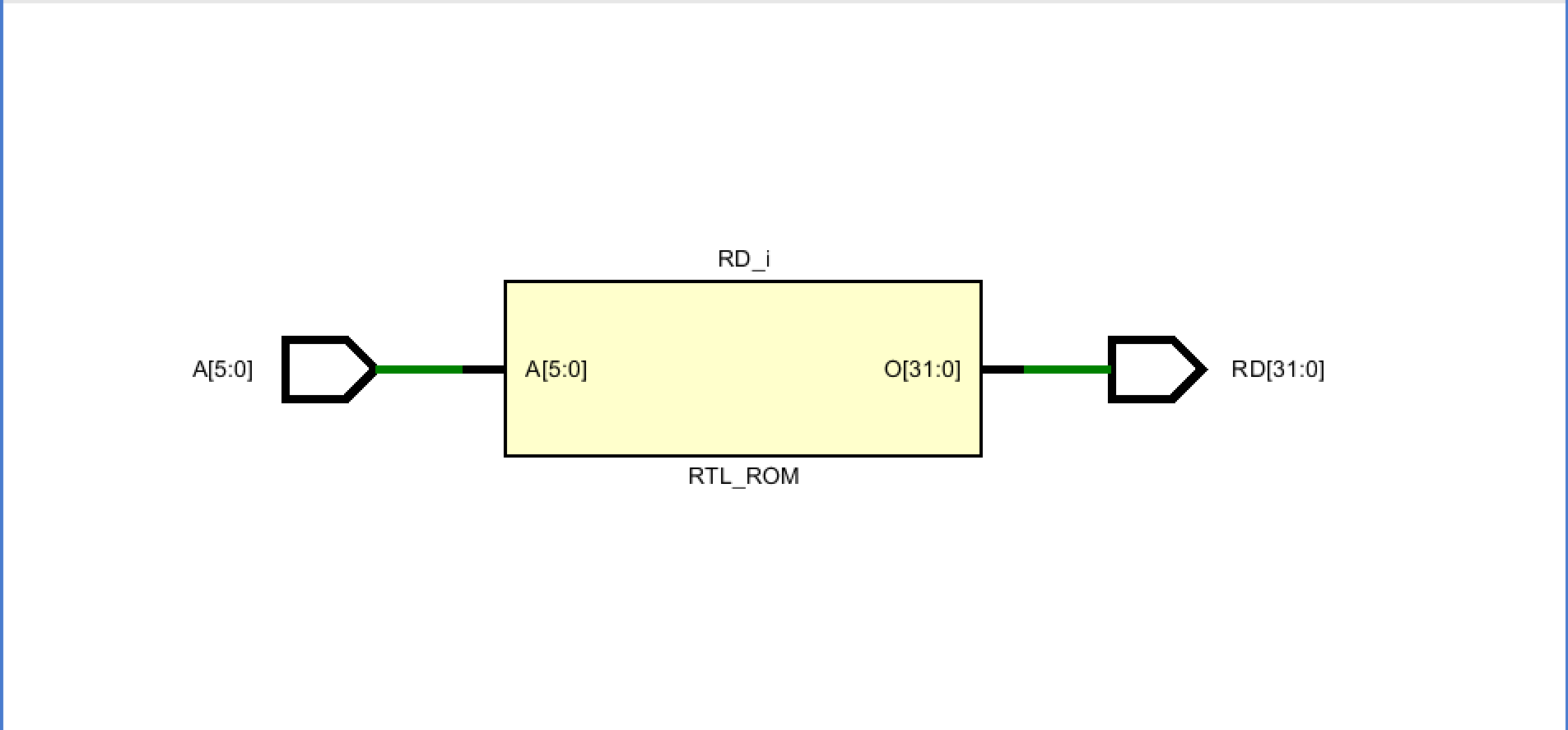
assign RD = RAM[A];

endmodule

代码分析：

本模块主要的特殊之处在于使用了readmemh函数，其中readmemh中的h表示按16进制读取数据，并保存到寄存器RAM中，再通过地址线A控制输出寄存器内容的位置。

逻辑仿真图：



仿真文件:

`define DATA\_WIDTH 32

module sim\_IMem;

reg [5:0] test\_address;

wire [`DATA\_WIDTH-1:0] test\_data;

IMem dut (

.A(test\_address),

.RD(test\_data)

);

initial begin

test\_address=0;

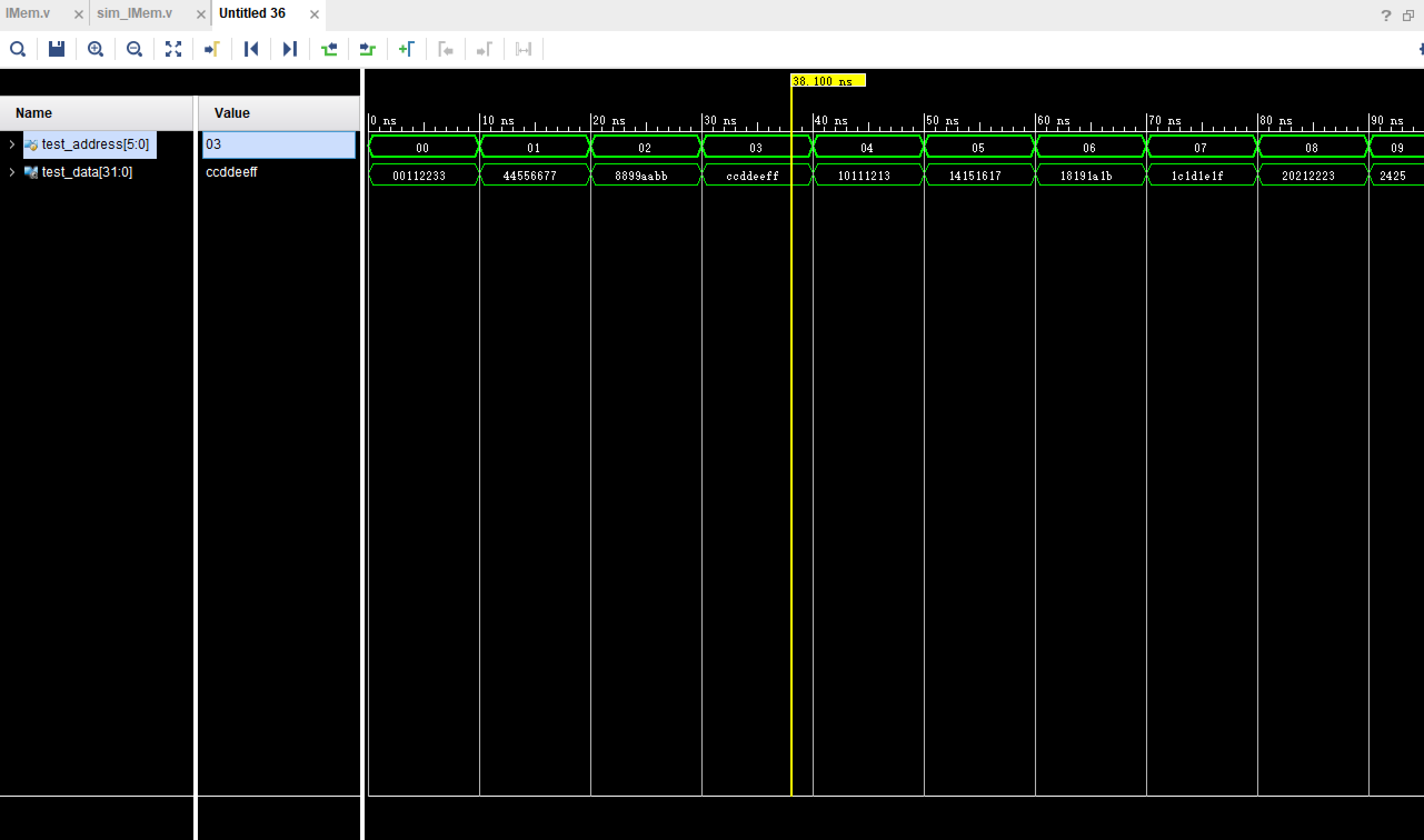
repeat(63) #10 test\_address=test\_address+1;

// 添加模拟代码以应用测试向量并检查结果

end

endmodule

仿真时序图：



分析:仿真文件主要为了观察是否寄存器是否成功从文件中读入数据，遍历所有地址，即检查寄存器所有位置的数据，与文件中的数据进行对比是否成功读入。实验发现成功读入目标文件的数据，实验成功。

3、RegFile寄存器：

设计文件：

`define DATA\_WIDTH 32

module RegFile

#(parameter ADDR\_SIZE = 5)

(input CLK, WE3,

input [ADDR\_SIZE:0] RA1, RA2, WA3,

input [`DATA\_WIDTH-1:0] WD3,

output [`DATA\_WIDTH-1:0] RD1, RD2);

reg [`DATA\_WIDTH-1:0] rf[2 \*\* ADDR\_SIZE-1:0];

always@(posedge CLK)

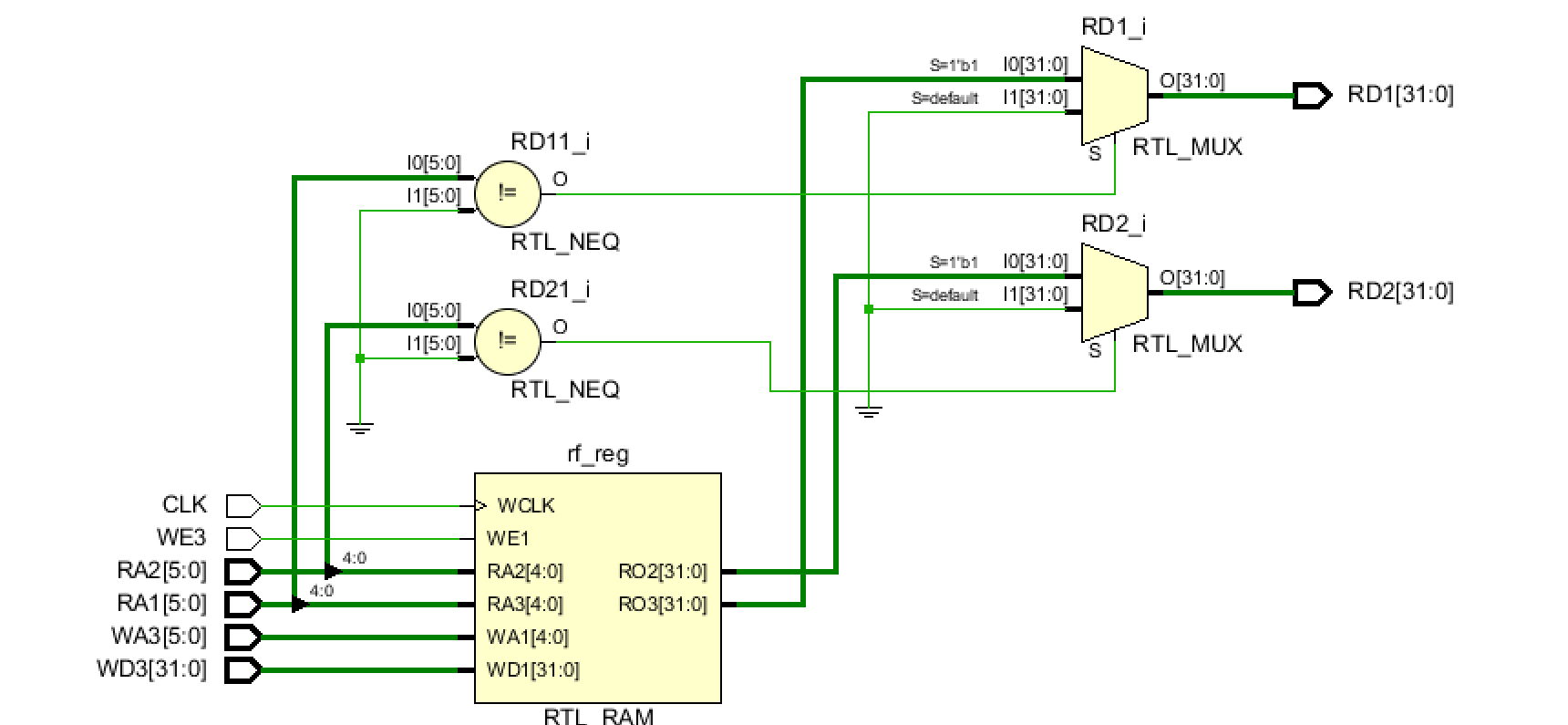
if(WE3) rf[WA3] <= WD3;

assign RD1 = (RA1 != 0) ? rf[RA1] : 0;

assign RD2 = (RA2 != 0) ? rf[RA2] : 0;

endmodule

逻辑仿真图：



代码分析：

本模块是设计一个有一端写入两端写出的寄存器，三个端口相互独立，所以声明三个地址寄存器，两个输入寄存器和一个输入总线。当WE3控制信号有效时，数据写入。当读数据地址不为0时，就读数据。

仿真文件：

module RegFile\_tb;

parameter ADDR\_SIZE = 5;

reg CLK;

reg WE3;

reg [ADDR\_SIZE:0] WA3;

reg [31:0] WD3;

reg [ADDR\_SIZE:0] RA1, RA2;

wire [31:0] RD1, RD2;

initial begin

CLK = 0;

forever #5 CLK = ~CLK;

end

initial begin

WE3 = 1;

WA3 = 8;

WD3 = 9;

RA1=0;

RA2=0;

#10 begin WA3=10;WD3=20; end

#10 WE3 = 0;

#10 RA1 = 8;

#10 RA2 = 10;

#100 $finish;

end

RegFile #(ADDR\_SIZE) uut (

.CLK(CLK),

.WE3(WE3),

.RA1(RA1),

.RA2(RA2),

.WA3(WA3),

.WD3(WD3),

.RD1(RD1),

.RD2(RD2)

);

endmodule

仿真时序图：



分析：

由时序仿真图可以看出，在WD3分别对位置08和0a进行写不同入操作，RA1和RA2成功读出写入值，实验成功。

**五、调试和心得体会**

本次实验相比以前最特殊的一点是使用了inout接口，即既能做输入也能做输出的接口，在一开始调试过程中由于于对inout不熟悉，无法分清内部高阻态和外部高阻态的区别，导致浪费了很多时间，让我意识到以后需要先对设计文件的代码理解透彻后再去进行仿真文件的设计，磨刀不误砍柴工。