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 存储器

存储器是数字系统中用以存储大量信息的设备或部件，是计算机和数字设备中的重要组成部分。存储器可分为随机存取存储器（RAM）和只读存储器（ROM）两大类。RAM由存储矩阵、地址译码器、读/写控制器、输入/输出、片选控制等几部分组成。

对于一个1K\*16位的双数据线RAM，CLK提供时钟信号，由片选信号CS控制存储器进行读/写工作，读/写控制分别由RE和WE实现，两者不同时为1，当RE=1时控制存储器进行读操作，WE=1时控制存储器进行写操作。此外还有清零控制信号Rst，当Rst=1使将RAM中所有存储单元的值置为0（考虑到RAM需要进行字扩展或位扩展来组成更大的存储器，因此Rst进行清零操作时不需要片选信号CS生效，同时也不需要考虑读/写信号，即Rst优先级最高，只要Rst生效就将RAM清零）。用10位输入信号Addr指定读/写操作的存储单元地址。在进行读操作时，将Addr指定的存储单元存储的16位二进制数值输出到Data\_out信号上；在进行写操作时，需要一个16位输入信号Data\_in，将其输入到Addr对应的存储单元中。

一个1K\*16位的双数据线RAM设计代码如下：

module RAM\_1Kx16(

Data\_out,Addr,Rst,RE,WE,CS,CLK,Data\_in

);

parameter Addr\_Width=10;

parameter Data\_Width=16;

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

output reg [Data\_Width-1:0] Data\_out;

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

input Rst;

input RE;

input WE;

input CS;

input CLK;

input [Data\_Width-1:0] Data\_in;

integer i;

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

initial begin

for (i=0;i<SIZE;i=i+1) RAM[i]=i;

end

always @(posedge CLK) begin

casex({CS,Rst,RE,WE})

4'bx1xx:for(i=0;i<SIZE;i=i+1) RAM[i]=0;

4'b1010:Data\_out<=RAM[Addr];

4'b1001:RAM[Addr]<=Data\_in;

default:Data\_out=16'bz;

endcase

end

endmodule

这里采用时钟同步的方式，清零、读/写存储器都在时钟上升沿完成，存储器保持状态直到下一个时钟上升沿。为了测试仿真的结果便于观察和验证，将存储器的所有存储单元存储值初始化为和片内地址相同。

8片1K\*16位RAM可以通过字位扩展连接成一个4K\*32位的存储器。具体实现是每两个RAM通过位扩展连接成一个1K\*32位存储器，这两个RAM在片内寻址时同一位置的存储矩阵行被同时选中，存储数据进行位拼接。4个这样的存储器再进行字扩展，就连接成1K\*32位存储器。字位扩展后，存储器寻址要用到12位地址，其中低10位用于片内寻址，高2位通过24译码器产生片选信号，选中存储地址所在的RAM芯片。设计代码：

module RAM\_4Kx32(

Data\_out,Addr,Rst,RE,WE,CLK,Data\_in

);

parameter Addr\_Width=12;

parameter Data\_Width=32;

output [Data\_Width-1:0] Data\_out;

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

input Rst;

input RE;

input WE;

input CLK;

input [Data\_Width-1:0] Data\_in;

wire [3:0] CS\_i;

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

RAM\_1Kx16 CS0\_H\_16bit(Data\_out[Data\_Width-1:Data\_Width/2],Addr[Addr\_Width-3:0],Rst,RE,WE,CS\_i[0],CLK,Data\_in[Data\_Width-1:Data\_Width/2]),

CS0\_L\_16bit(Data\_out[Data\_Width/2-1:0],Addr[Addr\_Width-3:0],Rst,RE,WE,CS\_i[0],CLK,Data\_in[Data\_Width/2-1:0]);

RAM\_1Kx16 CS1\_H\_16bit(Data\_out[Data\_Width-1:Data\_Width/2],Addr[Addr\_Width-3:0],Rst,RE,WE,CS\_i[1],CLK,Data\_in[Data\_Width-1:Data\_Width/2]),

CS1\_L\_16bit(Data\_out[Data\_Width/2-1:0],Addr[Addr\_Width-3:0],Rst,RE,WE,CS\_i[1],CLK,Data\_in[Data\_Width/2-1:0]);

RAM\_1Kx16 CS2\_H\_16bit(Data\_out[Data\_Width-1:Data\_Width/2],Addr[Addr\_Width-3:0],Rst,RE,WE,CS\_i[2],CLK,Data\_in[Data\_Width-1:Data\_Width/2]),

CS2\_L\_16bit(Data\_out[Data\_Width/2-1:0],Addr[Addr\_Width-3:0],Rst,RE,WE,CS\_i[2],CLK,Data\_in[Data\_Width/2-1:0]);

RAM\_1Kx16 CS3\_H\_16bit(Data\_out[Data\_Width-1:Data\_Width/2],Addr[Addr\_Width-3:0],Rst,RE,WE,CS\_i[3],CLK,Data\_in[Data\_Width-1:Data\_Width/2]),

CS3\_L\_16bit(Data\_out[Data\_Width/2-1:0],Addr[Addr\_Width-3:0],Rst,RE,WE,CS\_i[3],CLK,Data\_in[Data\_Width/2-1:0]);

Endmodule

其中24译码器模块的设计实现代码如下：

module Decoder24(

Y,I

);

output reg [3:0]Y;

input [1:0]I;

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

仿真测试代码：

module sim\_4Kx32(

);

reg [11:0] Addr;

reg [31:0] Data\_in;

wire [31:0] Data\_out;

reg Rst,RE,WE,CLK;

initial begin

Rst=0;RE=1;WE=0;CLK=0;

Data\_in=32'hFF00\_ABCD;

Addr=12'b1;

forever #10 CLK=~CLK;

end

initial begin

#50 Addr=12'b0010\_1110\_0101;

#50 Addr=12'b0110\_0011\_0000;

Rst=1;

#50 fork Rst=0;RE=1;join

#50 fork RE=0;WE=1;join

#50 fork RE=1;WE=0;join

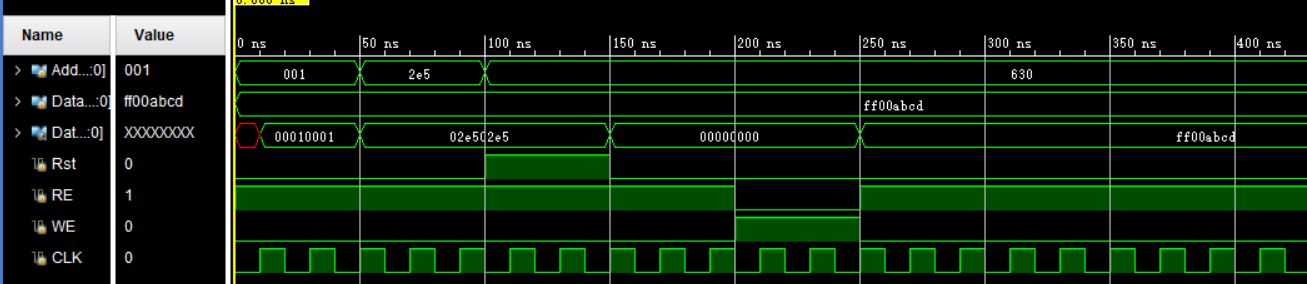
end

RAM\_4Kx32 test(Data\_out,Addr,Rst,RE,WE,CLK,Data\_in);

Endmodule

仿真波形结果：

-->



仿真代码中初始时Rst=0，RE=1，WE=0，Addr=001H，此时进行读操作，在第一个时钟上升沿到来时，读出地址为001H的存储单元存储的信息00010001H（由1K\*16位RAM的存储初始化的代码，高16位和低16位存储值相同，都等于片内寻址时的地址，并且高位补0，因此可以验证这个读操作读取的存储值是正确的）。50ns后Addr更改为2e5H，读出存储数同样是正确的。再过50ns，Rst=1，对所有存储单元清零，同时地址信号Addr变为630H，这时候此前读出的数会保持一段时间，50ns后Rst信号收回，读出630H的存储单元存储数为0。200ns时，RE=0，WE=1，向630H的存储单元写入数据fff0abcdH。写入信号持续50ns，之后RE=1，WE=0，读出630H的存储单元存储值正是此前写入的值。至此，存储器的所有功能都得到了验证，波形结果都是符合设计预期的。

2 指令存储器

指令存储器的功能比较简单，它没有写入功能，存储器根据输入的指令地址A取出存储器中存放在该地址的存储单元的指令RD作为输出。这个指令同样是一串二进制数，因此就读操作而言指令存储器和存放数据的存储器没有区别（这符合冯诺依曼计算机“存储程序”的思想，程序和数据都用二进制表示来存储，不加以区分，只在取指时根据时序关系分辨指令和数据）。设计代码：

module IMem(

A,RD

);

parameter Data\_Width=8;

parameter IMEM\_Size=64;

input [5:0]A;

output [Data\_Width-1:0]RD;

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

integer i=0;

initial begin

for (i=0;i<IMEM\_Size;i=i+1) RAM[i]=i;

end

assign RD=RAM[A];

endmodule

这里指令字长8位，存储容量一共可以存储64条指令，为便于测试观察，初始化存储器时每个存储单元存储的二进制串就是其地址。

仿真代码：

module sim\_IMem(

);

reg [5:0] A;

wire [7:0] RD;

integer i;

initial begin

A=6'b000001;

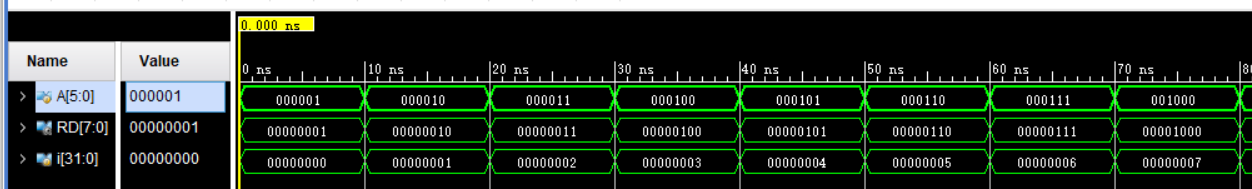
for (i=0;i<64;i=i+1) #10 A=A+1;

end

IMem test(A,RD);

Endmodule

仿真结果：



依次从1开始访问存储器的各个地址，输出的指令二进制数正好对应，这是符合设计结果的。

3 寄存器文件

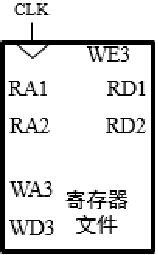
寄存器文件又称寄存器堆，是CPU中多个寄存器组成的阵列，通常由快速的静态随机读写存储器实现，具有专门的读端口与写端口，可以多路并发访问不同的寄存器。

读寄存器：根据输入的地址读出相应寄存器存储的数据（0号寄存器恒零）。

写寄存器：根据输入的地址信号，把输入的数据写入相应寄存器。

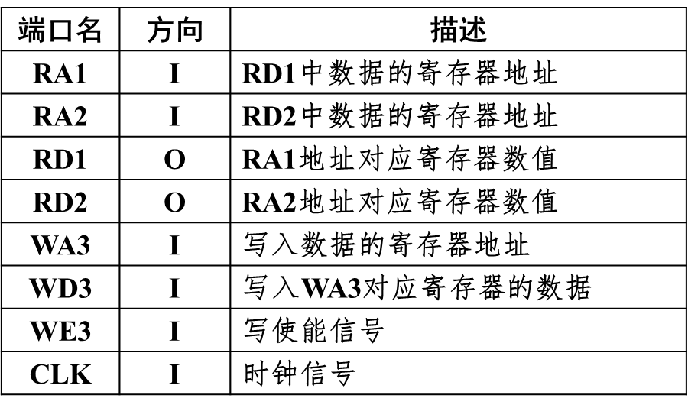
一个支持一次写入一个寄存器，或者并行写入2个寄存器的寄存器堆逻辑框图如下：

-->



各端口功能说明如下：

-->



1个32\*32位的寄存器堆设计代码如下：

`define DATA\_WIDTH 32

module RegFile(

CLK,WE3,RA1,RA2,WA3,WD3,RD1,RD2

);

parameter ADDR\_SIZE=5;

input CLK,WE3;

input [ADDR\_SIZE-1:0] 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];

integer i;

integer first=1;

initial begin

rf[0]=0;

for (i=1;i<2\*\*ADDR\_SIZE;i=i+1) rf[i]=first<<i;

end

always @(posedge CLK)

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

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

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

endmodule

寄存器堆在CLK上升沿进行读/写操作，这里为了测试观察方便，将每个寄存器堆初始化为0地址寄存器值全为0，其他地址上的寄存器存储值中只有和它的地址对应的一位上是1，其他位上是0。在每个时钟上升沿对输入端口进行检查，当写使能输入WE3=1时，控制寄存器堆进行写操作，将写入数据WD3写入输入地址WA3对应的寄存器中；同时根据输入的读地址RA1和RA2并行地再寄存器堆中读出对应地址的存储值RD1和RD2输出。

测试代码：

module sim\_RegFile(

);

reg CLK,WE3;

reg [31:0] WD3;

reg [4:0] WA3,RA1,RA2;

wire [31:0] RD1,RD2;

initial begin

CLK=0;WE3=0;RA1=0;RA2=0;WA3=5'b01101;WD3=32'hFEDC\_BA98;

forever #10 CLK=~CLK;

end

initial begin

#50 RA1=5'b11011;RA2=5'b00010;

#50 WE3=1;

#50 WE3=0;RA1=WA3;

end

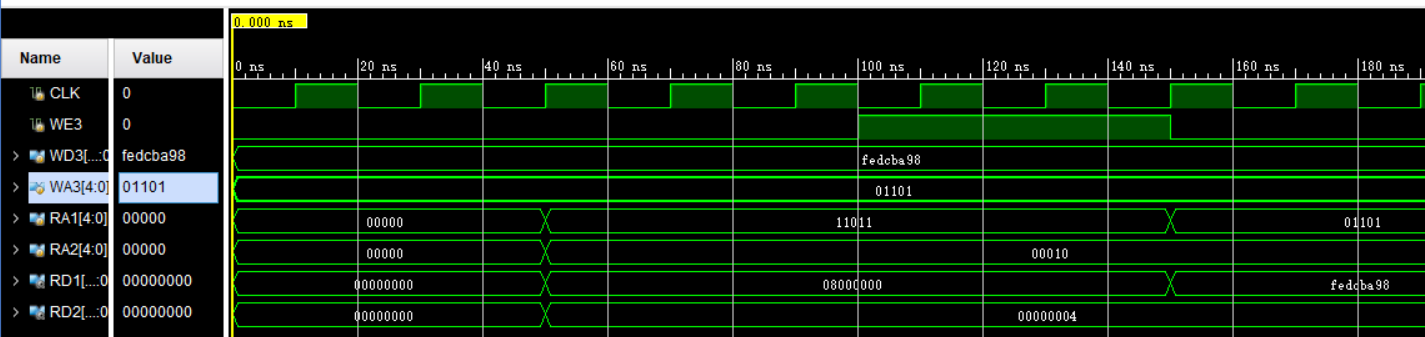
RegFile test(CLK,WE3,RA1,RA2,WA3,WD3,RD1,RD2);

Endmodule

初始化WE=0，RA1=RA2=0，因此会在第一个时钟上升沿读出零地址的寄存器，按照设计其值是0。WA=01101是要写入的地址，写入的值是16进制数FEDC\_BA98.

50ns后更改读地址分别为11011B（27）和00010B（2），观察输出数据；再过50ns，WE3=1，进行写操作，写信号持续50ns后收回，将第一个读地址RA1更改为刚刚写入的地址WE3，观察输出是否与写入一致。

仿真波形结果：



这里读、写数据均转化为16进制数，可以看到读地址为0时读出数是0；为其他地址时，地址对应的位上是1，其他位是0，这和设计代码中初始化的效果一致；在对目标地址写入修改存储值后，读出的数也与写入一致，表明读、写功能都准确实现。

**五、调试和心得体会**

1 在设计测试存储器阵列的时候，由于存储器的容量通常比较大，而我们更关心的是存储器的读写和寻址功能的实现正确与否，对于存储值并不关心，因此相比随机生成一个文本文件读入存储阵列中作为初始化，采用一定的次序或者方法让存储值有一定的规律，可以让测试的结果更清晰，容易验证。

2 在设计测试代码时，要注意各信号变化的时序关系，同时尽可能控制输入信号的变化能依此验证设计模块代码的各个功能。