实验四 存储器模块的设计及应用

**一、实验目的**

1. 掌握Verilog语言框架、编程和调试方法。
2. 掌握Verilog中的存储器电路工作原理。
3. 掌握存储器的实际应用。
4. 掌握Logisim中存储模块的使用。

**二、实验内容**

1. 设计一个静态存储器RAM。
2. 建立存储器的访问所需要的各种信号。
3. 对存储单元0#,1#,3#,5#,7#,9#或0#，2#，4#，6#，246#进行读写操作。
4. 观察、记录和分析仿真波形。
5. 练习Logisim软件中存储模块的使用。

**三、实验要求**

1. 分析模块的结构，画出其流程图。
2. 画出模块的电路图。
3. 分析电路的仿真波形，标出关键的数值（或在Logisim中完成验证）。
4. 记录设计和调试过程。

**四、实验代码及结果**

Vivado实现

1）256\*32bit静态存储器RAM

代码：

module RAM\_256x32(Data, Addr, Rst, R\_W, CS, CLK);

parameter Addr\_Width = 8;

parameter Data\_Width = 32;

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

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

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

input Rst, R\_W, CS, 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 : 32'bz;

always @(\*) begin

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

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

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

4'b100: RAM[Addr] <= Data;

default: Data\_i = 32'bz;

endcase

end

endmodule

module sim\_RAM();

parameter Addr\_Width = 8;

parameter Data\_Width = 32;

reg Rst, R\_W, CS, CLK;

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

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

integer i;

initial begin

CLK <= 0; CS <=1; Rst <= 1; R\_W <= 1; Addr <= 0;

fork

forever #10 CLK = ~CLK;

#20 Rst <= 0;

#20 for(i = 0; i < 5; i = i+1) #10 Addr <= 2\*i+1;

#90 R\_W <= 0;

#80 for(i = 0; i < 3; i = i+1) #10 Addr <= 4\*i+1;

#150 R\_W <= 1;

#140 for(i = 0; i < 5; i = i+1) #10 Addr <= 2\*i+1;

join

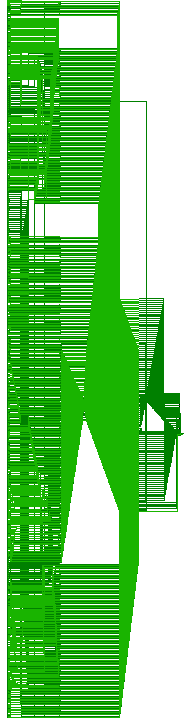
end

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

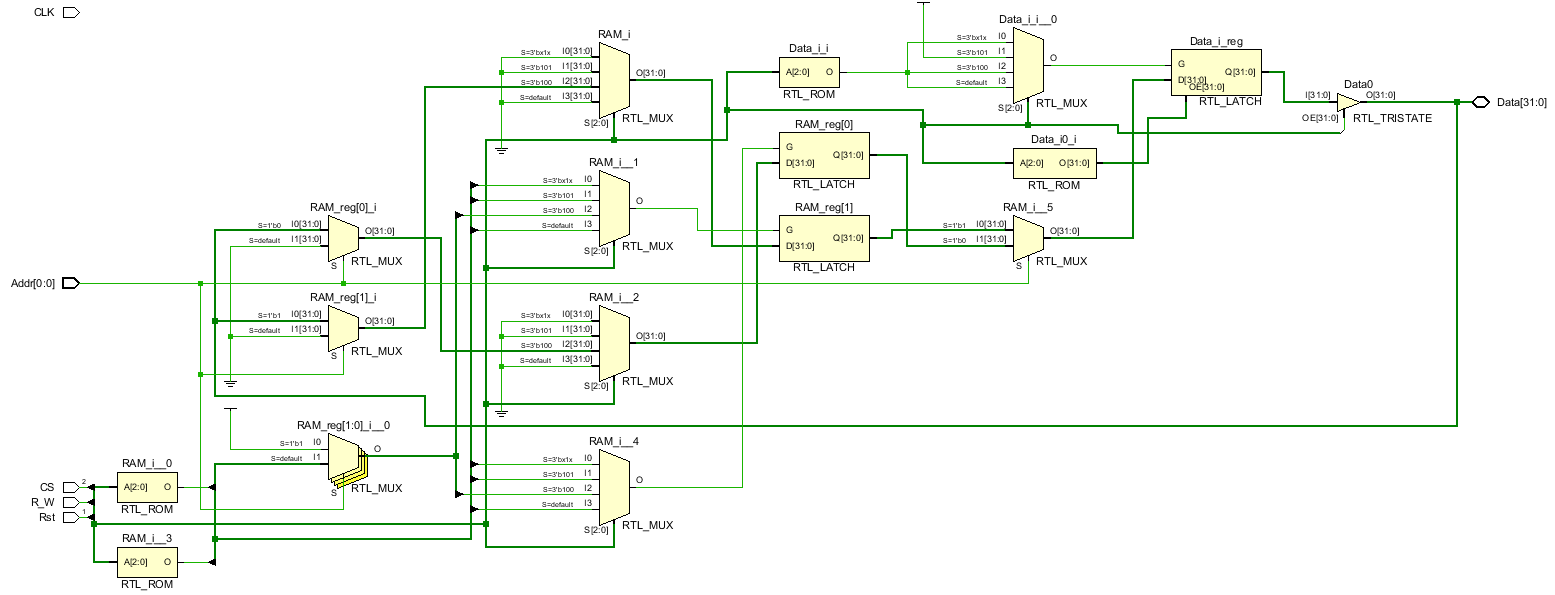
RAM\_256x32 RAM(Data, Addr, Rst, R\_W, CS, CLK);

endmodule

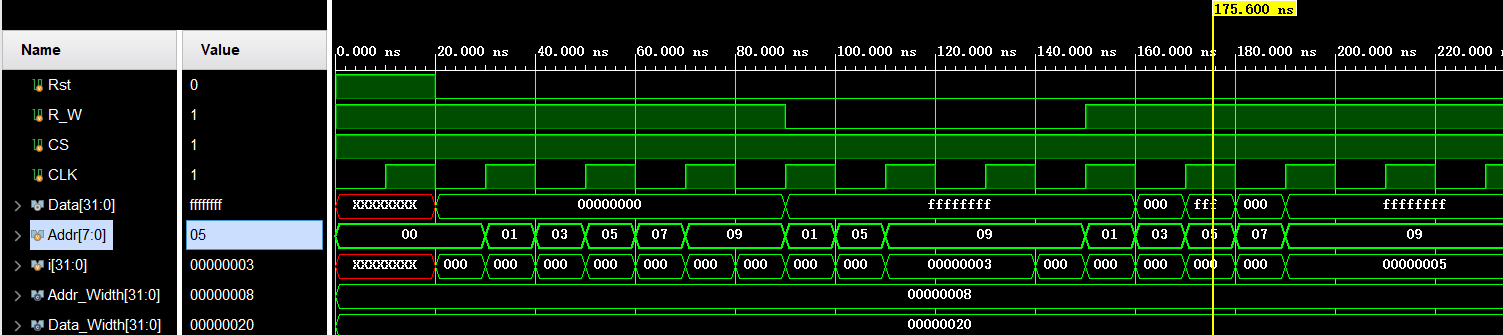
电路图：



由于256\*32bit存储器布线过于密集，为了更好地观察电路实现细节，可以参考2\*32bit存储器电路如下：



仿真波形：



0-20ns，Rst=1，对存储器中的所有存储单元清零；

20-90ns，R\_W=1，读取存储单元#0、#1、#3、#5、#7、#9，其值均为0；

90-150ns，R\_W=0，将值0xffff\_ffff写入存储单元#1、#5、#9；

150ns之后，R\_W=1，执行读操作，存储单元#1、#5、#9值为0xffff\_ffff，存储单元#3、#7值为0。

2）寄存器组

代码：

`define DATA\_WIDTH 32

module RegFile

#(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];

always @(posedge CLK)

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

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

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

endmodule

module sim\_RF();

parameter DATA\_WIDTH=32;

parameter ADDR\_SIZE=5;

reg CLK, WE3;

reg [DATA\_WIDTH-1:0] WD3;

reg [ADDR\_SIZE-1:0] RA1, RA2, WA3;

wire [DATA\_WIDTH-1:0] RD1, RD2;

initial begin

CLK = 0; WE3 = 1; WA3 = 1; WD3 = 2; RA1 = 0; RA2 = 0;

fork

forever #10 CLK = ~CLK;

#20 WA3 = 3;

#20 WD3 = 7;

#40 WE3 = 0;

#40 RA1 = 1;

#40 RA2 = 3;

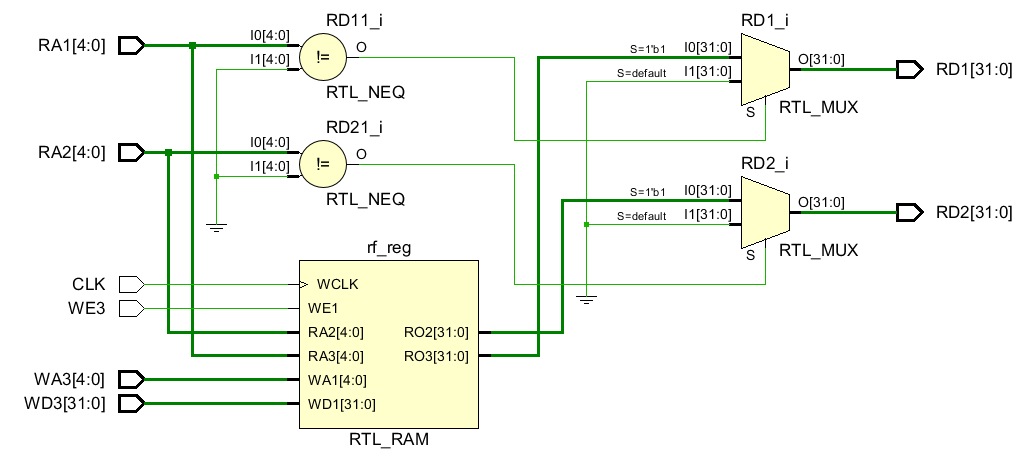
join

end

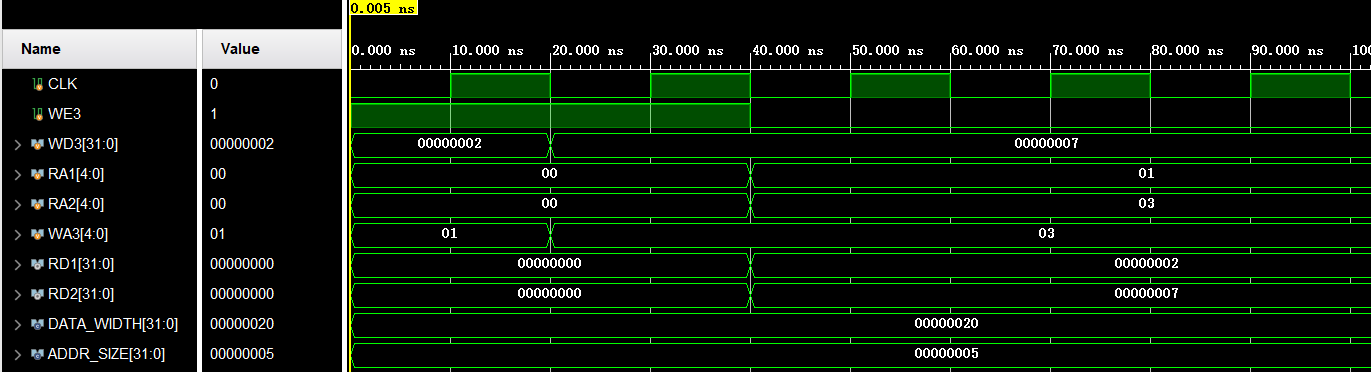
RegFile rf(CLK, WE3, RA1, RA2, WA3, WD3, RD1, RD2);

endmodule

电路图：



仿真波形：



0-40ns，WE3=1，执行写操作。0-20ns，WA3=1，WD3=2，向存储单元#1写入值2；20ns后，WA3=3，WD3=7，向存储单元#3写入值7；

40ns后，WE3=0，执行读操作。RA1=1，RD1=2，从存储单元#1读出值2；RA2=3，RD2=7，从存储单元#3读出值7。

3）指令存储器

代码：

`define DATA\_WIDTH 16

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("D:/Desktop/test.txt", RAM);

assign RD = RAM[A];

endmodule

module sim\_IMem();

parameter DATA\_WIDTH=16;

reg [5:0] A;

wire [DATA\_WIDTH-1:0] RD;

integer i;

initial begin

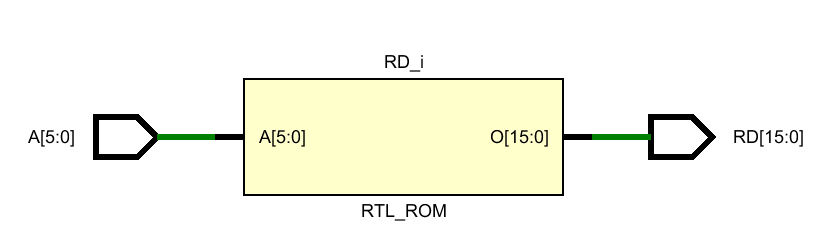
for(i = 0; i < 64; i = i+1) #10 A = i;

end

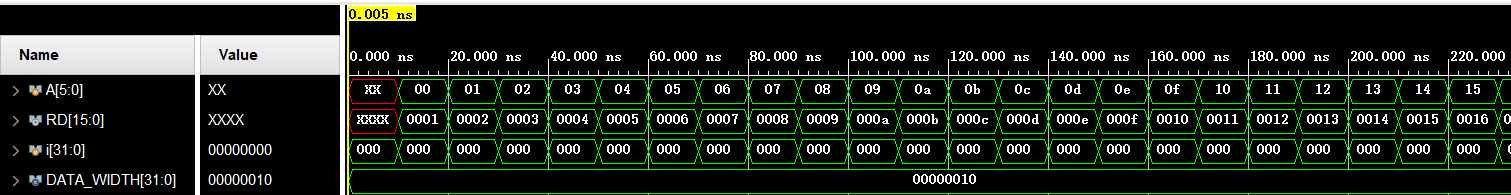
IMem test(A, RD);

endmodule

电路图：



仿真波形：



文件test.txt的内容为1-64共64个整数，并依次存入指令存储器中。

从存储单元#0读出值0001，#1读出值0002，#2读出值0003……

**五、调试和心得体会**

调试过程：系统函数$readmemh中需要使用反斜杠‘/’来表示目录层级。

心得体会：通过本次实验，我了解了Verilog中的存储器电路的工作原理，并进一步掌握了Verilog语言，同时也学会了系统函数$readmemh的使用方法以及在使用Vivado进行仿真时如何调试。