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，容量为256\*32bit。
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#进行读写操作。
4. 设计一个寄存器组，10\*32bit。
5. 观察、记录和分析仿真波形（Verilog）。
6. 测试电路输入输出正确性（Logisim）。

**三、实验要求**

1. 分析模块的结构，画出其流程图。
   1. RAM容量为256\*32bit

256 = 28，因此需要地址线的位宽为8，存储在Addr中。存储位长设计为32bit。

Rst设为复位信号，当Rst为低电位时，不论其他电位，将RAM存储单元全部置0。

R\_W控制读写信号，CS为使能信号。Data即作为数据的读入口，也作为输出口。

流程图为：

将Rst置为低电位，将RAM存储内容全置0

将R\_W置为高电位，会按Addr将对应存储单元内容读入Data中

将R\_W置为低电位，将Data的内容按Addr写入对应存储单元中

* 1. 寄存器组容量为10\*32bit。

23=8<10<24=16，因此需要地址线的位宽为4，存储在Addr中。存储位长设计为32bit。

所对应端口依次如下图：



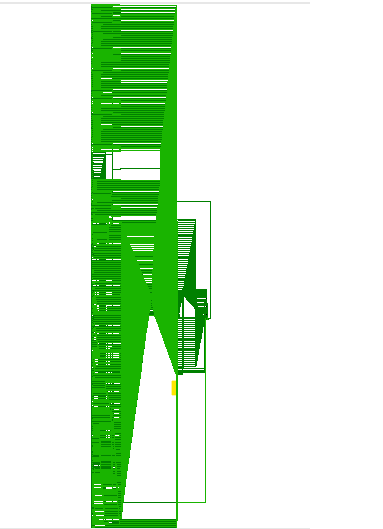
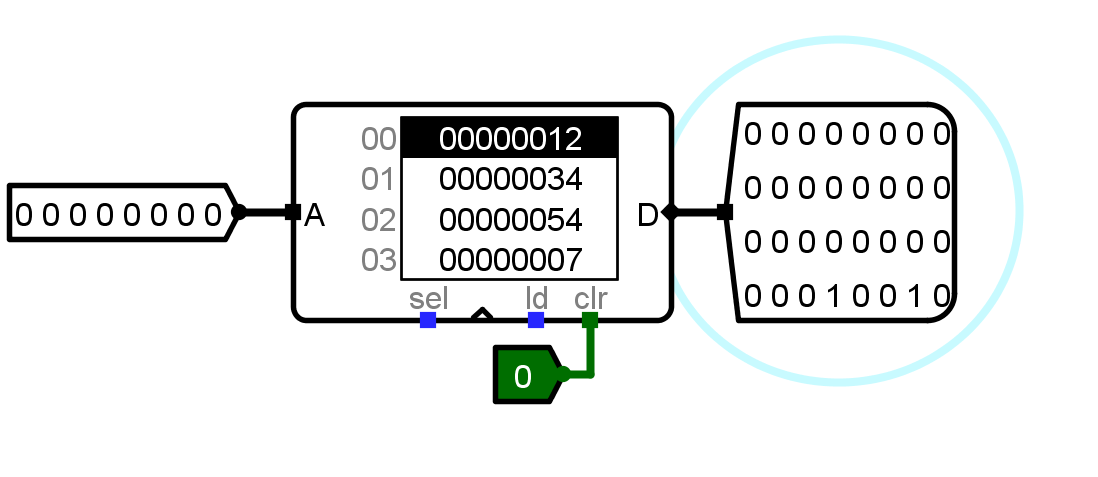
流程图为：

CLK时钟信号上升

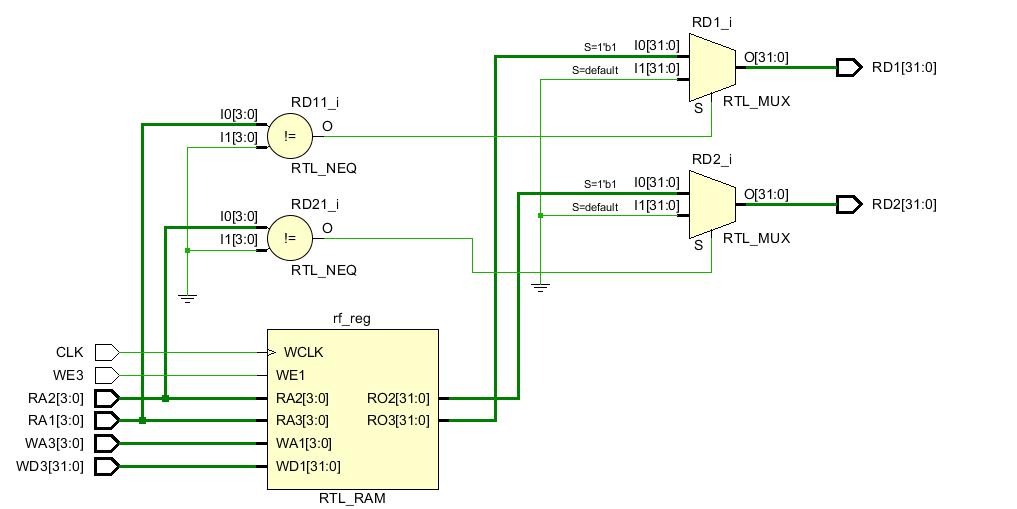
按WA内容将WD内容写入存储单元

RD按照对应的RA的地址对应的存储单元内读出内容

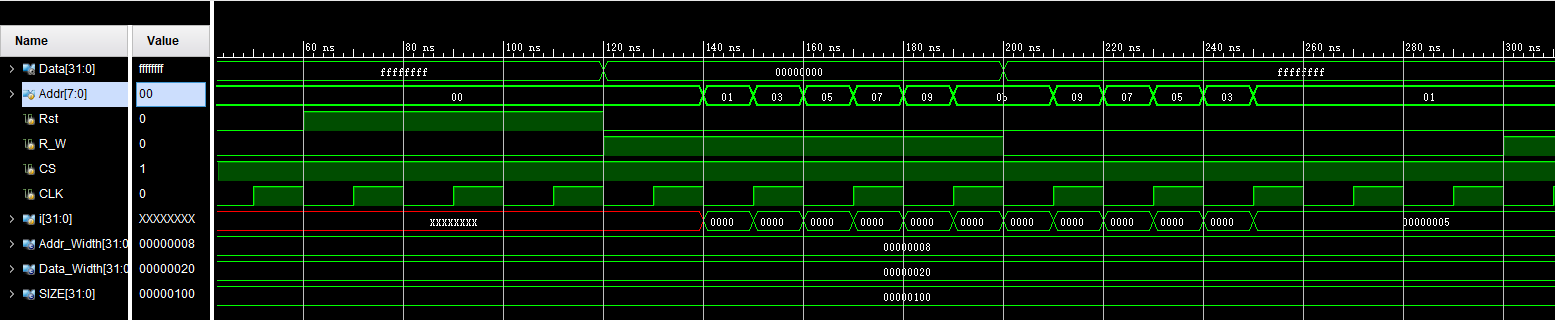
1. 画出模块的电路图。
   1. RAM容量为256\*32bit

* 1. 寄存器组容量为10\*32bit



1. 分析电路的仿真波形，标出关键的数值（或在Logisim中完成验证）。
   1. RAM容量为256\*32bit



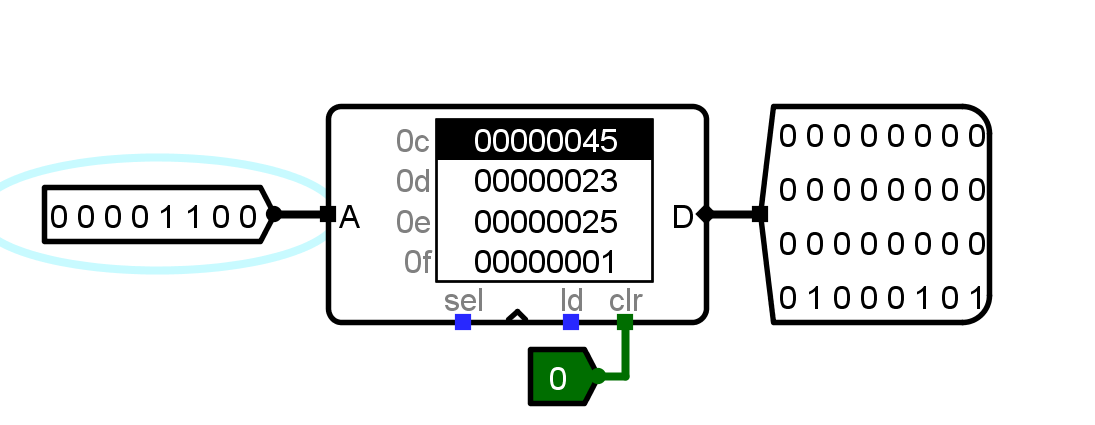
最开始未进行读写操作时，Data为ffffffff，Rst信号将RAM的存储单元全部清零，故随后将R\_W置高电平进行读操作时，从存储单元0#,1#,3#,5#,7#,9#中读出的值全为00000000（存于Data中）。随后将R\_W置低电平进行写操作时，Data清为ffffffff。

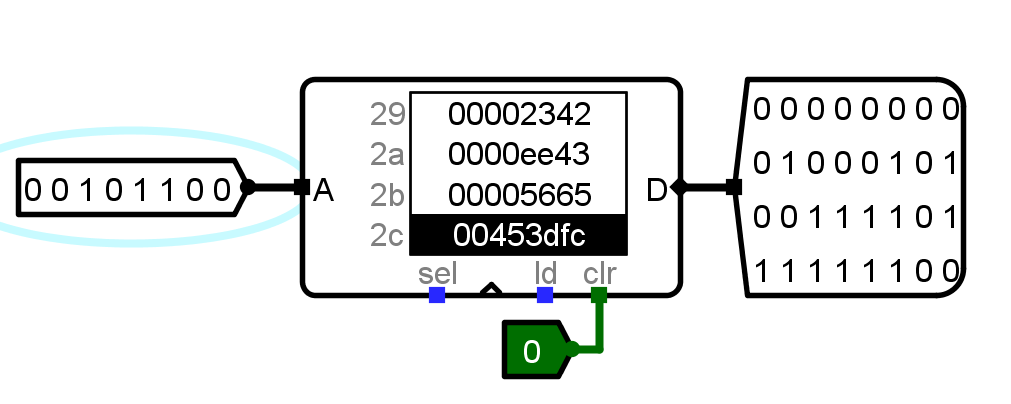
将Rst置为低电位，将RAM存储内容全置0

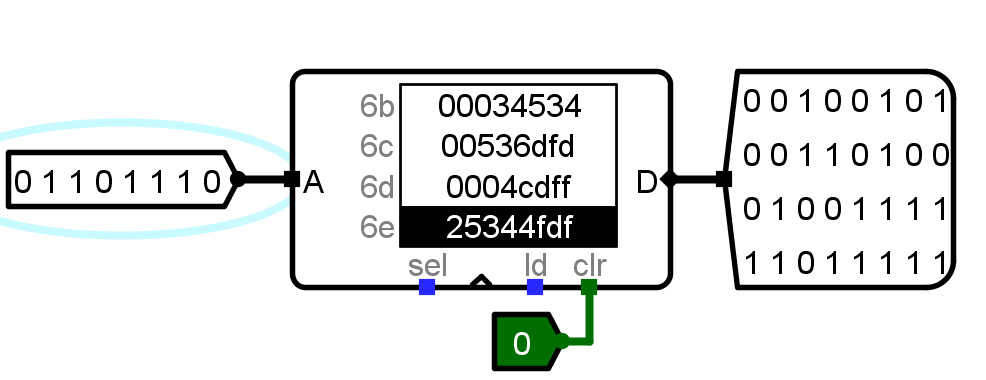
将R\_W置为高电位，依次将存储单元0#,1#,3#,5#,7#,9#中的内容读入Data中

将R\_W置为低电位，将Data的内容依次写入存储单元0#,1#,3#,5#,7#,9#中

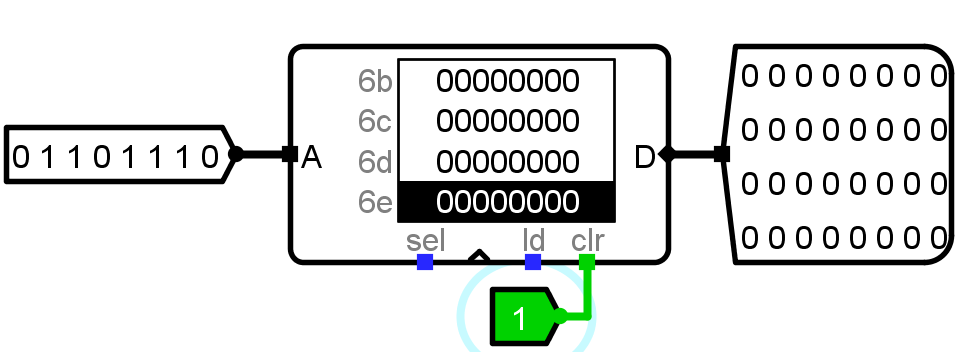
在Logisim中的RAM模块，可直接修改输入A的地址数值，模块会检索到存储单元，读取其中数字，且将存储单元设为一个数值后，会在输出D的读出的数值。多随机验证几个：



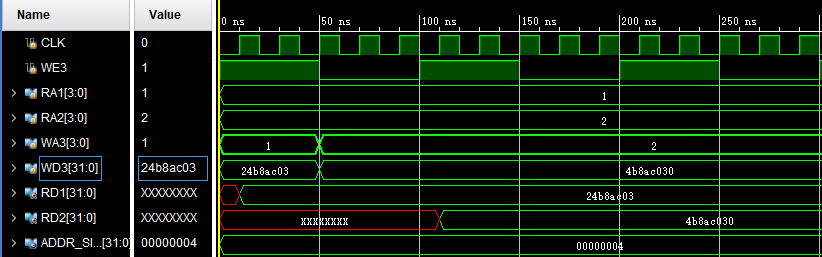




在CLK置1以后，RAM内容会全部清零。



* 1. 寄存器组容量为10\*32bit



将WA置为1，RA1置1，RA2置2。

按WA内容将WD内容写入RA1对应的存储单元，并读入到RD1

将WA内容置为2，将WD内容左移4位

按WA内容将WD内容写入RA2对应的存储单元，并读入到RD2

1. 记录设计和调试过程。

将地址线和数据位宽都设置为变量或者宏定义，方便设计时修改模块规模。

**四、实验代码及结果**

1. RAM容量为256\*32bit

RAM\_256x32bit.v

module RAM\_256x32bit(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];//读

4'b100: RAM[Addr] <= Data;//写

default: Data\_i = 32'bz;

endcase

end

endmodule

sim\_RAM\_256x32bit.v

module sim\_RAM\_256x32bit();

parameter Addr\_Width = 8;

parameter Data\_Width = 32;

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

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

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

reg Rst, R\_W, CS, CLK;

integer i;

initial begin

Rst = 0; R\_W = 0; CS = 1; CLK = 0;Addr = 8'h00;

fork

forever #10 CLK = ~CLK;

#60 Rst = ~Rst; #120 Rst = ~Rst;

#120 R\_W = ~R\_W; #200 R\_W = ~R\_W; #300 R\_W = ~R\_W;

begin

#140 Addr = 8'h01; for(i = 0;i < 5;i = i+1) #10 Addr = Addr + 2;

#10; for(i = 0;i < 5;i = i+1) #10 Addr = Addr - 2;

end

join

end

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

RAM\_256x32bit RAM\_256x32bit\_1(Data, Addr, Rst, R\_W, CS, CLK);

Endmodule

2. 寄存器组容量为10\*32bit

Reg\_File.v

`define DATA\_WIDTH 32

module Reg\_File

#(parameter ADDR\_SIZE = 4)

(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

sim\_Reg\_File.v

`define DATA\_WIDTH 32

module sim\_Reg\_File();

parameter ADDR\_SIZE = 4;

reg CLK, WE3;

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

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

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

initial begin

RA1 = 4'b0001; RA2 = 4'b0010;WA3 = 4'b0001;

WD3 = 32'h24b8ac03;

CLK = 0;WE3 = 1;

fork

forever #10 CLK = ~CLK;

forever #50 WE3 = ~WE3;

#50 WD3 = WD3 << 4;#50 WA3 = WA3 << 1;

join

end

Reg\_File Reg\_File\_1(CLK, WE3, RA1, RA2, WA3, WD3, RD1, RD2);

endmodule

**五、调试和心得体会**

最开始不清楚inout对应的仿真文件如何设置，并且由于仿真文件其实是通过实例化设计文件模块传递参数，最后的仿真波形中显示的是仿真文件中定义的变量，所以在仿真文件中定义RAM是看不到实际运行中RAM中的内容，因为RAM并不是从实例化设计文件模块中传递参数。