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。

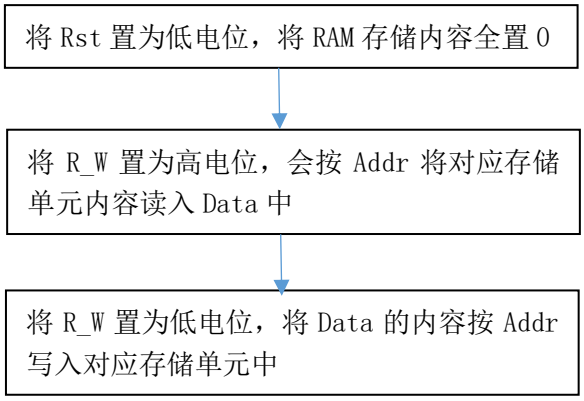
1.1 RAM 容量为 256*32bit

256 = 2^8，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：



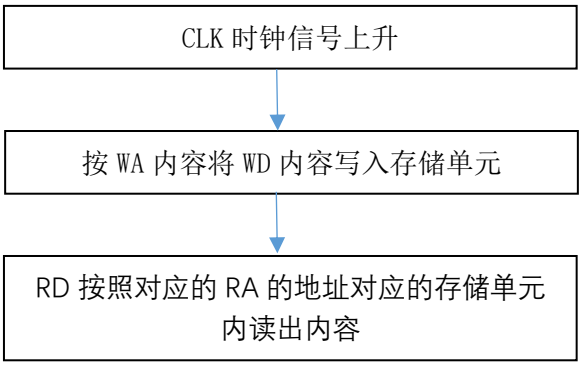
1.2 寄存器组容量为 10*32bit。

2^3=8<10<2^4=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：

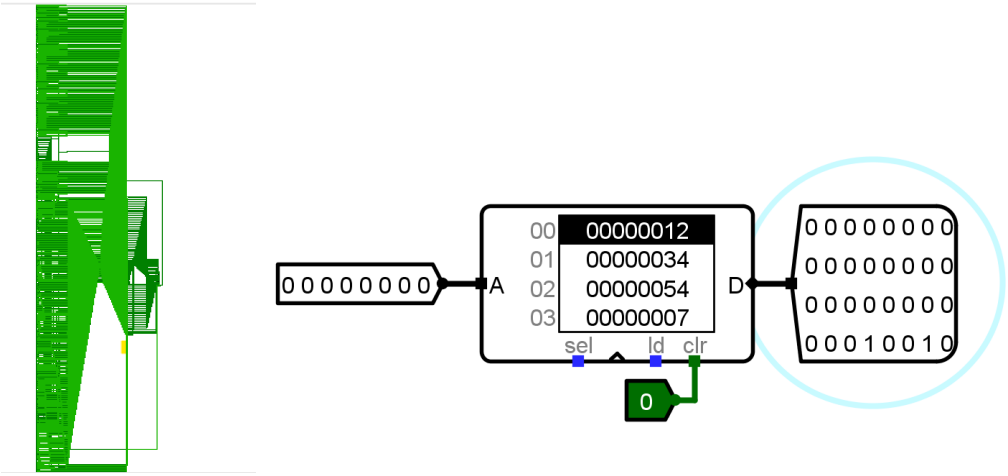
端口名	方向	描述
RA1[4:0]	I	RD1中数据的寄存器地址
RA2[4:0]	I	RD2中数据的寄存器地址
RD1[4:0]	O	RA1地址对应寄存器数值
RD2[4:0]	O	RA2地址对应寄存器数值
WA	I	写入数据的寄存器地址
WD	I	写入WA对应寄存器的数据
WE	I	写使能信号
clk	I	时钟信号

流程图为：

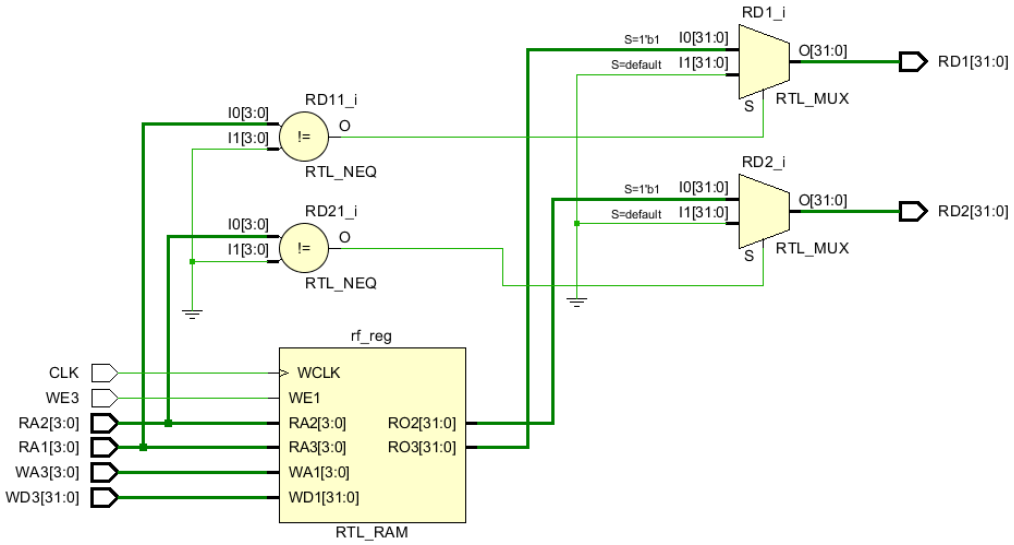


2. 画出模块的电路图。

2.1 RAM 容量为 256*32bit

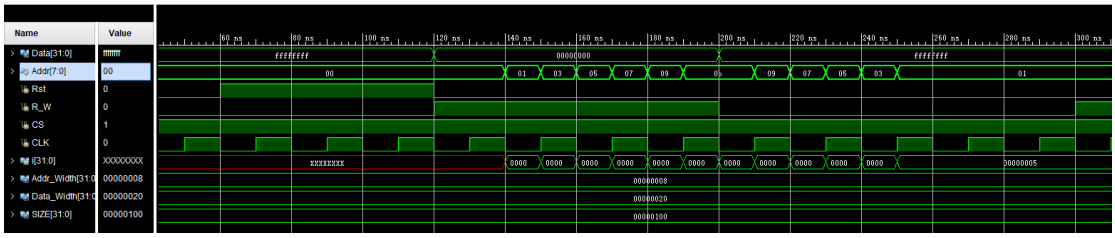


2.2 寄存器组容量为 10*32bit

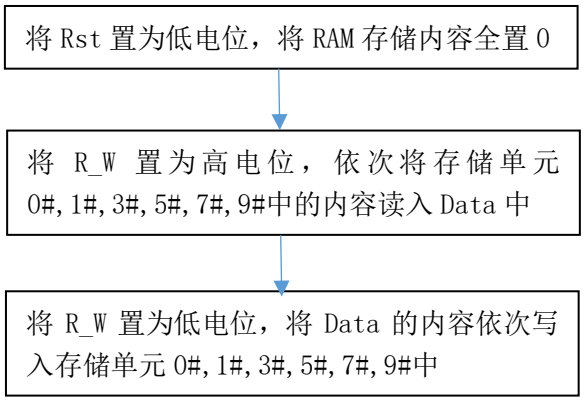


3. 分析电路的仿真波形，标出关键的数值（或在 Logisim 中完成验证）。

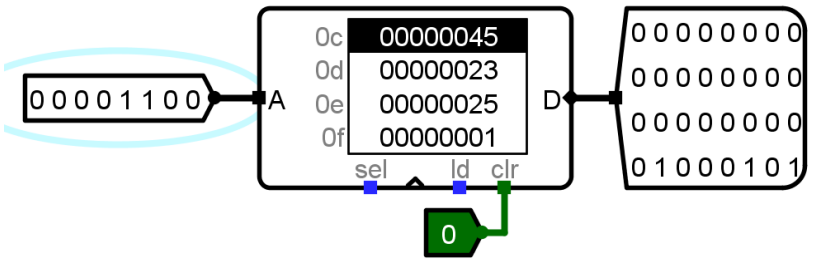
3.1 RAM 容量为 256*32bit

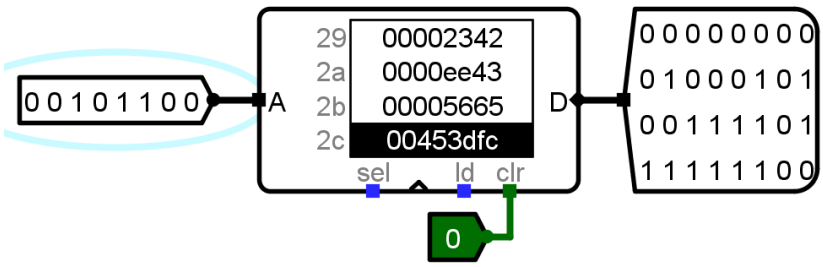


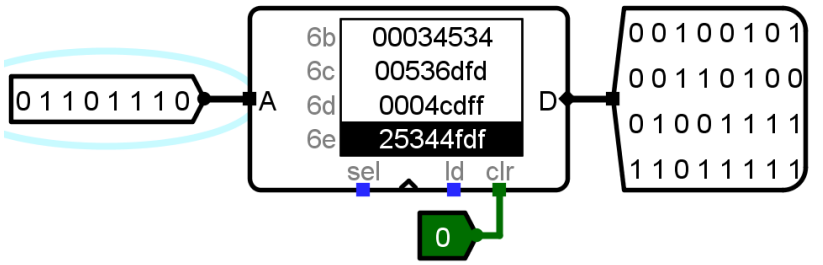
最开始未进行读写操作时，Data 为 ffffffff，Rst 信号将 RAM 的存储单元全部清零，故随后将 R_W 置高电平进行读操作时，从存储单元 0#,1#,3#,5#,7#,9#中读出的值全为 00000000（存于 Data 中）。随后将 R_W 置低电平进行写操作时，Data 清为 ffffffff。



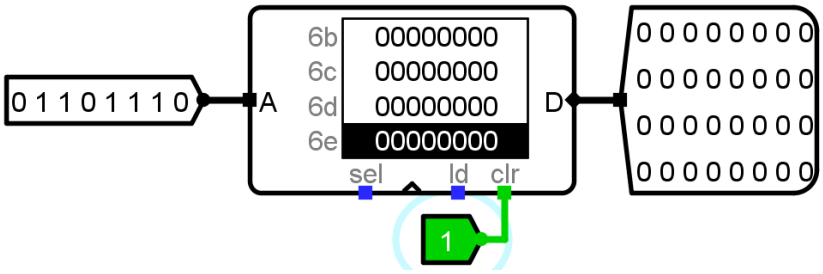
在 Logisim 中的 RAM 模块，可直接修改输入 A 的地址数值，模块会检索到存储单元，读取其中数字，且将存储单元设为一个数值后，会在输出 D 的读出的数值。多随机验证几个：



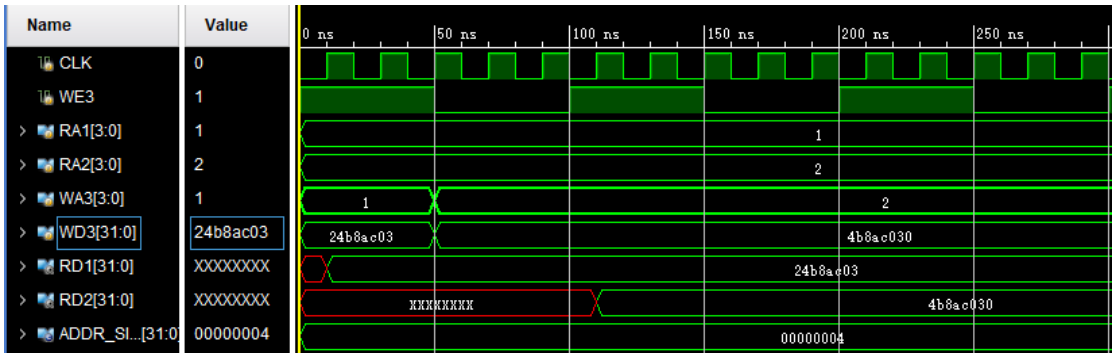


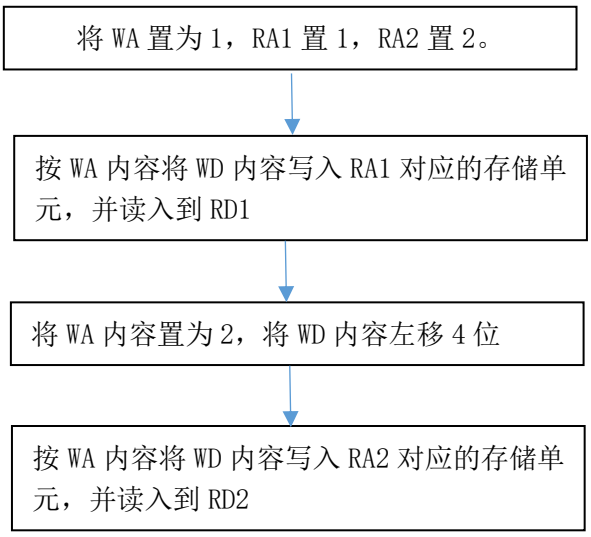


在 CLK 置 1 以后，RAM 内容会全部清零。



3.2 寄存器组容量为 10*32bit





4. 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。