# 同济大学计算机系

# 数字逻辑课程综合实验报告


学　　号 2352595

姓　　名 张嘉麟

专　　业 计算机科学与技术

授课老师 郭玉臣

# 一、实验内容

计数器：

计数器的功能是记忆脉冲的个数，它是数字系统中应用最广泛的基本时序逻辑构件。计数器所能记忆脉冲的最大数目称为该计数器的模。构成计数器的核心元件是触发器。本题中使用 logisim 画出同步模 8 电路原理图，实现由 3 个 JK 触发器组成的 3 位同步模 8 计数器，所有触发器的时钟都与同一个始终脉冲相连。本题通过实例化 JK 触发器与 7 段数码管来实现数字的显示；

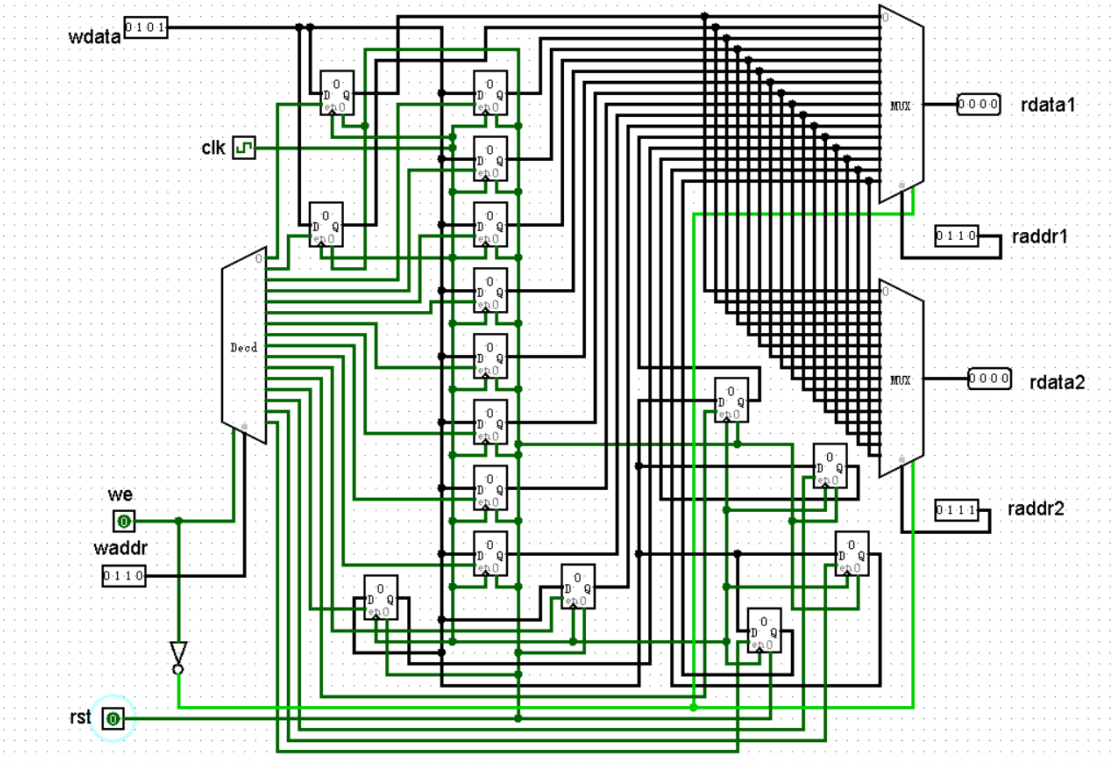
分频器：

每个计数器的脉冲输出频率等于其输入时钟频率除以计数模值，因此可以很容易地利用计数器由一个输入时钟信号获得分频后的时钟信号，这种应用称为分频。

# 二、硬件逻辑图

（实验步骤中要求用 logisim 画图的实验，在该部分给出 logisim 原理图，否则该部分在实验报告中不用写）

## （3）Regfiles 寄存器堆实验




# 三、模块建模

（该部分要求对实验中建模的所有模块进行功能描述，并列出各模块建模的

verilog 代码）

## （1）Ram 实验（双端口）

```
module ram(
    input clk,
    input ena,
    input wena,
    input [4:0] addr,
    input [31:0] data_in,
    output [31:0] data_out
    );
    reg [31:0] RAM [31:0];
    always @(posedge clk) begin
        if (ena && wena) begin
            RAM[addr] <= data_in; // 仅在写使能下写入数据
        end
    end
    assign data_out = (ena && !wena) ? RAM[addr] : 32'bz;
endmodule
```

## （2）Ram 实验（单端口）

```
module ram2(
    input clk,
    input ena,
    input wena,
    input [4:0] addr,
    inout [31:0] data
);

    reg[31:0] RAM [31:0];

    reg [31:0] inner_data;
    assign data = wena ? 32'bz : inner_data;

    always @(posedge clk)
    if(ena)
    begin
        if(wena)
            RAM[addr] <= data;
        else
            inner_data <= RAM[addr];
    end
    else
```

```
    begin
        inner_data = 32'bz;
    end

endmodule
```

## （3）Regfiles 寄存器堆实验

```
module Regfiles(
    input clk,
    input rst,
    input we,

    input [4:0] raddr1,
    input [4:0] raddr2,
    input [4:0] waddr,
    input [31:0] wdata,
    output [31:0] rdata1,
    output [31:0] rdata2
);

    wire [31:0] decoder_out;
    wire [31:0] regfile_out [31:0];

    decoder decoder_inst(waddr, we, decoder_out);

    genvar i;
    generate
        for(i = 0; i < 32; i = i + 1)
            pcreg pcreg_inst(clk, rst, decoder_out[i], wdata, regfile_out[i]);
    endgenerate

    selector32_1 selector32_1_inst(regfile_out[0],regfile_out[1],regfile_out[2],regfile_out[3],
    regfile_out[4],regfile_out[5],regfile_out[6],regfile_out[7],
    regfile_out[8],regfile_out[9],regfile_out[10],regfile_out[11],
    regfile_out[12],regfile_out[13],regfile_out[14],regfile_out[15],
    regfile_out[16],regfile_out[17],regfile_out[18],regfile_out[19],
    regfile_out[20],regfile_out[21],regfile_out[22],regfile_out[23],
    regfile_out[24],regfile_out[25],regfile_out[26],regfile_out[27],
    regfile_out[28],regfile_out[29],regfile_out[30],regfile_out[31], raddr1, rdata1, ~we);

    selector32_1 selector32_1_inst2(regfile_out[0],regfile_out[1],regfile_out[2],regfile_out[3],
    regfile_out[4],regfile_out[5],regfile_out[6],regfile_out[7],
    regfile_out[8],regfile_out[9],regfile_out[10],regfile_out[11],
    regfile_out[12],regfile_out[13],regfile_out[14],regfile_out[15],
```

```
        regfile_out[16],regfile_out[17],regfile_out[18],regfile_out[19],
        regfile_out[20],regfile_out[21],regfile_out[22],regfile_out[23],
        regfile_out[24],regfile_out[25],regfile_out[26],regfile_out[27],
        regfile_out[28],regfile_out[29],regfile_out[30],regfile_out[31], raddr2, rdata2, ~we);

endmodule
```

```
module selector32_1(
        input [31:0] iC0,
        input [31:0] iC1,
        input [31:0] iC2,
        input [31:0] iC3,
        input [31:0] iC4,
        input [31:0] iC5,
        input [31:0] iC6,
        input [31:0] iC7,
        input [31:0] iC8,
        input [31:0] iC9,
        input [31:0] iC10,
        input [31:0] iC11,
        input [31:0] iC12,
        input [31:0] iC13,
        input [31:0] iC14,
        input [31:0] iC15,
        input [31:0] iC16,
        input [31:0] iC17,
        input [31:0] iC18,
        input [31:0] iC19,
        input [31:0] iC20,
        input [31:0] iC21,
        input [31:0] iC22,
        input [31:0] iC23,
        input [31:0] iC24,
        input [31:0] iC25,
        input [31:0] iC26,
        input [31:0] iC27,
        input [31:0] iC28,
        input [31:0] iC29,
        input [31:0] iC30,
        input [31:0] iC31,
        input [4:0] iS,
        output reg [31:0] oZ,
        input ena
        );
```

```
    always @(*)
    begin
        if(ena)
        begin
            case(iS)
            5'd0 : oZ = iC0;
            5'd1 : oZ = iC1;
            5'd2 : oZ = iC2;
            5'd3 : oZ = iC3;
            5'd4 : oZ = iC4;
            5'd5 : oZ = iC5;
            5'd6 : oZ = iC6;
            5'd7 : oZ = iC7;
            5'd8 : oZ = iC8;
            5'd9 : oZ = iC9;
            5'd10: oZ = iC10;
            5'd11: oZ = iC11;
            5'd12: oZ = iC12;
            5'd13: oZ = iC13;
            5'd14: oZ = iC14;
            5'd15: oZ = iC15;
            5'd16: oZ = iC16;
            5'd17: oZ = iC17;
            5'd18: oZ = iC18;
            5'd19: oZ = iC19;
            5'd20: oZ = iC20;
            5'd21: oZ = iC21;
            5'd22: oZ = iC22;
            5'd23: oZ = iC23;
            5'd24: oZ = iC24;
            5'd25: oZ = iC25;
            5'd26: oZ = iC26;
            5'd27: oZ = iC27;
            5'd28: oZ = iC28;
            5'd29: oZ = iC29;
            5'd30: oZ = iC30;
            5'd31: oZ = iC31;
            endcase
        end
        else
            oZ = 32'bz;
    end
endmodule
```

```
module decoder(
	input [4:0] iData,
	input iEna,
	output [31:0] oData
);

	assign oData = (iEna ? (1 << iData) : 32'b0);

endmodule
```

```
module pcreg(
	input clk,
	input rst,
	input ena,
	input[31:0] data_in,
	output reg[31:0] data_out
	);

	always@ (posedge clk or posedge rst)
	begin
		if(rst)
			data_out <= 0;
		else if(ena)
			data_out <= data_in;
	end

endmodule
```

# 四、测试模块建模

（要求列写各建模模块的 test bench 模块代码）

## （1）Ram 实验（双端口）

```
	module ram_tb;
	reg clk;
	reg ena;
	reg wena;
	reg [4:0] addr;
	reg [31:0] data_in;
	wire [31:0] data_out;

	// 实例化 RAM 模块
	ram uut (
```

```
        .clk(clk),
        .ena(ena),
        .wena(wena),
        .addr(addr),
        .data_in(data_in),
        .data_out(data_out)
);

// 时钟信号生成
initial begin
        clk = 0;
        forever #5 clk = ~clk; // 10ns 时钟周期
end

// 测试序列
initial begin
        // 初始化信号
        ena = 0;
        wena = 0;
        addr = 0;
        data_in = 0;

        // 写操作测试
        #100;
        ena = 1;
        wena = 1;
        addr = 5'd0;
        data_in = 32'h12345678;
        #10;

        // 读操作测试
        wena = 0;
        #10;

        ena = 1;
        wena = 1;
        addr = 5'd0;
        data_in = 32'h87654321;
        #10;

        // 读操作测试
        wena = 0;
        #10;
        ena = 1;
```

```
        wena = 0;
        data_in = 32'h77777777;
        addr = 5'b00000;
        #10 addr = 5'b00001;
        #10 addr = 5'b00010;
        #10 addr = 5'b00011;
        #10 addr = 5'b00100;
        #10 addr = 5'b00101;
        #10 addr = 5'b00110;
        #10 addr = 5'b00111;
        #10 addr = 5'b01000;
        #10;
        $display("Read data at address 0: %h", data_out); // 应输出 12345678

        // 结束测试
        #100 $finish;
    end
endmodule
```

## （2）Ram 实验（单端口）

```
module ram2_tb();
    reg clk, ena, wena;
    reg [4:0] addr;
    wire [31:0] data;
    reg [31:0] data_in;
    reg wr;

    initial
    begin
        clk = 0;
        forever
        begin
            clk = #5 ~clk;
        end
    end

    assign data = wr ? data_in : 'bz;

    initial
    begin
        ena = 1;
        wena = 1;
        wr = 1;
        data_in = 127;
```

```
        for(addr = 0; addr < 31; addr = addr + 1)
        begin
            data_in = data_in + 1;
            #10;
        end

        wena = 0;
        wr = 0;
        for(addr = 0; addr < 31; addr = addr + 1)
        begin
            #10;
        end

        ena = 0;
    end

    ram2 ram2_inst(clk, ena, wena, addr, data);

endmodule
```

### （3）Regfiles 寄存器堆实验

```
module Regfiles_tb;
    reg clk;
    reg rst;
    reg we;
    reg [4:0] raddr1;
    reg [4:0] raddr2;
    reg [4:0] waddr;
    reg [31:0] wdata;
    wire [31:0] rdata1;
    wire [31:0] rdata2;
    initial
    begin
        rst = 0;
        clk = 0;
        forever
        begin
            clk = #5 ~clk;
        end
    end
    initial
    begin
```

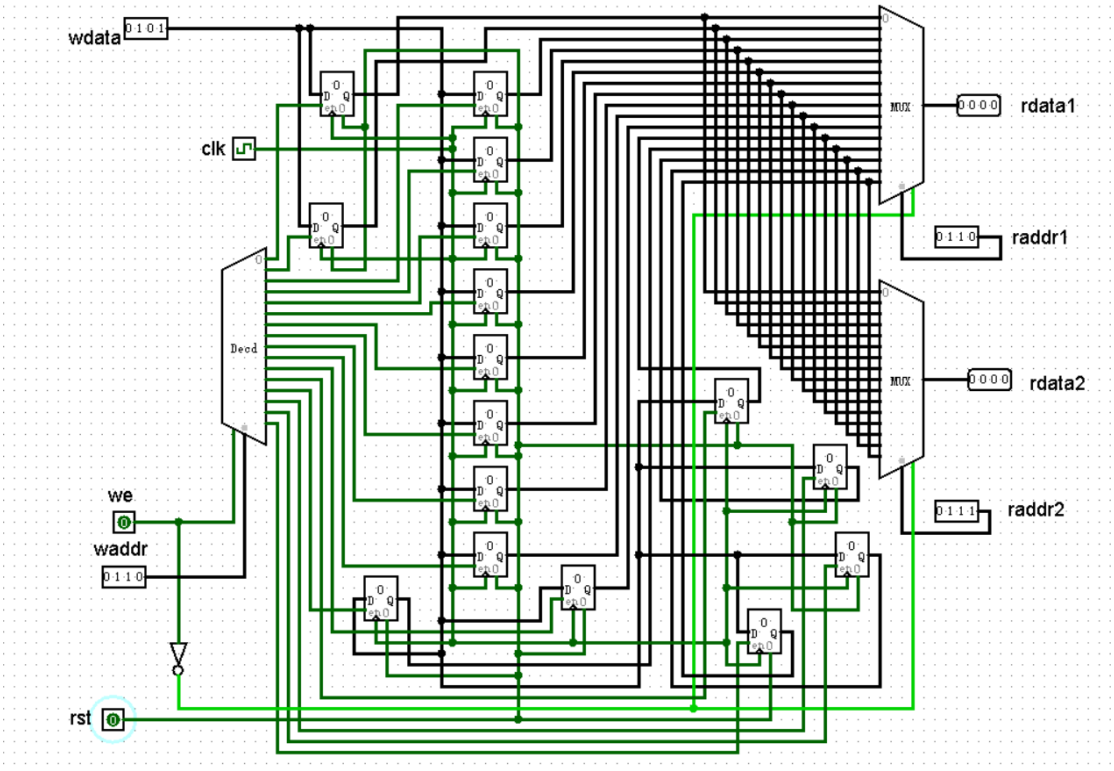
```
        we = 1;
        wdata = 32'b0;
        for(waddr = 0; waddr < 31; waddr = waddr + 1)
        begin
            wdata = wdata + 1;
            #10;
        end
        we = 0;
        for(raddr1 = 0; raddr1 < 31; raddr1 = raddr1 + 1)
        begin
            raddr2 = 30 - raddr1;
            #10;
        end
    end
    Regfiles Regfiles_inst(clk, rst, we, raddr1, raddr2, waddr, wdata, rdata1, rdata2);
endmodule
```

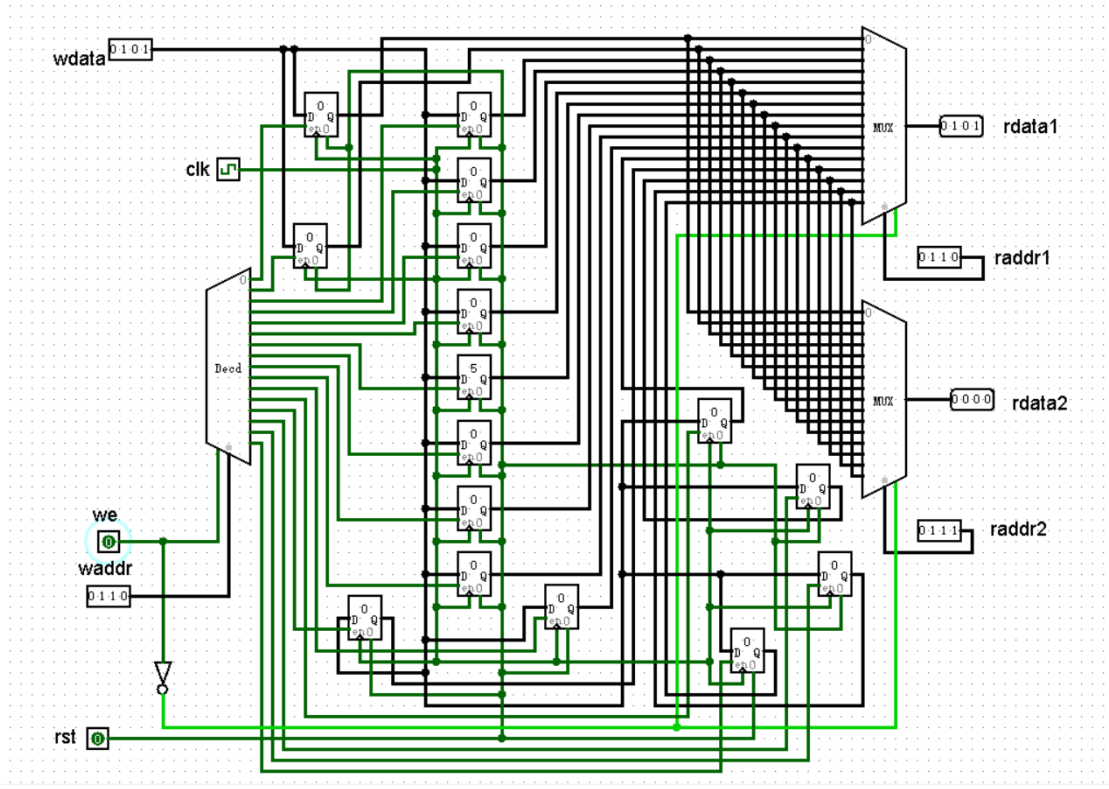
# 五、实验结果

（该部分可截图说明，要求 logisim 逻辑验证图、modelsim 仿真波形图、以及下板后的实验结果贴图（实验步骤中没有下板要求的实验，不需要下板贴图））

## 1. logisim 逻辑验证图

**Regfiles 寄存器堆实验**



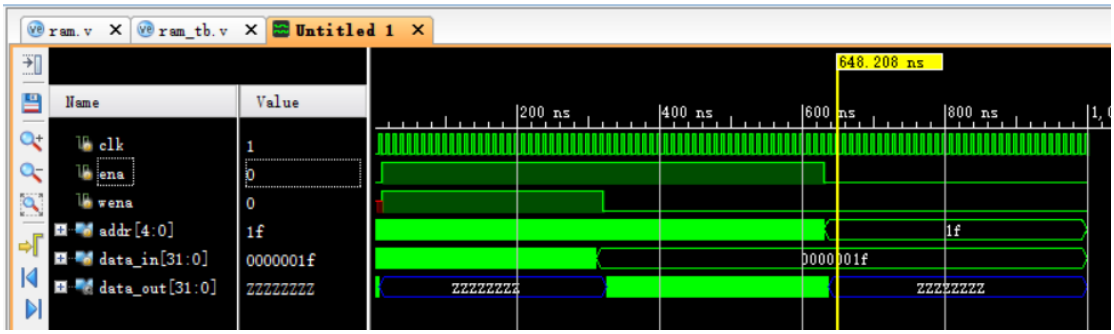


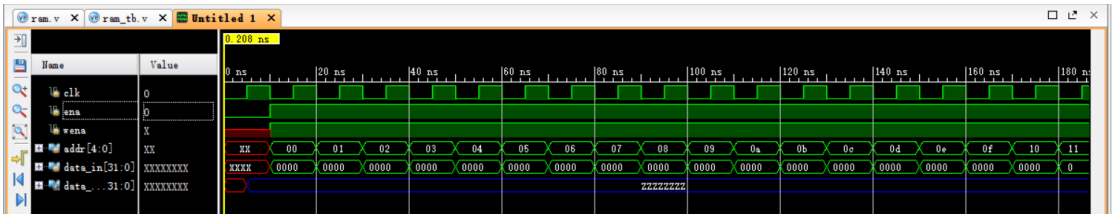

先将 rst 置 1，使得所有寄存器清零；然后将 rst 置 0 开始测试。先测试写入，将 we 置为高电平写有效。将 waddr 置为 0100，wdata 置为 0110，将 6 送入 4 号寄存器；将 waddr 置为 0110，wdata 置为 0101，将 5 送入 6 号寄存器；将 waddr 置为 1010，wdata 置为 0111，将 7 送入 10 号寄存器。然后测试读出，将 we 置为低电平读有效。将 raddr1、raddr2 分别置为 0100 和 1010，分别读出 4 号、10 号寄存器中的内容，rdata1 与 rdata2 的值分别为 0110、0111。

## 2. modelsim 仿真波形图

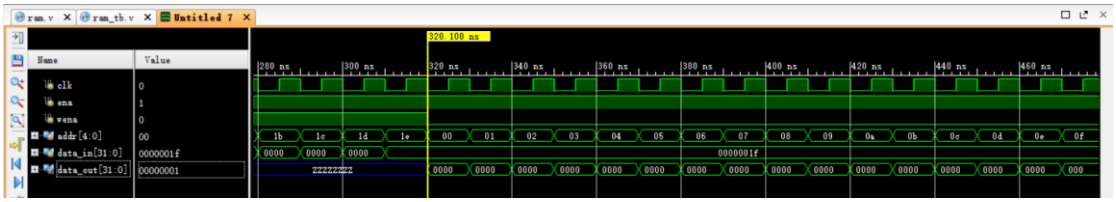
### （1）Ram 实验（双端口）

第一遍：问题在输出的时候会存在输出 zzzzzzz 的情况



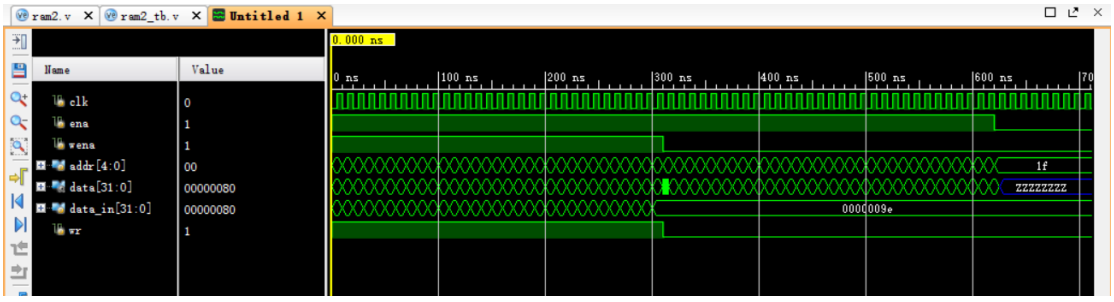



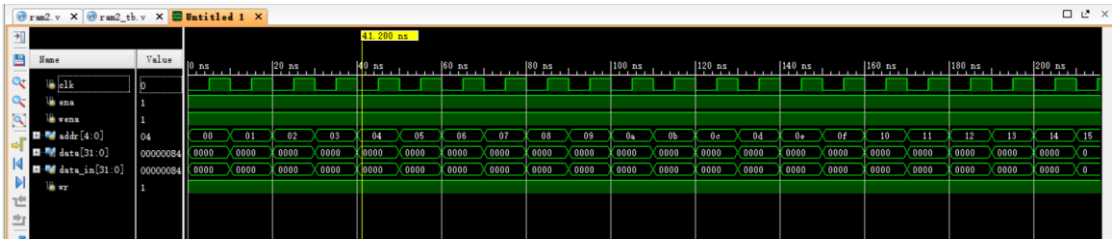
第二次更改好了对齐问题：



首先测试写入，ena 高电平有效，wena 高电平代表写有效：每当 CLK 上升沿到来时，从 data_in 读入数据到对应的 addr 地址处；然后测试读出，ena 高电平有效，wena 低电平代表读有效：每当 CLK 上升沿到来时，从对应的 addr 地址处读出数据到 data_out。

## （2）Ram 实验（单端口）

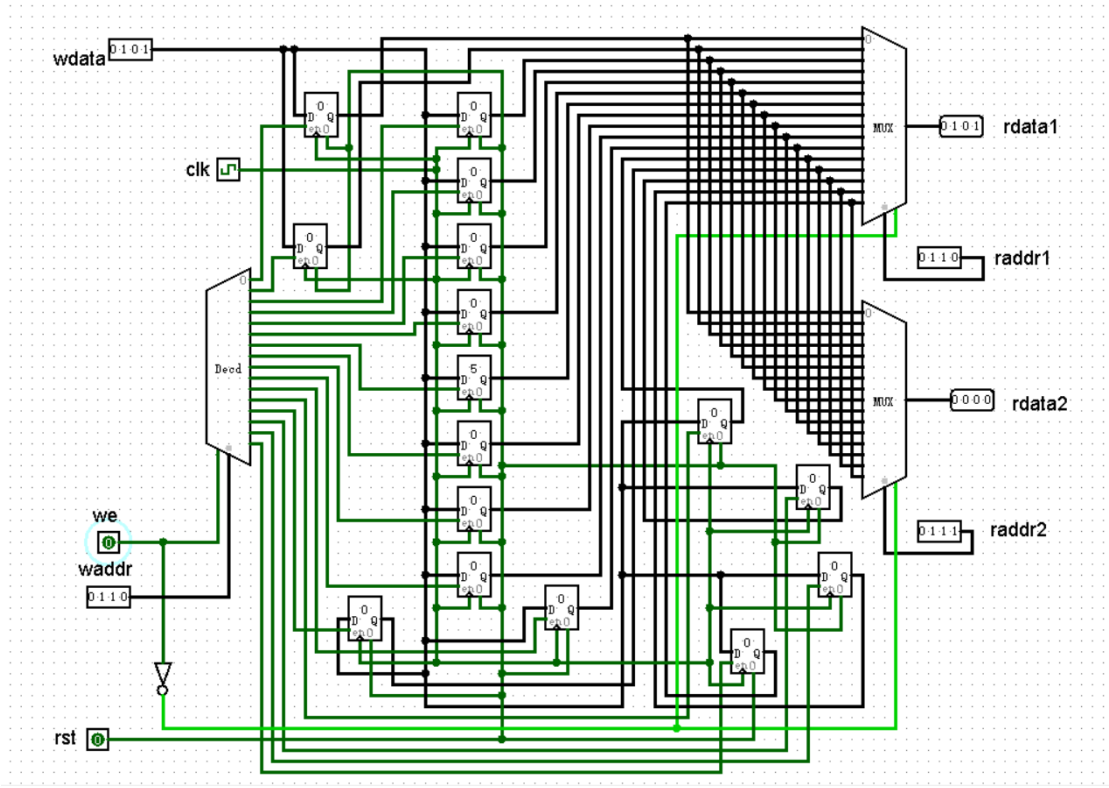


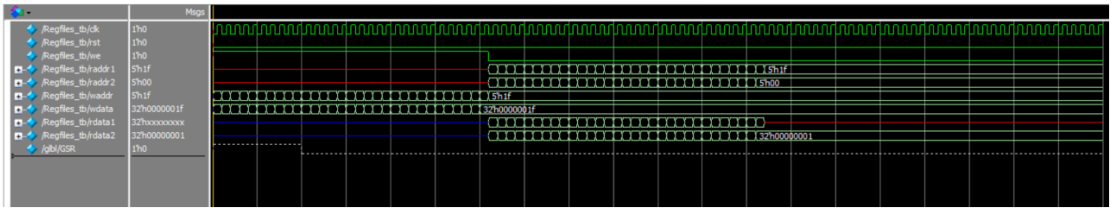
首先测试写入，ena 高电平有效，wena 高电平代表写有效：每当 CLK 上升沿到来时，从 data 读入数据到对应的 addr 地址处；然后测试读出，ena 高电平有

效，wena 低电平代表读有效：每当 CLK 上升沿到来时，从对应的 addr 地址处读出数据到 data。

## （3）Regfiles 寄存器堆实验




逻辑验证正确后：测试波形图



首先测试写入，we 高电平代表写有效：每当 CLK 上升沿到来时，从 wdata 读入数据到对应的 waddr 地址处；然后测试读出，we 低电平代表读有效：从对应的 raddr1、raddr2 地址处读出数据到 rdata1、rdata2。