一、　概念解释（10 分，　每小题 2 分）

1. false path：在电路中一种有物理连接的路径，但实际工作时信号不会沿此路径传播，这种路径称为 false path。
2. 环境约束：对电路工作的外在环境，包括电路的工作电压、工作温度以及工艺变化，以及电路与周边其它电路的关系，包括输入引脚的延时及驱动能力，输出引脚的负载及输出延时。
3. SDF：standard delay format，标准延时格式，是工业界都接受的一种延时描述格式。
4. 分布延时：将延时标注在电路中的每个门上的延时描述方式。
5. 上升延时：信号从 10%电平上升到 90%电平所需要的时间。

二、　（14 分）模块 bitwise 的 verilog 描述如下所示。请：

1. （8 分）找出其中的错误。
2. （6 分）请画出其电路图。

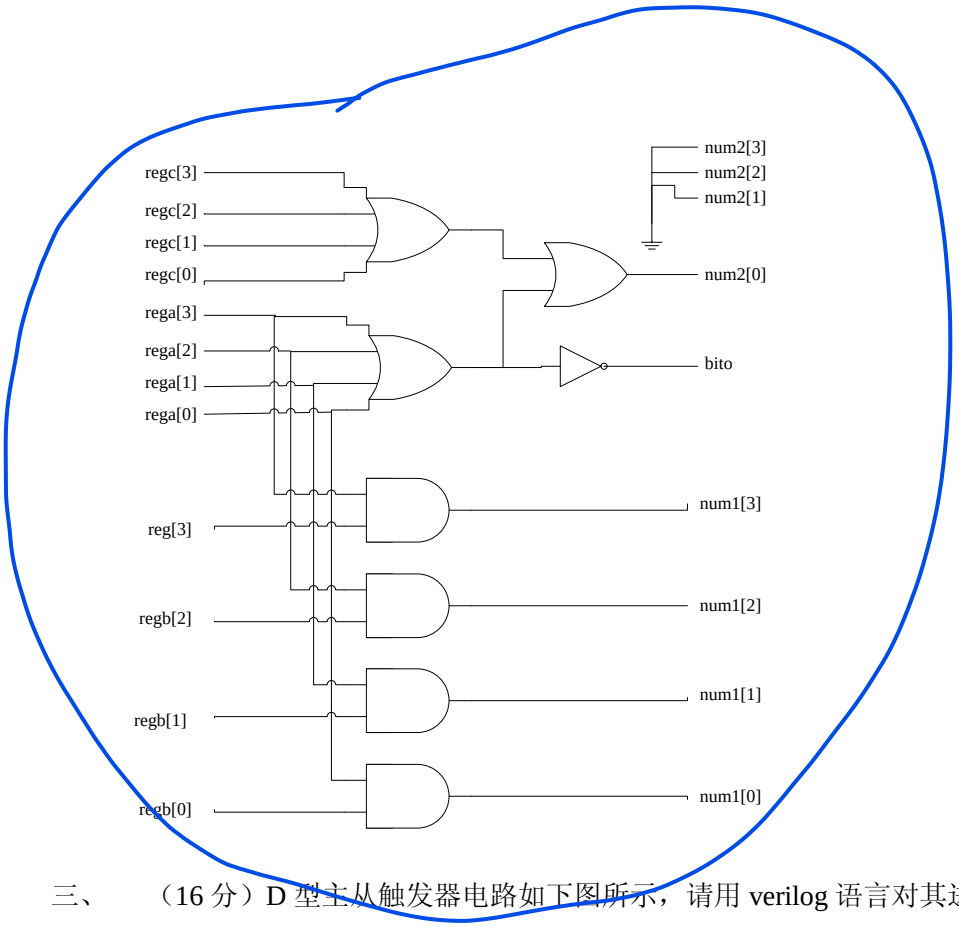
```
module bitwise (
        input    [3: 0] rega, regb, regc,
        output   [3: 0] num1, num2,
         output         bito,
    );
    assign bito =    !rega;
    always (rega, regb) begin
        num1 = rega & regb;
        num2 = rega || regc;
    end
endmodule
```

解答：

1、 错误有 4 处：

a) num1、num2 应为寄存器 reg 类型
b) bito 声明时，bito 后面不应该有逗号
c) always @(rega, regb)，少了@
d) always @(rega, regb)，敏感表不完全，应该再加上 regc，即 always @(rega, regb，regc)

2、 电路如下图所示。



三、 （16 分）D 型主从触发器电路如下图所示，请用 verilog 语言对其进行门级描述。

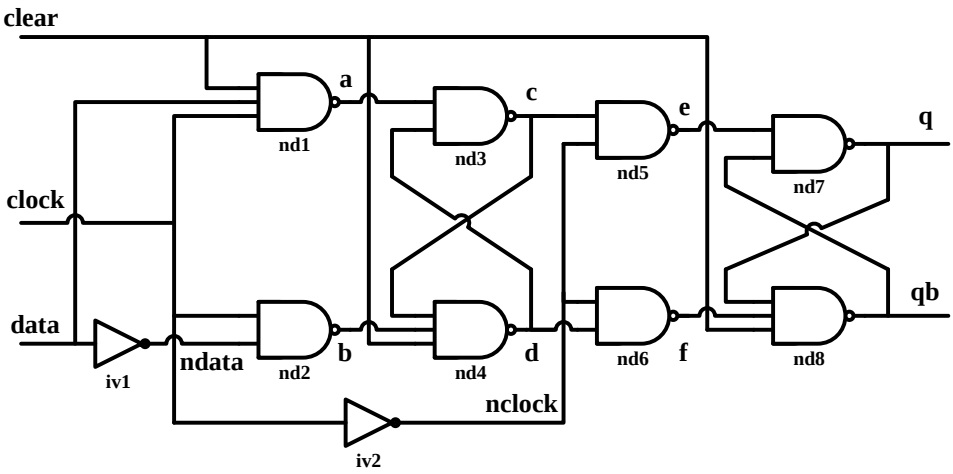


图 1　D 型主从触发器的电路结构图

解答：

```
module dff (
        input wire      clock,    clear,
                        data,
        output wire     q,    qb
              );
wire ndata, nclock;
wire a, b, c, d, e, f;
        not inv1(ndata, data),
              inv2(nclock, clock);
```

```
        nand    nd1(a, clear, data, clock),
                nd2(b, clock, ndata),
                nd3(c, a, b),
                nd4(d, c b, clear),
                nd5(e, c, nclock),
                nd6(f, nclock, d),
                nd7(q, e, qb),
                nd8(qb, q, f, clear);
endmodule
```

四、 （20 分）、某状态机 fsm 如下图所示，其中时钟信号 clk 的频率 100MHz，复位信号 clr 高电平有效。请：

1. （10 分）使用 verilog 语言描述该电路。

2. （10 分）编写该电路测试程序，要求使用 SHM 波形数据库记录复位结束后至少 100ns 波形。

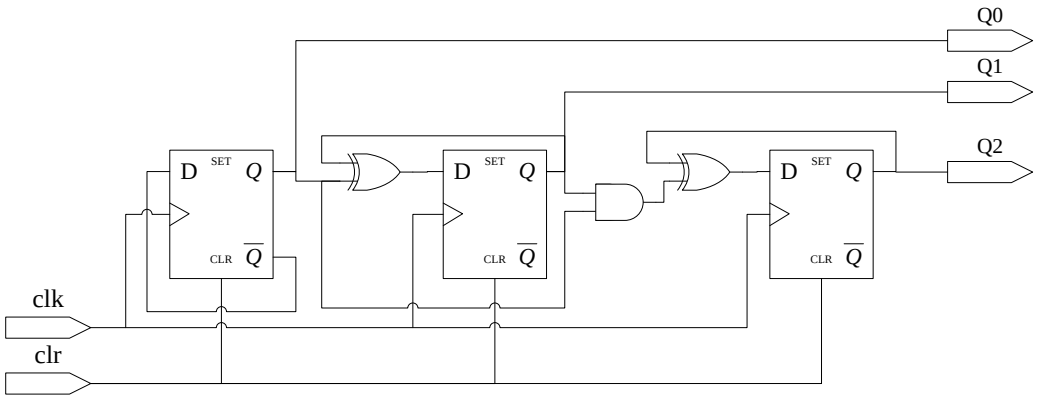


图 2　某有限状态机

解答：

1、 verilog 代码如下：

```
module fsm(
    input wire clk，
              clr,
    output reg Q0,
              Q1,
              Q2
    );
    always @(posedge clk, posedge clr)
    if(clr) begin
        q0 <= 1’b0;
        q1 <= 1’b0;
        q2 <= 1’b0;
    end
    else begin
        Q0 <= !Q0;
        Q1 <= Q0 ^ Q1;
        Q2 <= Q2 ^ (Q1 & Q0);
    end
endmodule
```

2、 测试程序

```
`timescale 1ns/1ns
module fsm_tb;
    reg clk， clr;
    wire Q0, Q1, Q2
    );
    fsm ufsm(
        .clk(clk),
        .clr(clr),
        .Q0(Q0),
        .Q1(Q1),
    always #5 clk = !clk;
    initial begin
        clk = 0;
        clr = 1;
     #20 clr = 0;
     #100 $finish;
    end
    initial begin
        $shm_open(“fsm.shm”);
        $shm_probe（）;
```

.Q2(Q2));	end endmodule

五、 **（20 分）**有一电路如下所示。Datai 是 8 位数据输入信号。同步复位信号 reset 低电平有效，当复位结束后，在时钟信号 clk 的上升沿，从 datai 送入数据。但当 bit4 为高电平时，datai 的低 4 位数据有效，高 4 位数据无效；当 bit4 为低电平时 datai 的 8 位数据有效。接收到的有效数据从 datao 送出，当 datao 送出的数据有效时，Do_en 信号输出高电平，否则为低电平。请用 verilog 实现此电路。

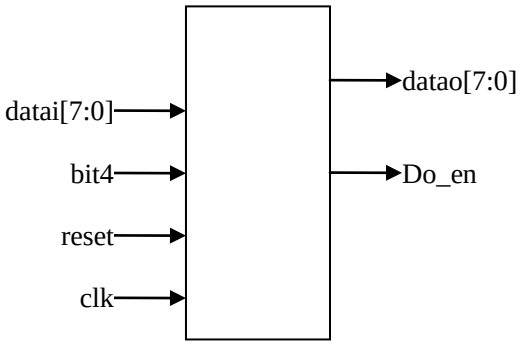
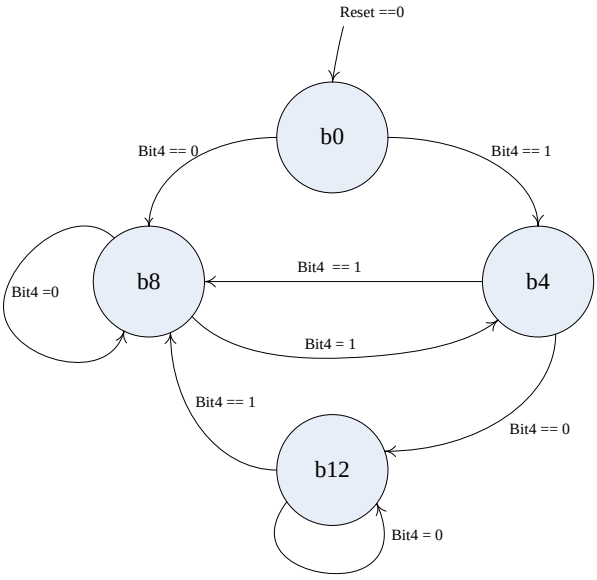


图 3 数据转发电路

解答：每个时钟周期送来的有效数据可能是 4 位，也可能是 8 位，由 bit4 信号指示。复位结束后，有效数据为 0。第一个送来的数据可能是 4 位，也可能是 8 位。如果第一个数据是 4 位，第二个数据是 8 位，则有效数据总共有 12 位。我们用状态机表示有效数据总共为 0 位、4 位、8 位、12 位，分别用状态 b0, b4, b8, b12 表示。其状态转换图为：



系统复位时，状态机处于 b0 状态，有效数据为 0 位。送来的数据在送出前，最多可能需要 12 位进行保存，我们用信号 dataiL[11：0]对输入数据进行保存。

复位结束后，状态转换如图所示。

b0 状态：有效数据为 0 位，不输出数据。若 bit4 为 1，则状态机进入 b4 状态，同时 dataiL[3:0]保存送来的 4 位有效数据 datai[3:0]。若 bit4 为 0，则状态机进入 b8 状态，同时 dataiL[7:0]保存送来的 8 位有效数据 datai[7:0]。

b4 状态：有效数据为 4 位，不足 8 位，因此不输出数据，do_en 为 0。若 bit4 为 1，则状态机进入 b8 状态，同时 dataiL[7:4]保存送来的 4 位有效数据 datai[3:0]。若 bit4 为 0，则状态机进入 b12 状态，同时 dataiL[11:4]保存送来的 4 位有效数据 datai[7:0]。

b8 状态：有效数据为 8 位，输出数据，do_en 为 1。若 bit4 为 1，则状态机进入 b4 状态，同时 dataiL[3:0]保存送来的 4 位有效数据 datai[3:0]。若 bit4 为 0，则状态机保持 b8 状态，同时 dataiL[7:0]保存送来的 4 位有效数据 datai[7:0]。

b12 状态：有效数据为 12 位，输出低 8 位数据，do_en 为 1。若 bit4 为 1，则状态机进入 b8 状态，同时 dataiL[3:0]保存已有的 4 位有效数据 datai[11:8]，同时 dataiL[7:4]保存送来的 4 位有效数据 datai[3:0]。若 bit4 为 0，则状态机保持 b12 状态，同时 dataiL[3:0]保存已有的 4 位有效数据 datai[11:8]，dataiL[11:4]保存送来的 8 位有效数据 datai[7:0]。

verilog 代码为：

```
module FIFO(
    input  wire   reset,
                  clk,
                  bit4,
    input wire [7 : 0] datai,
    output reg [7 : 0] datao,
    output reg do_en
    );

    reg [11 : 0] dataiL;

    parameter  b0 = 2'b00,
               b4 = 2'b01,
               b8 = 2'b10,
               b12= 2'b11;

    reg [1 : 0] state;

    always @(posedge clk, negedge reset)
        if(!reset)
            state <= b0;
        else
            case(state)
                b0: if(bit4)
                        state <= b4;
                    else
```

```
                        state <= b8;
            b4: if(bit4)
                        state <= b8;
                    else
                        state <= b12;
            b8: if(bit4)
                        state <= b4;
                    else
                        state <= b8;
            b12: if(bit4)
                        state <= b8;
                    else
                        state <= b12;
            default: state <= 2'bx;
        endcase

    always @(posedge clk)
        case(state)
            b0: if(!bit4)
                        dataiL[7 : 0]    <= datai;
                    else
                        dataiL[3 : 0]    <= datai[3 : 0];
            b4: if(!bit4)
                        dataiL[11: 4]    <= datai;
                    else
                        dataiL[7 : 4]    <= datai[3 : 0];
            b8: if(!bit4)
                        dataiL[7 : 0] <= datai;
                    else
                        dataiL[3 : 0]    <= datai[3 : 0];
            b12: if(!bit4)
                        dataiL[11 : 0]   <= {datai, dataiL[11 : 8]};
                    else
                        dataiL[7 : 0]    <= {datai[3:0], dataiL[11 : 8]};
            default: dataiL <= 'bx;
        endcase

    always @(*) begin
        datao = dataiL[7 : 0];
        do_en = state == b8 || state == b12;
    end

endmodule
```

六、 **（20 分）**一个 32×8 的异步 SRAM 的外部端口如图 4 所示，其读写信号时序图如图 5(时间单位 ns)所示。

1. （10 分）试建立此 SRAM 的 Verilog 行为模型。
2. （10 分）给出此 SRAM 模型的测试程序，要求测试程序使用 task：
   a) 向地址 12 写入 8’H55，地址 13 写 8’HAA。
   b) 再读取地址 12 和 13 的数据。

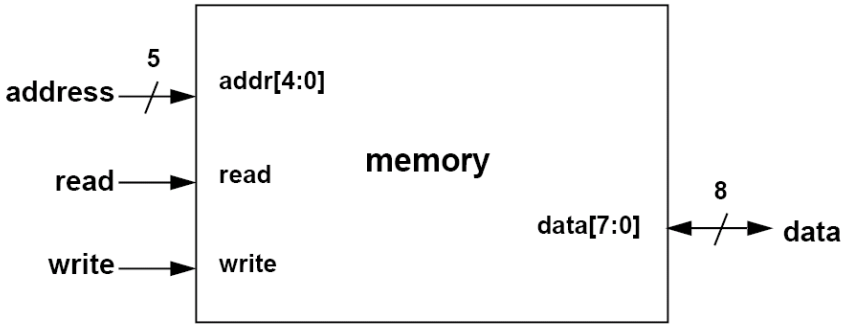


图 4　SRAM 的外部端口信号

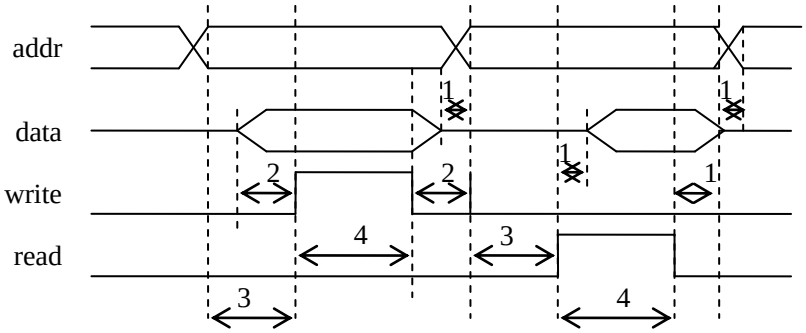


图 5　SRAM 外部端口信号时序图

答：

1、32x8 存储器功能模型： `timescale 1ns/1ns module mem32x8 (input wire [4:0] addr, input wire write, input wire read, inout wire [7:0] data	); reg [7:0] mem[4:0]; assign data = read ? mem(addr) : 8’bz; always @(posedge write) mem(addr) <= data; endmodule

2、测试程序

```
`timescale 1ns/1ns
module mem32x8_tb;
    reg [4:0] addr,
    reg   write,
    reg   read,
    wire [7:0] data
    reg [7 : 0] data_in;

mem32x8 umem (addr, write, read, data);
    assign data = write ? data_in : 8'bz;

    initial begin
        addr = 0;
        write = 0;
        read = 0;
        data_in = 0;
        mem_write(5'h12, 8'h55);
        mem_write(5'h13, 8'hAA);
        mem_read(5'h12);
        mem_read(5'h13);
        #10 $finish;
    end
```

```
task mem_write(
        input [4:0] ab;
        input [7:0] db;

        begin
            addr = ab;
        #1  data_in = db;
        #2  write = 1'b1;
        #4  write = 1'b0;
        #1  data_in = 8'bz;
        #1
        end
endtask
    Task mem_read(
        Input [4:0] ab;

        Begin
            addr = ab;
        #3  read = 1'b1;
        #4  read = 1'b0;
        #2
        end
endtask
endmodule
```