上海交通大学网络空间安全学院

数字系统设计作业报告

2017 年秋季学期

515030910067　杨超琪　never_say_never@sjtu.edu.cn

第 1 题

设计一个 Verilog 模块，产生图 1 所示的波形。

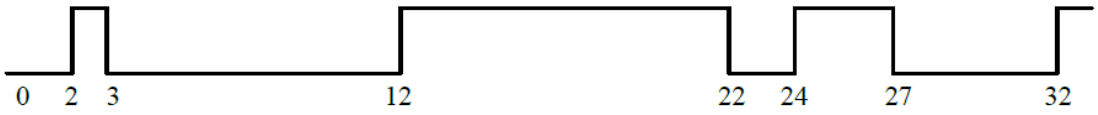


图 1、习题 1 波形图

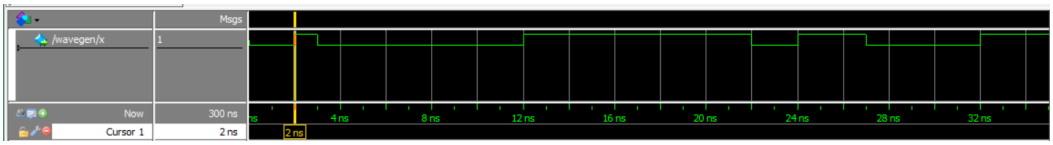
（1）设计模块

```
`timescale 10 ns / 1 ns
module  wavegen(x);
    // generate wave
    output reg x;
    // use initial module to change value
    initial
    begin
                x = 0;
          #0.2 x = 1;
          #0.1 x = 0;
          #0.9 x = 1;
          #1.0 x = 0;
          #0.2 x = 1;
          #0.3 x = 0;
          #0.5 x = 1;
    end
endmodule
```

（2）测试模块

```
`timescale 10 ns / 1 ns
`include "wavegen.v"
module testbench1(x); //test module
    // define an output
    output x;
    // example of module wavegen
    wavegen example(x);
    // monitor the process
    initial $monitor("it's at time %d, x=%d", $time, x);
endmodule
```

（3）测试波形图：如果很多，可以提供部分波形内容；



（4）显示输出（可选）：如果需要显示输出来说明模块设计的正确性；

```
run
# KERNEL: it's at time                    0, x=0
# KERNEL: it's at time                    2, x=1
# KERNEL: it's at time                    3, x=0
# KERNEL: it's at time                   12, x=1
# KERNEL: it's at time                   22, x=0
# KERNEL: it's at time                   24, x=1
# KERNEL: it's at time                   27, x=0
# KERNEL: it's at time                   32, x=1
```

第 2 题

8:3 编码器的真值表如表 1 所示，使用 always－case 结构，设计一个 8:3 编码器。并设计一个测试平台，对电路进行仿真。

（1）设计模块

```
module Encoder8x3(code, data);
    //define an output
    output reg [7:0] data;
    input [2:0] code;
    //use always module to change data value
    always @(*)
    case(code)//when code changes, data changes accordingly
        3'b000: data = 8'b0000_0001;
        3'b001: data = 8'b0000_0010;
        3'b010: data = 8'b0000_0100;
        3'b011: data = 8'b0000_1000;
        3'b100: data = 8'b0001_0000;
        3'b101: data = 8'b0010_0000;
        3'b110: data = 8'b0100_0000;
        3'b111: data = 8'b1000_0000;
        default: data = 8'b0000_0001;//default data
    endcase
endmodule
```

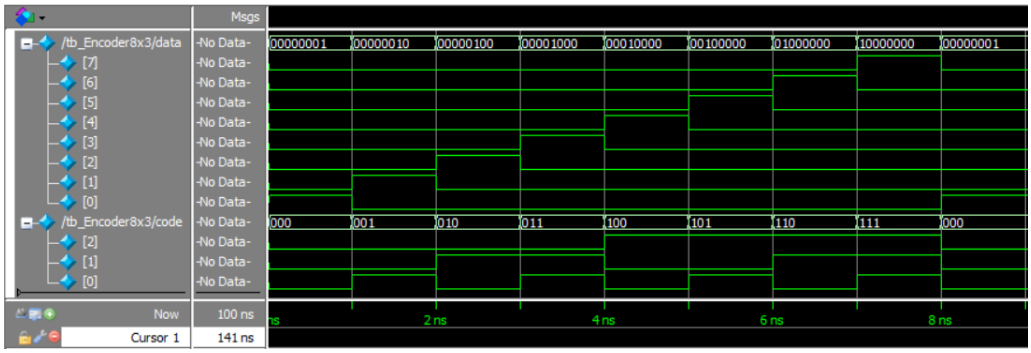
表 1、8:3 编码器真值表

输入 data[7:0]	输出 code[2:0]
0000_0001	0
0000_0010	1
0000_0100	2
0000_1000	3
0001_0000	4
0010_0000	5
0100_0000	6
1000_0000	7

（2）测试模块

```
`include "Encoder8x3.v"
module tb_Encoder8x3();
    //define code and data
    wire [7:0] data;
    reg [2:0] code;
    // give some code value
    initial
    begin
            code = 3'b000;
        #1  code = 3'b001;
        #1  code = 3'b010;
        #1  code = 3'b011;
        #1  code = 3'b100;
        #1  code = 3'b101;
        #1  code = 3'b110;
        #1  code = 3'b111;
        #1  code = 3'b000;
    end
    // example of module Encoder8x3
    Encoder8x3 example(code, data);
    //monitor the process
    initial $monitor("it's at time %d, data=%b", $time, data);
endmodule
```

（3）测试波形图：如果很多，可以提供部分波形内容；



（4）显示输出（可选）：如果需要显示输出来说明模块设计的正确性；

```
run
# KERNEL: it's at time                    0, data=00000001
# KERNEL: it's at time                    1, data=00000010
# KERNEL: it's at time                    2, data=00000100
# KERNEL: it's at time                    3, data=00001000
# KERNEL: it's at time                    4, data=00010000
# KERNEL: it's at time                    5, data=00100000
# KERNEL: it's at time                    6, data=01000000
# KERNEL: it's at time                    7, data=10000000
# KERNEL: it's at time                    8, data=00000001
```

第 3 题

a）如图 2 所示，使用 bufif0 和 bufif1 设计一个二选一多路选择器，并给出测试激励模块，和仿真测试结果。
b）以（a）设计的 2 选 1 多路选择器为底层模块，通过调用 2 选 1 多路选择器模块，设计一个 4 选 1 多路选择器，并给出仿真测试结果。

（1）设计模块

```
module mux2x1(dout, sel, din);
    // define output and input
    output dout;
    input [1:0] din;
    input sel;
    // use bufif0 and bufif1 to implement
    bufif1 b2(dout, din[1], sel);
    bufif0 b1(dout, din[0], sel);
endmodule
```

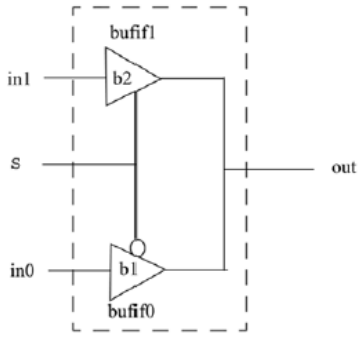


图 2、二选一多路选择器

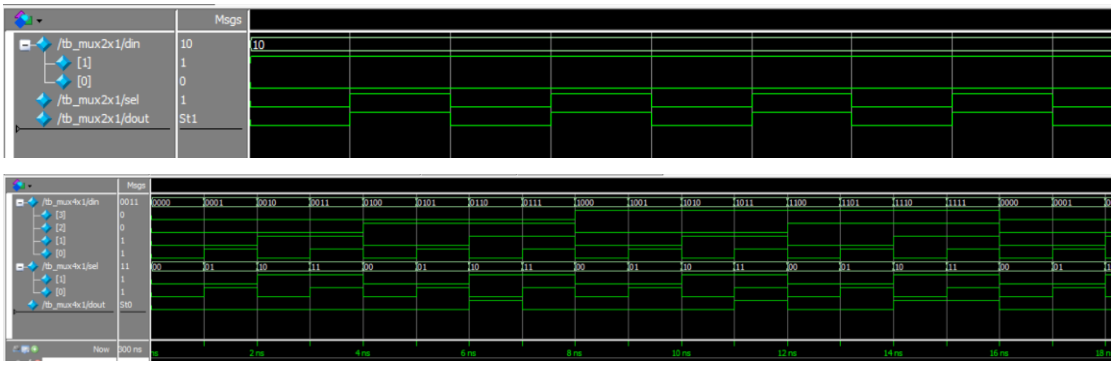
```
`include "mux2x1.v"
module mux4x1(dout, sel, din);
    // define output and input
    output dout;
    input [3:0] din;
    input [2:0] sel;
    wire dout1, dout2;
    // sel[0] control din[3:2]
    mux2x1 example1(dout1, sel[0], din[3:2]);
    // sel[0] control din[1:0]
    mux2x1 example2(dout2, sel[0], din[1:0]);
    // sel[1] control dout1,dout2
    mux2x1 example3(dout, sel[1], {dout1, dout2});
endmodule
```

（2）测试模块

```
`include "mux2x1.v"
module tb_mux2x1();
    // define signals
    reg [1:0] din;
    reg sel;
    wire dout;
    // initial din and sel
    initial
    begin
        din = 2'b10;
        sel = 0;
    end
    // change sel periodicly
    always #1 sel = ~sel;
    // example of mux2x1 module
    mux2x1 example(dout, sel, din);
    //monitor the process
    initial $monitor("it's at time %d, din=%b, sel=%b dout=%d", $time, din, sel, dout);
endmodule
```

```
`include "mux4x1.v"
module tb_mux4x1();
    // define signals
    reg [3:0] din;
    reg [1:0] sel;
    wire dout;
    // initial din and sel
    initial
    begin
        din = 4'b0000;
        sel = 2'b00;
    end
    // change sel periodicly
    always #1 sel = sel + 1;
    always #1 din = din + 1;
    // example of mux2x1 module
    mux4x1 example(dout, sel, din);
    //monitor the process
    initial $monitor("it's at time %d, din=%b, sel=%b dout=%d", $time, din, sel, dout);
endmodule
```

（3）测试波形图：如果很多，可以提供部分波形内容；



（4）显示输出（可选）：如果需要显示输出来说明模块设计的正确性；

```
# KERNEL: it's at time          128013, din=10, sel=1 dout=1
# KERNEL: it's at time          128014, din=10, sel=0 dout=0
# KERNEL: it's at time          128015, din=10, sel=1 dout=1
# KERNEL: it's at time          128016, din=10, sel=0 dout=0
# KERNEL: it's at time          128017, din=10, sel=1 dout=1
# KERNEL: it's at time          128018, din=10, sel=0 dout=0
# KERNEL: it's at time          128019, din=10, sel=1 dout=1
# KERNEL: it's at time          128020, din=10, sel=0 dout=0
# KERNEL: it's at time          128021, din=10, sel=1 dout=1
# KERNEL: it's at time          128022, din=10, sel=0 dout=0
# KERNEL: it's at time          128023, din=10, sel=1 dout=1
# KERNEL: it's at time          128024, din=10, sel=0 dout=0
# KERNEL: it's at time          128025, din=10, sel=1 dout=1
# KERNEL: it's at time          128026, din=10, sel=0 dout=0
# KERNEL: it's at time          128027, din=10, sel=1 dout=1
# KERNEL: it's at time          128028, din=10, sel=0 dout=0

# KERNEL: it's at time          163754, din=1010, sel=10 dout=0
# KERNEL: it's at time          163755, din=1011, sel=11 dout=1
# KERNEL: it's at time          163756, din=1100, sel=00 dout=0
# KERNEL: it's at time          163757, din=1101, sel=01 dout=0
# KERNEL: it's at time          163758, din=1110, sel=10 dout=1
# KERNEL: it's at time          163759, din=1111, sel=11 dout=1
# KERNEL: it's at time          163760, din=0000, sel=00 dout=0
# KERNEL: it's at time          163761, din=0001, sel=01 dout=0
# KERNEL: it's at time          163762, din=0010, sel=10 dout=0
# KERNEL: it's at time          163763, din=0011, sel=11 dout=0
# KERNEL: it's at time          163764, din=0100, sel=00 dout=0
# KERNEL: it's at time          163765, din=0101, sel=01 dout=0
# KERNEL: it's at time          163766, din=0110, sel=10 dout=1
# KERNEL: it's at time          163767, din=0111, sel=11 dout=0
# KERNEL: it's at time          163768, din=1000, sel=00 dout=0
# KERNEL: it's at time          163769, din=1001, sel=01 dout=0
# KERNEL: it's at time          163770, din=1010, sel=10 dout=0
```

（5）设计说明（可选）：如果有需要说明的部分。

Mux2x1 中：sel 为 0 时，读 din[0]，sel 为 1 时，都 din[1]。Mux4x1 中：sel 为 0 时，读 din[0]，sel 为 1 时，读 din[1]，sel 为 2 时，读 din[2]，sel 为 3 时，读 din[3]。

tb_mux2x1 测试模块中：设 din 固定为 2'10，而 sel 每个时间单位反转一次。

tb_mux4x1 测试模块中：设 din 初始值为 4'0000，而 sel 与 din 每个时间单位各加一。

第 4 题

设计一个 Verilog 模块，描述如图 3 所示的电路原理图表示的电路。并设计一个测试平台，对电路进行仿真。

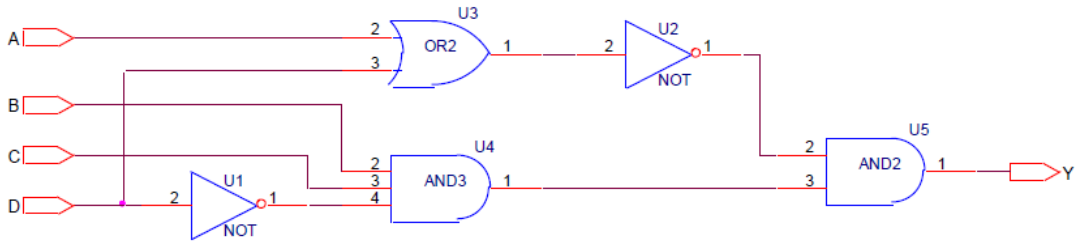


图 3、习题 4 电路原理图

（1）利用 Verilog 的基本门（门级原语），采用结构（Structural）描述方式设计；此时，模块名为：comb_str(Y, A, B, C, D)

（2）利用连续幅值语句，采用数据流（Dataflow）方式设计；此时，模块名为：comb_dataflow(Y, A, B, C, D)

（3）利用always 过程语句，采用行为和算法（Behvioral or algorithmic）方式设计；此时，模块名为：comb_behavior(Y, A, B, C, D)

（4）利用用户定义原语（UDP），使用真值表描述方式设计；此时，模块名为：comb_prim(Y, A, B, C, D)

（5）测试平台的模块名为：testbench_comb（）；注意，测试平台模块没有端口。

（1）设计模块

```
module comb_str(Y, A, B, C, D);
    // define of output and input
    input A, B, C, D;
    output Y;
    wire u1_after, u2_after, u3_after, u4_after;
    // use structural module
    or or1(u3_after, A, D);
    not not1(u1_after, D);
    and and1(u4_after, B, C, u1_after);
    not not2(u2_after, u3_after);
    and and2(Y, u2_after, u4_after);
endmodule
```

```
module comb_behavior(Y, A, B, C, D);
    // define output and input
    output reg Y;
    input A, B, C, D;
    // use always to present the relationship
    always @(A or B or C or D)
        Y <= ~(A|D) & B & C & (~D);
endmodule
```

```
module comb_behavior(Y, A, B, C, D);
    // define output and input
    output reg Y;
    input A, B, C, D;
    // use always to present the relationship
    always @(A or B or C or D)
        Y = ~(A|D) & B & C & (~D);
endmodule
```

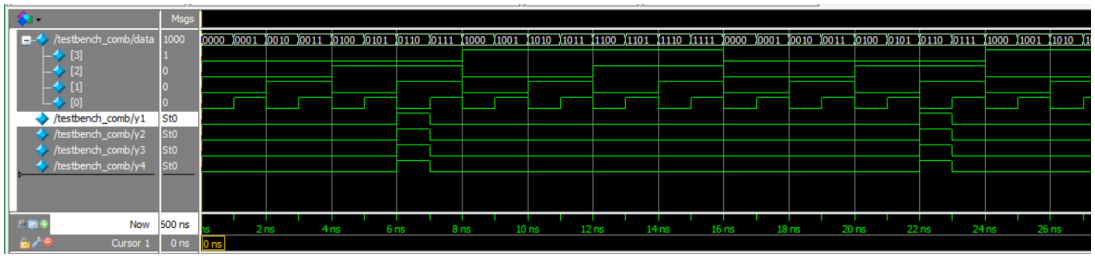
```
primitive comb_prim(Y, A, B, C, D);
    // define output and input
    output Y;
    input A, B, C, D;

    table
        // primitive table
        0 0 0 0 : 0;
        0 0 0 1 : 0;
        0 0 1 0 : 0;
        0 0 1 1 : 0;
        0 1 0 0 : 0;
        0 1 0 1 : 0;
        0 1 1 0 : 1;
        0 1 1 1 : 0;
        1 0 0 0 : 0;
        1 0 0 1 : 0;
        1 0 1 0 : 0;
        1 0 1 1 : 0;
        1 1 0 0 : 0;
        1 1 0 1 : 0;
        1 1 1 0 : 0;
        1 1 1 1 : 0;
    endtable
endprimitive
```

（2）测试模块

```
`include "comb_str.v"
`include "comb_dataflow.v"
`include "comb_behavior.v"
`include "comb_prim.v"
module testbench_comb();
    // define of output and input
    reg [3:0] data;
    wire y1, y2, y3, y4;
    // set initial data
    initial data = 4'b0000;
    // change data every time unit
    always #1 data = data + 1;
    // example of comb_str
    comb_str example(y1, data[3], data[2], data[1], data[0]);
    comb_dataflow example(y2, data[3], data[2], data[1], data[0]);
    comb_behavior example(y3, data[3], data[2], data[1], data[0]);
    comb_prim example(y4, data[3], data[2], data[1], data[0]);
    //monitor the process
    initial $monitor("it's at time %d, data=%b, y1=%b, y2=%b, y3=%b, y4=%b", $time, data, y1, y2, y3, y4);
endmodule
```

（3）测试波形图：如果很多，可以提供部分波形内容；



（4）显示输出（可选）：如果需要显示输出来说明模块设计的正确性；

Transcript

```
# it's at time                    0, data=0000, y1=0, y2=0, y3=0, y4=0
# it's at time                    1, data=0001, y1=0, y2=0, y3=0, y4=0
# it's at time                    2, data=0010, y1=0, y2=0, y3=0, y4=0
# it's at time                    3, data=0011, y1=0, y2=0, y3=0, y4=0
# it's at time                    4, data=0100, y1=0, y2=0, y3=0, y4=0
# it's at time                    5, data=0101, y1=0, y2=0, y3=0, y4=0
# it's at time                    6, data=0110, y1=1, y2=1, y3=1, y4=1
# it's at time                    7, data=0111, y1=0, y2=0, y3=0, y4=0
# it's at time                    8, data=1000, y1=0, y2=0, y3=0, y4=0
# it's at time                    9, data=1001, y1=0, y2=0, y3=0, y4=0
# it's at time                   10, data=1010, y1=0, y2=0, y3=0, y4=0
# it's at time                   11, data=1011, y1=0, y2=0, y3=0, y4=0
# it's at time                   12, data=1100, y1=0, y2=0, y3=0, y4=0
# it's at time                   13, data=1101, y1=0, y2=0, y3=0, y4=0
# it's at time                   14, data=1110, y1=0, y2=0, y3=0, y4=0
# it's at time                   15, data=1111, y1=0, y2=0, y3=0, y4=0
# it's at time                   16, data=0000, y1=0, y2=0, y3=0, y4=0
# it's at time                   17, data=0001, y1=0, y2=0, y3=0, y4=0
# it's at time                   18, data=0010, y1=0, y2=0, y3=0, y4=0
# it's at time                   19, data=0011, y1=0, y2=0, y3=0, y4=0
# it's at time                   20, data=0100, y1=0, y2=0, y3=0, y4=0
# it's at time                   21, data=0101, y1=0, y2=0, y3=0, y4=0
# it's at time                   22, data=0110, y1=1, y2=1, y3=1, y4=1
# it's at time                   23, data=0111, y1=0, y2=0, y3=0, y4=0
# it's at time                   24, data=1000, y1=0, y2=0, y3=0, y4=0
```

（5）设计说明（可选）：如果有需要说明的部分。

在这里我们设计了 comb_str，comb_dataflow，comb_behavior 与 comb_prim 四个模块。在测试模块中，输入初始值为 4’b0000，而后每个时间单位增加一，上述四个模块输出分别为 y1，y2，y3，y4。

第 5 题

根据下面的的布尔方程，设计两个组合逻辑电路模块：

i） Y1(A,B,C)=∑m(1,2,4,5)

（1）设计模块

```
module comb_Y1(Y, A, B, C);
    // define output and input
    output Y;
    input A, B, C;
    // use assign to present the relationship
    assign Y = (~A) & (~B) & C | (~A) & B & (~C) |
               A & (~B) & (~C) | A & (~B) & C;
endmodule
```

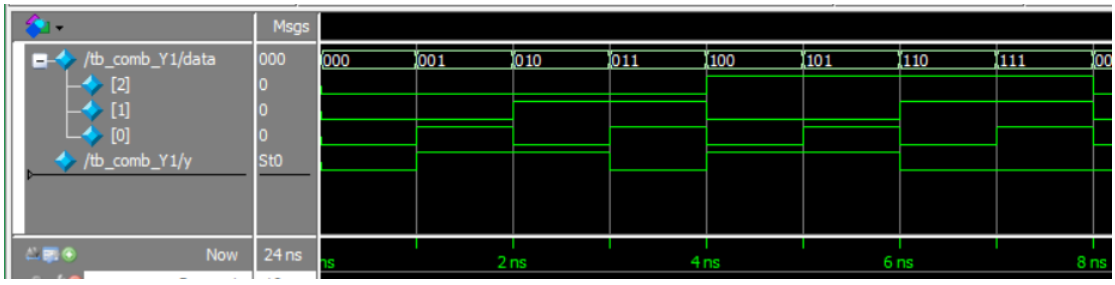
```
module comb_Y2(Y, A, B, C, D);
    // define output and input
    output Y;
    input A, B, C, D;
    // use assign to present the relationship
    assign Y = (~A) & B & (~C) & (~D) | (~A) & B & (~C) & D |
               (~A) & B & C & (~D) | (~A) & B & C & D |
                A & (~B) & C & D | A & B & (~C) & (~D) |
                A & B & (~C) & D;
endmodule
```

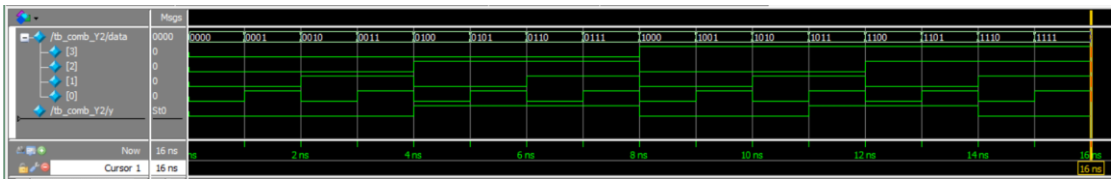
（2）测试模块

```
`include "comb_Y1.v"
module tb_comb_Y1();
    // define signals
    wire y;
    reg [2:0] data;
    // use initial to set data
    initial data = 3'b000;
    // use always to change data
    always
    begin
        #1 data = data + 1;
        // if the circle ends then stop
        if (data==3'b000) $stop;
        else ;
    end
    //example of comb_Y1
    comb_Y1 example(y, data[2], data[1], data[0]);
    // monitor the process
    initial $monitor("it's at time %d, data=%b, y=%b", $time, data, y);
endmodule
```

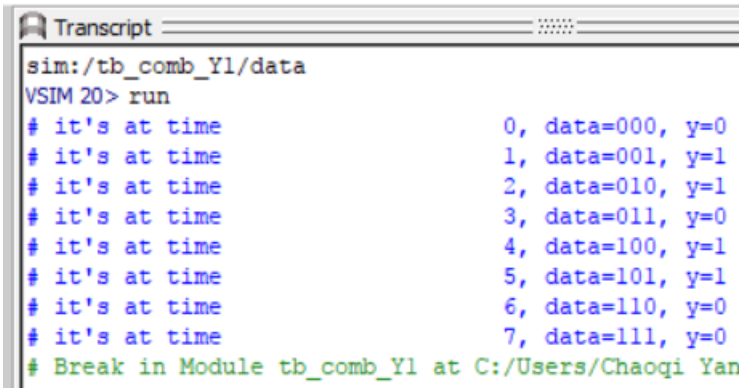
```
`include "comb_Y2.v"
module tb_comb_Y2();
    // define signals
    wire y;
    reg [3:0] data;
    // use initial to set data
    initial data = 4'b0000;
    // use always to change data
    always
    begin
        #1 data = data + 1;
        // if the circle ends then stop
        if (data==4'b0000) $stop;
        else ;
    end
    //example of comb_Y1
    comb_Y2 example(y, data[3], data[2], data[1], data[0]);
    // monitor the process
    initial $monitor("it's at time %d, data=%b, y=%b", $time, data, y);
endmodule
```

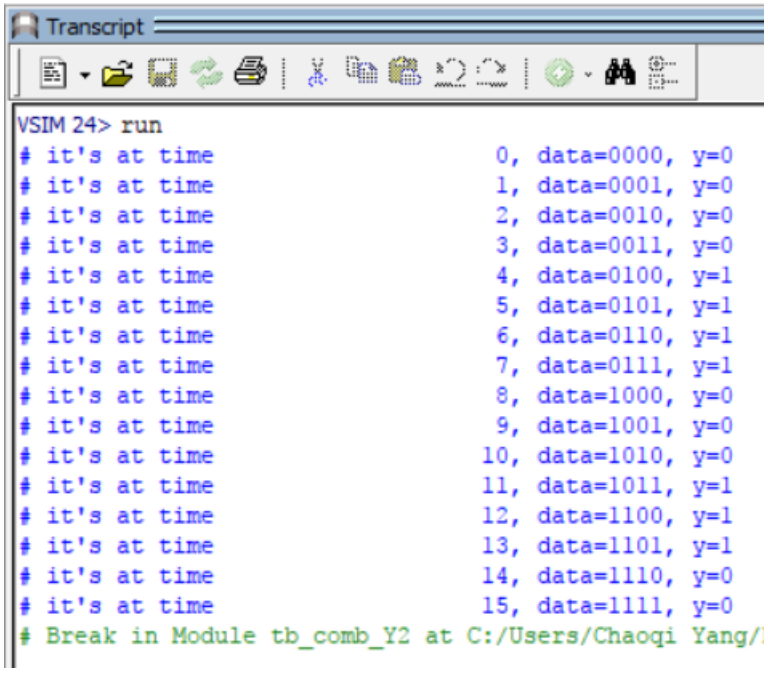
（3）测试波形图：如果很多，可以提供部分波形内容；





（4）显示输出（可选）：如果需要显示输出来说明模块设计的正确性；





（5）设计说明（可选）：如果有需要说明的部分。

在这里我们使用最小项之和来设计电路。两个测试模块中 data 的输入分别为 3’b000 ~ 3’b111 和 4’b0000～4’b1111，然后调用系统函数 stop 来终止测试，因此涵盖了所有可能的情况。

第 6 题

设计一个 Verilog 模块，使用 4 位输出码表示 8 位输入字中 1 的个数。并设计测试平台进行仿真验证。

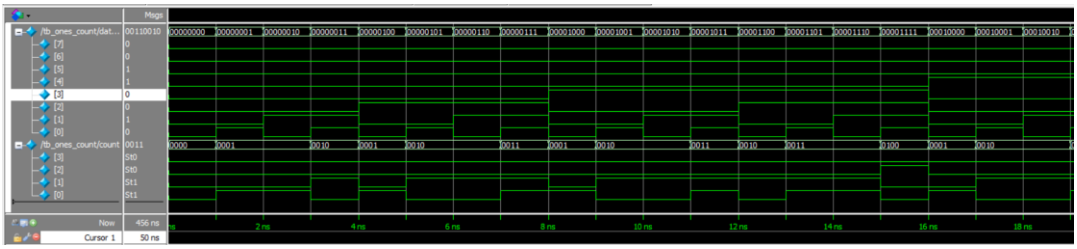
（1）设计模块

```
module ones_count(count, dat_in);
    // define output and input
    output [3:0] count;
    input [7:0] dat_in;
    // use assign to calculate count
    assign count = dat_in[0] + dat_in[1] + dat_in[2] +
                   dat_in[3] + dat_in[4] + dat_in[5] +
                   dat_in[6] + dat_in[7];
endmodule
```

（2）测试模块

```
`include "ones_count.v"
module tb_ones_count();
    // define signals
    reg [7:0] dat_in;
    wire [3:0] count;
    // initially set dat_in
    initial
    begin
        dat_in = 8'b0000_0000;
    end
    // use always module to change dat_in
    always
    begin
        #1 dat_in = dat_in + 1;
        if (dat_in == 8'b0000_0000)
            $stop;
        else ;
    end
    // example of ones_count
    ones_count example(count, dat_in);
    // monitor the process
    initial $monitor("it's at time %d, dat_in=%b, count=%b", $time, dat_in, count);
endmodule
```

（3）测试波形图：如果很多，可以提供部分波形内容；



（3）显示输出（可选）：如果需要显示输出来说明模块设计的正确性；

Transcript

```
sim:/tb_ones_count/count
VSIM 3> run
# it's at time                    0, dat_in=00000000, count=0000
# it's at time                    1, dat_in=00000001, count=0001
# it's at time                    2, dat_in=00000010, count=0001
# it's at time                    3, dat_in=00000011, count=0010
# it's at time                    4, dat_in=00000100, count=0001
# it's at time                    5, dat_in=00000101, count=0010
# it's at time                    6, dat_in=00000110, count=0010
# it's at time                    7, dat_in=00000111, count=0011
# it's at time                    8, dat_in=00001000, count=0001
# it's at time                    9, dat_in=00001001, count=0010
# it's at time                   10, dat_in=00001010, count=0010
# it's at time                   11, dat_in=00001011, count=0011
# it's at time                   12, dat_in=00001100, count=0010
# it's at time                   13, dat_in=00001101, count=0011
# it's at time                   14, dat_in=00001110, count=0011
# it's at time                   15, dat_in=00001111, count=0100
# it's at time                   16, dat_in=00010000, count=0001
# it's at time                   17, dat_in=00010001, count=0010
# it's at time                   18, dat_in=00010010, count=0010
# it's at time                   19, dat_in=00010011, count=0011
```

第 7 题

设计一个 10 进制计数器的 Verilog 模块。

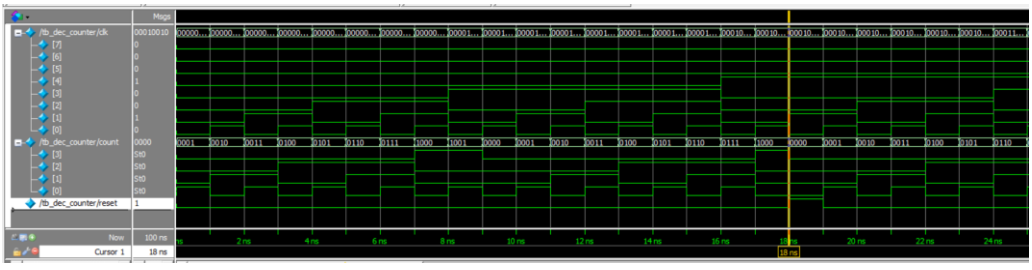
（1）设计模块

```
module dec_counter(count, clk, reset);
    // define output and input
    output reg [3:0] count;
    input [31:0] clk;
    input reset;
    // initial count
    initial count = 4'b0000;
    // count process
    always @(clk)
    begin
        if (reset)// if reset
            count <= 0;
        else      // if not reset
        begin
            count <= count + 1;
            // if count=10 then change to 0 in to next clk
            if (count == 4'b1010)
                count <= 4'b0000;
        end
    end
endmodule
```

（2）测试模块

```
`include "dec_counter.v"
module tb_dec_counter();
    // define signals
    reg [7:0] clk;
    wire [3:0] count;
    reg reset;
    // initially set clk and reset
    initial
    begin
        clk = 0;
        reset  =1'b0;
    end
    // create a clk and change reset every 7 time units
    always
    begin
        #1 clk = clk + 1;
        if (clk == 8'b0000_0000)
            $stop;
        else ;
    end
    always
    begin
        #18 reset = 1;
        #1 reset = 0;
    end
    // example of dec_counter
    dec_counter example(count, clk, reset);
    // monitor the process
    initial $monitor("it's at time %d, clk=%d, count=%d, reset=%b", $time, clk, count, reset);
endmodule
```

（3）测试波形图：如果很多，可以提供部分波形内容；



（4）显示输出（可选）：如果需要显示输出来说明模块设计的正确性；

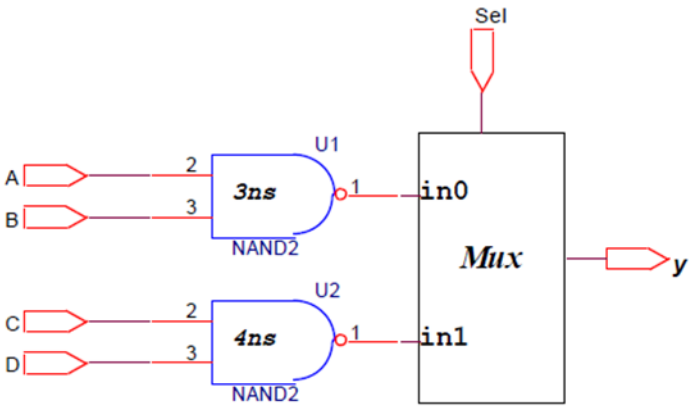
```
Transcript
sim:/tb_dec_counter/reset
VSIM 11> run
# it's at time                    0, clk=  0, count= 1, reset=0
# it's at time                    1, clk=  1, count= 2, reset=0
# it's at time                    2, clk=  2, count= 3, reset=0
# it's at time                    3, clk=  3, count= 4, reset=0
# it's at time                    4, clk=  4, count= 5, reset=0
# it's at time                    5, clk=  5, count= 6, reset=0
# it's at time                    6, clk=  6, count= 7, reset=0
# it's at time                    7, clk=  7, count= 8, reset=0
# it's at time                    8, clk=  8, count= 9, reset=0
# it's at time                    9, clk=  9, count= 0, reset=0
# it's at time                   10, clk= 10, count= 1, reset=0
# it's at time                   11, clk= 11, count= 2, reset=0
# it's at time                   12, clk= 12, count= 3, reset=0
# it's at time                   13, clk= 13, count= 4, reset=0
# it's at time                   14, clk= 14, count= 5, reset=0
# it's at time                   15, clk= 15, count= 6, reset=0
# it's at time                   16, clk= 16, count= 7, reset=0
# it's at time                   17, clk= 17, count= 8, reset=0
# it's at time                   18, clk= 18, count= 0, reset=1
# it's at time                   19, clk= 19, count= 1, reset=0
# it's at time                   20, clk= 20, count= 2, reset=0
# it's at time                   21, clk= 21, count= 3, reset=0
# it's at time                   22, clk= 22, count= 4, reset=0
```

（5）设计说明（可选）：如果有需要说明的部分。

测试台模块中，选择 clk 从 0 开始每隔一个时间单位增加一，一直到 255 时调用系统 stop 函数停止仿真，reset 函数每隔 18 个时间单位重置一次。

第 8 题

采用结构（Structural）描述方式设计一个 Verilog 模块，描述图 4 所示电路原理图表示的电路。



Mux 多路复用器真值表

sel	y
0	in0
1	in1

图 4、习题 8 电路原理图

（1）设计模块

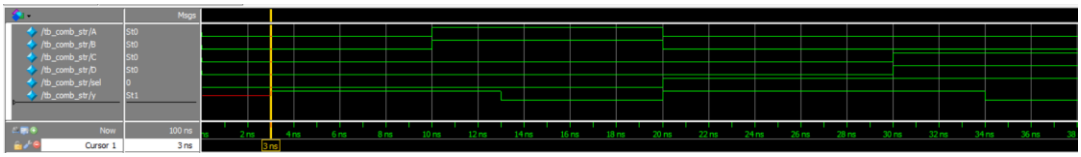
```
module mux2x1(dout, sel, din);
    // define output and input
    output dout;
    input [1:0] din;
    input sel;
    // use bufif0 and bufif1 to implement
    bufif1 b2(dout, din[1], sel);
    bufif0 b1(dout, din[0], sel);
    //initial $monitor("it's at time %d, din=%b, sel=%b dout=%d", $time, din, sel, dout);
endmodule
```

```
`include "mux2x1.v"
module comb_str(y, sel, A, B, C, D);
    // define output and input
    output y;
    input sel, A, B, C, D;
    wire in0, in1;
    // example of internal structure model
    nand #(3) nand1(in0, A, B);
    nand #(4) nand2(in1, C, D);
    mux2x1 example(y, sel, {in1, in0});
endmodule
```

（2）测试模块

```
`include "comb_str.v"
module tb_comb_str();
    // define signals
    wire y, A, B, C, D;
    reg sel=1'b0;
    reg [3:0] data;
    // use assign to set A,B,C,D
    assign {A, B, C, D} = data;
    // change data
    initial
    begin
            data = 4'b0000;
        # 10 data = 4'b1100;
        # 10 data = 4'b0000;
            sel = ~sel;
        # 10 data = 4'b0011;
        $stop;
    end
    // change sel every time unit
    // example of comb_str
    comb_str example(y, sel, A, B, C, D);
    // monitor the process
    initial $monitor("it's at time %d, ABCD=%b, sel=%d, y=%b", $time, data, sel, y);
endmodule
```

（3）测试波形图：如果很多，可以提供部分波形内容；



（4）显示输出（可选）：如果需要显示输出来说明模块设计的正确性；

```
VSIM 3> run
# it's at time                    0, ABCD=0000, sel=0, y=x
# it's at time                    3, ABCD=0000, sel=0, y=1
# it's at time                   10, ABCD=1100, sel=0, y=1
# it's at time                   13, ABCD=1100, sel=0, y=0
# it's at time                   20, ABCD=0000, sel=1, y=1
# it's at time                   30, ABCD=0011, sel=1, y=1
# it's at time                   34, ABCD=0011, sel=1, y=0
```

第 9 题

根据下面本源多项式公式，设计一个线性反馈移位寄存器，要求使用内部反馈方式设计。

$$P(x)=x^{26}+x^{8}+x^{7}+x+1$$

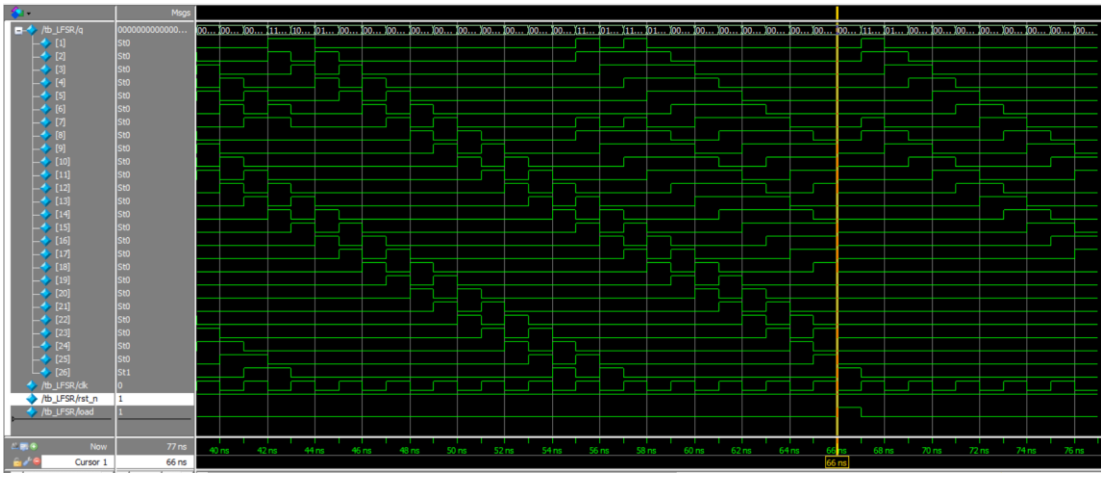
（1）设计模块

```
module LFSR( output reg [1:26] q, // 26 bit data output.
             input clk, // Clock input.
             input rst_n, // Synchronous reset input.
             input load, // Synchronous load input.
             input [1:26] din // 26 bit parallel data input.
           );
    always @(clk)
    begin
        if ( ~rst_n )// if reset
            q <= 0;
        else begin
            if (load)// if load
                q <= (|din) ? din : 26'b1;
            else begin
                if ( q == 0 )
                    q <= 1;
                else begin// change the q
                    q[9:26] <= q[8:25];
                    q[8] <= q[7] ^ q[26];
                    q[7] <= q[6] ^ q[26];
                    q[3:6] <= q[2:5];
                    q[2] <= q[1] ^ q[26];
                    q[1] <= q[26];
                end
            end
        end
    end
endmodule
```

（2）测试模块

```
`include "LFSR.v"
module tb_LFSR();
    // define signals
    wire [1:26] q;
    reg clk, rst_n, load;
    wire [1:26] din=26'b1;
    // create clock
    always #1 clk = ~clk;
    initial
    begin
        clk = 0;
        rst_n = 1;
        load = 0;
        #15 rst_n = 0;
        #1  rst_n = 1;
        #50 load = 1;
        #1  load = 0;
        #10 $stop;
    end
    // example of LFSR
    LFSR example(q, clk, rst_n, load, din);
    // monitor the process
    initial $monitor("it's at time %d, q=%b, rst_n=%d, load=%b", $time, q, rst_n, load);
endmodule
```

（3）测试波形图：如果很多，可以提供部分波形内容；



（4)显示输出(可选)：如果需要显示输出来说明模块设计的正确性；

Transcript

```
# it's at time                33, q=00000000000000001100001100, rst_n=1, load=0
# it's at time                34, q=00000000000000000110000110, rst_n=1, load=0
# it's at time                35, q=00000000000000000011000011, rst_n=1, load=0
# it's at time                36, q=11000011000000000001100001, rst_n=1, load=0
# it's at time                37, q=10100010100000000000110000, rst_n=1, load=0
# it's at time                38, q=01010001010000000000011000, rst_n=1, load=0
# it's at time                39, q=00101000101000000000001100, rst_n=1, load=0
# it's at time                40, q=00010100010100000000000110, rst_n=1, load=0
# it's at time                41, q=00001010001010000000000011, rst_n=1, load=0
# it's at time                42, q=11000110000101000000000001, rst_n=1, load=0
# it's at time                43, q=10100000000010100000000000, rst_n=1, load=0
# it's at time                44, q=01010000000001010000000000, rst_n=1, load=0
# it's at time                45, q=00101000000000101000000000, rst_n=1, load=0
# it's at time                46, q=00010100000000010100000000, rst_n=1, load=0
# it's at time                47, q=00001010000000001010000000, rst_n=1, load=0
# it's at time                48, q=00000101000000000101000000, rst_n=1, load=0
# it's at time                49, q=00000010100000000010100000, rst_n=1, load=0
# it's at time                50, q=00000001010000000001010000, rst_n=1, load=0
# it's at time                51, q=00000000101000000000101000, rst_n=1, load=0
# it's at time                52, q=00000000010100000000010100, rst_n=1, load=0
# it's at time                53, q=00000000001010000000001010, rst_n=1, load=0
# it's at time                54, q=00000000000101000000000101, rst_n=1, load=0
# it's at time                55, q=11000011000010100000000010, rst_n=1, load=0
# it's at time                56, q=01100001100001010000000001, rst_n=1, load=0
# it's at time                57, q=11110011110000101000000000, rst_n=1, load=0
# it's at time                58, q=01111001111000010100000000, rst_n=1, load=0
# it's at time                59, q=00111100111100001010000000, rst_n=1, load=0
# it's at time                60, q=00011110011110000101000000, rst_n=1, load=0
# it's at time                61, q=00001111001111000010100000, rst_n=1, load=0
# it's at time                62, q=00000111100111100001010000, rst_n=1, load=0
# it's at time                63, q=00000011110011110000101000, rst_n=1, load=0
# it's at time                64, q=00000001111001111000010100, rst_n=1, load=0
# it's at time                65, q=00000000111100111100001010, rst_n=1, load=0
# it's at time                66, q=00000000000000000000000001, rst_n=1, load=1
# it's at time                67, q=11000011000000000000000000, rst_n=1, load=0
# it's at time                68, q=01100001100000000000000000, rst_n=1, load=0
# it's at time                69, q=00110000110000000000000000, rst_n=1, load=0
# it's at time                70, q=00011000011000000000000000, rst_n=1, load=0
# it's at time                71, q=00001100001100000000000000, rst_n=1, load=0
# it's at time                72, q=00000110000110000000000000, rst_n=1, load=0
# it's at time                73, q=00000011000011000000000000, rst_n=1, load=0
# it's at time                74, q=00000001100001100000000000, rst_n=1, load=0
# it's at time                75, q=00000000110000110000000000, rst_n=1, load=0
# it's at time                76, q=00000000011000011000000000, rst_n=1, load=0
```

第 10 题

设计一个 Verilog 模块，描述如图 5 所示的电路原理图表示的电路。并设计一个测试平台，对电路进行仿真。要求：使用单一模块设计。

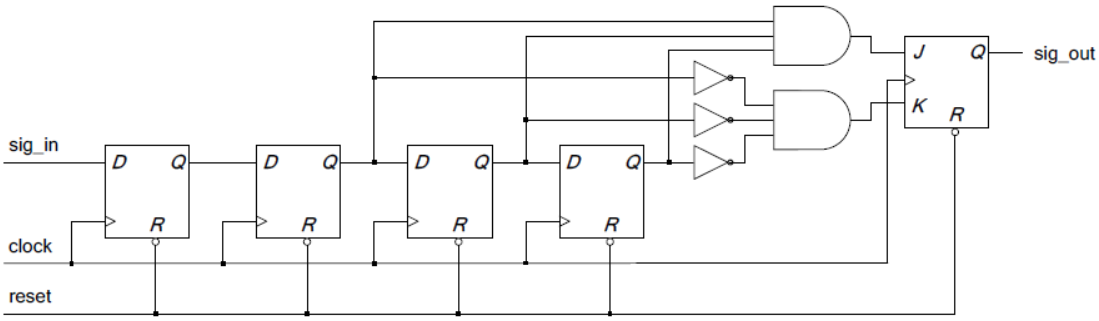


图 5、习题 10 电路原理图

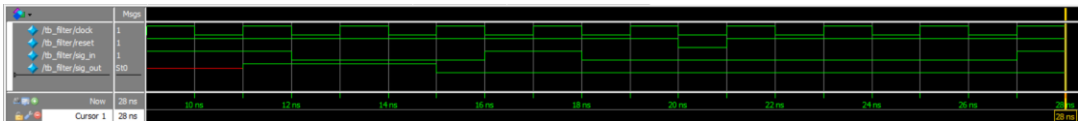
（1）设计模块

```
module filter(sig_out, clock, reset, sig_in);
    // define output and input
    output reg sig_out;
    input clock, reset, sig_in;
    reg [3:0] data;
    wire j, k;
    //
    always @(clock)
        if (~reset) begin
            sig_out = 0;
            data = 4'b0000;
        end
        else begin
            data[2:0] <= data[3:1];
            data[3] <= sig_in;
            case({j,k})
                2'b10: sig_out = 1;
                2'b01: sig_out = 0;
                2'b00: sig_out = sig_out;
                default: sig_out = ~sig_out;
            endcase
        end
    //
    assign j = data[3] & data[2] & data[1];
    assign k = ~data[3] & ~data[2] & ~data[1];
    initial $monitor("it's at time %d, data=%b, j,k=%b, reset=%d, sig_out=%b",
            $time, data, {j,k}, reset, sig_out);
endmodule
```

（2）测试模块

```
module tb_filter();
    // define signals
    reg clock, reset, sig_in, sig_out;
    // create clock
    always #1 clock = ~clock;
    // create signals
    initial
    begin
        clock = 0;
        reset = 1;
        #2 sig_in = 0; #2 sig_in = 1;
        #2 sig_in = 0; #2 sig_in = 1;
        #2 sig_in = 1; #2 sig_in = 0;
        #2 sig_in = 0; #2 sig_in = 1;
        #2 sig_in = 0; #2 reset = 0;
        #1 reset = 1;  #2 sig_in = 0;
        #2 sig_in = 0; #2 sig_in = 1;
        #1 $stop;
    end
    // example of filter
    filter example(sig_out, clock, reset, sig_in);
endmodule
```

（3）测试波形图：如果很多，可以提供部分波形内容；



（4）显示输出（可选）：如果需要显示输出来说明模块设计的正确性；

```
VSIM 22> run
# it's at time                  0, data=xxxx, j,k=xx, reset=1, sig_out=x
# it's at time                  2, data=0xxx, j,k=0x, reset=1, sig_out=x
# it's at time                  3, data=00xx, j,k=0x, reset=1, sig_out=x
# it's at time                  4, data=100x, j,k=00, reset=1, sig_out=x
# it's at time                  5, data=1100, j,k=00, reset=1, sig_out=x
# it's at time                  6, data=0110, j,k=00, reset=1, sig_out=x
# it's at time                  7, data=0011, j,k=00, reset=1, sig_out=x
# it's at time                  8, data=1001, j,k=00, reset=1, sig_out=x
# it's at time                  9, data=1100, j,k=00, reset=1, sig_out=x
# it's at time                 10, data=1110, j,k=10, reset=1, sig_out=x
# it's at time                 11, data=1111, j,k=10, reset=1, sig_out=1
# it's at time                 12, data=0111, j,k=00, reset=1, sig_out=1
# it's at time                 13, data=0011, j,k=00, reset=1, sig_out=1
# it's at time                 14, data=0001, j,k=01, reset=1, sig_out=1
# it's at time                 15, data=0000, j,k=01, reset=1, sig_out=0
# it's at time                 16, data=1000, j,k=00, reset=1, sig_out=0
# it's at time                 17, data=1100, j,k=00, reset=1, sig_out=0
# it's at time                 18, data=0110, j,k=00, reset=1, sig_out=0
# it's at time                 19, data=0011, j,k=00, reset=1, sig_out=0
# it's at time                 20, data=0000, j,k=01, reset=0, sig_out=0
# it's at time                 21, data=0000, j,k=01, reset=1, sig_out=0
# it's at time                 27, data=1000, j,k=00, reset=1, sig_out=0
```

（5）设计说明（可选）：如果有需要说明的部分。

在这里，波形不能很明显地显示该模块的功能，我们来看输出结果，这里明显地打印出每个时刻 sig_out 与 JK 触发器的关系，同时在 20 时刻有一次 reset。

第 11 题

设计一个能够递增和递减的 8 位双向循环计数器，计数器的示意图如图 6 所示。

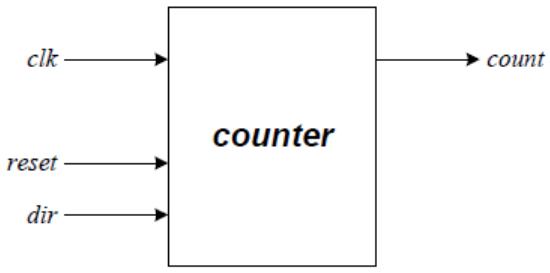


图 6、双向循环计数器

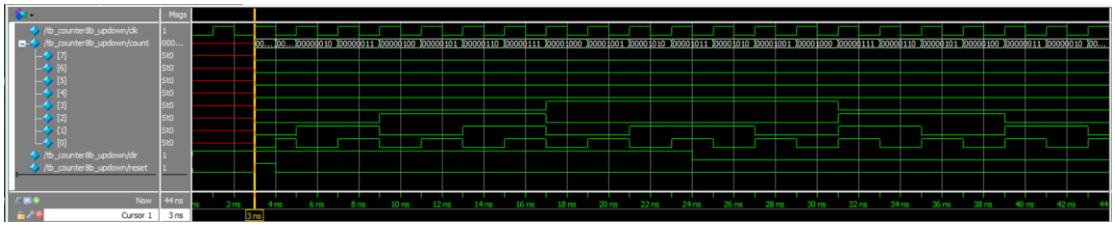
（1）设计模块

```
module counter8b_updown(count, clk, reset, dir);
    // define output and input
    output reg [7:0] count;
    input clk, reset, dir;
    // always module
    always @(posedge clk or posedge reset)
    begin
        if (reset) // if reset
            count <= 8'b0000_0000;
        else begin
            if (dir==1'b0) // if dir = 1
                count <= count - 1;
            else if (dir==1'b1) // if dir = 0
                count <= count + 1;
            else ;
        end
    end
endmodule
```

（2）测试模块

```
`include "counter8b_updown.v"
module tb_counter8b_updown();
    // define signals
    reg clk=1'b0;
    wire [7:0] count;
    reg dir, reset;
    // create clock
    always #1 clk = ~clk;
    //initial module
    initial begin
        dir = 1;
        reset = 0;
        #3 reset = 1;
        #1 reset = 0;
        #20 dir = 0;
        #20 $stop;
    end
    //example of counter8b_updown
    counter8b_updown example(count, clk, reset, dir);
    // monitor the process
    initial $monitor("it's at time %d, count=%b, reset=%b, dir=%b",
        $time, count, reset, dir);
endmodule
```

（3）测试波形图：如果很多，可以提供部分波形内容；



（4)显示输出(可选)：如果需要显示输出来说明模块设计的正确性；

```
VSIM 3> run
# it's at time                    0, count=xxxxxxxx, reset=0, dir=1
# it's at time                    3, count=00000000, reset=1, dir=1
# it's at time                    4, count=00000001, reset=0, dir=1
# it's at time                    5, count=00000010, reset=0, dir=1
# it's at time                    7, count=00000011, reset=0, dir=1
# it's at time                    9, count=00000100, reset=0, dir=1
# it's at time                   11, count=00000101, reset=0, dir=1
# it's at time                   13, count=00000110, reset=0, dir=1
# it's at time                   15, count=00000111, reset=0, dir=1
# it's at time                   17, count=00001000, reset=0, dir=1
# it's at time                   19, count=00001001, reset=0, dir=1
# it's at time                   21, count=00001010, reset=0, dir=1
# it's at time                   23, count=00001011, reset=0, dir=1
# it's at time                   24, count=00001011, reset=0, dir=0
# it's at time                   25, count=00001010, reset=0, dir=0
# it's at time                   27, count=00001001, reset=0, dir=0
# it's at time                   29, count=00001000, reset=0, dir=0
# it's at time                   31, count=00000111, reset=0, dir=0
# it's at time                   33, count=00000110, reset=0, dir=0
# it's at time                   35, count=00000101, reset=0, dir=0
# it's at time                   37, count=00000100, reset=0, dir=0
# it's at time                   39, count=00000011, reset=0, dir=0
# it's at time                   41, count=00000010, reset=0, dir=0
# it's at time                   43, count=00000001, reset=0, dir=0
```

（5）设计说明（可选）：如果有需要说明的部分。

在这里，我们在时刻 3 进行了 count 的 reset，异步清零，此刻开始递增计数；之后在时刻 24 同步开始递减计数。

第 12 题

设计一个 8 位算术逻辑单元（ALU），该单元的输入为操作数 a 和 b，以及操作码 oper，输出为 { c_out , sum }，并且具有如下表所示的功能。

操作码	功能
and	a + b + c_in
subtract	a + ~b + c_in
subtract_a	b + ~a + ~c_in
or_ab	{ 1'b0, a | b }
and_ab	{ 1'b0, a & b }
not_ab	{ 1'b0, (~a) & b }
exor	{ 1'b0, a ^ b }
exnor	{ 1'b0, a ~^ b }

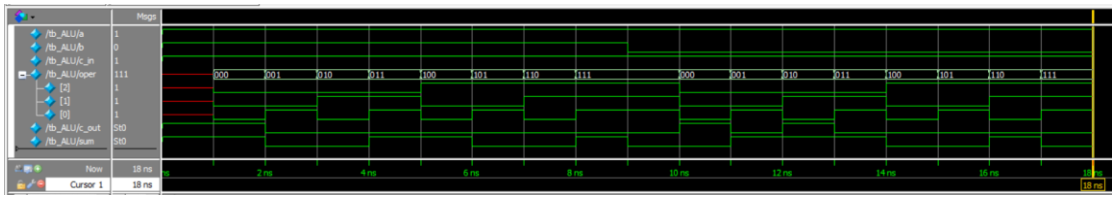
（1）设计模块

```
module ALU(c_out, sum, oper, a, b, c_in);
    // define output and input
    output reg c_out, sum;
    input [2:0] oper;
    input a, b, c_in;
    // always operate module
    always @(*)
    case(oper)
        3'b000: {c_out, sum} = a + b + c_in;
        3'b001: {c_out, sum} = a + ~b + c_in;
        3'b010: {c_out, sum} = b + ~a + c_in;
        3'b011: {c_out, sum} = {1'b0, a | b};
        3'b100: {c_out, sum} = {1'b0, a & b};
        3'b101: {c_out, sum} = {1'b0, ~a & b};
        3'b110: {c_out, sum} = {1'b0, a ^ b};
        3'b111: {c_out, sum} = {1'b0, a ~^ b};
        default: {c_out, sum} = a + b + c_in;
    endcase
endmodule
```

（2）测试模块

```
`include "ALU.v"
module tb_ALU();
    // define signals
    reg a, b, c_in;
    reg [2:0] oper;
    wire c_out, sum;
    // always module
    initial begin
        {a, b, c_in} = 3'b111;
        #1 oper = 3'b000;
        #1 oper = 3'b001;
        #1 oper = 3'b010;
        #1 oper = 3'b011;
        #1 oper = 3'b100;
        #1 oper = 3'b101;
        #1 oper = 3'b110;
        #1 oper = 3'b111;
        #1 {a, b, c_in} = 3'b101;
        #1 oper = 3'b000;
        #1 oper = 3'b001;
        #1 oper = 3'b010;
        #1 oper = 3'b011;
        #1 oper = 3'b100;
        #1 oper = 3'b101;
        #1 oper = 3'b110;
        #1 oper = 3'b111;
        #1 $stop;
    end
    // example of ALU
    ALU example(c_out, sum, oper, a, b, c_in );
    // monitor the process
    initial $monitor("it's at time %d, oper=%b, a,b,c_in=%b, sum=%b, c_out=%b", $time, oper, {a,b,c_in}, sum, c_out);
endmodule
```

（3）测试波形图：如果很多，可以提供部分波形内容；



（4）显示输出（可选）：如果需要显示输出来说明模块设计的正确性；

```
VSIM 3> run
# it's at time                  0, oper=xxx, a,b,c_in=111, sum=1, c_out=1
# it's at time                  1, oper=000, a,b,c_in=111, sum=1, c_out=1
# it's at time                  2, oper=001, a,b,c_in=111, sum=0, c_out=0
# it's at time                  3, oper=010, a,b,c_in=111, sum=0, c_out=0
# it's at time                  4, oper=011, a,b,c_in=111, sum=1, c_out=0
# it's at time                  5, oper=100, a,b,c_in=111, sum=1, c_out=0
# it's at time                  6, oper=101, a,b,c_in=111, sum=0, c_out=0
# it's at time                  7, oper=110, a,b,c_in=111, sum=0, c_out=0
# it's at time                  8, oper=111, a,b,c_in=111, sum=1, c_out=0
# it's at time                  9, oper=111, a,b,c_in=101, sum=0, c_out=0
# it's at time                 10, oper=000, a,b,c_in=101, sum=0, c_out=1
# it's at time                 11, oper=001, a,b,c_in=101, sum=1, c_out=0
# it's at time                 12, oper=010, a,b,c_in=101, sum=1, c_out=1
# it's at time                 13, oper=011, a,b,c_in=101, sum=1, c_out=0
# it's at time                 14, oper=100, a,b,c_in=101, sum=0, c_out=0
# it's at time                 15, oper=101, a,b,c_in=101, sum=0, c_out=0
# it's at time                 16, oper=110, a,b,c_in=101, sum=1, c_out=0
# it's at time                 17, oper=111, a,b,c_in=101, sum=0, c_out=0
```

（5）设计说明（可选）：如果有需要说明的部分。

我们在这里把不同的操作符用三位二进制编码来代替：000 表示 and；001 表示 substract；010 表示 substract_a；011 表示 or_ab；100 表示 and_ab；101 表示 not_ab；110 表示 exor；111 表示 exnor。

第 13 题

设计一个具有图 7 所示功能的计数器模块。

（1）设计模块

```
module shift_counter(count, clk, reset);
    // define output and input
    output reg [7:0] count=8'b0000_0001;
    input clk, reset;
    reg dir=1'b1;
    // always module
    always @(posedge clk)
    begin
        if(reset) begin  // if reset
            count <= 8'b0000_0001;
            dir <= 1'b1;
        end
        else if(dir) begin        // if shift left
            count[7:0] <= {count[6:0], 1'b0};
            if (count[6])
                dir = 1'b0;
        end
        else begin                // if shift right
            count[7:0] <= {1'b0, count[7:1]};
            if (count[1])
                dir = 1'b1;
        end
    end
endmodule
```

count[7:0]

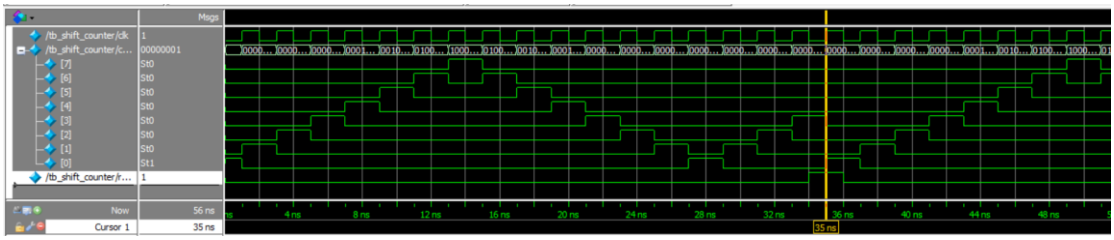
初始状态	0	0	0	0	0	0	0	1
	0	0	0	0	0	0	0	1
	0	0	0	0	0	0	0	1
	0	0	0	0	0	0	0	1
	0	0	0	0	0	0	1	0
	0	0	0	0	0	1	0	0
	0	0	0	0	1	0	0	0
	0	0	0	1	0	0	0	0
t	0	0	1	0	0	0	0	0
	0	1	0	0	0	0	0	0
	1	0	0	0	0	0	0	0
	0	1	0	0	0	0	0	0
	0	0	1	0	0	0	0	0
	0	0	0	1	0	0	0	0
	0	0	0	0	1	0	0	0
	0	0	0	0	0	1	0	0
	0	0	0	0	0	0	1	0
	0	0	0	0	0	0	0	1

图 7、习题 13 计数器计数模式

（2）测试模块

```
`include "shift_counter.v"
module tb_shift_counter();
    // define signals
    reg clk=1'b0;
    wire [7:0] count;
    reg reset;
    // create a clock
    always #1 clk = ~clk;
    // initial module
    initial
    begin
        reset = 1'b0;
        #34 reset = 1'b1;
        #2 reset = 1'b0;
        #20 $stop;
    end
    // example of shift_counter
    shift_counter example(count, clk, reset);
    // monitor the process
    initial $monitor("it's at time %d, count=%b, reset=%b", $time, count, reset);
endmodule
```

（3）测试波形图：如果很多，可以提供部分波形内容；



（4)显示输出(可选)：如果需要显示输出来说明模块设计的正确性；

```
VSIM 43> run
# it's at time                    0, count=00000001, reset=0
# it's at time                    1, count=00000010, reset=0
# it's at time                    3, count=00000100, reset=0
# it's at time                    5, count=00001000, reset=0
# it's at time                    7, count=00010000, reset=0
# it's at time                    9, count=00100000, reset=0
# it's at time                   11, count=01000000, reset=0
# it's at time                   13, count=10000000, reset=0
# it's at time                   15, count=01000000, reset=0
# it's at time                   17, count=00100000, reset=0
# it's at time                   19, count=00010000, reset=0
# it's at time                   21, count=00001000, reset=0
# it's at time                   23, count=00000100, reset=0
# it's at time                   25, count=00000010, reset=0
# it's at time                   27, count=00000001, reset=0
# it's at time                   29, count=00000010, reset=0
# it's at time                   31, count=00000100, reset=0
# it's at time                   33, count=00001000, reset=0
# it's at time                   34, count=00001000, reset=1
# it's at time                   35, count=00000001, reset=1
# it's at time                   36, count=00000001, reset=0
# it's at time                   37, count=00000010, reset=0
# it's at time                   39, count=00000100, reset=0
# it's at time                   41, count=00001000, reset=0
```

（5）设计说明（可选）：如果有需要说明的部分。

我们从波形图可以看出，在时刻 34 我们进行了 reset，而在时钟上条沿 35 的时候，输出才复位。

第 14 题

图 8 是一 256×8bits 的 SRAM 的框图，din[7:0]是 8 条数据输入线，dout[7:0]是 8 条数据输出线。wr 为写控制线，rd 为读控制线，cs 为片选控制线。其工作方式为：

（1）当 cs＝1，wr 信号由低变高（上升沿）时，din 上的数据将写入由 addr 所指定的存储单元；

（2）当 cs =1，rd＝0 时，由 addr 所指定的存储单元的内容将从 dout 的数据线上输出。

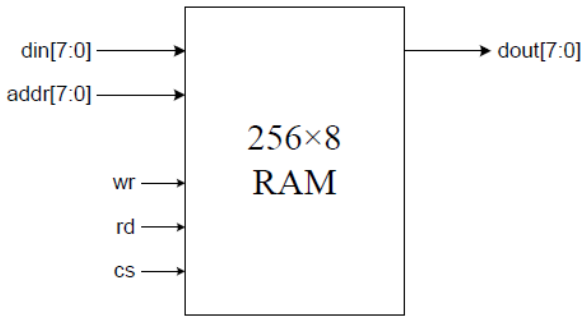


图 8、256×8bits 的 SRAM 框图

（1）设计模块

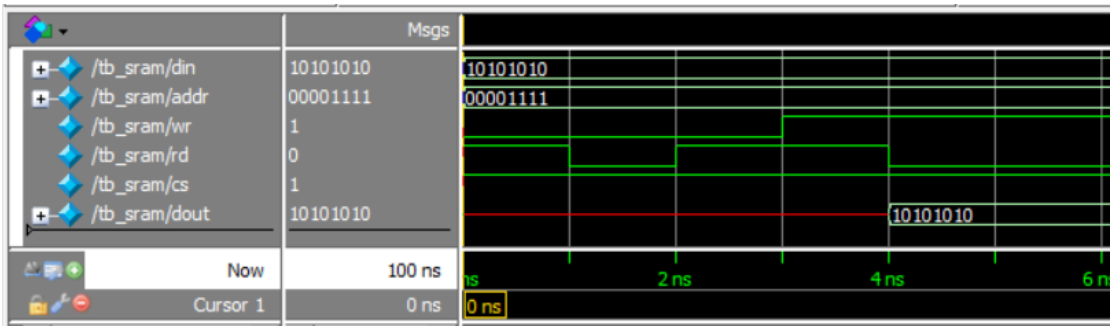
```
module sram( dout, din, addr, wr, rd, cs );
    // define output and input
    input wr, cs, rd;
    input [7:0] din;
    input [7:0] addr;
    output reg [7:0] dout;
    reg [7:0] ram [255:0];
    // always module
    always @(posedge cs or posedge wr or negedge rd)
    begin
        if (cs & wr) // din -> ram
            ram[addr] <= din;
        else;
        if(cs & ~rd) // ram -> dout
            dout <= ram[addr];
        else ;
    end
endmodule
```

（2）测试模块

```
`include "sram.v"
module tb_sram();
    // define signals
    reg [7:0] din;
    reg [7:0] addr;
    reg wr, rd, cs;
    wire [7:0] dout;
    //initial module
    initial
    begin
        cs = 1'b1;
        wr = 1'b0;
        rd = 1'b1;
        addr = 8'b0000_1111;
        din = 8'b1010_1010;
        #1 rd = 1'b0;
        #1 rd = 1'b1;
        #1 wr = 1'b1;
        #1 rd = 1'b0;

    end
    // example of sram
    sram example(dout, din, addr, wr, rd, cs);
    // monitor the process
    initial $monitor("it's at time %d, din=%b, ", $time, din,
    "cs=%b, wr=%b, rd=%b, addr=%b, dout=%b", cs, wr, rd, addr, dout);
endmodule
```

（3）测试波形图：如果很多，可以提供部分波形内容；



（4）显示输出（可选）：如果需要显示输出来说明模块设计的正确性；

```
VSIM 47> run
# it's at time                    0, din=10101010, cs=1, wr=0, rd=1, addr=00001111, dout=xxxxxxxx
# it's at time                    1, din=10101010, cs=1, wr=0, rd=0, addr=00001111, dout=xxxxxxxx
# it's at time                    2, din=10101010, cs=1, wr=0, rd=1, addr=00001111, dout=xxxxxxxx
# it's at time                    3, din=10101010, cs=1, wr=1, rd=1, addr=00001111, dout=xxxxxxxx
# it's at time                    4, din=10101010, cs=1, wr=1, rd=0, addr=00001111, dout=10101010
```

（5）设计说明（可选）：如果有需要说明的部分。

在输出结果中显示，时刻 0：初始状态；时刻 1：尝试读状态，但是没有读成功；时刻 2：关闭 rd 信号；时刻 3：从 din 写入 ram；时刻 4：从 ram 读出道 dout。

第 15 题

设计一个序列检测器，在时钟的每个下降沿检查数据。当检测到输入序列 din 中出现 1101 或 0110 时，输出 flag 为 1，否则输出为 0。

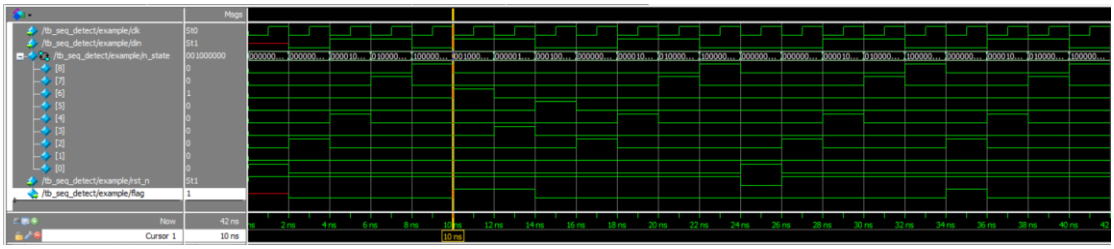
（1）设计模块

```
module seq_detect(output reg flag, input din, clk, rst_n);
    // define output and input
    parameter IDLE=9'b0_0000_0001, A=9'b0_0000_0010, B=9'b0_0000_0100,
    C=9'b0_0000_1000, D=9'b0_0001_0000, E=9'b0_0010_0000, F=9'b0_0100_0000,
    G=9'b0_1000_0000, H=9'b1_0000_0000;
    reg [8:0] n_state;
    // always module
    always @(negedge clk)
    begin                   // if rst_n
        if (~rst_n) begin
            n_state <= IDLE;
            flag <= 0;
        end
        else begin
            case(n_state) // state changes by time
                IDLE: n_state <= (din) ? A : B;
                A: n_state <= (din) ? C : B;
                B: n_state <= (din) ? D : B;
                C: n_state <= (din) ? A : E;
                D: n_state <= (din) ? G : B;
                E: n_state <= (din) ? F : B;
                F: n_state <= (din) ? C : B;
                G: n_state <= (din) ? C : H;
                H: n_state <= (din) ? F : B;
                default: n_state <= IDLE;
            endcase // define flag
            flag = ((n_state==F)|(n_state==H)) ? 1 : 0;
        end
    end
    // monitor the process
    initial $monitor("it's at time %d, din=%b, rst_n=%b, flag=%b, n_state=%b",$time, din, rst_n, flag, n_state);
endmodule
```

（2）测试模块

```
`include "seq_detect.v"
module tb_seq_detect();
    // define signals
    reg clk=0, rst_n, din;
    wire flag;
    // create clock
    always #1 clk = ~clk;
    // initial module
    initial
    begin // create some signals
        rst_n = 1;
        #2 din = 0; #2 din = 1; #2 din = 1;
        #2 din = 0; #2 din = 1; #2 din = 1;
        #2 din = 0; #2 din = 0; #2 din = 1;
        #2 din = 1; #2 din = 0; #2 rst_n = 0;
        #2 rst_n = 1; #2 din = 1; #2 din = 1;
        #2 din = 0; #2 din = 0; #2 din = 1;
        #2 din = 1; #2 din = 0; #2 $stop;
    end
    // example of seq_detect
    seq_detect example(flag, din, clk, rst_n);
endmodule
```

（3）测试波形图：如果很多，可以提供部分波形内容；

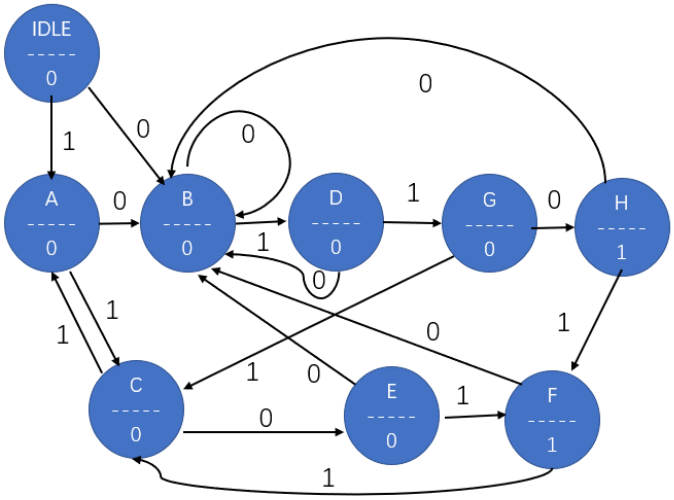


（4）显示输出（可选）：如果需要显示输出来说明模块设计的正确性；

```
VSIM 54> run
# it's at time                0, din=x, rst_n=1, flag=x, n_state=000000001
# it's at time                2, din=0, rst_n=1, flag=0, n_state=000000100
# it's at time                4, din=1, rst_n=1, flag=0, n_state=000010000
# it's at time                6, din=1, rst_n=1, flag=0, n_state=010000000
# it's at time                8, din=0, rst_n=1, flag=0, n_state=100000000
# it's at time               10, din=1, rst_n=1, flag=1, n_state=001000000
# it's at time               12, din=1, rst_n=1, flag=1, n_state=000001000
# it's at time               14, din=0, rst_n=1, flag=0, n_state=000100000
# it's at time               16, din=0, rst_n=1, flag=0, n_state=000000100
# it's at time               18, din=1, rst_n=1, flag=0, n_state=000010000
# it's at time               20, din=1, rst_n=1, flag=0, n_state=010000000
# it's at time               22, din=0, rst_n=1, flag=0, n_state=100000000
# it's at time               24, din=0, rst_n=0, flag=0, n_state=000000001
# it's at time               26, din=0, rst_n=1, flag=0, n_state=000000100
# it's at time               28, din=1, rst_n=1, flag=0, n_state=000010000
# it's at time               30, din=1, rst_n=1, flag=0, n_state=010000000
# it's at time               32, din=0, rst_n=1, flag=0, n_state=100000000
# it's at time               34, din=0, rst_n=1, flag=1, n_state=000000100
# it's at time               36, din=1, rst_n=1, flag=0, n_state=000010000
# it's at time               38, din=1, rst_n=1, flag=0, n_state=010000000
# it's at time               40, din=0, rst_n=1, flag=0, n_state=100000000
```

（5）设计说明（可选）：如果有需要说明的部分。

Moore 型状态自动机



第 16 题

设计一个能在串行输入比特流中检测到序列 10101010 的状态机。这里，输入序列的到达方式为最低有效位（LSB）先到，即：第一个 0 先到，然后是 1，接着，又是第二个 0，…，等等。当检测到输入序列 din 中出现 10101010 时，输出 Ready 为 1，否则输出为 0。

（1）设计模块

```
module moore(output reg flag, input din, clk, rst);
    // define output and input
    parameter IDLE=9'b0_0000_0001, A=9'b0_0000_0010, B=9'b0_0000_0100,
    C=9'b0_0000_1000, D=9'b0_0001_0000, E=9'b0_0010_0000, F=9'b0_0100_0000,
    G=9'b0_1000_0000, H=9'b1_0000_0000;
    reg [8:0] state;
    // signals module
    always @(posedge clk, posedge rst)
    begin
        if (rst) begin      // if reset
            flag <= 0;
            state <= IDLE;
        end
        else begin          // state changes
            flag <= (state==H) ? 1 : 0;
            case(state)
               IDLE: state <= (din) ? IDLE : A;
               A: state <= (din) ? B : A;
               B: state <= (din) ? IDLE : C;
               C: state <= (din) ? D : A;
               D: state <= (din) ? IDLE : E;
               E: state <= (din) ? F : A;
               F: state <= (din) ? IDLE : G;
               G: state <= (din) ? H : A;
               H: state <= (din) ? IDLE : G;
               default: state <= IDLE;
            endcase
        end
    end
    // monitor the process
    initial $monitor("it's at time %d, din=%b, rst=%b, flag=%b, state=%b",
            clk, din, rst, flag, state);
endmodule
```

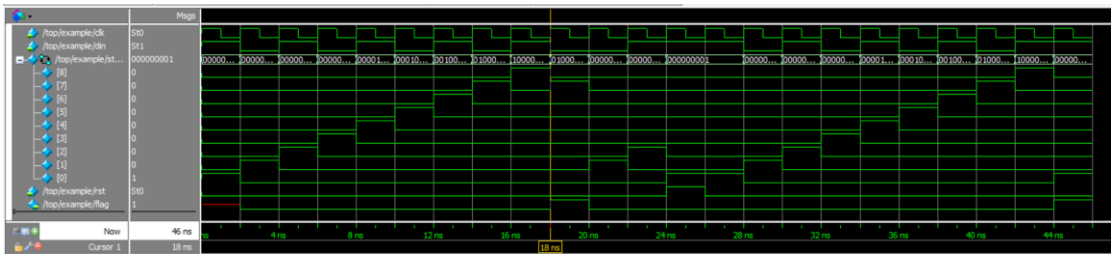
```
module mealy(output reg flag, input din, clk, rst);
    // define output and input
    parameter IDLE=8'b0000_0001, A=8'b0000_0010, B=8'b0000_0100,
    C=8'b0000_1000, D=8'b0001_0000, E=8'b0010_0000, F=8'b0100_0000,
    G=8'b1000_0000;
    reg [7:0] p_state,n_state;
    // always module
    always @(posedge clk or posedge rst)
        if (rst) begin            // if reset
            flag <= 0;
            p_state <= IDLE;
        end
        else p_state <= n_state;
    always @(*) begin
       case(p_state)              // state changes
           IDLE: n_state = (din) ? IDLE : A;
           A: n_state = (din) ? B : A;
           B: n_state = (din) ? IDLE : C;
           C: n_state = (din) ? D : A;
           D: n_state = (din) ? IDLE : E;
           E: n_state = (din) ? F : A;
           F: n_state = (din) ? IDLE : G;
           G: n_state = (din) ? F : A;
           default: n_state = IDLE;
        endcase                   // flag
        flag = ((p_state==G) && (din==1)) ? 1 : 0;
    end
    // monitor the process
    initial $monitor("it's at time %d, din=%b, rst=%b, flag=%b, state=%b",
            $time, din, rst, flag, n_state);
endmodule
```

（2）测试模块

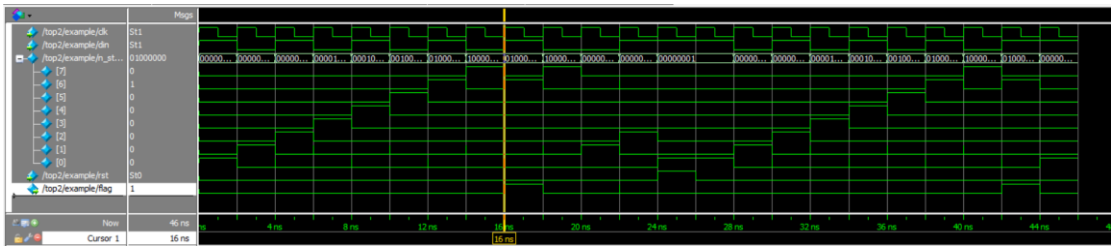
```
`include "moore.v"
`include "mealy.v"
module top();
    // define signals
    reg clk=1, rst_n, din;
    wire flag1, flag2;
    // create clock
    always #1 clk = ~clk;
    // initial module
    initial
    begin // create some signals
        rst_n = 0; din = 1;
        #2 din = 0; #2 din = 1; #2 din = 0;
        #2 din = 1; #2 din = 0; #2 din = 1;
        #2 din = 0; #2 din = 1; #2 din = 0;
        #2 din = 0; #2 din = 1; #2 rst_n = 1;
        #2 rst_n = 0; #2 din = 0; #2 din = 1;
        #2 din = 0; #2 din = 1; #2 din = 0;
        #2 din = 1; #2 din = 0; #2 din = 1;
        #2 $stop;
    end
    // example of moore and meanly
    moore example1(flag1, din, clk, rst_n);
    mealy example2(flag2, din, clk, rst_n);
endmodule
```

（3）测试波形图：如果很多，可以提供部分波形内容；

Moore 型



Mealy 型



（4）显示输出（可选）：如果需要显示输出来说明模块设计的正确性；

Moore 型

```
VSIM 66> run
# it's at time                  0, din=1, rst=0, flag=x, state=000000001
# it's at time                  2, din=0, rst=0, flag=0, state=000000010
# it's at time                  4, din=1, rst=0, flag=0, state=000000100
# it's at time                  6, din=0, rst=0, flag=0, state=000001000
# it's at time                  8, din=1, rst=0, flag=0, state=000010000
# it's at time                 10, din=0, rst=0, flag=0, state=000100000
# it's at time                 12, din=1, rst=0, flag=0, state=001000000
# it's at time                 14, din=0, rst=0, flag=0, state=010000000
# it's at time                 16, din=1, rst=0, flag=0, state=100000000
# it's at time                 18, din=0, rst=0, flag=1, state=010000000
# it's at time                 20, din=0, rst=0, flag=0, state=000000010
# it's at time                 22, din=1, rst=0, flag=0, state=000000100
# it's at time                 24, din=1, rst=1, flag=0, state=000000001
# it's at time                 26, din=1, rst=0, flag=0, state=000000001
# it's at time                 28, din=0, rst=0, flag=0, state=000000010
# it's at time                 30, din=1, rst=0, flag=0, state=000000100
# it's at time                 32, din=0, rst=0, flag=0, state=000001000
# it's at time                 34, din=1, rst=0, flag=0, state=000010000
# it's at time                 36, din=0, rst=0, flag=0, state=000100000
# it's at time                 38, din=1, rst=0, flag=0, state=001000000
# it's at time                 40, din=0, rst=0, flag=0, state=010000000
# it's at time                 42, din=1, rst=0, flag=0, state=100000000
# it's at time                 44, din=1, rst=0, flag=1, state=000000001
```

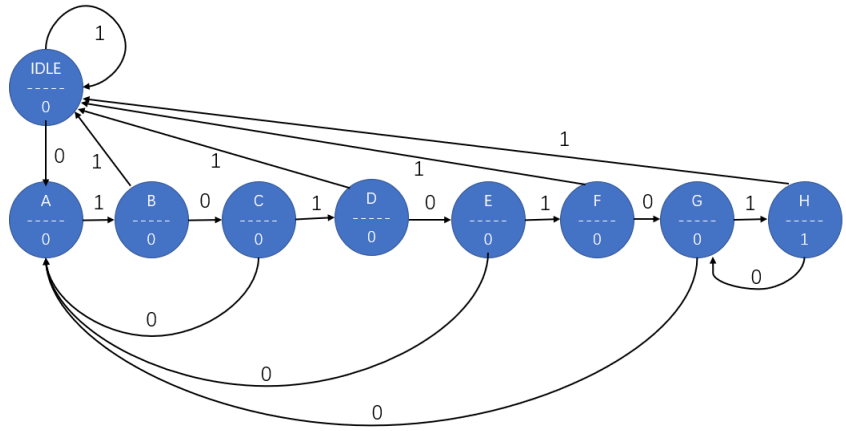
Mealy 型

```
VSIM 69> run
# it's at time                  0, din=1, rst=0, flag=0, state=00000001
# it's at time                  2, din=0, rst=0, flag=0, state=00000010
# it's at time                  4, din=1, rst=0, flag=0, state=00000100
# it's at time                  6, din=0, rst=0, flag=0, state=00001000
# it's at time                  8, din=1, rst=0, flag=0, state=00010000
# it's at time                 10, din=0, rst=0, flag=0, state=00100000
# it's at time                 12, din=1, rst=0, flag=0, state=01000000
# it's at time                 14, din=0, rst=0, flag=0, state=10000000
# it's at time                 16, din=1, rst=0, flag=1, state=01000000
# it's at time                 18, din=0, rst=0, flag=0, state=10000000
# it's at time                 20, din=0, rst=0, flag=0, state=00000010
# it's at time                 22, din=1, rst=0, flag=0, state=00000100
# it's at time                 24, din=1, rst=1, flag=0, state=00000001
# it's at time                 26, din=1, rst=0, flag=0, state=00000001
# it's at time                 28, din=0, rst=0, flag=0, state=00000010
# it's at time                 30, din=1, rst=0, flag=0, state=00000100
# it's at time                 32, din=0, rst=0, flag=0, state=00001000
# it's at time                 34, din=1, rst=0, flag=0, state=00010000
# it's at time                 36, din=0, rst=0, flag=0, state=00100000
# it's at time                 38, din=1, rst=0, flag=0, state=01000000
# it's at time                 40, din=0, rst=0, flag=0, state=10000000
# it's at time                 42, din=1, rst=0, flag=1, state=01000000
# it's at time                 44, din=1, rst=0, flag=0, state=00000001
```

（5）设计说明（可选）：如果有需要说明的部分。

从波形图或者输出图中可以看出，moore 型状态自动机在时间点 18 时，输出 flag 为 1，而 mealy 型状态自动机在时间点 16 时，输出 flag 态为 1。

Moore 型：



Mealy 型：