Verilog 代码设计.

```
// Moore_101 检测
module Moore101(
    input wire clk,
    input wire rst_n,
    input wire cin,
    output reg detected
);

parameter S0 = 2'b00;
parameter S1 = 2'b01;
parameter S2 = 2'b10;
parameter S3 = 2'b11;
```

```
reg [1:0] cur_st , next_st;
always @ (posedge clk or
          negedge rst_n) begin
    if (!rst_n) cur_st <= S0;
    else cur_st <= next_st;
end //状态寄存器

always @ (*) begin
    case (cur_st)
        S0: next_st = (cin == 1'b1) ?
            S1 : S0;
        S1 : next_st = (cin == 1'b0) ?
            S2 : S1;
        S2 : next_st = (cin == 1'b1) ?
            S3 : S0;
```

```
            S3: next_st = (cin == 1'b1) ?
                    S1 : S2;
            default : next_st = S0;
        endcase
    end

    always @ (*) begin
        detected = (cur_st == S3) ?
                        1'b1 : 1'b0;
    end  // 摩尔型输出只和状态有关.

endmodule.
```

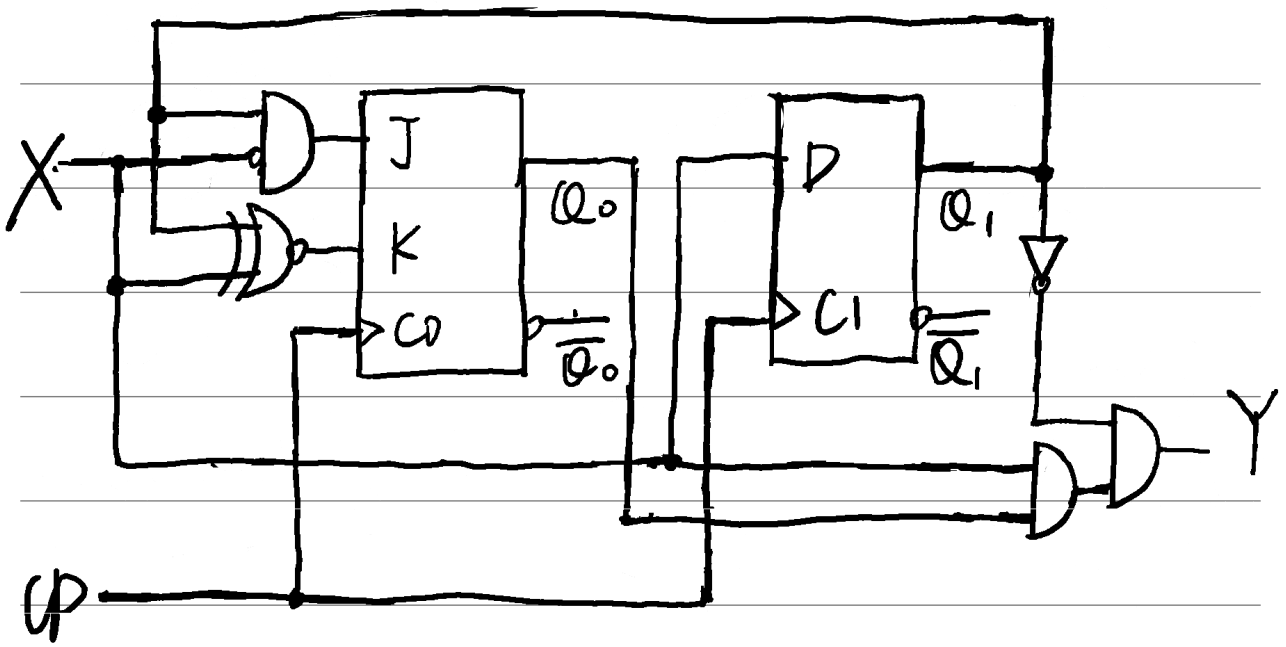
// . Mealy_101检测.

```
module Mealy101 (
    input wire clk,
    input wire rst_n,
    input wire din,
    output reg detected
);
    parameter S0 = 2'b00;
    parameter S1 = 2'b01;
    parameter S2 = 2'b10;

    reg [1:0] cur_st, next_st;

    always @(posedge clk or
                negedge rst_n) begin
```

```
        if(!rst_n) cur_st <= S0;
        else cur_st <= next_st;
    end.
end

    always @(*) begin // @(cin or cur_st)
        case(cur_st)
            S0: begin
                next_st = cin ? S1 : S0;
                detected = 1'b0;
            end
            S1: begin.
                next_st = (!cin) S2 : S1;
                detected = 1'b0;
            end
```

```
        S2: begin
            next_st = cin ? S1 : S0;
            detected = cin ? 1'b1 : 1'b0;
        end.
        default : begin.
            next_st = S0; .
            dectected = 1'b0;
        end
    endcase
end
endmodule
```

用JK、D触发器检测101序列.




```
module JK_D1(
    input  wire clk,
    input  wire rst_n,
    input  wire cin,
    output wire detected
);
```

```
wire Q0, Q1;      //输入
wire J0, K0, D1;  //输出.

JK_FF JK0(.clk(clk), .rst_n(rst_n),
          .J(J0), .K(K0), .Q(Q0));
D_FF D1(.clk(clk), .rst_n(rst_n),
        .D(D1), .Q(Q1));

assign J0 = ~cin & Q1;
assign K0 = cin ^ Q1;
assign D1 = cin;
assign detected = Q0 & ~Q1 & cin;

endmodule
```

```
// 4位计数器.
module counter (
    input wire load,    // 加载指定数据
    input wire en,      // 使能时计数.
    input wire clk,
    input wire rst_n,
    output reg q,
    input wire [3:0] cin,
    output reg [3:0] cout
);

    always @(posedge clk or
                negedge rst_n) begin
        if (!rst_n) begin
            cout <= 4'b0000;
```

```
            q <= 1'b0;
        end else if (load) begin
            cout <= cin
        end else if (en) begin
            if (cout == 4'b1111) begin
                cout <= 4'b0000;
                q <= 1'b1;
            end else begin
                cout <= cout + 1;
            end
        end.
    end.
endmodule
```

```
//异步清零D触发器.
module DFF(
    input wire clk,
    input wire rst_n,
    input wire D,
    output reg Q
);
    always @(posedge clk or
                negedge rst_n) begin
        if(!rst_n) Q <= 1'b0;
        else Q <= D;
    end
endmodule. //异步复位，同步时钟.
```

```
// 同步D触发器.
module DFF (
        input wire clk,
        input wire rst_n,
        input wire D,
        output reg Q
);
    always @(posedge clk) begin.
        if(!rst_n) Q<=1'b0;
        else Q<=D;.
    end.
endmodule.
    // 要等待时钟才能复位.
```

```
//异步D触发器（无时钟）
module DFF(
    input wire en,
    input wire D,
    output reg Q
);
    always @(en or d) begin
      if(en) Q=D;      //无效时锁存
    end
endmodule.
```

```
//同步清零的移位寄存器.
module shift_register (
    input wire clk,
    input wire rst,
    input wire cin,
    output wire cout
);
    reg [7:0] shift_reg; //8位寄存器

    always @(posedge clk)
        if (rst) begin
            shift_reg <= 8'b00000000;
        end else begin
            shift_reg <= {shift_reg[6:0],cin};
        end
```

```
    end
    assign cout = shift_reg[7];
endmodule
```

```
// D锁存器：
 module D_latch (
      input wire clk,
      input wire D,
      output wire Q
);
      always @(*) begin
          if (clk) Q = D;  //电平敏感
      end
/*    always @(posedge clk) begin
          Q <= D;          //边沿敏感.
      end.                 // 触发器.
 */
endmodule
```

```
// 16个 8位数据的存储器
module memory (
    input wire clk,
    input wire we,  //写使能
    input wire [3:0] addr,
    input wire [7:0] cin,
    output wire [7:0] cout
);
  reg [7:0] mem [0:15];
  //16个8位存储单元,[0:15]地址范围
                    深度

  always @(posedge clk) begin
      if(we) mem[addr] <= cin;
  end.
```

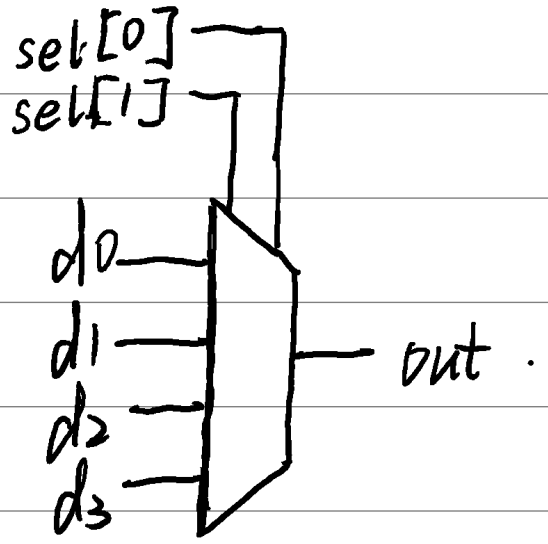
```
always @ (*) begin
    cout = mem [addr];
end.
/* 写操作（同步） \
\ 读操作(异步) .*/

/*同步读写                                    \
always @ (posedge clk) begin
    if(we) mem [addr] <= cin;
    cout <= mem[addr];
end
\                                           */
endmodule
```

```
// n位移位寄存器
module shift_n #(
    parameter n = 16
)(
    input wire [n-1:0] R,
    output reg [n-1:0] Q,
    input wire L,
    input wire w,
    input wire clk
);

always @(posedge clk) begin
    if(L) Q <= R;
    else begin
        Q[n-2:0] <= Q[n-1:1];
```

```
            Q[n-1] <= w;  //右移.
        end.
    end.
endmodule.

/*  Q <= {w, Q[n-1:1]};.                    \
\       拼接更直观                          */
```

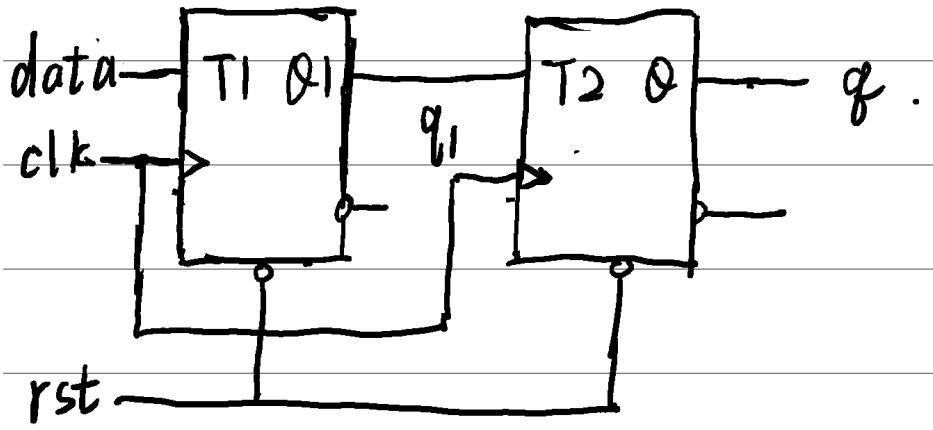
```
// 4选1多路选择器
module mux4 (
    input wire d0,
    input wire d1,
    input wire d2,
    input wire d3,
    input wire [1:0] sel,
    output reg out
);
    always @(sel) begin
        case (sel)
            2'b00: out = d0;
            2'b01: out = d1;
            2'b10: out = d2;
            2'b11: out = d3;
```

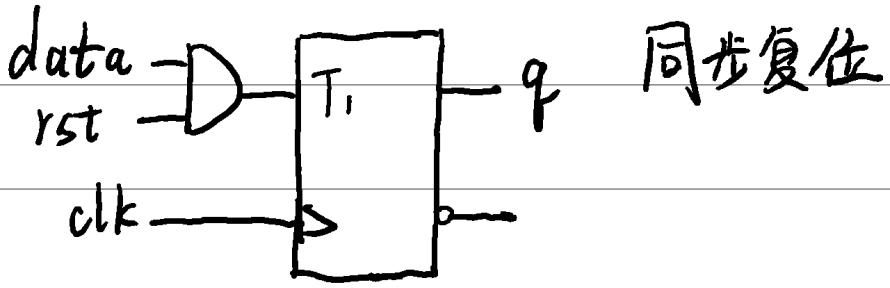
```
        endcase
    end
endmodule
```



```
//异步复位串联T触发器.
module tff2 (
    input wire data, clk, rst,
    output reg q
);
  reg q1 ;

  always @(posedge clk or negedge
           rst) begin.
    if(!rst) begin
           q1 <= 0;
           q <= 0;
    end else begin.
           q1 <= q1 ^ data;
           q <= q ^ q1;
```

```
        end.
    end.
endmodule.
```





```
//奇偶校验.
module check_parity (
        input wire [31:0] data,
        output wire even,
        output wire odd
);
    reg check;  integer i;
    always @(*) begin.
        check = 1'b0;
        for (i=0; i<32; i=i+1) begin
            check = check ^ data[i];
        end.
    end.
    assign  even = check, odd = ~check;
endmodule
```

/* 归约操作符

assign even = ^ data

assign odd = ~ ^data.

*/

```
//4位数值比较.
module cmp(
    input wire [3:0] A,
    input wire [3:0] B,
    output wire A_bigger,
    output wire B_bigger,
    output wire equal
);
    assign A_bigger = (A > B);
    assign B_bigger = (B > A);
    assign equal =(A == B);
endmodule
```