NEXYS2 案例(76)

郑利浩，等

2010-12-28

赵峰

目录

例 1：2 输入逻辑门

```
//Example 1:2-input gates
Module gates2(
Input wire a;
Input wire b;
Output wire[5:0] z
);
assign    z[5]=a&b;
assign    z[4]=～(a&b);
assign    z[3]=a|b;
assign    z[2]=  ～(a|b);
assign    z[1]=aˆb;
assign    z[0]=a～ˆb;

endmodule
```

例 2：多输入逻辑门

```
//Example 2: 4-input gates
module gates4(
input wire [4:1] x,
output wire [6:1] z
);

assign z[6] = &x;
assign z[5] = ~&x;
assign z[4] = |x;
assign z[3] = ~|x;
assign z[2] = ^x;
assign z[1] = ~^x;

endmodule
```

```
//Example 2: 4-input gates - top level
module    gates4_top(
input wire [3:0] sw,
output wire [5:0] ld
);

gates4 X4 (.x(sw),
           .z(ld)
);

endmodule
```

例 3：多数表决电路

```
// Example 3: majority4
module majority4 (
input wire    a,
input wire    b,
input wire    c,
input wire    d,
output wire    f
);

assign f = b & c & d | a & c & d | a & b & d | a & b & c;

endmodule

// Example 3: majority4    –    top-level
module    majority4_top (
input wire [3:0] sw,
output wire [0:0] ld
);

majority4 M1 ( .a(sw[3]),
          .b(sw[2]),
          .c(sw[1]),
          .d(sw[0]),
          .f(ld[0])
);

endmodule
```

例 4：二位比较器

```
// Example 4: comp2bit
module comp2bit (
input wire [1:0] a,
input wire [1:0] b,
output wire a_eq_b,
output wire a_gt_b,
output wire a_lt_b
);

assign a_eq_b = ~b[1] &  ~b[0] &  ~a[1] &  ~a[0]
                      | ~b[1] &  b[0] & ~a[1] &  a[0]
                      |  b[1] & ~b[0] &  a[1] & ~a[0]
                      |  b[1] &  b[0] &  a[1] &  a[0];

assign a_gt_b = ~b[1] & a[1]
                      | ~b[1] & ~b[0] &  a[0]
                      | ~b[0] &  a[1] &  a[0];

assign a_lt_b = b[1] & ~a[1]
                    | b[1] &  b[0] & ~a[0]
                    | b[0] & ~a[1] & ~a[0];
endmodule



// Example 4: 2-bit comparator _top-level
module comp2bit_top(
input wire [3:0] sw,
output wire [2:0] ld
);

comp2bit U1 (.a(sw[3:2]),
        .b(sw[1:0]),
        .a_eq_b(ld[1]),
        .a_gt_b(ld[0]),
        .a_lt_b(ld[2])
);
endmodule
```

例 5：映射报告（Map Report）

```
Design Summary
--------------
Number of errors:          0
Number of warnings:        0
Logic Utilization:
   Number of 4 input LUTs:                  6 out of     9,312       1%
Logic Distribution:
   Number of occupied Slices:               3 out of     4,656       1%
      Number of Slices containing only related logic:        3 out of           3 100%
      Number of Slices containing unrelated logic:           0 out of           3     0%
         *See NOTES below for an explanation of the effects of unrelated logic.
   Total Number of 4 input LUTs:            6 out of     9,312       1%
   Number of bonded IOBs:                   8 out of       232       3%
```

例 1 映射报告中的 Design summary

```
Design Summary
--------------
Number of errors:          0
Number of warnings:        0
Logic Utilization:
   Number of 4 input LUTs:                  1 out of     9,312       1%
Logic Distribution:
   Number of occupied Slices:               1 out of     4,656       1%
      Number of Slices containing only related logic:        1 out of           1 100%
      Number of Slices containing unrelated logic:           0 out of           1     0%
         *See NOTES below for an explanation of the effects of unrelated logic.
   Total Number of 4 input LUTs:            1 out of     9,312       1%
   Number of bonded IOBs:                   5 out of       232       2%
```

例 3 映射报告中的 Design summary

例 6：2 选 1 多路选择器：if 语句

```
// 例 6a：使用逻辑方程实现的 2 选 1 MUX
module mux21a (
input wire a,
input wire b,
input wire s,
output wire y
);

assign y = ~ s & a | s & b;

endmodule
```

```
// 例 6b：使用 if 语句实现的 2 选 1 MUX
module mux21b (
input wire a,
input wire b,
input wire s,
output reg y
);

always @ (a, b, s)
    if (s == 0)
        y = a;
    else
        y = b;

endmodule

// 例 6c：使用 if 语句实现的 2 选 1 MUX
module mux21c (
input wire a,
input wire b,
input wire s,
output reg y
);

always @ ( * )
    if (s == 0)
        y = a;
    else
        y = b;

endmodule

// 例 6d：使用“？”实现的 2 选 1 MUX
module mux21d (
input wire a,
input wire b,
input wire s,
output wire y
);

assign y = s ? b : a;
endmodule
```

```
// 例 6e：2 选 1 MUX
module mux21_top (
input wire [1:0] sw,
input wire [0:0] btn,
output wire [0:0] ld
);

mux21c M1 (.a(sw[0]),
           .b(sw[1]),
           .s(btn[0]),
           .y(ld[0])
);

endmodule
```

例 7：4 选 1 多路选择器：模块例化

```
// 例 7a：使用模块例化的 4 选 1 MUX
module mux41 (
input wire [3:0] c,
input wire [1:0] s,
output wire z
);

// 内部信号
wire v ;        // output of mux M1
wire w;         // output of mux M2
// 模块例化
mux21c M1 ( .a (c[0] ),
            .b ( c[1] ),
            .s ( s[0] ),
            .y ( v )
);
mux21c M2 ( .a (c[2] ),
            .b ( c[3] ),
            .s ( s[0] ),
            .y ( w )
);
mux21c M3 ( .a ( v ),
            .b ( w ),
            .s ( s[1] ),
            .y ( z )
);
```

```
Endmodule

// 例 7b：使用逻辑方程实现 4 选 1 MUX
module mux41b (
input wire [3:0] c,
input wire [1:0] s,
output wire z
);

assign z = ~ s[1] & ~ s[0] & c[0]
          | ~ s[1] &    s[0] & c[1]
          |    s[1] & ~ s[0] & c[2]
          |    s[1] &    s[0] & c[3];
endmodule
```

例 8：4 选 1 多路选择器：case 语句

```
// 例 8：使用 case 语句实现 4 选 1 MUX
module mux41c (
input wire [3:0] c,
input wire [1:0] s,
output reg z
);

always @ ( * )
    case (s)
        0: z = c[0];
        1: z = c[1];
        2: z = c[2];
        3: z = c[3];
        default: z = c[0];
    endcase
endmodule
```

例 9：一个 4 位 2 选 1 多路选择器

```
// 例 9a：使用逻辑方程实现 4 位 2 选 1 MUX
module mux24a (
input wire [3:0] a,
input wire [3:0] b,
input wire s,
output wire [3:0] y
);

assign y = {4{~s}} & a | {4{s}} & b;
endmodule


// 例 9b：使用 if 语句实现 4 位 2 选 1 MUX
module mux24b (
input wire [3:0] a,
input wire [3:0] b,
input wire s,
output reg [3:0] y
);

always @ (*)
      if (s ==0)
            y = a;
      else
            y = b;
endmodule


// 例 9c：使用“？”运算符实现 4 位 2 选 1 MUX
module mux24c (
input wire [3:0] a,
input wire [3:0] b,
input wire s,
output wire [3:0] y
);

assign y = s ? b : a;

endmodule
```

```
// 例 9d：4 位 2 选 1 MUX
module mux24_top (
input wire [7:0] sw,
input wire [0:0] btn,
output wire [3:0] ld
);

mux24b QM1 ( .a (sw[3:0]),
             .b (sw[7:4]),
             .s (btn[0]),
             .y (ld)
);

endmodule
```

例 10：通用多路选择器：参数（Parameter）

```
// 例 10a：使用 parameter 实现通用的 2 选 1 MUX
module mux2g
# ( parameter N =4 )
( input wire [N-1:0] a,
  input wire [N-1:0] b,
  input wire s,
  output reg [N-1:0] y
);

always @ ( * )
      if (s == 0)
            y = a;
      else
            y = b;

endmodule


// 例 10b：使用 paramter 实现 8 选 1 MUX
module mux28 (
input wire [7:0] a,
input wire [7:0] b,
input wire s,
output wire [7:0] y
);
```

```
mux2g # (
      .N(8))
M8 ( .a(a),
       .b(b),
       .s(s),
       .y(y)
);

endmodule
```

例 11：毛刺（Glitches）

```
// 例 11a：带有毛刺信号的 2 选 1 MUX
module mux21g (
input wire a,
input wire b,
input wire s,
output wire y
);

// 内部信号
wire nots;
wire c;
wire d;

assign #10 nots = ~s;
assign #10 c = nots & a;
assign #10 d = s & b;
assign #10 y = c | d;

endmodule


// 例 11b：毛刺信号被消除的 2 选 1 MUX
module mux21gr (
input wire a,
input wire b,
input wire s,
output wire y
);
```

```
// 内部信号
wire nots;
wire c;
wire d;
wire e;
assign #10 nots = ~s;
assign #10 c = nots & a;
assign #10 d = s & b;
assign #10 e = a & b;
assign #10 y = c | d | e;
endmodule
```

例 12：7 段译码器：逻辑方程

```
// 例 12：十六进制 – 7 段译码器；a – g 低电平有效
module hex7seg_le (
input wire [3:0] x,
output wire [6:0] a_to_g
);

assign a_to_g[6] = ~x[3] & ~x[2] & ~x[1] &  x[0]          // a
                 | ~x[3] &  x[2] & ~x[1] & ~x[0]
                 |  x[3] &  x[2] & ~x[1] &  x[0]
                 |  x[3] & ~x[2] &  x[1] &  x[0];
assign a_to_g[5] =  x[2] & x[1] & ~x[0]                   // b
                 |  x[3] & x[1] &  x[0]
                 | ~x[3] & x[2] & ~x[1] &  x[0]
                 |  x[3] & x[2] & ~x[1] & ~x[0];

assign a_to_g[4] = ~x[3] & ~x[2] &  x[1] & ~x[0]          // c
                 |  x[3] &  x[2] &  x[1]
                 |  x[3] &  x[2] & ~x[0];

assign a_to_g[3] = ~x[3] & ~x[2] & ~x[1] &  x[0]          // d
                 | ~x[3] &  x[2] & ~x[1] & ~x[0]
                 |  x[3] & ~x[2] &  x[1] & ~x[0]
                 |  x[2] &  x[1] &  x[0];

assign a_to_g[2] = ~x[3] &  x[0]                          // e
                 | ~x[3] &  x[2] & ~x[1]
                 | ~x[2] & ~x[1] &  x[0];
assign a_to_g[1] = ~x[3] & ~x[2] &  x[0]                  // f
```

```
                        | ~x[3] & ~x[2] &    x[1]
                        | ~x[3] &    x[1] &    x[0]
                        |    x[3] &    x[2] & ~x[1] & x[0];

assign a_to_g[0] = ~x[3] & ~x[2] & ~x[1]                    // g
                        |    x[3] &    x[2] & ~x[1] & ~x[0]
                        | ~x[3] &    x[2] &    x[1] & x[0];

endmodule
```

例 13：7 段译码器：case 语句

```
// 例 13a：十六进制 – 7 段解码管；a – g 低电平有效
module hex7seg (
input wire [3:0] x,
output reg [6:0] a_to_g
);

always @ ( * )
    case (x)
        0: a_to_g = 7'b0000001;
        1: a_to_g = 7'b1001111;
        2: a_to_g = 7'b0010010;
        3: a_to_g = 7'b0000110;
        4: a_to_g = 7'b1001100;
        5: a_to_g = 7'b0100100;
        6: a_to_g = 7'b0100000;
        7: a_to_g = 7'b0001111;
        8: a_to_g = 7'b0000000;
        9: a_to_g = 7'b0000100;
        'hA: a_to_g = 7'b0001000;
        'hB: a_to_g = 7'b1100000;
        'hC: a_to_g = 7'b0110001;
        'hD: a_to_g = 7'b1000010;
        'hE: a_to_g = 7'b0110000;
        'hF: a_to_g = 7'b0111000;
        default: a_to_g = 7'b0000001;    // 0
    endcase
endmodule
```

```
// 例 13b：hex7seg_top
module hex7seg_top (
input wire [3:0] sw,
output wire [6:0] a_to_g,
output wire [3:0] an,
output wire dp
);

assign an = 4'b0000;          // all digits on
assign dp = 1;                // dp off
hex7seg D4 ( .x (sw) ,
                .a_to_g ( a_to_g )
);
endmodule
```

例 14：复用 7 段显示管

```
// 例 14：复用 7 段显示管
module mux7seg (
input wire [3:0] btn,
output reg [6:0] a_to_g,
output wire [3:0] an,
output wire dp
);
wire [15:0] x;
wire [1:0] s;
reg [3:0] digit;

assign x = 'h1234;
assign an = ~btn;
assign s[1] = btn[2] | btn[3];
assign s[0] = btn[1] | btn[3];
assign dp = 1;

// 4 位 4 选 1 MUX: mux44
always @ ( * )
    case (s)
        0: digit = x[3:0];
        1: digit = x[7:4];
        2: digit = x[11:8];
        3: digit = x[15:12];
        default: digit = x[3:0];
```

```
        endcase

// 7-segment decoder: hex7seg
always @ ( * )
        case (digit)
                0: a_to_g = 7'b0000001;
                1: a_to_g = 7'b1001111;
                2: a_to_g = 7'b0010010;
                3: a_to_g = 7'b0000110;
                4: a_to_g = 7'b1001100;
                5: a_to_g = 7'b0100100;
                6: a_to_g = 7'b0100000;
                7: a_to_g = 7'b0001111;
                8: a_to_g = 7'b0000000;
                9: a_to_g = 7'b0000100;
                'hA: a_to_g = 7'b0001000;
        'hB: a_to_g = 7'b1100000;
                'hC: a_to_g = 7'b0110001;
                'hD: a_to_g = 7'b1000010;
                'hE: a_to_g = 7'b0110000;
                'hF: a_to_g = 7'b0111000;
                default: a_to_g = 7'b0000001;    // 0
        endcase

        endmodule
```

例 15：7 段显示管：x7seg 和 x7segb

```
//例 15a：x7seg：显示 4 位的十六进制数
module x7seg (
input wire [15:0] x,
input wire clk,
input wire clr,
output reg [6:0] a_to_g,
output reg [3:0] an,
output wire dp
);
wire [1:0] s;
reg [3:0] digit;
wire [3:0] aen;
reg [19:0] clkdiv;
assign dp = 1;
```

```
assign s = clkdiv[19:18];           // count every 5.2 ms
assign aen = 4' b1111;                 // enable all digits

// 4 位  4 选 1 MUX: mux44
always @ ( * )
      case (s)
            0: digit = x[3:0];
            1: digit = x[7:4];
            2: digit = x[11:8];
            3: digit = x[15:12];
            default: digit = x[3:0];
      endcase

// 7 段数码管：hex7seg
always @ ( * )
      case (digit)
            0: a_to_g = 7'b0000001;
            1: a_to_g = 7'b1001111;
            2: a_to_g = 7'b0010010;
            3: a_to_g = 7'b0000110;
            4: a_to_g = 7'b1001100;
            5: a_to_g = 7'b0100100;
            6: a_to_g = 7'b0100000;
            7: a_to_g = 7'b0001111;
            8: a_to_g = 7'b0000000;
            9: a_to_g = 7'b0000100;
            'hA: a_to_g = 7'b0001000;
            'hB: a_to_g = 7'b1100000;
            'hC: a_to_g = 7'b0110001;
            'hD: a_to_g = 7'b1000010;
            'hE: a_to_g = 7'b0110000;
            'hF: a_to_g = 7'b0111000;
            default: a_to_g = 7'b0000001;    // 0
      endcase

// Digit select: ancode
always @ ( * )
      begin
      an = 4'b1111;
            if (aen[s] == 1)
                  an[s] = 0;
      end

//  时钟分频器
```

```
always @ (posedge clk or posedge clr)
      begin
            if (clr == 1)
                  clkdiv <= 0;
            else
                  clkdiv <= clkdiv + 1;
      end
      endmodule


// 例 15b：x7seg_top
module x7seg_top (
input wire clk,
input wire [3:3] btn,
output wire [6:0] a_to_g,
output wire [3:0] an,
output wire dp
);

wire [15:0] x;

assign x = 'h1234;            // test display value

x7seg X1 ( .x (x) ,
                .clk(clk),
                .clr(btn[3]),
                .a_to_g(a_to_g),
                .an(an),
                .dp(dp)
  );

      endmodule

// 例 15c：x7segb - Display 7-seg with leading blanks
module x7segb (
input wire [15:0] x,
input wire clk,
input wire clr,
output reg [6:0] a_to_g,
output reg [3:0] an,
output wire dp
);

wire [1:0] s;
```

```
reg [3:0] digit;
wire [3:0] aen;
reg [19:0] clkdiv;

assign dp = 1;
assign s = clkdiv[19:18];          // count every 5.2 ms
// set aen[3:0] for leading blanks
assign aen[3] = x[15] | x[14] | x[13] | x[12];
assign aen[2] = x[15] | x[14] | x[13] | x[12]
                | x[11] | x[10] | x[9] | x[8];
assign aen[1] = x[15] | x[14] | x[13] | x[12]
                | x[11] | x[10] | x[9] | x[8]
                | x[7] | x[6] | x[5] | x[4];
                assign aen[0] = 1;    // digit 0 always on

// 4 位 4 选 1 MUX: mux44
always @ ( * )
    case (s)
        0: digit = x[3:0];
        1: digit = x[7:4];
        2: digit = x[11:8];
        3: digit = x[15:12];
        default: digit = x[3:0];
    endcase

// 7 段解码管：hex7seg
always @ ( * )
    case (digit)
        0: a_to_g = 7'b0000001;
        1: a_to_g = 7'b1001111;
        2: a_to_g = 7'b0010010;
        3: a_to_g = 7'b0000110;
        4: a_to_g = 7'b1001100;
        5: a_to_g = 7'b0100100;
        6: a_to_g = 7'b0100000;
        7: a_to_g = 7'b0001111;
        8: a_to_g = 7'b0000000;
        9: a_to_g = 7'b0000100;
        'hA: a_to_g = 7'b0001000;
        'hB: a_to_g = 7'b1100000;
        'hC: a_to_g = 7'b0110001;
        'hD: a_to_g = 7'b1000010;
        'hE: a_to_g = 7'b0110000;
        'hF: a_to_g = 7'b0111000;
```

```
            default: a_to_g = 7'b0000001;    // 0
        endcase

// Digit select
always @ ( * )
        begin
            an = 4'b1111;
            if (aen[s] == 1)
                an[s] = 0;
        end

// Clock divider
always @ (posedge clk or posedge clr)
        begin
            if (clr == 1)
                clkdiv <= 0;
            else
                clkdiv <= clkdiv + 1;
        end
       endmodule

// 例 15d：x7seg_top
module x7segb_top (
input wire clk,
input wire [3:0] btn,
input wire [7:0] sw,
output wire [6:0] a_to_g,
output wire [3:0] an,
output wire dp
);
wire [15:0] x;

// 串联开关和后 3 个按钮
assign x = {sw, btn[2:0], 5'b01010};      // digit 0 = A

x7segb X2 ( .x (x) ,
                .clk(clk),
                .clr(btn[3]),
                .a_to_g(a_to_g),
                .an(an),
                .dp(dp)
  );

        endmodule
```

例 16：使用 Verilog 任务（Task）的 4 位比较器

```
// 例 16a：使用一个任务（task）的 4-位比较器
module comp4t (
input wire [3:0] x,
input wire [3:0] y,
output reg gt,
output reg eq,
output reg lt
);
// 内部变量
reg [4:0] G;
reg [4:0] L;
reg [4:1] E;
integer i;

always @ ( * )
begin
    G[0] = 0;
    L[0] = 0;
    for (i=0; i<4; i = i + 1)
    comp1bit(x[i], y[i], G[i], L[i], G[i+1], L[i+1], E[i+1]);
    gt = G[4];
    eq = E[4];
    lt = L[4];
end

task comp1bit(
input x,
input y,
input Gin,
input Lin,
output Gout,
output Lout,
output Eout
);
begin
    Gout = x & ~y | x & Gin | ~y & Gin;
    Eout = ~x & ~y & ~Gin & ~Lin | x & y & ~Gin & ~Lin;
    Lout = ~x & y | ~x & Lin | y & Lin;
end
endtask

endmodule
```

```
// 例 16b：omp4t_top
module comp4t_top (
input wire [7:0] sw,
output wire [2:0] ld
);

comp4t C4 ( .x(sw[7:4]),
            .y(sw[3:0]),
            .gt(ld[2]),
            .eq(ld[1]),
            .lt(ld[0])
);

endmodule
```

例 17：使用关系运算符的 N 位比较器

```
// 例 17：使用关系运算符实现的 N 位比较器
module comp
# (parameter N = 8)
(input wire [N-1:0] x,
 input wire [N-1:0] y,
 output reg gt,
 output reg eq,
 output reg lt
);

always @ ( * )
begin
    gt = 0;
    eq = 0;
    lt = 0;
    if (x > y)
        gt = 1;
    if (x == y)
        eq =1;
    if (x < y)
        lt = 1;
end
endmodule
```

例 18：3-8 译码器：逻辑方程

```
// 例 18a：3-to-8 译码器：逻辑方程
module decode38a (
input wire [2:0] a,
output wire [7:0] y
);

assign y[0] = ~a[2] & ~a[1] & ~a[0];
assign y[1] = ~a[2] & ~a[1] &  a[0];
assign y[2] = ~a[2] &  a[1] & ~a[0];
assign y[3] = ~a[2] &  a[1] &  a[0];
assign y[4] =  a[2] & ~a[1] & ~a[0];
assign y[5] =  a[2] & ~a[1] &  a[0];
assign y[6] =  a[2] &  a[1] & ~a[0];
assign y[7] =  a[2] &  a[1] &  a[0];

endmodule


// 例 18b: decode38a_top
module decode38a_top (
input wire [2:0] sw,
output wire [7:0] ld
);
decode38a   D1( .a(sw),
                .y(ld)
);

endmodule
```

例 19：3-8 译码器：for 循环

```
// 例 19: 3-8 译码器
module decode38b (
input wire [2:0] a,
output reg [7:0] y
);
integer i;

always @ ( * )
    for (i = 0; i <= 7; i = i+1)
```

```
        if (a == i)
            y[i] = 1;
        else
            y[i] = 0;
endmodule
```

例 20：8-3 编码器：逻辑方程

```
// 例 20：8-3 译码器：逻辑方程
module encode83a(
input wire [7:0] x,
output wire [2:0] y,
output wire valid
);

assign y[2] = x[7] | x[6] | x[5] | x[4];
assign y[1] = x[7] | x[6] | x[3] | x[2];
assign y[0] = x[7] | x[5] | x[3] | x[1];
assign valid = | x;

endmodule


// 例 20：encode83a_top
module encode83a_top (
input wire [7:0] sw,
output wire [2:0] ld,
output wire dp
);

wire valid;
assign dp = ~valid;

encode83a    E1( .x(sw),
                 .y(ld),
                 .valid(valid)
);

endmodule
```

例 21：8-3 编码器：for 循环

```
// 例 21：8-3 编码器：for 循环
module encode83b (
input wire [7:0] x,
output reg [2:0] y,
output reg valid
);
integer i;

always @ ( * )
begin
    y = 0;
    valid = 0;
    for (i = 0; i <= 7; i = i+1)
        if (x[i] == 1)
            begin
                y = i;
                valid = 1;
            end
end
endmodule
```

例 22：8-3 优先编码器

```
// 例 22：8-3 优先编码器
module pencode83 (
input wire [7:0] x,
output reg [2:0] y,
output reg valid
);

integer i;

always @ ( * )
begin
    y = 0;
    valid = 0;
    for (i = 0; i <= 7; i = i+1)
        if (x[i] ==1)
            begin
```

```
                y = i;
                valid = 1;
            end
end

endmodule


// 例 22：pencode83_top
module pencode83_top (
input wire [7:0] sw,
output wire [2:0] ld,
output wire dp
);

wire valid;
assign dp = ~valid;

pencode83    P1( .x(sw),
                 .y(ld),
                 .valid(valid)
);

endmodule
```

例 23：4 位二进制-BCD 码转换器：逻辑方程

```
// Example 23: 4-bit binary-to-BCD converter
module binbcd4 (
input wire [3:0] b,
output wire [4:0] p
);

assign p[4] = b[3] & b[2] | b[3] & b[1];
assign p[3] = b[3] & ~b[2] & ~b[1];
assign p[2] = ~b[3] & b[2] | b[2] & b[1];
assign p[1] = b[3] & b[2] & ~b[1] | ~b[3] & b[1];
assign p[0] = b[0];

endmodule
```

例 24：8 位二进制-BCD 码转换器

```
// 例 24：8-位二进制-BCD 码转换器
module binbcd8 (
input wire [7:0] b,
output reg [9:0] p
);

// 中间变量
reg [17:0] z;
integer i;
always @ ( * )
    begin
        for (i = 0; i <=17; i = i + 1)
            z[i] = 0;
            z[10:3] = b;     // shift b left 3 places

        repeat (5)                                  // 重复 5 次
        begin
            if (z[11:8] > 4)                        // 如果个位大于 4
                z[11:8] = z[11:8] +3;               // 加 3
            if (z[15:12] > 4)                       // 如果十位大于 4
                z[15:12] = z[15:12] +3;             // 加 3
            z[17:1] = z[16:0];                      // 左移一位
        end
        p = z[17:8];                                // BCD
    end
endmodule

// 例 24：binbcd8_top
module binbcd8_top (
input wire clk,
input wire [3:3] btn,
input wire [7:0] sw,
output wire [7:0] ld,
output wire [6:0] a_to_g,
output wire [3:0] an,
output wire dp
);
wire [15:0] x;
wire [9:0] p;

// 串联 0 和 binbcd8 的输出
assign x = {6' b000000, p};
```

```
// 在 LED 上显示开关的二进制值
assign ld = sw;

// 在 7 段显示管上显示开关的十进制值
binbcd8 B1( .b(sw),
            .p(p)
);

x7segb X2 ( .x(x),
            .clk(clk),
            .clr(btn[3]),
            .a_to_g(a_to_g),
            .an(an),
            .dp(dp)
);

endmodule
```

例 25：4 位二进制到格雷码的转换器

```
// 例 25：二进制到格雷码的转换器
module bin2gray (
input wire [3:0] b,
output wire [3:0] g
);

assign g[3] = b[3];
assign g[2:0] = b[3:1] ^ b[2:0];
endmodule
```

例 26：4 位格雷码到二进制码的转换器

```
// 例 26：格雷码到二进制码的转换器
module gray2bin (
input wire [3:0] g,
output reg [3:0] b
);

integer i;

always @ ( * )
```

```
    begin
        b[3] = g[3];
        for (i = 2; i >= 0; i = i-1)
            b[i] = b[i+1] ^ g[i];
    end

endmodule
```

例 27：四位加法器：逻辑表达式

```
// Example 27: 4-bit adder
module   adder4a(
input wire [3:0]   a,
input wire [3:0]   b,
output wire [3:0]   s,
output wire   cf,
output wire   ovf
);

wire [4:0]   c;

assign c[0]=0;
assign s = a ^ b ^ c[3:0];
assign c[4:1] = a & b | c[3:0] & (a ^ b);
assign cf = c[4];
assign ovf = c[3] ^ c[4];

endmodule

// Example 6.2 : adder4a_top
module adder4a_top(
input wire clk,
input wire [3:3] btn,
input wire [7:0] sw,
output wire [0:0] ld,
output wire [6:0] a_to_g,
output wire [3:0] an,
output wire dp
);

wire [15:0] x;
wire cf;
```

```
wire [3:0] s;

assign x[15:12] = sw[7:4];
assign x[11:8] = sw[3:0];
assign x[7:4] = {3'b000,cf};
assign x[3:0] = s;

adder4a A1 (.a(sw[7:4]),
        .b(sw[3:0]),
        .s(s),
        .cf(cf),
.        ovf(ld[0])
);

x7seg X1 (.x(x),
        .clk(clk),
        .clr(btn[3]),
        .a_to_g(a_to_g),
        .an(an),
        .dp(dp)
);

endmodule
```

例 28：四位加法器：行为描述

```
// Example 28: 4-bit behavioral adder
module adder4b(
input wire [3:0] a,
input wire [3:0] b,
output reg [3:0] s,
output reg cf,
output reg ovf
);
reg [4:0] temp;
always @(*)
    begin
        temp = {1'b0,a}+{1'b0,b};
        s = temp[3:0];
        cf = temp[4];
        ovf = s[3] ^ a[3] ^ b[3] ^ cf;
    end
endmodule
```

例 29：N 位加法器：行为描述

```
// Example 29: N-bit adder
module adder
#(parameter N=8)
( input wire [N-1:0] a,
  input wire [N-1:0] b,
  output reg [N-1:0] y
);

always @(*)
    begin
        y = a + b;
    end
endmodule
```

例 30:四位加/减法器:逻辑表达式

```
// Example 30: 4-bit adder/subtractor
module   addsub4(
input wire [3:0]   a,
input wire [3:0]   b,
input wire    E,
output wire [3:0]   s,
output wire    cf,
output wire    ovf
);

wire [4:0]   c;
wire [3:0]   bx;

assign bx = b ^ {4{E}};
assign c[0]=E;
assign s = a ^ bx ^ c[3:0];
assign c[4:1] = a & bx | c[3:0] & (a ^ bx);
assign cf = c[4];
assign ovf = c[3] ^ c[4];

endmodule
```

例 31：N 位减法器：行为描述

```
// Example 31: N-bit subtractor
module    subtract
#(parameter N=8)
(input wire [N-1:0]    a,
input wire [N-1:0]    b,
output reg [N-1:0]    y
);

always @(*)
    begin
        y = a - b;
    end
endmodule
```

例 32：四位移位器

```
// Example 32: 4-bit shifter
module shift4(
input wire [3:0] d,
input wire [2:0] s,
output reg [3:0] y
);

always@(*)
   case(s)
          0: y = d;                    // noshift
          1: y= {1'b0,d[3:1]};         // shr
          2: y= {d[2:0],1'b0};         // shl
          3: y= {d[0],d[3:1]};          // ror
          4: y= {d[2:0],d[3]};         // rol
          5: y= {d[3],d[3:1]};         // asr
          6: y= {d[1:0],d[3:2]};       // ror2
          7: y= d;                      // noshift
          default: y = d;
   endcase
endmodule
```

例 33：与常数相乘

```
// Example 33: multiply x by 100
module mult100(
input wire [8:0] x,
output wire [15:0] p
);
assign p = {1'b0,x,6'b000000} + {2'b00,x,5'b00000} + {5'b00000,x,2'b00};
endmodule
```

例 34：四位乘法器

```
// Example 34: 4-bit multiplier
module mult4(
input wire [3:0] a,
input wire [3:0] b,
output reg [7:0] p
      );
reg [7:0] pv;
reg [7:0] bp;
integer i;

always @(*)
    begin
        pv = 8'b00000000;
          bp = {4'b0000,b};
          for(i = 0; i <= 3; i= i + 1)
              begin
                  if(a[i]==1)
                      pv = pv + bp;
                    bp = {bp[6:0],1'b0};
               end
        p = pv;
     end

endmodule

// Example 34: mult4_top
module mult4_top(
input wire clk,
input wire [3:3] btn,
```

```
input wire [7:0] sw,
output wire [6:0] a_to_g,
output wire [3:0] an,
output wire dp
);
wire [15:0] x;
wire [7:0] p;

assign x[15:12] = sw[7:4];
assign x[11:8] = sw[3:0];
assign x[7:0] = p;

mult4 M1 (.a(sw[7:4]),
        .b(sw[3:0]),
        .p(p)
);

x7seg X1 (.x(x),
          .clk(clk),
          .clr(btn[3]),
          .a_to_g(a_to_g),
          .an(an),
          .dp(dp)
);

endmodule
```

例 35：用 task 实现 8 位除法电路

```
// Example 35: Division 8/4 = 8:4
module div84(
input wire [7:0] numerator,
input wire [3:0] denominator,
output reg [7:0] quotient,
output reg [3:0] remainder
      );
reg [3:0] remH;
reg [3:0] remL;
reg [3:0] quotH;
reg [3:0] quotL;

always @(*)
    begin
```

```
            div4({1'b0,numerator[7:4]},denominator,quotH,remH);
             div4({remH,numerator[3:0]},denominator,quotL,remL);
             quotient[7:4] = quotH;
             quotient[3:0] = quotL;
             remainder = remL;
      end

task div4(
input [7:0] numer,
input [3:0] denom,
output [3:0] quot,
output [3:0] rem
);
begin : D4
      reg [4:0] d;
      reg [4:0] n1;
      reg [3:0] n2;
begin
           d = {1'b0,denom};
            n2 = numer[3:0];
            n1 = {1'b0,numer[7:4]};
            repeat(4)
                begin
                     n1 = {n1[3:0],n2[3]};    //shl n1:n2
                      n2 = {n2[2:0],1'b0};
                      if(n1>=d)
                         begin
                              n1 = n1 - d;
                               n2[0] = 1;
                         end
                  end
             quot = n2;
             rem = n1[3:0];
       end
end
endtask

endmodule
```

例 36：四位 ALU

```
// Example 36: 4-bit ALU
module alu4(
input wire [2:0] alusel,
input wire [3:0] a,
input wire [3:0] b,
output reg nf,
output reg zf,
output reg cf,
output reg ovf,
output reg [3:0] y
      );
reg [4:0] temp;

always @(*)
   begin
       cf = 0;
         ovf = 0;
         temp = 5'b00000;
       case(alusel)
                3'b000: y=a;
                  3'b001:
                     begin
                         temp = {1'b0,a} + {1'b0,b};
                         y = temp[3:0];
                         cf = temp[4];
                         ovf = y[3]^a[3]^b[3]^cf;
                      end
                  3'b010:
                     begin
                         temp = {1'b0,a} - {1'b0,b};
                         y = temp[3:0];
                         cf = temp[4];
                         ovf = y[3]^a[3]^b[3]^cf;
                      end
                  3'b011:
                     begin
                         temp = {1'b0,b} - {1'b0,a};
                         y = temp[3:0];
                         cf = temp[4];
                         ovf = y[3]^a[3]^b[3]^cf;
                      end
                  3'b100: y = ~a;
```

```
                3'b101: y = a&b;
                3'b110: y = a|b;
                3'b111: y = a^b;
                default: y = a;
        endcase
        nf = y[3];
        if(y==4'b0000)
            zf = 1;
        else
            zf = 0;
    end
endmodule
```

例 37：边沿触发的 D 触发器

```
// 例 37：边沿触发的 D 触发器
module flipflop (
input wire clk,
input wire D,
output wire q,
output wire notq
);
wire f1, f2, f3, f4, f5, f6;

assign #5 f1 = ~ (f4 & f2);
assign #5 f2 = ~ (f1 & f5);
assign #5 f3 = ~ (f6 & f4);
assign #5 f4 = ~ (f3 & clk);
assign #5 f5 = ~ (f4 & clk & f6);
assign #5 f6 = ~ (f5 & D);
assign q = f1;
assign notq = f2;

endmodule
```

例 38：带有置位和清零端的边沿 D 触发器

```
// 例 38：带有清零和置位端的 D 触发器
module flipflopcs (
input wire clk,
input wire D,
input wire set,
input wire clr,
output q,
output notq
);
wire f1, f2, f3, f4, f5, f6;

assign #5 f1 = ~ (f4 & f2 & ~set);
assign #5 f2 = ~(f1 & f5 & ~clr);
assign #5 f3 = ~(f6 & f4 & ~set);
assign #5 f4 = ~(f3 & clk & ~clr);
assign #5 f5 = ~(f4 & clk & f6 & ~set);
assign #5 f6 = ~(f5 & D & ~clr);
assign q = f1;
assign notq = f2;

endmodule
```

例 39：Verilog 中的 D 触发器

```
// 例 39：带清零端的 D 触发器
module Dff (
input wire clk,
input wire clr,
input wire D,
output reg q
);

always @ (posedge clk or posedge clr)
    if (clr == 1)
        q <= 0;
    else
        q <= D;
endmodule
```

例 40：带异步清零和置位端的 D 触发器

```
// 例 40：带置位和清零端的 D 触发器
module Dffsc (
input wire clk,
input wire clr,
input wire set,
input wire D,
output reg q
);

always @ (posedge clk or posedge clr or posedge set)
    if (set == 1)
        q <= 1;
    else if (clr == 1)
        q <= 0;
    else
        q <= D;
endmodule
```

例 41：2 分频计数器

```
// 例 41：二分频计数器
module div2cnt (
input wire clk,
input wire clr,
output reg q0
);

wire D;      // D 触发器的输入

assign D = ~q0;

// D 触发器
always @ (posedge clk or posedge clr)
    if (clr == 1)
        q0 <= 0;
    else
        q0 <= D;
endmodule
```

例 42：一位寄存器

```
// 例 42a: 1 位寄存器
module reg1bit (
input wire load,
input wire clk,
input wire clr,
input wire inp0,
output reg q0
);

wire D;

assign D = q0 & ~load | inp0 & load;

// D 触发器
always @ (posedge clk or posedge clr)
    if (clr == 1)
        q0 <= 0;
    else
        q0 <= D;
endmodule

// 例 42b: 1 位寄存器
module reg1bitb (
input wire load,
input wire clk,
input wire clr,
input wire inp0,
output reg q0
);

// 带 load 信号的 1 位寄存器
always @ (posedge clk or posedge clr)
    if (clr == 1)
        q0 <= 0;
    else if (load == 1)
        q0 <= inp0;
endmodule
```

例 43：4 位寄存器

```
// 例 43: 4 位寄存器
module reg4bit (
input wire load,
input wire clk,
input wire clr,
input wire [3:0] inp,
output reg [3:0] q
);

// 带 load 信号的 4 位寄存器
always @ (posedge clk or posedge clr)
      if (clr == 1)
            q <= 0;
      else if (load == 1)
            q <= inp;
endmodule
```

例 44：N 位寄存器

```
// 例 44：带有异步清零和加载信号的 N 位寄存器
module register
# (parameter N = 8)
( input wire load,
  input wire clk,
  input wire clr,
  input wire [N-1:0] d,
  output reg [N-1:0] q
);

always @ (posedge clk or posedge clr)
      if (clr == 1)
            q <= 0;
      else if (load == 1)
            q <= d;
endmodule
```

例 45：移位寄存器

```
// 例 45：4 位移位寄存器
module ShiftReg (
input wire clk,
input wire clr,
input wire data_in,
output reg [3:0] q
);

// 4 位移位寄存器
always @ (posedge clk or posedge clr)
begin
    if (clr == 1)
        q <= 0;
    else
    begin
        q[3] <= data_in;
        q[2:0] <= q[3:1];
    end
end
endmodule
```

例 46：环形计数器

```
// 例 46：环形计数器
module ring4(
input wire clk,
input wire clr,
output reg [3:0] q
);

// 4 位环形寄存器
always @ (posedge clk or posedge clr)
    begin
        if (clr == 1)
            q <= 1;
        else
            begin
                q[3] <= q[0];
                q[2:0] <= q[3:1];
```

```
            end
    end
endmodule
```

例 47：防抖按钮

```
// 例 47：4 个按钮开关抖动的消除
module debounce4(
input wire [3:0] inp,
input wire cclk,
input wire clr,
output wire [3:0] outp
);

reg [3:0] delay1;
reg [3:0] delay2;
reg [3:0] delay3;

always @ (posedge cclk or posedge clr)
begin
    if (clr == 1)
        begin
            delay1 <= 4'b0000;
            delay2 <= 4'b0000;
            delay3 <= 4'b0000;
        end
    else
        begin
            delay1 <= inp;
delay2 <= delay1;
            delay3 <= delay2;
        end
end

assign outp = delay1 & delay2 & delay3;

endmodule
```

例 48：时钟脉冲

```
// 例 48：时钟脉冲
module clock_pulse (
input wire inp,
input wire cclk,
input wire clr,
output wire outp
);

reg delay1;
reg delay2;
reg delay3;

always @ (posedge cclk or posedge clr)
begin
    if (clr == 1)
        begin
            delay1 <= 0;
            delay2 <= 0;
            delay3 <= 0;
        end
    else
        begin
            delay1 <= inp;
            delay2 <= delay1;
            delay3 <= delay2;
        end
end
assign outp = delay1 & delay2 & ~delay3;

endmodule
```

例 49：3 位计数器

```
// 例 49：3 位计数器
module count3a (
input wire clr,
input wire clk,
output reg [2:0] q
);
wire [2:0] D;
```

```
assign D[2] = ~q[2] & q[1] & q[0]
                | q[2] & ~q[1]
                | q[2] & ~q[0];

assign D[1] = ~q[1] & q[0]
                | q[1] & ~q[0];

assign D[0] = ~q[0];

// 3 个 D 触发器
always @ (posedge clk or posedge clr)
    if (clr == 1)
        q <= 0;
    else
        q <= D;

endmodule


//  例 49b：3 位计数器
module count3b (
input wire clr,
input wire clk,
output reg [2:0] q
);

// 3 位计数器
always @ (posedge clk or posedge clr)
    begin
        if (clr == 1)
            q <= 0;
        else
            q <= q + 1;
    end

endmodule
```

例 50：模-5 计数器

```
//  例 50：模-5 计数器
module mod5cnt (
```

```
input wire clr,
input wire clk,
output reg [2:0] q
);

// 模-5 计数器
always @ (posedge clk or posedge clr)
    begin
        if (clr == 1)
            q <= 0;
        else if (q == 4)
            q <= 0;
        else
            q <= q + 1;
    end

endmodule
```

例 51：N 位计数器

```
// 例 51：N-位计数器
module counter
# (parameter N =8)
(input wire clr,
 input wire clk,
 output reg [N-1:0] q
);

// N-位计数器
always @ (posedge clk or posedge clr)
    begin
        if (clr == 1)
            q <= 0;
        else
            q <= q + 1;
    end

endmodule
```

例 52：时钟分频器：模-10K 计数器

```
// 例 52：时钟分频器
module clkdiv (
input wire mclk,
input wire clr,
output wire clk190,
output wire clk48
);
reg [24:0] q;

// 25 位计数器
always @ (posedge mclk or posedge clr)
    begin
        if (clr == 1)
            q <= 0;
        else
            q <= q + 1;
    end

assign clk190 = q[17];    // 190 Hz
assign clk48 = q[19];     // 47.7 Hz
endmodule


// 例 52：模-10,000 计数器
module mod10Kcnt (
input wire clr,
input wire clk,
output reg [13:0] q
);

// 模-10k 计数器
always @ (posedge clk or posedge clr)
    begin
        if (clr == 1)
            q <= 0;
        else if (q == 9999)
            q <= 0;
        else
            q <= q + 1;
    end
endmodule
```

```
// 例 52: 14-位二进制 – BCD 码转换器
module binbcd14 (
input wire [13:0] b,
output reg [16:0] p
);

// 中间变量
reg [32:0] z;
integer i;

always @ ( * )
    begin
        for (i = 0; i <= 32; i = i+1)
            z[i] = 0;
        z[16:3] = b;    // shift b left 3 places

        repeat (11)                          // 重复 11 次
        begin
            if (z[17:14] > 4)                // 如果个位大于 4
                z[17:14] = z[17:14] + 3;     // 加 3
            if (z[21:18] > 4)                // 如果十位大于 4
                z[21:18] = z[21:18] + 3;     // 加 3
            if (z[25:22] > 4)                // 如果百位大于 4
                z[25:22] = z[25:22] + 3;     // 加 3
            if (z[29:26] > 4)                // 如果千位大于 4
                z[29:26] = z[29:26] + 3;     // 加 3
            z[32:1] = z[31:0];               // 左移一位
        end
    p = z[30:14];                            // BCD 输出
    end
endmodule


// 例 52：x7segbc – Display 7-seg with leading blanks
// 输入时钟信号 cclk 应为 190 Hz
module x7segbc (
input wire [15:0] x,
input wire cclk,
input wire clr,
output reg [6:0] a_to_g,
output reg [3:0] an,
output wire dp
);
reg [1:0] s;
```

```
reg [3:0] digit;
wire [3:0] aen;

assign dp = 1;              // decimal points off
// set aen[3:0] for leading blanks
assign aen[3] = x[15] | x[14] | x[13] | x[12];
assign aen[2] = x[15] | x[14] | x[13] | x[12]
                          | x[11] | x[10] | x[9] | x[8];
assign aen[1] = x[15] | x[14] | x[13] | x[12]
                          | x[11] | x[10] | x[9] | x[8]
                          | x[7] | x[6] | x[5] | x[4];
assign aen[0] = 1;        // digit 0 always on

// Quad 4-to-1 MUX: mux44
always @ ( * )
      case (s)
            0: digit = x[3:0];
            1: digit = x[7:4];
            2: digit = x[11:8];
            3: digit = x[15:12];
            default: digit = x[3:0];
      endcase

// 7 段解码器：hex7seg
always @ ( * )
      case (digit)
            0: a_to_g = 7'b0000001;
            1: a_to_g = 7'b1001111;
            2: a_to_g = 7'b0010010;
            3: a_to_g = 7'b0000110;
            4: a_to_g = 7'b1001100;
            5: a_to_g = 7'b0100100;
            6: a_to_g = 7'b0100000;
            7: a_to_g = 7'b0001111;
            8: a_to_g = 7'b0000000;
            9: a_to_g = 7'b0000100;
            'hA: a_to_g = 7'b0001000;
            'hB: a_to_g = 7'b1100000;
            'hC: a_to_g = 7'b0110001;
            'hD: a_to_g = 7'b1000010;
            'hE: a_to_g = 7'b0110000;
            'hF: a_to_g = 7'b0111000;
            default: a_to_g = 7'b0000001;    // 0
      endcase
```

```
// 数字选择
always @ ( * )
    begin
        an = 4'b1111;
        if (aen[s] == 1)
            an[s] = 0;
    end

// 2-位计数器
always @ (posedge cclk or posedge clr)
    begin
        if (clr ==1)
            s <= 0;
        else
            s <= s + 1;
    end
endmodule


// 例 52: 模-10,000 计数器
module mod10Kcnt_top(
input wire mclk,
input wire [3:3] btn,
output wire [6:0] a_to_g,
output wire [3:0] an,
output wire dp
);

wire [16:0] p;
wire clr, clk48, clk190;
wire [13:0] b;

assign clr = btn[3];

clkdiv U1 ( .mclk(mclk),
            .clr(clr),
            .clk190(clk190),
            .clk48(clk48)
);

mod10Kcnt U2 ( .clr(clr),
                .clk(clk48),
                .q(b)
```

```
);

binbcd14 U3 ( .b(b),
              .p(p)
);

x7segbc U4 ( .x(p[15:0]),
             .cclk(clk190),
             .clr(clr),
             .a_to_g(a_to_g),
             .an(an),
             .dp(dp)
);

endmodule
```

例 53：任意波形的产生

```
// 例 53：Morse 码的字母 A
module morsea (
input wire clr,
input wire clk,
output wire a
);

reg [2:0] q;

// 3 位计数器
always @ (posedge clk or posedge clr)
    begin
        if (clr == 1)
            q <= 0;
        else
            q <= q + 1;
    end

assign a = ~q[1] & q[0] | q[2] & ~q[0];

endmodule
```

例 54：脉冲宽度调制器（PWM）

```
// Example 54: PWM signal
module pwmN
# (parameter N = 4)
(input wire clk,
 input wire clr,
 input wire [N-1:0] duty,
 input wire [N-1:0] period,
 output reg pwm
);

reg [N-1:0] count;

always @ (posedge clk or posedge clr)
      if (clr ==1)
            count <= 0;
      else if (count == period - 1)
            count <= 0;
      else
            count <= count + 1;

always @ ( * )
      if (count < duty)
            pwm <= 1;
      else
            pwm <= 0;
endmodule
```

例 56：把开关数据加载到一个寄存器

```
// Example 56: Store switch settings in register
module sw2reg_top (
input wire mclk,
input wire [3:0] btn,
input wire [7:0] sw,
input wire [7:0] ld,
output wire [6:0] a_to_g,
output wire [3:0] an,
output wire dp
);
```

```
wire [7:0] q;
wire clr, clk190, clkp;
wire [15:0] x;

assign clr = btn[3];
assign x = {8' b00000000, q};
assign ld = sw;

clkdiv U1 ( .mclk(mclk),
            .clr(clr),
            .clk190(clk190)
);
clock_pulse U2 ( .inp(btn[0]),
                 .cclk(clk190),
                 .clr（clr）
                 .outp(clkp)
);
register # ( .N(8))
U3 ( .load(btn[2]),
     .clk(clkp),
     .clr(clr),
     .d(sw),
     .q(q)
);
x7segbc U4 ( .x(x),
             .cclk(clk190),
             .clr(clr),
             .a_to_g(a_to_g),
             .an(an),
             .dp(dp)
);
endmodule
```

例 57：把数据移入一个移位寄存器

```
// 例 57：8 位移位寄存器
module shift_reg8 (
input wire clk,
input wire clr,
input wire din,
output reg [7:0] q
```

```
);

// 8 位移位寄存器
always @ (posedge clk or posedge clr)
    begin
        if (clr ==1)
            q <= 0;
        else
            begin
                q[0] <= din;            // shift in q[0]
                q[7:1] <= q[6:0];
            end
    end
endmodule

// 例 57：Shift in pushbutton data
module shift_reg8_top (
input wire mclk,
input wire [3:0] btn,
output wire [7:0] ld
);
wire clr, clk190, clkp, btn01;

assign clr = btn[3];
assign btn01 = btn[0] | btn[1];

clkdiv U1 ( .mclk(mclk),
            .clr(clr),
            .clk190(clk190)
);
clock_pulse U2 ( .inp(btn01),
                 .cclk(clk190),
                 .clr(clr),
                 .outp(clkp)
);
shift_reg8 U3 ( .clk(clkp),
                .clr(clr),
                .din(btn[1]),
                .q(ld)
);
endmodule
```

例 58：7 段显示管的滚动显示

```
// 7-segment 译码器: hex7seg
always @ ( * )
      case (digit)
            0: a_to_g = 7'b0000001;
            1: a_t0_g = 7'b1001111;
            2: a_t0_g = 7'b0010010;
            3: a_t0_g = 7'b0000110;
            4: a_t0_g = 7'b1001100;
            5: a_t0_g = 7'b0100100;
            6: a_t0_g = 7'b0100000;
            7: a_t0_g = 7'b0001111;
            8: a_t0_g = 7'b0000000;
            9: a_t0_g = 7'b0000100;
            'hA: a_t0_g = 7'b0001000;
            'hB: a_t0_g = 7'b1100000;
            'hC: a_t0_g = 7'b0110001;
            'hD: a_t0_g = 7'b1111110;          // - 划线
            'hE: a_t0_g = 7'b0110000;
            'hF: a_t0_g = 7'b1111111;          // 空白
            default: a_t0_g = 7'b1111111;    // 空白
      endcase


// 例 58: shift_array
module shift_array (
input wire clk,
input wire clr,
output wire [15:0] x
);

reg [0:63] msg_array;
parameter PHONE_NO = 64'h248D656D1490FFFF;

always @ (posedge clk or posedge clr)
begin
      if (clr == 1)
            begin
                  msg_array <= PHONE_NO;
            end
      else
            begin
                  msg_array[0:59] <= msg_array[4:63];
```

```
                msg_array[60:63] <= msg_array[0:3];
            end
end

assign x = msg_array[0:15];

endmodule


// 例 58: x7seg_msg – 显示滚动信息
// 输入 cclk 必须为 190 Hz
module x7seg_msg(
input wire [15:0] x,
input wire cclk,
input wire clr,
output reg [6:0] a_to_g,
output reg [3:0] an,
output wire dp
);

reg [1:0] s;
reg [3:0] digit;
wire [3:0] aen;

assign dp = 1;
assign aen = 4'b1111;      // all digits on

// Quad 4 选 1 MUX：  mux44
always @ ( * )
case (s)
      0: digit = x[3:0];
      1: digit = x[7:4];
      2: digit = x[11:8];
      3: digit = x[15:12];
      default: digit = x[3:0];
endcase

// 7 段解码器：hex7seg
always @ ( * )
case (digit)
      0: a_to_g = 7'b0000001;
      1: a_to_g = 7'b1001111;
      2: a_to_g = 7'b0010010;
      3: a_to_g = 7'b0000110;
```

```
        4: a_to_g = 7'b1001100;
        5: a_to_g = 7'b0100100;
        6: a_to_g = 7'b0100000;
        7: a_to_g = 7'b0001111;
        8: a_to_g = 7'b0000000;
        9: a_to_g = 7'b0000100;
        'hA: a_to_g = 7'b0001000;
        'hB: a_to_g = 7'b1100000;
        'hC: a_to_g = 7'b0110001;
        'hD: a_to_g = 7'b1111110;          // - 划线
        'hE: a_to_g = 7'b0110000;
        'hF: a_to_g = 7'b1111111;          // 空白
        default: a_to_g = 7'b1111111;    // 空白
endcase

// 数字选择
always @ ( * )
begin
        an = 4'b1111;
        if (aen[s] == 1)
                an[s] = 0;
end

// 2 位计数器
always @ (posedge cclk or posedge clr)
begin
        if (clr ==1)
                s <= 0;
        else
           s<=s+1;
end
endmodule
// 例 58: 在 7 段显示管中滚动显示电话号码：BASYS-2 开发板
module scroll_top(
input wire mclk,
input wire [3:3] btn,
output wire [6:0] a_to_g,
output wire [3:0] an,
output wire dp
);

wire clr, clk190, clk3;
wire [15:0] x;
```

```
assign clr = btn[3];
clkdiv U1 ( .mclk(mclk),
            .clr(clr),
            .clk3(clk3),
            .clk190(clk190)
);

shift_array U2 ( .clk (clk3),
                 .clr(clr),
                 .x(x)
);

x7seg_msg U3 ( .x(x),
                 .cclk(clk190),
                 .clr(clr),
                 .a_to_g(a_to_g),
                 .an(an),
                 .dp(dp)
);
    endmodule
```

例 59：Fibonacci 序列

```
// 例 59: Fibonacci 序列
module fib (
input wire clk,
input wire clr,
output wire [13:0] f
);
reg [13:0] fn, fn1;

always @ (posedge clk or posedge clr)
    begin
        if (clr == 1)
            begin
                fn <= 0;
                fn1<= 1;
            end
        else if (fn1 < 9999)
            begin
                fn <= fn1;
                fn1 <= fn + fn1;
```

```
                end
        end
assign f = fn;
        endmodule


// 例 59: Fibonacci 序列: BASYS-2 开发板
module fib_top (
input wire mclk,
input wire [3:3] btn,
output wire [6:0] a_to_g,
output wire [3:0] an,
output wire dp
);

wire [16:0] p;
wire clr, clk3, clk190;
wire [13:0] b;
assign clr = btn[3];

clkdiv U1 ( .mclk(mclk),
            .clr(clr),
            .clk190(clk190),
            .clk3(clk3)
);

fib U2 ( .clk(clk3),
         .clr(clr),
         .f(b)
);

Binbcd14 U3 ( .b(b),
              .p(p)
);
x7seg_msg U4 ( .x(p[15:0]),
               .cclk(clk190),
               .clr(clr),
               .a_to_g(a_to_g),
               .an(an),
               .dp(dp)
);
        endmodule
```

例 60：序列检测器

```
// Example 60a: Detect 1101 with Moore machine
module seqdeta(
input wire clk,
input wire clr,
input wire din,
output reg dout
);
reg [2:0] present_state, next_state;
parameter S0 = 3'b000, S1 = 3'b001, S2 = 3'b010,
          S3 = 3'b011, S4 = 3'b100;      // states

//State registers
always @(posedge clk or posedge clr)
  begin
    if( clr == 1)
      present_state <= S0;
    else
      present_state <= next_state;
  end

//C1 module
always @(*)
  begin
    case(present_state)
      S0: if(din == 1)
            next_state <= S1;
          else
            next_state <= S0;
      S1: if(din == 1)
            next_state <= S2;
            else
            next_state <= S0;
      S2: if(din == 0)
            next_state <= S3;
            else
              next_state <= S2;
      S3: if(din == 1)
            next_state <= S4;
            else
            next_state <= S0;
      S4: if(din == 0)
            next_state <= S0;
```

```
                    else
                    next_state <= S2;
                default: next_state <= S0;
        endcase
end

//C2 module
always @(*)
   begin
        if(present_state == S4)
            dout = 1;
        else
            dout = 0;
   end

endmodule


// Example 60b: Detect 1101 with Mealy machine
module seqdetb(
input wire clk,
input wire clr,
input wire din,
output reg dout
);
reg [1:0] present_state, next_state;
parameter S0 = 3'b00, S1 = 3'b01,
                S2 = 3'b10, S3 = 3'b11;      // states

//State registers
always @(posedge clk or posedge clr)
   begin
        if( clr == 1)
            present_state <= S0;
        else
            present_state <= next_state;
   end

//C1 module
always @(*)
   begin
      case(present_state)
                S0: if(din == 1)
                            next_state <= S1;
```

```
                    else
                            next_state <= S0;
                S1: if(din == 1)
                                next_state <= S2;
                    else
                            next_state <= S0;
                S2: if(din == 0)
                            next_state <= S3;
                     else
                            next_state <= S2;
                 S3: if(din == 1)
                            next_state <= S1;
                     else
                            next_state <= S0;
            default: next_state <= S0;
        endcase
end

//C2 module
always @( posedge clk or posedge clr)
   begin
        if( clr == 1)
            dout <= 0;
        else
            if((present_state == S3) && (din==1))
               dout <= 1;
            else
               dout <= 0;
   end

endmodule


// Example 60: Detect sequence 1101
module seqdeta_top(
input wire mclk,
input wire [3:0] btn,
output wire [0:0] ld
);

wire clr, clk190, clkp, btn01;

assign clr = btn[3];
assign btn01 = btn[0] | btn[1];
```

```
clkdiv U1 (.mclk(mclk),
          .clr(clr),
          .clk190(clk190)
);
clock_pulse U2 (.inp(btn01),
            .cclk(clk190),
            .clr(clr),
            .outp(clkp)
);

seqdeta U3 (.clk(clkp),
            .clr(clr),
            .din(btn[1]),
            .dout(ld[0])
);
endmodule
```

例 61：门锁代码

```
// Example 61: Detect doorlock code from switch settings
module doorlock(
input wire clk,
input wire clr,
input wire [7:0] sw,
input wire [1:0] bn,
output reg pass,
output reg fail
      );
reg [3:0] present_state, next_state;
parameter S0 = 4'b0000, S1 = 4'b0001, S2 = 4'b0010, S3 = 4'b0011,
               S4 = 4'b0100, E1 = 4'b0101, E2 = 4'b0110, E3 = 4'b0111, E4 = 4'b1000;
//State registers
always @(posedge clk or posedge clr)
   begin
       if(clr==1)
           present_state <= S0;
         else
           present_state <= next_state;
   end

//C1 module
```

```
 always @(*)
    begin
         case(present_state)
            S0: if(bn == sw[7:6])
                    next_state <= S1;
                    else
                       next_state <= E1;
            S1: if(bn == sw[5:4])
                    next_state <= S2;
                    else
                       next_state <= E2;
            S2: if(bn == sw[3:2])
                    next_state <= S3;
                    else
                       next_state <= E3;
            S3: if(bn == sw[1:0])
                    next_state <= S4;
                    else
                       next_state <= E4;
            S4: if(bn == sw[7:6])
                    next_state <= S1;
                    else
                       next_state <= E1;
            E1: next_state <= E2;
            E2: next_state <= E3;
            E3: next_state <= E4;
            E4: if(bn == sw[7:6])
                    next_state <= S1;
                    else
                       next_state <= E1;
            default: next_state <= S0;
          endcase
      end

//C2 module
always @(*)
   begin
       if(present_state == S4)
           pass = 1;

else
           pass = 0;
if(present_state == E4)
           fail = 1;
```

```
        else
            fail = 0;
    end

endmodule


// Example 61: Doorlock code from switch settings
module doorlock_top(
input wire mclk,
input wire [7:0] sw,
input wire [3:0] btn,
output wire [5:4] ld
      );

wire clr,clk190,clkp,btn012;
wire [1:0] bn;

assign clr = btn[3];
assign btn012 = btn[0] | btn[1] | btn[2];
assign bn[1] = btn[2];
assign bn[0] = btn[1];

clkdiv U1 (.mclk(mclk),
          .clr(clr),
          .clk190(clk190),
);

clock_pulse U2 (.inp(btn012),
          .cclk(clk190),
          .clr(clr),
          .outp(clkp)
);

doorlock U3 (.clk(clkp),
          .clr(clr),
             .sw(sw),
             .bn(bn),
             .pass(ld[5]),
             .fail(ld[4])
);

endmodule
```

例 62：交通灯

```
//Example 62a: traffic lights
module traffic(
input wire clk,
input wire clr,
output reg [5:0] lights
);
reg [2:0] state;
reg [3:0] count;

parameter S0 = 3'b000, S1 = 3'b001, S2 = 3'b010, //states
          S3 = 3'b011, S4 = 3'b100, S5 = 3'b101;
parameter SEC5 = 4'b1110, SEC1 = 4'b0010;

always @(posedge clk or posedge clr)
  begin
    if(clr==1)
      begin
        state <= S0;
        count <= 0;
      end
    else
      case(state)
        S0: if(count<SEC5)
              begin
                state <= S0;
                count <= count + 1;
              end
            else
              begin
                state <= S1;
                count <= 0;
              end
        S1: if(count<SEC1)
              begin
                state <= S1;
                count <= count + 1;
              end
            else
              begin
                state <= S2;
                count <= 0;
              end
```

```
                S2: if(count<SEC1)
                                state <= S2;
                                  count <= count + 1;
                              end
                             else
                               begin
                                    state <= S3;
                                      count <= 0;
                                end
                 S3: if(count<SEC5)
                           begin
                                state <= S3;
                                  count <= count + 1;
                              end
                             else
                               begin
                                    state <= S4;
                                      count <= 0;
                                end
                 S4: if(count<SEC1)
                           begin
                                state <= S4;
                                  count <= count + 1;
                              end
                             else
                               begin
                                    state <= S5;
                                      count <= 0;
                                end
                 S5: if(count<SEC1)
                           begin
                                state <= S5;
                                  count <= count + 1;
                              end
                             else
                               begin
                                    state <= S0;
                                      count <= 0;
                                end
                    default state <= S0;
            endcase
    end

always @(*)
```

```
   begin
        case(state)
                S0: lights = 6'b100001;
                S1: lights = 6'b100010;
                S2: lights = 6'b100100;
                S3: lights = 6'b001100;
                S4: lights = 6'b010100;
                S5: lights = 6'b100100;
                   default lights = 6'b100001;
         endcase
   end

endmodule


//Example 62b: clock divider
module clkdiv(
input wire clk,
input wire clr,
output wire clk3
);
reg [24:0] q;

//    25-bit counter
always @(posedge clk or posedge clr)
   begin
        if(clr == 1)
           q <= 0;
        else
           q <= q + 1;
   end

assign clk3 = q[24];          // 3 Hz

endmodule


// Example 62: traffic_lights_top
module traffic_lights_top(
input wire mclk,
input wire [3:3] btn,
output wire [7:2] ld
);
wire clk3;
```

```
wire clr;

assign clr = btn[3];

clkdiv U1 (.mclk(mclk),
          .clr(clr),
          .clk3(clk3)
);

traffic U2 (.clk(clk3),
          .clr(clr),
          .lights(ld)
);

endmodule
```

例 63：GCD 算法(1)

```
module gcd1(
input wire [3:0] x,
input wire [3:0] y,
output reg [3:0] gcd
      );
reg [3:0] xs,ys;

always @(*)
      begin
          xs = x;
ys = y;
while(xs != ys)
                begin
                   if(xs < ys)
                       ys = ys - xs;
                    else
                       xs = xs - ys;
                end
             gcd = xs;
       end

endmodule
```

例 64：GCD 算法(2)

```
//Example 64a: gcd_datapath
module gcd_datapath(
input wire clk,
input wire clr,
input wire xmsel,
input wire ymsel,
input wire xld,
input wire yld,
input wire gld,
input wire [3:0] xin,
input wire [3:0] yin,
output wire [3:0] gcd,
output reg eqflg,
output reg ltflg
);
wire [3:0] xmy,ymx,gcd_out;
wire [3:0] x,y,x1,y1;

assign xmy = x - y;
assign ymx = y - x;
always @(*)
    begin
        if(x == y)
            eqflg = 1;
        else
            eqflg = 0;
     end

always @(*)
    begin
        if(x<y)
            ltflg = 1;
        else
            ltflg = 0;
     end

mux2g #(
  .N(4))
M1 (.a(xmy),
  .b(xin),
  .s(xmsel),
  .y(x1)
```

```
);

mux2g #(
   .N(4))
M2 (.a(ymx),
   .b(yin),
   .s(ymsel),
   .y(y1)
);

register #(
   .N(4))
R1 (.load(xld),
   .clk(clk),
   .clr(clr),
   .d(x1),
   .q(x)
);
register #(
     .N(4))
R2 (.load(yld),
   .clk(clk),
   .clr(clr),
   .d(y1),
   .q(y)
);

register #(
   .N(4))
R3 (.load(gld),
   .clk(clk),
   .clr(clr),
   .d(x),
   .q(gcd_out)
);

assign gcd = gcd_out;

endmodule


//Example 64b: gcd_control
module gcd_control(
input wire clk,
```

```
input wire clr,
input wire go,
input wire eqflg,
input wire ltflg,
output reg xmsel,
output reg ymsel,
output reg xld,
output reg yld,
output reg gld
);
reg [2:0] present_state, next_state;

parameter start = 3'b000, input1 = 3'b001, test1 = 3'b010,
                test2 = 3'b011, update1 = 3'b100, update2 = 3'b101,
                    done = 3'b110;    //states
//state registers
always @(posedge clk or posedge clr)
   begin
       if(clr == 1)
           present_state <= start;
        else
           present_state <= next_state;
   end

//C1 module
always @(*)
   begin
       case(present_state)
           start: if(go == 1)
                       next_state = input1;
                   else
                       next_state = start;
             input1: next_state = test1;
             test1: if(eqflg == 1)
                       next_state = done;
                   else
                       next_state = test2;
             test2: if(ltflg == 1)
                       next_state    = update1;
                   else
                       next_state    = update2;
             update1: next_state = test1;
             update2: next_state = test1;
             done: next_state = done;
```

```
            default next_state    = start;
        endcase
    end

//C2 module
  always @(*)
     begin
          xld = 0; yld = 0; gld = 0;
           xmsel = 0; ymsel = 0;
           case(present_state)
                input1:
                     begin xld = 1; yld = 1;
                               xmsel = 1; ymsel = 1;
                      end
update1: yld = 1;
                  update2: xld = 1;
                  done: gld = 1;
                  default ;
           endcase
       end

endmodule


module gcd(
input wire clk,
input wire clr,
input wire go,
input wire [3:0] xin,
input wire [3:0] yin,
output wire [3:0] gcd_out
);
wire eqflg,ltflg,xmsel,ymsel;
wire xld,yld,gld;

gcd_datapath U1
  (.clk(clk),
    .clr(clr),
    .xmsel(xmsel),
    .ymsel(ymsel),
    .xld(xld),
    .yld(yld),
    .gld(gld),
    .xin(xin),
```

```
    .yin(yin),
    .gcd(gcd_out),
    .eqflg(eqflg),
    .ltflg(ltflg)
  );

gcd_control U2
    (.clk(clk),
      .clr(clr),
        .go(go),
        .eqflg(eqflg),
        .ltflg(ltflg),
        .xmsel(xmsel),
        .ymsel(ymsel),
        .xld(xld),
        .yld(yld),
        .gld(gld)
    );

endmodule


//Example 64d: gcd_top
module gcd_top(
input wire mclk,
input wire [3:0] btn,
input wire [7:0] sw,
output wire [7:0] ld,
output wire dp,
output wire [6:0] a_to_g,
output wire [3:0] an
);
wire clk25,clk190,clr;
wire [15:0] x;
wire [3:0] gcds;

assign clr = btn[3];
assign x = {12'h00,gcds};
assign ld = sw;
clkdiv U1 (.mclk(mclk),
    .clk190(clk190),
    .clk25(clk25)
);
```

```
gcd U2 (.clk(clk25),
    .clr(clr),
    .go(btn[0]),
    .xin(sw[7:4]),
    .yin(sw[3:0]),
    .gcd_out(gcds)
);

x7segbc U3 (.x(x),
    .cclk(clk190),
    .clr(clr),
    .a_to_g(a_to_g),
    .an(an),
    .dp(dp)
);

endmodule
```

例 65：整数平方根算法实现

```
//Example 65a: Square root datapath
module SQRTpath(
input wire clk,
input wire reset,
input wire    ald,
input wire    sqld,
input wire    dld,
input wire    outld,
input wire    [7:0] sw,
output reg lteflg,
output wire [3:0] root
);
wire [7:0] a;
wire [8:0] sq,s;
wire [4:0] del,dp2;
wire [3:0] dml;

assign s = sq + {4'b0000,del};          // adder8
assign dp2 = del + 2;
assign dml = del[4:1] - 1;
always @(*)
        begin
```

```
                if(sq <= {1'b0,a})
                    lteflg <= 1;
                else
                    lteflg <= 0;
            end

regr2 #(
        .N(8),
            .BIT0(0),
            .BIT1(0))
aReg (.load(ald),
        .clk(clk),
            .reset(reset),
            .d(sw),
            .q(a)
);

regr2 #(
        .N(9),
            .BIT0(1),
            .BIT1(0))
sqReg (.load(sqld),
        .clk(clk),
            .reset(reset),
            .d(s),
            .q(sq)
);

regr2 #(
        .N(5),
            .BIT0(1),
            .BIT1(1))
delReg (.load(dld),
        .clk(clk),
            .reset(reset),
            .d(dp2),
            .q(del)
);

regr2 #(
.N(4),
        .BIT0(0),
        .BIT1(0),
);
```

```
outReg (.load(outld),
        .clk(clk),
           .reset(reset),
           .d(dml),
           .q(root)
);

endmodule


//Example 65b: N-bit register with reset and load
//Resets to initial value of lowest 2 bits
module regr2
#(parameter N = 4,
parameter BIT0 = 1,
parameter BIT1 = 1)
(input wire load,
  input wire clk,
  input wire reset,
  input wire [N-1:0] d,
  output reg [N-1:0] q
);

always @(posedge clk or posedge reset)
        if(reset == 1)
             begin
                  q[N-1:2] <= 0;
                     q[0] <= BIT0;
                     q[1] <= BIT1;
             end
           else if(load == 1)
                  q <= d;

endmodule


//Example 65C: Square root control
module SQRTctrl(
input wire clk,
input wire clr,
input wire lteflg,
input wire go,
output reg ald,
output reg sqld,
```

```
output reg dld,
output reg outld
);
reg[1:0] present_state, next_state;
parameter start = 2'b00, test = 2'b01, update = 2'b10,
                done = 2'b11;//

// State registers
always @(posedge clk or posedge clr)
   begin
        if(clr == 1)
                present_state <= start;
           else
                present_state <= next_state;
   end

//C1 module
always @(*)
   begin
       case(present_state)
              start: if(go == 1)
                                next_state = test;
                                else
                                   next_state = start;
                test:    if(lteflg == 1)
                                next_state = update;
                                else
                                   next_state = done;
                update: next_state = test;
                done: next_state = done;
default next_state = start;
         endcase
   end

 //C2 module
 always @(*)
    begin
          ald = 0; sqld = 0;
           dld = 0; outld = 0;
         case(present_state)
              start: ald = 1;
                 test: ;
                 update:
                    begin
```

```
                    sqld = 1; dld = 1;
                end
            done: outld = 1;
            default ;
        endcase
    end

endmodule


//Example 65d: Integer Square Root
module SQRT(
input wire clk,
input wire clr,
input wire go,
input wire [7:0] sw,
output wire done,
output wire [3:0] root
);
wire lteflg,ald,sqld,dld,outld;
assign done = outld;

SQRTctrl sqrt1(.clk(clk),
        .clr(clr),
            .lteflg(lteflg),
            .go(go),
            .ald(ald),
            .sqld(sqld),
            .dld(dld),
            .outld(outld)
);

SQRTpath sqrt2 (.clk(clk),
        .reset(clr),
            .ald(ald),
            .sqld(sqld),
            .dld(dld),
            .outld(outld),
            .sw(sw),
            .lteflg(lteflg),
            .root(root)
);

endmodule
```

```
//Example 65e: sqrt_top
module sqrt_top(
input wire mclk,
input wire [3:0] btn,
input wire [7:0] sw,
output wire dp,
output wire [6:0] a_to_g,
output wire [3:0] an
);
wire clk25,clk190,clr,done;
wire [15:0] x;
wire [9:0] p;
wire [3:0] root;
wire [7:0] b,r;

assign clr = btn[3];
assign r = {4'b0000,root};
assign x = {6'b000000,p};
assign ld = sw;
clkdiv U1 (.mclk(mclk),
           .clr(clr),
              .clk190(clk190),
              .clk25(clk25)
);

sqrt U2 (.clk(clk25),
           .clr(clr),
              .go(btn[0]),
              .sw(sw),
              .done(done),
              .root(root)
);

mux2g #(
         .N(8))
U3 (.a(sw),
      .b(r),
      .s(done),
      .y(b)
);

binbcd8 U4 (.b(b),
      .p(p)
);
```

```
x7segbc U5 (.x(x),
        .cclk(clk190),
         .clr(clr),
         .a_to_g(a_to_g),
         .an(an),
         .dp(dp)
);

endmodule
```

例 66：GCD 算法(3)

```
//Example 66a: gcd3
module gcd3(
input wire clk,
input wire clr,
input wire go,
input wire [3:0] xin,
input wire [3:0] yin,
output reg done,
output reg [3:0] gcd
);
reg [3:0] x,y;
reg calc;

always @(posedge clk or posedge clr)
   begin
        if(clr == 1)
             begin
                  x <= 0;
                   y <= 0;
                   gcd <= 0;
                   done <= 0;
                   calc <= 0;
             end
           else
             begin
                  done <= 0;
                   if(go == 1)
                   begin
                   x <= xin;
```

```
                    y <= yin;
                    calc <= 1;
                end
                else
                  begin
                      if(calc == 1)
                          if(x == y)
                              begin
                                  gcd <= x;
                                   done <= 1;
                                   calc <= 0;
                              end
                           else
                              if(x < y)
                                 y <= y - x;
                              else
                                 x <= x - y;
                    end
            end
   end

endmodule


//Example 66b: gcd3_top
module gcd3_top(
input wire mclk,
input wire [3:0] btn,
input wire [7:0] sw,
output wire [7:0] ld,
output wire dp,
output wire [6:0] a_to_g,
output wire [3:0] an
);
wire clk25,clk190,clr,gol,done;
wire [15:0] x;
wire [3:0] gcds,btnd;

assign clr = btn[3];
assign x = {12'h000,gcds};
assign ld = sw;

clkdiv U1 (.mclk(mclk),
           .clr(clr),
```

```
                .clk190(clk190),
                .clk25(clk25)
);

debounce4 U2 (.inp(btn),
            .cclk(clk190),
                .clr(clr),
                .outp(btnd)
);

clock_pulse U3 (.inp(btnd[0]),
            .cclk(clk25),
                .clr(clr),
                .outp(go1)
);

gcd3 U4 (.clk(clk25),
            .clr(clr),
                .go(go1),
                .xin(sw[7:4]),
                .yin(sw[3:0]),
                .done(done),
                .gcd(gcds)
);

x7segbc U5 (.x(x),
            .cclk(clk190),
                .clr(clr),
                .a_to_g(a_to_g),
                .an(an),
                .dp(dp)
);

endmodule
```

例 67：整数平方根(2)

```
//Example 67a: sqirt2
module sqirt2(
input wire clk,
input wire clr,
input wire go,
```

```
input wire [7:0] sw,
output reg done,
output reg [3:0] root
);
reg [7:0] a;
reg [8:0] square;
reg [4:0] delta;
reg calc;

always @(posedge clk or posedge clr)
   begin
      if(clr == 1)
         begin
            a <= 0;
             square <= 0;
             delta <= 0;
             root <= 0;
             done <= 0;
             calc <= 0;
          end
       else
          if(go == 1)
             begin
                 a <= sw;
                 square <= 1;
                 delta <= 3;
                 calc <= 1;
                 done <= 0;
              end
           else
             begin
                 if(calc == 1)
                    if(square > a)
                       begin
                          root <= delta[4:1] - 1;
                           done <= 1;
                           calc <= 0;
                       end
                    else
                       begin
                          square <= square + delta;
                           delta <= delta + 2;
                       end
              end
```

```
   end
endmodule


//Example 67b: sqirt2_top
module sqirt2_top(
input wire mclk,
input wire [3:0] btn,
input wire [7:0] sw,
output wire [7:0] ld,
output wire dp,
output wire [6:0] a_to_g,
output wire [3:0] an
);
wire clk25, clk190, clr, gol, done;
wire [15:0] x;
wire [9:0] p;
wire [3:0] root, btnd;
wire [7:0] b, r;

assign clr = btn[3];
assign r = {4'b0000,root};
assign x = {6'b000000,p};
assign ld = sw;

clkdiv U1 (.mclk(mclk),
           .clr(clr),
              .clk190(clk190),
.clk25(clk25)
);

debounce4 U2 (.inp(btn),
           .cclk(clk190),
              .clr(clr),
              .outp(btnd)
);

clock_pulse U3 (.inp(btnd[0]),
           .cclk(clk25),
              .clr(clr),
              .outp(go1)
);

sqirt2 U4 (.clk(clk25),
```

```
            .clr(clr),
               .go(go1),
               .sw(sw),
               .done(done),
               .root(root)
);

mux2g #(.N(8),)
     U5 (.a(sw),
          .b(r),
           .s(done),
           .y(b)
);

binbcd8 U6 (.b(b),
            .p(p)
);

x7segbc U7 (.x(x),
            .cclk(clk190),
               .clr(clr),
               .a_to_g(a_to_g),,
               .an(an),
               .dp(dp)
);

endmodule
```

例 68： Verilog ROM

```
// 例 68a：ROM
module rom8(
input wire [2:0] addr,
output wire [7:0] M
);

parameter N = 8;                    // no. of bits in rom word
parameter N_WORDS = 8;              // no. of words in rom
reg [N-1:0] rom [0:N_WORDS-1];
parameter data = 64'h00C8F9AF64956CD4;      // left index of data
parameter IXLEFT = N*N_WORDS - 1;
integer i;
```

```
initial
begin
      for (i = 0; i < N_WORDS; i = i + 1)
            rom[i] = data[(IXLEFT - N*i) -:N];
end

assign M = rom[addr];

      endmodule


// 例 68b：rom8_top
module rom8_top (
input wire mclk,
input wire [3:0] btn,
output wire [7:0] ld,
output wire dp,
output wire [6;0] a_to_g,
output wire [3:0] an
);
wire clk25, clk190, clr, go1, done;
wire [15:0] x;
wire [2:0] addr;
wire [7:0] M;
wire [3:0] gcds, btnd;

assign clr = btn[3];
assign x = {12'h000, gcds};
assign ld = M;

clkdiv U1 ( .mclk(mclk),
                .clr(clr),
                .clk190(clk190),
                .clk25(clk25)
);

debounce U2 ( .inp(btn),
                    .cclk(clk190),
                    .clr(clr),
                    .outp(btnd)
);

clock_pulse U3 ( .inp(btnd[0]),
```

```
                    .cclk(clk25),
                    .clr(clr),
                    .outp(go1)
);

gcd3 U4 ( .clk(clk25),
          .clr(clr),
          .go(go1),
          .xin(M[7:4]),
          .yin(M[3:0]),
          .done(done),
          .gcd(gcds)
);

x7segbc U5 ( .x(x),
             .cclk(clk190),
             .clr(clr),
             .a_to_g(a_to_g),
             .an(an),
             .dp(dp)
);

counter #( .N(3))
U6 ( .clr(clr),
     .clk(go1)
     .q(addr)
);

rom8 U7 ( .addr(addr),
          .M(M)
);

    endmodule
```

例 69：分布式 RAM/ROM

```
// 例 69：dist_rom16_top
module dist_rom16_top (
input wire mclk,
input wire [3:0] btn,
output wire [7:0] ld,
output wire [6:0] a_to_g,
```

```
output wire dp,
output wire [3:0] an
);
wire clk25, clk190, clr, go1, done;
wire [15:0] x;
wire [2:0] addr;
wire [7:0] M;
wire [3:0] gcds, btnd;

assign clr = btn[3];
assign x = {12'h000, gcds};
assign ld = M;

clkdiv U1 ( .mclk(mclk),
            .clr(clr),
            .clk190(clk190),
            .clk25(clk25)
);

Debounce4 U2 ( .inp(btn),
               .cclk(clk190),
               .clr(clr),
               .outp(btnd)
);

clock_pulse U3 ( .inp(btnd[0]),
                 .cclk(clk25),
                 .clr(clr),
                 .outp(go1)
);

gcd3 U4 ( .clk(clk25),
          .clr(clr),
          .go(go1),
          .xin(M[7:4]),
          .yin(M[3:0]),
          .done(done),
          .gcd(gcds)
);

x7segbc U5 ( .x(x),
             .cclk(clk190),
             .clr(clr),
             .a_to_g(a_to_g),
```

```
                    .an(an),
                    .dp(dp)
);

counter #( .N(3))
U6 ( .clr(clr),
        .clk(go1)
        .q(addr)
);
dist_rom16 U7 ( .a (addr),          // Bus [3:0]
                        .spo(M)          // Bus [7:0]
);

endmodule
```

例 70：块 RAM/ROM

```
// 例 70：brom8x16_top
module brom8x16_top (
input wire mclk,
input wire [3:0] btn,
output wire [7:0] ld,
output wire [6;0] a_to_g,
output wire dp,
output wire [3:0] an
);
wire clk190, clr, clkp;
wire [15:0] x;
wire [2:0] addr;
assign clr = btn[3];
assign ld = {5'b00000, addr};

clkdiv U1 ( .mclk(mclk),
                .clr(clr),
                .clk190(clk190),
);

clock_pulse U2 ( .inp(btnd[0]),
                        .cclk(clk190),
                        .clr(clr),
                        .outp(clkp)
);
```

```
x7segb U3 ( .x(x),
                .cclk(clk190),
                .clr(clr),
                .a_to_g(a_to_g),
                .an(an),
                .dp(dp)
);

counter #( .N(3))
U4 ( .clr(clr),
      .clk(clkp)
      .q(addr)
);
brom8x16 U5 ( .a (addr),        // Bus [2:0]
                    .clk(clkp),
                    .dout(x)          // Bus [15:0]
);

endmodule
```

例 71：VGA – 条纹显示

```
// 例 71a：vga_640x480
module vga_640x480 (
input wire clk,
input wire clr,
output reg hsync,
output reg vsync,
output reg [9:0] hc,
output reg [9:0] vc,
output reg vidon
);

parameter hpixels = 10'b1100100000;
      // Value of pixels in a horizontal line = 800
parameter vlines = 10'b1000001001;
      // Number of horizontal lines in the display = 521
parameter hbp = 10'b0010010000;
      // 行显示后沿 = 144 (128+16)
parameter hfp = 10'b1100010000;
      // 行显示前沿 = 784 (128+16+640)
```

```
parameter vbp = 10'b0000011111;
        // 场显示后沿 = 31 (2+29)
parameter vfp = 10'b0111111111;
        // 场显示前沿 = 511 (2+29+480)
reg vsenable;       // Enable for the Vertical counter

// 行同步信号计数器
always @ (posedge clk or posedge clr)
    begin
        if (clr == 1)
            hc <= 0;
        else
            begin
                if (hc == hpixels - 1)
                    begin
                        // The counter has reached the end of pixel count
                        hc <= 0;       // 计数器复位
                        vsenable <= 1;
                        // Enable the vertical counter to increment
                    end
                else
                    begin
                        hc <= hc + 1;      // Increment the horizontal counter
                        vsenable <= 0;    // Leave the vsenable off
                    end
            end
    end

// 产生 hsync 脉冲
// 当 hc 为 0 – 127 时，行同步脉冲为低
always @ ( * )
    begin
if (hc < 96)
            hsync = 0;
        else
            hsync = 1;
    end

// 场同步信号计数器
always @ (posedge clk or posedge clr)
    begin
        if (clr == 1)
            vc <= 0;
        else
```

```
            if (vsenable == 1)
                begin
                    if (vc == vlines - 1)
                        // Reset when the number of lines is reached
                        vc <= 0;
                    else
                        vc <= vc + 1;     // 场计数器加 1
                end
    end

// 产生 vsync 脉冲
// 当 hc 为 0 或 1 时，场同步脉冲为低
always @ ( * )
    begin
        if ( vc < 2)
            vsync = 0;
         else
            vsync = 1;
    end

// Enable video out when within the porches
always @ ( * )
    begin
        if ((hc < hfp) && (hc > hbp) && (vc < vfp) && (vc >vbp))
            vidon = 1;
         else
            vidon = 0;
    end

    endmodule


// 例 71b：vga_stripes
module vga_stripes(
input wire vidon,
input wire [9:0] hc, vc,
output reg [2:0] red, green,
output reg [1:0] blue
);

// 输出 16 行宽的红绿条纹
always @(*)
    begin
        red = 0;
```

```
        green = 0;
        blue = 0;
        if (vidon == 1)
            begin
                red = {vc[4], vc[4], vc[4]};
                green = ~{vc[4], vc[4], vc[4]};
            end
    end
    endmodule


// 例 71c：vga_stripes_top
module vga_stripes_top(
input wire mclk,
input wire [3:0] btn,
output wire hsync, vsync,
output wire [2:0] red, green,
output wire [1:0] blue
);

wire clk25, clr, vidon;
wire [9:0] hc, vc;

assign clr = btn[3];

clkdiv U1(.mclk(mclk),
          .clr(clr),
          .clk25(clk25)
);

    vga_640x480 U2(.clk(clk25),
                   .clr(clr),
                   .hsync(hsync),
                   .vsync(vsync),
                   .hc(hc),
                   .vc(vc),
                   .vidon(vidon)
);

vga_stripes U3 (.vidon(vidon),
                .hc(hc),
                .vc(vc),
                .red(red),
                .green(green),
```

```
                .blue(blue)
);

    endmodule
```

例 72：VGA – PROM

```
// 例 72a：prom_DMH
module prom_DMH(
input wire [3:0] addr,
output wire [0:31] M
);

reg [0:31] rom[0:15];

parameter data= {
        32'b01111110000011000001101000000010,      // 0
        32'b01000001000011000001101000000010,      // 1
        32'b01000000100010100010101000000010,      // 2
        32'b01000000010010100010101000000010,      // 3
        32'b01000000001010100010101000000010,      // 4
        32'b01000000001010010100101000000010,      // 5
        32'b01000000001010010100101000000010,      // 6
        32'b01000000001010010100101111111110,       // 7
        32'b01000000001010001000101000000010,      // 8
        32'b01000000001010001000101000000010,      // 9
        32'b01000000001010001000101000000010,      // 10
        32'b01000000001010001000101000000010,      // 11
        32'b01000000010010000000101000000010,      // 12
        32'b01000000100010000000101000000010,      // 13
        32'b01000001000010000000101000000010,      // 14
        32'b01111110000010000000101000000010        // 15
        };

integer i;

initial
    begin
        for(i=0; i<16; i=i+1)
            rom[i] = data[(511-32*i)-:32];
     end
```

```
assign M = rom[addr];

    endmodule


// 例 72b：vga_initials
module vga_initials(
input wire vidon,
input wire [9:0] hc,
input wire [9:0] vc,
input wire [0:31] M,
input wire [7:0] sw,
output wire [3:0] rom_addr4,
output reg [2:0] red,
output reg [2:0] green,
output reg [1:0] blue
);

parameter hbp = 10'b0010010000;     // 行显示后沿 = 144 (128 +16)
parameter vbp = 10'b0000011111;     // 场显示后沿 = 31 (2+29)
parameter W = 32;
parameter H = 16;

wire [10:0] C1, R1, rom_addr, rom_pix;
reg spriteon, R, G, B;
assign C1 = {2'b00, sw[3:0], 5'b00001};
assign R1 = {2'b00, sw[7:4], 5'b00001};
assign rom_addr = vc - vbp - R1;
assign rom_pix = hc - hbp - C1;
assign rom_addr4 = rom_addr[3:0];

// Enable sprite video out when within the sprite region
always @ ( * )
    begin
        if ((hc >= C1 + hbp) && (hc < C1 + hbp + W) &&
            (vc >= R1 + vbp) && (vc < R1 + vbp + H))
                spriteon = 1;
        else
                spriteon = 0;
    end

// 输出视频色彩信号
always @(*)
    begin
```

```
            red = 0;
            green = 0;
            blue = 0;
            if ((spriteon == 1) && (vidon == 1))
                begin
                    R = M[rom_pix];
                    G = M[rom_pix];
                    B = M[rom_pix];
                    red = {R,R,R);
                    green = {G,G,G};
                    blue = {B,B};
                end
        end
        endmodule


// 例 72c：vga_initials_top
module vga_initials_top (
input wire mclk,
input wire [3:0] btn,
input wire [7:0] sw,
output wire hsync,
output wire vsync,
output wire [2:0] red,
output wire [2:0] green,
output wire [1:0] blue
);
wire clr, clk25, vidon;
wire [9:0] hc, vc;
wire [0:31] M;
wire [3:0] rom_addr4;

assign clr = btn[3];

clkdiv U1 ( .mclk(mclk),
            .clr(clr),
            .clk25(clk25)
);

vga_640x480 U2 ( .clk(clk25),
                 .clr(clr),
                 .hsync(hsync),
                 .vsync(vsync),
                 .hc(hc),
```

```
                        .vc(vc),
                        .vidon(vidon)
);

vga_initials U3 ( .vidon(vidon),
                  .hc(hc),
                  .vc(vc),
                  .M(M),
                  .sw(sw),
                  .rom_addr4(rom_addr4),
                  .red(red),
                  .green(green),
                  .blue(blue)
);
prom_DMH U4 ( .addr(rom_addr4),
                   .M(M)
);
    endmodule
```

例 73：块 ROM 中的 sprite

```
// 例 73a：vga_bsprite
module vga_bsprite (
input wire vidon,
input wire [9:0] hc,
input wire [9:0] vc,
input wire [7:0] M,
input wire [7:0] sw,
output wire [15:0] rom_addr16,
output reg [2:0] red,
output reg [2:0] green,
output reg [1:0] blue
);
parameter hbp = 10'b0010010000;    // Horizontal back porth = 144 (128 +16)
parameter vbp = 10'b0000011111;    // Vertical back porth = 31 (2+29)
parameter W = 240;
parameter H = 160;

wire [10:0] C1, R1, xpix, ypix;
wire [16:0] rom_addr1, rom_addr2;
reg spriteon, R, G, B;
```

```
assign C1 = {2'b00, sw[3:0], 5'b00001};
assign R1 = {2'b00, sw[7:4], 5'b00001};
assign ypix = vc - vbp - R1;
assign xpix = hc - hbp - C1;
// rom_addr1 = y*(128+64+32+16) = y*240
assign rom_addr1 = {ypix, 7'b0000000} + {1'b0, ypix, 6'b000000}
                        + {2'b00, ypix, 5'b00000} + {3'b000, ypix, 4'b0000};
// rom_addr2 = y*240 + x
assign rom_addr2 = rom_addr1 + {8'b00000000, xpix};
assign rom_addr16 = rom_addr2[15:0];

// Enable sprite video out when within the sprite region
always @ ( * )
    begin
        if ((hc >= C1 + hbp) && (hc < C1 + hbp + W) &&
            (vc >= R1 + vbp) && (vc < R1 + vbp + H))
            spriteon = 1;
        else
            spriteon = 0;
    end

// Output video color signals
always @ (*)
    begin
        red = 0;
        green = 0;
        blue = 0;
        if ((spriteon == 1) && (vidon == 1))
            begin
                red = M[7:5];
                green = M[4:2];
                blue = M[1:0];
            end
    end
    endmodule


// 例 38c：vga_bsprite_top
module vga_bsprite_top (
input wire mclk,
input wire [3:0] btn,
input wire [7:0] sw,
output wire hsync,
output wire vsync,
```

```
output wire [2:0] red,
output wire [2:0] green,
output wire [1:0] blue
);
wire clr, clk25, vidon;
wire [9:0] hc, vc;
wire [7:0] M;
wire [15:0] rom_addr16;

assign clr = btn[3];

clkdiv U1    ( .mclk(mclk),
                .clr(clr),
                .clk25(clk25)
);

vga_640x480 U2 ( .clk(clk25),
                 .clr(clr),
                 .hsync(hsync),
                 .vsync(vsync),
                 .hc(hc),
                 .vc(vc),
                 .vidon(vidon)
);

vga_bsprite U3 ( .vidon(vidon),
                 .hc(hc),
                 .vc(vc),
                 .M(M),
                 .sw(sw),
                 .rom_addr16(rom_addr16),
                 .red(red),
                 .green(green),
                 .blue(blue)
);

loons240x160 U4 ( .addra(rom_addr16),    // Bus[15;0]
                  .clka(clk25),
                  .douta(M)              // Bus[7;0]
);
      endmodule
```

例 74：屏幕保护程序

```
// 例 74a：vga_ScreenSaver
module vga_ScreenSaver(
input wire vidon,
input wire [9:0] hc,
input wire [9:0] vc,
input wire [7:0] M,
input wire [9:0] C1,
input wire [9:0] R1,
output wire [15:0] rom_addr16,
output reg [2:0] red,
output reg [2:0] green,
output reg [1:0] blue
);

parameter hbp = 10'b0010010000;    // Horizontal back porch = 144 (128+16)
parameter vbp = 10'b0000011111;    // Vertical back porch = 31 (2+29)
parameter W = 240;
parameter H = 160;
wire [9:0] xpix, ypix;
wire [16:0] rom_addr1, rom_addr2;
reg spriteon;

assign ypix = vc - vbp - R1;
assign xpix = hc - hbp - C1;
// rom_addr1 = y*(128+64+32+16) = y*240
assign rom_addr1 = {ypix, 7'b0000000} + {1'b0, ypix, 6'b000000}
                        + {2'b00, ypix, 5'b00000} + {3'b000, ypix, 4'b0000};
// rom_addr2 = y*240 + x
assign rom_addr2 = rom_addr1 + {8'b00000000, xpix};
assign rom_addr16 = rom_addr2[15:0];

// Enable sprite video out when within the sprite region
always @ ( * )
    begin
        if ((hc >= C1 + hbp) && (hc < C1 + hbp + W) && (vc >= R1 + vbp)
                                        && (vc < R1 + vbp + H))
            spriteon = 1;
        else
            spriteon = 0;
    end

// 输出视频色彩信号
```

```
    always @ ( * )
begin
            red = 0;
            green = 0;
            blue = 0;
            if ((spriteon == 1) && (vidon == 1))
                begin
                        red = M[7:5];
                        green = M[4:2];
                        blue = M[1:0];
                    end
        end
    endmodule


// 例 74b：vga_bounce
module bounce (
input wire cclk,
input wire clr,
input wire go,
output wire [9:0] c1,
output wire [9:0] r1
);

parameter C1max = 400;
parameter R1max = 320;
reg [9:0] clv, rlv, dcv, drv;
reg calc;

always @ (posedge cclk or posedge clr)
    begin
        if (clr == 1)
            begin
                clv = 80;
                rlv = 140;
                dcv = 1;
                drv = -1;
                calc = 0;
            end
        else
            if (go == 1)
                calc = 1;
            else
                begin
```

```
                    if (calc == 1)
                        begin
                            clv = clv + dcv;
                            rlv = rlv + drv;
                            if ((clv < 1) || (clv >= C1max))
                                dcv = 0 - dcv;
                            if ((rlv < 1) || (rlv >= R1max))
                                drv = 0 - drv;
                        end
                end
        end

assign c1 = clv;
assign r1 = rlv;

        endmodule


// 例 74c：vga_ScreenSaver_top
module vga_ScreenSaver_top(
input wire mclk,
input wire [3:0] btn,
output wire hsync,
output wire vsync,
output wire [2:0] red,
output wire [2:0] green,
output wire [1:0] blue
);
wire clr, clk25, clk190, vidon, go1;
wire [9:0] hc, vc, C1, R1;
wire [7:0] M;
wire [15:0] rom_addr16;

assign clr = btn[3];

clkdiv U1    ( .mclk(mclk),
               .clr(clr),
               .clk190(clk190),
               .clk25(clk25)
);

vga_640x480 U2 ( .clk(clk25),
                 .clr(clr),
                 .hsync(hsync),
```

```
                        .vsync(vsync),
                        .hc(hc),
                        .vc(vc),
                        .vidon(vidon)
);

vga_ScreenSaver U3 ( .vidon(vidon),
                            .hc(hc),
                            .vc(vc),
                            .M(M),
                            .C1(C1),
                            .R1(R1),
                            .rom_addr16(rom_addr16),
                            .red(red),
                            .green(green),
                            .blue(blue)
);

loons240x160 U4 ( .addr(rom_addr16),    // Bus[15;0]
                        .clk(clk25),
                        .dout(M)                // Bus[7;0]
);

clock_pulse U5 ( .inp(btn[0]),
                        .cclk(clk190),
                        .clr(clr),
                        .outp(go1)
);
bounce U6 ( .cclk(clk190),
                    .clr(clr),
                    .go(go1),
                    .c1(C1),
                    .r1(R1)
);

      endmodule
```

例 75：键盘

```
//Example 41a: keyboard
module keyboard(
input wire clk25,
```

```
input wire clr,
input wire PS2C,
input wire PS2D,
output wire [15:0] xkey
);
reg PS2Cf, PS2Df;
reg [7:0] ps2c_filter, ps2d_filter;
reg [10:0] shift1, shift2;
assign xkey = {shift2[8:1],shift1[8:1]};

//filter for PS2 clock and data

always @(posedge clk25 or posedge clr)
    begin
        if(clr == 1)
            begin
ps2c_filter <= 0;
                    ps2d_filter <= 0;
                    PS2Cf <= 1;
                    PS2Df <= 1;
            end
         else
            begin
                    ps2c_filter[7] <= PS2C;
                    ps2c_filter[6:0] <= ps2c_filter[7:1];
                    ps2d_filter[7] <= PS2D;
                    ps2d_filter[6:0] <= ps2d_filter[7:1];
                    if(ps2c_filter == 8'b11111111)
                       PS2Cf <= 1;
                    else
                       if(ps2c_filter == 8'b00000000)
                            PS2Cf <= 0;
                    if(ps2d_filter == 8'b11111111)
                       PS2Df <= 1;
                    else
                       if(ps2d_filter == 8'b00000000)
                            PS2Df <= 0;
            end
     end

//Shift register used to clock in scan codes from PS2
always @(negedge PS2Cf or posedge clr)
    begin
        if(clr == 1)
```

```
            begin
                shift1 <= 0;
                shift2 <= 1;
            end
        else
            begin
                shift1 <= {PS2Df,shift1[10:1]};
                shift2 <= {shift1[0],shift2[10:1]};
            end
    end

endmodule


module keyboard_top(
input wire mclk,
input wire PS2C,
input wire PS2D,
input wire [3:0] btn,
output wire [6:0] a_to_g,
output wire dp,
output wire [3:0] an
);
wire pclk, clk25, clk190, clr;
wire [15:0] xkey;

assign clr = btn[3];

clkdiv U1 (.mclk(mclk),
    .clr(clr), .clk190(clk190), .clk25(clk25)
);

keyboard U2 (.clk25(clk25),
    .clr(clr), .PS2C(PS2C), .PS2D(PS2D), .xkey(xkey)
);

x7segbc U3 (.x(xkey),
    .cclk(clk190), .clr(clr), .a_to_g(a_to_g),
     .an(an), .dp(dp)
);

endmodule
```

例 76：鼠标

```
// Example 42a: mouse_ctrl
module mouse_ctrl(
input wire clk25,
input wire clr,
input wire sel,
inout wire PS2C,
inout wire PS2D,
output reg [7:0] byte3,
output reg [8:0] x_data,
output reg [8:0] y_data
);
reg [3:0] state;

parameter start = 4'b0000, clklo = 4'b0001, datlo = 4'b0010,
                relclk = 4'b0011, sndbyt = 4'b0100, wtack = 4'b0101,
                    wtclklo = 4'b0110, wtcdrel = 4'b0111, wtclklo1 = 4'b1000,
                    wtclkhi1 = 4'b1001, getack = 4'b1010, wtclklo2 = 4'b1011,
                    wtclkhi2 = 4'b1100, getmdata = 4'b1101;
reg PS2Cf, PS2Df, cen, den, sndflg;
reg ps2cin, ps2din, ps2cio, ps2dio;
reg [7:0] ps2c_filter, ps2d_filter;
reg [8:0] x_mouse_v, y_mouse_v, x_mouse_d, y_mouse_d;
reg [10:0] Shift1, Shift2, Shift3;
reg [9:0] f4cmd;
reg [3:0] bit_count, bit_count1;
reg [5:0] bit_count3;
reg [11:0] count;
parameter COUNT_MAX = 12'h9C4;    // 2500 100us
parameter BIT_COUNT_MAX =4'b1010;    // 10
parameter BIT_COUNT1_MAX = 4'b1100;    // 12 ack
parameter BIT_COUNT3_MAX = 6'b100001;    // 33

//tri-state buffers
always @(*)
    begin
        if(cen == 1)
            ps2cio = ps2cin;
        else
            ps2cio = 1'bz;
        if(den == 1)
            ps2dio = ps2din;
        else
```

```
            ps2dio = 1'bz;
    end

assign PS2C = ps2cio;
assign PS2D = ps2dio;

//filter for PS2 clock and data
always @(posedge clk25 or posedge clr)
    begin
        if(clr == 1)
            begin
                ps2c_filter <= 0;
                ps2d_filter <= 0;
                PS2Cf <= 1;
                PS2Df <= 1;
            end
         else
            begin
                ps2c_filter[7] <= PS2C;
                  ps2c_filter[6:0] <= ps2c_filter[7:1];
                ps2d_filter[7] <= PS2D;
                  ps2d_filter[6:0] <= ps2d_filter[7:1];
                  if(ps2c_filter == 8'b11111111)
                        PS2Cf <= 1;
                  else
                        if(ps2c_filter == 8'b00000000)
                              PS2Cf <= 0;
                  if(ps2d_filter == 8'b11111111)
                        PS2Df <= 1;
                  else
                        if(ps2d_filter == 8'b00000000)
                              PS2Df <= 0;
                    end
    end

// State machine for reading mouse
always @(posedge clk25 or posedge clr)
    begin
        if(clr == 1)
          begin
                state <= start;
                  cen <= 0;
                  den <= 0;
                  ps2cin <= 0;
```

```
                count <= 0;
                bit_count3 <= 0;
                bit_count1 <= 0;
                Shift1 <= 0;
                Shift2 <= 0;
                Shift3 <= 0;
                x_mouse_v <= 0;
                y_mouse_v <= 0;
                x_mouse_d <= 0;
                y_mouse_d <= 0;
                sndflg <= 0;
end
        else
         case(state)
                start:
                        begin
                                cen <= 1;                   // enable clock output
                                  ps2cin <= 0;              // start bit
                                  count <= 0;
                                  state <= clklo;
                         end
                   clklo:
                        if(count < COUNT_MAX)
                            begin
                                 count <= count + 1;
                                  state <= clklo;
                             end
                         else
                            begin
                                 state <= datlo;
                                  den <= 1;                 //enable data output
                             end
                     datlo:
                        begin
                              state <= relclk;
                                cen <= 0;                   // release clock
                         end
                     relclk:
                        begin
                              sndflg <= 1;
                                state <= sndbyt;
                         end
                     sndbyt:
                        if(bit_count < BIT_COUNT_MAX)
```

```
            state <= sndbyt;
        else
            begin
                state <= wtack;
                sndflg <= 0;
                den <= 0;                   // release data
            end
wtack:                                      // wait for data low
    if(PS2Df == 1)
            state <= wtack;
    else
            state <= wtclklo;
wtclklo:                                    // wait for clock low
    if(PS2Cf == 1)
            state <= wtclklo;
        else
            state <= wtcdrel;
wtcdrel:                            // wait to release clock and data
    if((PS2Cf == 1)&&(PS2Df == 1))
            begin
                state <= wtclklo1;
                bit_count1 <= 0;
            end
        else
            state <= wtcdrel;
wtclklo1:                                   // wait for clock low
    if(bit_count1 < BIT_COUNT1_MAX)
        if(PS2Cf == 1)
            state <= wtclklo1;
        else
            begin
                state <= wtclkhi1;      //get ack byte FA
                Shift1 <= {PS2Df, Shift1[10:1]};
            end
        else
            state <= getack;
wtclkhi1:                                   // wait for clock high
    if(PS2Cf == 0)
        state <= wtclkhi1;
        else
        begin
            state <= wtclklo1;
            bit_count1 <= bit_count1 + 1;
        end
```

```
    getack:                                  // get ack FA
        begin
            y_mouse_v <= Shift1[9:1];
             x_mouse_v <= Shift2[8:0];
             byte3 <= {Shift1[10:5], Shift1[1:0]};
             state <= wtclklo2;
             bit_count3 <= 0;
         end
    wtclklo2:                                //wait for clock low
if(bit_count3 < BIT_COUNT3_MAX)
            if(PS2Cf == 1)
                state <= wtclklo2;
            else
                begin
                    state <= wtclkhi2;
                       Shift1 <= {PS2Df, Shift1[10:1]};
                       Shift2 <= {Shift1[0], Shift2[10:1]};
                       Shift3 <= {Shift2[0], Shift3[10:1]};
                 end
         else
            begin
                 x_mouse_v <= {Shift3[5], Shift2[8:1]};
                                                  //x velocity
                  y_mouse_v <= {Shift3[6], Shift1[8:1]};
                                                  //y velocity
                  byte3 <= Shift3[8:1];
                  state <= getmdata;
             end
    wtclkhi2:
        if(PS2Cf == 0)
            state <= wtclkhi2;
         else
            begin
                 state <= wtclklo2;
                  bit_count3 <= bit_count3 + 1;
             end
    getmdata:                    // read mouse data and keep going
        begin
            x_mouse_d <= x_mouse_d + x_mouse_v;
                                                  //x distance
              y_mouse_d <= y_mouse_d + y_mouse_v;
                                                  //y distance
              bit_count3 <= 0;
              state <= wtclklo2;
```

```
                    end
                default;
        endcase
    end
// send F4 command to moouse
always @(negedge PS2Cf or posedge clr)
    begin
        if(clr == 1)
    begin
                f4cmd <= 10'b1011110100;        //stop-parity-F4
                  ps2din <= 0;
                  bit_count <= 0;
             end
        else
            if(sndflg == 1)
                begin
                    ps2din <= f4cmd[0];
                      f4cmd[8:0] <= f4cmd[9:1];
                      f4cmd[9] <= 0;
                      bit_count <= bit_count + 1;
                 end
    end

//Output select
always @(*)
    begin
        if(sel == 0)
            begin
                x_data <= x_mouse_v;
                  y_data <= y_mouse_v;
             end
        else
            begin
                x_data <= x_mouse_d;
                  y_data <= y_mouse_d;
             end
    end

endmodule


// Example 52b: mouse_top
module mouse_top(
input wire mclk,
```

```
inout wire PS2C,
inout wire PS2D,
input wire [3:0] btn,
output wire [3:0] ld,
output wire [6:0] a_to_g,
output wire dp,
output wire [3:0] an
);
wire clk25, clk190, cllr;
wire [7:0] byte3;
wire [8:0] x_data, y_data;
wire [15:0] xmouse;

assign clr = btn[3];
assign xmouse = {x_data[7:0], y_data[7:0]};
assign ld[0] = y_data[8];
assign ld[1] = x_data[8];
assign ld[2] = byte3[1];          //right button
assign ld[3] = byte3[0];          //left button

clkdiv U1 (.mclk(mclk),
          .clr(clr), .clk190(clk190), .clk25(clk25)
);

mouse_ctrl U2 (.clk25(clk25),
          .clr(clr), .sel(btn[0]), .PS2C(PS2C), .PS2D(PS2D),
             .byte3(byte3), .x_data(x_data), .y_data(y_data)
);

x7segbc U3 (.x(xmouse),
          .cclk(clk190), .clr(clr), .a_to_g(a_to_g),
             .an(an), .dp(dp)
);


endmodule


// Example 13.6 vga_mouse_top
module vga_mouse_top(
input wire mclk,
inout wire PS2C,
inout wire PS2D,
input wire [3:0] btn,
```

```
output wire [1:0] ld,
output wire hsync,
output wire vsync,
output wire [2:0] red,
output wire [2:0] green,
output wire [1:0] blue
  );
wire clk25, clk190, clr, vidon, sel;
wire [7:0] byte3;
wire [8:0] x_data, y_data;
wire [9:0] hc, vc, cursor_row, cursor_col;
wire [0:31] M;
wire [3:0] rom_addr4;
parameter XC = 10'b0101000000;          //320
parameter YC = 10'b0011110000;          //240

assign cle = btn[3];
assign sel = 1;
assign ld[0] = byte3[1];
assign ld[1] = byte3[0];
assign cursor_row = YC - {y_data[8], y_data};
assign cursor_col = XC + {x_data[8], x_data};

clkdiv U1 (.mclk(mclk), .clr(clr), .clk190(clk190), .clk25(clk25));

vga_640x480 U2 (.clk(clk25), .clr(clr), .hsync(hsync),
                .vsync(vsync), .hc(hc), .vc(vc), .vidon(vidon));

vga_mouse_initials U3 (.vidon(vidon), .hc(hc), .vc(vc), .M(M),
                .cursor_row(cursor_row), .cusor_col(cursor_col),
                    .rom_addr4(rom_addr4), .red(red), .green(green), .blue(blue));

prom_DMH U4 (.addr(rom_addr4), .M(M));

mouse_crl U5 (.clk25(clk25), .clr(clr), .sel(sel), .PS2C(PS2C),
                .PS2D(PS2D), .byte3(byte3), .x_data(x_data), .y_data(y_data));

endmodule
```