module test01();

wire a , b , c , d ;

and(a , b , c , d) ;

parameter s1 = 2'b00 ,

s2 = 2'b01 ;

// adder8 #(parameter的参数) myadder8() ;

endmodule

module add1(cin,sum,cout,a,b);

output cout ;

output sum ;

input a , b ; input cin ;

assign {cout , sum} = a + b + cin ;

endmodule

module add4 (cin, sum, cout, a, b);

output[3:0] sum;

output cout;

input[3:0] a, b;

input cin;

wire c1, c2, c3;

add1 f0(cin, sum[0], c1, a[0], b[0]);

f1(c1, sum[1], c2, a[1], b[1]);

f2(c2, sum[2], c3, a[2], b[2]);

f3(c3, sum[3], cout, a[3], b[3]) ;

endmodule

module sub4 (cin, res, cout, a, b);

output[3:0] res ; output cout ;

input[3:0] a, b ; input cin ;

reg[3:0] res ;

reg cout;

always @(a, b)

begin

{cout, res} = a - b - cin ;

end

endmodule

module code8\_3(y , x) ;

output reg[2:0] y ;

input[7:0] x ;

always @(x)

begin

case(x[7:0])

8'b00000001: y[2:0] = 3'b000;

8'b00000010: y[2:0] = 3'b001;

8'b00000100: y[2:0] = 3'b010;

8'b00001000: y[2:0] = 3'b011;

8'b00010000: y[2:0] = 3'b100;

8'b00100000: y[2:0] = 3'b101;

8'b01000000: y[2:0] = 3'b110;

8'b10000000: y[2:0] = 3'b111;

default: y[2:0] = 3'b000;

endcase

end

endmodule

module decode3\_8(y , x , g1 , g2 , g3) ;

output reg[7:0] y ;

input[2:0] x ; input g1 , g2 , g3 ;

always @(x)

begin

if(g1 == 0) y = 8'b1111\_1111;

else if(g2 == 1 && g3 == 1) y = 8'b1111\_1111;

else begin

y = 8'b0000\_0001 << x;

y = ~y;

end

end

endmodule

module counter8(out,cout,data,load,cin,CLK);

output[7:0] out ; output cout ;

input[7:0] data ; input load , cin , CLK ;

reg[7:0] out ;

always @(posedge CLK)

begin

if(load) out = data ;

end

assign cout = &out & cin ;

endmodule

module counter4(cout , cnt , en , clr , CLK , load , start) ;

output reg cout ; output reg[3:0] cnt ;

input en , clr , CLK , load , start ;

always @(posedge CLK)

begin

if(clr) cnt = 0 ;

else if(load) cnt = start ;

else if(en) begin

if(cnt == 4'b1111) cout = 1 ;

else cout = 0 ;

cnt = cnt + 1 ;

end else cnt = cnt ;

end

endmodule

module counter24 (ten, one, cout, clk, clr);

output reg[3:0] ten, one; //输出

output reg cout; //进位信号

input clk, clr;

always @(posedge clk)

begin

if (clr) begin ten <= 0; one <= 0; end

else begin

if({ten, one} == 8'b0010\_0011)

begin ten <= 0; one <= 0; cout <= 1 ; end

else if(one==4'b1001)

begin one <= 0; ten <= ten+1 ; cout <= 0 ; end

else

begin one <= one + 1 ; cout <= 0 ; end

end

end

endmodule

module counter60(cnt , cout , clk , clr) ;

output reg[5:0] cnt ; output reg cout ;

input clk , clr ;

always @(posedge clk)

begin

if(clr) cnt = 0 ;

else begin

if(cnt == 60) begin cnt = 0 ; cout = 1 ; end

else cnt = cnt + 1 ;

end

end

endmodule

module yb\_cnt4(q, clk, rst);

output[3:0] q;

input clk, rst;

reg[3:0] q;

reg[3:0] qn;

always @ (posedge clk)

begin if(!rst) begin q[0] = 0; qn[0] = 1; end

else begin q[0] = ~q[0]; qn[0] =~q[0]; end

end

always @ (posedge qn[0])

begin if(!rst) begin q[1] = 0; qn[1] = 1; end

else begin q[1] = ~q[1]; qn[1] =~q[1]; end

end

always @ (posedge qn[1])

begin if(!rst) begin q[2] = 0; qn[2] = 1; end

else begin q[2] = ~q[2]; qn[2] =~q[2]; end

end

always @ (posedge qn[2])

begin if(!rst) begin q[3] = 0; qn[3] = 1; end

else begin q[3] = ~q[3]; qn[3] =~q[3]; end

end

endmodule

module comparator3(y1 , y2 , y3 , a , b ) ;

output reg y1 , y2 , y3 ;

input[2:0] a , b ;

always @(a or b)

begin

y1 = 0 ; y2 = 0 ; y3 = 0 ;

if(a > b) y1 = 1 ;

else if(a < b) y3 = 1 ;

else y2 = 1 ;

end

endmodule

module voter3(out , a , b , c ) ;

output out ;

input a , b , c ;

assign out = a&b | a&c | b&c ;

endmodule

module vote7 ( pass,vote );

output pass;

input [6:0] vote;

reg[2:0] sum;

integer i;

reg pass;

always @(vote)

begin

sum = 0;

for(i = 0;i<=6;i = i+1)

if(vote[i]) sum = sum+1;

if(sum[2]) pass = 1;

else pass = 0;

end

endmodule

module mux2\_1(out , select , a , b) ;

output out ;

input a , b , select ;

reg out ;

always @(\*)

begin

if(!select) out = a ;

else out = b ;

end

endmodule

module mux4\_1\_01(out,in1,in2,in3,in4,cntrl1,cntrl2) ;

output out ;

input in1 , in2 , in3 , in4 , cntrl1 , cntrl2 ;

assign out = (in1 & ~cntrl1 & ~cntrl2) |

(in2 & ~cntrl1 & cntrl2) |

(in3 & cntrl1 & ~cntrl2) |

(in4 & cntrl1 & cntrl2) ;

endmodule

module mux4\_1\_02(out,in1,in2,in3,in4,cntrl1,cntrl2) ;

output out ; reg out ;

input in1 , in2 , in3 , in4 , cntrl1 , cntrl2 ;

always @(\*)

begin

case({cntrl1 , cntrl2})

2'b00 : out = in1 ;

2'b01 : out = in2 ;

2'b10 : out = in3 ;

2'b11 : out = in4 ;

default : out = 1'bx ;

endcase

end

endmodule

module mux4\_1\_03(out,in1,in2,in3,in4,cntrl1,cntrl2) ;

output out ;

input in1 , in2 , in3 , in4 , cntrl1 , cntrl2 ;

assign out = cntrl1? (cntrl2? in4 :in3) : (cntrl2? in2 :in1) ;

endmodule

module dmux4\_1(y0 , y1 , y2 , y3 , x , sel) ;

output reg y0 , y1 , y2 , y3 ;

input x ; input[1:0] sel ;

always @(\*)

begin

y0 = 0 ; y1 = 0 ; y2 = 0 ; y3 = 0 ;

case(sel)

2'b00: y0 = x ;

2'b01: y1 = x ;

2'b10: y2 = x ;

2'b11: y3 = x ;

default : ;

endcase

end

endmodule

module fsm\_seq101(y , CLK , clr , x ) ;

output y ; reg y ;

input x , CLK , clr ;

reg[1:0] state , next\_state ;

parameter s0=2'b00,s1=2'b01,s2=2'b11, s3=2'b10;

always @(posedge CLK or posedge clr)

begin

if(clr) state = s0 ;

else state <= next\_state ;

end

always @(state or x)

begin

case(state)

s0 :begin

if(x) next\_state = s1 ;

else next\_state = s0 ;

end

s1 :begin

if(x) next\_state = s1 ;

else next\_state = s2 ;

end

s2 :begin

if(x) next\_state = s3 ;

else next\_state = s0 ;

end

s3 :begin

if(x) next\_state = s1 ;

else next\_state = s2 ;

end

default : next\_state = s0 ;

endcase

end

always @(state)

begin

case(state)

s3 : y = 1'b1 ;

default : y = 1'b0 ;

endcase

end

endmodule