4-bit up down

```
module updown(out, reset, clk, mode);
output reg [2:0]out;
input clk, reset, mode;
always @(posedge clk)
begin
case({reset, mode})
2'b10:out=0;
2'b11:out=0;
2'b00:out=out+1;
2'b01:out=out-1;
endcase
end
endmodule
module updown_tb;
reg reset;
reg clk;
reg mode;
wire [2:0]out;
updown udc(
.out(out),
.reset(reset),
.clk(clk),
.mode(mode)
);
```

```
initial begin

reset=1; clk=1; mode=0;
#100; reset=0; mode=0;
#200; reset=0; mode=1;
end

always #25 clk = ~clk
endmodule
```

3-bit updown

```
module three_bit_updown_counter (clk, up_down, clear, count);

input clk, up_down, clear;
output reg [2:0]count;
always @(posedge clk or posedge clear)
 begin
   if (clear)
     begin
       count <= 3'b000;
     end
     else
       begin //------
```

```
        if (up_down)
          begin
            count <= count + 1;
          end
        else
          begin
            count <= count - 1;
          end
      end //------
  end
endmodule

module test_counter;
 reg clk;
 reg up_down;
 reg clear;

 wire[2:0]count;

three_bit_updown_counter abc (
  .clk(clk),
  .up_down(up_down),
  .clear(clear),
  .count(count)
   );
```

```
initial begin
     clear =1; clk=0; up_down=0;
#100  clear =0; up_down=1;
#200  up_down=0;
#200
$finish;

end

  always #10 clk = ~clk;

endmodule



####### siso
module siso(shift_out, shift_in, clk);

input shift_in;
input clk;
output shift_out;
 reg shift_out;
 reg [2:0]data;

always @(posedge clk)
```

```
begin
data[0] <= shift_in;
data[1] <= data[0];
data[2] <= data[1];
shift_out <= data[2];
end
endmodule

module siso_tb_v;

reg shift_in;
reg clk;

wire shift_out;

siso sample (
.shift_out(shift_out),
.shift_in(shift_in),
.clk(clk)
);

initial begin
 shift_in = 0;
 clk = 0;
```

```
 #100; shift_in = 1;
 #100; shift_in = 0;
 #100; shift_in = 1;
end
always #10 clk = ~clk;
endmodule

#### half adder
module halfadder(a, b, s, c);
input a;
input b;
output s;
output c;

assign {c,s}=a+b;
endmodule

module halfadder_test();
reg a,b;
wire s,c;
halfadder h1(a,b,s,c);
initial
begin
  a=1'b1;  b=1'b1;
```

```
#20 a=1'b1;  b=1'b0;
#20  a=1'b0;  b=1'b1;
#20 a=1'b0;  b=1'b0;
end
endmodule

#### full adder
module full_add(a,b,Cin,s,Co);
  input a,b,Cin;
  output s,Co;
  wire x,y,z;
  xor(s,a,b,Cin);
  xor(x,a,b);
  and(z,x,Cin);
  and(y,a,b);
  or(Co,z,y);
endmodule

module tb_full_add;
  reg a,b,Cin;
  wire s,Co;
  full_add Myfull_add(a,b,Cin,s,Co);
  initial
  begin
    #100 a=1'b0;
```

```
    b=1'b0;
    Cin=1'b0;

    #100
    a=1'b1;
    b=1'b1;
    Cin=1'b1;

    #100
    a=1'b0;
    b=1'b1;
    Cin=1'b0;

    #100
    a=1'b1;
    b=1'b0;
    c=1'b1;


  end
endmodule


####  demux(run nao hoite pare)
module demux1_4(in1,s,out1);
```

```
input in1;
input[1:0]s;
output[3:0]out1;
reg[3:0]out1;
always @(in1,s)
begin
  case(s)
    2'b00:begin out1[0]=in1;
      out1[1]=1'b0;
      out1[2]=1'b0;
      out1[3]=1'b0;
    end
    2'b01:begin out1[0]=1'b0;
      out1[1]=in1;
      out1[2]=1'b0;
      out1[3]=1'b0;
    end
    2'b10:begin out1[0]=1'b0;
      out1[1]=1'b0;
      out1[2]=in1;
      out1[3]=1'b0;
    end

    2'b11:begin out1[0]=1'b0;
      out1[1]=1'b0;
```

```
      out1[2]=1'b0;
      out1[3]=in1;
    end
    default: out1=3'b000;
  endcase
 end
endmodule


####  4 bit add sub
module addsub(a,b,c,s);
 input [3:0] a;
 input [3:0] b;
 input c;
 output reg [4:0]s;
 always@(a,b,c)
 begin
  if(c)
    s= a-b;
  else
    s= a+b;
  end
 endmodule

 module addsub_test;
```

```
 reg [3:0] a;
 reg [3:0] b;
 reg c;
 wire [4:0] s;
 addsub uut (
 .a(a),
 .b(b),
 .c(c),
 .s(s)
 );
 initial
 begin
   a =4'b 0111;
   b =4'b 0010;
   c = 0;
   #100;
   a =4'b 1000;
   b =4'b 0011;
   c = 1;
   #100;
   a =4'b 1100;
   b =4'b 0111;
   c = 1;
 end
endmodule
```

siso – sumaiya

//Serial In Serial Out.....

```
module sisomod(clk,clear,si,so);

input clk,si,clear;

output so;

reg so;

reg [3:0] tmp;

always @(posedge clk )

begin

if (clear)
```

```
tmp <= 4'b0000;

else

tmp <= tmp << 1;

tmp[0] <= si;

so = tmp[3];

end

endmodule
```

TEST BENCH

```
module sisot_b;

reg clk;

reg clear;

reg si;
```

```
wire so;

sisomod uut (.clk(clk), .clear(clear),.si(si),.so(so));

initial begin

clk = 0;

clear = 0;

si = 0;

#5 clear=1'b1;

#5 clear=1'b0;

#10 si=1'b1;

#10 si=1'b0;

#10 si=1'b0;

#10 si=1'b1;

#10 si=1'b0;
```

```
#10 si=1'bx;

end

always #5 clk = ~clk;

initial #150 $stop;

endmodule
```

2.Serial In Parallal Out....

```
module sipo(clk,in,rst,q);
 input clk,rst;
 input in;
 output reg [3:0]q;

 always @ (posedge clk, posedge rst)
  begin
    if(rst)
      q <= 0;
      else
      q <= {in,q[3:1]};
  end
endmodule


module sipotb();
 reg clk,in,rst;
 wire [3:0]q;
 sipo dut(clk,in,rst,q);

 initial
  begin
    clk = 1;
    forever #5 clk = ~clk;
  end
```

```
 initial
   begin
     rst=1;
     #10 rst=0;in=1;
     #10 in=0;
     #10 in=1;
     #10 in=0;
     end

 initial
   begin
     $dumpfile("sipotb.vcd");
     $dumpvars(0,sipotb);
     #100 $finish;
     //
   end

endmodule
```

3.Parallal In parallal Out.....

```
// Code your design here
module pipo(clk,rst,ld,in,q);
  input clk,rst,ld;
  input [3:0]in;
  output reg [3:0] q;
  always @(posedge clk)
     begin
       if (rst)
         q <= 0;
       else if(ld)
        q=in[3:0];


     end
endmodule

// Code your testbench here

module pipotb();
  reg clk,rst,ld;
```

```
 reg [3:0]in;
 wire [3:0]q;
 pipo m(clk,rst,ld,in,q);
 initial
  begin
    $dumpfile("pipotb.vcd");
    $dumpvars(0,pipotb);
  end
 initial
  begin
    clk=1;
    forever #5 clk=~clk;
  end
  initial
  begin
    #10 rst=1;
    #10 rst=0;ld=1;
    #10 in=4'b1001;

    #200 $finish;
  end
endmodule
```

4.Parallal In Serial Out...

```
module piso(in,ld,clk,rst,q);
  input ld,clk,rst;
  input [3:0] in;
  output q;
  reg [3:0]qq;
  assign q=qq[0];
  always @ (posedge clk, posedge rst)
    begin
      if(rst)
         qq<=0;
      else if(ld)
          qq <=in;
      else
        qq <= {1'b0,qq[3:1]};
    end
endmodule
```

```
module pisotb();
 reg ld,clk,rst;
 reg [3:0] in;
 wire q;
 piso dut(in,ld,clk,rst,q);

  initial
   begin
     clk = 1;
     forever #5 clk = ~clk;
   end

 initial
   begin
     $dumpfile("pisotb.vcd");
     $dumpvars(0,pisotb);
     #100 $finish;
   end

  initial
   begin
     rst=1;
     #10 rst=0;ld=1;in=4'b0111;
     #10 ld=0;in=0;
   end
```

```
endmodule
```