* 16 to 1 Mux
  + *Pure Behavioral:*

module mux16to1 (in,sel,out);  
  input[15:0] in;  
  input[3:0] sel;  
  output out;  
  assign out = in[sel] //non constant index in expression on RHS generates a MUX (during synthesis)  
endmodule  
  
module muxtest;  
  reg[15:0] A;  
  reg[3:0] S;  
  wire F;  
  
  mux16to1 myMux(.in(A), .sel(S), .out(F));  
  
  initial  
   begin  
    $dumpfile("mux16to1.vcd");  
    $dumpvars(0,muxtest);  
    $monitor($time, "A=%h, S=%h F= %b", A,S,F);  
    #5 A=16'h3f0a; S= 4'h0;  
    #5 S=4'h1;  
    #5 S=4'h6;  
    #5 S=4'hc;  
    #5 $finish  
   end  
endmodule

* *Using five 4 to 1 Muxes*

module mux4to1 (in,sel,out);  
  input[3:0] in;  
  input[1:0] sel;  
  output out;  
  assign out = in[sel];  
endmodule

//structural description, using behavioral description of 4 to 1 mux  
module mux16to1(in,sel,out);  
  input[15:0] in;  
  input[3:0] sel;

  output out;  
  wire[3:0] t;  
  
  mux4to1 M1 (.in(in[3:0]),.sel(sel[1:0]), .out(t[0]));  
  mux4to1 M2 (.in(in[7:4]),.sel(sel[1:0]), .out(t[1]));  
  mux4to1 M3 (.in(in[11:8]),.sel(sel[1:0]), .out(t[2]));  
  mux4to1 M4 (.in(in[15:12]),.sel(sel[1:0])  
  mux4to1 M5 (.in(t),.sel(sel[3:2]), .out(out));

endmodule

* *Use three 2 to 1 Mux to model 4 to 1 Mux*

module mux2to1 (in,sel,out);  
  input[1:0] in;  
  input sel;  
  output out;  
  assign out = in[sel];  
endmodule

module mux4to1(in,sel,out);  
  input[3:0] in;  
  input[1:0] sel;

  output out;  
  wire[1:0] t;  
  
  mux4to1 M1 (.in(in[1:0]),.sel(sel[0]), .out(t[0]));  
  mux4to1 M2 (.in(in[3:2]),.sel(sel[0]), .out(t[1]));  
  mux4to1 M3 (.in(t),.sel(sel[1]), .out(out));

endmodule

* *Structural Modelling of 2 to 1 Mux*

module mux2to1 (in,sel,out);  
  input[1:0] in;  
  input sel;  
  output out;  
  wire t0,t1,t2;

  NOT G0 (t0, sel)  
  AND G1 (t1,in[0],t0);  
  AND G2 (t2, in[1], sel);  
  OR G3 (out, t1,t2);  
endmodule

* 16-bit Adder

Generation of status flags: *sign, zero, carry, parity, overflow*

* *Pure Behavioral*

module ALU (X,Y,Z,Sign,Zero,Carry,Parity,  
  input[15:0] X,Y;

  output[15:0] Z;  
  output Sign,Zero,Carry,Parity,  
  
  assign {Carry,Z} = X + Y;  
  assign Sign = Z[15];  
  assign Zero = ~|Z;  
  assign Parity = ~^Z;  
  assign Overflow = (X[15] & Y[15] & ~Z[15]) |  
                    (~X[15] & ~Y[15] & Z[15]);  
endmodule  
  
module alutest  
  reg[15:0] X,Y;  
  wire[15:0] Z;  
  wire S,ZR,C,P,O;  
  initial  
    begin  
      $dumpfile("alu.vcd");  
      $dumpvars(0,alutest);  
      $monitor($time, "X=%h, Y=%h, Z=%h, S=%b, ZR=%b, C=%b, P=%b, O=%b", X,Y,Z,S,ZR,C,P,O);      
       #5 X=16'h8fff; Y=16'h8000;  
       #5 X=16'hfffe; Y=16'h0002;  
       #5 X=16'hAAAA; Y=16'h5555;  
       #5 $finish

    end  
endmodule

* *Use Four 4-bit Adders*

module ALU (X,Y,Z,Sign,Zero,Carry,Parity,  
  input[15:0] X,Y;

  output[15:0] Z;  
  output Sign,Zero,Carry,Parity,

    wire[3:1] c;  
  
  adder4 A0(Z[3:0],c[1],X[3:0],Y[3:0],

  adder4 A1(Z[7:4],c[2],X[7:4],Y[7:4],

  adder4 A2(Z[11:8],c[3],X[11:8],Y[11:

  adder4 A3(Z[15:12],Carry,X[15:12],Y[  
  assign Sign = Z[15];  
  assign Zero = ~|Z;  
  assign Parity = ~^Z;  
  assign Overflow = (X[15] & Y[15] & ~Z[15]) |  
                    (~X[15] & ~Y[15] & Z[15]);  
endmodule

module adder4 (S, cout, A, B, cin);  
  input[3:0] A, B;

  output[3:0] S;

     input cin;

     output cout;  
  assign {cout,s} = A + B + cin;  
    
endmodule

* *Model 4-bit Adder as Ripple Carry Adder (4 Full Adders)*

module adder4 (S, cout, A, B, cin);  
  input[3:0] A, B;

  output[3:0] S;

  input cin;

  output cout;

  wire c1,c2,c3;  
  fulladder FA0(S[0], c1, A[0], B[0], 1'b0);  
  fulladder FA1(S[1], c2, A[1], B[1], c1);

  fulladder FA2(S[2], c3, A[2], B[2], c2);

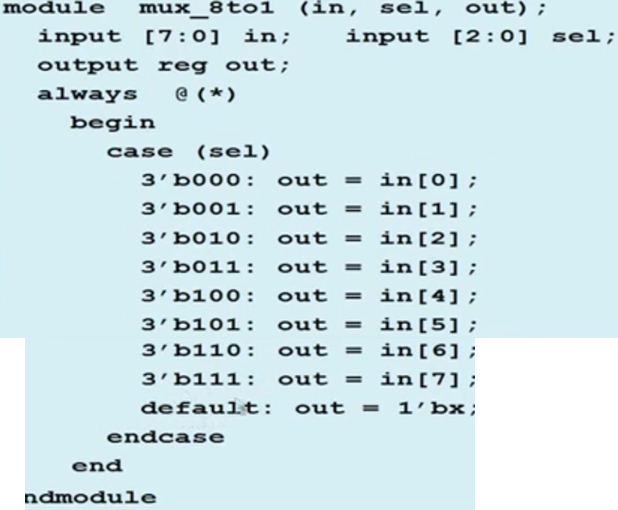
  fulladder FA3(S[3], cout, A[3], B[3], c3);    
endmodule

* *Model Full Adder using 3 XOR and 2 OR Gates*

module fulladder (s,cout,a,b,c);  
  input a,b,c;  
  output s, cout;  
  wire s1,c1,c2;  
  xor G1 (s1,a,b), G2(s,s1,c), G3(cout,c2,c1);  
  and G4 (c1,a,b), G5(c2,s1,c);  
endmodule

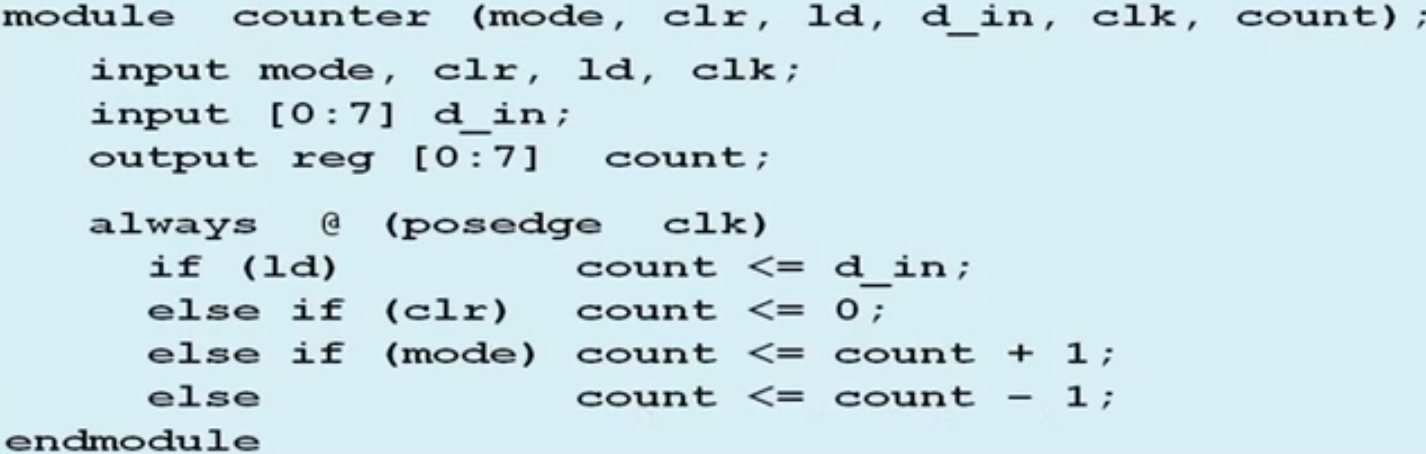
* Improvement: Carry Lookahead Adder Instead of Ripple Carry Adder
  + generate carry's in parallel, so gate delay is less (in ripple carry adder delay is due to 4 carry delays)
  + See Digital Design notes

* Mux using procedural block

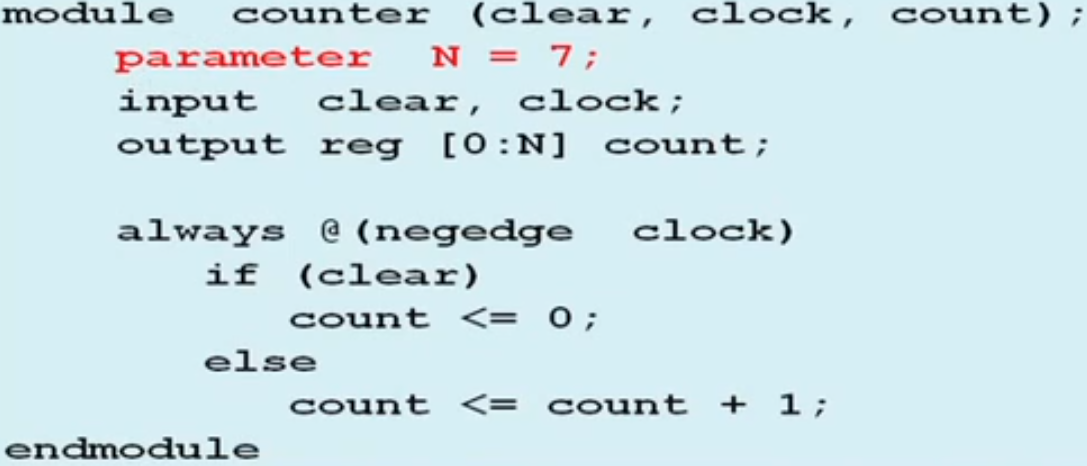


Need the default case as Verilog is a four value system, for cases involving z/x default case will come into play

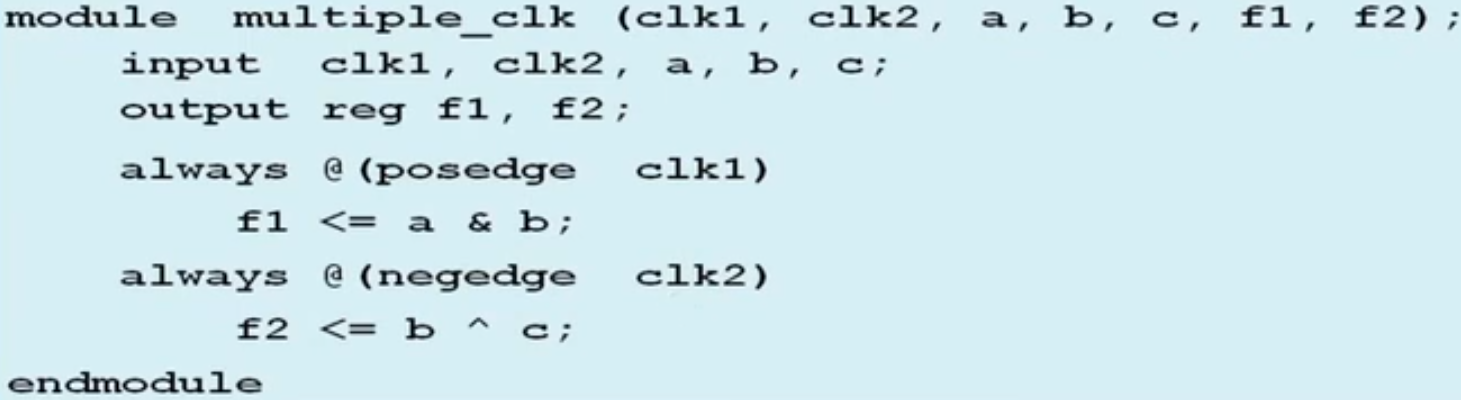
* Synchronous Up Down Counter using Procedural Blocks



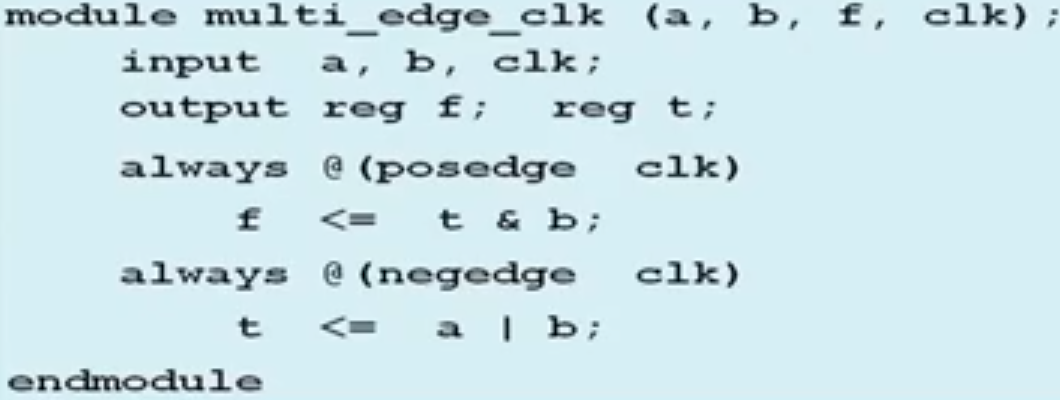
* N-bit Counter



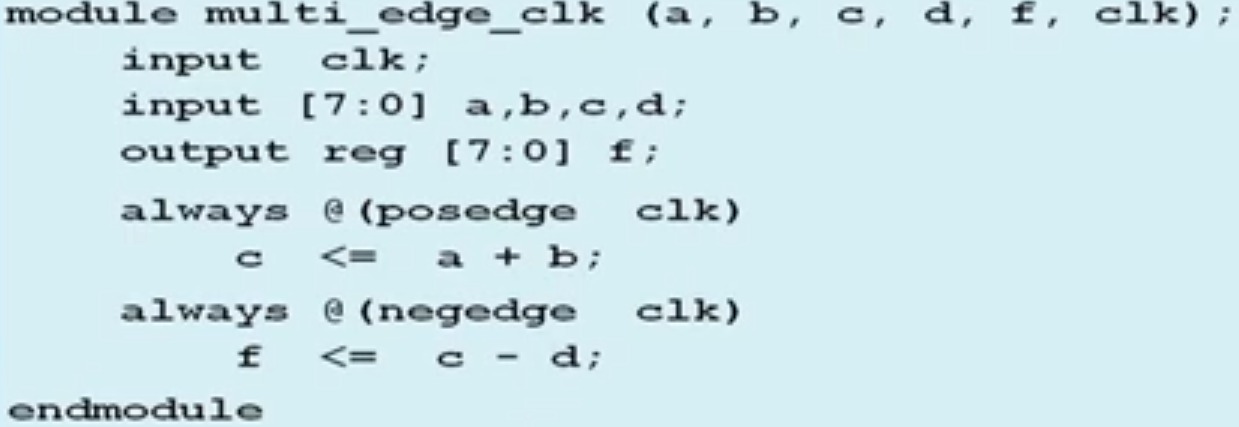
* More than 1 Clocks



* Using Multiple Edges of the Same Clock



t calculated during negedge, then used during posedge

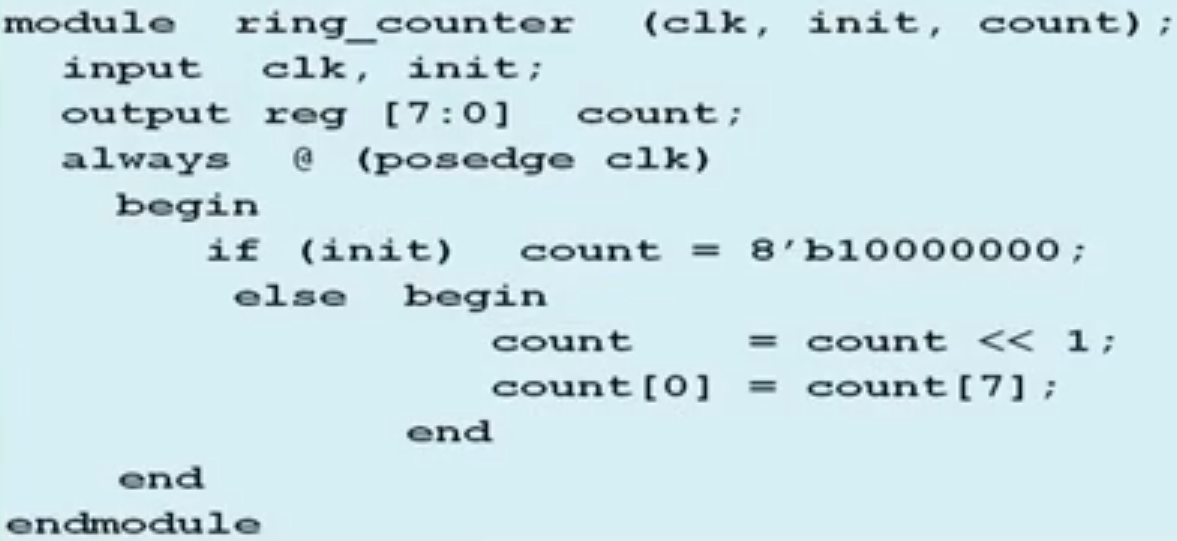


In above example, 2 operations every clock cyele:

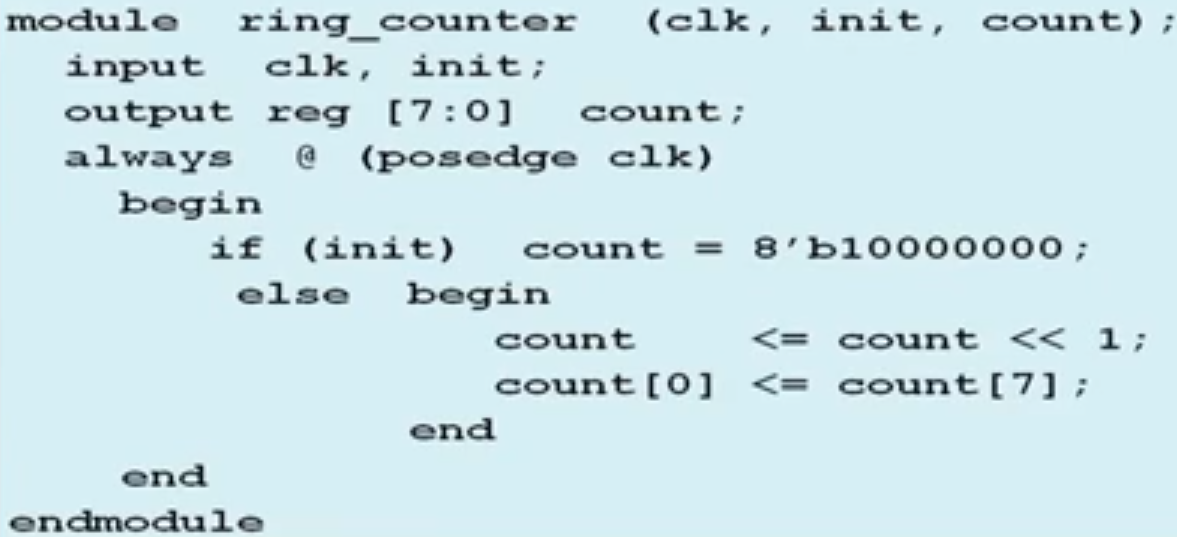
1. c changed during rising edge
2. that value is used during falling edge

* Ring Counter

***Wrong implementation: Blocking statements cause count to become 8'b0, then shift***



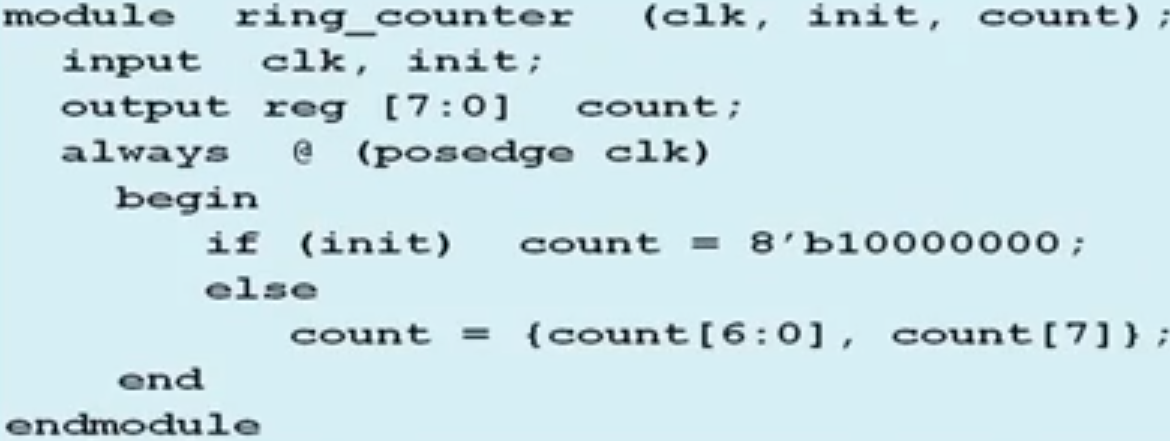
***Correct Implementation: Using non-blocking assignments:***



RHS's calculated first (for both assignments)  
then, assigned to LHS

Now ring counter rotaion works as expeceted

***Correct implementation using blocking assigments:***



* Parameterized N-bit Up/Down Counter

