INTRODUCTION TO VERILOG (Contd..)

EC39004 : VLSI LABORATORY

4 February 2019

Sequential Logic Example Code 1

```
module dff (q, qbar, d, set, reset, clk);
input d, set, reset, clk;
output reg q;
 output qbar;
assign qbar = ~q;
always@(posedge clk)
begin
        if (reset = = 1)  q<=1'b0;
        else if (set == 1) q<= 1'b1;
        else q<=d;
end
endmodule
```

Sequential Logic Example Code 2

```
module dff (q, qbar, d, set, reset, clk);
input d, set, reset, clk;
 output reg q; output qbar;
assign qbar = ~q;
always@(posedge clk or negedge set or negedge
   reset)
begin
        if (reset = = 0)  q<=1'b0;
        else if (set == 0) q<= 1'b1;
        else q<=d;
end
endmodule
```

Sequential Logic Example Code 3

```
module incomp_state_spec (curr_state, flag);
input [1:0] curr_state;
output reg [1:0] flag;
always@(curr_state)
case (curr_state)
0, 1:flag=2'b10;
3: flag = 2'b00;
endcase
endmodule
```

Sequential Logic Example Code 4

```
module incomp_state_spec (curr_state, flag);
input [1:0] curr_state;
output reg [1:0] flag;
always@(curr_state)
begin
flag = 2'b00;
case (curr_state)
0, 1: flag=2'b10;
3: flag = 2'b00;
endcase
end
endmodule
```

Sequential Logic Example Code 5

```
// Up-down counter (synchronous clear)
module counter (mode, clr, ld, d_in, clk,count);
input mode, clr, ld, clk;
input [7:0] d_in;
output reg [7:0] count;
always @ (posedge clk)
begin
   if (ld)
      count <= d_in;
       else if (clr)
      count <= 8'b0;
      else if (mode)
       count <= count+8'b1;
       else
       count<=count-8'b1;
end
endmodule
```

Sequential Logic Example Code 6

```
// Parameterized design:: an N-bit counter

module    counter (clear, clock, count);
   parameter N = 7;
   input      clear, clock;
   output reg [0:N-1] count;

   always @(negedge clock)
     if (clear)
       count <= 0;
       else
       count <= count + 1;
endmodule
```

- Parameterized design makes any design general for any no: of bits using the keyword "parameter"
- Parameter values are substituted before simulation or synthesis

Sequential Logic Example Code 7

```
// Using more than one clocks in a module
module multiple_clk (clk1,clk2, a,b,c,f1,f2);
input  clk1, clk2, a, b,c;
output reg f1, f2;
always@(posedge clk1)
f1<= a&b;
always@(negedge clk2)
f2<= b^c;
endmodule
```

Sequential Logic Example Code 8

```
// Using multiple edge of the same clock
module multi_edge_clk (a, b, f, clk);
input a, b, clk;
output reg f;
reg t;
always@(posedge clk)
f<=t&b;
always@(negedge clk)
t<=a|b;
end
```

Sequential Logic Example Code 9

```
// A ring counter
module ring_counter (clk, init, count);
 input clk, init;
 output reg [7:0] count;
 always @ (posedge clk)
    begin
       if (init)
        count = 8'b10000000;
        else
          begin
            count = count << 1;
            count[0] = count[7];
          end
    end
    endmodule
```

- This is the wrong version.
- Since blocking assignments are used, rotation will not take place correctly

Sequential Logic Example Code 10

```
// A ring counter (Modified version 1)
module ring_counter (clk, init, count);
 input clk, init;
 output reg [7:0] count;
 always @ (posedge clk)
    begin
       if (init)
        count = 8'b10000000;
        else
          begin
            count <= count << 1;
            count[0] <= count[7];
          end
    end
    endmodule
```

- This is the correct version.
- Since non-blocking assignments are used, rotation will take place correctly.

Sequential Logic Example Code 11

```
// A ring counter (Modified version 2)
module ring_counter (clk, init, count);
 input clk, init;
 output reg [7:0]count;
 always @ (posedge clk)
   begin
      if (init)
       count = 8'b10000000;
      else
        count = {count[6:0], count[7]};
   end
   endmodule
```

- This is a correct way of modeling using blocking assignment.

Pipelining: Example

- **Consider the following arithmetic computation:**

  X = (A + B) * (B – C + D)

  Y = (B – C + D)

- **Suppose we break into three stages:**
  - **S1: S1_T1 = A+B; S1_T2 = B-C; S1_T3 = D;**
  - **S2: S2_T1 = S1_T1; S2_T4 = S1_T2 + S1_T3;**
  - **S3: S3_X = S2_T1 * S2_T4; S3_Y = S2_T4;**

Pipelining Example Code

```
module pipeline_example(a,b,c,d,x,y,clk);
input clk;
input [7:0] a,b,c,d;
output [18:0] x; output [9:0] y;
wire [18:0] x; wire [9:0] y;
reg [8:0] s1_t1, s2_t1, s1_t2;
reg [7:0] s1_t3;
reg [9:0] s2_t4, s3_y;
reg [18:0] s3_x;
assign x=s3_x;
 assign y=s3_y;
always@(posedge clk)
begin
s1_t1<=a+b; s1_t2<=b-c; s1_t3<=d;
s2_t1<=s1_t1; s2_t4<=s1_t2+s1_t3;
s3_x<=s2_t1*s2_t4; s3_y<=s2_t4;
```

Assignment:
Pipelined Implementation of 32 bit Barrel shifter

Consider a controlled barrel shifter that accepts two inputs: a 32-bit unsigned data input A, a second input B denoting the shift amount and a control signal R using ONLY 2:1 multiplexers as the sole circuit building block. When R = 0, the circuit performs the left shift operation; otherwise when R = 1, it performs a right shift operation on the same input data A. **Pipeline the circuit such that only a single 2:1 multiplexer comes in the critical path.**