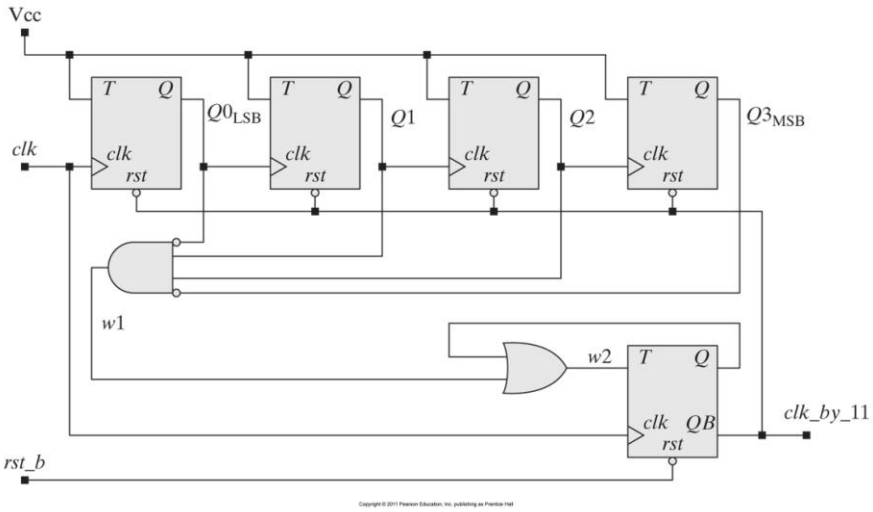
COE 405, Term 122

Design & Modeling of Digital Systems

HW# 4 Solution

Due date: Saturday, March 30

Q.1. The schematic shown below is for Divide_by_11, a frequency divider that divides clk by 11 and asserts its output for one clock cycle. The unit consists of a chain of toggle-type flip-flops with additional logic to form an output pulse every 11^{th} pulse of clk. The asynchronous signal rst_b is active-low and drives Q to 1.



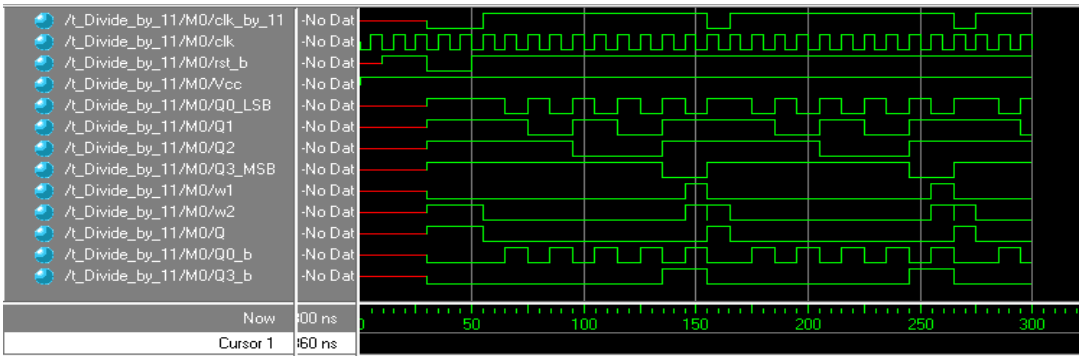
(i) Develop a primitive model of a rising-edge triggered T-type flip-flop.

(ii) Develop a structural Verilog model of Divide_ by_11.

(iii) Verify the correctness of your model by using the following test bench:

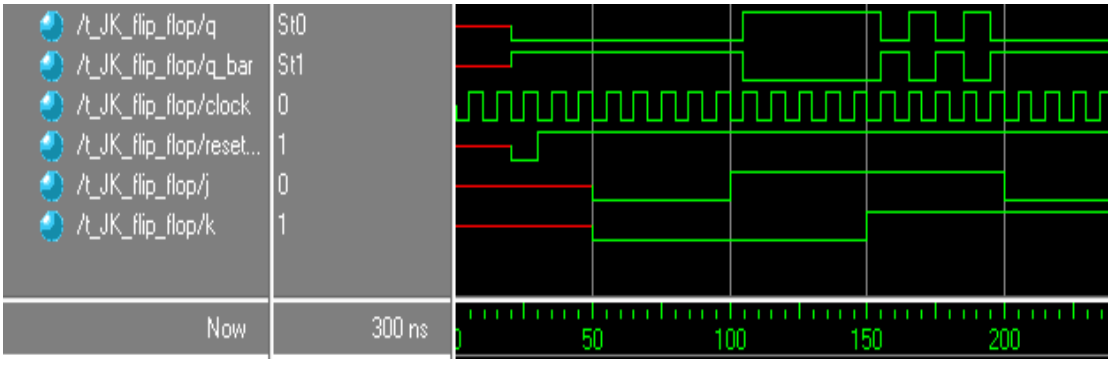
```
module t_Divide_by_11();
    wire clk_by_11;
    reg clk, rst_b;
    supply1 Vcc;
    Divide_by_11 M0 (clk_by_11, clk, rst_b, Vcc);
initial #500 $finish;
initial begin clk = 0; forever #5 clk = ~clk; end
initial begin #10 rst_b = 1; #20 rst_b = 0; #20 rst_b = 1; end

endmodule
```

```
primitive T_prim (output reg Q, input clk, rst_b, toggle);
table
// clk rst_b toggle state Q/next_state
? 0 ? : ? : 1 ; // rst_b seting flip flop to 1
(01) 1 0 : ? : - ; // Rising clock edge
(01) 1 1 : 0 : 1 ;
(01) 1 1 : 1 : 0 ;
(1?) 1 ? : ? : - ; // Falling clock
? 1 (??) : ? : - ; // Steady clock, ignore data
? (?1) ? : ? : - ; // Steady clock, ignore data
endtable // transitions
endprimitive

module Divide_by_11 (output clk_by_11, input clk, rst_b, Vcc);
        wire Q0_LSB, Q1, Q2, Q3_MSB, w1, w2;
        T_prim T0 (Q0_LSB, clk, clk_by_11, Vcc);
        T_prim T1 (Q1, Q0_LSB, clk_by_11, Vcc);
        T_prim T2 (Q2, Q1, clk_by_11, Vcc);
        T_prim T3 (Q3_MSB, Q2, clk_by_11, Vcc);
        or (w2, w1, Q);
        and (w1, Q0_b, Q1, Q2, Q3_b);
        not (Q0_b, Q0_LSB);
        not (Q3_b, Q3_MSB);
        T_prim T4 (Q, clk, rst_b, w2);
        not (clk_by_11, Q);
endmodule

module t_Divide_by_11();
        wire clk_by_11;
        reg clk, rst_b;
        supply1 Vcc;
        Divide_by_11 M0 (clk_by_11, clk, rst_b, Vcc);
initial #500 $finish;
initial begin clk = 0; forever #5 clk = ~clk; end
initial begin #10 rst_b = 1; #20 rst_b = 0; #20 rst_b = 1; end
endmodule
```



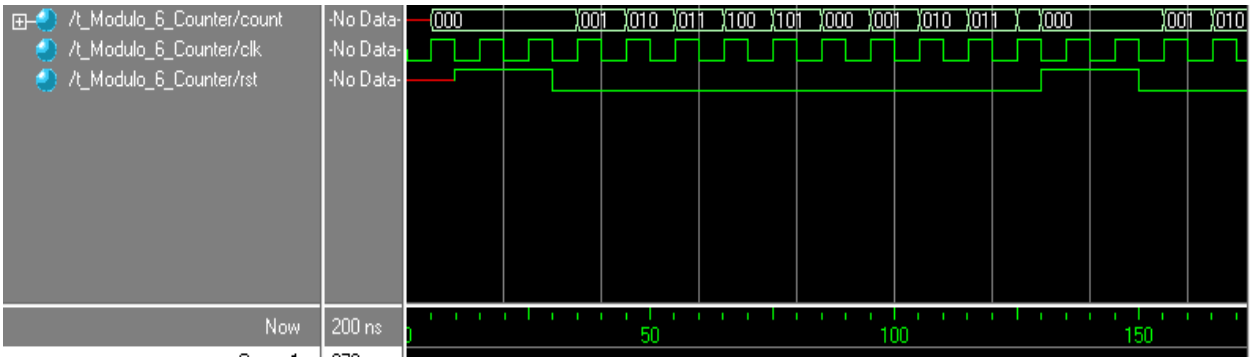
Q.2. Write a behavioral Verilog model of a rising-edge triggered J-K flip-flop with active-low asynchronous reset. Write a test bench to verify the correctness of your model.

```
module JK_flip_flop (output reg q, output q_bar, input j, k, clock, reset_bar);
assign q_bar = ~q;
always @( posedge clock, negedge reset_bar)
        if (reset_bar == 0) q <= 0;
        else case ({j,k})
                2'b00: q <= q;
                2'b01: q <= 0;
                2'b10: q <= 1;
                2'b11: q <= ~q;
        endcase
endmodule

module t_JK_flip_flop ();
        wire q, q_bar;
        reg clock, reset_bar;
        reg j, k;
        JK_flip_flop M0 (q, q_bar, j, k, clock, reset_bar);
initial #500 $finish;
initial begin clock = 0; forever #5 clock = ~clock; end
initial fork
#20 reset_bar = 0;
#30 reset_bar = 1;
#50 j = 0;
#50 k = 0;
#100 j = 1;
#150 k = 1;
#200 j = 0;
join
endmodule
```

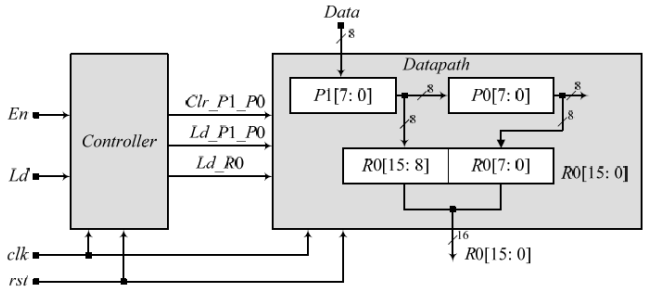


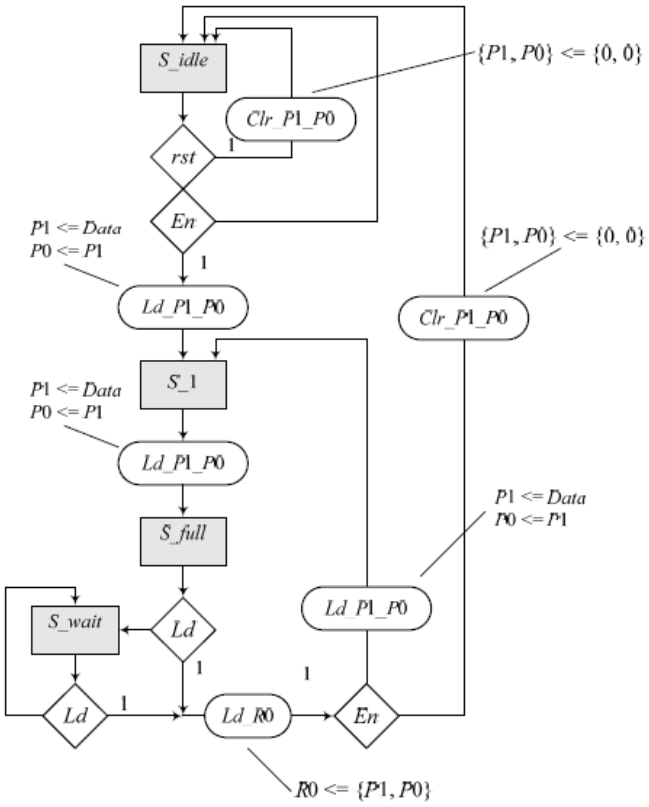
Q.3. Write a behavioral Verilog model of a modulo-6 counter. Assume that the flip-flops are rising-edge triggered and the reset is asynchronous and active high. Write a test bench to verify the correctness of your model.

```
// Assumption: positive-edge sensitive, active-high reset.
// Assumption: count from reset value
module Modulo_6_Counter (
        output reg [2: 0] count,
        input clk, rst
);
always @ (posedge clk, posedge rst)
        if (rst == 1) count <= 0;
        else if (count < 5) count <= count + 1;
        else count <= 0;
endmodule
module t_Modulo_6_Counter ();
        wire [2: 0] count;
        reg clk, rst;
        Modulo_6_Counter M0 (count, clk, rst);
initial #200 $finish;
initial begin
        clk = 0;
        forever #5 clk = ~clk;
end
initial begin
#10 rst = 1;
#20 rst = 0;
#100 rst = 1; // Reset 0n-the-fly
#20 rst = 0;
end
endmodule
```



Q.4. Write a behavioral Verilog model of the data path unit described below. Assume that the flip-flops are rising-edge triggered and the reset is synchronous and active high. Write a test bench to verify the correctness of your model.






```
module PipeDataPath (
output reg [15: 0] R0,
input [7:0] Data,
input Clr_P1_P0, Ld_P1_P0, Ld_R0, clock, rst
);
reg [7: 0] P1, P0;
always @ (posedge clock) begin
```

```
if (rst == 1'b1) begin P1 <= 0; P0 <= 0; R0 <= 0; end
else begin
if (Clr_P1_P0) begin P0 <= 0; P1 <= 0; end
if (Ld_P1_P0) begin P1 <= Data; P0 <= P1; end
if (Ld_R0) R0 <= {P1, P0};
end
end
endmodule
// Test Plan
// verify power-up reset
// Verify pipeline action
// Verify load of R0 action
module t_PipeDataPath ();
wire [15: 0] R0;
reg [7:0] Data;
reg Clr_P1_P0, Ld_P1_P0, Ld_R0, clock, rst;
PipeDataPath M0(R0, Data, Clr_P1_P0, Ld_P1_P0, Ld_R0, clock, rst);
initial #700 $finish;
initial begin clock = 0; forever #5 clock = ~clock; end
initial fork
Data = 8'haa;
#10 rst = 0; // Power-up reset
#20 rst = 1;
#50 rst = 0;
#20 Ld_P1_P0 = 0;
#70 Ld_P1_P0 = 1;
#80 Ld_P1_P0 = 0;
#120 Ld_R0 = 1;
#130 Ld_R0 = 0;
#200 Data = 8'hbb;
#210 Ld_P1_P0 = 1;
#220 Ld_P1_P0 = 0;
#230 Ld_R0 = 1;
#240 Ld_R0 = 0;
#250 Clr_P1_P0 = 1;
#260 Clr_P1_P0 = 0;
join
endmodule
```