Exercise 1

Wednesday, April 7, 2021 2:00 PM

1. $S = x \oplus y \oplus Q$

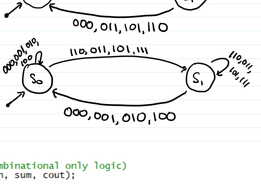
$C = xy + yQ + xQ$

State diagram:

$S \rightarrow$

$S_0 = 0$

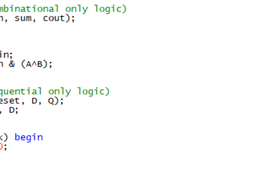
$S_1 = 1$



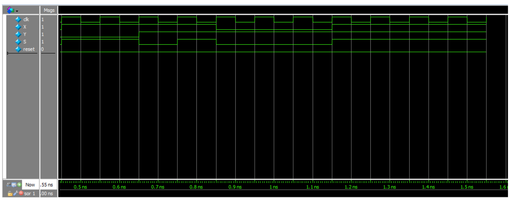
$C \rightarrow$

$S_0 = 0$

$S_1 = 1$

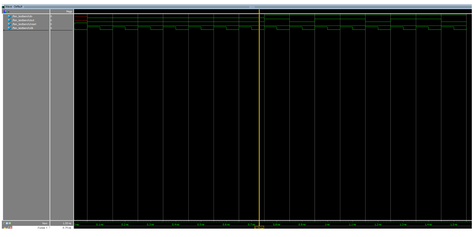



```
// Full adder module, (combinational only logic)
module fullAdder(A, B, cin, sum, cout);
    input logic A,B,cin;
    output logic sum,cout;

    assign sum = A ^ B ^ cin;
    assign cout = A&B | cin & (A^B);
endmodule

// D flip flop module (sequential only logic)
module D_Flip_Flop(clk, reset, D, Q);
    input logic clk, reset, D;
    output logic Q;

    always_ff @(posedge clk) begin
        if (reset) Q <= 1'b0;
        else Q <= D;
    end
endmodule

// Top level module (takes input x, y)
module DE1_SOC(X, Y, clk, S, reset);
    input logic X, Y, clk, reset;
    logic Q, C;
    output logic S;

    // Full adder is connected to the D flip flop as specified in the question
    fullAdder FA (.A(X), .B(Y), .cin(Q), .sum(S), .cout(C));
    D_Flip_Flop FL (.clk(clk), .reset(reset), .D(C), .Q(Q));
endmodule


module DE1_SOC_testbench();
    logic X, Y, S, clk, reset;

    DE1_SOC dut (.X, .Y, .clk, .S, .reset);

    // clock setup
    parameter clock_period = 100;

    initial begin
        clk <= 0;
        X <= 0;
        Y <= 0;
        reset <= 0;
        forever #(clock_period / 2) clk <= ~clk;
    end // initial

    initial begin
        reset <=1;    @(posedge clk);
                      @(posedge clk);
        reset <=0;    @(posedge clk);
                      @(posedge clk);
                      @(posedge clk);
                      @(posedge clk);
        X <= 1;       @(posedge clk);
                      @(posedge clk);
        Y <= 1;       @(posedge clk);
                      @(posedge clk);
        X <= 0; Y <=1; @(posedge clk);
                       @(posedge clk);
                       @(posedge clk);

        X<= 1;        @(posedge clk);
                      @(posedge clk);
                      @(posedge clk);
                      @(posedge clk);

        $stop;
    end // initial
endmodule
```



2.

```
module fsm (clk, reset, in, out);
    input logic clk, reset, in; //input for clk, reset, an a bit input
    output logic out; //output bit

    enum {S0, S1, S2, S3, S4} ps, ns; //all five states, present state, next state

    always_comb begin //comb next state logic
        case (ps)
            S0: if (in) ns = S4;
                else ns = S2;
            S1: if (in) ns = S4;
                else ns = S3;
            S2: if (in) ns = S1;
                else ns = S1;
            S3: if (in) ns = S0;
                else ns = S3;
            S4: if (in) ns = S3;
                else ns = S3;
        endcase
    end

    // equation for output as it depends on present state and input, in
    assign out = ((ps == S0) & in) | ((ps == S1) & !in) | ((ps == S2) & !in) | ((ps == S3) & in));

        //sequential logic (DFFs)
        always_ff @(posedge clk) begin
            if (reset)
            ps <= S0;
            else
            ps <= ns;
        end
endmodule

module fsm_testbench();

    logic clk, reset, in, out;

    //devices under test
    fsm dut (.clk, .reset, .in, .out);

    //clock setup
    parameter clock_period = 100;

    initial begin
        clk <= 0;
        forever #(clock_period /2) clk <= ~clk;
    end
    //initial simulation
    initial begin
        reset <= 1;             @(posedge clk);
        reset <= 0; in<=0;      @(posedge clk);
        reset <= 0; in<=0;      @(posedge clk);
        reset <= 0; in<=0;      @(posedge clk);
        reset <= 0; in<=0;      @(posedge clk);
        reset <= 0; in<=0;      @(posedge clk);
        reset <= 0; in<=1;      @(posedge clk);
        reset <= 0; in<=1;      @(posedge clk);
        reset <= 0; in<=1;      @(posedge clk);
        reset <= 0; in<=1;      @(posedge clk);
        reset <= 0; in<=1;      @(posedge clk);
        reset <= 0; in<=1;      @(posedge clk);
        reset <= 0; in<=1;      @(posedge clk);
        reset <= 0; in<=1;      @(posedge clk);

        $stop; //end simulation
    end
endmodule
```



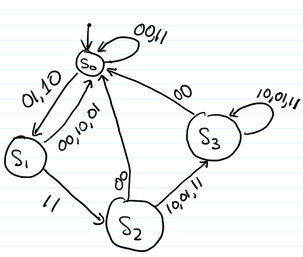
3.

$S_0 = 00$

$S_1 = 01$

$S_2 = 10$

$S_3 = 11$



4. When using non blocking, both modules are the same even though it is different in the placement of the x and y, on the posedge clk, both x and y values gets updated at the same time, allowing x to get its value through input a and b, and then y would get its value through x and c. This is true in both code when it is non blocking. However, it is the opposite for blocking as x has to be updated before y in order for y to capture the value of x through inputs a and b. When it is blocking, the line of code has to execute before the next line, so the order is extremely important when using blocking.