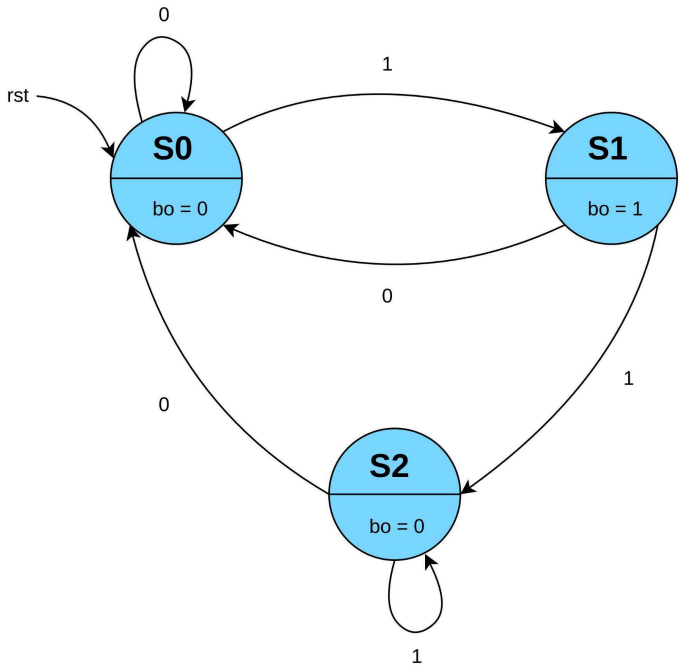
Module: R3: DLD + DSD
Section: Sequential Circuits **Task:** Assignment 2

Assignment 2
Sequential Circuits

➢ **Question 1: Design button press synchronizer:**

a. FSM Diagram:



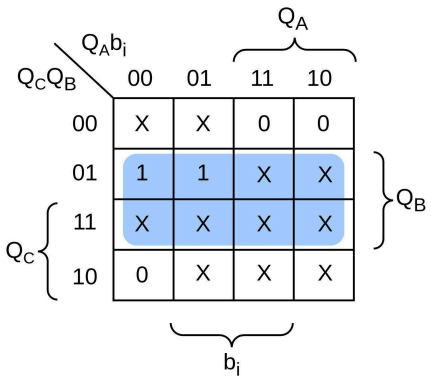
b. Truth Table:

One-Hot Encoding	S0 = 001	S1 = 010	S2 = 100

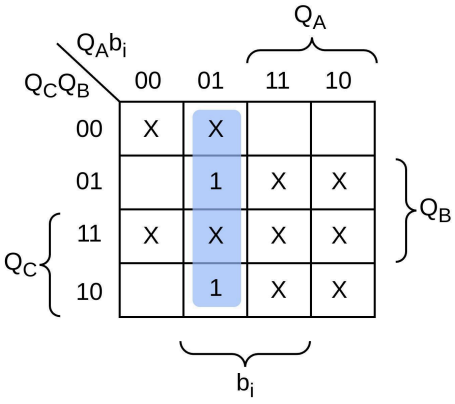
Present State			Input	Next State			Output	Inputs to Flip-Flops		
Q_C	Q_B	Q_A	b_i	Q_{C+1}	Q_{B+1}	Q_{A+1}	b_o	D_C	D_B	D_A
0	0	1	0	0	0	1	0	0	0	1
0	0	1	1	0	1	0	0	0	1	0
0	1	0	0	0	0	1	1	0	0	1
0	1	0	1	1	0	0	1	1	0	0
1	0	0	0	0	0	1	0	0	0	1
1	0	0	1	1	0	0	0	1	0	0

c. Circuit Diagram:

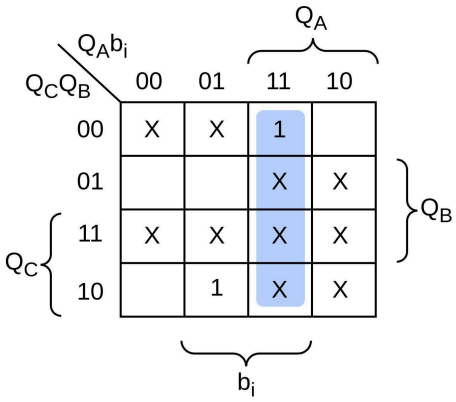
Using K-Maps:



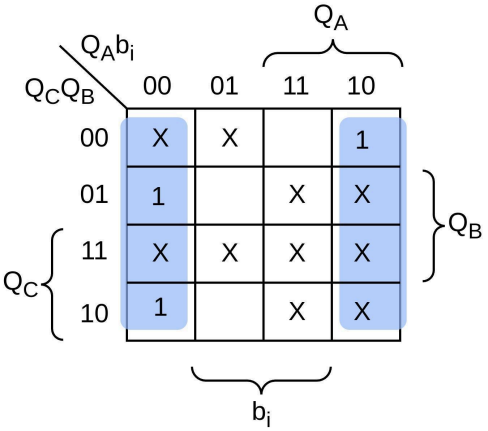
$b_o = Q_B$



$D_C = Q_A' b_i$

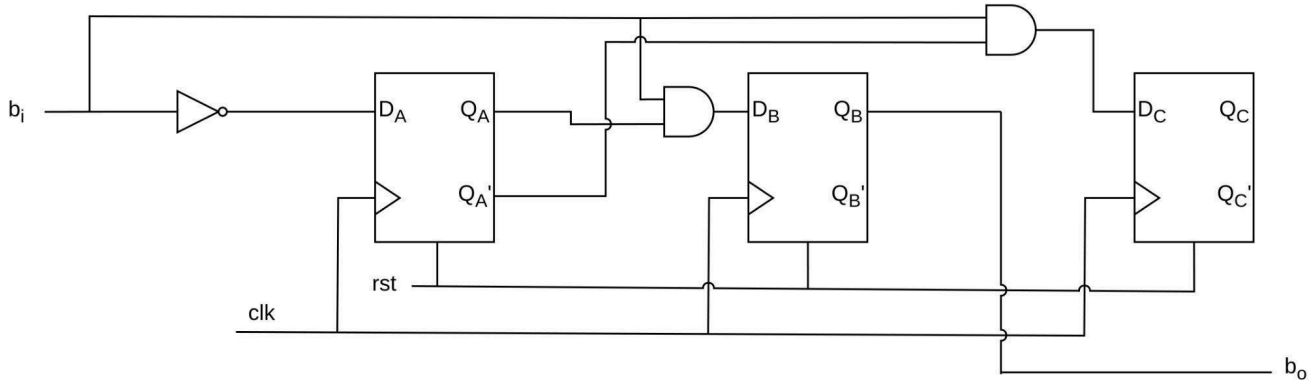


$D_B = Q_A b_i$



$D_A = b_i'$

Here's the final circuit:



d. Verilog Code:

```
    module d_ff(input clk, rst, d, output reg q);

    always @(posedge clk or posedge rst) begin
        if (rst)
            q <= 0;
        else
            q <= d;
    end
    endmodule

    module d_ff2 (input clk, rst, d, output reg q);

    always @(posedge clk or posedge rst) begin
        if (rst)
       //LSB = 1 for 1-Hot Encoding
            q <= 1;
        else
            q <= d;
    end
    endmodule

module button_press_synchronizer (input clk, rst, bi, output reg
bo);
```

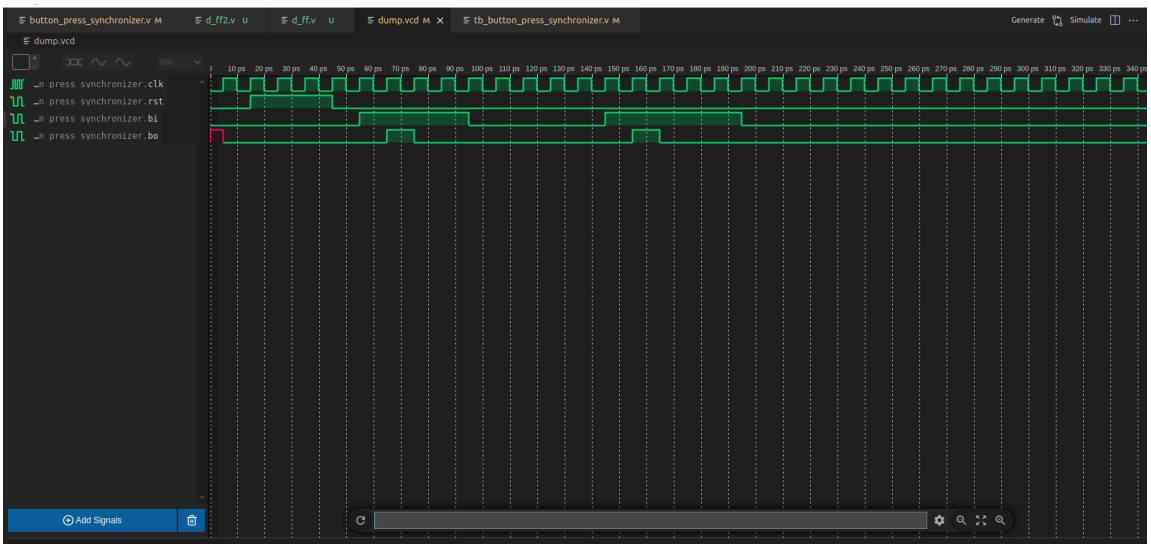
```
        wire Da, Db, Dc, Qa, Qb, Qc;

        d_ff2 A(.d(Da), .clk(clk), .rst(rst), .q(Qa));
        d_ff B(.d(Db), .clk(clk), .rst(rst), .q(Qb));
        d_ff C(.d(Dc), .clk(clk), .rst(rst), .q(Qc));


        assign Da = ~bi;
        assign Db = bi & Qa;
        assign Dc = ~Qa & bi;
        assign bo = Qb;
endmodule
```

e. Testbench:

```
    module tb_button_press_synchronizer;
    reg clk, rst, bi;
    wire bo;

    button_press_synchronizer m1 (.clk(clk), .rst(rst), .bi(bi),
.bo(bo));

    initial begin
        clk = 0;
        rst = 0;
        bi = 0;
    end

    always #5 clk = ~clk;
    initial begin

        $dumpvars;

        repeat(2) @(posedge clk);

        rst = 1;
        repeat(3) @(posedge clk);

        rst = 0;
        @(posedge clk);
        bi = 1;
```

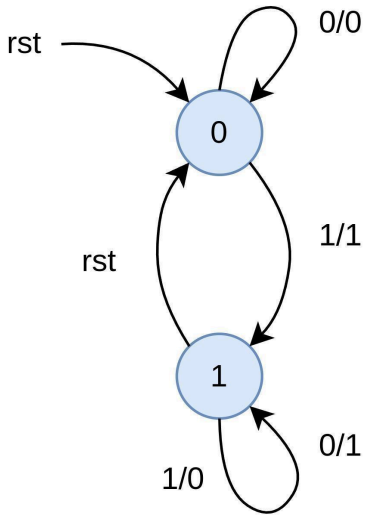
```
        repeat (1.5) @(posedge clk)
        $display("Checking for Output Pulse");
        if (bo != 1) begin
            $display("Output is Incorrect Error: Line: 31");
            $finish;
            end

        repeat (1.5) @(posedge clk);
        $display("Checking if Output had one cycle");
        if (bo != 0) begin
            $display("Output is Incorrect Error: Line: 38");
            $finish;
            end

        bi = 0;

        repeat(5) @(posedge clk);

        //Asserting Input again
        bi = 1;
        repeat(5) @(posedge clk);
        bi = 0;

        if (bo != 0) begin
            @(posedge clk)
            if (bo != 1) begin
            $display("Output is Incorrect Line: 49");
            $finish;
            end
        end

        repeat(15) @(posedge clk);


        $display("All Test cases Passed!");
        $finish;
    end

    endmodule
```

f. Output:



➢ **Question 2: Serial Input 2's Complementer:**

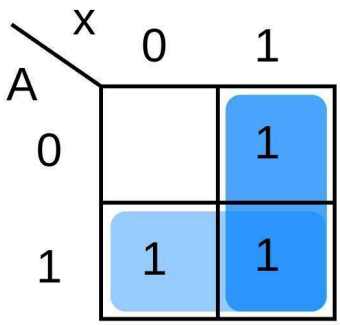
g. State Diagram:



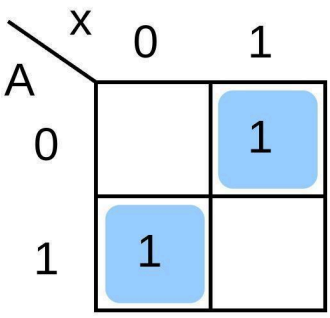
h. State Table:

Present State	Input	Next State	Output	Flip-Flop Input
Qt	x	Q(t+1)	y	D
0	0	0	0	0
0	1	1	1	1
1	0	1	1	1
1	1	1	0	1

K-maps:

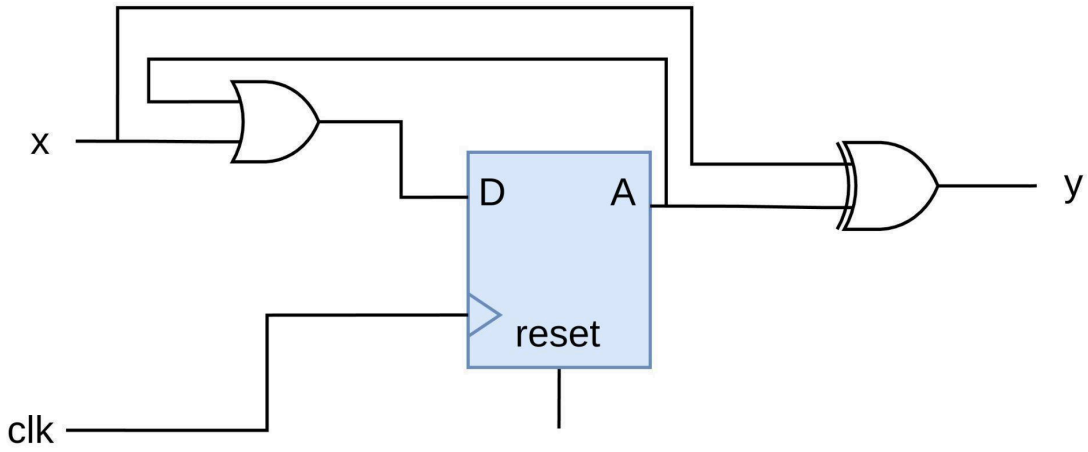


D = x + A

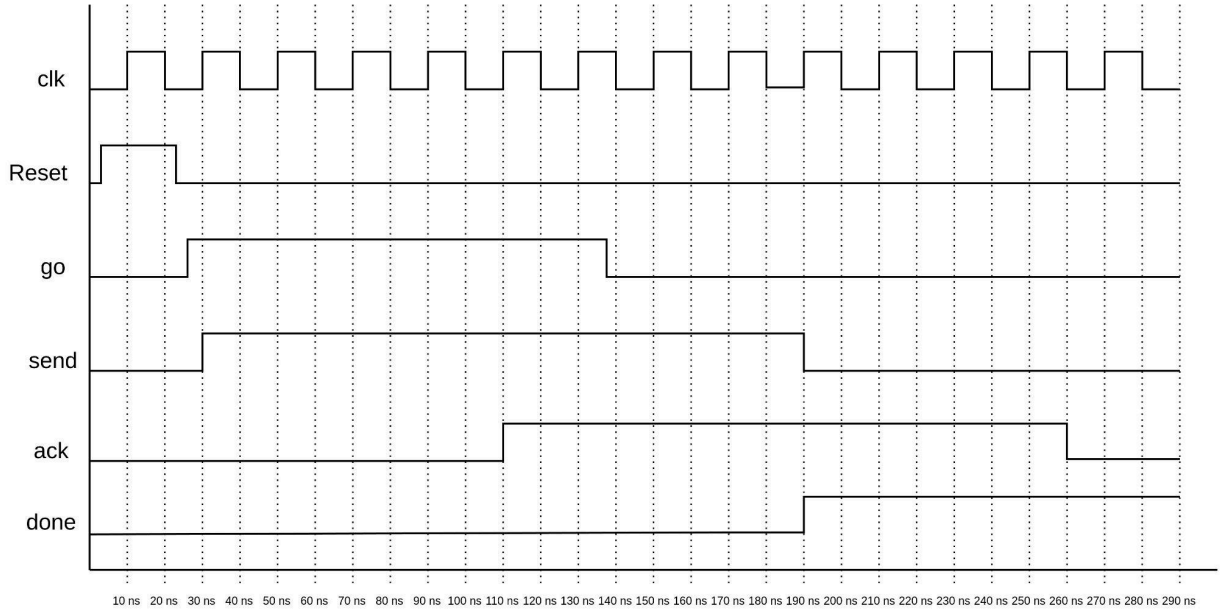


$y = x \oplus A$

i. Circuit Diagram:

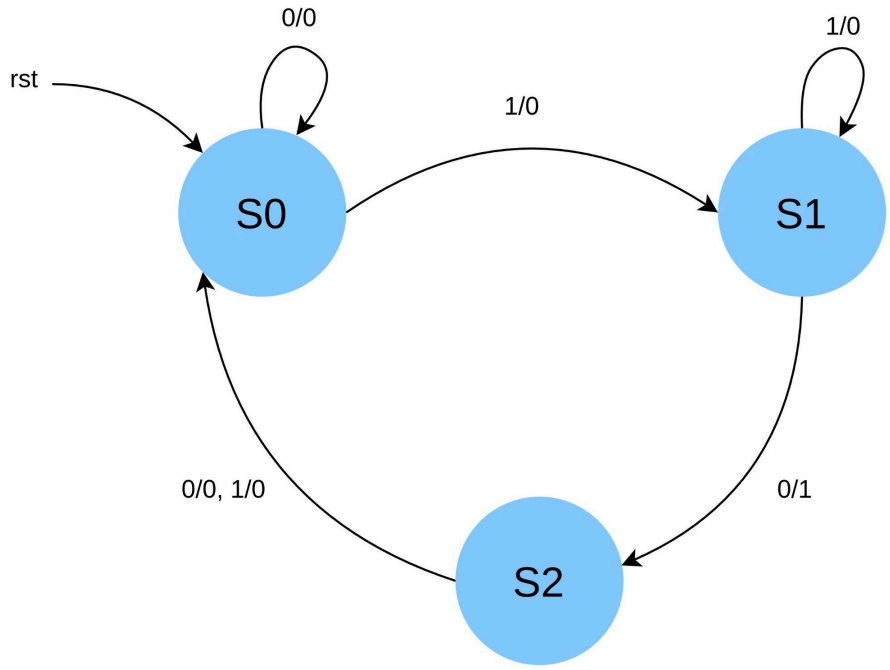


➢ **Question 3: Timing Diagram:**



➢ **Question 4: Pulse Detector:**

a. FSM Diagram:

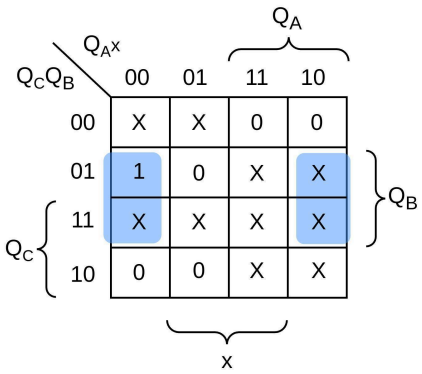


b. Truth Table:

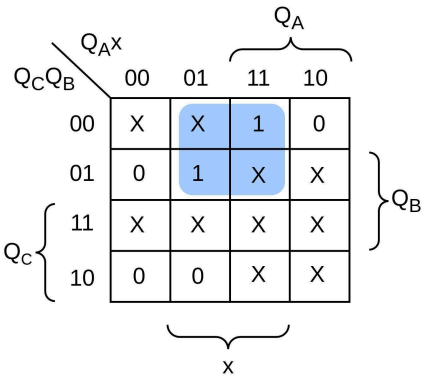
One-Hot Encoding	S0 = 001	S1 = 010	S2 = 100

Present State			Input	Next State			Output	Inputs to Flip-Flops		
Q_C	Q_B	Q_A	x	Q_{C+1}	Q_{B+1}	Q_{A+1}	y	D_C	D_B	D_A
0	0	1	0	0	0	1	0	0	0	1
0	0	1	1	0	1	0	0	0	1	0
0	1	0	0	1	0	0	1	1	0	0
0	1	0	1	0	1	0	0	0	1	0
1	0	0	0	0	0	1	0	0	0	1
1	0	0	1	0	0	1	0	0	0	1

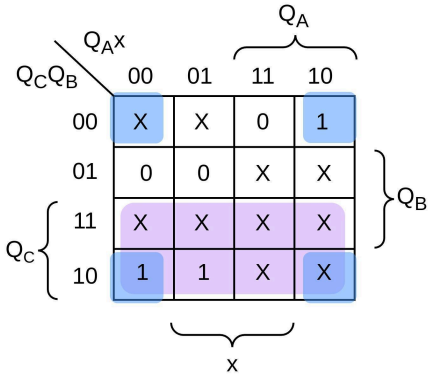
c. Circuit Diagram:



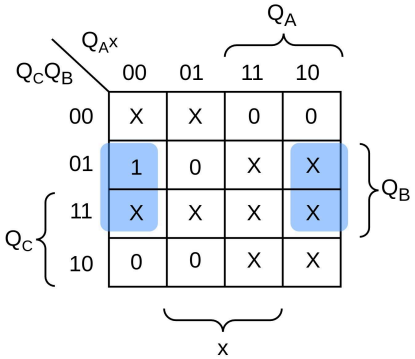
$D_C = x'Q_B$



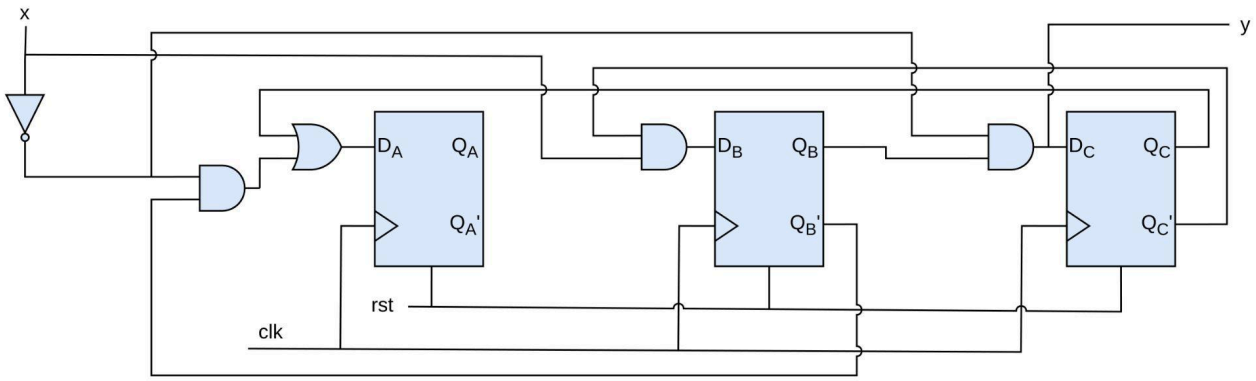
$D_B = xQ_c'$



$D_A = Q_B'x' + Q_C$



$y = x'\,Q_C$



d. Verilog Code:

```
module d_ff (input clk, rst, d, output reg q);

always @(posedge clk or posedge rst) begin
    if (rst)
    q <= 1'b0;
    else
    q <= d;
end

endmodule

module d_ff2 (input clk, rst, d, output reg q);

always @(posedge clk or posedge rst) begin
    if (rst)
    //LSB = 1 for 1-Hot Encoding
    q <= 1'b1;
    else
    q <= d;
end

endmodule



    module pulse_detector (input clk, rst, x, output reg y);
```

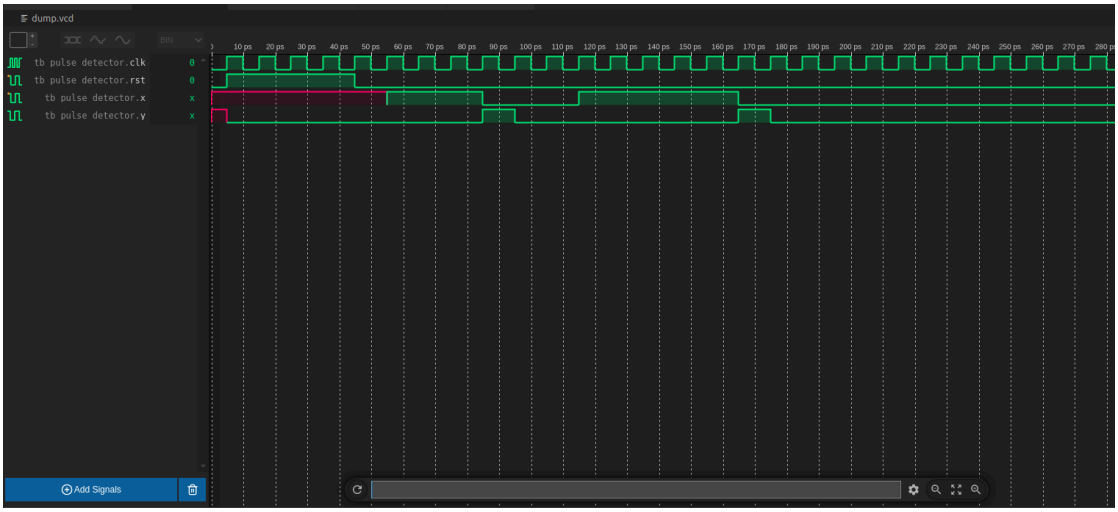
```
    wire Qa, Qb, Qc, Da, Db, Dc;

    d_ff2 A (.d(Da), .clk(clk), .rst(rst), .q(Qa));
    d_ff  B (.d(Db), .clk(clk), .rst(rst), .q(Qb));
    d_ff  C (.d(Dc), .clk(clk), .rst(rst), .q(Qc));

    assign Da = ((~Qb & ~x) | Qc);
    assign Db = ~Qc & x;
    assign Dc = Qb & ~x;

    assign y = Qb & ~x;

    endmodule
```

e. Testbench:

```
module tb_pulse_detector;

reg clk, rst, x;
wire y;

pulse_detector m0 (.clk(clk), .rst(rst), .x(x), .y(y));

always #5 clk = ~clk;

initial begin
    clk = 0;
    rst = 0;
end

initial begin
    $dumpvars;

    @(posedge clk);
    rst = 1;

    repeat(4) @(posedge clk);

    rst = 0;

    $display("Check the Output after reset");
    if (y !=0 ) begin
        $display("Incorrect Output after reset");
```

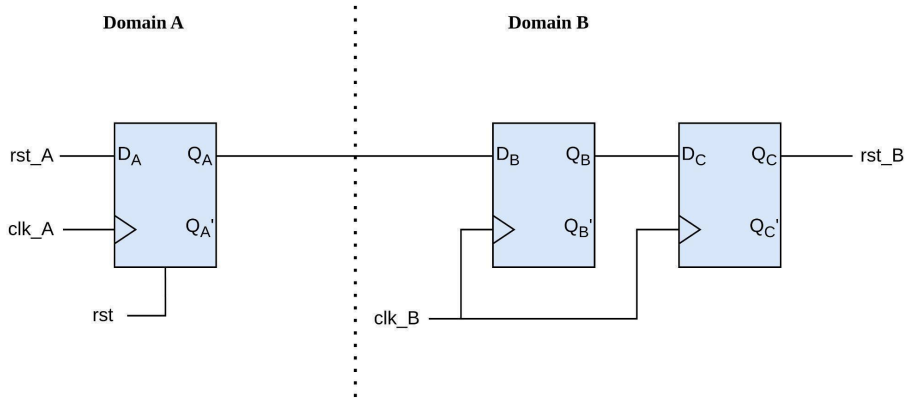
```
    end

    @(posedge clk);
    x = 1;

    repeat(3) @(posedge clk);
    x = 0;

    repeat(2) @(negedge clk);
    if (y != 1) begin
        @(negedge clk);
        if (y != 0) begin
            $display("Incorrect Output");
            $finish;
        end
    end

    @(posedge clk);
    x = 1;

    repeat(5) @(posedge clk);
    x = 0;

    repeat(2) @(negedge clk);
    if (y != 1) begin
        @(negedge clk);
        if (y != 0) begin
            $display("Incorrect Output");
            $finish;
        end
    end


    repeat(10) @(posedge clk);
    $display("All Test Cases Passed!");
    $finish;
end
endmodule
```

f. Output:



➢ **Question 5: Asynhcronus reset:**
Since we are dealing with 2 different clock domains. Hence, in order to avoid any glitches or metastability, I'll be using a two-flop synchronizer ciruit to map the **reset_A** signal synchronously with clock of domain B (**clk_B**)

a. Circuit Diagram:



b. Verilog Code:

```
module d_ff (input clk, rst, d, output reg q);

always @(posedge clk or posedge rst) begin
    if (rst)
        q <= 1'b0;
    else
        q <= d;
end
endmodule
```

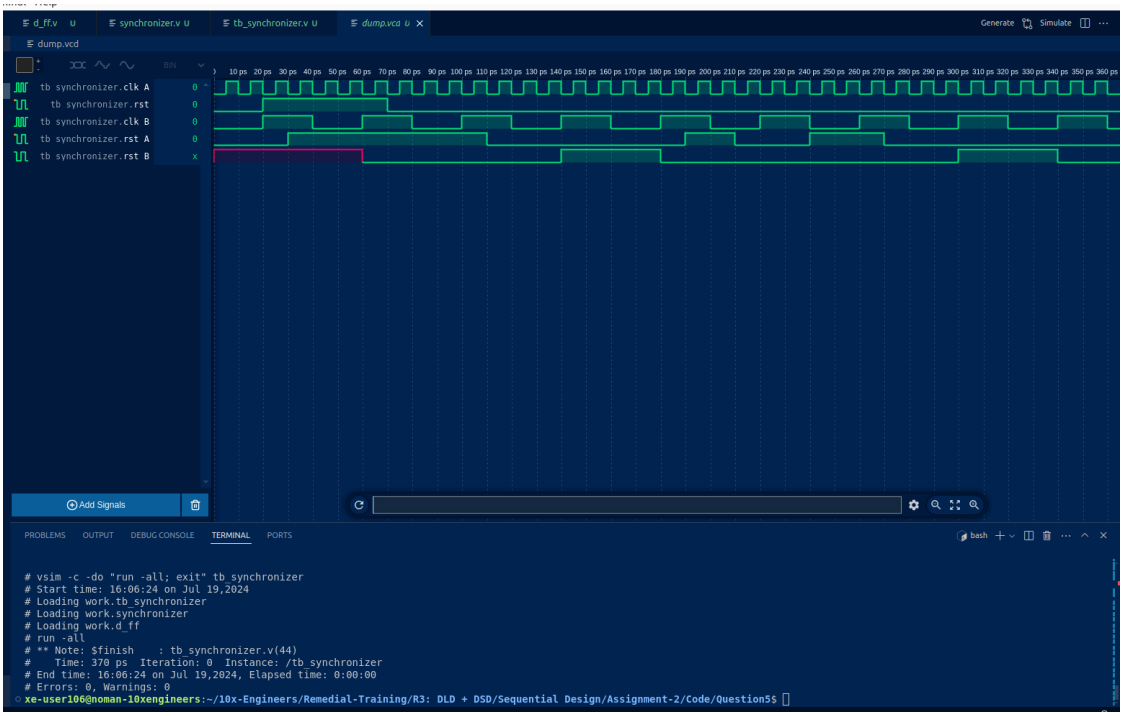
```
    module synchronizer (input clk_A, clk_B, rst_A, rst, output
reg rst_B);

    wire w0, w1, w2;

    d_ff m0 (.clk(clk_A), .rst(rst), .d(rst_A), .q(w0));

    //Two Flops Synchronizer
    d_ff m1 (.clk(clk_B), .rst(1'b0), .d(w0), .q(w1));
    d_ff m2 (.clk(clk_B), .rst(1'b0), .d(w1), .q(w2));

    assign rst_B = w2;

    endmodule
```

c. Testbench:

```
    module tb_synchronizer;
    reg clk_A, clk_B, rst, rst_A;
    wire rst_B;

    synchronizer dut (.clk_A(clk_A), .clk_B(clk_B), .rst(rst),
.rst_A(rst_A), .rst_B(rst_B));

    always #5 clk_A = ~clk_A;
    always #20 clk_B = ~clk_B;

    initial begin
        $dumpvars;

        clk_A = 0;
        clk_B = 0;
        rst = 0;
        rst_A = 0;

        #20;

        rst = 1;
        #10;
```

```
        rst_A = 1;
        #40;

        rst = 0;
        #40;

        rst_A = 0;
        #80;

        rst_A = 1;
        #20;

        rst_A = 0;
        #30;

        rst_A = 1;
        #30;

        rst_A = 0;
        #100;

        $finish;

    end
    endmodule
```

d. Output:


d_ff.v U
synchronizer.v U
tb_synchronizer.v U
dump.vcd U
Generate
Simulate
dump.vcd
tb synchronizer.clk A
tb synchronizer.rst
tb synchronizer.clk B
tb synchronizer.rst A
tb synchronizer.rst B
Add Signals
PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
bash
vsim -c -do "run -all; exit" tb_synchronizer
Start time: 16:06:24 on Jul 19,2024
Loading work.tb_synchronizer
Loading work.synchronizer
Loading work.d_ff
run -all
** Note: $finish : tb_synchronizer.v(44)
Time: 370 ps Iteration: 0 Instance: /tb_synchronizer
End time: 16:06:24 on Jul 19,2024, Elapsed time: 0:00:00
Errors: 0, Warnings: 0
xe-user106@noman-10xengineers:~/10x-Engineers/Remedial-Training/R3: DLD + DSD/Sequential Design/Assignment-2/Code/Question5$