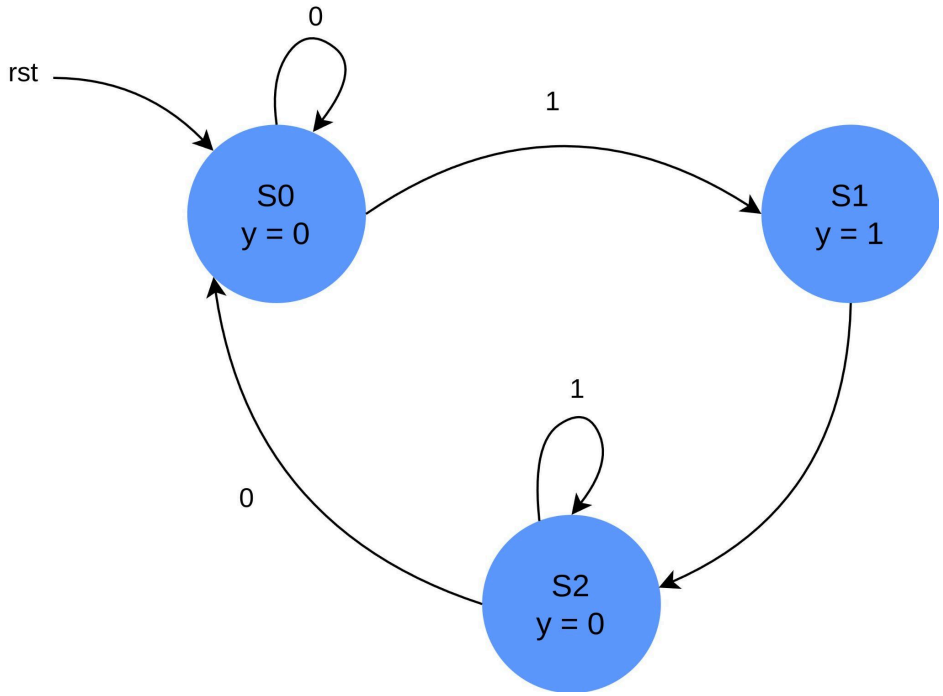
Module: R3: DLD + DSD
Section: Sequential Circuits **Task:** Assignment 2

Assignment 2
Sequential Circuits

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

a. FSM Diagram:



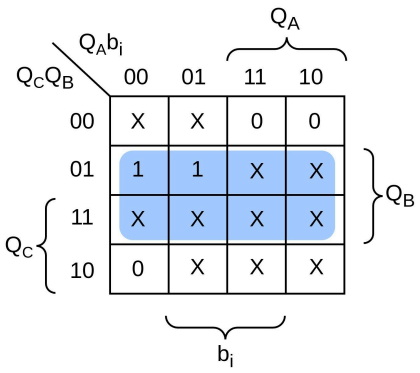
b. Truth Table:

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

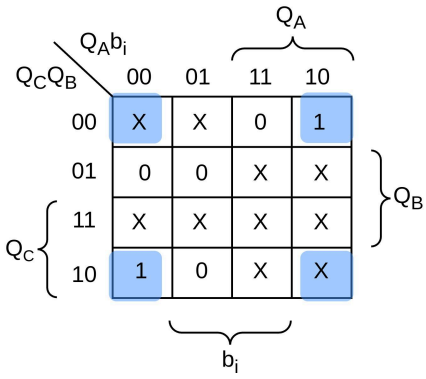
Present State			Input	Next State			Output	Inputs to Flip-Flops		
Q_C	Q_B	Q_A	b_i	Q_{C+1}	Q_{B+1}	Q_{A+1}	b_o	D_C	D_B	D_A
0	0	1	0	0	0	1	0	0	0	1
0	0	1	1	0	1	0	0	0	1	0
0	1	0	0	1	0	0	1	1	0	0
0	1	0	1	1	0	0	1	1	0	0
1	0	0	0	0	0	1	0	0	0	1
1	0	0	1	1	0	0	0	1	0	0

c. Circuit Diagram:

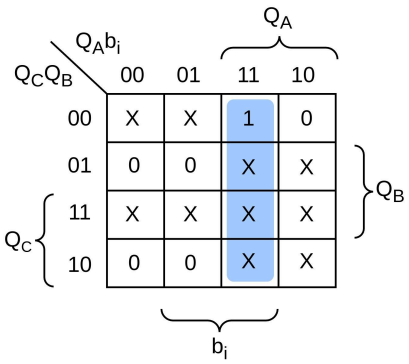
Using K-Maps:



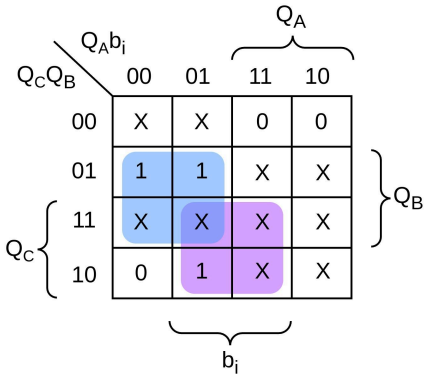
$b_o = Q_B$



$D_A = Q_B'b_i'$

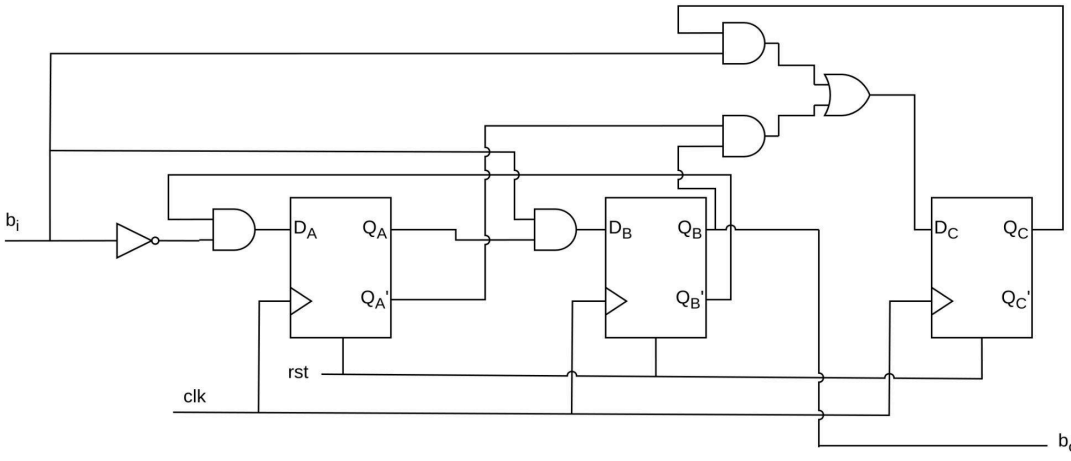


$D_B = Q_Ab_i$



$D_C = Q_BQ_A' + Q_Cb_i$

Here's the final circuit:



d. Verilog Code:

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


        localparam S0 = 3'b001;
        localparam S1 = 3'b010;
        localparam S2 = 3'b100;

        reg [2:0] state;



        always @(posedge clk or posedge rst)
        begin
            if (rst)
                state <= S0;
            else
             begin
                case (state)
                S0 : state <= bi? S1 : S0;

                S1 : state <= S2;

                S2 : state <= bi ? S2 : S0;
            endcase
            end
        end

        always @(state)
            case (state)
            S0 : bo <= 0;

            S1 : bo <= 1;

            S2 : bo <= 0;

            default : bo <= bo;
            endcase
        endmodule
```

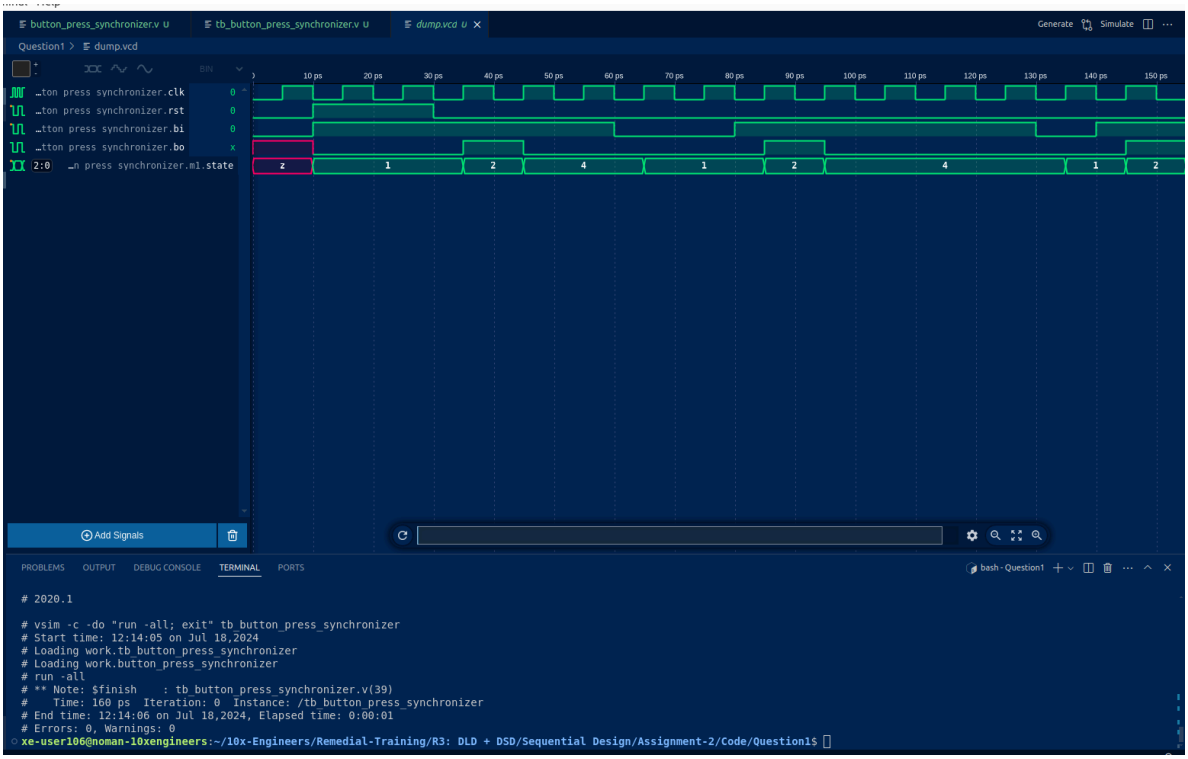
e. Testbench:

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

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

    always #5 clk = ~clk;
    initial begin

        $dumpvars;

        clk = 0;
        rst = 0;
        bi = 0;

        #10;
        rst = 1;
        bi = 1;

        #20
        rst = 0;

        #30;
        bi = 0;

        #20;
        bi = 1;

        #50;
        bi = 0;

        #10;
        bi = 1;

        #20;
        bi = 0;

        $finish;
```

```
    end
```

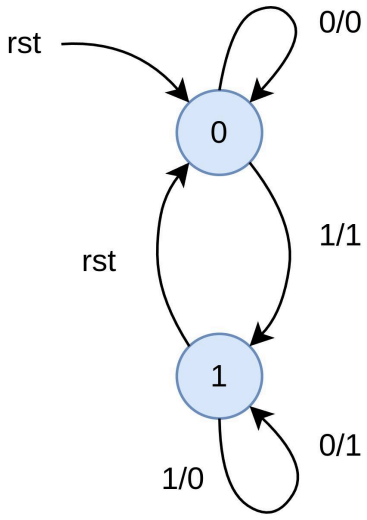
```
    endmodule
```

f. Output:



➢ Question 2: Serial Input 2's Complementer:

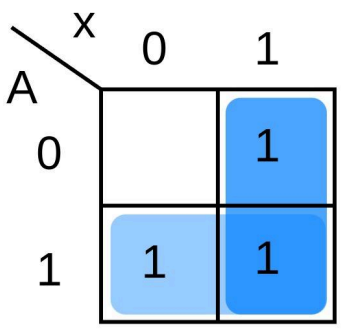
g. State Diagram:



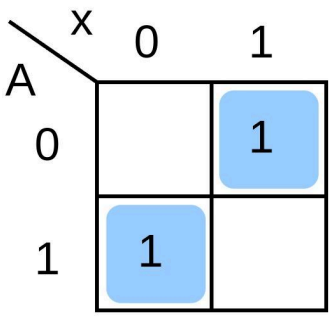
h. State Table:

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:

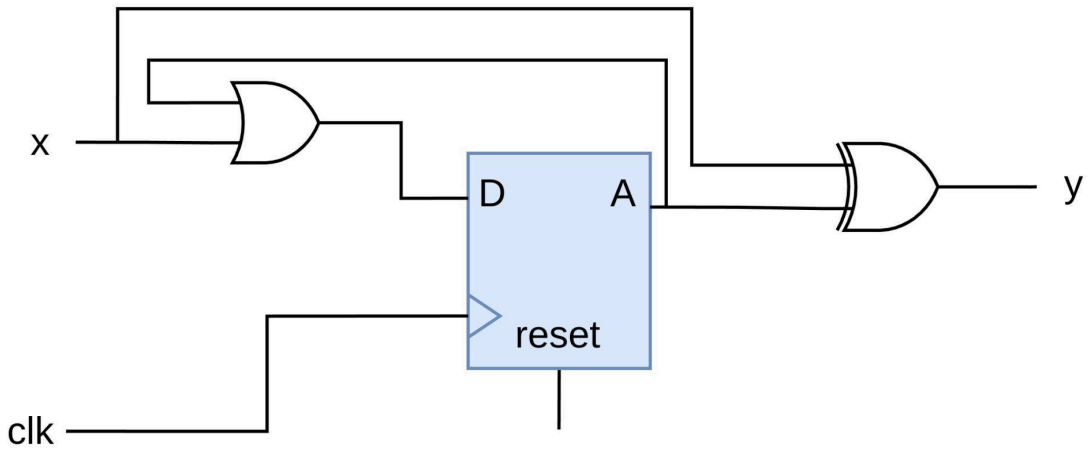


$D = x + A$

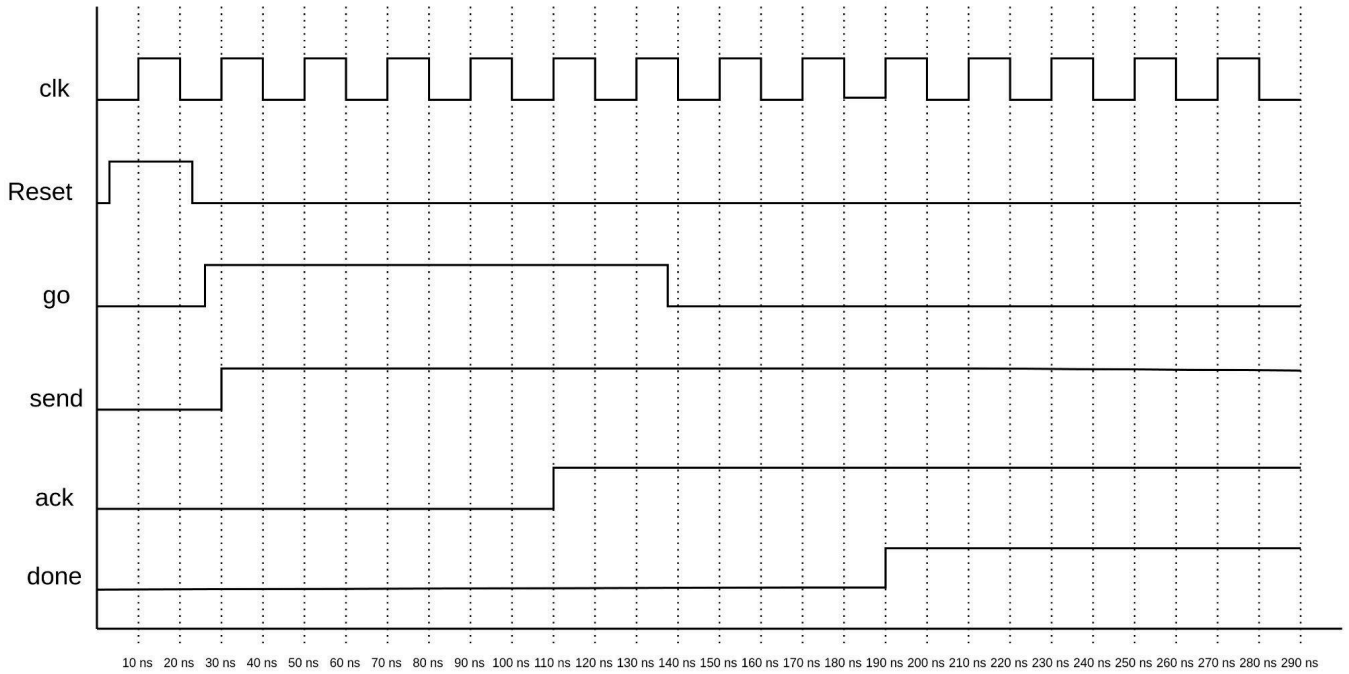


$y = x \oplus A$

i. Circuit Diagram:

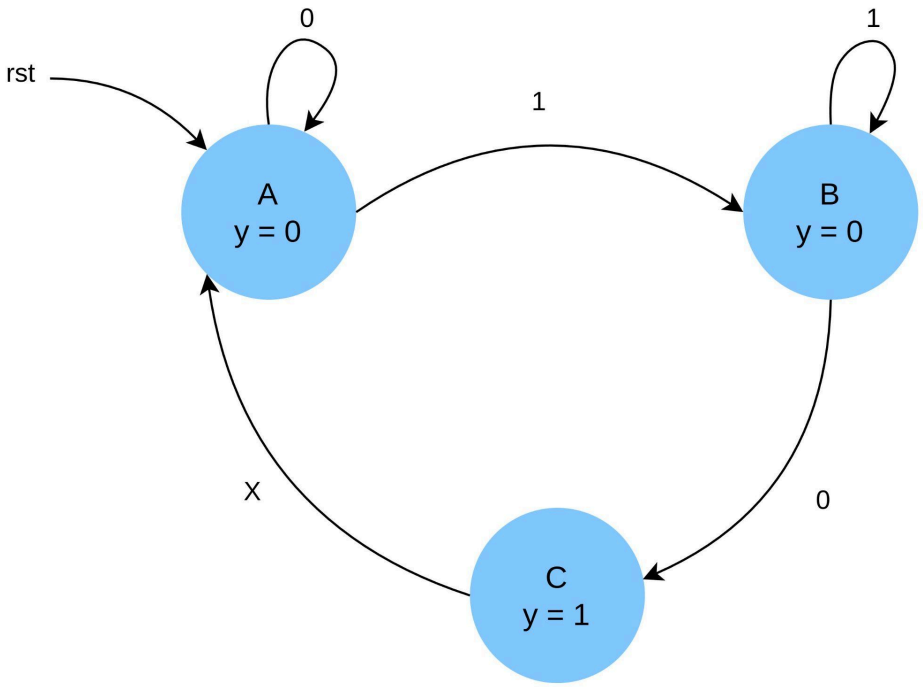


➢ Question 3: Timing Diagram:



➢ Question 4: Pulse Detector:

a. FSM Diagram:

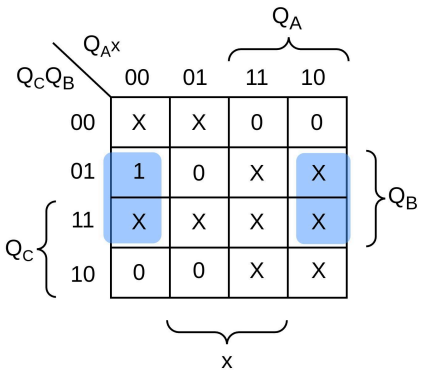


b. Truth Table:

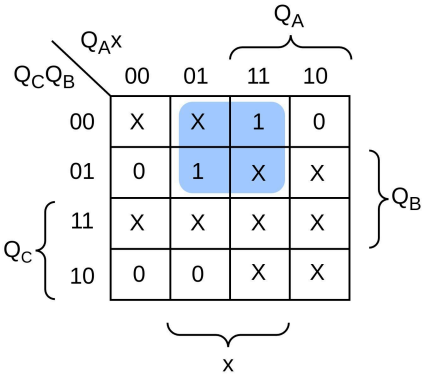
One-Hot Encoding	A = 001	B = 010	C = 100

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	1	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

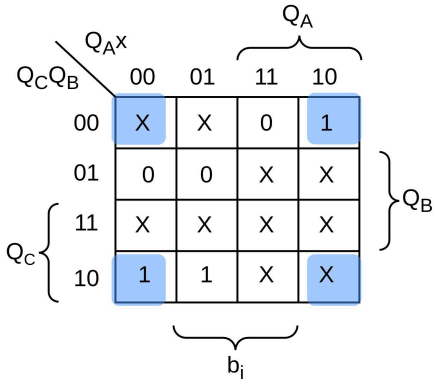
c. Circuit Diagram:



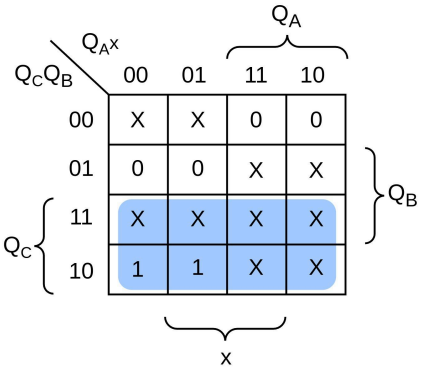
$D_C = x'Q_B$



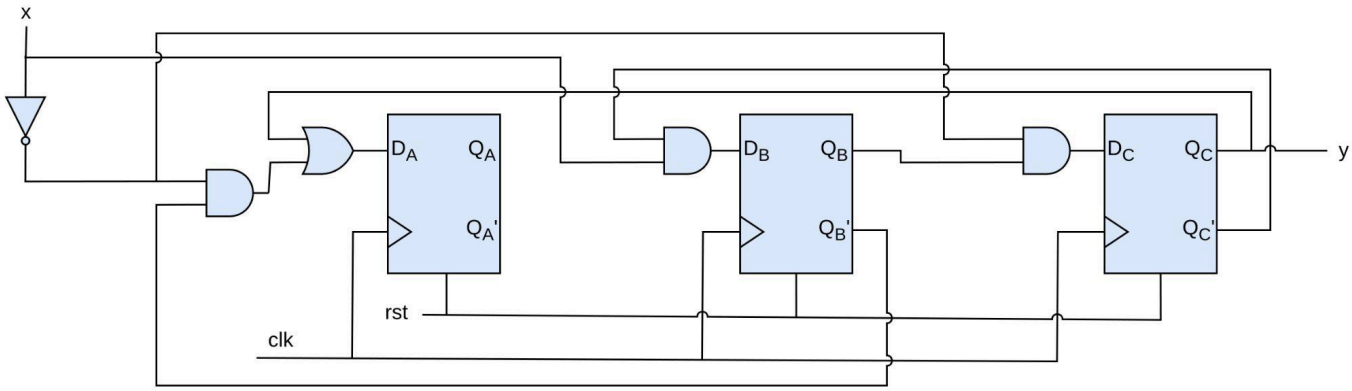
$D_B = xQ_c'$



$D_A = Q_B'x' + Q_C$



$y = Q_C$



d. Verilog Code:

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

localparam A = 3'b001;
localparam B = 3'b010;
localparam C = 3'b100;

reg [2:0] state, next_state;
reg x_previous;

//State-Transition
always @(posedge clk or rst)
begin
if (rst)
    state <= A;
else
state <= next_state;
end

//Next-State Logic
always @(*)
begin
case (state)

A :
begin
    if (!x)
    next_state <= A;
```

```
        else
        next_state <= B;
    end

    B :
    begin
        if (!x && x_previous)
        next_state <= C;
        else
        next_state <= B;
    end

    C : next_state <= A;
    default : next_state <= A;
    endcase
    end

    //Output
    always @(posedge clk or posedge rst)
    begin
        if (rst)
            y <= 0;
        else if (state == C)
            y <= 1;
        else
            y <= 0;
    end

    //Detect Falling-Edge
    always @(posedge clk or posedge rst)
    begin
        if (rst)
            x_previous <= 1;
        else
            x_previous <= x;
    end

    endmodule
```

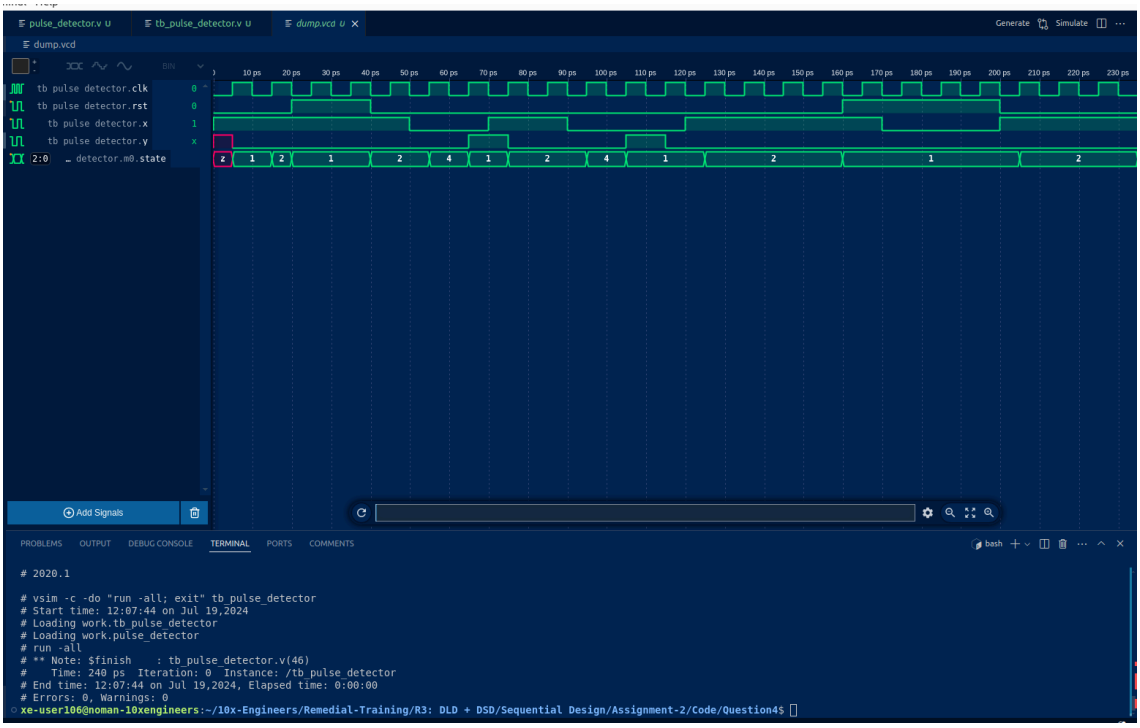
e. Testbench:

```
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
    $dumpvars;

    clk = 0;
    rst = 0;
    x = 1;
    #20;

    rst = 1;
    #20;

    rst = 0;
    #10;

    x = 0;
    #20;

    x = 1;
    #20;

    x = 0;
    #30;

    x = 1;
    #40;

    rst = 1;
    #10;

    x = 0;
    #30;

    x = 1;
    rst = 0;
    #40;
```

```
    $finish;
end
endmodule
```

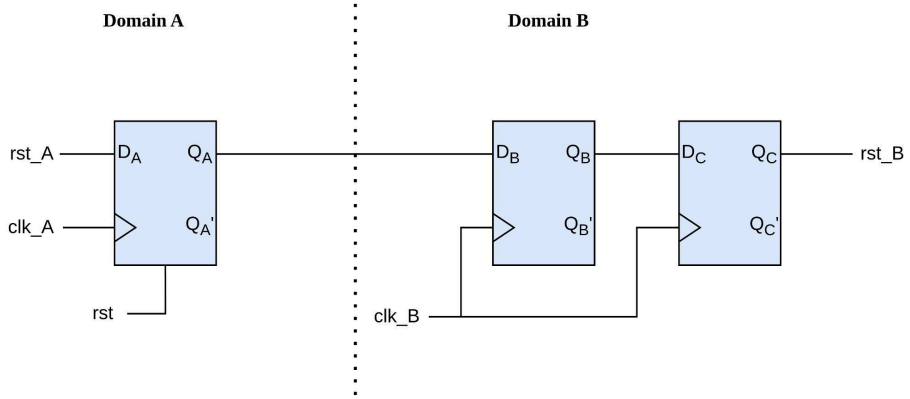
f. Output:



➢ **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
```

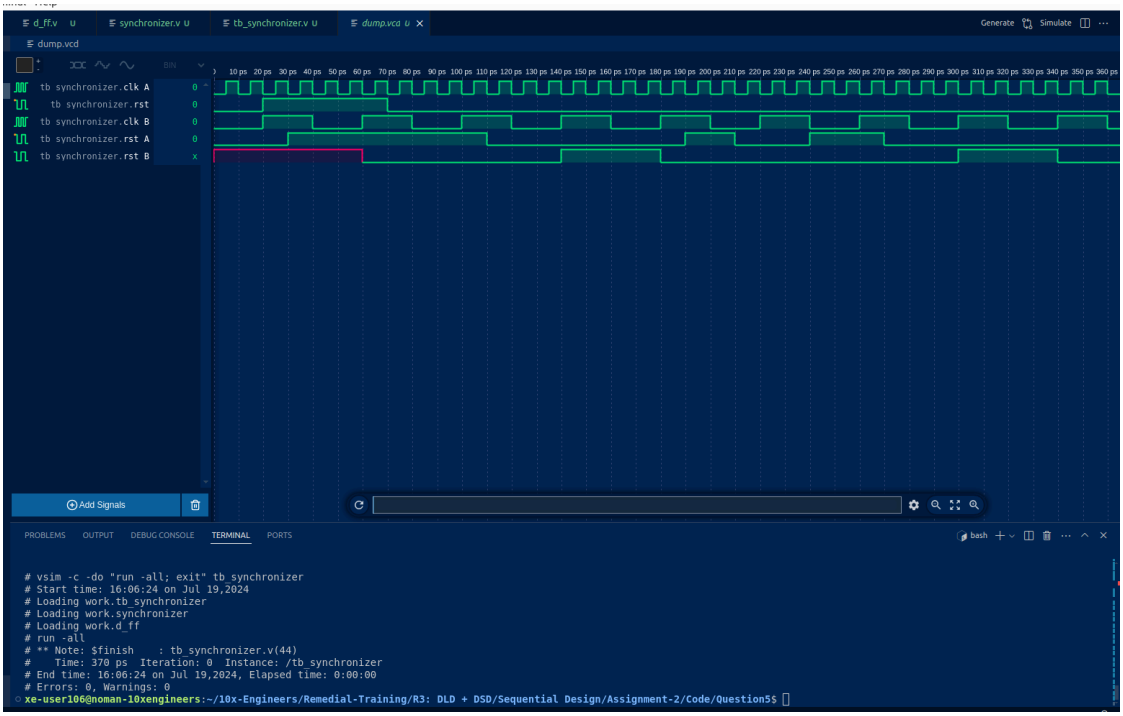
```
    module synchronizer (input clk_A, clk_B, rst_A, rst, output
reg rst_B);

    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: $finish : tb_synchronizer.v(44)
Time: 370 ps Iteration: 0 Instance: /tb_synchronizer
End time: 16:06:24 on Jul 19,2024, Elapsed time: 0:00:00
Errors: 0, Warnings: 0
xe-user106@noman-10xengineers:~/10x-Engineers/Remedial-Training/R3: DLD + DSD/Sequential Design/Assignment-2/Code/Question5$