Module: R3: DLD + DSD
Section: Sequential Circuits **Task:** Assignment 1

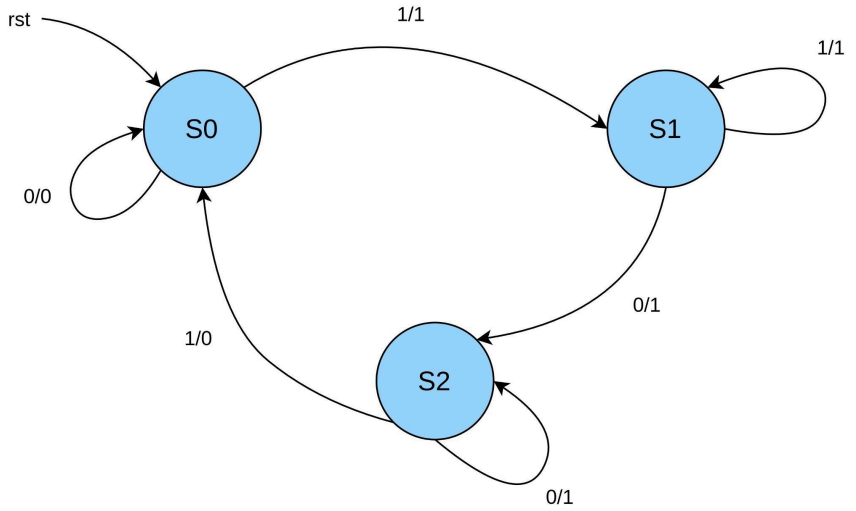
Assignment 1

Sequential Circuits

➢ **Question 1: Design a simple toggle circuit that alternates its output state every time a push-button is pressed. The circuit needs to remember its current state even after the push button is released.**

a. We're using a T flip-flop for the circuit design because it's specifically designed to toggle its output state based on a clock signal and a toggle input (T input). This ensures that the output state changes only when the button is pressed, regardless of how long the button is held. The T flip-flop's edge-triggered behavior makes it ideal for this task, as it changes state only on clock signal transitions, providing reliable and consistent toggling functionality.

b. Design:

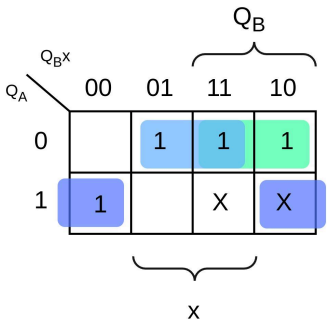


Here's the State-Table:

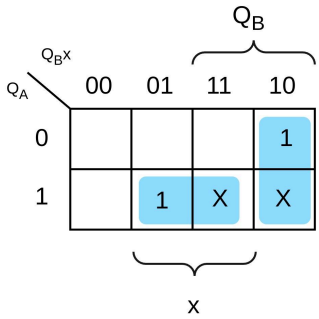
Present State		Input	Next State		Output	FF Inputs	
Q_A	Q_B	x (button)	Q_{A+1}	Q_{B+1}	y	T_A	T_B
0	0	0	0	0	0	0	0
0	0	1	0	1	1	0	1
0	1	0	1	0	1	1	1
0	1	1	0	1	1	0	0
1	0	0	1	0	1	0	0

1	0	1	0	0	0	1	0
X	X	X	X	X	X	X	X
X	X	X	X	X	X	X	X

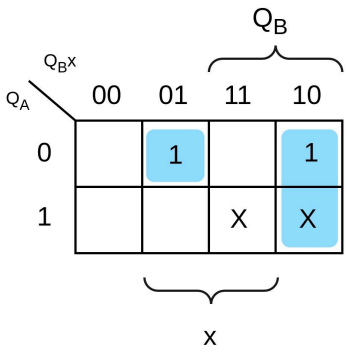
Using K-Maps:



$y = Q_B + (A \wedge x)$

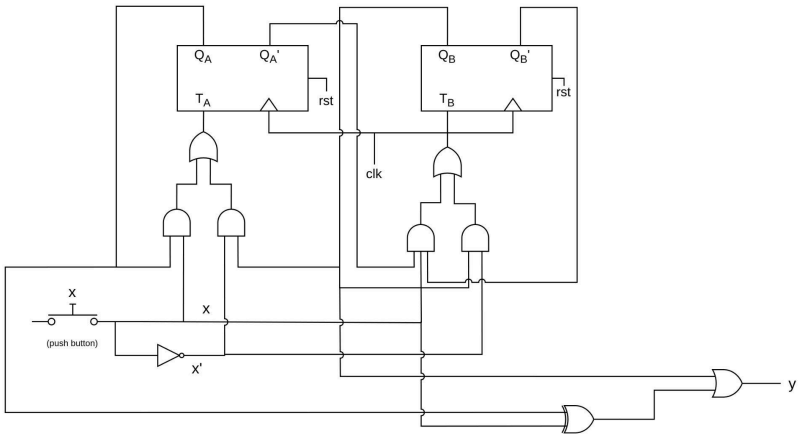


$T_A = Ax + Q_Bx'$



$T_B = Q_A' Q_B' x + Q_Bx'$

- Schematic:



- **Verilog Code:**
  a. **t_ff.v:**

```
module t_ff (input t, clk, rst, output reg q);

 always @(posedge clk or posedge rst)
 begin
     if (rst)
         q <= 1'b0;
     else if (t)
        q <= ~q;
     else
         q <= q;
 end
 endmodule
```

  b. **toggle_circuit.v:**

```
module toggle_circuit (input button, clk, rst, output reg y);

reg Ta, Tb;
wire A, B;

t_ff m0(.t(Ta), .clk(clk), .rst(rst), .q(A));
t_ff m1(.t(Tb), .clk(clk), .rst(rst), .q(B));

always @(*)
begin
    assign Ta = (A & button ) | (B & (~button));
    assign Tb = (~A & ~B & button) | (B & ~button);
end

always @(posedge clk or posedge rst) begin
    if (rst)
        begin
        y <= 1'b0;
        Ta <= 0;
        Tb <= 0;
        end
    else if (button)
        y <= B | (A ^ button);
```

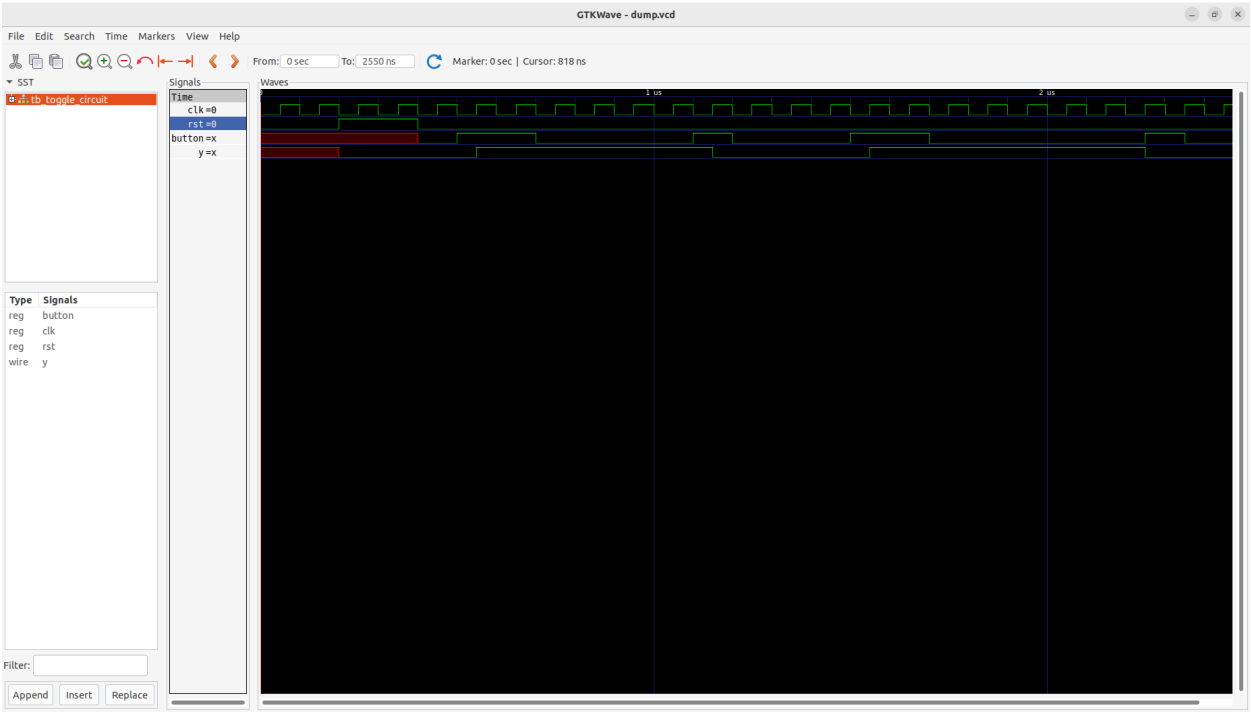
```
end
endmodule
```

➢ **Testbench:**

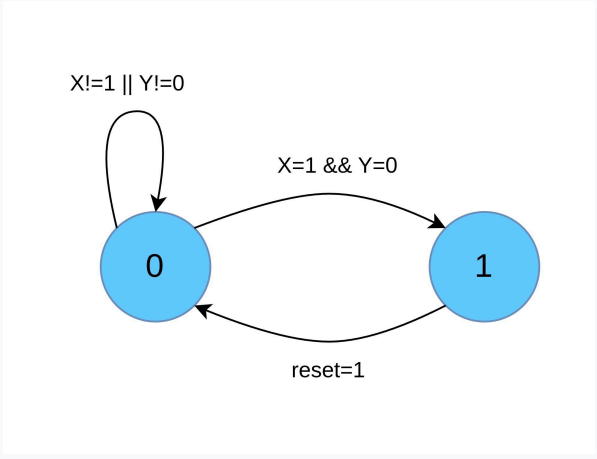
```
`timescale 10ns/1ns

module tb_toggle_circuit;

reg clk, rst, button;
wire y;

toggle_circuit dut (.clk(clk), .rst(rst), .button(button), .y(y));

initial begin
    $dumpvars;

    clk = 0;
    rst = 0;

    #20;

    rst = 1;

    #20;
    rst = 0;
    button = 0;

    #10;
    button = 1;

    #20;
    button = 0;

    #20;
    button = 0;

    #20;
    button = 1;

    #10;
    button = 0;
```

```
    #30;
    button = 1;

    #10;
    button = 1;

    #10;
    button = 0;

    #30;
    button = 0;

    #25;
    button = 1;

    #10;
    button = 0;

    #20;
    button = 1;
    $finish;

end


initial begin
    forever #5 clk = ~clk;
end
endmodule
```

➢ **Output:**



➢ Question 2: Design a State Machine with 2 States.



- **Verilog Code:**
  - **Using Blocking-Statements:**

    File 1: FSB_wbs.v

```
    module FSMs_wbs (input clk, rst, X, Y, output reg
state);


    localparam A = 1'b0;
    localparam B = 1'b1;



    always @(posedge clk or posedge rst)
```

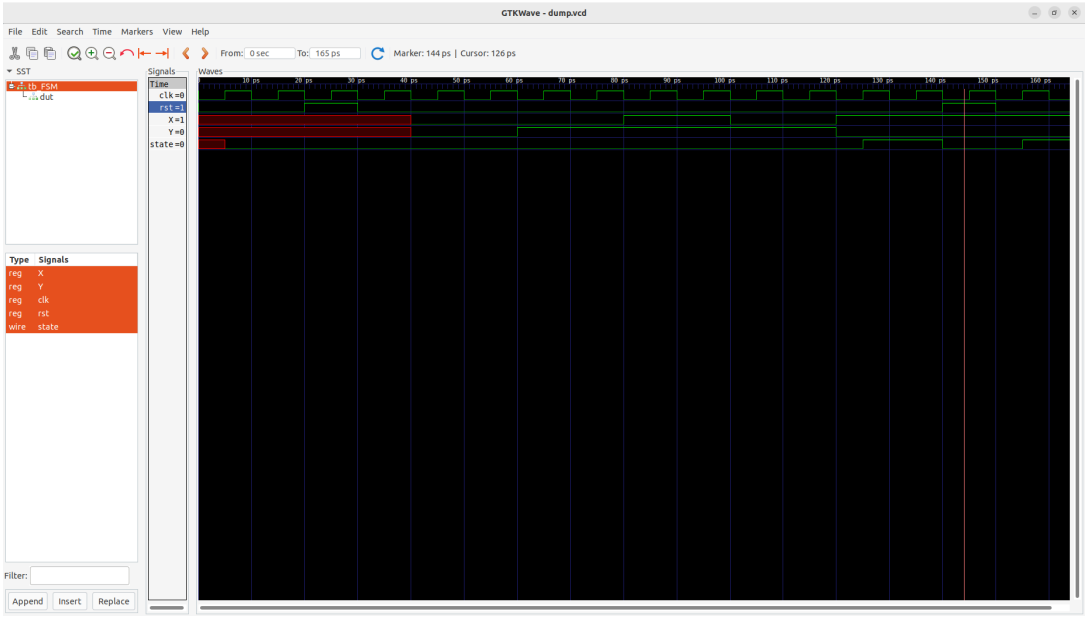
```
    begin
        if (rst)
        state = A;
        else
        begin
            case ({X,Y})
            2'b10 : state = B;
            default : state = A;
            endcase
        end
    end
    endmodule
```

File 2: tb_FSM.v

```
module tb_FSM;

    // Inputs
    reg clk;
    reg rst;
    reg X;
    reg Y;

    // Outputs
    wire state;

    // Instantiate the design under test (DUT)
    FSM_wbs dut (
        .clk(clk),
        .rst(rst),
        .X(X),
        .Y(Y),
        .state(state)
    );

    // Generate the clock and reset
    always #5 clk = ~clk;
    initial begin
        $dumpvars;

        clk = 0;
        rst = 0;
```

```
        #20;
        rst = 1;
        #10;
        rst = 0;
        #10;

        X = 0;
        Y = 0;
        #20;

        X = 0;
        Y = 1;
        #20;

        X = 1;
        Y = 1;
        #20;

        X = 0;
        Y = 1;
        #20;

        X = 1;
        Y = 0;
        #20;

        rst = 1;

        #10;
        rst = 0;

        X = 1;
        Y = 0;
        #20;

        $finish;
    end

endmodule
```

- **Output:**

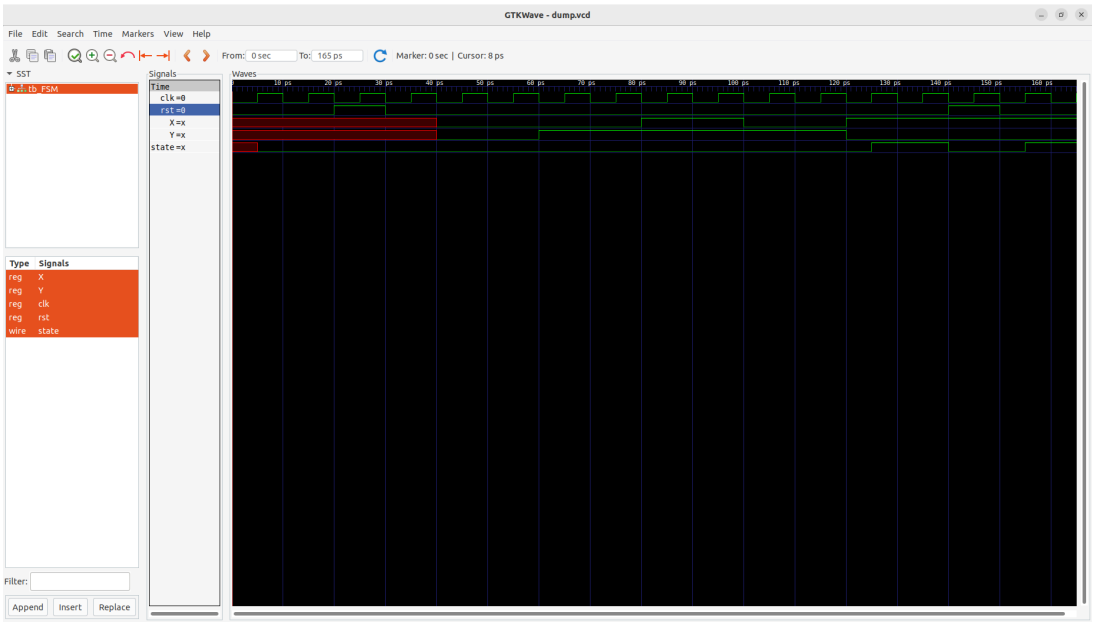


i. Using non-blocking Statements:

File 1: FSB_wobs.v

```
module FSM_wobs (input clk, rst, X, Y, output reg state);

    localparam A = 1'b0;
    localparam B = 1'b1;


    always @(posedge clk or posedge rst)
    begin
        if (rst)
        state <= A;
        else
        begin
            case ({X,Y})
            2'b10 : state <= B;
            default : state <= A;
            endcase
        end
    end
    endmodule
```

- **Output:**



Here are my findings:

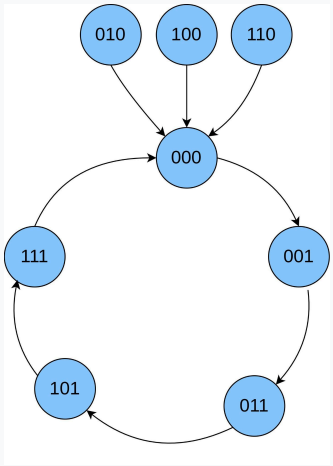
- Both blocking and non-blocking assignments yield equivalent behavior in this case.

- The output waveform remains the same, indicating consistent state transitions.
- Both assignment styles ensure proper sequencing within the clock cycle.

➢ **Question 3: Design a Counter using T-Flip Flop:**

Since we are counting 0 though 7, so we need to take three bits to accommodate highest count (111). Hence, total three flip-flops will be required to implement this circuit. Here's a state diagram:

1. **State Diagram:**



2. Excitation Table for T-Flip Flops:

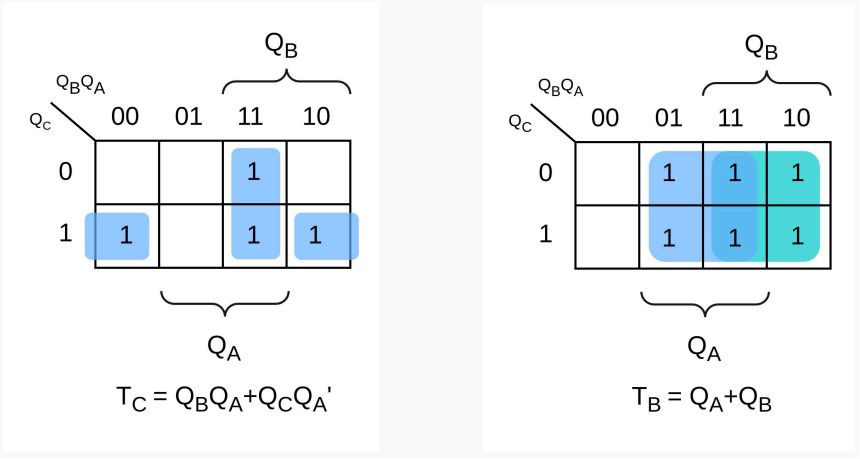
Q_n	Q_{n+1}	T
0	0	0
0	1	1
1	0	1
1	1	0

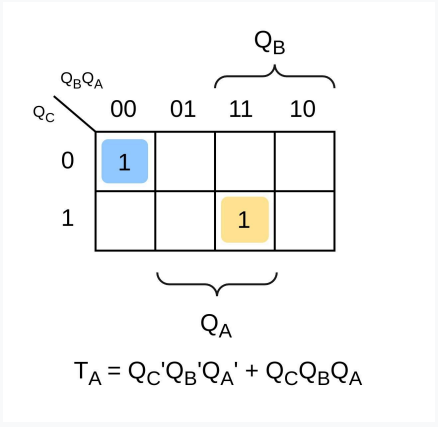
3. Excitation Table for Counter:

Present State			Next State			FF Inputs		
Q_C	Q_B	Q_A	Q_{C+1}	Q_{B+1}	Q_{A+1}	T_C	T_B	T_A
0	0	0	0	0	1	0	0	1
0	0	1	0	1	1	0	1	0
0	1	1	1	0	1	1	1	0
1	0	1	1	1	1	0	1	0
1	1	1	0	0	0	1	1	1
0	1	0	0	0	0	0	1	0
1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	1	1	0

In order to build a self-correcting counter, we need to consider unused states (greyed states in Table). I have set the next state for all of these unused states to be 000. Hence, whenever, the circuit enters an unknown state, the circuit will automatically moves to 000 state for self-correction.

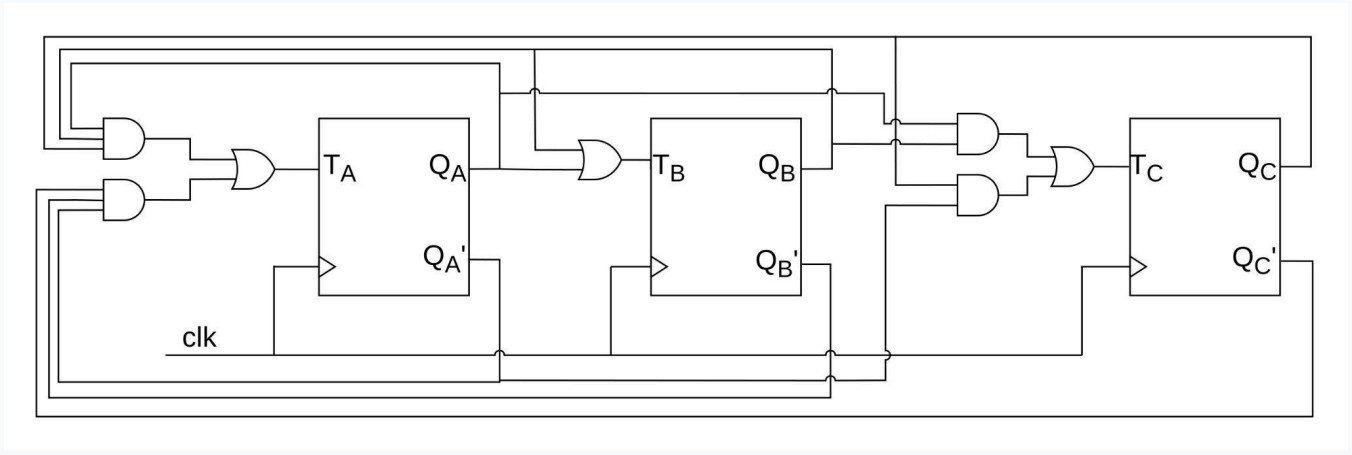
4. Using K-Maps:



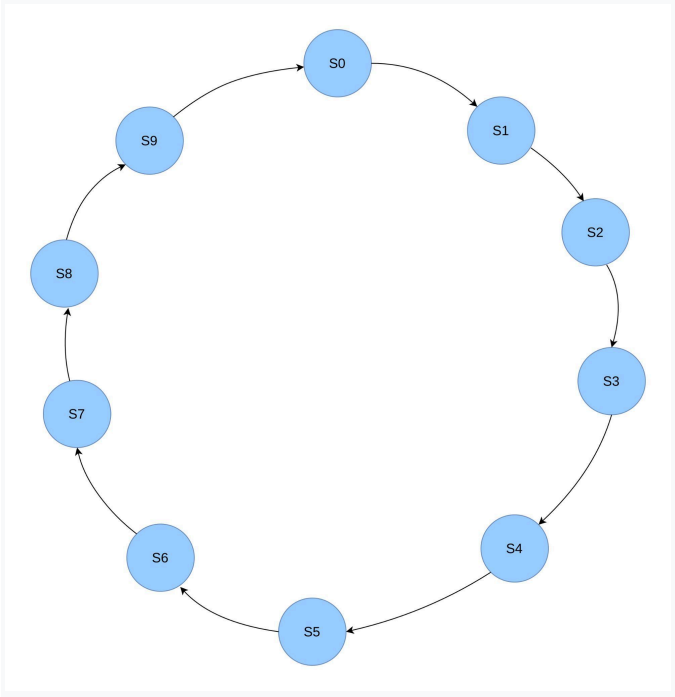


$T_A = Q_C'Q_B'Q_A' + Q_CQ_BQ_A$

5. Circuit Diagram:



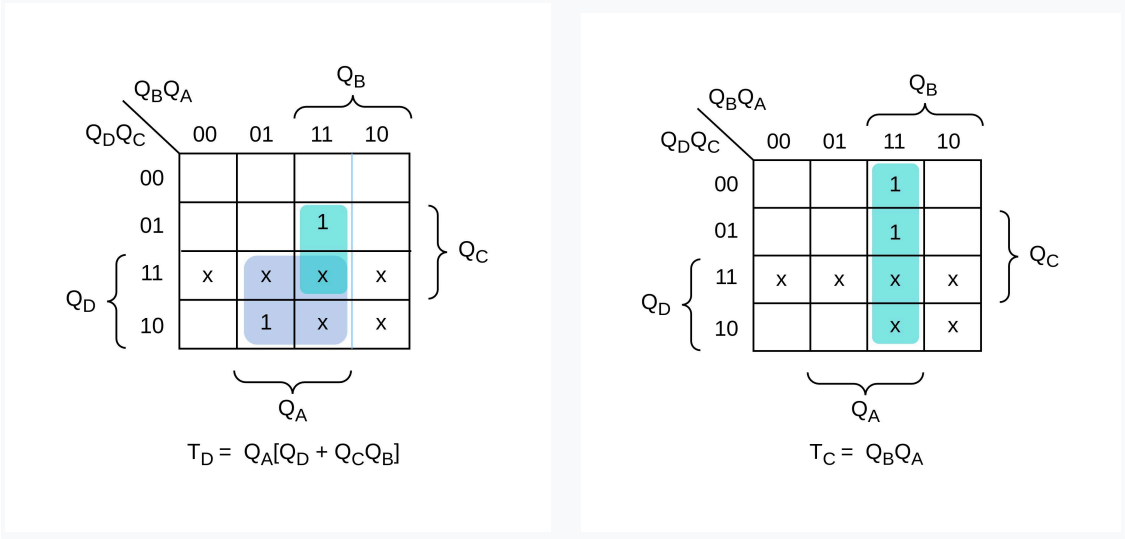
➢ **Question 4: Design a modulo-10 frequency divider:**

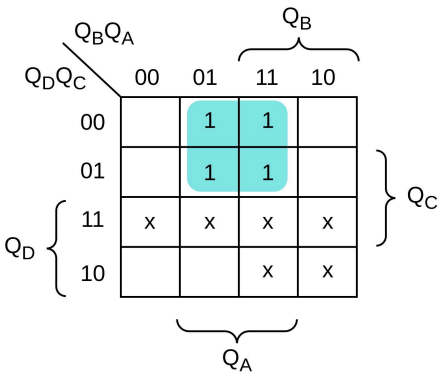


1. Excitation Table for Counter:

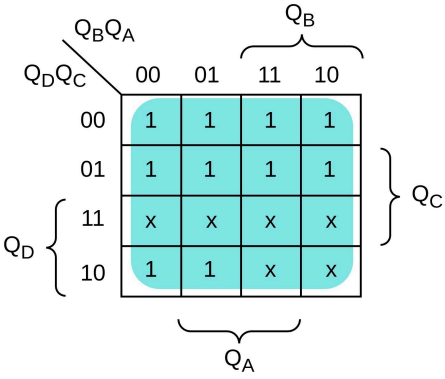
Present State				Next State				FF Inputs			
Q_D	Q_C	Q_B	Q_A	Q_{D+1}	Q_{C+1}	Q_{B+1}	Q_{A+1}	T_D	T_C	T_B	T_A
0	0	0	0	0	0	0	1	0	0	0	1
0	0	0	1	0	0	1	0	0	0	1	1
0	0	1	0	0	0	1	1	0	0	0	1
0	0	1	1	0	1	0	0	0	1	1	1
0	1	0	0	0	1	0	1	0	0	0	1
0	1	0	1	0	1	1	0	0	0	1	1
0	1	1	0	0	1	1	1	0	0	0	1
0	1	1	1	1	0	0	0	1	1	1	1
1	0	0	0	1	0	0	1	0	0	0	1
1	0	0	1	0	0	0	0	1	0	0	1

2. Using K-Maps:



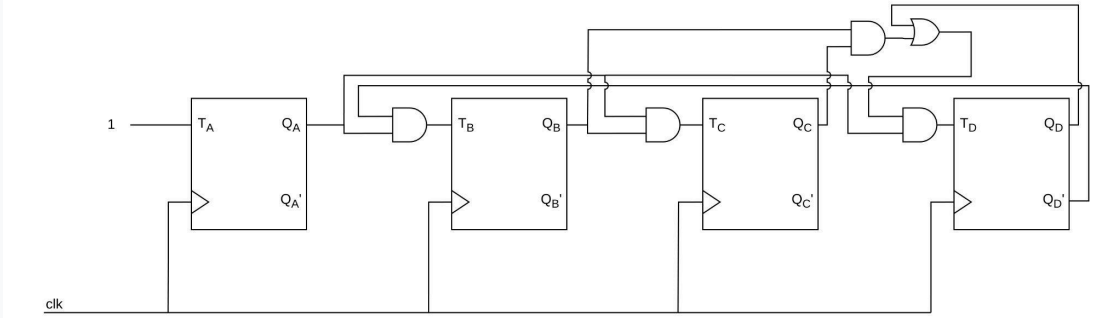


$T_B = Q_AQ_D'$



$T_A = 1$

3. Circuit Diagram:



4. Verilog Code:

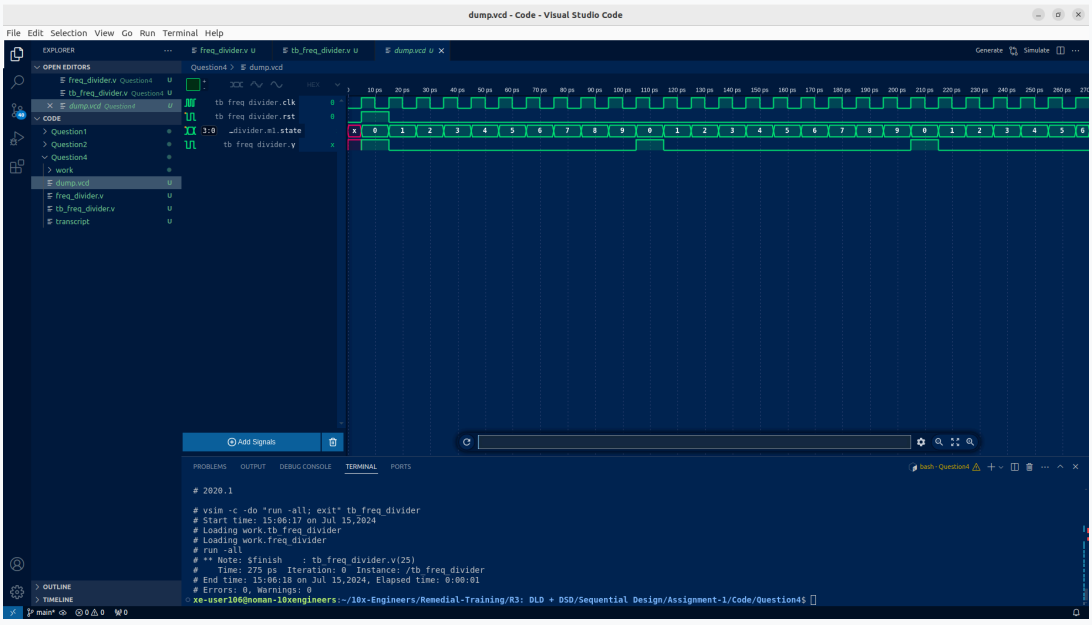
```
    module freq_divider (input clk, rst, output reg y);

    reg [3:0] state;
    localparam S0 = 4'd0;
    localparam S1 = 4'd1;
    localparam S2 = 4'd2;
    localparam S3 = 4'd3;
    localparam S4 = 4'd4;
    localparam S5 = 4'd5;
    localparam S6 = 4'd6;
    localparam S7 = 4'd7;
    localparam S8 = 4'd8;
    localparam S9 = 4'd9;

    always @(posedge clk or posedge rst)
    begin
```

```
    if (rst)
    state <= S0;
    else
    state <= state;
    end

    always @(posedge clk or posedge rst)
    begin
    case (state)
    S0 : state <= S1;
    S1 : state <= S2;
    S2 : state <= S3;
    S3 : state <= S4;
    S4 : state <= S5;
    S5 : state <= S6;
    S6 : state <= S7;
    S7 : state <= S8;
    S8 : state <= S9;
    S9 : state <= S0;
    endcase
    end

    always @(state)
    begin
    case (state)
    S0 : y <= 1;
    S1 : y <= 0;
    S2 : y <= 0;
    S3 : y <= 0;
    S4 : y <= 0;
    S5 : y <= 0;
    S6 : y <= 0;
    S7 : y <= 0;
    S8 : y <= 0;
    S9 : y <= 0;
    endcase
    end
    endmodule
```

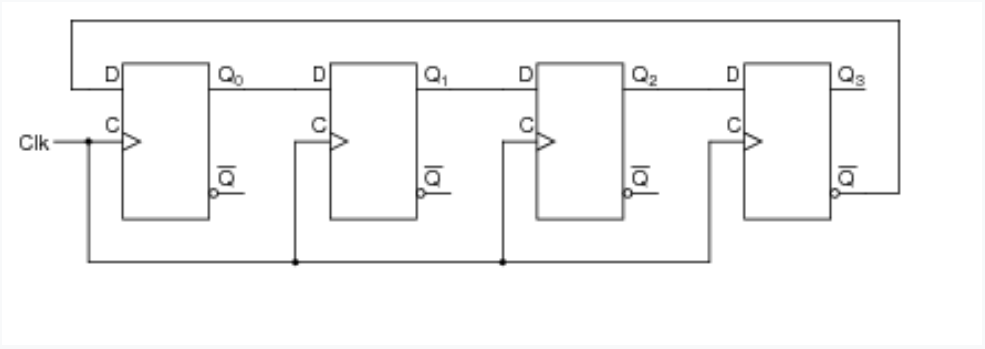
5. **Testbench:**

```
module tb_freq_divider;
reg clk, rst;
wire y;

freq_divider m1 (.clk(clk), .rst(rst), .y(y));

always #5 clk = ~clk;

initial begin
$dumpvars;

clk = 0;
rst = 0;
#5;

rst = 1;
#10;

rst = 0;
#10;

#250;
$finish;
end
endmodule
```

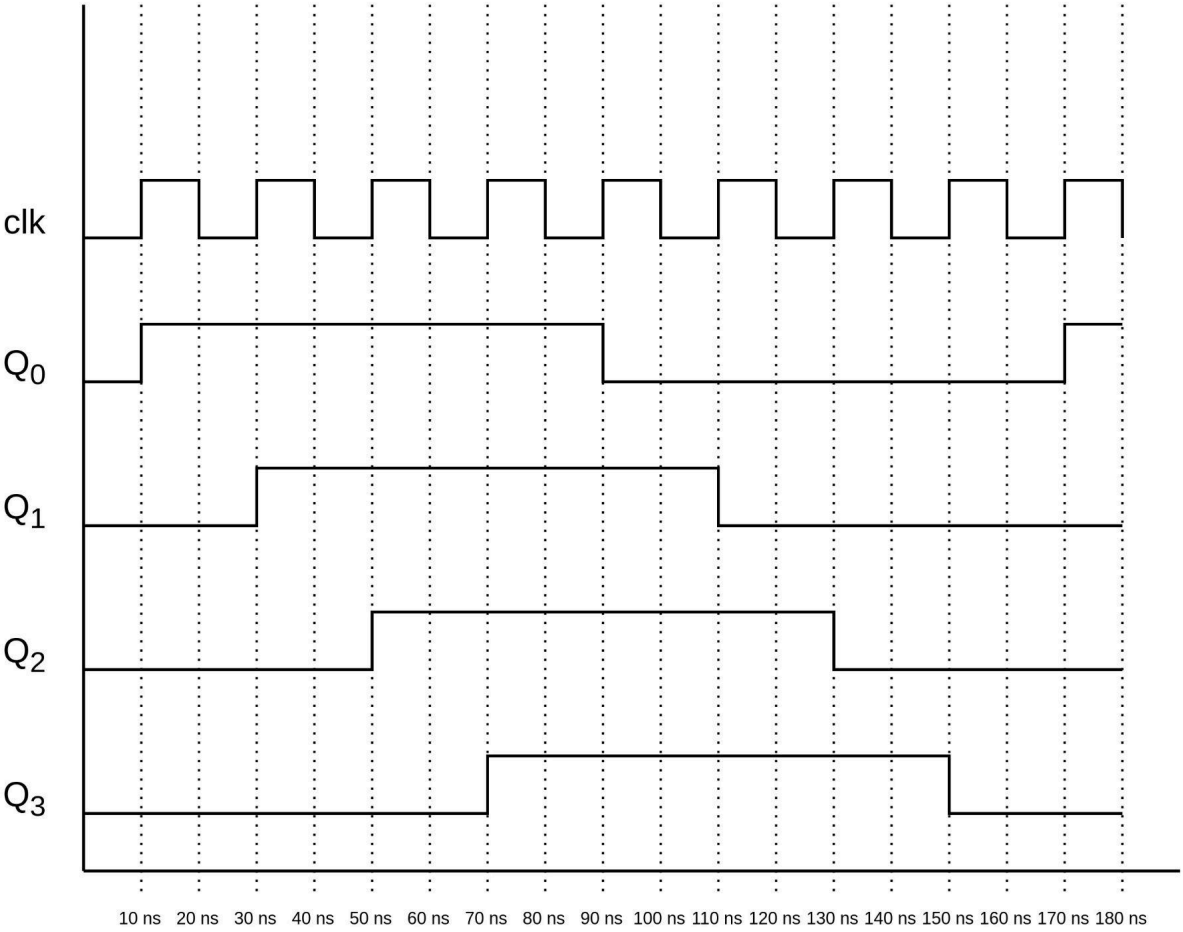
6. Output:



➢ **Question 5: Waveform of Johnson Counter:**



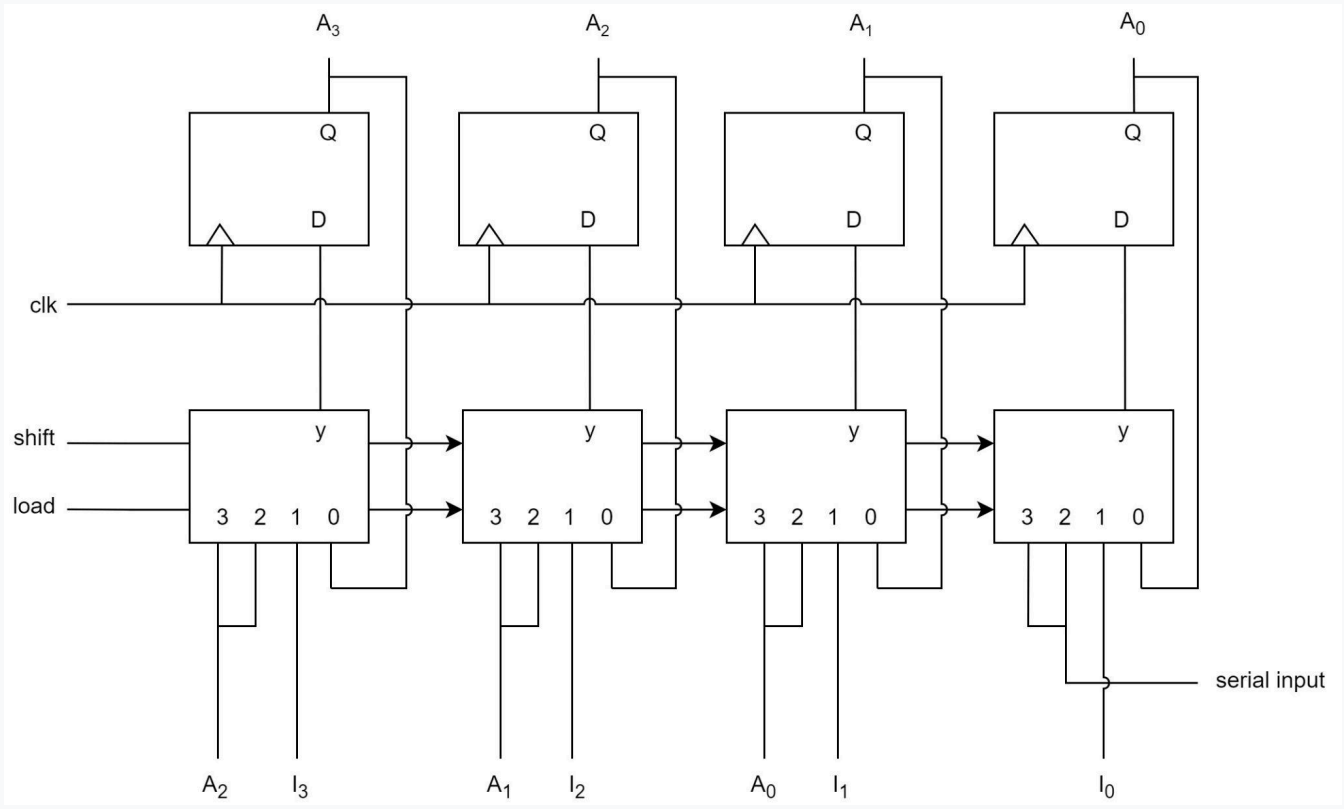
1. **Waveform:**



➢ Question 6: Shift Register:

Operation Table		
Shift	Load	Register Operation
0	0	No Change
0	1	Load Parallel Data
1	X	Shift register (Assumed left shift)

Using Table, the circuit diagram will be as follows:



1. Verilog Code:

```
    module shift_register (input clk, rst, load, shift, input
[3:0] data_in, output reg [3:0] A);

    always @(posedge clk or posedge rst)
    if (rst)
        A <= 4'd0;
    else
    begin
```

```
        case ({shift, load})
        2'b00 : A <= A;
        2'b01 : A <= data_in;
        default : A <= A << 1;
        endcase
    end
    endmodule
```

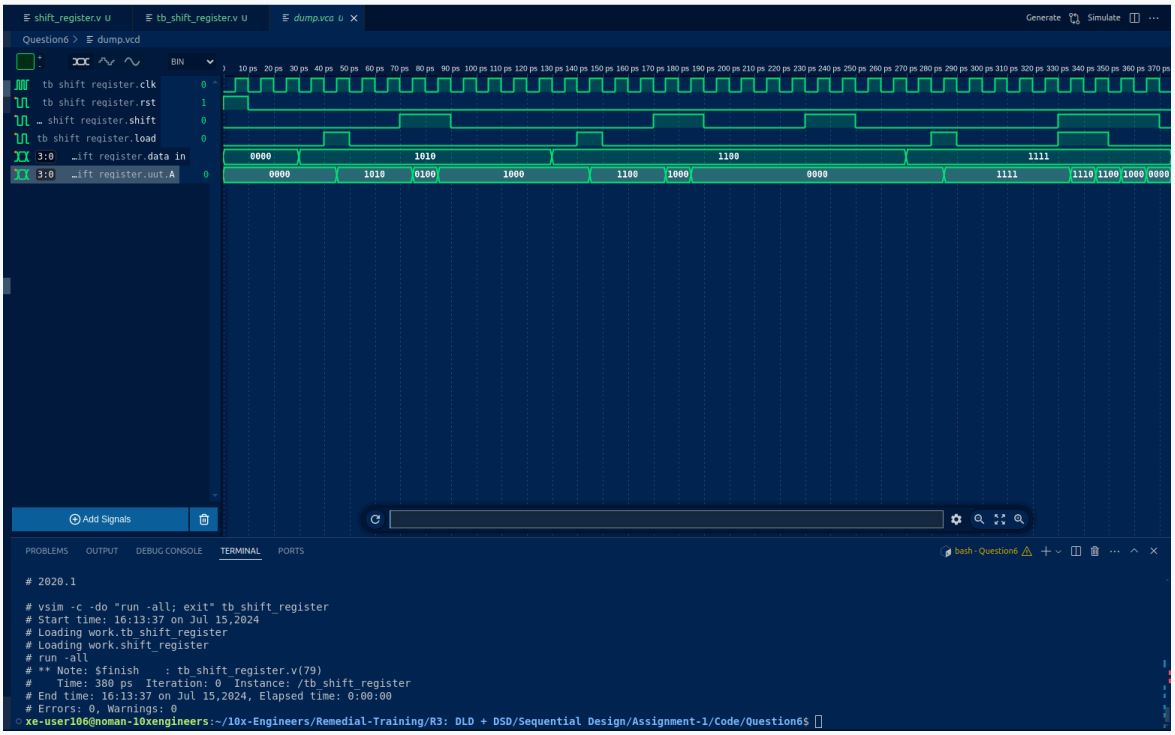
2. **Testbench:**

```
    //Author: Noman Rafiq
//Dated: July 15, 2024

module tb_shift_register;

    // Inputs
    reg [3:0] data_in;
    reg shift;
    reg load;
    reg clk;
    reg rst;

    // Output
    wire [3:0] A;

    // Instantiate the shift register
    shift_register uut (
        .data_in(data_in),
        .shift(shift),
        .load(load),
        .clk(clk),
        .rst(rst),
        .A(A)
    );

    // Clock generation
    always #5 clk = ~clk;

    // Initial values
    initial begin

        $dumpvars;
```

```
        clk = 0;
        rst = 1;
        data_in = 4'b0000;
        shift = 0;
        load = 0;

        // Reset
        #10 rst = 0;

        // Test case 1: Load parallel data
        #20 data_in = 4'b1010;
        #10 load = 1;
        #10 load = 0;

        // Test case 2: Shift register
        #20 shift = 1;
        #20 shift = 0;

        // Test case 3: No change
        #20;

        // Test case 4: Load new data and then shift
        #20 data_in = 4'b1100;
        #10 load = 1;
        #10 load = 0;
        #20 shift = 1;
        #20 shift = 0;

        // Test case 5: Shift without loading new data
        #40 shift = 1;
        #20 shift = 0;

        // Test case 6: Shifting while shift and load are set to 1
        #20 data_in = 4'b1111;
        #10 load = 1;
        #10 load = 0;

        #40 shift = 1;
            load = 1;
        #20 shift = 1;
            load = 0;
```

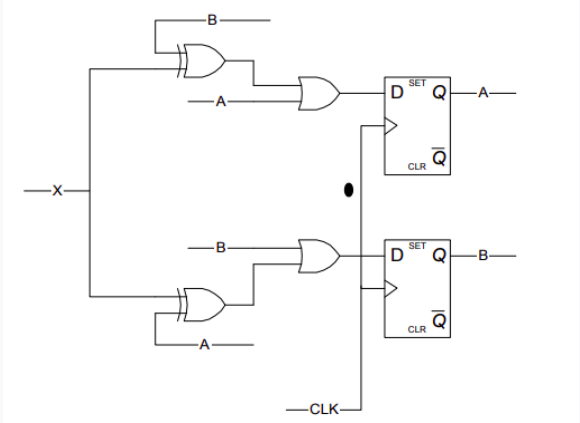
```
        #20 shift = 0;
            load = 0;
        // End simulation
        #10 $finish;
    end

endmodule
```

3. Output:



➢ Question 7: Derive the state table and state diagram of the following circuits:

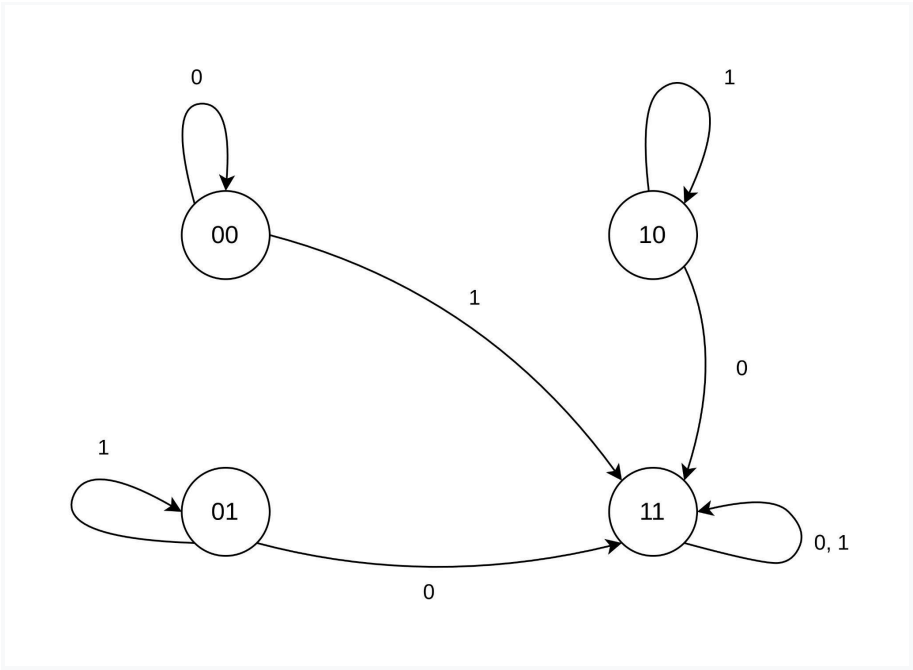
a. Diagram:



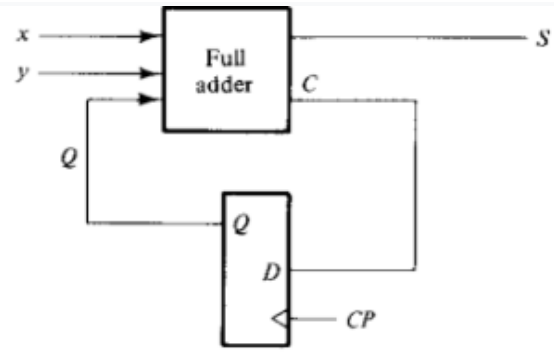
1. State Table:

Present State		Input	Next State	
A	B	X	A+	B+
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	1
1	0	1	1	0
1	1	0	1	1
1	1	1	1	1

2. State Diagram:



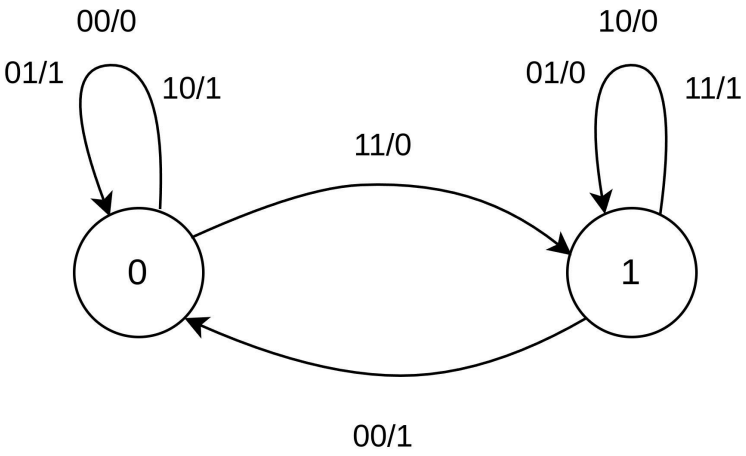
b. Diagram:



1. State Table:

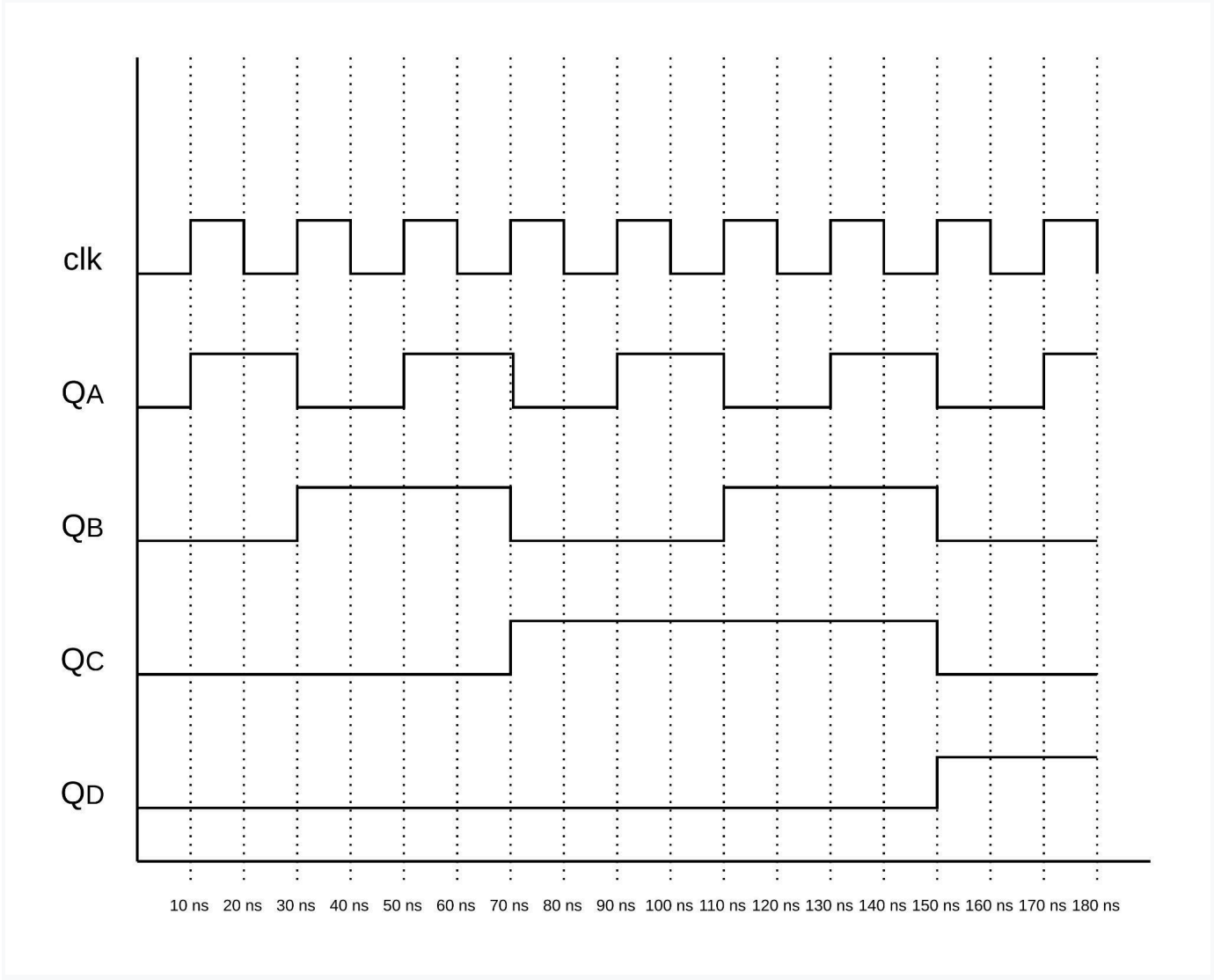
Present State	Inputs		Next State	Output	Carry
Qn	x	y	Qn+1	S	C
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	0	1	0
0	1	1	1	0	1
1	0	0	0	1	0
1	0	1	1	0	1
1	1	0	1	0	1
1	1	1	1	1	1

2. State Diagram:

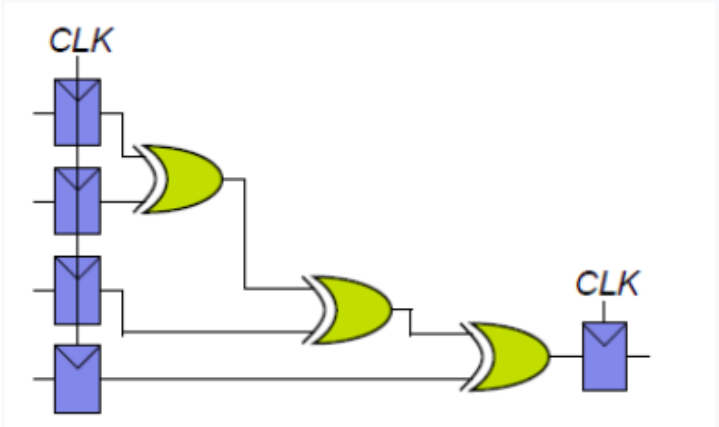


➢ **Question 8: Timing Diagram:**

It is a synchronous counter with an upwards direction .i.e. Synchronous up counter. Here's the timing diagram:



➢ **Question 9: Clock:**



A. If there is no clock skew, what is the maximum operating frequency of this circuit?

To find the maximum operating frequency, we need to find minimum clock duration and the Max Operating frequency will be inverse of that.

$$TC \geq Tpcq + Tpd + Tsetup$$
$$TC \geq 70 + (100*3) + 60 = 430 \text{ ps}$$
$$\text{Max Frequency} = 1/TC = 2.33 \text{ GHz}$$

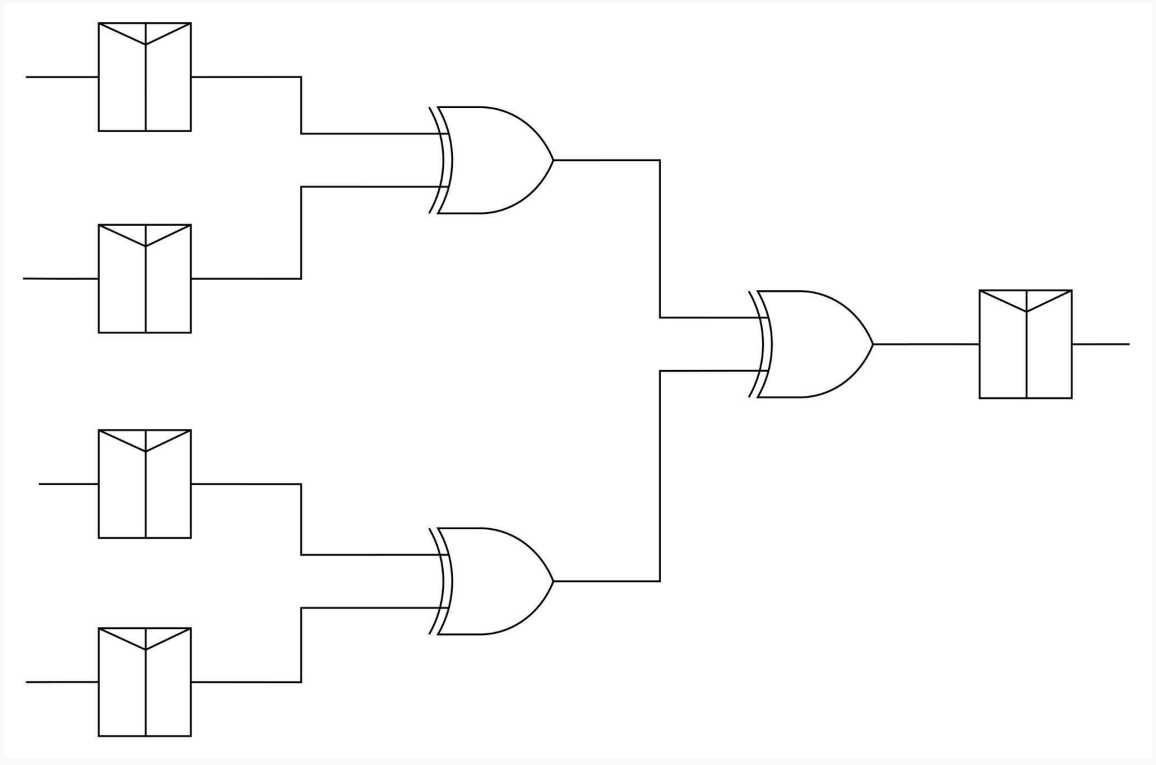
B. How much clock skew can the circuit tolerate before it might experience a hold time violation?

$$Tccq + Tcd \geq Thold + Tskew$$
$$50 + 55 \geq 20 + Tskew$$
$$Tskew \leq 85 \text{ ps}$$

C. Redesign the circuit so that it can be operated at a 3 GHz frequency?

Minimum clock duration for 3 GHz will be:

$$T_C = 1/ (3 * 10^6) = 333.3 \text{ ns}$$



D. How much clock skew can your circuit tolerate before it might experience a hold time violation?

```
Tccq + 2*Tcd ≥ Thold + Tskew
50 + 2*55 ≥ 20 + Tskew
Tskew ≤ 160 - 20
Tskew ≤ 140 ps
```