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

# Assignment 1

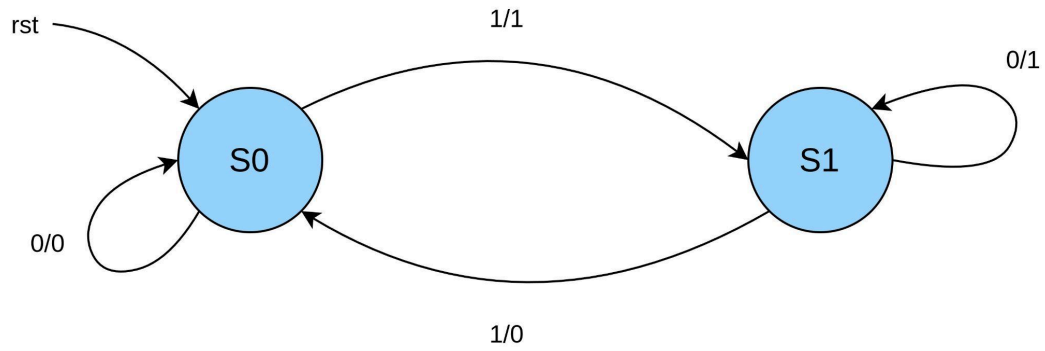
## Sequential Circuits

---

➢ **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:**



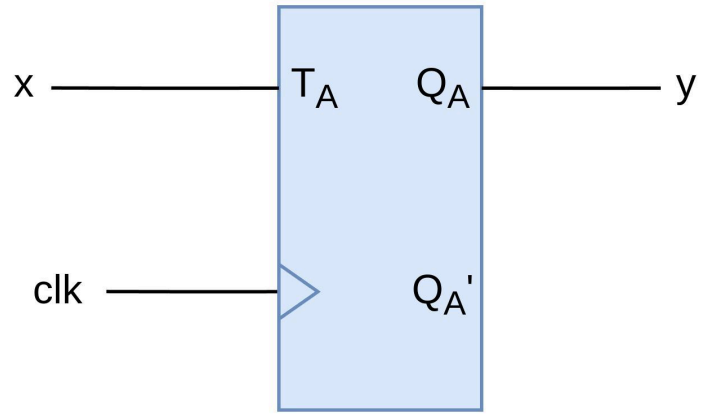

Here's the State-Table:

| Present State | Input | Next State | Output | FF Inputs |
|---|---|---|---|---|
| $Q_t$ | x (button) | $Q_{t+1}$ | y | T |
| 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 1 | 1 |
| 1 | 0 | 1 | 1 | 0 |
| 1 | 1 | 0 | 0 | 1 |

From the table, it is evident that:

$$y = Q_{(t+1)}$$
$$T = x$$

- **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;
 end
 endmodule
```

  - **b. toggle_circuit.v:**

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

wire Q;

t_ff m0(.t(button), .clk(clk), .rst(rst), .q(Q));

assign y = Q;
```

```
endmodule
```

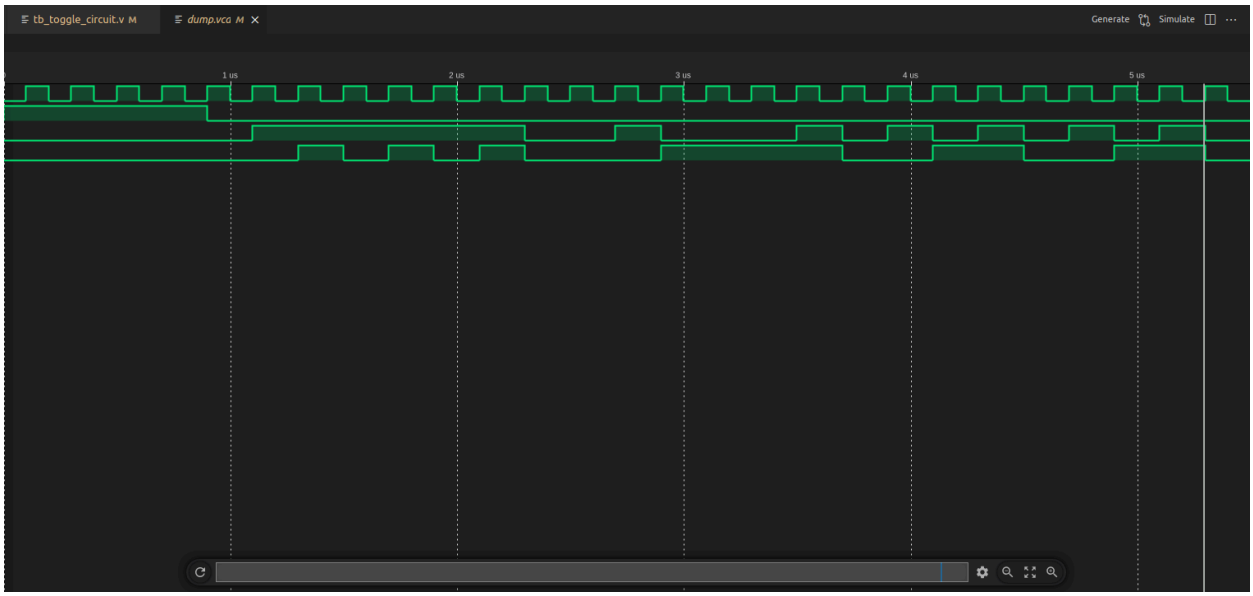
➢ **Testbench:**

```
`timescale 10ns/1ns

module tb_toggle_circuit;

reg button;
reg clk;
reg rst;
wire y;

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

initial begin
    // Initialize inputs
    button = 0;
    clk = 0;
    rst = 1;
    $dumpvars;

    repeat (5) @(posedge clk);

    // Release reset
    rst = 0;

    // Check initial output
    $display("Checking initial output...");

    if (y != 0) begin
    $display("Error: Initial output is incorrect");
    $finish;
    end

    @(posedge clk)
```

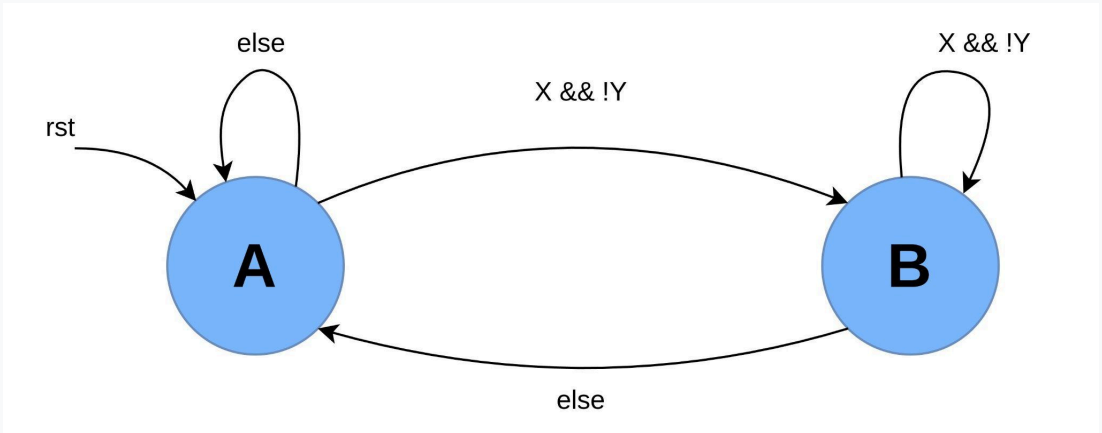
```
    // Test button toggle
    $display("Toggling button and checking output...");
    button = 1;
    @(posedge clk);
    if (y != 0) begin
        $display("Error: Output did not toggle on");
        $finish;
    end

    repeat (5) @(posedge clk)
    // Test button toggle again
    $display("Toggling button again and checking output...");
    button = 0;
    repeat(2) @(posedge clk);
    if (y != 0) begin
        $display("Error: Output did not toggle off");
        $finish;
    end

    //Remember State
    $display("Toggling button again and checking output...");
    button = 1;
    @(posedge clk)
    button = 0;
    repeat(2) @(posedge clk);
    if (y != 1) begin
        $display("Error: Output was not remembered");
        $finish;
    end

    // Repeat toggle test 10 times
    $display("Repeating toggle test 10 times...");

    repeat (10) button = @(posedge clk) ~button;

    repeat (10) begin
        @(posedge clk);
        if (y != button) begin
            $display("Error: Output did not toggle correctly");
            $finish;
        end
        else
```

```
            $display("Tests Passed Successfully");
            $finish;

    end
end
    // Clock generator
    always #10 clk = ~clk;

endmodule
```

➢ **Output:**




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




■ **Verilog Code:**

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

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

    reg next_state;

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

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

        A : begin if (X && ~Y) begin
            next_state = B;
        end
        else next_state = A;
        end

        B : begin if (X && ~Y) begin
            next_state = B;
        end
        else next_state = A;
        end

        default : next_state = A;
        endcase
    end
    endmodule
```

**File 2: tb_FSM.v**

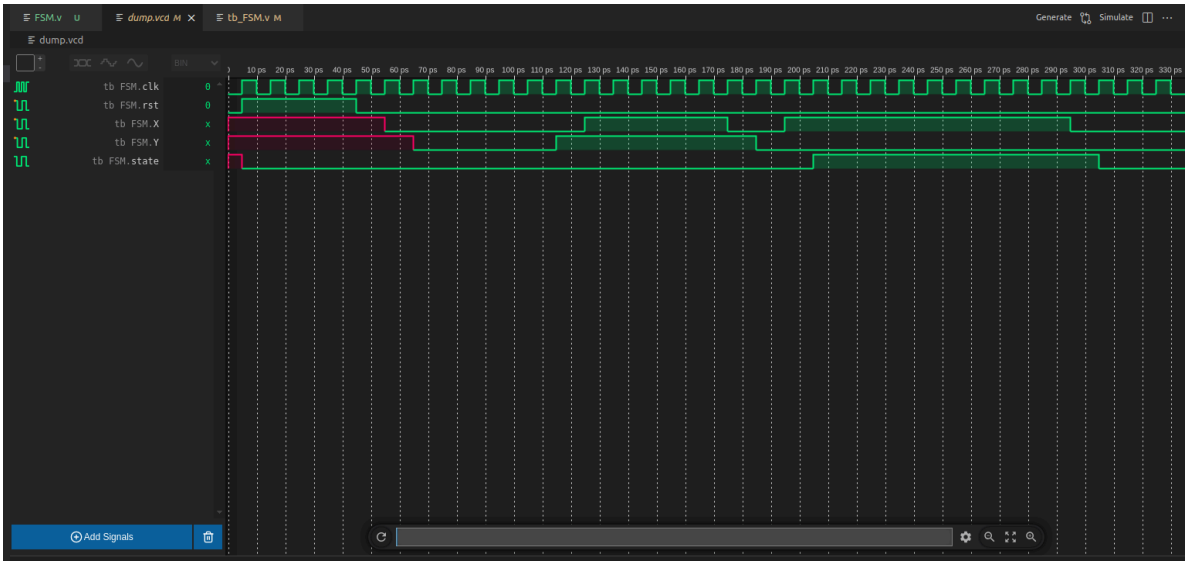
```
module tb_FSM;
```

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

    wire state;

    FSM 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;

        repeat (4) @(posedge clk)
        rst = 1;

        @(posedge clk)
        rst = 0;

        repeat(2) @(posedge clk)
        X = 0;
        Y = 0;
        if (state != 0)
            $display("Error: Initial state hasn't been
reset");

        repeat(5) @(posedge clk)
        X = 0;
        Y = 1;

        if (state != 0) begin
            $display("Error");
```

```
            $finish;
        end

        repeat(5) @(posedge clk)
        X = 1;
        Y = 1;

        if (state != 0) begin
            $display("Error");
            $finish;
        end

        repeat (2) @(posedge clk)
        X = 0;
        Y = 0;

        repeat(5) @(posedge clk)
        X = 1;
        Y = 0;

        if (state != 1) begin
            $display("Error: The FSM failed to transition
from A to B");
            $finish;
        end

        repeat(5) @(posedge clk)

        //No change in input, should stay in state B
        if (state != 1) begin
            $display("Error: The FSM failed to stay in B");
            $finish;
        end

        repeat(5) @(posedge clk)
        X = 0;
        Y = 1;

        if (state != 0) begin
            $display("Error: The FSM failed to transition
from B to A");
            $finish;
```

```
        end

        $display("All Test Cases Passed Successfully");
        $finish;
    end

endmodule
```

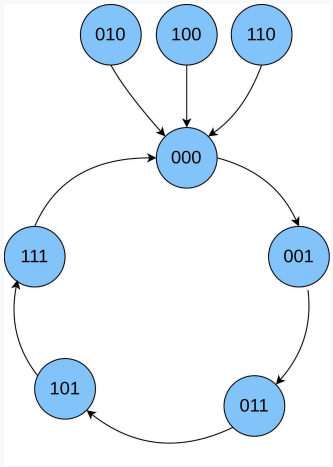
- **Output:**




➢ **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:

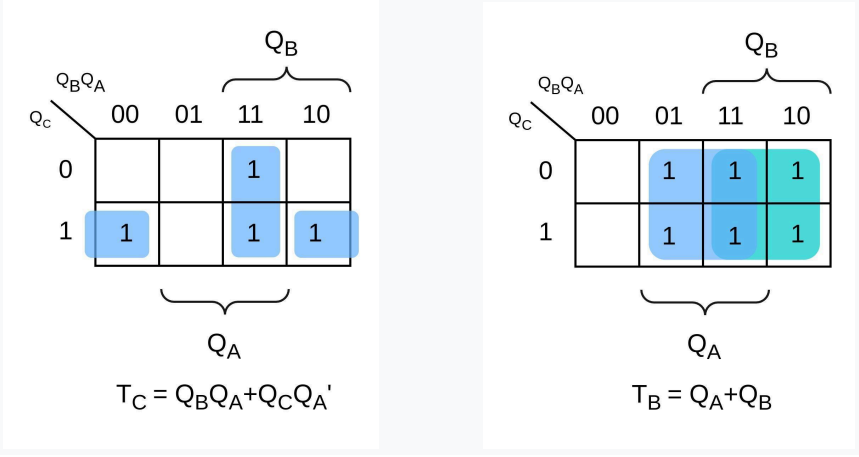
| $Q_n$ | $Q_{n+1}$ | T |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

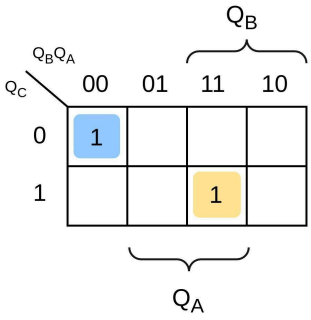
### 3. Excitation Table for Counter:

| 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.

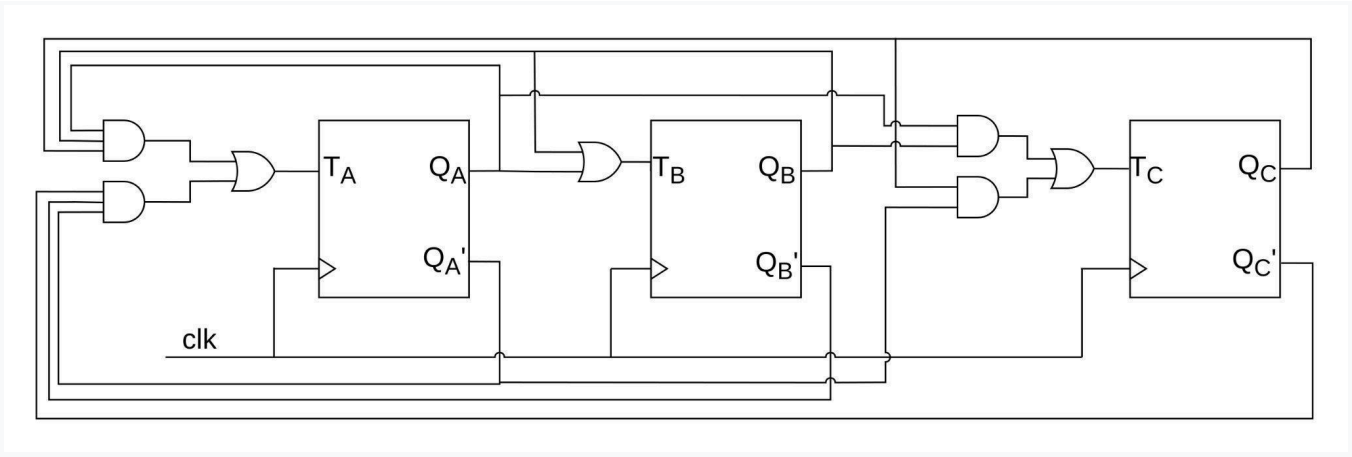
### 4. Using K-Maps:



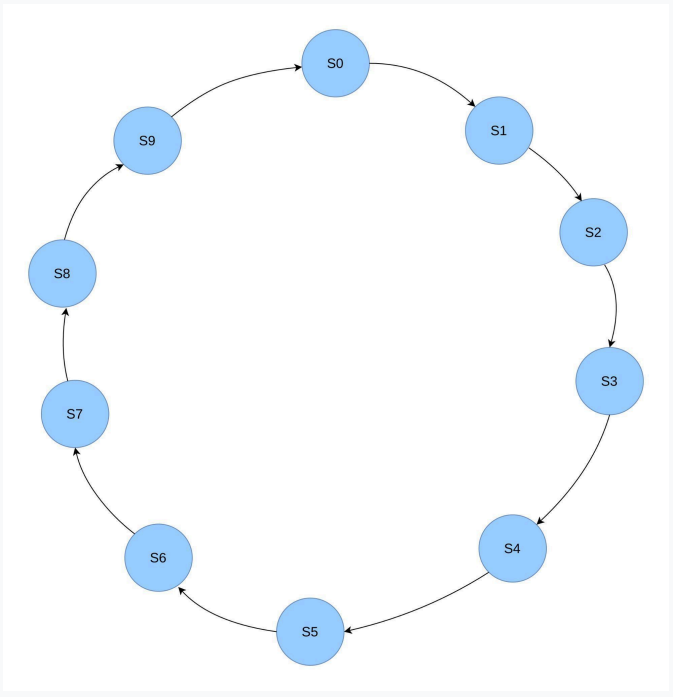



$$T_A = Q_C'Q_B'Q_A' + Q_CQ_BQ_A$$

**5. Circuit Diagram:**



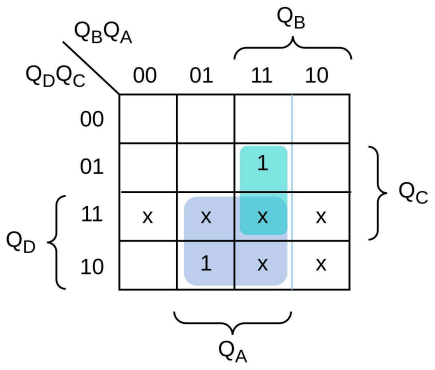

➢ **Question 4: Design a modulo-10 frequency divider:**



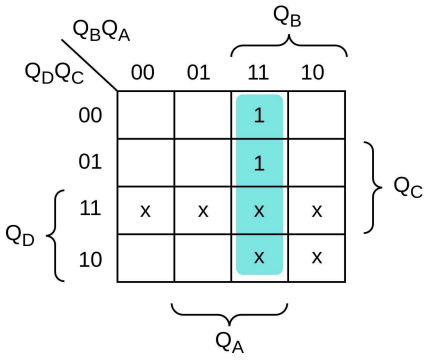
## 1. Excitation Table for Counter:

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

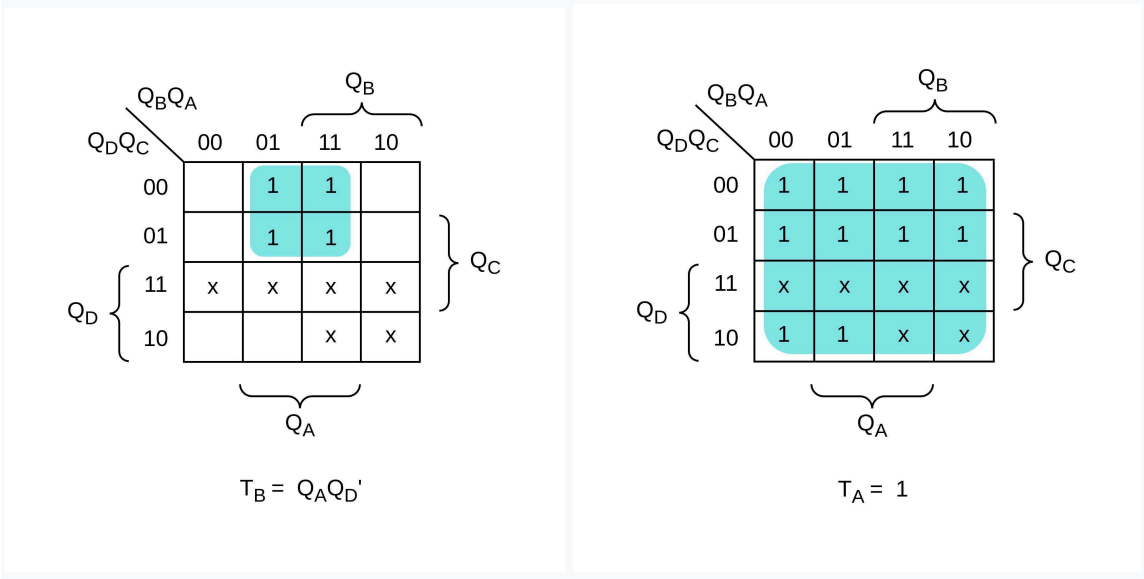
## 2. Using K-Maps:

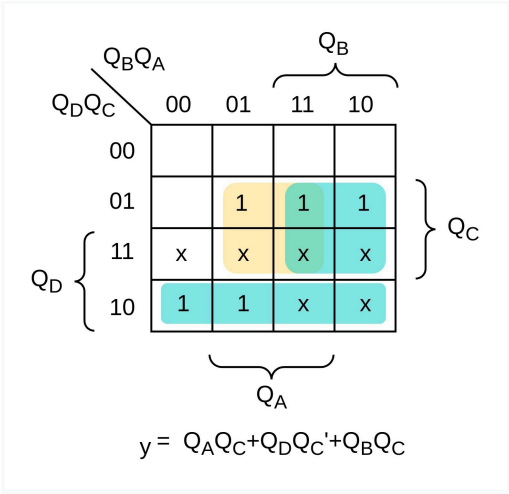



$T_D = Q_A[Q_D + Q_CQ_B]$

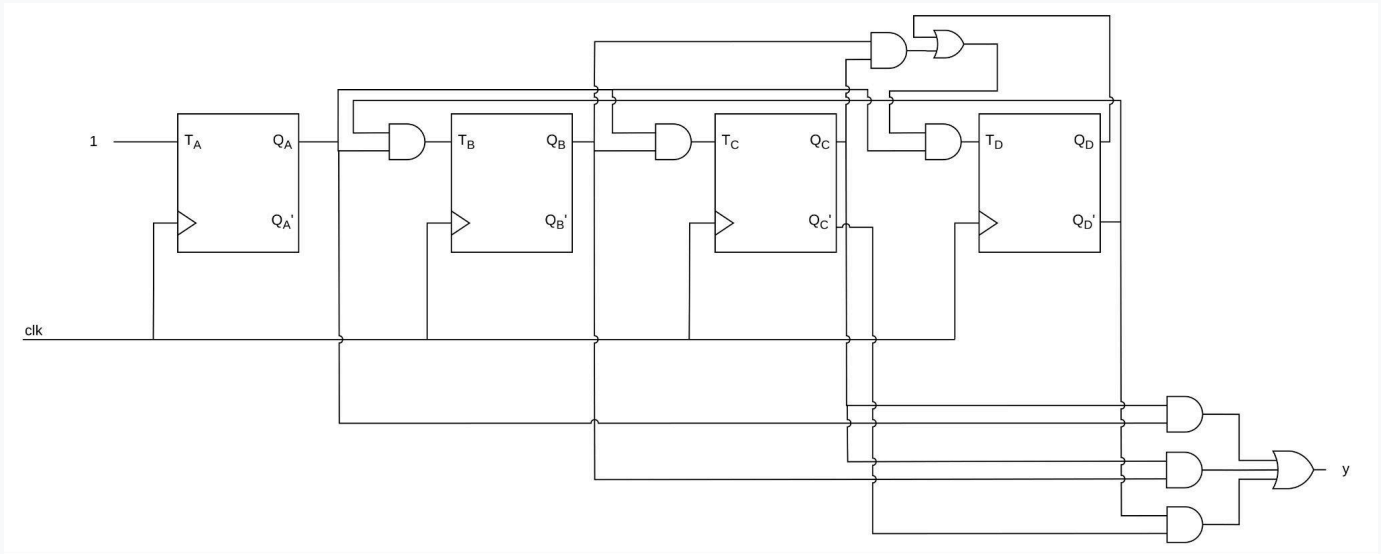



$T_C = Q_BQ_A$




### 3. Circuit Diagram:




### 4. Verilog Code:

```
module t_ff (input T, clk, rst, output reg Q);

always @(posedge clk or posedge rst) begin
    if (rst)
    Q <= 0;
    else if (T)
    Q <= ~Q;
end
endmodule
```

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

    wire Qa, Qb, Qc, Qd;
    wire Tb, Tc, Td;

    t_ff A(.T(1'b1), .clk(clk), .rst(rst), .Q(Qa));
    t_ff B(.T(Tb), .clk(clk), .rst(rst), .Q(Qb));
    t_ff C(.T(Tc), .clk(clk), .rst(rst), .Q(Qc));
    t_ff D(.T(Td), .clk(clk), .rst(rst), .Q(Qd));

    assign Tb = Qa & ~Qd;
    assign Tc = Qb & Qa;
    assign Td = Qa & (Qd | (Qc & Qb));

    assign y = (Qd & ~Qc) | (Qa & Qc) | (Qb & Qc);


    endmodule
```

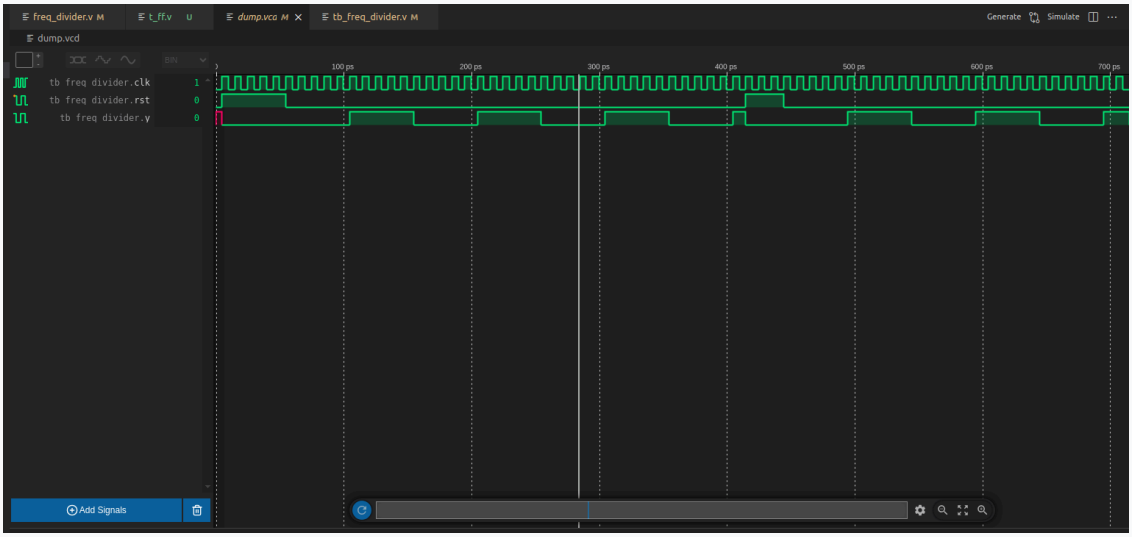
5. **Testbench:**

```
//`timescale 1ns/1ns

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

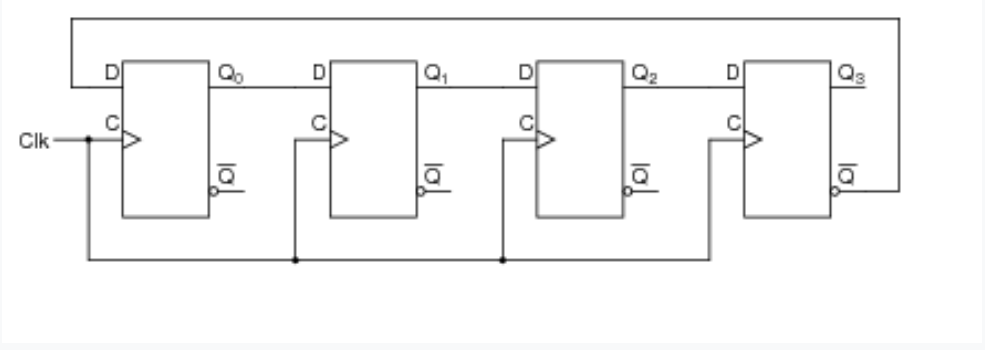
```
    freq_divider m1 (
        .clk(clk),
        .rst(rst),
        .y(y)
    );

    initial begin
        $dumpvars;
        clk = 0;
        rst = 0;
    end

    always #5 clk = ~clk; // clock generator

    initial begin

        rst = 0;
        @(posedge clk);

        rst = 1;
        repeat (5) @(posedge clk);

        if (y != 0) begin
            $display("Initial Output is Incorrect");
        end
        rst = 0;

        // Test sequence
        repeat (6) @(posedge clk);

        if (y != 1) begin
            $display("Failed to divide the frequency by 10 after
reset");
        end
        repeat (30) @(posedge clk);

        rst = 1;
        repeat (3) @(posedge clk);
        if (y != 0) begin
            $display("Error: Output is Incorrect");
        end
```

```
        rst = 0;

        repeat (5.5) @(posedge clk);
        //@(negedge clk);
        if (y != 1) begin
            $display("Failed to divide the frequency by 10 after
reset");
            $finish;
        end
        repeat (22) @(posedge clk);

        $display("All test cases Passed");

        $finish;
    end
endmodule
```

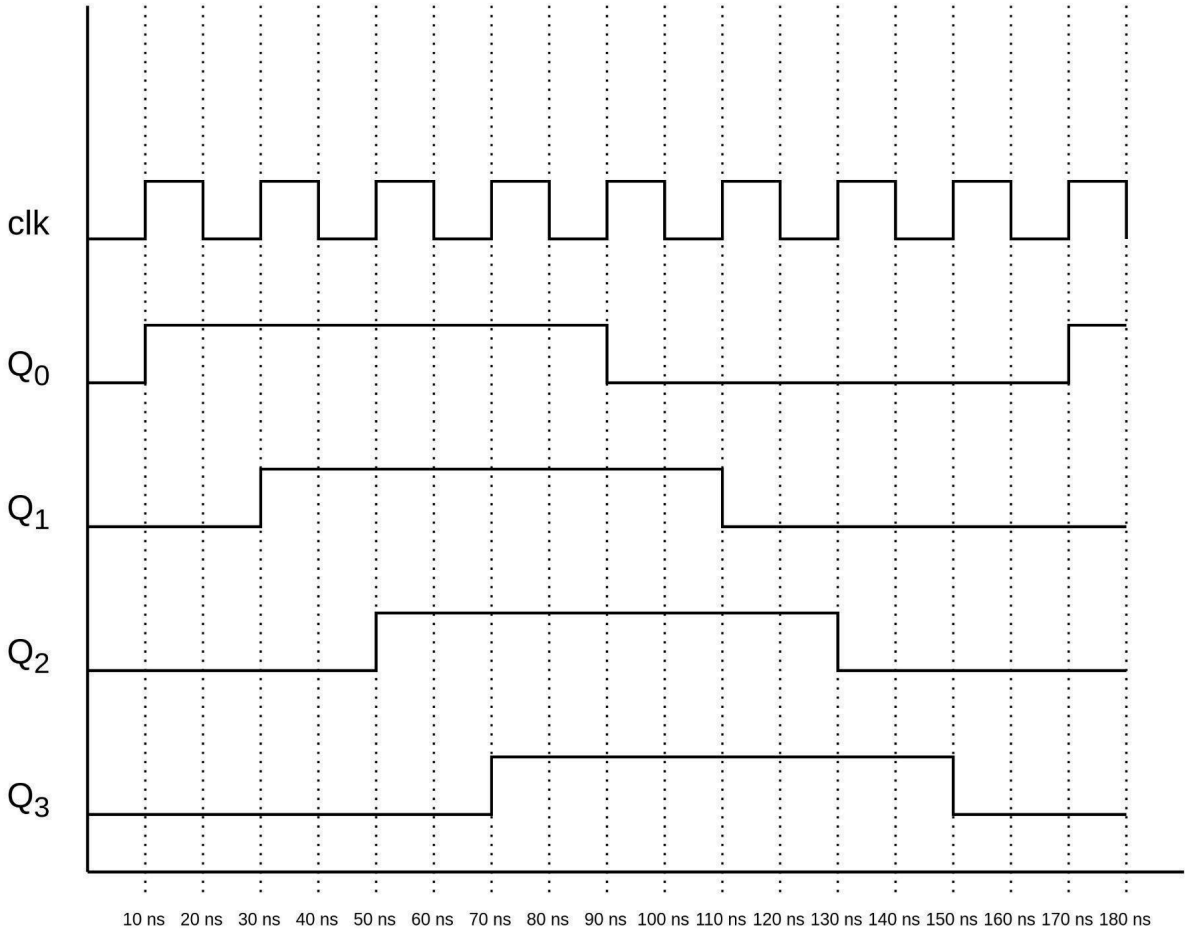
### 6. Output:




## ➢ Question 5: Waveform of Johnson Counter:



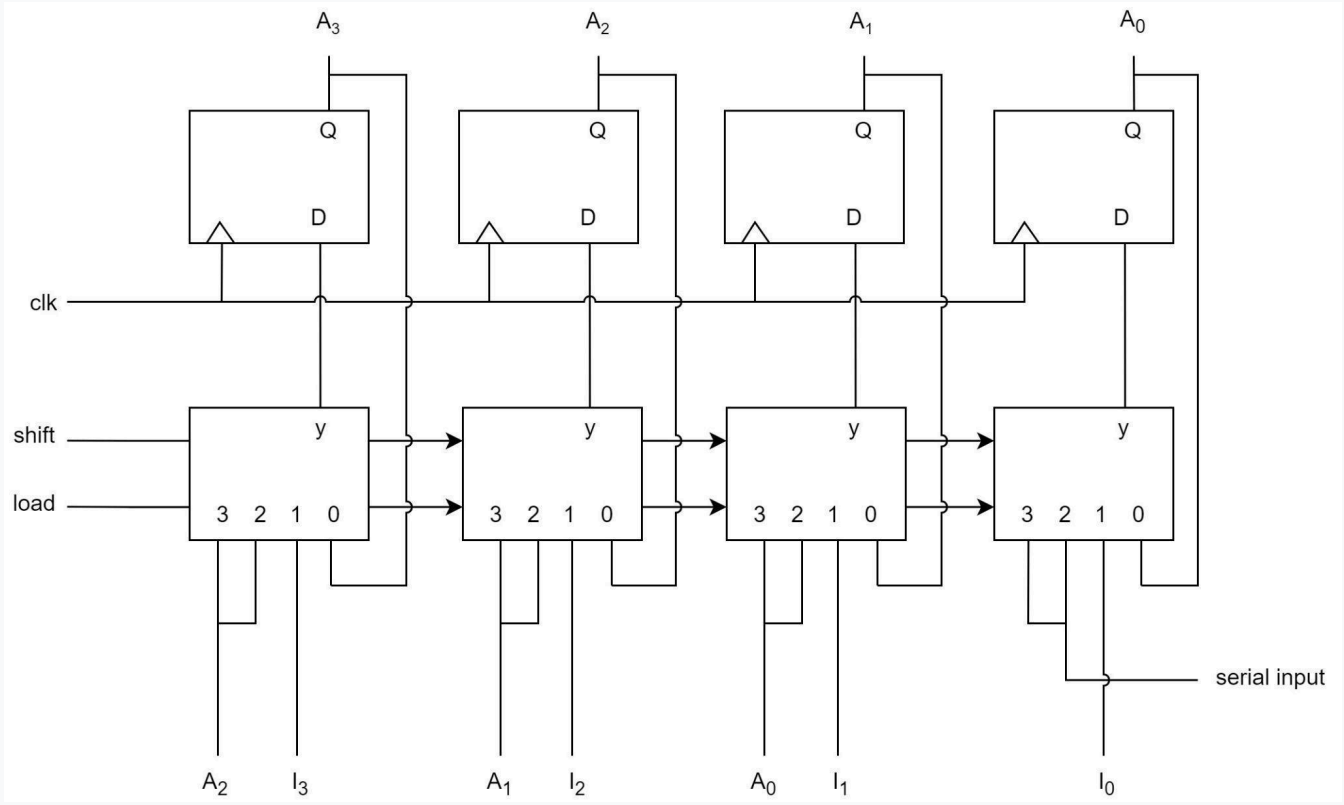
1. **Waveform:**




➢ **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:**

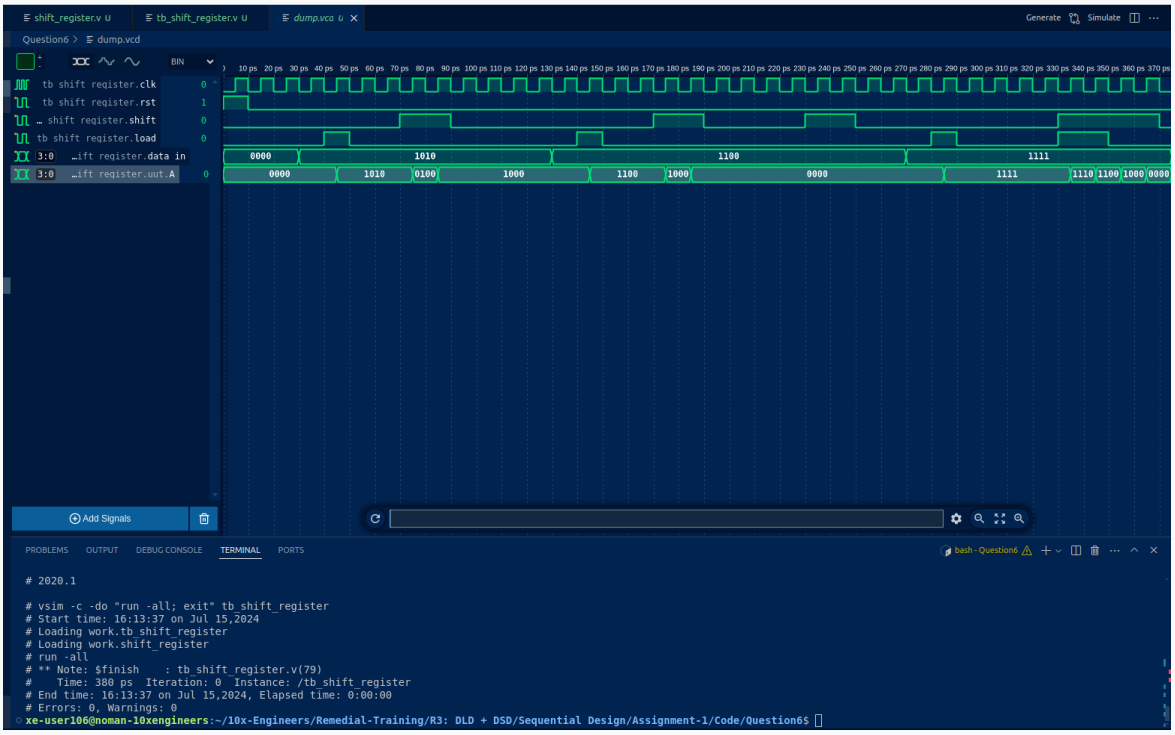
```
    module shift_register (input clk, rst, load, shift, input
[3:0] data_in, output reg [3:0] A);

    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;
```

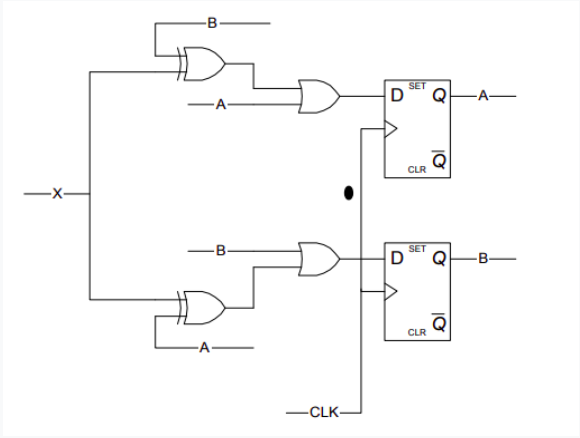
```
        // Test case 1: Load parallel data
        #20 data_in = 4'b1010;
        #10 load = 1;
        #10 load = 0;

        // 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:

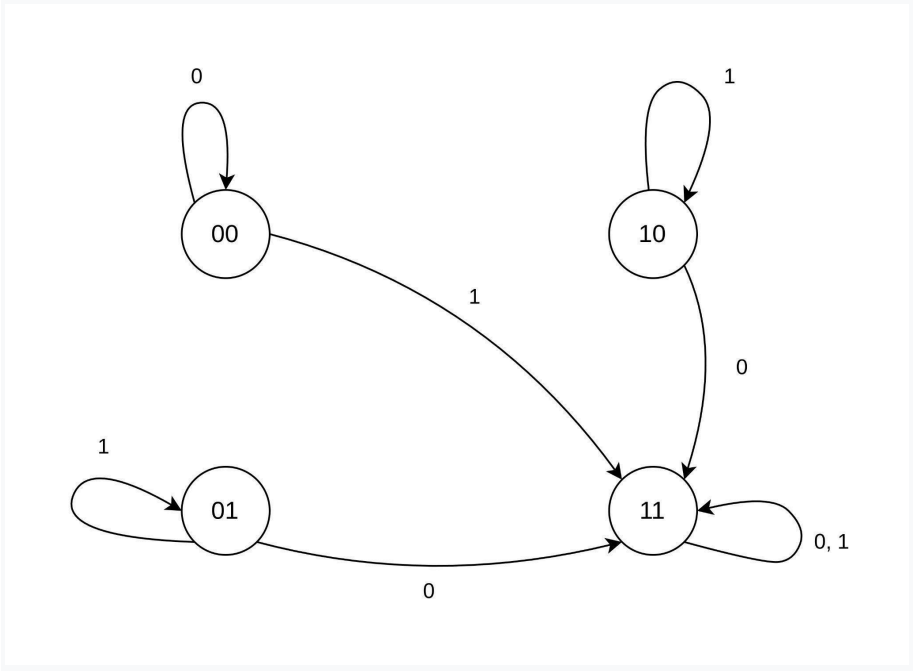



### 1. State Table:

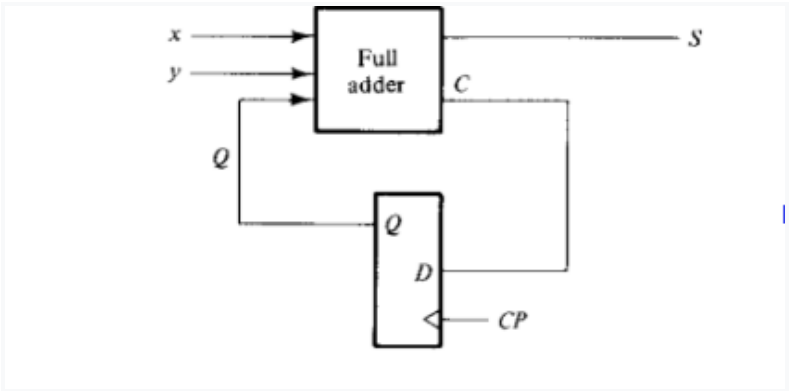
| Present State | | Input | Next State | |
|---|---|---|---|---|
| A | B | X | A+ | B+ |
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 1 |
| 0 | 1 | 0 | 1 | 1 |

| | | | | |
|---|---|---|---|---|
| 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:**



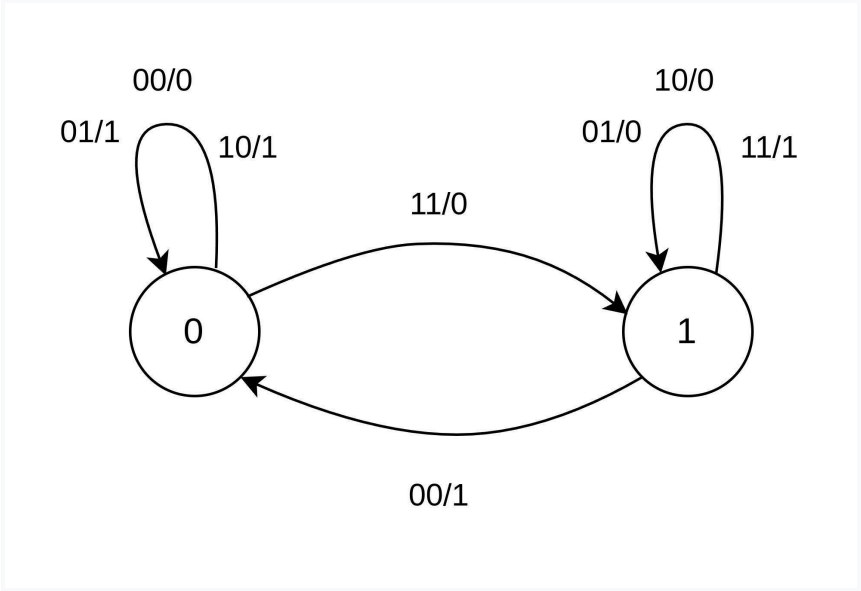

**b. Diagram:**



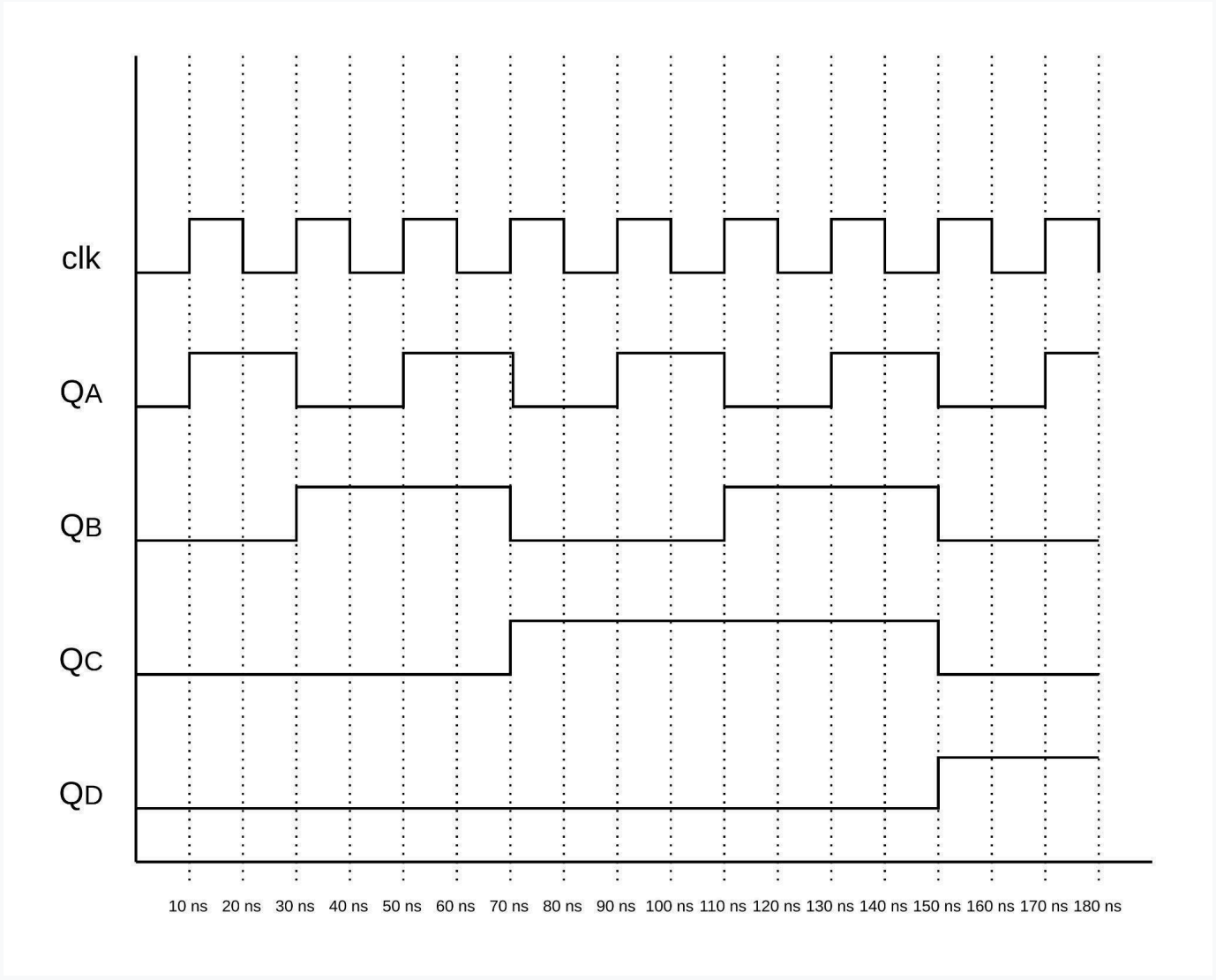
### 1. State Table:

| 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 |

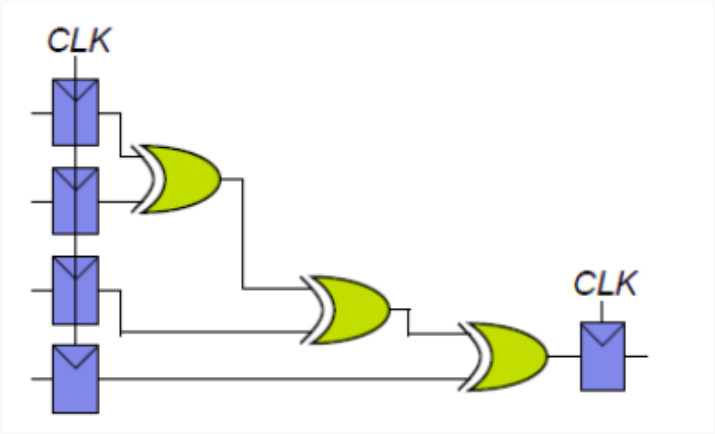
### 2. State Diagram:




➢ **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.

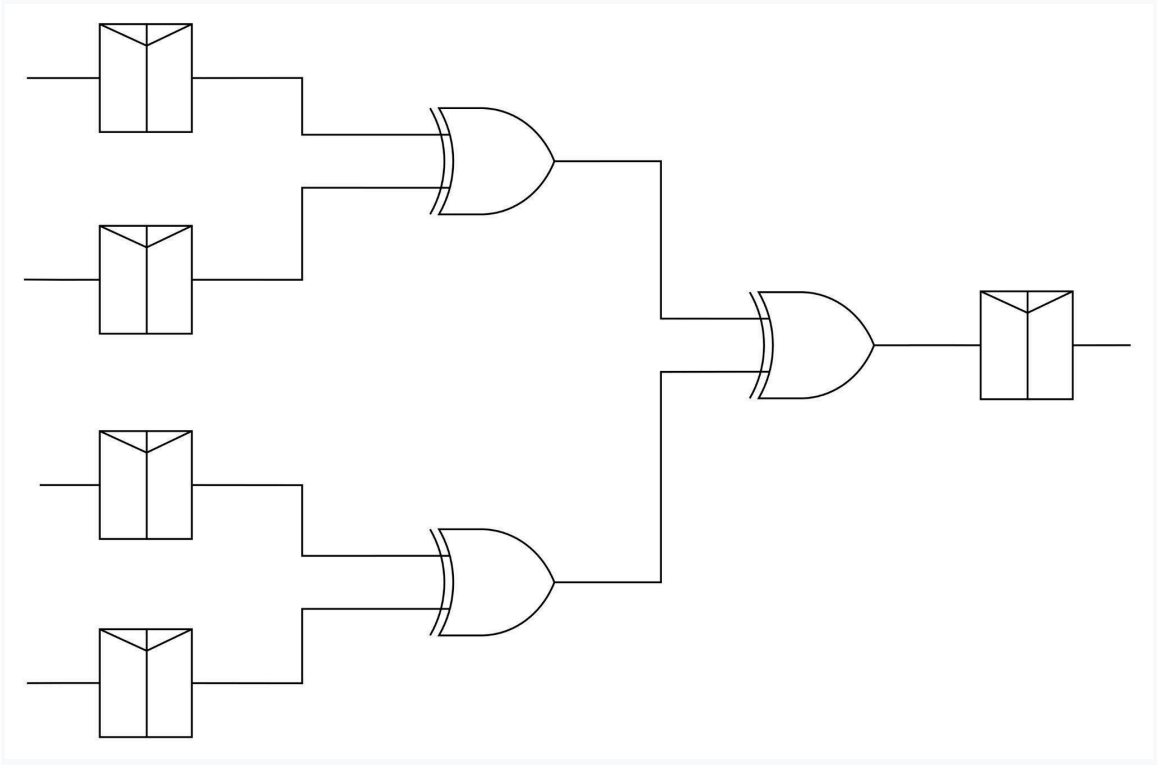
$$TC \geq Tpcq + Tpd + Tsetup$$
$$TC \geq 70 + (100*3) + 60 = 430\ ps$$
$$Max\ Frequency = 1/TC = 2.33\ GHz$$

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

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

**C. Redesign the circuit so that it can be operated at a 3 GHz frequency?**
Minimum clock duration for 3 GHz will be:

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



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

$$T_{ccq} + 2*T_{cd} \geq T_{hold} + T_{skew}$$

$$50 + 2*55 \geq 20 + T_{skew}$$

$$T_{skew} \leq 160 - 20$$

$$T_{skew} \leq 140\ ps$$