CE221-L Digital Logic Design Lab

Lab # 10

Submitted by:

- **Shayan Rizwan** [2024585]

Submitted to: Sir Irfanullah

Semester: 3rd

Faculty of Computer Science and Engineering

GIK Institute of Engineering Sciences and Technology

Task # 1:

Implement a synchronous **SR flip-flop in Verilog** with positive-edge clock triggering, including Set, Reset, Hold, and Invalid state functionality according to standard SR flip-flop truth table behavior.

Design Source [Code]

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 12/01/2025 12:02:51 AM
// Design Name:
// Module Name: sr_flipflop
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//    Positive-edge triggered SR flip-flop with synchronous reset.
//    Implements classic SR flip-flop behavior with defined reset
state.
//    Note: S=R=1 remains invalid/undefined per flip-flop
fundamentals.
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module sr_flipflop(
    input  wire S,
    input  wire R,
    input  wire CLK,
    output reg  Q,
```

```
    output reg  Qbar
);

initial
    begin
        Q = 1'b0;
        Qbar = 1'b1;
    end
always @(posedge CLK) begin
    case ({S, R})
        2'b00: begin        // No change state: outputs retain
previous values
            Q    <= Q;
            Qbar <= Qbar;
        end

        2'b01: begin
            Q    <= 1'b0;      // Reset operation: Q cleared to 0,
Qbar set to 1
            Qbar <= 1'b1;
        end

        2'b10: begin
            Q    <= 1'b1;      // Set operation: Q set to 1, Qbar
cleared to 0
            Qbar <= 1'b0;
        end

        2'b11: begin
            Q    <= 1'bx;      // Invalid condition: both set and
reset asserted
            Qbar <= 1'bx;
        end

        default: begin
            // Catch-all for undefined states (good practice)
            Q    <= 1'bx;
            Qbar <= 1'bx;
        end

    endcase
end

endmodule
```

Test Bench [Code]

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 12/01/2025 12:02:51 AM
// Design Name:
// Module Name: sr_flipflop_tb
// Project Name:
// Target Devices:
// Tool Versions:
// Description: Testbench for SR Flipflop verification
//
// Dependencies: sr_flipflop.v
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module sr_flipflop_tb;


    // Testbench signals
    reg S, R, CLK; // inputs are defined as reg (in the testbench)
    wire Q, Qbar; // outputs are defined as wire (in the testbench)
```

```
// Instantiate the Unit Under Test (UUT)
sr_flipflop uut (
    .S(S),
    .R(R),
    .CLK(CLK),
    .Q(Q),
    .Qbar(Qbar)
);

// Clock generation (100MHz clock - 10ns period)
initial begin
    CLK = 0;
    forever #5 CLK = ~CLK;  // 10ns period clock
end

// Test sequence
initial begin
    // Initialize inputs
    S = 0;
    R = 0;

    // Wait for global reset
    #10;

    // Test case 1: No change (S=0, R=0)
    S = 0; R = 0;
    #10;
```

```
        // Test case 2: Reset (S=0, R=1)
        S = 0; R = 1;
        #10;

        // Test case 3: Set (S=1, R=0)
        S = 1; R = 0;
        #10;

        // Test case 4: Invalid state (S=1, R=1)
        S = 1; R = 1;
        #10;

        // Test case 5: Return to reset state
        S = 0; R = 1;
        #10;

        // Test case 6: Return to set state
        S = 1; R = 0;
        #10;

        // End simulation
        $display("SR Flipflop testbench completed");
        $finish; // DO NOT FORGET THE SEMI COLON AT THE END OF
$finish;
    end
endmodule
```

Output:

Task # 2:

Implement a synchronous **JK flip-flop** in Verilog with positive-edge clock triggering, including Hold, Reset, Set, and Toggle functionality according to standard JK flip-flop truth table behavior, eliminating the invalid state present in SR flip-flops.

Design Source [Code]

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 12/01/2025 12:02:51 AM
// Design Name: JK Flip-Flop
// Module Name: jk_flipflop
// Project Name:
// Target Devices:
// Tool Versions:
// Description: Positive-edge triggered JK flip-flop with synchronous operation
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//    JK flip-flop eliminates invalid state of SR flip-flop using toggle functionality
```

```
//
//////////////////////////////////////////////////////////////////////////////
//////////////


module jk_flipflop(
    output reg Q,         // Main output
    output reg Qbar,      // Complementary output
    input wire CLK,       // Positive-edge trigger clock
    input wire J,         // Set condition input
    input wire K          // Reset condition input
);


    // Sequential logic with positive-edge trigger
    always @(posedge CLK) begin
        case({J, K})
            2'b00: Q <= Q;       // No change: retain previous state
            2'b01: Q <= 1'b0;    // Reset: clear output to 0
            2'b10: Q <= 1'b1;    // Set: set output to 1
            2'b11: Q <= ~Q;      // Toggle: invert current state
        endcase


        Qbar <= ~Q;  // Generate complementary output
        // This is another syntax to define the sequential logic
inside the 'always' block.
    end


endmodule
```

Test Bench [Code]

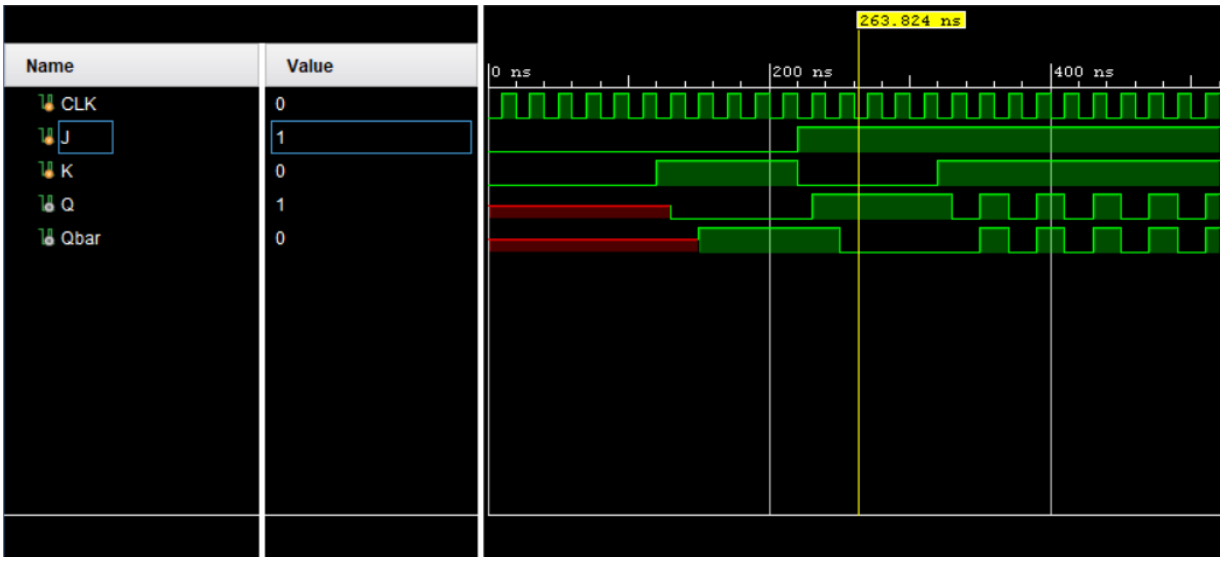
```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 12/01/2025 12:02:51 AM
// Design Name: JK Flip-Flop Testbench
// Module Name: jk_flipflop_tb
// Project Name:
// Target Devices:
// Tool Versions:
// Description: Comprehensive testbench for JK flip-flop verification
//
// Dependencies: jk_flipflop.v
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//    Tests all JK input combinations including toggle functionality
//
//////////////////////////////////////////////////////////////////////////////////


module jk_flipflop_tb;


    // Inputs (test stimuli)
    reg CLK;
```

```
    reg J;
    reg K;

    // Outputs (monitor these)
    wire Q;
    wire Qbar;

    // Instantiate Unit Under Test (UUT)
    jk_flipflop uut (
        .Q(Q),
        .Qbar(Qbar),
        .CLK(CLK),
        .J(J),
        .K(K)
    );

    // Clock generation: 50MHz clock (20ns period)
    initial begin
        CLK = 0;
        forever #10 CLK = ~CLK;  // 20ns period = 50MHz
    end

    // Test sequence
    initial begin
        // Initialize all inputs
        CLK = 0;
        J = 0;
        K = 0;
```

```
        // Wait for initial stabilization
        #20;

        // Test Case 1: No change (J=0, K=0)
        J = 0; K = 0;
        #100;

        // Test Case 2: Reset (J=0, K=1)
        J = 0; K = 1;
        #100;

        // Test Case 3: Set (J=1, K=0)
        J = 1; K = 0;
        #100;

        // Test Case 4: Toggle (J=1, K=1) - Critical test case
        J = 1; K = 1;
        #100;

        // Test Case 5: Verify toggle continues
        J = 1; K = 1;
        #100;

        // End simulation
        $display("JK Flip-Flop test completed successfully");
        $finish;
    end
```

```
endmodule
```

Output



Task # 3:

Implement a synchronous **T flip-flop in Verilog** with positive-edge clock triggering that toggles its output state when enabled and retains its current state when disabled, including proper reset functionality.

Design Source [Code]

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 12/01/2025 12:02:51 AM
// Design Name: T Flip-Flop Derived from JK Flip-Flop
// Module Name: t_flipflop
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//    T Flip-Flop implementation by connecting J and K inputs together.
//    This creates a JK flip-flop where J=K=T, resulting in:
//    - T=0: J=K=0 -> Hold state (no change)
//    - T=1: J=K=1 -> Toggle state (invert output)
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
```

```
// Additional Comments:
//    Fundamental relationship: T Flip-Flop = JK Flip-Flop with J=K=T
//
//////////////////////////////////////////////////////////////////////////////

module t_flipflop(
    output reg Q,         // Flip-flop output
    input wire T,         // Toggle enable input
    input wire CLK,       // Positive-edge trigger clock
    input wire RST        // Asynchronous active-high reset
);

    // Sequential logic with asynchronous reset
    always @(posedge CLK or posedge RST) begin
        if (RST) // Same as: if (RST == 1'b1)
            Q <= 1'b0;           // Reset condition (highest priority)
        else begin
            // T Flip-Flop behavior derived from JK Flip-Flop truth table:
            // When J=K=T, the JK behavior becomes:
            case(T)
                1'b0: Q <= Q;    // T=0: J=0,K=0 -> Hold state (JK: no change)
                1'b1: Q <= ~Q;   // T=1: J=1,K=1 -> Toggle state (JK: toggle)
            endcase

            /*
```

```
            JK Flip-Flop Truth Table (for reference):
            J  K  |  Q(t+1)
            ---------------
            0  0  |   Q(t)    // Hold
            0  1  |     0     // Reset
            1  0  |     1     // Set
            1  1  |   ~Q(t)   // Toggle

            When we set J=K=T:
            T  |  J  K  |  Q(t+1)  | T Flip-Flop Behavior
            -------------------------------------------
            0  |  0  0  |   Q(t)   | Hold (no change)
            1  |  1  1  |   ~Q(t)  | Toggle (invert)
            */
        end
    end

endmodule
```

Test Bench [Code]

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 12/01/2025 12:02:51 AM
// Design Name: T Flip-Flop Testbench
```

```
// Module Name: t_flipflop_tb
// Project Name:
// Target Devices:
// Tool Versions:
// Description: Comprehensive testbench for T flip-flop verification
//    Tests reset functionality, hold state, and toggle operation
//
// Dependencies: t_flipflop.v
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//    Verifies asynchronous reset priority and toggle behavior
//
//////////////////////////////////////////////////////////////////////////////
//////////////


module t_flipflop_tb;

    // Inputs (test stimuli)
    reg T;
    reg CLK;
    reg RST;

    // Outputs (monitor these)
    wire Q;

    // Instantiate Unit Under Test (UUT)
    t_flipflop uut (
```

```
        .Q(Q),
        .T(T),
        .CLK(CLK),
        .RST(RST)
    );

    // Clock generation: 50MHz clock (20ns period)
    initial begin
        CLK = 0;
        forever #10 CLK = ~CLK;  // 20ns period = 50MHz
    end

    // Test sequence
    initial begin
        // Initialize all inputs
        CLK = 0;
        RST = 1;    // Assert reset initially
        T = 0;

        // Test Case 1: Initial reset condition
        #15;        // Wait past first clock edge

        // Test Case 2: Release reset, T=0 (hold state)
        RST = 0;
        T = 0;
        #40;

        // Test Case 3: T=1 (toggle mode - multiple cycles)
        T = 1;
```

```
        #80;        // Allow multiple toggle cycles

        // Test Case 4: T=0 (hold current state)
        T = 0;
        #40;

        // Test Case 5: T=1 (resume toggling)
        T = 1;
        #40;

        // Test Case 6: Asynchronous reset assertion during
operation
        RST = 1;
        #20;

        // Test Case 7: Continue operation after reset release
        RST = 0;
        T = 1;
        #60;

        // End simulation
        $display("T Flip-Flop test completed successfully");
        $finish;
    end


endmodule
```

Output