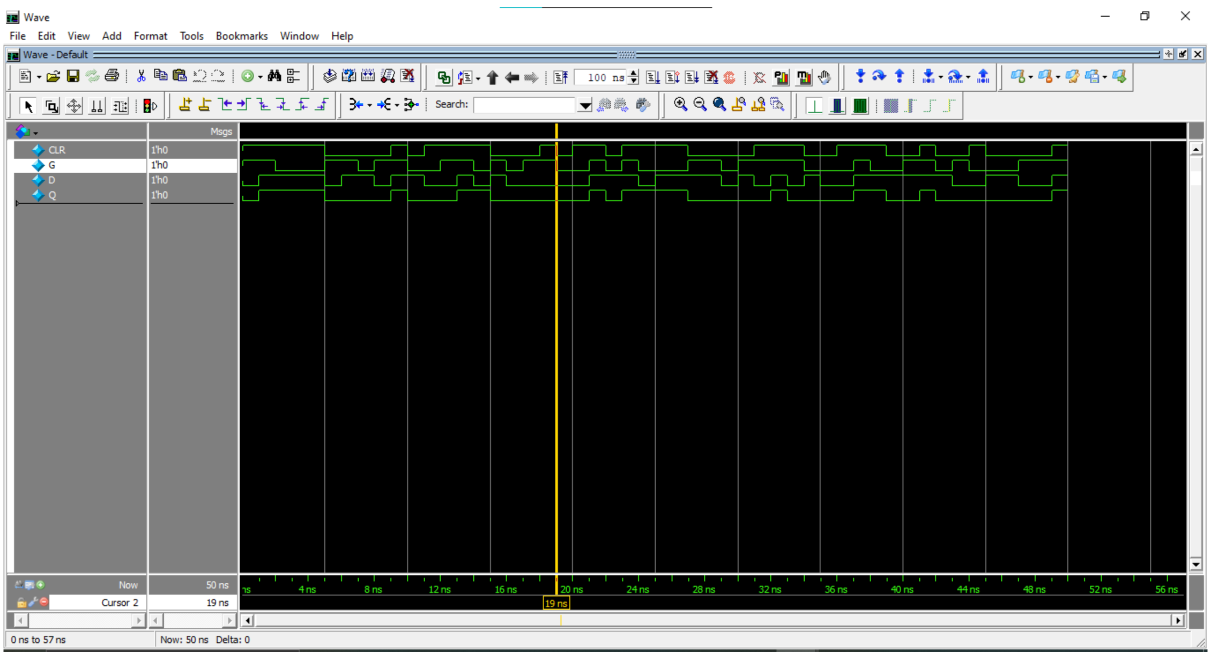
Sequential Logic Design

1.

```
module D_latch (
    input wire D,   // Data Input
    input wire G,   // Enable (Gate)
    input wire CLR, // Active Low Clear
    output reg Q    // Output
);
    always @(*) begin
        if (!CLR) begin     // Active low clear
            Q = 0;
        end else if (G) begin   // When G is high, latch follows D
            Q = D;
        end
        // When G is low, Q retains its value (latch behavior)
    end
endmodule

module D_latch_tb;
    reg D, G, CLR;
    wire Q;

    // Instantiate the data latch
    D_latch uut (
        .D(D),
        .G(G),
        .CLR(CLR),
        .Q(Q)
    );

    // Testbench logic
    initial begin
        // Randomized test sequence
        repeat (50) begin
            D = $random;
            G = $random;
            CLR = $random;
            #1; // Delay to avoid treating G as a clock
            $display("Time=%0t | CLR=%b, G=%b, D=%b | Q=%b", $time, CLR, G, D,
Q);
        end

        // End simulation
        $finish;
    end

endmodule
```



```
# Time=1 | CLR=1, G=1, D=0 | Q=0
# Time=2 | CLR=1, G=1, D=1 | Q=1
# Time=3 | CLR=1, G=0, D=1 | Q=1
# Time=4 | CLR=1, G=0, D=1 | Q=1
# Time=5 | CLR=1, G=0, D=1 | Q=1
# Time=6 | CLR=0, G=1, D=0 | Q=0
# Time=7 | CLR=0, G=1, D=1 | Q=0
# Time=8 | CLR=0, G=0, D=1 | Q=0
# Time=9 | CLR=0, G=1, D=0 | Q=0
# Time=10 | CLR=1, G=1, D=1 | Q=1
# Time=11 | CLR=0, G=0, D=1 | Q=0
# Time=12 | CLR=1, G=0, D=0 | Q=0
# Time=13 | CLR=1, G=1, D=0 | Q=0
# Time=14 | CLR=1, G=1, D=1 | Q=1
# Time=15 | CLR=1, G=0, D=0 | Q=1
# Time=16 | CLR=0, G=1, D=1 | Q=0
# Time=17 | CLR=0, G=0, D=0 | Q=0
# Time=18 | CLR=0, G=1, D=0 | Q=0
# Time=19 | CLR=1, G=1, D=0 | Q=0
# Time=20 | CLR=0, G=0, D=0 | Q=0
# Time=21 | CLR=1, G=0, D=0 | Q=0
# Time=22 | CLR=1, G=1, D=1 | Q=1
# Time=23 | CLR=0, G=0, D=1 | Q=0
# Time=24 | CLR=1, G=1, D=1 | Q=1
# Time=25 | CLR=1, G=0, D=0 | Q=1
# Time=26 | CLR=1, G=0, D=1 | Q=1
# Time=27 | CLR=1, G=0, D=1 | Q=1
# Time=28 | CLR=0, G=1, D=1 | Q=0
# Time=29 | CLR=0, G=1, D=1 | Q=0
# Time=30 | CLR=0, G=0, D=0 | Q=0
```

2. A.

```verilog
module tff (
    input wire t,        // Toggle input
    input wire clk,      // Clock input
    input wire rstn,     // Active low asynchronous reset
    output reg q,        // Output
    output wire qbar     // Complementary output
);
    assign qbar = ~q;

    always @(posedge clk or negedge rstn) begin
        if (!rstn)       // Active low reset
            q <= 0;
        else if (t)      // Toggle output when t = 1
            q <= ~q;
    end

endmodule
```

B.

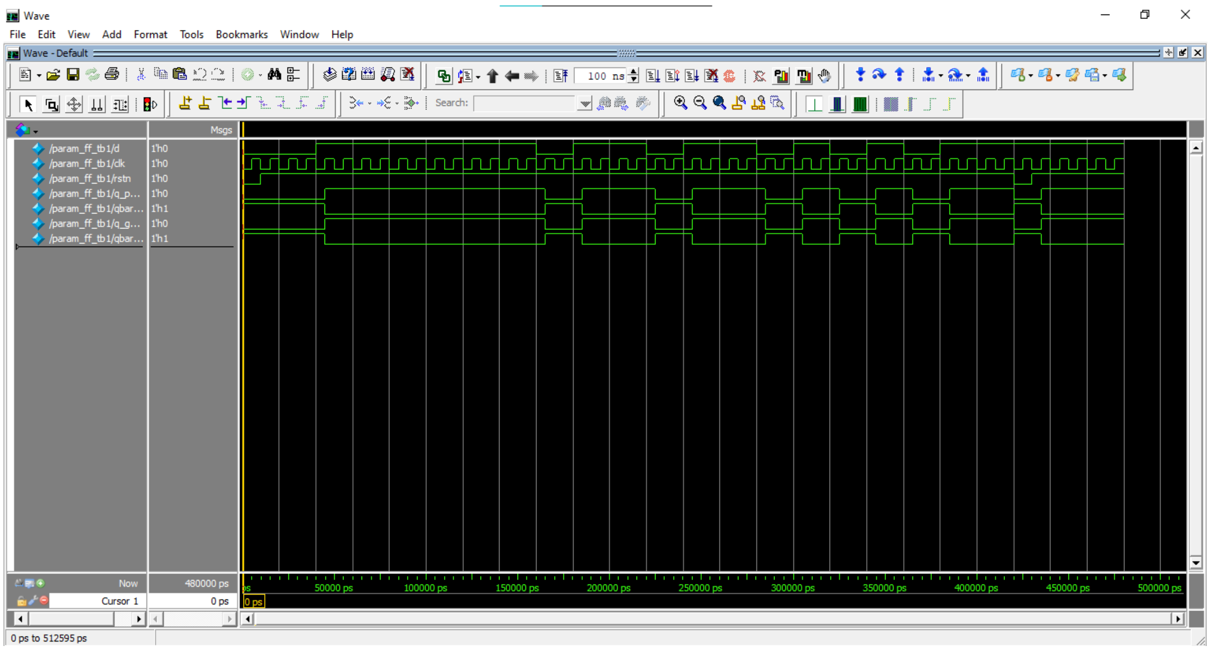
```verilog
module dff (
    input wire d,       // Data input
    input wire rstn,    // Active low asynchronous reset
    input wire clk,     // Clock input
    output reg q,       // Flip-Flop output
    output wire qbar    // Complementary output (inverted q)
);
    assign qbar = ~q;       // Complementary output

    always @(posedge clk or negedge rstn) begin
        if (!rstn) begin    // Asynchronous reset (active low)
            q <= 0;         // Reset output to 0
        end else begin
            q <= d;         // On clock edge, store input d
        end
    end

endmodule
```

C.

```
module parameterized_ff
#(
    parameter FF_TYPE = "DFF"  // Parameter to select flip-flop type: "DFF" or
"TFF"
)(
    input wire d,      // Data or toggle input
    input wire rstn,   // Active low reset
    input wire clk,    // Clock input
    output reg q,      // Output
    output wire qbar   // Complement of the output
);

    // qbar is always the complement of q
    assign qbar = ~q;

    // Always block triggered on the rising edge of clk or falling edge of
rstn
    always @(posedge clk or negedge rstn) begin
        if (!rstn) begin
            // If reset is active (low), set q to 0
            q <= 0;
        end else begin
            if (FF_TYPE == "DFF") begin
                // D Flip-Flop behavior: q follows d
                q <= d;
            end else if (FF_TYPE == "TFF") begin
                // T Flip-Flop behavior: q toggles if d (toggle input) is 1
                q <= q ^ d;
            end
        end
    end

endmodule
/*
    generate
        if (FF_TYPE == "DFF") begin : dff_inst
            dff dff_module (
                .d(d),
                .rstn(rstn),
                .clk(clk),
                .q(q),
                .qbar(qbar)
            );
        end else if (FF_TYPE == "TFF") begin : tff_inst
            tff tff_module (
                .t(d),      // 'd' acts as 't' in T Flip-Flop mode
                .rstn(rstn),
                .clk(clk),
                .q(q),
                .qbar(qbar)
            );
        end
    endgenerate
*/
```

D. 1.

```
module param_ff_tb1;

    reg d, clk, rstn;
    wire q_param, qbar_param;
    wire q_golden, qbar_golden;

    // Instantiate parameterized Flip-Flop with DFF behavior
    param_ff #(.FF_TYPE("DFF")) uut (
        .d(d),
        .rstn(rstn),
        .clk(clk),
        .q(q_param),
        .qbar(qbar_param)
    );

    // Instantiate golden model (DFF)
    dff golden_model (
        .d(d),
        .rstn(rstn),
        .clk(clk),
        .q(q_golden),
        .qbar(qbar_golden)
    );

    // Clock Generation
    always #5 clk = ~clk;  // Generate a 10ns clock period

    // Task to check correctness
   task check_output;
      if (q_param !== q_golden || qbar_param !== qbar_golden)
         $display("Mismatch at time %0t: q_param=%b, q_golden=%b", $time, q_param, q_golden);
      else
         $display("Pass at time %0t: q_param=%b, q_golden=%b", $time, q_param, q_golden);
   endtask

    initial begin
        // Initialize signals
        clk = 0;
        rstn = 0;
        d = 0;

        // Apply Reset
        #10 rstn = 1;

        // Apply test vectors using repeat loop
        repeat (20) begin
            #10 d = $random;
            #10 check_output;
        end

        // Apply reset and observe outputs
        #10 rstn = 0;
        #10 rstn = 1;
        check_output;

        // Finish simulation
        #50 $finish;
    end

endmodule
```

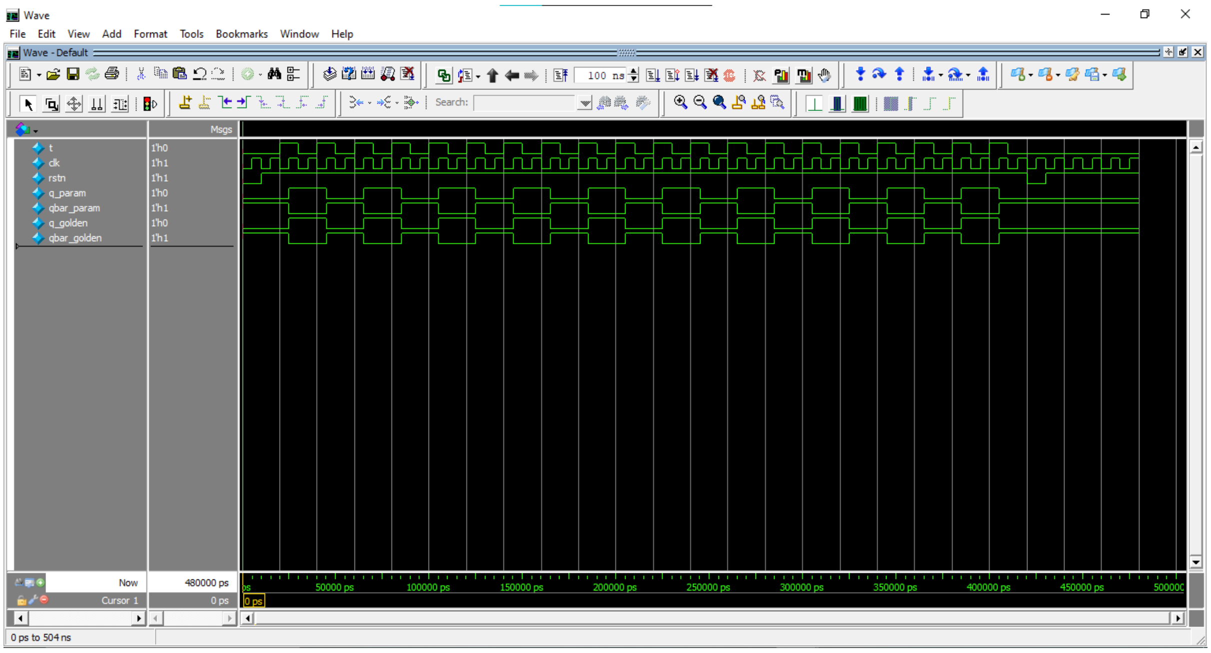


```
# Pass at time 30000: q_param=0, q_golden=0
# Pass at time 50000: q_param=1, q_golden=1
# Pass at time 70000: q_param=1, q_golden=1
# Pass at time 90000: q_param=1, q_golden=1
# Pass at time 110000: q_param=1, q_golden=1
# Pass at time 130000: q_param=1, q_golden=1
# Pass at time 150000: q_param=1, q_golden=1
# Pass at time 170000: q_param=0, q_golden=0
# Pass at time 190000: q_param=1, q_golden=1
# Pass at time 210000: q_param=1, q_golden=1
# Pass at time 230000: q_param=0, q_golden=0
# Pass at time 250000: q_param=1, q_golden=1
# Pass at time 270000: q_param=1, q_golden=1
# Pass at time 290000: q_param=0, q_golden=0
# Pass at time 310000: q_param=1, q_golden=1
# Pass at time 330000: q_param=0, q_golden=0
# Pass at time 350000: q_param=1, q_golden=1
# Pass at time 370000: q_param=0, q_golden=0
# Pass at time 390000: q_param=1, q_golden=1
# Pass at time 410000: q_param=1, q_golden=1
# Pass at time 430000: q_param=0, q_golden=0
```

D. 2.

```
module param_ff_tb2;

    reg t, clk, rstn;
    wire q_param, qbar_param;
    wire q_golden, qbar_golden;

    // Instantiate parameterized Flip-Flop with TFF behavior
    param_ff #(.FF_TYPE("TFF")) uut (
        .d(t),  // In TFF mode, d acts as t
        .rstn(rstn),
        .clk(clk),
        .q(q_param),
        .qbar(qbar_param)
    );

    // Instantiate golden model (TFF)
    tff golden_model (
        .t(t),
        .rstn(rstn),
        .clk(clk),
        .q(q_golden),
        .qbar(qbar_golden)
    );

    // Clock Generation
    always #5 clk = ~clk;  // Generate a 10ns clock period
    // Task to check correctness
    task check_output;
      if (q_param !== q_golden || qbar_param !== qbar_golden)
          $display("Mismatch at time %0t: q_param=%b, q_golden=%b", $time, q_param, q_golden);
```

```
            else
                $display("Pass at time %0t: q_param=%b, q_golden=%b", $time, q_param, q_golden);
        endtask

    initial begin
        // Initialize signals
        clk = 0;
        rstn = 0;
        t = 0;

        // Apply Reset
        #10 rstn = 1;

        // Apply test vectors using repeat loop
        repeat (20) begin
            #10 t = 1; // Toggle
            #10 t = 0;
            check_output;
        end

        // Apply reset and observe outputs
        #10 rstn = 0;
        #10 rstn = 1;
        check_output;

        // Finish simulation
        #50 $finish;
    end

endmodule
```


Wave
File Edit View Add Format Tools Bookmarks Window Help
Wave - Default
Msgs
t 1'h0
clk 1'h1
rstn 1'h1
q_param 1'h0
qbar_param 1'h1
q_golden 1'h0
qbar_golden 1'h1
Now 480000 ps
Cursor 1 0 ps
0 ps
ps 50000 ps 100000 ps 150000 ps 200000 ps 250000 ps 300000 ps 350000 ps 400000 ps 450000 ps 500000
0 ps to 504 ns

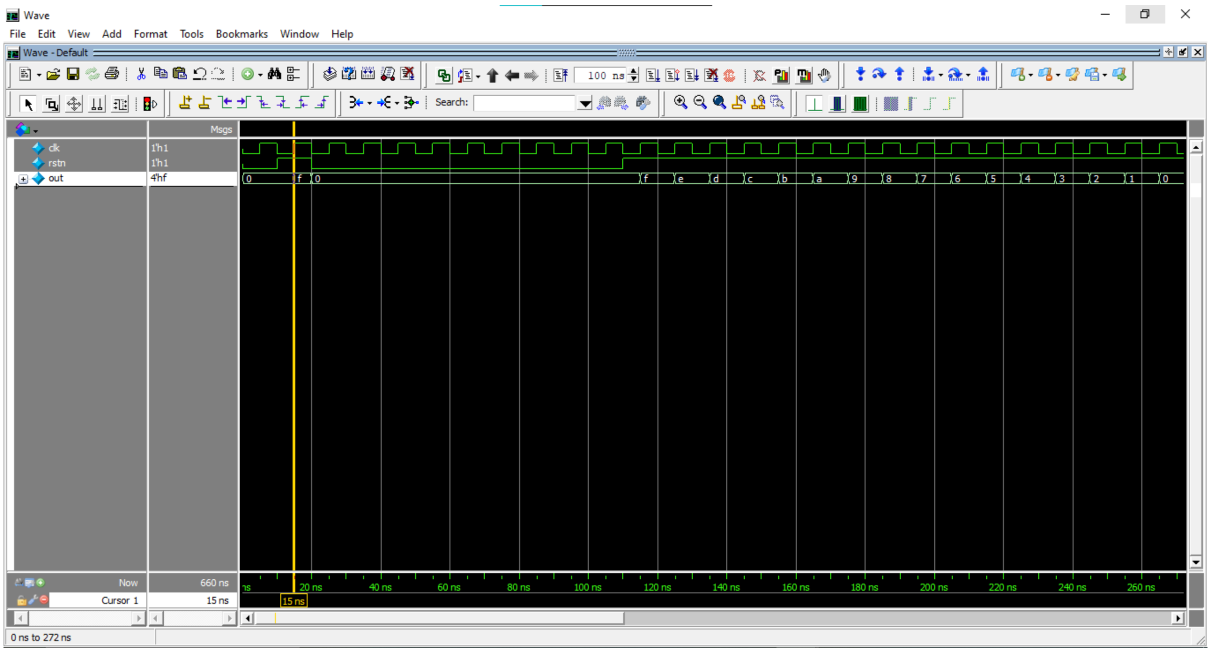
```
# Pass at time 30000: q_param=1, q_golden=1
# Pass at time 50000: q_param=0, q_golden=0
# Pass at time 70000: q_param=1, q_golden=1
# Pass at time 90000: q_param=0, q_golden=0
# Pass at time 110000: q_param=1, q_golden=1
# Pass at time 130000: q_param=0, q_golden=0
# Pass at time 150000: q_param=1, q_golden=1
# Pass at time 170000: q_param=0, q_golden=0
# Pass at time 190000: q_param=1, q_golden=1
# Pass at time 210000: q_param=0, q_golden=0
# Pass at time 230000: q_param=1, q_golden=1
# Pass at time 250000: q_param=0, q_golden=0
# Pass at time 270000: q_param=1, q_golden=1
# Pass at time 290000: q_param=0, q_golden=0
# Pass at time 310000: q_param=1, q_golden=1
# Pass at time 330000: q_param=0, q_golden=0
# Pass at time 350000: q_param=1, q_golden=1
# Pass at time 370000: q_param=0, q_golden=0
# Pass at time 390000: q_param=1, q_golden=1
# Pass at time 410000: q_param=0, q_golden=0
# Pass at time 430000: q_param=0, q_golden=0
```

```
// run.do
vlib wrok

# Compile Design and Testbenches
vlog param_FF.v
vlog D_FF.v
vlog T_FF.v
vlog param_FF_tb1.v
vlog param_FF_tb2.v

# Run Testbench 1 (DFF mode)
vsim -voptargs=+acc param_ff_tb1
add wave *
run -all
quit

# Run Testbench 2 (TFF mode)
vsim -voptargs=+acc param_FF_tb2
add wave *
run -all
quit
```

3.

```
module ripple_counter (
    input wire clk,     // Clock input
    input wire rstn,    // Active low reset
    output wire [3:0] out // 4-bit output
);

    // Internal signals for the flip-flop outputs
    wire [3:0] q;

    // Generate block to instantiate D flip-flops
    genvar i;
    generate
        for (i = 0; i < 4; i = i + 1) begin : gen_ff
            dff ff (
                .d(~q[i]),  // Toggle input
                .rstn(rstn),
                .clk(i == 0 ? clk : q[i-1]), // First flip-flop, clocked by the main clock
                .q(q[i]),
                .qbar()
            );
        end
    endgenerate

    // Assign the outputs
    assign out = q;
endmodule

module ripple_counter_tb;

    // Declare testbench signals
    reg clk;
    reg rstn;
    wire [3:0] out;

    // Instantiate the 4-bit ripple counter
    ripple_counter uut (
        .clk(clk),
        .rstn(rstn),
        .out(out)
    );

    // Clock generation: 10 time unit period
    always begin
        #5 clk = ~clk; // Generate a clock with a period of 10 time units
    end
```

```verilog
    // Stimulus process: Generate randomized reset and check the output
    initial begin
        // Initialize signals
        clk = 0;
        rstn = 0;

        // Apply reset
        #10 rstn = 1; // Assert reset after 10 time units

        // Randomize and test the counter's behavior
        #10 rstn = 0; // De-assert reset

        // Wait for a few clock cycles and then check the output
        #40;

        // Randomize the reset and observe the ripple counter
        repeat (10) begin
            #20 rstn = $random; // Randomize reset signal
            #10; // Wait for a clock cycle
        end

        // End simulation after 200 time units
        #200;
        $stop;
    end

    // Monitor the outputs to check correctness
    initial begin
        $monitor("At time %t: out = %b", $time, out);
    end
endmodule
```


Wave
File Edit View Add Format Tools Bookmarks Window Help
Wave - Default
100 ns
Search:
Msgs
clk 1'h1
rstn 1'h1
out 4'hf
0 f 0 f e d c b a 9 8 7 6 5 4 3 2 1 0
Now 660 ns
Cursor 1 15 ns
ns 20 ns 40 ns 60 ns 80 ns 100 ns 120 ns 140 ns 160 ns 180 ns 200 ns 220 ns 240 ns 260 ns
15 ns
0 ns to 272 ns

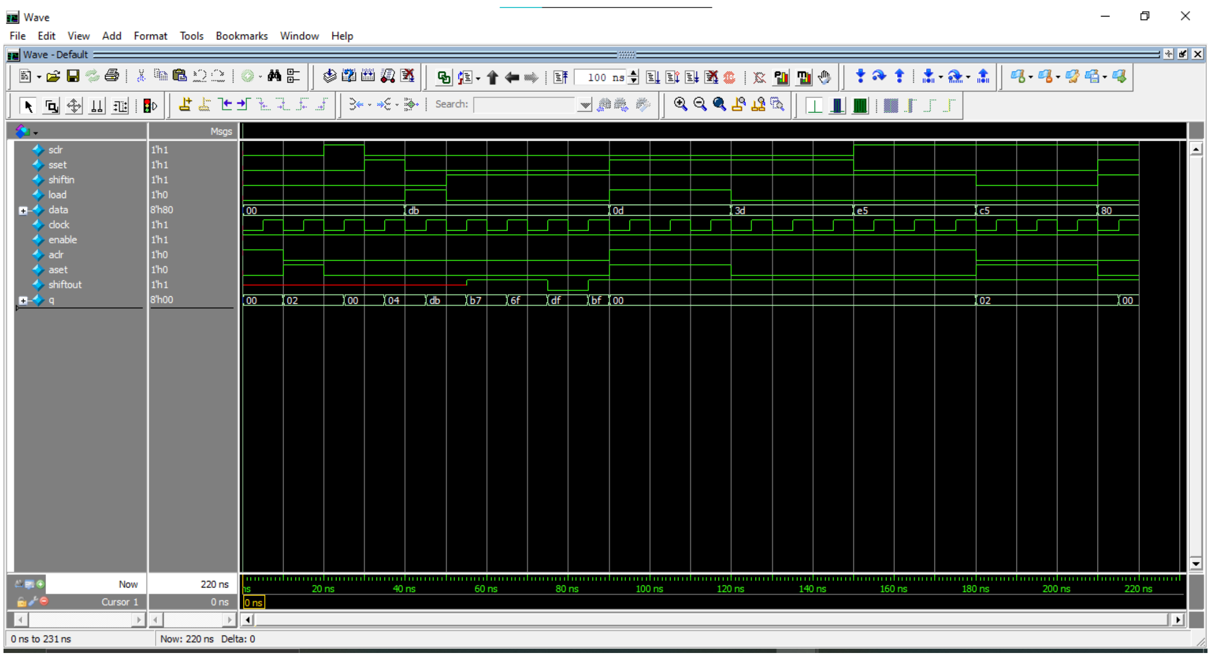
4.

```
module param_shift_register #(
    parameter LOAD_AVALUE = 1,
    parameter LOAD_SVALUE = 1,
    parameter SHIFT_WIDTH = 8,
    parameter SHIFT_DIRECTION = "LEFT" // "LEFT" or "RIGHT"
)(
    input wire sclr,          // Synchronous clear
    input wire sset,          // Synchronous set
    input wire shiftin,       // Serial shift data input
    input wire load,          // Synchronous parallel load
    input wire [SHIFT_WIDTH-1:0] data, // Data input
    input wire clock,         // Clock input
    input wire enable,        // Clock enable
    input wire aclr,          // Asynchronous clear
    input wire aset,          // Asynchronous set
    output reg shiftout,      // Serial shift data output
    output reg [SHIFT_WIDTH-1:0] q // Data output
);

always @(posedge clock or posedge aclr or posedge aset) begin
    if (aclr) begin
        q <= 0;
    end else if (aset) begin
        q <= LOAD_AVALUE;
    end else if (enable) begin
        if (sclr) begin
            q <= 0;
        end else if (sset) begin
            q <= LOAD_SVALUE;
        end else if (load) begin
            q <= data;
        end else begin
            if (SHIFT_DIRECTION == "LEFT") begin
                shiftout <= q[SHIFT_WIDTH-1];
                // shiftout gets the leftmost bit of q
                q <= {q[SHIFT_WIDTH-2:0], shiftin};
                // Shift left and append shiftin to the rightmost bit
            end else if (SHIFT_DIRECTION == "RIGHT") begin
                shiftout <= q[0];  // shiftout gets the rightmost bit of q
                q <= {shiftin, q[SHIFT_WIDTH-1:1]};
               // Shift right and append shiftin to the leftmost bit
            end
        end
    end
end
endmodule
```

```verilog
module param_shift_register_tb;
    // Declare testbench signals
    reg sclr;
    reg sset;
    reg shiftin;
    reg load;
    reg [7:0] data;  // SHIFT_WIDTH = 8
    reg clock;
    reg enable;
    reg aclr;
    reg aset;
    wire shiftout;
    wire [7:0] q;

    // Instantiate the parameterized shift register
    param_shift_register #(
        .LOAD_AVALUE(2),           // LOAD_AVALUE = 2
        .LOAD_SVALUE(4),           // LOAD_SVALUE = 4
        .SHIFT_DIRECTION("LEFT"),  // SHIFT_DIRECTION = "LEFT"
        .SHIFT_WIDTH(8)            // SHIFT_WIDTH = 8
    ) uut (
        .sclr(sclr),
        .sset(sset),
        .shiftin(shiftin),
        .load(load),
        .data(data),
        .clock(clock),
        .enable(enable),
        .aclr(aclr),
        .aset(aset),
        .shiftout(shiftout),
        .q(q)
    );

    // Clock generation: 10 time unit period
    always begin
        #5 clock = ~clock; // Generate a clock with a period of 10 time units
    end

    // Task to verify expected behavior and output q
    task verify_q(input [7:0] expected_q);
       begin
           if (q == expected_q)
               $display("TEST PASSED: Expected q = %b, Actual q = %b", expected_q, q);
           else
               $display("TEST FAILED: Expected q = %b, Actual q = %b", expected_q, q);
       end
   endtask
```

```
// Stimulus process: Generate test cases
initial begin
    // Initialize signals
    clock = 0;
    sclr = 0;
    sset = 0;
    shiftin = 0;
    load = 0;
    data = 8'b00000000;
    enable = 1;
    aclr = 0;
    aset = 0;

    // Test 2.1: Verify Asynchronous Clear (aclr)
    aclr = 1; // Assert aclr
    aset = 0; // Deassert aset
    #10; // Wait a few time units
    verify_q(8'b00000000); // q should be cleared

    // Test 2.2: Verify Asynchronous Set (aset)
    aclr = 0; // Deassert aclr
    aset = 1; // Assert aset
    #10; // Wait a few time units
    verify_q(8'b00000010); // q should be loaded with LOAD_AVALUE (2)

    // Test 2.3: Verify Synchronous Clear (sclr)
    aclr = 0; // Deassert aclr
    aset = 0; // Deassert aset
    sclr = 1; // Assert sclr
    sset = 0; // Deassert sset
    #10; // Wait a few time units
    verify_q(8'b00000000); // q should be cleared due to sclr

    // Test 2.4: Verify Synchronous Set (sset)
    sclr = 0; // Deassert sclr
    sset = 1; // Assert sset
    #10; // Wait a few time units
    verify_q(8'b00000100); // q should be loaded with LOAD_SVALUE (4)

    // Test 2.5: Verify Load Functionality
    sclr = 0; // Deassert sclr
    sset = 0; // Deassert sset
    load = 1; // Assert load
    data = 8'b11011011; // Set input data
    #10; // Wait a few time units
    verify_q(8'b11011011); // q should be loaded with the data value
```

```verilog
        // Test 2.6: Verify Shifting Functionality (Shift Left)
        load = 0; // Deassert load
        sclr = 0; // Deassert sclr
        sset = 0; // Deassert sset
        shiftin = 1; // Set serial shift input
        #10; // Wait for 1 clock cycle
        verify_q(8'b10110111); // Check the shifting behavior
        #10;
        verify_q(8'b01101111); // Continue checking shifting

        // Randomized tests: Generate random values for control signals
        repeat (5) begin
            #20;
            sclr = $random;
            sset = $random;
            shiftin = $random;
            load = $random;
            data = $random;
            aclr = $random;
            aset = $random;
            #10;
        end
        // End simulation
        $stop;
    end
    // Monitor the outputs to check correctness
    initial begin
        $monitor("At time %t: q = %b, shiftout = %b", $time, q, shiftout);
    end
endmodule
```

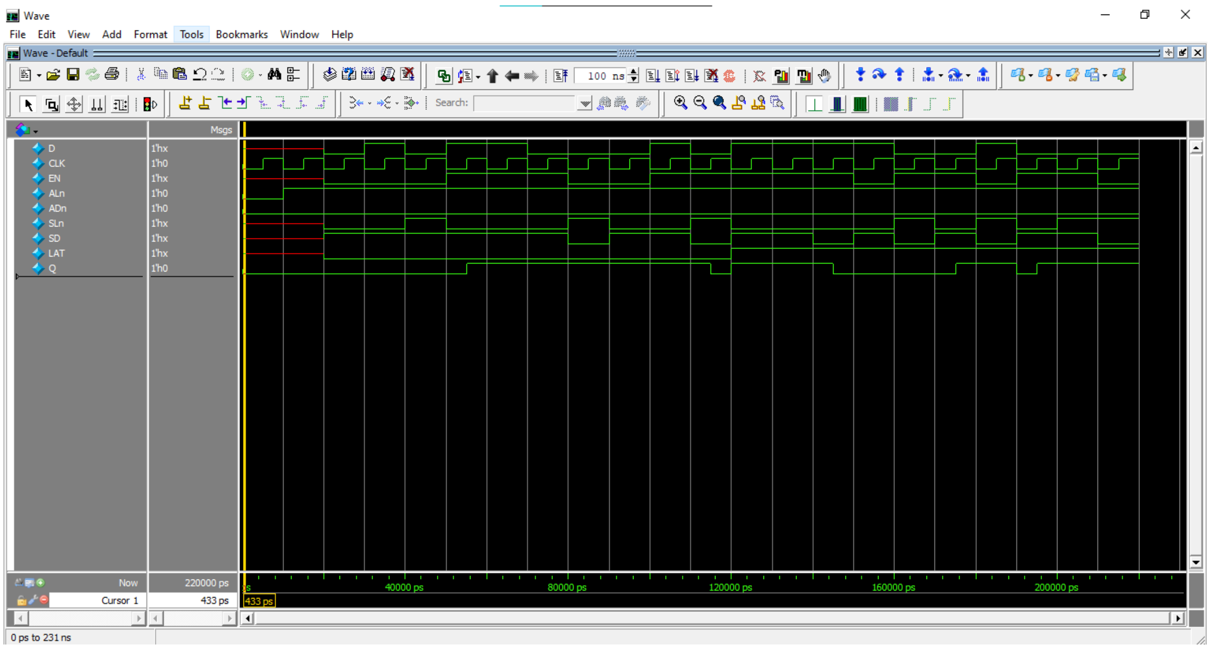


```
sclr      1'h1
sset      1'h1
shiftin   1'h1
load      1'h0
data      8'h80    00 db 0d 3d e5 c5 80
clock     1'h1
enable    1'h1
aclr      1'h0
aset      1'h0
shiftout  1'h1
q         8'h00    00 02 00 04 db b7 6f df bf 00 02 00
Now       220 ns
Cursor 1  0 ns
```

```
# At time                    0: q = 00000000, shiftout = x
# TEST PASSED: Expected q = 00000000, Actual q = 00000000
# At time                   10: q = 00000010, shiftout = x
# TEST PASSED: Expected q = 00000010, Actual q = 00000010
# At time                   25: q = 00000000, shiftout = x
# TEST PASSED: Expected q = 00000000, Actual q = 00000000
# At time                   35: q = 00000100, shiftout = x
# TEST PASSED: Expected q = 00000100, Actual q = 00000100
# At time                   45: q = 11011011, shiftout = x
# TEST PASSED: Expected q = 11011011, Actual q = 11011011
# At time                   55: q = 10110111, shiftout = 1
# TEST PASSED: Expected q = 10110111, Actual q = 10110111
# At time                   65: q = 01101111, shiftout = 1
# TEST PASSED: Expected q = 01101111, Actual q = 01101111
# At time                   75: q = 11011111, shiftout = 0
# At time                   85: q = 10111111, shiftout = 1
# At time                   90: q = 00000000, shiftout = 1
# At time                  180: q = 00000010, shiftout = 1
# At time                  215: q = 00000000, shiftout = 1
```

5.

```
module SLE (
    input wire D,       // Data input
    input wire CLK,     // Clock input
    input wire EN,      // Enable
    input wire ALn,     // Asynchronous Load (Active Low)
    input wire ADn,     // Asynchronous Data (Active Low)
    input wire SLn,     // Synchronous Load (Active Low)
    input wire SD,      // Synchronous Data
    input wire LAT,     // Latch Enable
    output reg Q        // Output
);

    // Always block for asynchronous operations (Asynchronous Load)
    always @(posedge CLK or negedge ALn) begin
        if (!ALn) begin
            Q <= ADn;  // Asynchronous load when ALn is low
        end
        else if (EN) begin
            if (!SLn) begin
                Q <= SD;  // Synchronous load when SLn is low
            end
            else if (!LAT) begin
                Q <= D;  // Flip-Flop Mode (LAT=0)
            end
        end
    end

    // Always block for latch behavior (LAT=1)
    always @(*) begin
        if (LAT) begin
            Q = D;  // Latch Mode (Transparent when LAT=1)
        end
    end

endmodule


module SLE_tb;
    // Testbench Signals
    reg D, CLK, EN, ALn, ADn, SLn, SD, LAT;
    wire Q;

    // Instantiate the DUT (Device Under Test)
    SLE uut (
        .D(D),
        .CLK(CLK),
        .EN(EN),
        .ALn(ALn),
        .ADn(ADn),
        .SLn(SLn),
        .SD(SD),
        .LAT(LAT),
        .Q(Q)
    );

    // Clock Generation (50% Duty Cycle)
    initial begin
        CLK = 0;
        forever #5 CLK = ~CLK; // Toggle clock every 5ns
    end

    // Test Sequence
    initial begin
        // Step 1: Activate ALn (Set to 0)
        ALn = 0;
        ADn = $random;  // Random asynchronous data
        #10;
        $display("TEST: ALn Activated | Expected Q = %b, Actual Q = %b", ADn, Q);

        // Step 2: Deactivate ALn (Set to 1)
        ALn = 1;
        #10;
        // Step 3: Flip-Flop Mode (LAT = 0)
        LAT = 0;
        repeat (10) begin
            {D, EN, SLn, SD} = $random; // Randomize inputs
            #10;
            $display("Flip-Flop Mode | D = %b, EN = %b, SLn = %b, SD = %b, Q =
%b", D, EN, SLn, SD, Q);
        end

        // Step 4: Latch Mode (LAT = 1)
        LAT = 1;
        repeat (10) begin
            {D, EN, SLn, SD} = $random; // Randomize inputs
            #10;
            $display("Latch Mode | D = %b, EN = %b, SLn = %b, SD = %b, Q = %b",
D, EN, SLn, SD, Q);
        end

        $finish;
    end
endmodule
```



	Msgs
D	1'hx
CLK	1'h0
EN	1'hx
ALn	1'h0
ADn	1'h0
SLn	1'hx
SD	1'hx
LAT	1'hx
Q	1'h0
Now	220000 ps
Cursor 1	433 ps

0 ps to 231 ns

```
# TEST: ALn Activated | Expected Q = 0, Actual Q = 0
# Flip-Flop Mode | D = 0, EN = 0, SLn = 0, SD = 1, Q = 0
# Flip-Flop Mode | D = 1, EN = 0, SLn = 0, SD = 1, Q = 0
# Flip-Flop Mode | D = 0, EN = 0, SLn = 1, SD = 1, Q = 0
# Flip-Flop Mode | D = 1, EN = 1, SLn = 0, SD = 1, Q = 1
# Flip-Flop Mode | D = 1, EN = 1, SLn = 0, SD = 1, Q = 1
# Flip-Flop Mode | D = 0, EN = 1, SLn = 0, SD = 1, Q = 1
# Flip-Flop Mode | D = 0, EN = 0, SLn = 1, SD = 0, Q = 1
# Flip-Flop Mode | D = 0, EN = 0, SLn = 0, SD = 1, Q = 1
# Flip-Flop Mode | D = 1, EN = 1, SLn = 0, SD = 1, Q = 1
# Flip-Flop Mode | D = 0, EN = 1, SLn = 1, SD = 0, Q = 0

# Latch Mode | D = 1, EN = 1, SLn = 0, SD = 1, Q = 1
# Latch Mode | D = 1, EN = 1, SLn = 0, SD = 1, Q = 1
# Latch Mode | D = 1, EN = 1, SLn = 0, SD = 0, Q = 0
# Latch Mode | D = 1, EN = 0, SLn = 0, SD = 1, Q = 0
# Latch Mode | D = 0, EN = 1, SLn = 1, SD = 0, Q = 0
# Latch Mode | D = 0, EN = 1, SLn = 0, SD = 1, Q = 1
# Latch Mode | D = 1, EN = 0, SLn = 1, SD = 0, Q = 1
# Latch Mode | D = 0, EN = 1, SLn = 0, SD = 1, Q = 1
# Latch Mode | D = 0, EN = 1, SLn = 1, SD = 1, Q = 1
# Latch Mode | D = 0, EN = 0, SLn = 1, SD = 0, Q = 1
```