Lab # 12

Shift Registers in Verilog

Objectives:

- Understand the concept and working principles of different types of shift registers.
- Implement SISO, SIPO, and Universal Shift Registers using Verilog.
- Write and verify testbenches for shift-register modules.
- Analyze the timing behavior of sequential circuits.
- Observe data shifting operations and parallel/serial data flow.
- Develop confidence in designing sequential building blocks for complex digital systems.

Introduction to Shift Registers

A shift register is a set of flip-flops connected in series and used to **store and move (shift)** data. Shift registers are sequential logic circuits used for storing and shifting digital data. They are constructed using a chain of flip-flops (typically D flip-flops), where each flip-flop stores one bit. At every active clock edge, the bits shift from one flip-flop to the next according to the mode of operation.

Shift registers are widely used in:

- Data conversion (serial-to-parallel, parallel-to-serial)
- Delay circuits
- Communication systems
- Counters
- Finite State Machines

- Digital signal processing

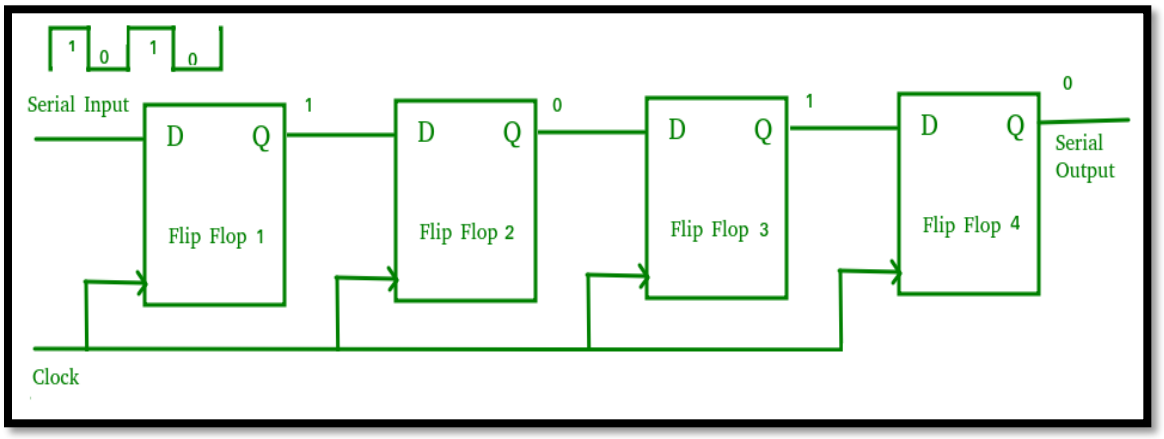
A shift register can operate in different modes depending on how data is loaded and shifted:

- Serial Input / Serial Output
- Serial Input / Parallel Output
- Universal Shift Register (can operate in multiple modes)

Types of Shift Registers

1. **Serial-In Serial-Out Shift Register (SISO)**

- The shift register, which allows serial input (one bit at a time through a single data line) and produces a serial output, is known as a Serial-In Serial-Out shift register. Since there is only one output, the data leaves the shift register one bit at a time in a serial pattern, thus the name Serial-In Serial-Out Shift Register.
- The logic circuit shown below illustrates a serial-in, serial-out shift register. The circuit consists of four D flip-flops, which are connected serially. All these flip-flops are synchronous with each other since the same clock signal is applied to each flip-flop.



Description

- Data enters one bit at a time (serial input).
- Data exists one bit at a time (serial output).
- Bits are shifted through the register with each clock cycle.

Applications

- Delay elements
- Serial data transmission

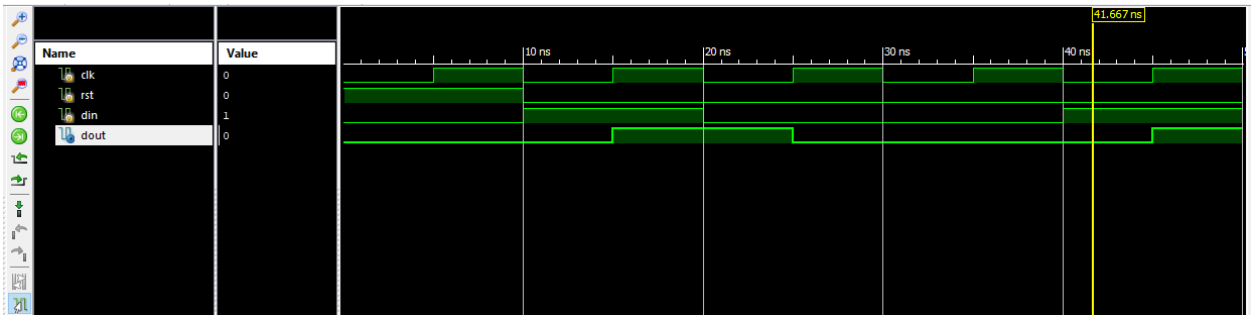
Verilog Code:

```
`timescale 1ns / 1ps
module SISO(input clk, input rst, input din, output dout);
reg [3:0] q;

always @(posedge clk or posedge rst) begin
if (rst)
q <= 4'b0000;
else
q[0] <= q[1];
q[1] <= q[2];
q[2] <= q[3];
q[3] <= din;
end

assign dout = q[3];
endmodule
```

Test Bench

```
`timescale 1ns / 1ps
module SISO_TB();
   reg clk, rst, din;
   wire dout;

   // Instantiate the DUT
   SISO uut(.clk(clk), .rst(rst), .din(din), .dout(dout));

   always #5 clk = ~clk;

   initial begin
      // Initialize signals
      clk = 0;
      rst = 1;
      din = 0;

      #10;
      rst = 0;

      din = 1;
      #10;

      din = 0;
      #10;

      din = 0;
      #10;

      din = 1;
      #10;

      $finish;
   end
endmodule
```

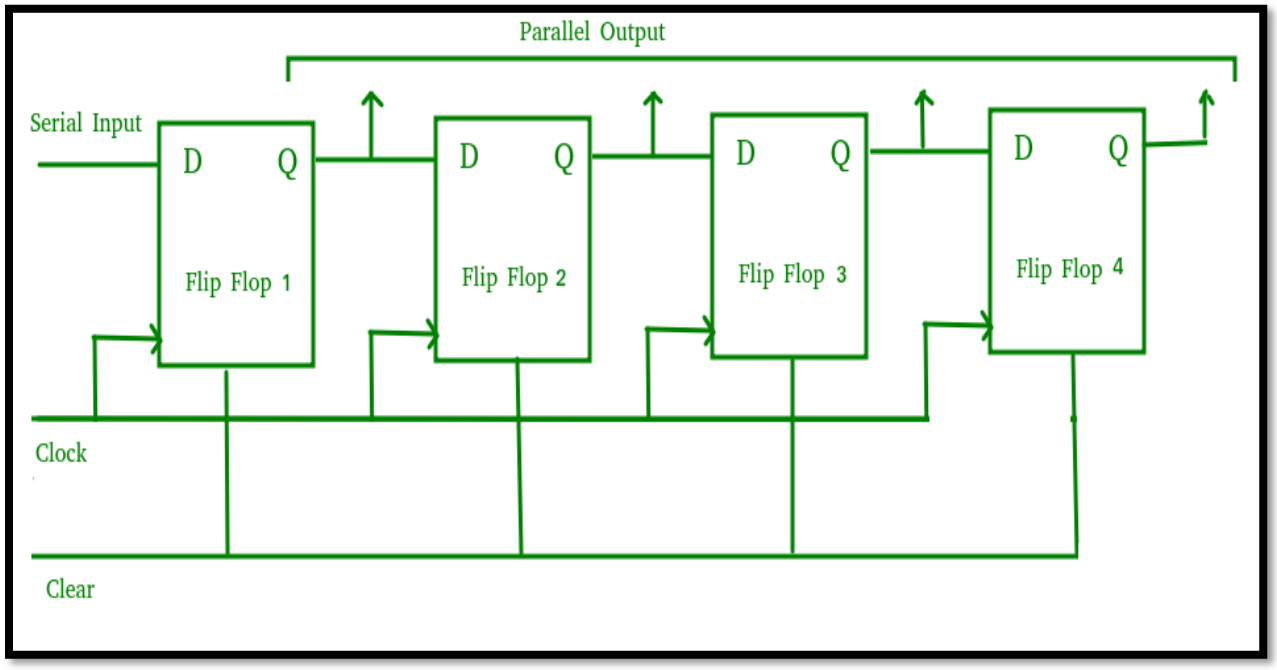
OUTPUT



2. Serial-In Parallel-Out shift Register (SIPO)

The shift register, which allows serial input (one bit at a time through a single data line) and produces a parallel output, is known as a Serial-In Parallel-Out (SIPO) shift register.

The logic circuit given below shows a serial-in-parallel-out shift register. The circuit consists of four D flip-flops, which are connected. The clear (CLR) signal is connected in addition to the clock signal to all 4 flip-flops to RESET them. The output of the first flip-flop is connected to the input of the next flip-flop and so on. All these flip-flops are synchronous with each other since the same clock signal is applied to each flip-flop.



Description

- Data enters serially.
- After several clock cycles, all bits can be read in parallel.

Applications

- Serial-to-parallel conversion
- Communication receivers

Verilog Code:

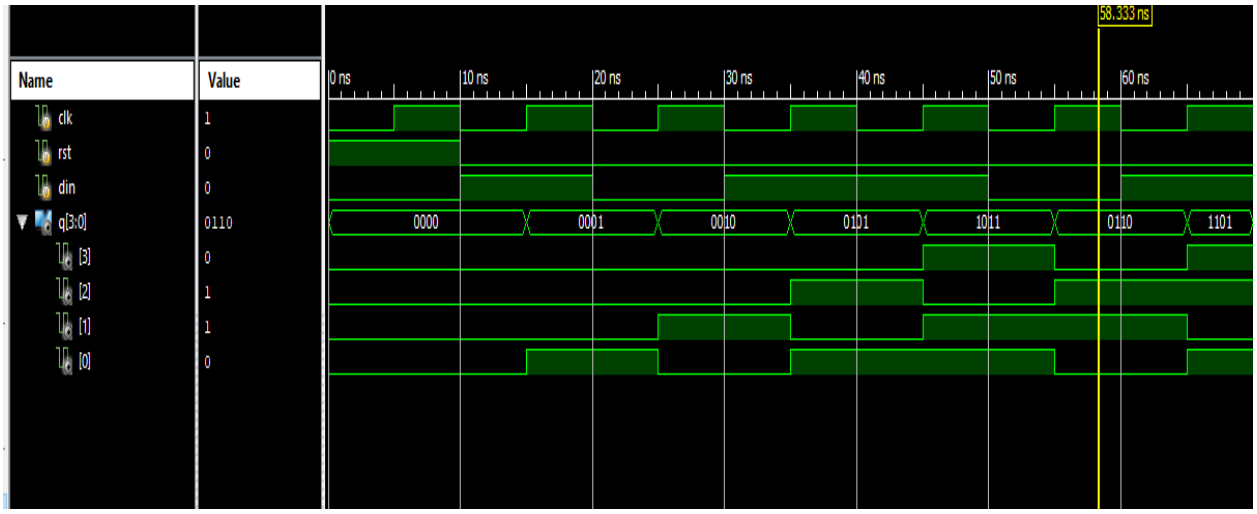
```
`timescale 1ns / 1ps
module SIPO (
    input clk,
    input rst,
    input din,
    output reg [3:0] q
);
always @(posedge clk or posedge rst) begin
    if (rst)
        q <= 4'b0000;
    else begin
        q[3] <= q[2];
        q[2] <= q[1];
        q[1] <= q[0];
        q[0] <= din;
    end
end
endmodule
```

Test Bench

```
`timescale 1ns / 1ps

module tb_SIPO;

    reg clk;
    reg rst;
    reg din;
    wire [3:0] q;

    // Instantiate DUT
    SIPO uut ( .clk(clk), .rst(rst),
        .din(din),
        .q(q) );
```

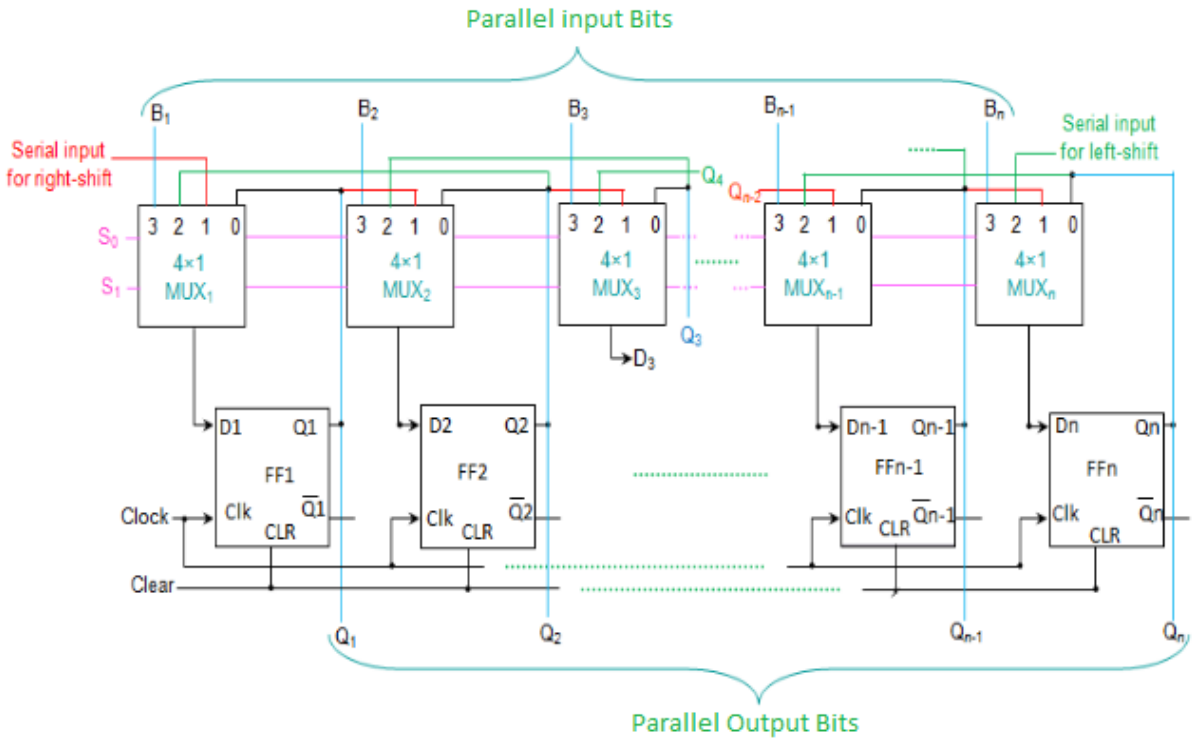
```
    initial begin
        clk = 0;
        forever #5 clk = ~clk;
    end

    initial begin

        rst = 1;
        din = 0;

        #10;
        rst = 0;

        din = 1;  #10;   // Bit 1
        din = 0;  #10;   // Bit 2
        din = 1;  #10;   // Bit 3
        din = 1;  #10;   // Bit 4
        din = 0;  #10;
        din = 1;  #10;

        $finish;

    end
endmodule
```

OUTPUT



Universal Shift Register:

A universal shift register is a general-purpose digital circuit that supports left and right shifting, parallel loading, and both serial and parallel data input/output, combining the features of unidirectional and bidirectional shift registers with the ability to hold data statically. It operates in three modes, Shift Left, Shift Right, and Parallel Load, based on control signals, and retains its current data when no control signal is active.



- Internally, it uses flip-flops to store bits and multiplexers to select the input source for each operation.
- Shift Left/Right operations move data by one position, replacing the vacant bit with serial input and discarding the shifted-out bit.

- Parallel Load allows simultaneous loading of data into all bits, while idle mode keeps the register unchanged.

The working of the Universal shift register depends on the inputs given to the select lines. The register operations performed for the various inputs of select lines are as follows

A typical Universal Shift Register uses **two mode control bits (S1, S0):**

S1	s0	Register operation
0	0	No changes
0	1	Shift right
1	0	Shift left
1	1	Parallel load

Advantages of Universal Shift Register

- **Multifunctionality:** left and right shifts, parallel load, and static hold can all be performed.
- **Flexibility:** can be used in many different digital systems.

Functions of a Universal Shift Register

- A typical USR supports **four main operations**:

  (1) Hold / No change

- The data remains the same inside the register.
  Useful when no new operation is needed.

  (2) Shift Left

  All bits move one position to the **left**:

q3 q2 q1 q0 → q2 q1 q0 din_left

The new bit entering from the right (LSB) is **din_left**, also called **serial left input**.

(3) Shift Right

All bits move one position to the **right**:

q3 q2 q1 q0 → din_right q3 q2 q1

The new bit entering from the left (MSB) is **din_right**, also called **serial right input**.

(4) Parallel Load

All bits are loaded at the **same time**:

q <= parallel_in;

This is called **PIPO loading**.

Verilog Code:

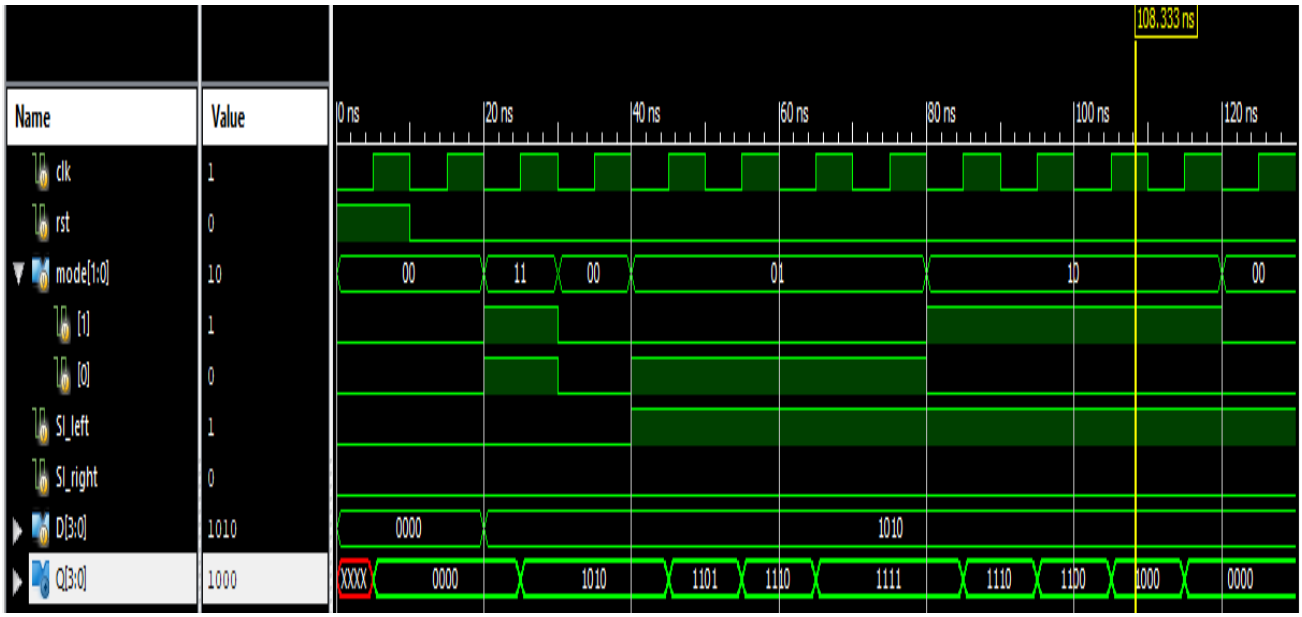
```
`timescale 1ns / 1ps
module universal_SR (
   input clk, rst, input [1:0] mode, input SI_left,
   input SI_right, input [3:0] D,
   output reg [3:0] Q);

   always @(posedge clk) begin
      if (rst) begin
         Q[3] <= 0;
         Q[2] <= 0;
         Q[1] <= 0;
         Q[0] <= 0;
      end
      else begin
         case(mode)
            2'b00: begin          // HOLD
               // Do nothing
            end

            2'b01: begin          // SHIFT RIGHT
               Q[3] <= SI_left;
               Q[2] <= Q[3];
               Q[1] <= Q[2];
               Q[0] <= Q[1];
            end

            2'b10: begin          // SHIFT LEFT
               Q[3] <= Q[2];
               Q[2] <= Q[1];
               Q[1] <= Q[0];
               Q[0] <= SI_right;
            end

            2'b11: begin          // PARALLEL LOAD
               Q[3] <= D[3];
               Q[2] <= D[2];
               Q[1] <= D[1];
               Q[0] <= D[0];
            end
         endcase
      end
   end

endmodule
```

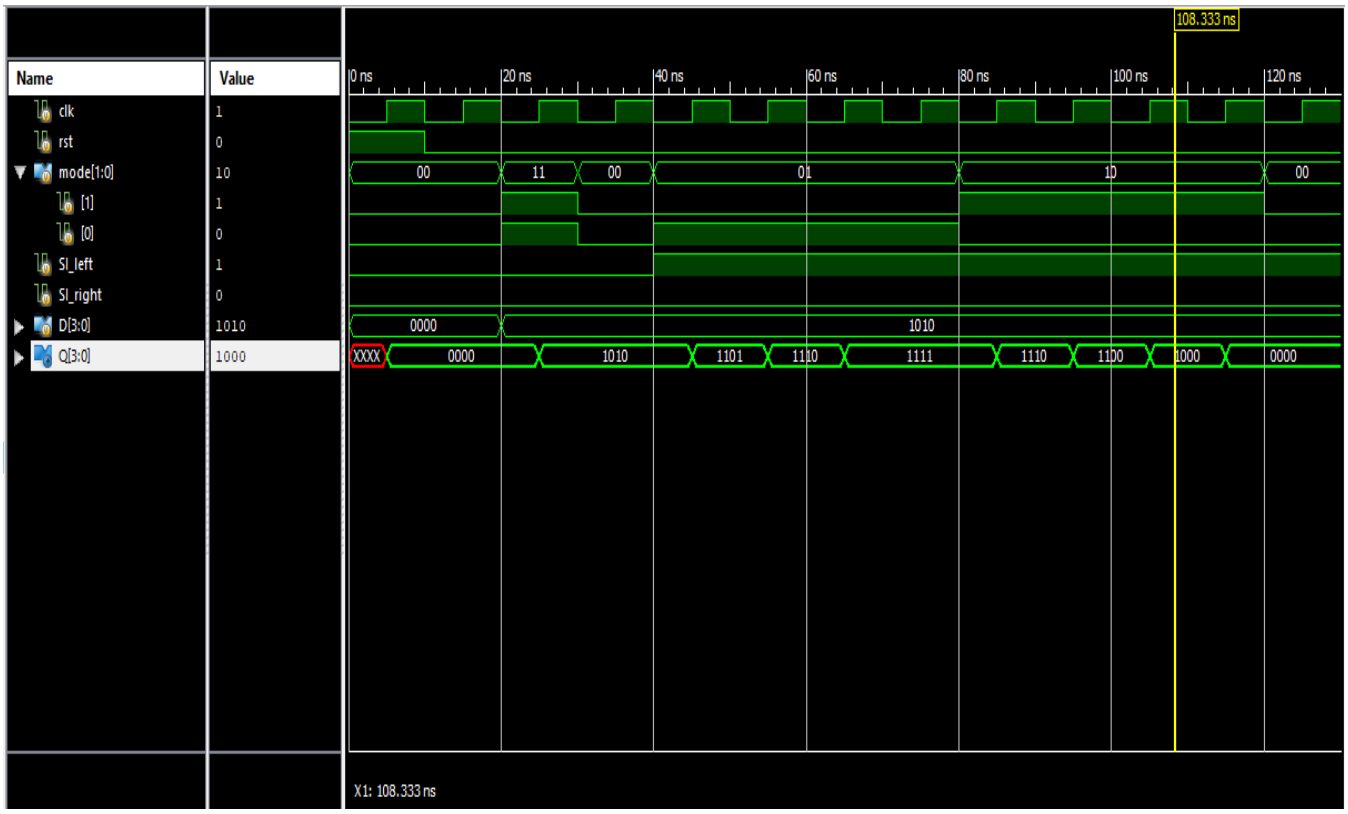
Test Bench:

```
`timescale 1ns / 1ps

module universal_SR_TB;
    reg clk;
    reg rst;
    reg [1:0] mode;
    reg SI_left;
    reg SI_right;
    reg [3:0] D;
    wire [3:0] Q;

    universal_SR uut (
        .clk(clk),
        .rst(rst),
        .mode(mode),
        .SI_left(SI_left),
        .SI_right(SI_right),
        .D(D),
        .Q(Q)
    );

    initial begin
        clk = 0;
        forever #5 clk = ~clk;
    end

    initial begin
        // Initialize inputs
        rst = 1;
        mode = 2'b00;
        SI_left = 0;
        SI_right = 0;
        D = 4'b0000;

        #10;
        rst = 0;

        #10;
        mode = 2'b11;
        D = 4'b1010;
        #10;

        mode = 2'b00;
        #10;
```

```
        mode = 2'b01;
        SI_left = 1;
        #40;

        mode = 2'b10;
        SI_right = 0;
        #40;
        mode = 2'b00;
        #10;
        $finish;
    end
endmodule
```

OUTPUT




Name
Value
clk
1
rst
0
mode[1:0]
10
[1]
1
[0]
0
SI_left
1
SI_right
0
D[3:0]
1010
Q[3:0]
1000
108.333 ns
0 ns
20 ns
40 ns
60 ns
80 ns
100 ns
120 ns
00
11
00
01
10
00
0000
1010
XXXX
0000
1010
1101
1110
1111
1110
1100
1000
0000
X1: 108.333 ns

CE221L- Digital Logic Design Lab

RUBRIC SHEET

Criteria	Excellent (Full Marks)	Good (Partial Marks)	Needs Improvement (Low Marks)	Max Marks
Code Writing (Verilog) **(Apply)** (4)	Writes fully optimized and error-free Verilog code for combinational and sequential circuits independently.	Writes mostly correct code with minor errors, requiring limited guidance.	The code contains major errors or is incomplete, requiring significant help to correct.	**/4**
Test Bench **(Construct)** (3)	Develops accurate and complete test benches to simulate and verify circuit functionality without errors.	Creates test benches with minor issues that require small corrections.	Fails to create a proper test bench or is unable to verify functionality.	**/3**
FPGA Implementation **(Construct & Apply)** (3)	Successfully implements the design on the FPGA trainer board, demonstrating correct and reliable hardware performance.	Implement the design with minor issues that require troubleshooting.	Unable to correctly implement the design on the FPGA, or the hardware fails.	**/3**
Design Objectives **(Apply & Construct)** (3)	Clearly states and applies theoretical concepts to fully achieve all given design objectives.	States and applies concepts partially, achieving some design objectives.	Fails to state or apply theoretical concepts, achieving none or minimal objectives.	**/3**
Viva **(Collaborate)** (2)	Communicates ideas clearly and confidently, demonstrating full understanding of concepts and the design process.	Communicates with some hesitation or minor conceptual gaps.	Struggles to explain concepts or answer questions correctly.	**/2**
Total				**/15**

Name: _____ **Instructor Signature:** _____

Reg #: _____ **Date:** _____