**ASSIGNMENT-9**

**S24CSEU1869- AKRITAH SAHU**

**Ans 1.**

**Shift Registers:**

A Shift Register is a type of sequential logic circuit used in digital electronics. It is composed of flip-flops connected in a series, where the output of one flip-flop becomes the input of the next. The primary function of a shift register is to store data and shift it either to the left or right when a clock signal is applied.

Types of Shift Registers:

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

* Data is input serially (bit by bit) and output serially.
* Used for simple data transfer over a single line.

2. **Serial-In Parallel-Out (SIPO)**

* Data is input serially, but all bits are made available in parallel.
* Used when data needs to be processed in parallel after serial transmission.

3. **Parallel-In Serial-Out (PISO)**

* All bits are loaded simultaneously (parallel), and then output serially.
* Ideal for converting parallel data to serial form (e.g., for communication).

4. **Parallel-In Parallel-Out (PIPO)**

* All bits are input and output in parallel.
* Used for fast data transfer and temporary data storage.

Applications of Shift Registers:

1. **Data Storage**
   * Temporarily holds data in flip-flops.
   * Acts as a memory buffer.
2. **Data Transfer**
   * Transfers data between components using fewer lines (serial communication).
3. **Data Manipulation**
   * Used in operations like multiplication, division, or delay of digital signals.
4. **Serial-to-Parallel and Parallel-to-Serial Conversion**
   * Essential in communication systems to interface between serial and parallel buses.
5. **Counters**
   * Implement ring counters or Johnson counters for timing or sequencing applications.

**Ans 2.**

module register\_4bit (

input wire clk, // Clock input

input wire reset, // Asynchronous reset

input wire load, // Load control signal

input wire [3:0] data\_in, // 4-bit parallel data input

output reg [3:0] data\_out // 4-bit data output

);

always @(posedge clk or posedge reset) begin

if (reset)

data\_out <= 4'b0000; // Clear register on reset

else if (load)

data\_out <= data\_in; // Load data on rising clock edge if load is high

// else retain previous value

end

endmodule

module tb\_register\_4bit\_parallel\_load;

// Testbench signals

reg clk;

reg reset;

reg load;

reg [3:0] data\_in;

wire [3:0] data\_out;

// Instantiate the Unit Under Test (UUT)

register\_4bit\_parallel\_load uut (

.clk(clk),

.reset(reset),

.load(load),

.data\_in(data\_in),

.data\_out(data\_out)

);

// Clock generation

initial begin

clk = 0;

forever #5 clk = ~clk; // 10ns clock period

end

// Stimulus

initial begin

// Initialize inputs

reset = 1;

load = 0;

data\_in = 4'b0000;

// Apply reset

#10 reset = 0;

// Load 1010

#10 data\_in = 4'b1010; load = 1;

#10 load = 0;

// Load 1100 (load is low, so should not change)

#10 data\_in = 4'b1100;

// Load 1111 (load high, should update)

#10 load = 1;

#10 load = 0;

// Reset again

#10 reset = 1;

#10 reset = 0;

#20 $finish;

end

// Monitor outputs

initial begin

$monitor("Time=%0t | reset=%b | load=%b | data\_in=%b | data\_out=%b",

$time, reset, load, data\_in, data\_out);

end

// Dumping waveforms

initial begin

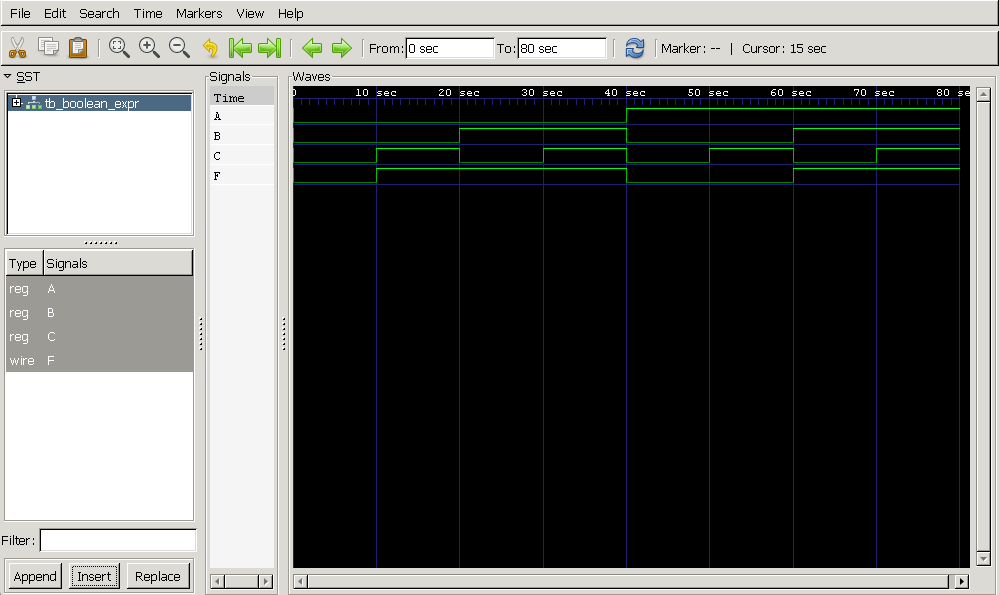
$dumpfile("register\_4bit\_wave.vcd"); // Create dump file

$dumpvars(0, tb\_register\_4bit\_parallel\_load); // Dump all variables in this module

end

endmodule

//Waveform



**Ans 3.**

module pipo\_register (

input wire clk, // Clock signal

input wire rst, // Synchronous reset

input wire [3:0] data\_in, // 4-bit parallel data input

output reg [3:0] data\_out // 4-bit parallel data output

);

always @(posedge clk) begin

if (rst)

data\_out <= 4'b0000; // Reset the register to 0

else

data\_out <= data\_in; // Load input data on rising edge

end

endmodule

//TestBench

`timescale 1ns / 1ps

module pipo\_register\_tb;

// Testbench signals

reg clk;

reg rst;

reg [3:0] data\_in;

wire [3:0] data\_out;

// Instantiate the PIPO register

pipo\_register uut (

.clk(clk),

.rst(rst),

.data\_in(data\_in),

.data\_out(data\_out)

);

// Clock generation: 10ns period

initial begin

clk = 0;

forever #5 clk = ~clk; // Toggle every 5ns

end

// Test sequence

initial begin

// Create dumpfile for waveform

$dumpfile("pipo\_register\_tb.vcd"); // VCD file name

$dumpvars(0, pipo\_register\_tb); // Dump all variables in this module

// Initialize inputs

rst = 1;

data\_in = 4'b0000;

// Apply reset

#10;

rst = 0;

// Apply data inputs

#10 data\_in = 4'b1010;

#10 data\_in = 4'b1100;

#10 data\_in = 4'b1111;

// Apply reset again

#10 rst = 1;

#10 rst = 0;

// More inputs after reset

#10 data\_in = 4'b0011;

#10 data\_in = 4'b0001;

// Finish simulation

#20 $finish;

end

// Monitor signals

initial begin

$monitor("Time=%0t | rst=%b | data\_in=%b | data\_out=%b", $time, rst, data\_in, data\_out);

end

endmodule

//Waveform

