📘 **Day 23 Topic: Verilog codes for SIPO Register Using Cadence (Xcelium)**

SIPO (Serial-In Parallel-Out) Register

A SIPO register is a type of shift register where data enters serially (one bit at a time) and comes out in parallel (all bits together).

It uses a series of flip-flops connected in sequence and a common clock.
Each clock pulse shifts the input bit into the next flip-flop.
After all bits are entered, the data is available simultaneously at all outputs.

Key Points

- Input: Serial (one bit at a time).
- Output: Parallel (all bits at once).
- Clock: Common for all flip-flops.
- Storage Capacity: Equal to the number of flip-flops.

Example

If the serial input data is 1, 0, 1, 1, after four clock pulses the register holds 1 0 1 1, and all bits appear together at the parallel outputs.

🔹 **1.1) Design Code**

sipo.v

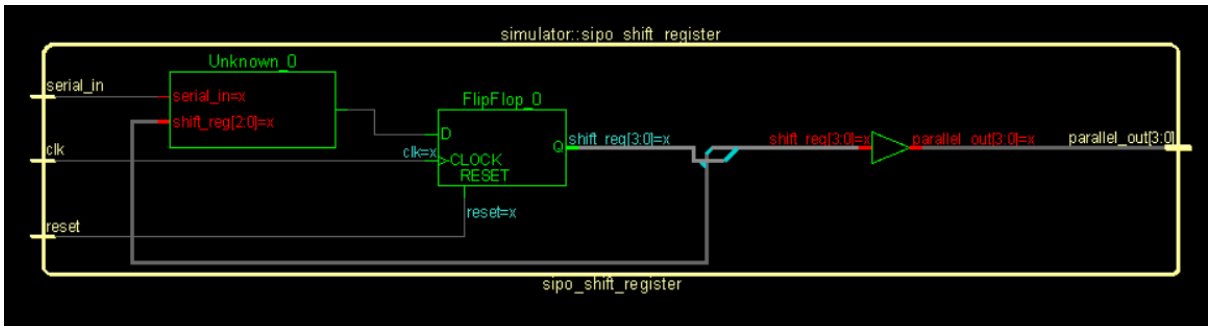
```
// Module: Serial In Parallel Out (SIPO) Shift Register

module sipo_shift_register (
    input clk,            // clock signal
    input reset,          // asynchronous reset
    input serial_in,      // serial data input
    output [3:0] parallel_out // 4-bit parallel output
);
    reg [3:0] shift_reg;  // internal 4-bit register

    // Shift operation on each rising edge of clock
    always @(posedge clk or posedge reset) begin
        if (reset)
            shift_reg <= 4'b0000;                  // clear register on reset
        else
            shift_reg <= {shift_reg[2:0], serial_in}; // shift left and insert serial bit
    end

    assign parallel_out = shift_reg;  // output entire register
endmodule
```

🔹 **1.2) Test Bench Code**

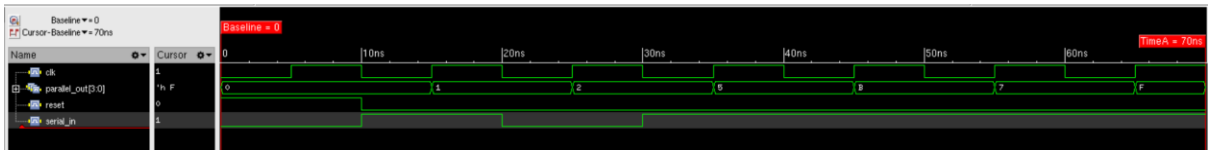
tb.v

```
// Testbench: SIPO Shift Register

module tb_sipo_shift_register;
    reg clk;
    reg reset;
    reg serial_in;
    wire [3:0] parallel_out;

    // Instantiate DUT
    sipo_shift_register uut (
        .clk(clk),
        .reset(reset),
        .serial_in(serial_in),
        .parallel_out(parallel_out)
    );

    // Clock generation (10 ns period)
    always #5 clk = ~clk;

    initial begin
        // Initialize
        clk = 0;
        reset = 1;
        serial_in = 0;
        #10 reset = 0;

        // Apply serial input bits
        serial_in = 1; #10;   // bit 1
        serial_in = 0; #10;   // bit 2
        serial_in = 1; #10;   // bit 3
        serial_in = 1; #10;   // bit 4

        // Observe parallel output
        #20;
        $finish;
    end
endmodule
```

1.3) Schematic



1.4) Wave Forms



Applications

- **Serial-to-Parallel conversion (used in data communication).**
- **Microprocessor systems to receive serial data and process it in parallel form.**
- **Data storage and temporary registers.**