📘 **Day 25 Topic: Verilog codes for PISO Register Using Cadence (Xcelium)**

PISO Register (Parallel In Serial Out) – Simple Description

A PISO register takes parallel data (many bits at once) and sends it one bit at a time as serial output with each clock pulse.

- Parallel In: All bits are loaded together.
- Serial Out: Bits come out one by one.
- Control: A Load signal is used —
    - When Load = 1, data is loaded in parallel.
    - When Load = 0, data shifts out serially on each clock.

✅ Use: Converts parallel data into serial form for data transmission.

🔹 **1.1) Design Code**

piso.v

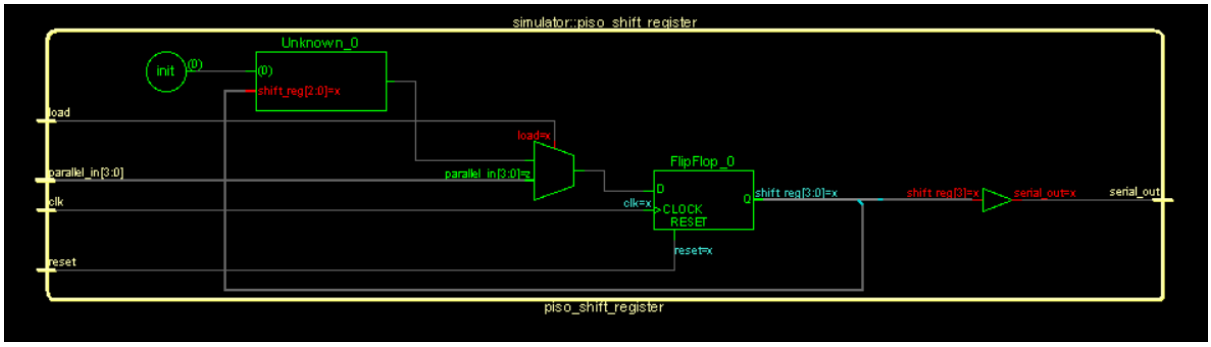
```
// Module: Parallel In Serial Out (PISO) Shift Register

module piso_shift_register (
    input clk,                // clock signal
    input reset,              // asynchronous reset
    input load,               // control signal: 1 = load, 0 = shift
    input [3:0] parallel_in, // 4-bit parallel data input
    output serial_out         // serial data output
);
    reg [3:0] shift_reg;      // internal 4-bit register

    always @(posedge clk or posedge reset) begin
        if (reset)
            shift_reg <= 4'b0000;            // clear on reset
        else if (load)
            shift_reg <= parallel_in;        // load all bits in parallel
        else
            shift_reg <= {shift_reg[2:0], 1'b0}; // shift left, LSB fills with 0
    end

    assign serial_out = shift_reg[3];        // MSB is shifted out serially
endmodule
```

🔹 **1.2) Test Bench Code**

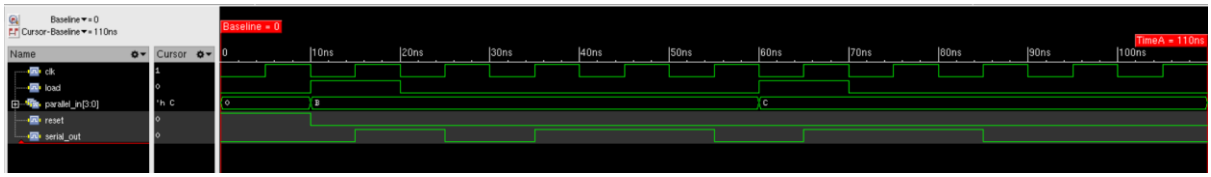
tb.v

```
// Testbench: PISO Shift Register
module tb_piso_shift_register;
    reg clk;
    reg reset;
    reg load;
    reg [3:0] parallel_in;
    wire serial_out;
// Instantiate DUT (Design Under Test)
    piso_shift_register uut (
        .clk(clk),
        .reset(reset),
        .load(load),
        .parallel_in(parallel_in),
        .serial_out(serial_out)
    );
// Clock generation (10 ns period)
    always #5 clk = ~clk;
    initial begin
        // Initialize
        clk = 0;
        reset = 1;
        load = 0;
        parallel_in = 4'b0000;
        #10 reset = 0;
        // Load data 1011
        load = 1;
        parallel_in = 4'b1011;
        #10;
        // Start shifting
        load = 0;
        #40;
        // Load new data 1100
        load = 1;
        parallel_in = 4'b1100;
        #10;
        // Shift again
        load = 0;
        #40;
        // End simulation
        $finish;
    end
endmodule
```

1.3) Schematic



1.4) Wave Forms



Applications

- **Data transmission: To send multiple bits serially over a single line (e.g., UART, SPI).**
- **Microprocessor systems: To reduce the number of data lines.**
- **Communication systems: For converting parallel sensor data into serial form.**
- **Signal processing: For serial input to shift-based circuits.**