📘 **Day 24 Topic: Verilog codes for PIPO Register Using Cadence (Xcelium)**

PIPO (Parallel-In Parallel-Out) Register

A PIPO register is a type of shift register where data is loaded and taken out in parallel, meaning all bits enter and exit at the same time.

Each flip-flop in the register receives its input bit simultaneously when a clock pulse is applied.
After the clock edge, all stored bits are available together at the outputs.

Key Points

- Input: Parallel (all bits enter at once).
- Output: Parallel (all bits appear at once).
- Clock: Common for all flip-flops.
- Storage Capacity: Equal to the number of flip-flops.

Example

If the inputs are D3 D2 D1 D0 = 1 0 1 1, all these bits are loaded together on one clock pulse, and the outputs Q3 Q2 Q1 Q0 immediately show 1 0 1 1.

🔹 **1.1) Design Code**

pipo.v

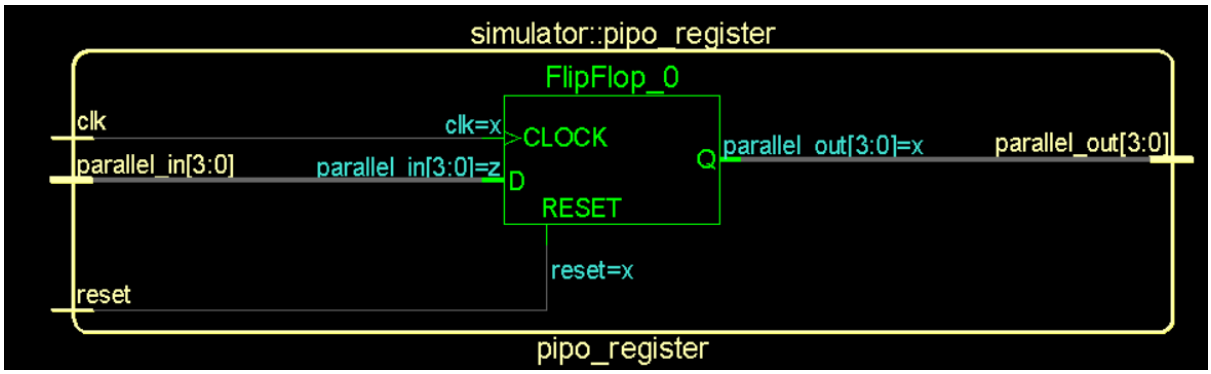
```
// Module: Parallel In Parallel Out (PIPO) Register

module pipo_register (
    input clk,              // clock signal
    input reset,            // asynchronous reset
    input [3:0] parallel_in, // 4-bit parallel data input
    output reg [3:0] parallel_out // 4-bit parallel data output
);

    // Load all bits at once on rising edge of clock
    always @(posedge clk or posedge reset) begin
        if (reset)
            parallel_out <= 4'b0000;       // clear register on
        else
            parallel_out <= parallel_in;   // load all bits in p
    end
endmodule
```

- **1.2) Test Bench Code**

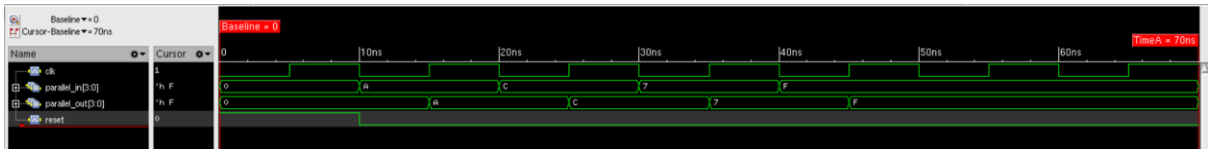
tb.v

```
// Testbench: PIPO Register

module tb_pipo_register;
    reg clk;
    reg reset;
    reg [3:0] parallel_in;
    wire [3:0] parallel_out;

    // Instantiate DUT (Design Under Test)
    pipo_register uut (
        .clk(clk),
        .reset(reset),
        .parallel_in(parallel_in),
        .parallel_out(parallel_out)
    );

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

    initial begin
        // Initialize
        clk = 0;
        reset = 1;
        parallel_in = 4'b0000;
        #10 reset = 0;

        // Apply parallel data
        parallel_in = 4'b1010; #10;
        parallel_in = 4'b1100; #10;
        parallel_in = 4'b0111; #10;
        parallel_in = 4'b1111; #10;

        // End simulation
        #20;
        $finish;
    end
endmodule
```

- **1.3) Schematic**



- **1.4) Wave Forms**



Applications

- **Temporary data storage in microprocessors and digital systems.**
- **Fast data transfer between devices.**
- **Used in register files and ALUs for parallel operations.**