📘 Day 26 Topic: Verilog codes for Universal Register Using Cadence (Xcelium)

Universal Shift Register

A Universal Shift Register is a type of register that can store data and shift it in both directions — left and right.
It is called *"universal"* because it can perform multiple operations:

1. Hold data (no change)
2. Shift left
3. Shift right
4. Parallel load (load all bits at once)

Key Points

- Can accept input serially or in parallel.
- Can output data serially or in parallel.
- Controlled by select lines (usually two) that decide the operation mode.
- Uses flip-flops (mostly D flip-flops) and multiplexers for direction control.

Operations Table Example

Control Inputs	Operation
00	Hold (no change)
01	Shift right
10	Shift left
11	Parallel load

🔹 1.1) Design Code

uni_reg.v

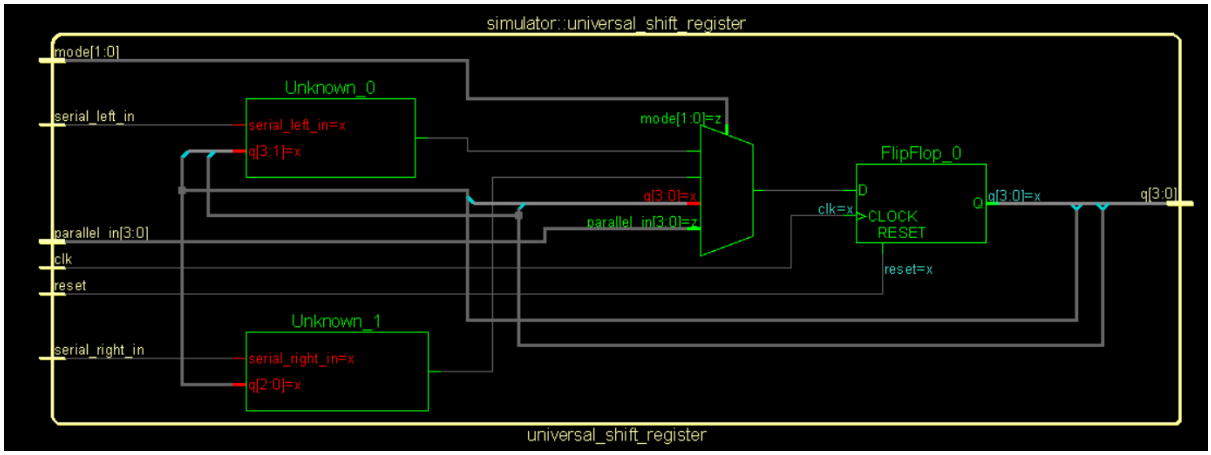
```
// Module: Universal Shift Register (4-bit)

module universal_shift_register (
    input clk,                // clock signal
    input reset,              // asynchronous reset
    input [1:0] mode,         // mode control input
    input [3:0] parallel_in,  // 4-bit parallel input
    input serial_left_in,     // serial input for left shift
    input serial_right_in,    // serial input for right shift
    output reg [3:0] q        // 4-bit register output
);

    always @(posedge clk or posedge reset) begin
        if (reset)
            q <= 4'b0000; // clear register
        else begin
            case (mode)
                2'b00: q <= q; // Hold
                2'b01: q <= {serial_left_in, q[3:1]};  // Shift Right
                2'b10: q <= {q[2:0], serial_right_in}; // Shift Left
                2'b11: q <= parallel_in; // Parallel Load
                default: q <= q;
            endcase
        end
    end
endmodule
```

- **1.2) Test Bench Code**

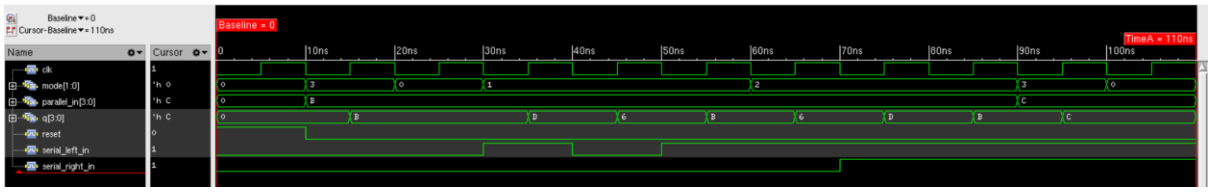
tb.v

```
// Testbench: Universal Shift Register
module tb_universal_shift_register;
    reg clk;
    reg reset;
    reg [1:0] mode;
    reg [3:0] parallel_in;
    reg serial_left_in;
    reg serial_right_in;
    wire [3:0] q;
    // Instantiate DUT
    universal_shift_register uut (
        .clk(clk),
        .reset(reset),
        .mode(mode),
        .parallel_in(parallel_in),
        .serial_left_in(serial_left_in),
        .serial_right_in(serial_right_in),
        .q(q));
    // Clock generation (10ns period)
    always #5 clk = ~clk;
    initial begin
        // Initialize
        clk = 0;
        reset = 1;
        mode = 2'b00;
        parallel_in = 4'b0000;
        serial_left_in = 0;
        serial_right_in = 0;
        #10 reset = 0;
        mode = 2'b11; parallel_in = 4'b1011; #10;
        mode = 2'b00; #10;
        mode = 2'b01; serial_left_in = 1; #10;
        serial_left_in = 0; #10;
        serial_left_in = 1; #10;
        mode = 2'b10; serial_right_in = 0; #10;
        serial_right_in = 1; #10;
        serial_right_in = 1; #10;
        mode = 2'b11; parallel_in = 4'b1100; #10;
        mode = 2'b00; #10;
        $finish;
    end
endmodule
```

1.3) Schematic



1.4) Wave Forms



Applications

- **Data transfer between systems.**
- **Data conversion (serial ↔ parallel).**
- **Arithmetic and logic operations.**
- **Temporary data storage in CPUs.**