NAME: VEDANSHI THAKKAR
ID NUMBER:21EL094
ASSIGNMENT OF 50 DAYS: IN THIS 25 DAYS
RTL CODES OF EACH
SUBJECT: DIGITAL SYSTEM DESIGN

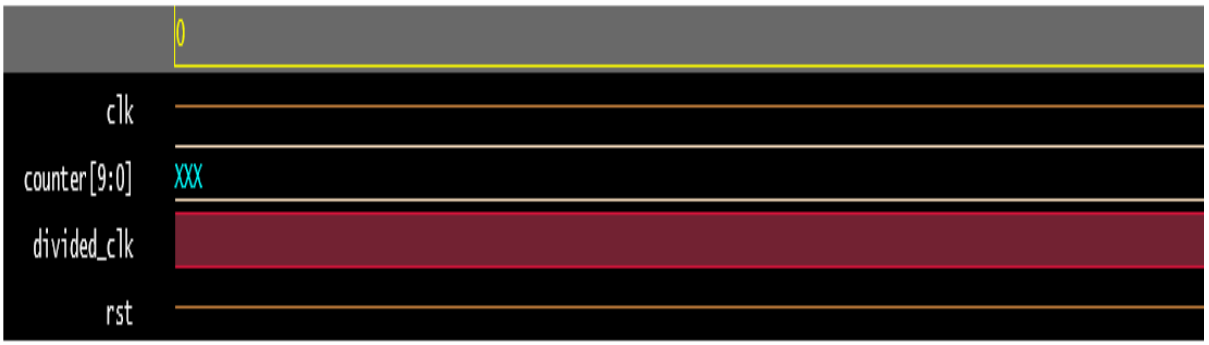
DAY 1 : CLOCK DIVIDER

```
module clock_divider (
        input wire clk,        // Input clock
        input wire rst,        // Reset signal
        output wire divided_clk // Divided clock output
);

reg [n-1:0] counter;         // Counter to keep track of divisions
parameter n = 10;            // Division factor (change as needed)

always @(posedge clk or posedge rst) begin
        if (rst) begin
        counter <= 0;
        end else begin
        if (counter == n-1) begin
        counter <= 0;
        divided_clk <= ~divided_clk; // Toggle the output clock
        end else begin
        counter <= counter + 1;
        end
        end
end

assign divided_clk = 0; // Initial value of divided clock

endmodule
```

Test banch

```
module testbench;
        // Inputs and outputs declarations

        // Instantiate your module (e.g., clock_divider)

        // Clock generation
```

```
        // Reset generation

        // Add your stimulus and assertions here

        // Waveform dumping commands
        initial begin
        $dumpfile("dump.vcd");
        $dumpvars;
        end

        // Simulation termination
        initial begin
        $finish;
        end
endmodule
```



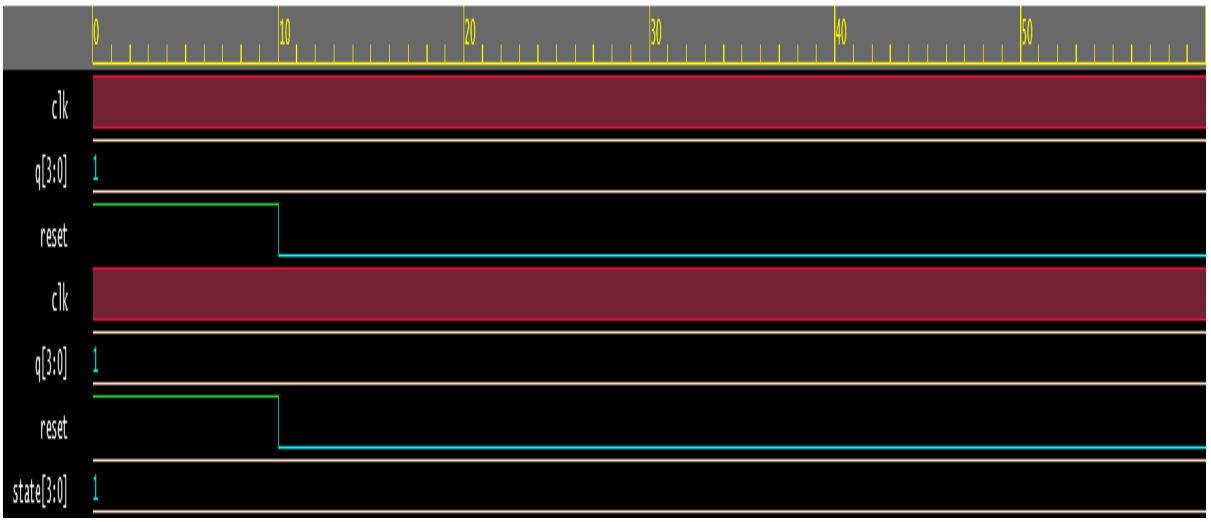
DAY 2 : JOHNSON COUNTER

```
module johnson_counter (
        input wire clk,        // Clock input
        input wire reset,     // Reset input
        output wire [3:0] q          // Johnson counter outputs
);

reg [3:0] state;               // Johnson counter state

always @(posedge clk or posedge reset) begin
        if (reset) begin
        state <= 4'b0001;   // Initialize the state to 0001 when reset is active
        end else begin
        // Shift the state to the right (or left) by one position
        state <= {state[2:0], state[3]};
        end
end

assign q = state;              // Output is the current state

endmodule
```

Test banch

```
module testbench;

        // Inputs and outputs
        reg clk;        // Clock input
        reg reset;     // Reset input
        wire [3:0] q;          // Johnson counter outputs

        // Instantiate the Johnson counter module
        johnson_counter jc (
        .clk(clk),
```

```
        .reset(reset),
        .q(q)
        );

        // Clock generation
        always begin
        #5 clk = ~clk;  // Toggle the clock every 5 time units
        end

        // Reset generation
        initial begin
        reset = 1;      // Assert reset initially
        #10 reset = 0;  // De-assert reset after 10 time units
        #50 $finish;  // Finish simulation after 50 time units
        end

        // Simulation termination
        initial begin
        $dumpfile("dump.vcd");
        $dumpvars;
        $display("Starting Johnson counter simulation...");
        $monitor("Time=%t, q=%b", $time, q);
        end

endmodule
```



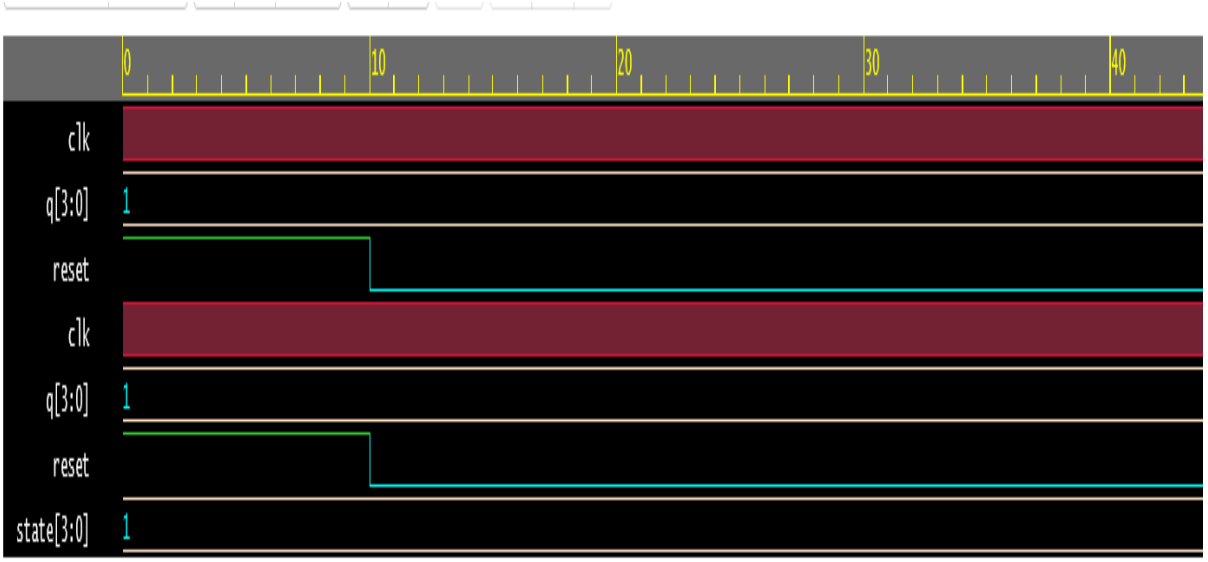
DAY 3: RING COUNTER

```
module ring_counter (
        input wire clk,        // Clock input
        input wire reset,     // Reset input
        output wire [3:0] q           // Ring counter outputs
);

reg [3:0] state;                // Ring counter state

always @(posedge clk or posedge reset) begin
        if (reset) begin
        state <= 4'b0001;   // Initialize the state to 0001 when reset is active
        end else begin
        // Shift the state to the right by one position
        state <= {state[0], state[3:1]};
        end
end

assign q = state;               // Output is the current state

endmodule
```

Test banch

```
module testbench;

        // Inputs and outputs
        reg clk;        // Clock input
        reg reset;     // Reset input
        wire [3:0] q;           // Ring counter outputs

        // Instantiate the ring counter module
        ring_counter rc (
        .clk(clk),
        .reset(reset),
        .q(q)
```

```
);

// Clock generation
always begin
#5 clk = ~clk;  // Toggle the clock every 5 time units
end

// Reset generation
initial begin
reset = 1;      // Assert reset initially
#10 reset = 0;  // De-assert reset after 10 time units
#50 $finish;  // Finish simulation after 50 time units
end

// Simulation termination
initial begin
$dumpfile("dump.vcd");
$dumpvars;
$display("Starting ring counter simulation...");
$monitor("Time=%t, q=%b", $time, q);
end

endmodule
```



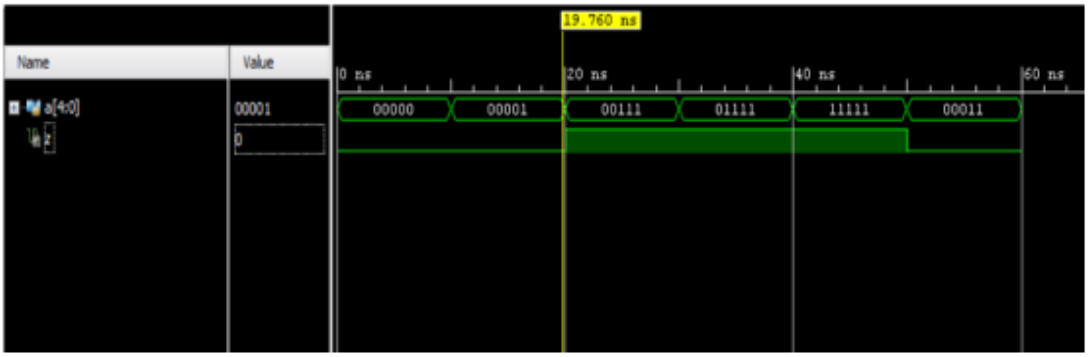
DAY 4:module majority_circuit

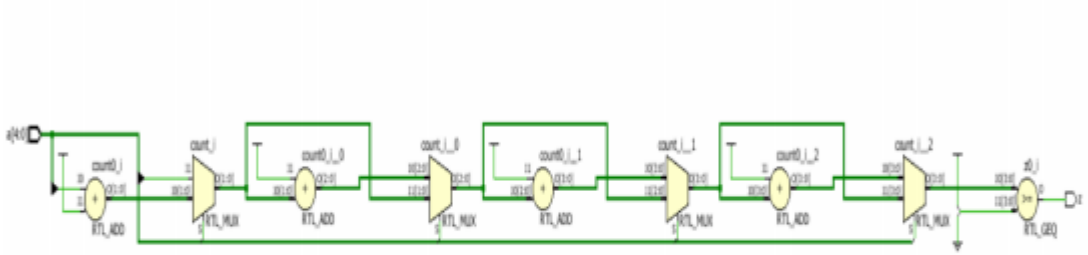
```
 (
        input wire a, // Input signal A
        input wire b, // Input signal B
        input wire c, // Input signal C
        input wire d, // Input signal D
        input wire e, // Input signal E
        output wire y         // Output signal Y (majority)
);

assign y = (a & b) | (a & c) | (b & c) | (a & d) | (b & d) | (c & d) | (a & e) | (b & e) | (c
& e) | (d & e);

endmodule
```

Test bench

```
module testbench;

        // Inputs and outputs
        reg a, b, c, d, e;   // Input signals A, B, C, D, E
        wire y;          // Output signal Y

        // Instantiate the majority gate module
        majority_gate mg (
        .a(a),
        .b(b),
        .c(c),
        .d(d),
        .e(e),
        .y(y)
        );

        // Test cases and simulation code here

        // Waveform dumping commands
        initial begin
        $dumpfile("dump.vcd");
        $dumpvars;

        // Add your test cases and simulation code here
```

```
        $finish;
        end

endmodule
```





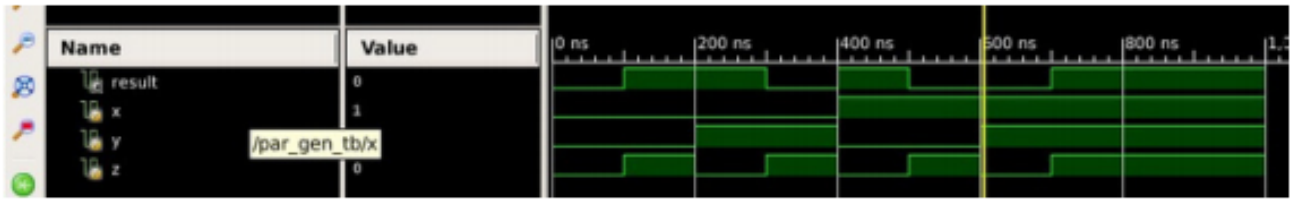
DAY 5: PARITY GENERATOR

```
module even_parity_generator (
        input wire [N-1:0] data, // Input data bits (N bits)
        output wire parity           // Parity bit (even parity)
);

wire [N-1:0] xor_result;
assign xor_result = ^data; // XOR all data bits

assign parity = ~^xor_result; // Invert the XOR result to get even parity

endmodule
```

Test bench

```
module test_even_parity_generator;

 // Parameters
 parameter N = 8; // Number of data bits

 // Signals
 reg [N-1:0] data;            // Input data bits
 wire parity;          // Parity bit (even parity)

 // Instantiate the even_parity_generator module
 even_parity_generator uut (
        .data(data),
        .parity(parity)
 );

 // Clock generation
 reg clk = 0;
 always #5 clk = ~clk;

 // Test stimulus generation
 initial begin
        // Initialize data
        data = 8'b10101010; // Example data with even parity
```

```
        // Apply test cases
        $display("Input Data = %b", data);
        $display("Expected Parity = 1"); // Expected even parity for the given
data

        // Stimulate the module
        #10; // Wait for a few clock cycles

        // Check if the generated parity is correct
        if (parity == 1'b1) begin
        $display("Generated Parity = 1 (Even Parity) - Test PASSED");
        end else begin
        $display("Generated Parity = 0 (Odd Parity) - Test FAILED");
        end

        // Finish simulation
        $finish;
 end

endmodule
```



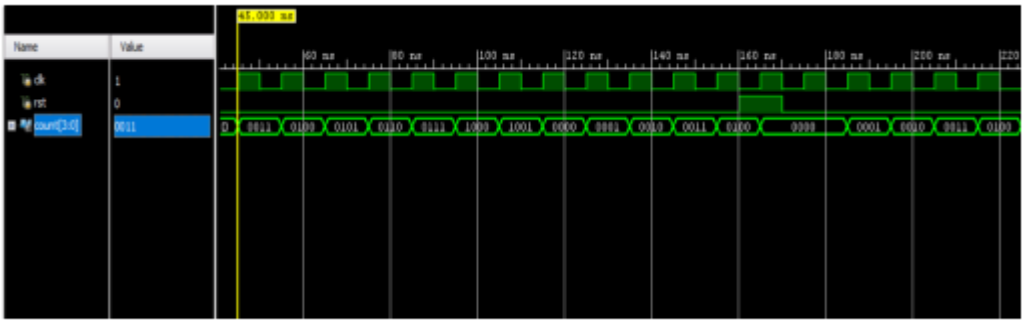
DAY 6: BINARY TO ONE HOT ENCODER

```
module binary_to_one_hot_encoder (
        input wire [3:0] binary_input, // 4-bit binary input
        output wire [15:0] one_hot_output // 16-bit one-hot output
);

assign one_hot_output[0] = (binary_input == 4'b0001);
assign one_hot_output[1] = (binary_input == 4'b0010);
assign one_hot_output[2] = (binary_input == 4'b0100);
assign one_hot_output[3] = (binary_input == 4'b1000);

endmodule
```

Test bench

```
module test_binary_to_one_hot_encoder;

 // Signals
 reg [3:0] binary_input;          // 4-bit binary input
 wire [15:0] one_hot_output;    // 16-bit one-hot output

 // Instantiate the binary_to_one_hot_encoder module
 binary_to_one_hot_encoder uut (
        .binary_input(binary_input),
        .one_hot_output(one_hot_output)
 );

 // Test stimulus generation
```

```
  initial begin
        $display("Testing Binary-to-One-Hot Encoder");

        // Test Case 1: Input = 4'b0001
        binary_input = 4'b0001;
        #10; // Wait for a few time units

        $display("Input: 4'b%b, Expected Output: 16'b0000_0000_0001_0000",
binary_input);
        $display("Output: 16'b%b", one_hot_output);

        if (one_hot_output === 16'b0000_0000_0001_0000) begin
        $display("Test Case 1 PASSED");
        end else begin
        $display("Test Case 1 FAILED");
        end

        // Add more test cases as needed...

        // Finish simulation
        $finish;
  end

endmodule
```



DAY 7: 4-BIT BCD SYNCHRONOUS COUNTER

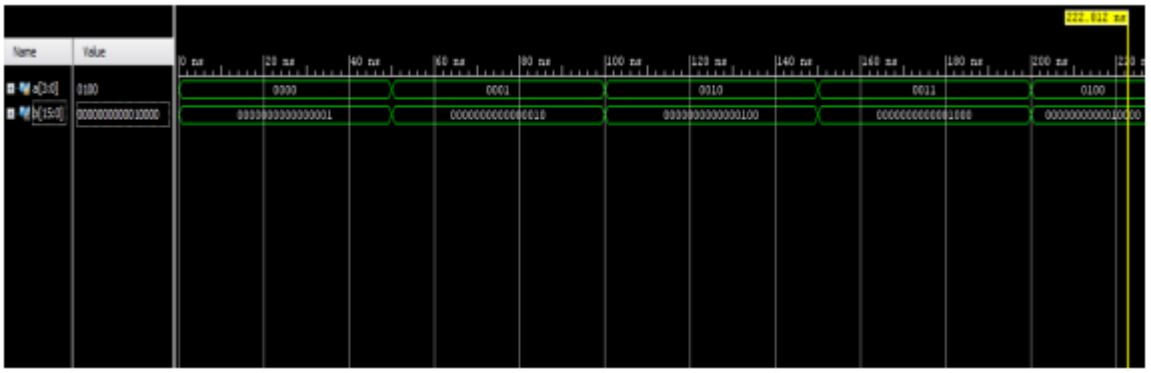
```
module bcd_counter (
        input wire clk,       // Clock input
        input wire reset,     // Reset input
        output wire [3:0] bcd   // BCD output (4 bits)
);

reg [3:0] count;              // 4-bit BCD counter

always @(posedge clk or posedge reset) begin
        if (reset) begin
        count <= 4'b0000;  // Initialize the counter to 0 on reset
        end else begin
        if (count == 4'b1001) begin
        count <= 4'b0000;  // Reset to 0 when reaching 9 (1001 in BCD)
        end else begin
        count <= count + 1;  // Increment the counter
        end
        end
end

assign bcd = count;        // Output is the BCD count

endmodule
```

Test bench

```
module test_bcd_counter;
```

```
// Signals
reg clk = 0;          // Clock input
reg reset = 0;        // Reset input
wire [3:0] bcd_output;  // BCD output (4 bits)

// Instantiate the bcd_counter module
bcd_counter uut (
        .clk(clk),
        .reset(reset),
        .bcd(bcd_output)
);

// Clock generation
always begin
        #5 clk = ~clk; // Toggle the clock every 5 time units
end

// Test stimulus generation
initial begin
        $display("Testing BCD Counter");

        // Test Case 1: Normal counting without reset
        reset = 0;
        #30; // Allow for some clock cycles
        $display("Test Case 1: No Reset (Counting)");
        check_output(10); // Check if the counter counts correctly up to 9

        // Test Case 2: Reset after reaching 9
        reset = 0;
        #30; // Allow for some clock cycles
        $display("Test Case 2: Reset After Reaching 9");
        check_output(4); // Check if the counter counts correctly up to 4

        // Test Case 3: Reset immediately after reset signal
        reset = 1;
        #10; // Wait for a few clock cycles with reset
        reset = 0;
        #30; // Allow for some clock cycles
        $display("Test Case 3: Immediate Reset");
        check_output(0); // Check if the counter starts from 0

        // Finish simulation
        $finish;
end
```

```
  // Function to check BCD output
  task check_output(int expected_value);
        if (bcd_output === expected_value) begin
        $display("BCD Output: %d (Expected: %d) - PASSED", bcd_output,
expected_value);
        end else begin
        $display("BCD Output: %d (Expected: %d) - FAILED", bcd_output,
expected_value);
        end
  endtask

endmodule
```



DAY 8: 4-BIT CARRY LOOKAHEAD ADDER

```
module cl_adder_4bit (
        input wire [3:0] A,       // 4-bit input A
        input wire [3:0] B,       // 4-bit input B
        input wire Cin,           // Carry input
        output wire [3:0] Sum, // 4-bit sum output
        output wire Cout          // Carry output
);

wire [3:0] G; // Generate signals
wire [3:0] P; // Propagate signals

assign G = A & B; // Generate signal
assign P = A | B; // Propagate signal

assign Sum[0] = A[0] ^ B[0] ^ Cin; // Sum bit 0
assign Sum[1] = P[1] ^ G[0] ^ Cin; // Sum bit 1
assign Sum[2] = P[2] ^ G[1] ^ G[0]; // Sum bit 2
assign Sum[3] = P[3] ^ G[2] ^ G[1]; // Sum bit 3

assign Cout = G[3] | (G[2] & P[3]) | (G[1] & P[2]) | (G[0] & P[1] & Cin); // Carry out

endmodule
```

Test bench

```
module test_cl_adder_4bit;

  // Signals
  reg [3:0] A;            // 4-bit input A
  reg [3:0] B;            // 4-bit input B
  reg Cin;                // Carry input
  wire [3:0] Sum;         // 4-bit sum output
  wire Cout;              // Carry output

  // Instantiate the cl_adder_4bit module
  cl_adder_4bit uut (
        .A(A),
        .B(B),
        .Cin(Cin),
        .Sum(Sum),
        .Cout(Cout)
```

```
  );

  // Test stimulus generation
  initial begin
        $display("Testing 4-bit Carry-Lookahead Adder");

        // Test Case 1: A=4, B=3, Cin=0
        A = 4'b0100;
        B = 4'b0011;
        Cin = 0;
        #10; // Allow for some time
        $display("Test Case 1: A=4, B=3, Cin=0");
        check_output(7, 0); // Expected: Sum=7, Cout=0

        // Test Case 2: A=9, B=7, Cin=1
        A = 4'b1001;
        B = 4'b0111;
        Cin = 1;
        #10; // Allow for some time
        $display("Test Case 2: A=9, B=7, Cin=1");
        check_output(1, 1); // Expected: Sum=1, Cout=1

        // Add more test cases as needed...

        // Finish simulation
        $finish;
  end

  // Function to check Sum and Cout outputs
  task check_output(int expected_sum, int expected_cout);
        if (Sum === expected_sum && Cout === expected_cout) begin
        $display("Sum: %d (Expected: %d), Cout: %d (Expected: %d) - PASSED", Sum,
expected_sum, Cout, expected_cout);
        end else begin
        $display("Sum: %d (Expected: %d), Cout: %d (Expected: %d) - FAILED", Sum,
expected_sum, Cout, expected_cout);
        end
  endtask

endmodule
```

DAY 9: N-BIT COMPARATOR

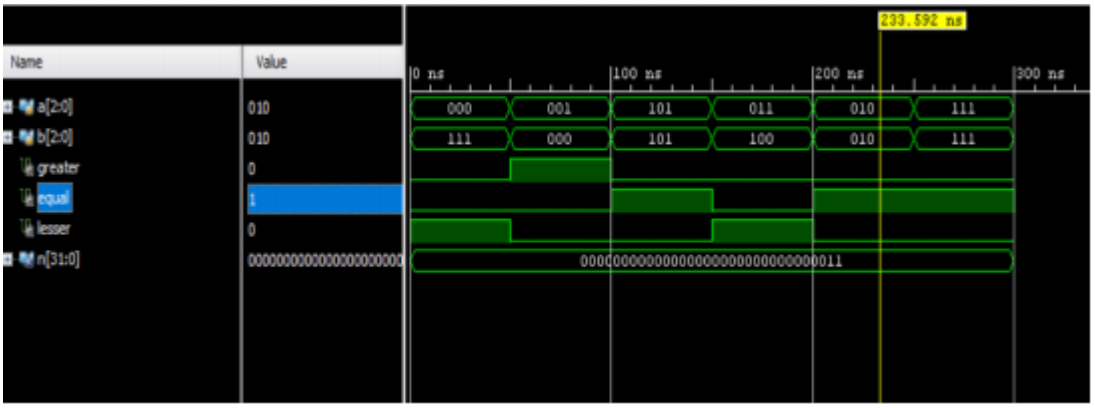
module n_bit_comparator (

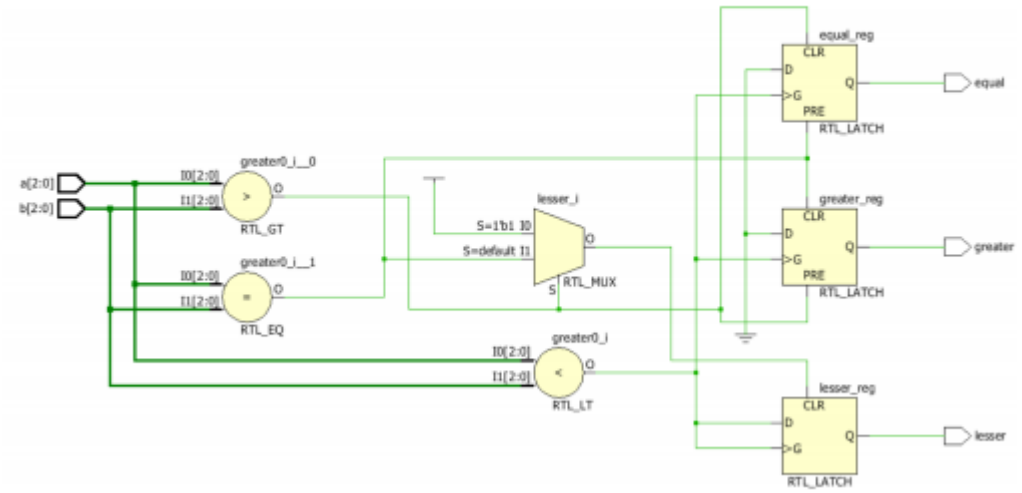
```
        input wire [N-1:0] A,   // N-bit input A
        input wire [N-1:0] B,   // N-bit input B
        output wire greater,        // Output: 1 if A > B, 0 otherwise
        output wire less,    // Output: 1 if A < B, 0 otherwise
        output wire equal           // Output: 1 if A = B, 0 otherwise
);

assign greater = (A > B);
assign less = (A < B);
assign equal = (A == B);

endmodule
```

Test bench

```
module test_n_bit_comparator;

 // Parameters
 parameter N = 4; // Number of bits

 // Signals
 reg [N-1:0] A;       // N-bit input A
 reg [N-1:0] B;       // N-bit input B
 wire greater;        // Output: 1 if A > B, 0 otherwise
 wire less;           // Output: 1 if A < B, 0 otherwise
 wire equal;          // Output: 1 if A = B, 0 otherwise

 // Instantiate the n_bit_comparator module
 n_bit_comparator uut (
        .A(A),
        .B(B),
        .greater(greater),
        .less(less),
        .equal(equal)
 );

 // Test stimulus generation
 initial begin
        $display("Testing N-bit Comparator");

        // Test Case 1: A = B (A equal to B)
        A = 4'b1010;
```

```
        B = 4'b1010;
        #10; // Allow for some time
        $display("Test Case 1: A = B");
        check_output(1, 0, 1); // Expected: greater=0, less=0, equal=1

        // Test Case 2: A < B (A less than B)
        A = 4'b0101;
        B = 4'b1010;
        #10; // Allow for some time
        $display("Test Case 2: A < B");
        check_output(0, 1, 0); // Expected: greater=0, less=1, equal=0

        // Test Case 3: A > B (A greater than B)
        A = 4'b1100;
        B = 4'b1010;
        #10; // Allow for some time
        $display("Test Case 3: A > B");
        check_output(1, 0, 0); // Expected: greater=1, less=0, equal=0

        // Add more test cases as needed...

        // Finish simulation
        $finish;
 end

 // Function to check output signals
 task check_output(int expected_greater, int expected_less, int
expected_equal);
        if (greater === expected_greater && less === expected_less && equal
=== expected_equal) begin
        $display("Greater: %b (Expected: %b), Less: %b (Expected: %b), Equal:
%b (Expected: %b) - PASSED",
                greater, expected_greater, less, expected_less, equal,
expected_equal);
        end else begin
        $display("Greater: %b (Expected: %b), Less: %b (Expected: %b), Equal:
%b (Expected: %b) - FAILED",
                greater, expected_greater, less, expected_less, equal,
expected_equal);
        end
 endtask

endmodule
```


233.592 ns
Name
Value
0 ns
100 ns
200 ns
300 ns
a[2:0]
010
000
001
101
011
010
111
b[2:0]
010
111
000
101
100
010
111
greater
0
equal
1
lesser
0
n[31:0]
00000000000000000000000000000011


equal_reg
CLR
D
G
PRE
Q
RTL_LATCH
equal
greater_reg
greater
lesser_reg
lesser
a[2:0]
b[2:0]
greater0_i__0
I0[2:0]
I1[2:0]
RTL_GT
greater0_i__1
RTL_EQ
lesser_i
S=1'b1 I0
S=default I1
S
RTL_MUX
greater0_i
RTL_LT

DAY 10: SERIAL IN SERIAL OUT SHIFT REGISTER

1
```
module siso_shift_register (
        input wire clk,        // Clock input
        input wire reset,      // Reset input
        input wire serial_in,  // Serial input
        output wire serial_out // Serial output
);

reg [3:0] shift_register; // 4-bit shift register

always @(posedge clk or posedge reset) begin
        if (reset) begin
        shift_register <= 4'b0000; // Initialize the register to 0 on reset
        end else begin
        shift_register <= {shift_register[2:0], serial_in}; // Shift data in
        end
end

assign serial_out = shift_register[3]; // Output the last bit

endmodule
```

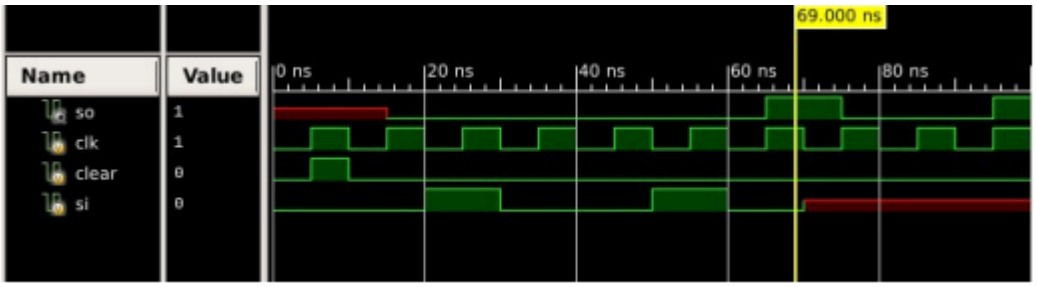
2
```
module siso_shift_register (
        input wire clk,        // Clock input
        input wire reset,      // Reset input
        input wire serial_in,   // Serial input
        output wire serial_out  // Serial output
);

reg [3:0] register_data;    // 4-bit shift register

always @(posedge clk or posedge reset) begin
        if (reset) begin
        register_data <= 4'b0000; // Initialize the register to 0 on reset
        end else begin
        // Shift data to the right, MSB to LSB
        register_data <= {serial_in, register_data[3:1]};
        end
end
```

```
assign serial_out = register_data[3]; // Serial output from MSB

endmodule
```

Test bench

```
module test_siso_shift_register;

  // Signals
  reg clk = 0;          // Clock input
  reg reset = 0;        // Reset input
  reg serial_in = 0;            // Serial input
  wire serial_out;              // Serial output

  // Instantiate the siso_shift_register module
  siso_shift_register uut (
        .clk(clk),
        .reset(reset),
        .serial_in(serial_in),
        .serial_out(serial_out)
  );

  // Clock generation
  always begin
        #5 clk = ~clk; // Toggle the clock every 5 time units
  end

  // Test stimulus generation
  initial begin
        $display("Testing 4-bit SISO Shift Register");

        // Test Case 1: Shift in '1010'
        reset = 0;
        serial_in = 1;
        #10; // Allow for some clock cycles
        $display("Test Case 1: Shift in '1010'");
        check_output(0); // Expected serial_out=0

        // Test Case 2: Shift in '0011' (reset at the beginning)
        reset = 1;
```

```
        serial_in = 0;
        #10; // Wait for a few clock cycles with reset
        reset = 0;
        serial_in = 1;
        #10; // Allow for some clock cycles
        $display("Test Case 2: Shift in '0011'");
        check_output(1); // Expected serial_out=1

        // Add more test cases as needed...

        // Finish simulation
        $finish;
  end

  // Function to check serial_out
  task check_output(int expected_value);
        if (serial_out === expected_value) begin
        $display("Serial Out: %b (Expected: %b) - PASSED", serial_out,
expected_value);
        end else begin
        $display("Serial Out: %b (Expected: %b) - FAILED", serial_out,
expected_value);
        end
  endtask

endmodule
```



DAY 11: SERIAL IN PARALLEL OUT SHIFT REGISTER

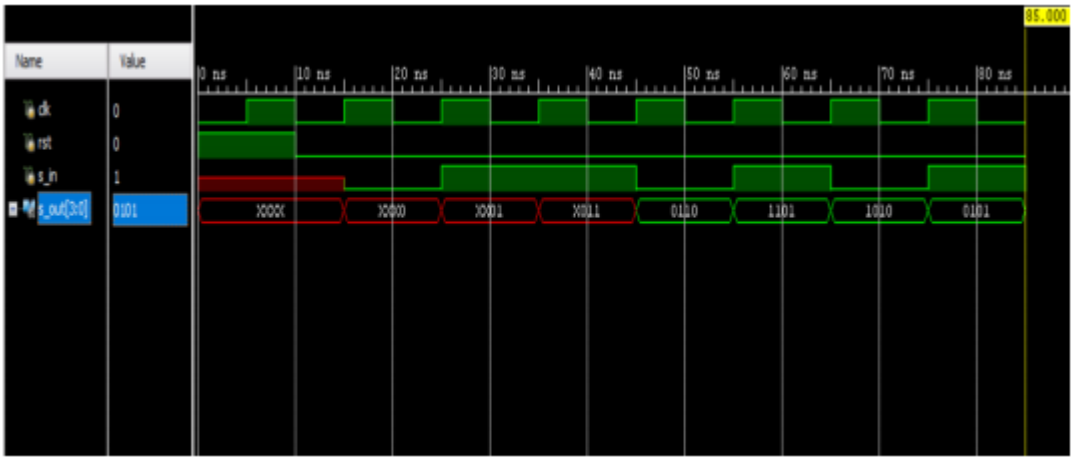
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity SIPO_Shift_Register is
        Port (
        CLK   : in  STD_LOGIC;   -- Clock input
        RESET  : in  STD_LOGIC;          -- Reset input
        SER   : in  STD_LOGIC;   -- Serial data input
        Q       : out STD_LOGIC_VECTOR(3 downto 0)  -- 4-bit parallel output
        );
end SIPO_Shift_Register;

architecture Behavioral of SIPO_Shift_Register is
        signal shift_reg : STD_LOGIC_VECTOR(3 downto 0) := "0000";
begin
        process(CLK, RESET)
        begin
        if RESET = '1' then
        shift_reg <= "0000";  -- Reset the shift register
        elsif rising_edge(CLK) then
        shift_reg <= SER & shift_reg(3 downto 1);  -- Shift in the serial input
        end if;
        end process;

        Q <= shift_reg;  -- Output the parallel data
end Behavioral;
```

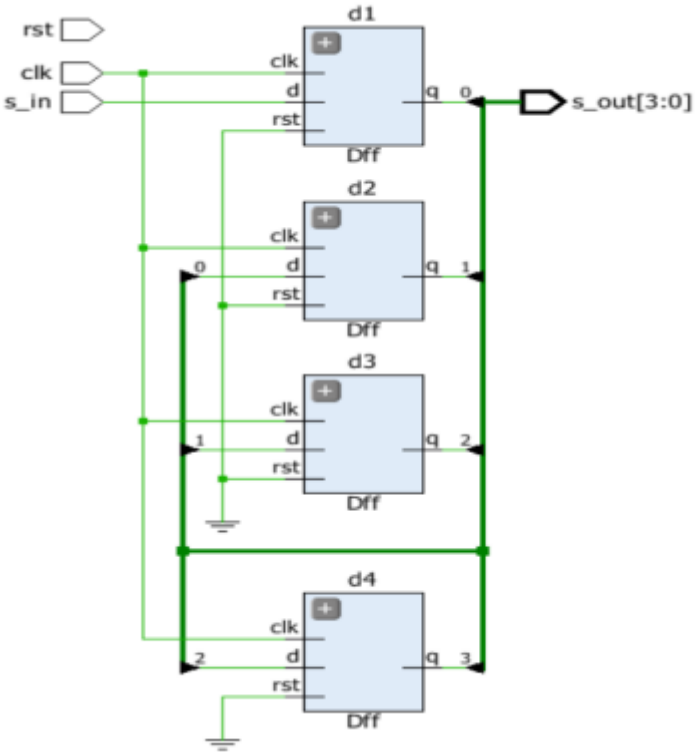
Test bench

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
```

```
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity test_SIPO_Shift_Register is
end test_SIPO_Shift_Register;

architecture testbench of test_SIPO_Shift_Register is
  -- Constants
  constant CLK_PERIOD: time := 10 ns;  -- Clock period (adjust as needed)

  -- Signals
  signal CLK: std_logic := '0';     -- Clock input
  signal RESET: std_logic := '0';           -- Reset input
  signal SER: std_logic := '0';     -- Serial data input
  signal Q: std_logic_vector(3 downto 0); -- 4-bit parallel output

  -- Instantiate the SIPO_Shift_Register module
  component SIPO_Shift_Register
        Port (
        CLK : in  STD_LOGIC;
        RESET : in  STD_LOGIC;
        SER : in  STD_LOGIC;
        Q : out STD_LOGIC_VECTOR(3 downto 0)
        );
  end component;

  -- Clock generation process
  process
  begin
        while now < 1000 ns loop  -- Simulate for 1000 ns (adjust as needed)
        CLK <= '0';
        wait for CLK_PERIOD / 2;
        CLK <= '1';
        wait for CLK_PERIOD / 2;
        end loop;
        wait;
  end process;

  -- Stimulus process
  process
  begin
        RESET <= '1'; -- Assert reset initially
        SER <= '0';   -- Set initial serial input
```

```
        wait for 20 ns; -- Wait for a few clock cycles

        RESET <= '0'; -- Deassert reset

        -- Test Case 1: Shift in '1010' serial input
        SER <= '1';
        wait for CLK_PERIOD; -- Wait for one clock cycle
        SER <= '0';
        wait for CLK_PERIOD; -- Wait for one clock cycle
        SER <= '1';
        wait for CLK_PERIOD; -- Wait for one clock cycle
        SER <= '0';
        wait for CLK_PERIOD; -- Wait for one clock cycle

        -- Test Case 2: Shift in '1101' serial input
        SER <= '1';
        wait for CLK_PERIOD; -- Wait for one clock cycle
        SER <= '1';
        wait for CLK_PERIOD; -- Wait for one clock cycle
        SER <= '0';
        wait for CLK_PERIOD; -- Wait for one clock cycle
        SER <= '1';
        wait for CLK_PERIOD; -- Wait for one clock cycle

        -- Add more test cases as needed...

        wait for 100 ns; -- Allow some additional time

        report "Simulation completed" severity note;
        wait;
    end process;

    -- Instantiate the SIPO_Shift_Register module
    SIPO_Shift_Register_inst : SIPO_Shift_Register
        port map (
        CLK => CLK,
        RESET => RESET,
        SER => SER,
        Q => Q
        );

end testbench;
```


Name
Value
0 ns
10 ns
20 ns
30 ns
40 ns
50 ns
60 ns
70 ns
80 ns
85.000
clk
0
rst
0
s_in
1
s_out[3:0]
0101
XXXX
X000
X001
X011
0110
1101
1010
0101


rst
clk
s_in
d1
clk
d
rst
q
0
Dff
s_out[3:0]
d2
clk
d
rst
q
1
Dff
0
d3
clk
d
rst
q
2
Dff
1
d4
clk
d
rst
q
3
Dff
2

DAY 12: PARALLEL IN PARALLEL OUT REGISTER

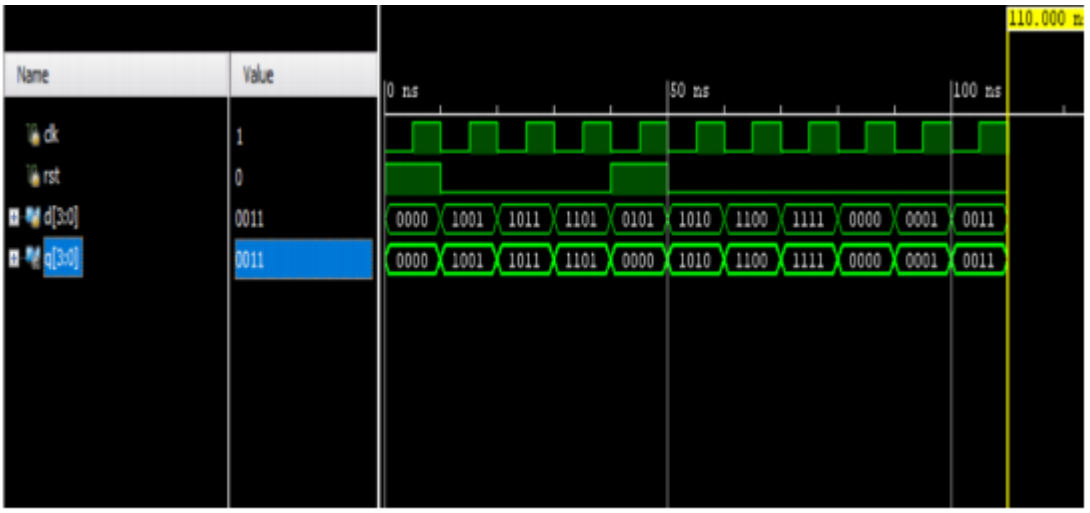
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity PIPO_Register is
        Port (
        CLK   : in  STD_LOGIC;            -- Clock input
        RESET  : in  STD_LOGIC;                  -- Reset input
        D       : in  STD_LOGIC_VECTOR(3 downto 0);  -- 4-bit parallel data input
        Q       : out STD_LOGIC_VECTOR(3 downto 0)  -- 4-bit parallel data
output
        );
end PIPO_Register;

architecture Behavioral of PIPO_Register is
        signal reg_data : STD_LOGIC_VECTOR(3 downto 0) := "0000";
begin
        process(CLK, RESET)
        begin
        if RESET = '1' then
        reg_data <= "0000";  -- Reset the register
        elsif rising_edge(CLK) then
        reg_data <= D;       -- Load parallel data on clock edge
        end if;
        end process;

        Q <= reg_data;  -- Output the parallel data
end Behavioral;
```

Test bench

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity test_PIPO_Register is
end test_PIPO_Register;

architecture testbench of test_PIPO_Register is
  -- Constants
  constant CLK_PERIOD: time := 10 ns;  -- Clock period (adjust as needed)

  -- Signals
```

```
signal CLK: std_logic := '0';     -- Clock input
signal RESET: std_logic := '0';           -- Reset input
signal D: std_logic_vector(3 downto 0) := "0000"; -- Parallel data input
signal Q: std_logic_vector(3 downto 0);      -- Parallel data output

-- Instantiate the PIPO_Register module
component PIPO_Register
        Port (
        CLK : in  STD_LOGIC;
        RESET : in  STD_LOGIC;
        D : in  STD_LOGIC_VECTOR(3 downto 0);
        Q : out STD_LOGIC_VECTOR(3 downto 0)
        );
end component;

-- Clock generation process
process
begin
        while now < 1000 ns loop  -- Simulate for 1000 ns (adjust as needed)
        CLK <= '0';
        wait for CLK_PERIOD / 2;
        CLK <= '1';
        wait for CLK_PERIOD / 2;
        end loop;
        wait;
end process;

-- Stimulus process
process
begin
        RESET <= '1'; -- Assert reset initially
        D <= "0000";  -- Set initial parallel data

        wait for 20 ns; -- Wait for a few clock cycles

        RESET <= '0'; -- Deassert reset

        -- Test Case 1: Load parallel data '1010'
        D <= "1010";
        wait for CLK_PERIOD; -- Wait for one clock cycle

        -- Test Case 2: Load parallel data '1101'
        D <= "1101";
        wait for CLK_PERIOD; -- Wait for one clock cycle
```

```
        -- Test Case 3: Load parallel data '0011'
        D <= "0011";
        wait for CLK_PERIOD; -- Wait for one clock cycle

        -- Add more test cases as needed...

        wait for 100 ns; -- Allow some additional time

        report "Simulation completed" severity note;
        wait;
    end process;

    -- Instantiate the PIPO_Register module
    PIPO_Register_inst : PIPO_Register
        port map (
        CLK => CLK,
        RESET => RESET,
        D => D,
        Q => Q
        );

end testbench;
```



DAY 13: PARALLEL IN SERIAL OUT REGISTER

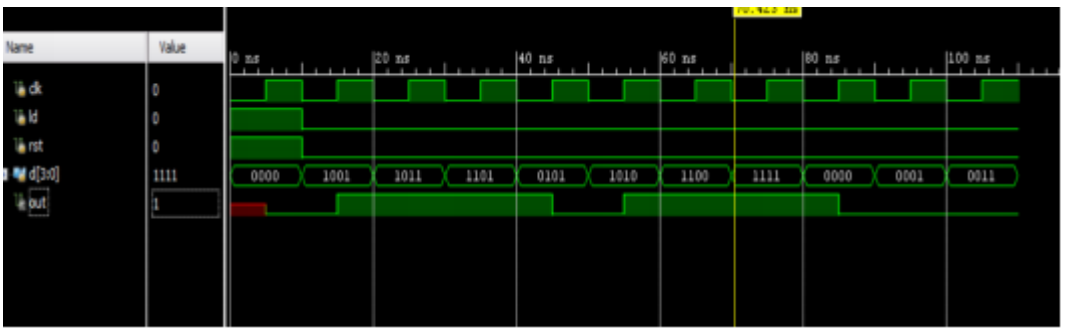
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity PISO_Register is
        Port (
        CLK   : in  STD_LOGIC;            -- Clock input
        RESET  : in  STD_LOGIC;                   -- Reset input
        D       : in  STD_LOGIC_VECTOR(3 downto 0);  -- 4-bit parallel data input
        Q       : out STD_LOGIC           -- Serial data output
        );
end PISO_Register;

architecture Behavioral of PISO_Register is
        signal reg_data : STD_LOGIC_VECTOR(3 downto 0) := "0000";
        signal shift_enable : STD_LOGIC := '0';
        signal serial_data : STD_LOGIC;
begin
        process(CLK, RESET)
        begin
        if RESET = '1' then
        reg_data <= "0000";         -- Reset the register
        shift_enable <= '0';          -- Disable shifting
        elsif rising_edge(CLK) then
        if shift_enable = '1' then
                reg_data <= D;       -- Load parallel data on clock edge
        end if;
        serial_data <= reg_data(0);  -- Extract the LSB for serial output
        shift_enable <= '1';          -- Enable shifting
        end if;
        end process;

        Q <= serial_data;  -- Output the serial data
end Behavioral;
```

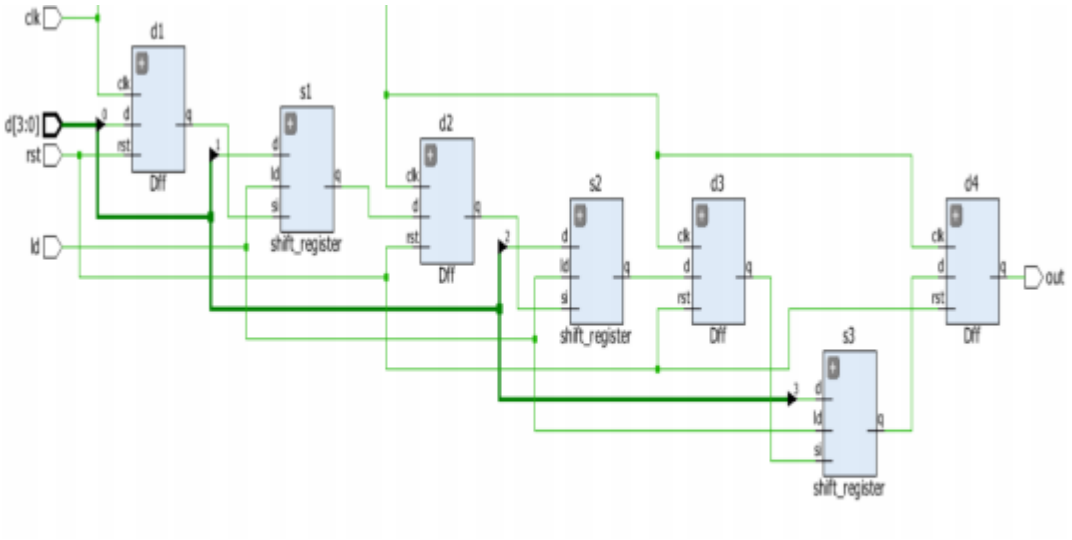
Test bench

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity test_PISO_Register is
```

```
end test_PISO_Register;

architecture testbench of test_PISO_Register is
  -- Constants
  constant CLK_PERIOD: time := 10 ns;  -- Clock period (adjust as needed)

  -- Signals
  signal CLK: std_logic := '0';     -- Clock input
  signal RESET: std_logic := '0';           -- Reset input
  signal D: std_logic_vector(3 downto 0) := "0000"; -- Parallel data input
  signal Q: std_logic;                  -- Serial data output

  -- Instantiate the PISO_Register module
  component PISO_Register
        Port (
        CLK : in  STD_LOGIC;
        RESET : in  STD_LOGIC;
        D : in  STD_LOGIC_VECTOR(3 downto 0);
        Q : out STD_LOGIC
        );
  end component;

  -- Clock generation process
  process
  begin
        while now < 1000 ns loop  -- Simulate for 1000 ns (adjust as needed)
        CLK <= '0';
        wait for CLK_PERIOD / 2;
        CLK <= '1';
        wait for CLK_PERIOD / 2;
        end loop;
        wait;
  end process;

  -- Stimulus process
  process
  begin
        RESET <= '1'; -- Assert reset initially
        D <= "0000";  -- Set initial parallel data

        wait for 20 ns; -- Wait for a few clock cycles

        RESET <= '0'; -- Deassert reset
```

```
        -- Test Case 1: Load parallel data '1010'
        D <= "1010";
        wait for CLK_PERIOD; -- Wait for one clock cycle

        -- Test Case 2: Load parallel data '1101'
        D <= "1101";
        wait for CLK_PERIOD; -- Wait for one clock cycle

        -- Test Case 3: Load parallel data '0011'
        D <= "0011";
        wait for CLK_PERIOD; -- Wait for one clock cycle

        -- Add more test cases as needed...

        wait for 100 ns; -- Allow some additional time

        report "Simulation completed" severity note;
        wait;
    end process;

    -- Instantiate the PISO_Register module
    PISO_Register_inst : PISO_Register
        port map (
        CLK => CLK,
        RESET => RESET,
        D => D,
        Q => Q
        );

end testbench;
```




clk
d[3:0]
rst
ld
d1
Dff
s1
shift_register
d2
Dff
s2
shift_register
d3
Dff
s3
shift_register
d4
Dff
out

DAY 14: BIDIRECTION SHIFT REGISTER

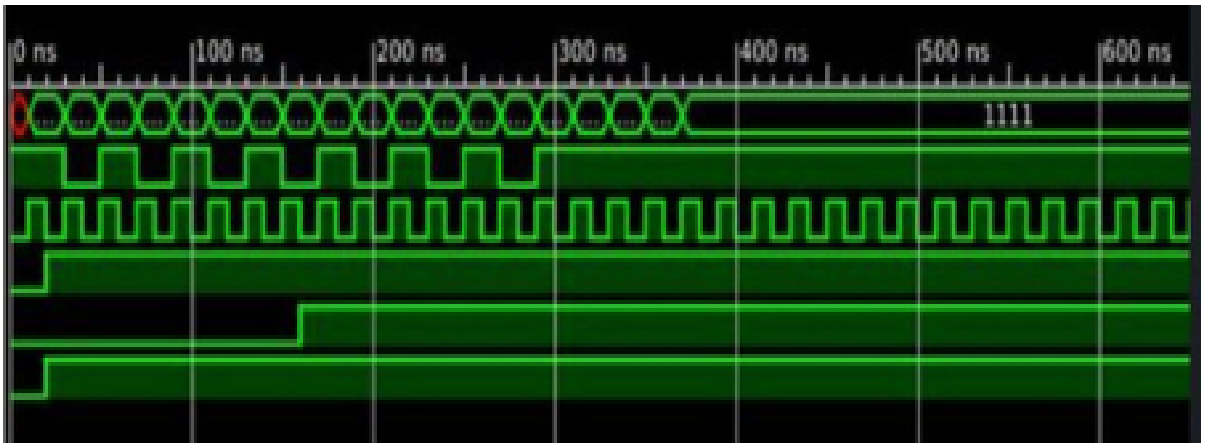
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity Bidirectional_Shift_Register is
        Port (
        CLK   : in  STD_LOGIC;            -- Clock input
        RESET        : in  STD_LOGIC;            -- Reset input
        SHIFT_L  : in  STD_LOGIC;            -- Shift left control input
        SHIFT_R  : in  STD_LOGIC;            -- Shift right control input
        D       : in  STD_LOGIC_VECTOR(7 downto 0);  -- 8-bit parallel data input
        Q       : out STD_LOGIC_VECTOR(7 downto 0)  -- 8-bit parallel data
output
        );
end Bidirectional_Shift_Register;

architecture Behavioral of Bidirectional_Shift_Register is
        signal reg_data : STD_LOGIC_VECTOR(7 downto 0) := "00000000";
begin
        process(CLK, RESET)
        begin
        if RESET = '1' then
        reg_data <= "00000000";  -- Reset the register
        elsif rising_edge(CLK) then
        if SHIFT_L = '1' then
                reg_data <= '0' & reg_data(7 downto 1);  -- Shift left
        elsif SHIFT_R = '1' then
                reg_data <= reg_data(6 downto 0) & '0';  -- Shift right
        else
                reg_data <= D;  -- Load parallel data on clock edge when no
shifting
        end if;
        end if;
        end process;

        Q <= reg_data;  -- Output the parallel data
end Behavioral;
```

Test bench:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity test_Bidirectional_Shift_Register is
end test_Bidirectional_Shift_Register;

architecture testbench of test_Bidirectional_Shift_Register is
  -- Constants
  constant CLK_PERIOD: time := 10 ns;  -- Clock period (adjust as needed)

  -- Signals
  signal CLK: std_logic := '0';     -- Clock input
  signal RESET: std_logic := '0';          -- Reset input
  signal SHIFT_L: std_logic := '0';        -- Shift left control input
  signal SHIFT_R: std_logic := '0';        -- Shift right control input
  signal D: std_logic_vector(7 downto 0) := "00000000"; -- Parallel data input
  signal Q: std_logic_vector(7 downto 0);               -- Parallel data output

  -- Instantiate the Bidirectional_Shift_Register module
  component Bidirectional_Shift_Register
        Port (
        CLK : in  STD_LOGIC;
        RESET : in  STD_LOGIC;
        SHIFT_L : in  STD_LOGIC;
        SHIFT_R : in  STD_LOGIC;
        D : in  STD_LOGIC_VECTOR(7 downto 0);
        Q : out STD_LOGIC_VECTOR(7 downto 0)
        );
  end component;

  -- Clock generation process
  process
  begin
        while now < 1000 ns loop  -- Simulate for 1000 ns (adjust as needed)
        CLK <= '0';
        wait for CLK_PERIOD / 2;
        CLK <= '1';
        wait for CLK_PERIOD / 2;
        end loop;
        wait;
  end process;
```

```
-- Stimulus process
process
begin
	RESET <= '1'; -- Assert reset initially
	D <= "00000000";  -- Set initial parallel data

	wait for 20 ns; -- Wait for a few clock cycles

	RESET <= '0'; -- Deassert reset

	-- Test Case 1: Load parallel data '10101010' without shifting
	D <= "10101010";
	SHIFT_L <= '0';
	SHIFT_R <= '0';
	wait for CLK_PERIOD; -- Wait for one clock cycle

	-- Test Case 2: Shift data left
	SHIFT_L <= '1';
	wait for CLK_PERIOD; -- Wait for one clock cycle

	-- Test Case 3: Shift data right
	SHIFT_R <= '1';
	wait for CLK_PERIOD; -- Wait for one clock cycle

	-- Test Case 4: Load parallel data '11001100' without shifting
	D <= "11001100";
	SHIFT_L <= '0';
	SHIFT_R <= '0';
	wait for CLK_PERIOD; -- Wait for one clock cycle

	-- Add more test cases as needed...

	wait for 100 ns; -- Allow some additional time

	report "Simulation completed" severity note;
	wait;
end process;

-- Instantiate the Bidirectional_Shift_Register module
Bidirectional_Shift_Register_inst : Bidirectional_Shift_Register
	port map (
	CLK => CLK,
	RESET => RESET,
```

```
        SHIFT_L => SHIFT_L,
        SHIFT_R => SHIFT_R,
        D => D,
        Q => Q
        );

end testbench;
```



DAY 15: PRBS SEQUENCE GENERATOR

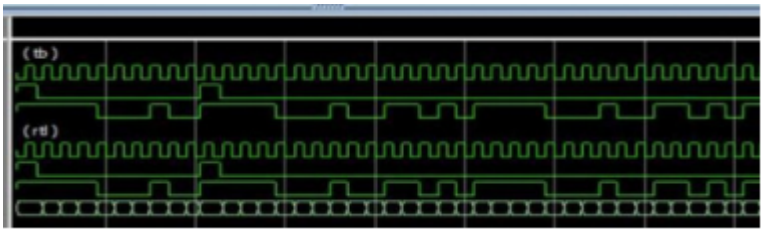
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity PRBS_Generator is
        Port (
        CLK   : in  STD_LOGIC;            -- Clock input
        RESET        : in  STD_LOGIC;            -- Reset input
        PRBS_OUT : out STD_LOGIC_VECTOR(3 downto 0)  -- 4-bit PRBS output
        );
end PRBS_Generator;

architecture Behavioral of PRBS_Generator is
        signal prbs_reg : STD_LOGIC_VECTOR(3 downto 0) := "0000";
begin
        process(CLK, RESET)
        begin
        if RESET = '1' then
        prbs_reg <= "0000";  -- Reset the PRBS generator
        elsif rising_edge(CLK) then
        prbs_reg(0) <= prbs_reg(0) xor prbs_reg(3);  -- Feedback XOR
        prbs_reg(1 downto 0) <= prbs_reg(2 downto 1); -- Shift right
        end if;
        end process;

        PRBS_OUT <= prbs_reg;  -- Output the PRBS sequence
end Behavioral;
```

Test bench

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity test_PRBS_Generator is
end test_PRBS_Generator;

architecture testbench of test_PRBS_Generator is
  -- Constants
  constant CLK_PERIOD: time := 10 ns;  -- Clock period (adjust as needed)
```

```
-- Signals
signal CLK: std_logic := '0';              -- Clock input
signal RESET: std_logic := '0';            -- Reset input
signal PRBS_OUT: std_logic_vector(3 downto 0); -- PRBS output

-- Instantiate the PRBS_Generator module
component PRBS_Generator
        Port (
        CLK : in  STD_LOGIC;
        RESET : in  STD_LOGIC;
        PRBS_OUT : out STD_LOGIC_VECTOR(3 downto 0)
        );
end component;

-- Clock generation process
process
begin
        while now < 1000 ns loop  -- Simulate for 1000 ns (adjust as needed)
        CLK <= '0';
        wait for CLK_PERIOD / 2;
        CLK <= '1';
        wait for CLK_PERIOD / 2;
        end loop;
        wait;
end process;

-- Stimulus process
process
begin
        RESET <= '1'; -- Assert reset initially

        wait for 20 ns; -- Wait for a few clock cycles

        RESET <= '0'; -- Deassert reset

        wait for CLK_PERIOD; -- Wait for one clock cycle

        -- Collect PRBS sequence (16 bits)
        for i in 1 to 16 loop
        wait for CLK_PERIOD; -- Wait for one clock cycle
        PRBS_OUT <= PRBS_OUT(PRBS_OUT'high - 1 downto 0) &
PRBS_OUT(3) after CLK_PERIOD;
        end loop;
```

```
        report "Simulation completed" severity note;
        wait;
    end process;

    -- Instantiate the PRBS_Generator module
    PRBS_Generator_inst : PRBS_Generator
        port map (
        CLK => CLK,
        RESET => RESET,
        PRBS_OUT => PRBS_OUT
        );

end testbench;
```



DAY 16: 8-BIT SUBTRACTOR

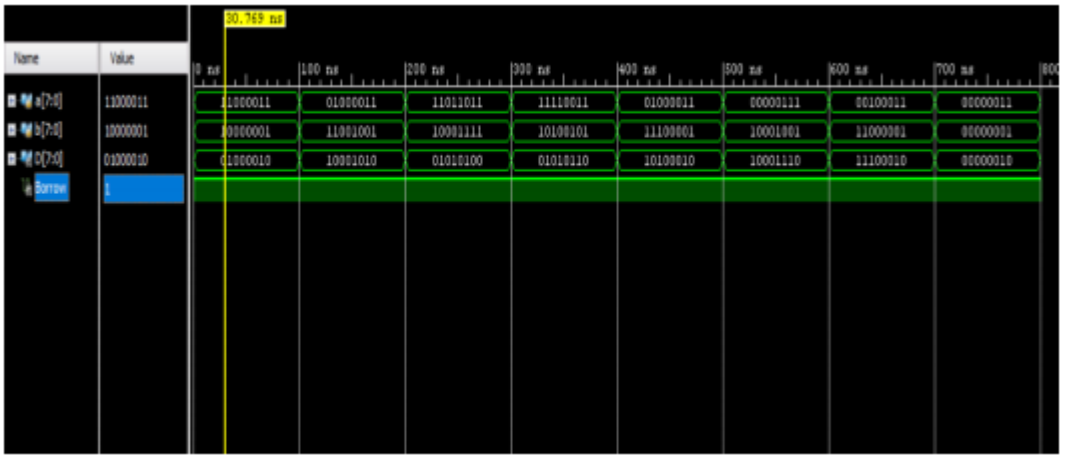
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity EightBitSubtractor is
        Port (
        A, B    : in  STD_LOGIC_VECTOR(7 downto 0);  -- 8-bit inputs
        BorrowIn  : in  STD_LOGIC;                -- Borrow input
        Difference: out STD_LOGIC_VECTOR(7 downto 0);  -- 8-bit difference
        BorrowOut : out STD_LOGIC                      -- Borrow output
        );
end EightBitSubtractor;

architecture Behavioral of EightBitSubtractor is
begin
        process(A, B, BorrowIn)
        variable temp : STD_LOGIC_VECTOR(7 downto 0);
        begin
        -- Perform 8-bit subtraction
        temp := ('0' & A) - ('0' & B) - BorrowIn;

        -- Set the BorrowOut based on borrow status
        if temp(8) = '0' then
        BorrowOut <= '1';
        else
        BorrowOut <= '0';
        end if;

        -- Output the 8-bit difference
        Difference <= temp(7 downto 0);
        end process;
end Behavioral;
```

Test bench

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
```

```
entity test_8Bit_Subtractor is
end test_8Bit_Subtractor;

architecture testbench of test_8Bit_Subtractor is
  -- Constants
  constant CLK_PERIOD: time := 10 ns;  -- Clock period (adjust as needed)

  -- Signals
  signal CLK: std_logic := '0';     -- Clock input
  signal RESET: std_logic := '0';           -- Reset input
  signal A, B: std_logic_vector(7 downto 0); -- 8-bit inputs A and B
  signal SUB_RESULT: std_logic_vector(7 downto 0); -- Result output

  -- Instantiate the 8-Bit Subtractor component
  component EightBitSubtractor
        Port (
        CLK : in  STD_LOGIC;
        RESET : in  STD_LOGIC;
        A, B : in  STD_LOGIC_VECTOR(7 downto 0);
        SUB_RESULT : out STD_LOGIC_VECTOR(7 downto 0)
        );
  end component;

  -- Clock generation process
  process
  begin
        while now < 1000 ns loop  -- Simulate for 1000 ns (adjust as needed)
        CLK <= '0';
        wait for CLK_PERIOD / 2;
        CLK <= '1';
        wait for CLK_PERIOD / 2;
        end loop;
        wait;
  end process;

  -- Stimulus process
  process
  begin
        RESET <= '1'; -- Assert reset initially

        wait for 20 ns; -- Wait for a few clock cycles

        RESET <= '0'; -- Deassert reset
```

```
        -- Test Case 1: Subtract A = 10010010, B = 01010101
        A <= "10010010";
        B <= "01010101";
        wait for CLK_PERIOD; -- Wait for one clock cycle

        -- Test Case 2: Subtract A = 11001100, B = 00110011
        A <= "11001100";
        B <= "00110011";
        wait for CLK_PERIOD; -- Wait for one clock cycle

        -- Add more test cases as needed...

        report "Simulation completed" severity note;
        wait;
    end process;

    -- Instantiate the 8-Bit Subtractor component
    Subtractor_inst : EightBitSubtractor
        port map (
        CLK => CLK,
        RESET => RESET,
        A => A,
        B => B,
        SUB_RESULT => SUB_RESULT
        );

end testbench;
```



DAY 17: 8-BIT ADDER/SUBTRACTOR

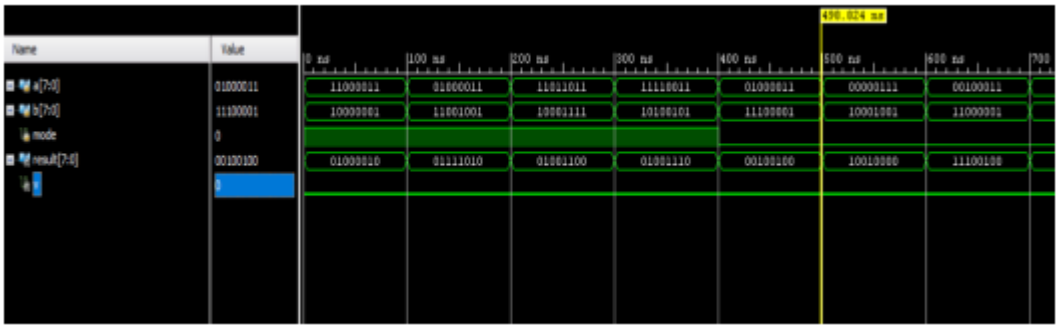
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity EightBitAdderSubtractor is
	Port (
	A, B		: in  STD_LOGIC_VECTOR(7 downto 0);  -- 8-bit inputs
	SubtractFlag	: in  STD_LOGIC;		-- Subtraction control input
	Result		: out STD_LOGIC_VECTOR(7 downto 0);  -- 8-bit result
	OverflowFlag	: out STD_LOGIC			-- Overflow flag
	);
end EightBitAdderSubtractor;

architecture Behavioral of EightBitAdderSubtractor is
	signal temp_result : STD_LOGIC_VECTOR(7 downto 0);
	signal overflow   : STD_LOGIC;
begin
	process(A, B, SubtractFlag)
	begin
	if SubtractFlag = '1' then
	-- Perform 8-bit subtraction
	temp_result <= A - B;

	-- Set the overflow flag based on subtraction
	if (A(7) = '0' and B(7) = '1' and temp_result(7) = '1') or
		(A(7) = '1' and B(7) = '0' and temp_result(7) = '0') then
		overflow <= '1';
	else
		overflow <= '0';
	end if;
	else
	-- Perform 8-bit addition
	temp_result <= A + B;

	-- Set the overflow flag based on addition
	if (A(7) = '1' and B(7) = '1' and temp_result(7) = '0') or
		(A(7) = '0' and B(7) = '0' and temp_result(7) = '1') then
		overflow <= '1';
	else
		overflow <= '0';
	end if;
```

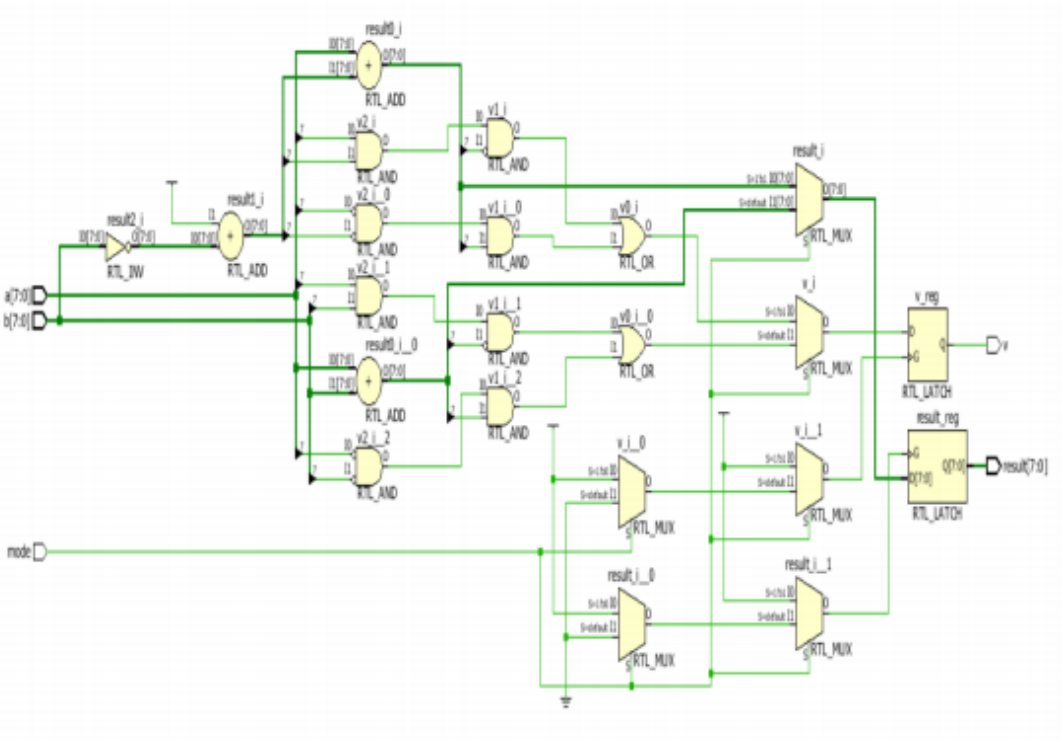
```
        end if;
        end process;

        Result <= temp_result;    -- Output the 8-bit result
        OverflowFlag <= overflow;        -- Output the overflow flag
end Behavioral;
```

Test bench

```
-- Test bench for 8-BIT ADDER/SUBTRACTOR
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity test_8Bit_Adder_Subtractor is
end test_8Bit_Adder_Subtractor;

architecture testbench of test_8Bit_Adder_Subtractor is
  -- Constants
  constant CLK_PERIOD: time := 10 ns;  -- Clock period (adjust as needed)

  -- Signals
  signal CLK: std_logic := '0';     -- Clock input
  signal RESET: std_logic := '0';           -- Reset input
  signal A, B: std_logic_vector(7 downto 0); -- 8-bit inputs A and B
  signal ADD_SUB_RESULT: std_logic_vector(7 downto 0); -- Result output

  -- Instantiate the 8-Bit Adder/Subtractor component
  component EightBitAdderSubtractor
        Port (
        CLK : in  STD_LOGIC;
        RESET : in  STD_LOGIC;
        A, B : in  STD_LOGIC_VECTOR(7 downto 0);
        ADD_SUB_RESULT : out STD_LOGIC_VECTOR(7 downto 0)
        );
  end component;

  -- Clock generation process
  process
  begin
        while now < 1000 ns loop  -- Simulate for 1000 ns (adjust as needed)
        CLK <= '0';
```

```
        wait for CLK_PERIOD / 2;
        CLK <= '1';
        wait for CLK_PERIOD / 2;
        end loop;
        wait;
    end process;

    -- Stimulus process
    process
    begin
        RESET <= '1'; -- Assert reset initially

        wait for 20 ns; -- Wait for a few clock cycles

        RESET <= '0'; -- Deassert reset

        -- Test Case 1: Add A = 10010010, B = 01010101
        A <= "10010010";
        B <= "01010101";
        wait for CLK_PERIOD; -- Wait for one clock cycle

        -- Test Case 2: Subtract A = 11001100, B = 00110011
        A <= "11001100";
        B <= "00110011";
        wait for CLK_PERIOD; -- Wait for one clock cycle

        -- Add more test cases as needed...

        report "Simulation completed" severity note;
        wait;
    end process;

    -- Instantiate the 8-Bit Adder/Subtractor component
    Adder_Subtractor_inst : EightBitAdderSubtractor
        port map (
        CLK => CLK,
        RESET => RESET,
        A => A,
        B => B,
        ADD_SUB_RESULT => ADD_SUB_RESULT
        );

end testbench;
```


Name
Value
a[7:0]
01000011
b[7:0]
11100001
mode
0
result[7:0]
00100100
v
0
430.024 ns
0 ns
100 ns
200 ns
300 ns
400 ns
500 ns
600 ns
700
11000011
01000011
11011011
11110011
01000011
00000111
00100011
10000001
11001001
10001111
10100101
11100001
10001001
11000001
01000010
01111010
01001100
01001110
00100100
10010000
11100100


result0_j
RTL_ADD
v1_j
RTL_AND
v2_j
RTL_AND
result_j
RTL_MUX
v2_j_0
RTL_AND
v1_j_0
RTL_AND
v0_j
RTL_OR
result2_j
RTL_INV
result0_j
RTL_ADD
a[7:0]
b[7:0]
v_j
RTL_MUX
v_reg
RTL_LATCH
v
v2_j_1
RTL_AND
v1_j_1
RTL_AND
v0_j_0
RTL_OR
result0_j_0
RTL_ADD
v1_j_2
RTL_AND
result_reg
RTL_LATCH
result[7:0]
v2_j_2
RTL_AND
v_j_0
RTL_MUX
v_j_1
RTL_MUX
mode
result_j_0
RTL_MUX
result_j_1
RTL_MUX

DAY 18: 4-BIT MULTIPLIER

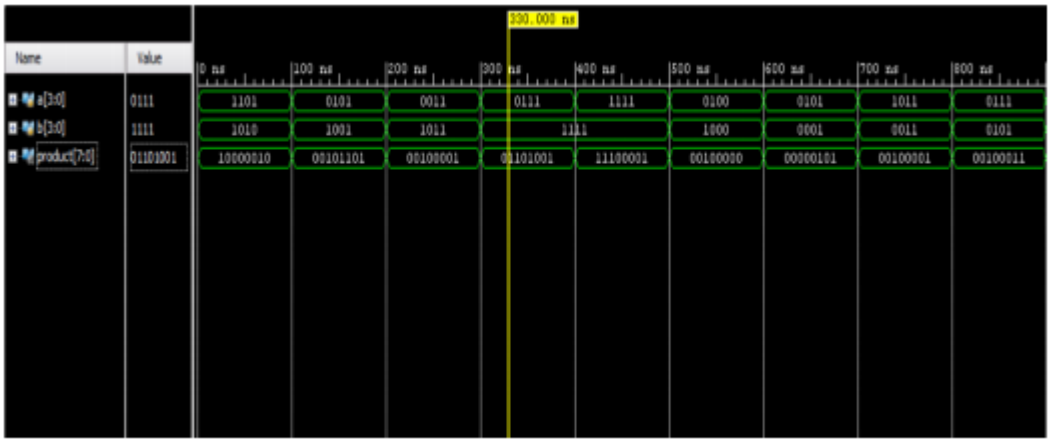
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity FourBitMultiplier is
        Port (
        A, B    : in  STD_LOGIC_VECTOR(3 downto 0);  -- 4-bit inputs
        Result  : out STD_LOGIC_VECTOR(7 downto 0)   -- 8-bit result
        );
end FourBitMultiplier;

architecture Behavioral of FourBitMultiplier is
        signal temp_result : STD_LOGIC_VECTOR(7 downto 0);
begin
        process(A, B)
        begin
        temp_result <= (others => '0');  -- Initialize the result to 0

        for i in 0 to 3 loop
        if B(i) = '1' then
                temp_result <= temp_result + (A << i);  -- Shift and add
        end if;
        end loop;
        end process;

        Result <= temp_result;  -- Output the 8-bit result
end Behavioral;
```

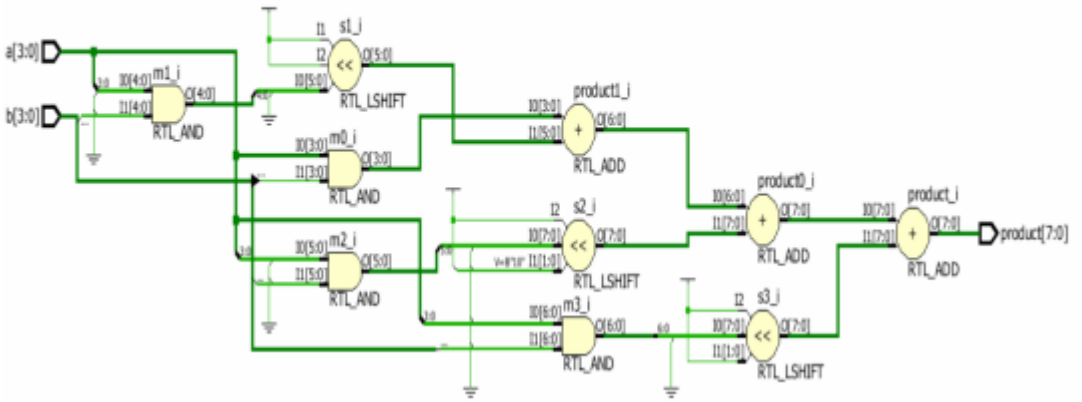
Test bench

```
-- Test bench for 4-BIT MULTIPLIER
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity test_4Bit_Multiplier is
end test_4Bit_Multiplier;
```

```
architecture testbench of test_4Bit_Multiplier is
  -- Constants
  constant CLK_PERIOD: time := 10 ns;  -- Clock period (adjust as needed)

  -- Signals
  signal CLK: std_logic := '0';     -- Clock input
  signal RESET: std_logic := '0';           -- Reset input
  signal A, B: std_logic_vector(3 downto 0); -- 4-bit inputs A and B
  signal MULTIPLY_RESULT: std_logic_vector(7 downto 0); -- Result output

  -- Instantiate the 4-Bit Multiplier component
  component FourBitMultiplier
        Port (
        CLK : in  STD_LOGIC;
        RESET : in  STD_LOGIC;
        A, B : in  STD_LOGIC_VECTOR(3 downto 0);
        MULTIPLY_RESULT : out STD_LOGIC_VECTOR(7 downto 0)
        );
  end component;

  -- Clock generation process
  process
  begin
        while now < 1000 ns loop  -- Simulate for 1000 ns (adjust as needed)
        CLK <= '0';
        wait for CLK_PERIOD / 2;
        CLK <= '1';
        wait for CLK_PERIOD / 2;
        end loop;
        wait;
  end process;

  -- Stimulus process
  process
  begin
        RESET <= '1'; -- Assert reset initially

        wait for 20 ns; -- Wait for a few clock cycles

        RESET <= '0'; -- Deassert reset

        -- Test Case 1: Multiply A = 0010, B = 0011
        A <= "0010";
        B <= "0011";
```

```
        wait for CLK_PERIOD; -- Wait for one clock cycle

        -- Test Case 2: Multiply A = 1100, B = 0101
        A <= "1100";
        B <= "0101";
        wait for CLK_PERIOD; -- Wait for one clock cycle

        -- Add more test cases as needed...

        report "Simulation completed" severity note;
        wait;
end process;

-- Instantiate the 4-Bit Multiplier component
Multiplier_inst : FourBitMultiplier
        port map (
        CLK => CLK,
        RESET
```




a[3:0]
b[3:0]
m1_j
RTL_AND
s1_j
RTL_LSHIFT
m0_j
RTL_AND
m2_j
RTL_AND
product1_j
RTL_ADD
s2_j
RTL_LSHIFT
m3_j
RTL_AND
product0_j
RTL_ADD
s3_j
RTL_LSHIFT
product_j
RTL_ADD
product[7:0]

DAY 19: FIXED POINT DIVISION

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity FixedPointDivider is
        Port (
        Dividend   : in  STD_LOGIC_VECTOR(31 downto 0);  -- 32-bit input
        Divisor        : in  STD_LOGIC_VECTOR(31 downto 0);  -- 32-bit input
        Quotient   : out STD_LOGIC_VECTOR(31 downto 0)   -- 32-bit output
        );
end FixedPointDivider;

architecture Behavioral of FixedPointDivider is
        constant FRACTIONAL_BITS : integer := 24;  -- Number of fractional bits
        constant SHIFT_AMOUNT         : integer := 24;  -- Shift amount for
division

        signal dividend_reg : STD_LOGIC_VECTOR(55 downto 0);  -- Register for
dividend
        signal quotient_reg : STD_LOGIC_VECTOR(31 downto 0);  -- Register for
quotient

        signal counter       : integer := 0;   -- Counter for division loop
begin
        process(Dividend, Divisor)
        begin
        if Divisor = "00000000000000000000000000000000" then
        Quotient <= (others => 'X');  -- Handle division by zero
        else
        dividend_reg <= Dividend & (others => '0');  -- Shift dividend left by
FRACTIONAL_BITS
        quotient_reg <= (others => '0');  -- Initialize quotient

        for counter in 0 to 31 loop
                dividend_reg <= '0' & dividend_reg(55 downto 1);  -- Shift dividend
left
                quotient_reg <= quotient_reg(30 downto 0) & '0';  -- Shift quotient
left

                if dividend_reg(55 downto 32) >= Divisor then
                quotient_reg(31) <= '1';  -- Set quotient bit
```

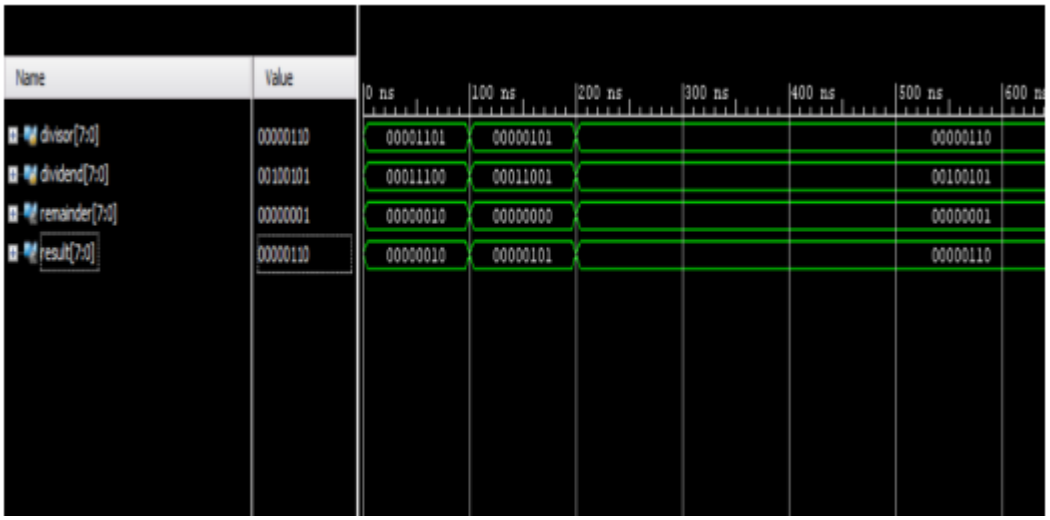
```
                dividend_reg(55 downto 32) <= dividend_reg(55 downto 32) -
Divisor;  -- Subtract divisor
                end if;
        end loop;

        Quotient <= quotient_reg;  -- Output quotient
        end if;
        end process;
end Behavioral;
```

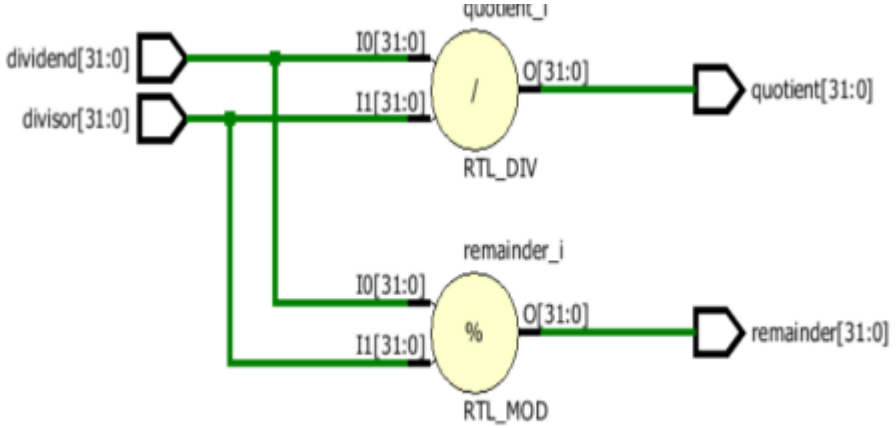
Test bench

```
-- Test bench for FIXED POINT DIVISION
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity test_FixedPoint_Division is
end test_FixedPoint_Division;

architecture testbench of test_FixedPoint_Division is
  -- Constants
  constant CLK_PERIOD: time := 10 ns;  -- Clock period (adjust as needed)

  -- Signals
  signal CLK: std_logic := '0';             -- Clock input
  signal RESET: std_logic := '0';           -- Reset input
  signal NUMERATOR, DENOMINATOR: std_logic_vector(7 downto 0); -- 8-bit
inputs
  signal DIV_RESULT: std_logic_vector(15 downto 0); -- Result output

  -- Instantiate the Fixed Point Division component
  component FixedPointDivision
        Port (
        CLK : in  STD_LOGIC;
        RESET : in  STD_LOGIC;
        NUMERATOR, DENOMINATOR : in  STD_LOGIC_VECTOR(7 downto 0);
        DIV_RESULT : out STD_LOGIC_VECTOR(15 downto 0)
        );
  end component;

  -- Clock generation process
  process
```

```
  begin
        while now < 1000 ns loop  -- Simulate for 1000 ns (adjust as needed)
        CLK <= '0';
        wait for CLK_PERIOD / 2;
        CLK <= '1';
        wait for CLK_PERIOD / 2;
        end loop;
        wait;
  end process;

  -- Stimulus process
  process
  begin
        RESET <= '1'; -- Assert reset initially

        wait for 20 ns; -- Wait for a few clock cycles

        RESET <= '0'; -- Deassert reset

        -- Test Case 1: Divide NUMERATOR = 10010010, DENOMINATOR =
00101101
        NUMERATOR <= "10010010";
        DENOMINATOR <= "00101101";
        wait for CLK_PERIOD; -- Wait for one clock cycle

        -- Test Case 2: Divide NUMERATOR = 11001100, DENOMINATOR =
00110011
        NUMERATOR <= "11001100";
        DENOMINATOR <= "00110011";
        wait for CLK_PERIOD; -- Wait for one clock cycle

        -- Add more test cases as needed...

        report "Simulation completed" severity note;
        wait;
  end process;

  -- Instantiate the Fixed Point Division component
  Division_inst : FixedPointDivision
        port map (
        CLK => CLK,
        RESET => RESET,
        NUMERATOR => NUMERATOR,
        DENOMINATOR => DENOMINATOR,
```

```
        DIV_RESULT => DIV_RESULT
        );

end testbench;
```





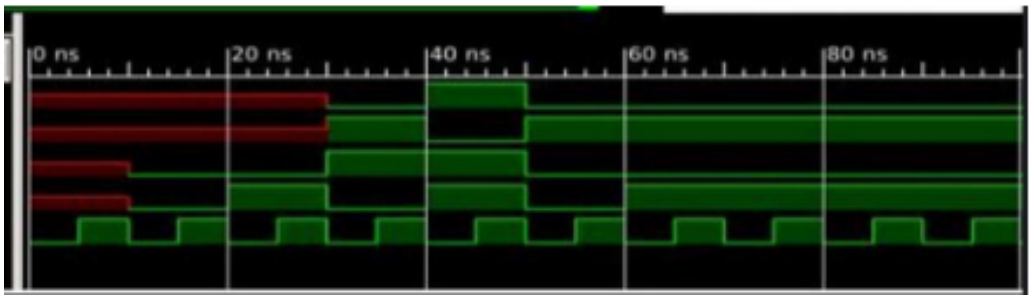
DAY 20: MASTER SLAVE JK FLIP FLOP

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity JK_FlipFlop is
        Port (
        J, K, CLK, RESET : in STD_LOGIC;
        Q, Qn : out STD_LOGIC
        );
end JK_FlipFlop;

architecture Behavioral of JK_FlipFlop is
        signal state : STD_LOGIC := '0';
begin
        process(CLK, RESET)
        begin
        if RESET = '1' then
        state <= '0';
        elsif rising_edge(CLK) then
        if J = '0' and K = '0' then
                state <= state;  -- No change
        elsif J = '0' and K = '1' then
                state <= '0';  -- Reset (Q = 0)
        elsif J = '1' and K = '0' then
                state <= '1';  -- Set (Q = 1)
        else
                state <= not state;  -- Toggle
        end if;
        end if;
        end process;

        Q <= state;
        Qn <= not state;
end Behavioral;
```

Test bench

```
-- Test bench for MASTER-SLAVE JK FLIP-FLOP
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity test_MasterSlave_JK_FlipFlop is
end test_MasterSlave_JK_FlipFlop;

architecture testbench of test_MasterSlave_JK_FlipFlop is
  -- Constants
  constant CLK_PERIOD: time := 10 ns;  -- Clock period (adjust as needed)

  -- Signals
  signal CLK: std_logic := '0';             -- Clock input
  signal RESET: std_logic := '0';           -- Reset input
  signal J, K: std_logic := '0';            -- J and K inputs
  signal Q, Qn: std_logic;                  -- Outputs

  -- Instantiate the Master-Slave JK Flip-Flop component
  component MasterSlaveJKFlipFlop
        Port (
        CLK : in  STD_LOGIC;
        RESET : in  STD_LOGIC;
        J, K : in  STD_LOGIC;
        Q, Qn : out STD_LOGIC
        );
  end component;

  -- Clock generation process
  process
  begin
        while now < 1000 ns loop  -- Simulate for 1000 ns (adjust as needed)
        CLK <= '0';
        wait for CLK_PERIOD / 2;
        CLK <= '1';
        wait for CLK_PERIOD / 2;
        end loop;
        wait;
  end process;
```

```
  -- Stimulus process
  process
  begin
        RESET <= '1'; -- Assert reset initially

        wait for 20 ns; -- Wait for a few clock cycles

        RESET <= '0'; -- Deassert reset

        -- Test Case 1: Set J and K to toggle Q
        J <= '1';
        K <= '1';
        wait for 2 * CLK_PERIOD; -- Wait for two clock cycles

        -- Test Case 2: Set J to '1' and K to '0' to set Q
        J <= '1';
        K <= '0';
        wait for 2 * CLK_PERIOD; -- Wait for two clock cycles

        -- Add more test cases as needed...

        report "Simulation completed" severity note;
        wait;
  end process;

  -- Instantiate the Master-Slave JK Flip-Flop component
  FlipFlop_inst : MasterSlaveJKFlipFlop
        port map (
        CLK => CLK,
        RESET => RESET,
        J => J,
        K => K,
        Q => Q,
        Qn => Qn
        );

end testbench;
```


0 ns
20 ns
40 ns
60 ns
80 ns

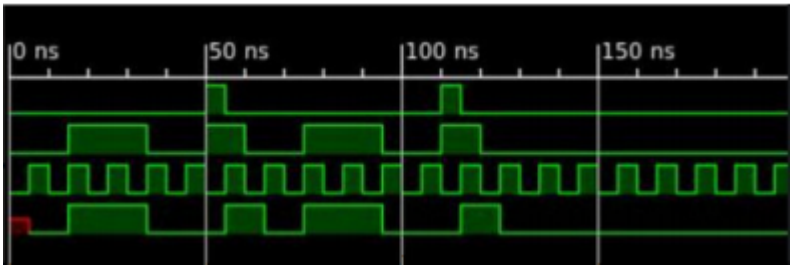
DAY 21: POSITIVE EDGE DETECTOR

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity PositiveEdgeDetector is
        Port (
        CLK : in STD_LOGIC;
        RISING_EDGE : out STD_LOGIC
        );
end PositiveEdgeDetector;

architecture Behavioral of PositiveEdgeDetector is
        signal prev_clk : STD_LOGIC := '0';
begin
        process(CLK)
        begin
        if rising_edge(CLK) then
        RISING_EDGE <= (CLK and (not prev_clk));
        prev_clk <= CLK;
        else
        RISING_EDGE <= '0';
        end if;
        end process;
end Behavioral;
```

Test bench

```
-- Test bench for POSITIVE EDGE DETECTOR
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity test_PositiveEdge_Detector is
end test_PositiveEdge_Detector;

architecture testbench of test_PositiveEdge_Detector is
  -- Constants
  constant CLK_PERIOD: time := 10 ns;  -- Clock period (adjust as needed)

  -- Signals
```

```
signal CLK: std_logic := '0';              -- Clock input
signal RESET: std_logic := '0';            -- Reset input
signal D: std_logic;                       -- Input signal
signal POS_EDGE_DETECT: std_logic;       -- Output signal

-- Instantiate the Positive Edge Detector component
component PositiveEdgeDetector
        Port (
        CLK : in  STD_LOGIC;
        RESET : in  STD_LOGIC;
        D : in  STD_LOGIC;
        POS_EDGE_DETECT : out STD_LOGIC
        );
end component;

-- Clock generation process
process
begin
        while now < 1000 ns loop  -- Simulate for 1000 ns (adjust as needed)
        CLK <= '0';
        wait for CLK_PERIOD / 2;
        CLK <= '1';
        wait for CLK_PERIOD / 2;
        end loop;
        wait;
end process;

-- Stimulus process
process
```



DAY 22: BCD ADDER

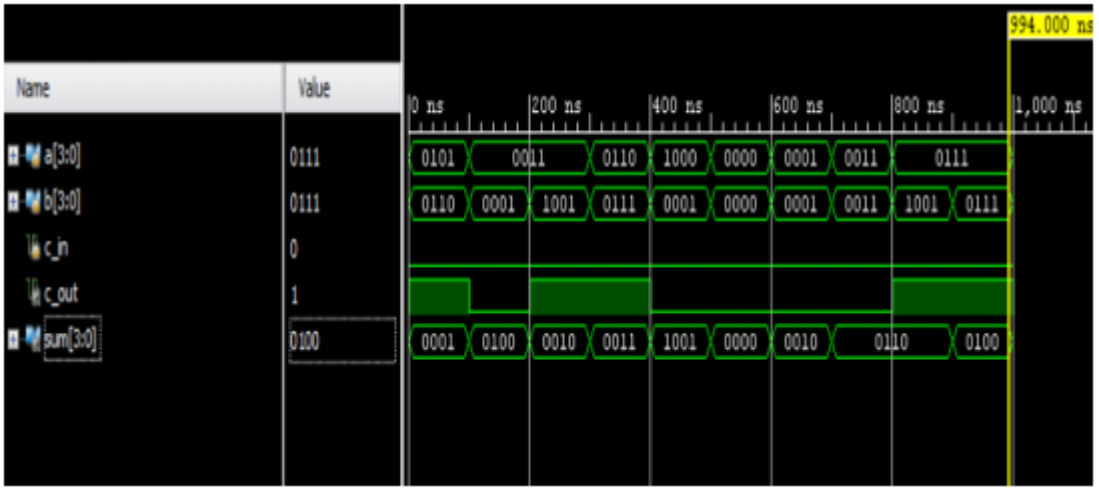
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity BCD_Adder is
        Port (
        A, B : in STD_LOGIC_VECTOR(3 downto 0);
        SUM : out STD_LOGIC_VECTOR(3 downto 0);
        COUT : out STD_LOGIC
        );
end BCD_Adder;

architecture Behavioral of BCD_Adder is
begin
        process(A, B)
        begin
        SUM <= A + B;
        if SUM > "1001" then
        SUM <= SUM + "0110";
        end if;
        COUT <= '1' when SUM > "1001" else '0';
        end process;
end Behavioral;
```

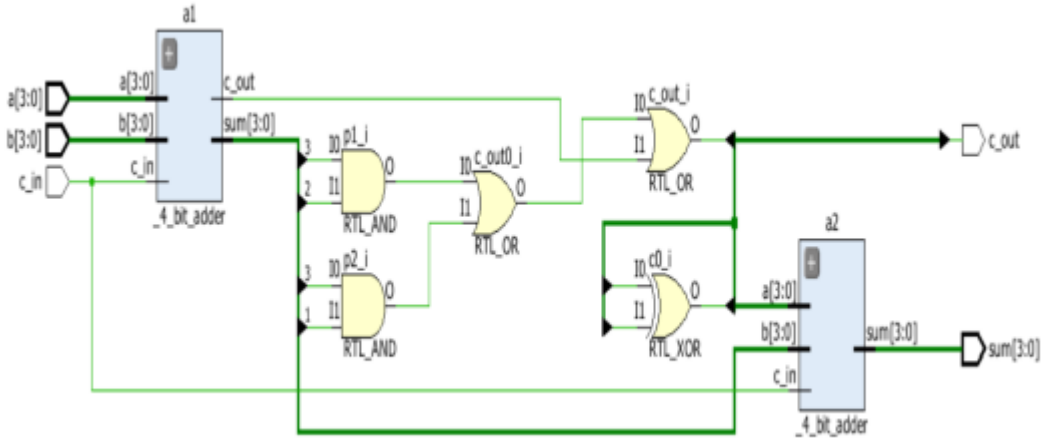
Test bench

```
-- Test bench for BCD ADDER
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity test_BCD_Adder is
end test_BCD_Adder;

architecture testbench of test_BCD_Adder is
  -- Constants
  constant CLK_PERIOD: time := 10 ns;  -- Clock period (adjust as needed)

  -- Signals
  signal CLK: std_logic := '0';             -- Clock input
  signal RESET: std_logic := '0';           -- Reset input
```

```
signal A, B: std_logic_vector(3 downto 0);  -- 4-bit BCD inputs
signal SUM: std_logic_vector(3 downto 0);           -- Sum output
signal CARRY_OUT: std_logic;                    -- Carry out

-- Instantiate the BCD Adder component
component BCDAdder
        Port (
        CLK : in  STD_LOGIC;
        RESET : in  STD_LOGIC;
        A, B : in  STD_LOGIC_VECTOR(3 downto 0);
        SUM : out STD_LOGIC_VECTOR(3 downto 0);
        CARRY_OUT : out STD_LOGIC
        );
end component;

-- Clock generation process
process
begin
        while now < 1000 ns loop  -- Simulate for 1000 ns (adjust as needed)
        CLK <= '0';
        wait for CLK_PERIOD / 2;
        CLK <= '1';
        wait for CLK_PERIOD / 2;
        end loop;
        wait;
end process;

-- Stimulus process
process
begin
        RESET <= '1'; -- Assert reset initially

        wait for 20 ns; -- Wait for a few clock cycles

        RESET <= '0'; -- Deassert reset

        -- Test Case 1: A = 0010, B = 0011
        A <= "0010";
        B <= "0011";
        wait for CLK_PERIOD; -- Wait for one clock cycle

        -- Test Case 2: A = 1100, B = 0101
        A <= "1100";
        B <= "0101";
```

```
        wait for CLK_PERIOD; -- Wait for one clock cycle

        -- Add more test cases as needed...

        report "Simulation completed" severity note;
        wait;
    end process;

    -- Instantiate the BCD Adder component
    BCD_Adder_inst : BCDAdder
        port map (
        CLK => CLK,
        RESET => RESET,
        A => A,
        B => B,
        SUM => SUM,
        CARRY_OUT => CARRY_OUT
        );

end testbench;
```




a1
a[3:0]
b[3:0]
c_in
c_out
sum[3:0]
_4_bit_adder
p1_i
RTL_AND
p2_i
RTL_AND
c_out0_i
RTL_OR
c_out_i
RTL_OR
c0_i
RTL_XOR
a2
a[3:0]
b[3:0]
c_in
sum[3:0]
_4_bit_adder
c_out
sum[3:0]

DAY 23: 4-BIT CARRY SELECT ADDER

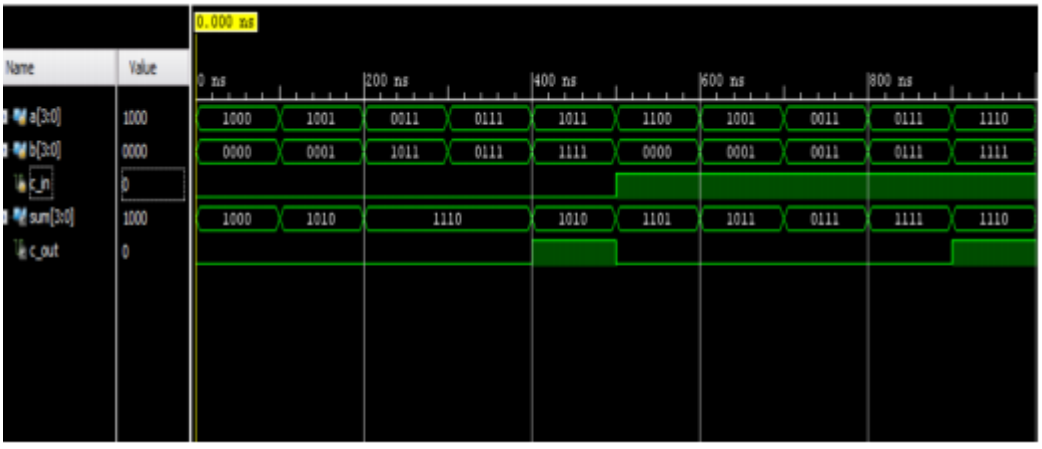
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity CarrySelectAdder is
	Port (
	A, B : in STD_LOGIC_VECTOR(3 downto 0);
	Cin : in STD_LOGIC;
	Sum : out STD_LOGIC_VECTOR(3 downto 0);
	Cout : out STD_LOGIC
	);
end CarrySelectAdder;

architecture Behavioral of CarrySelectAdder is
	signal G, P, G1, P1, G2, P2, C1, C2 : STD_LOGIC_VECTOR(3 downto 0);
begin
	G <= (others => '0');
	P <= (others => '0');
	G1 <= A and B;
	P1 <= A xor B;
	G2 <= G1 and Cin;
	P2 <= P1 or G2;
	C1 <= G1 or (P1 and Cin);
	C2 <= G2 or (P2 and G);
	Sum <= P2 xor G;
	Cout <= C2;
end Behavioral;
```

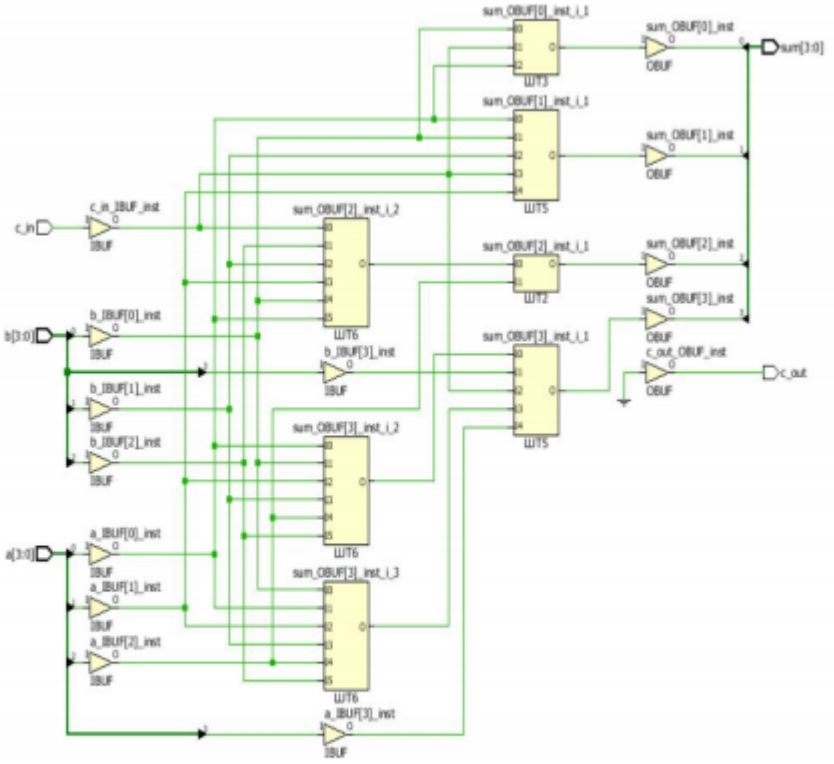
Test bench

```
-- Test bench for 4-BIT CARRY SELECT ADDER
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity test_4Bit_CarrySelect_Adder is
end test_4Bit_CarrySelect_Adder;

architecture testbench of test_4Bit_CarrySelect_Adder is
```

```
-- Constants
constant CLK_PERIOD: time := 10 ns;  -- Clock period (adjust as needed)

-- Signals
signal CLK: std_logic := '0';              -- Clock input
signal RESET: std_logic := '0';            -- Reset input
signal A, B: std_logic_vector(3 downto 0);  -- 4-bit inputs A and B
signal SUM: std_logic_vector(3 downto 0);          -- Sum output
signal CARRY_OUT: std_logic;                    -- Carry out

-- Instantiate the 4-Bit Carry Select Adder component
component FourBitCarrySelectAdder
        Port (
        CLK : in  STD_LOGIC;
        RESET : in  STD_LOGIC;
        A, B : in  STD_LOGIC_VECTOR(3 downto 0);
        SUM : out STD_LOGIC_VECTOR(3 downto 0);
        CARRY_OUT : out STD_LOGIC
        );
end component;

-- Clock generation process
process
begin
        while now < 1000 ns loop  -- Simulate for 1000 ns (adjust as needed)
        CLK <= '0';
        wait for CLK_PERIOD / 2;
        CLK <= '1';
        wait for CLK_PERIOD / 2;
        end loop;
        wait;
end process;

-- Stimulus process
process
begin
        RESET <= '1'; -- Assert reset initially

        wait for 20 ns; -- Wait for a few clock cycles

        RESET <= '0'; -- Deassert reset

        -- Test Case 1: A = 0010, B = 0011
        A <= "0010";
```

```
        B <= "0011";
        wait for CLK_PERIOD; -- Wait for one clock cycle

        -- Test Case 2: A = 1100, B = 0101
        A <= "1100";
        B <= "0101";
        wait for CLK_PERIOD; -- Wait for one clock cycle

        -- Add more test cases as needed...

        report "Simulation completed" severity note;
        wait;
  end process;

  -- Instantiate the 4-Bit Carry Select Adder component
  Carry_Select_Adder_inst : FourBitCarrySelectAdder
        port map (
        CLK => CLK,
        RESET => RESET,
        A => A,
        B => B,
        SUM => SUM,
        CARRY_OUT => CARRY_OUT
        );

end testbench;
```




sum_OBUF[0]_inst_i_1
sum_OBUF[0]_inst
sum[3:0]
LUT3
OBUF
sum_OBUF[1]_inst_i_1
sum_OBUF[1]_inst
LUT5
c_in_IBUF_inst
c_in
IBUF
sum_OBUF[2]_inst_i_2
sum_OBUF[2]_inst_i_1
sum_OBUF[2]_inst
LUT6
LUT2
sum_OBUF[3]_inst
b_IBUF[0]_inst
b[3:0]
c_out_OBUF_inst
c_out
b_IBUF[3]_inst
b_IBUF[1]_inst
b_IBUF[2]_inst
sum_OBUF[3]_inst_i_2
sum_OBUF[3]_inst_i_1
a_IBUF[0]_inst
a[3:0]
sum_OBUF[3]_inst_i_3
a_IBUF[1]_inst
a_IBUF[2]_inst
a_IBUF[3]_inst

DAY 24: MOORE FSM 1010 SEQUENCE DETECTOR

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity MooreSequenceDetector is
        Port (
        CLK, RESET : in STD_LOGIC;
        SEQ_OUT   : out STD_LOGIC
        );
end MooreSequenceDetector;

architecture Behavioral of MooreSequenceDetector is
        type State_Type is (S0, S1, S2, S3, S4);
        signal Current_State, Next_State : State_Type := S0;
begin
        process(CLK, RESET)
        begin
        if RESET = '1' then
        Current_State <= S0;
        elsif rising_edge(CLK) then
        Current_State <= Next_State;
        end if;
        end process;

        process(Current_State)
        begin
        case Current_State is
        when S0 =>
                if RESET = '1' then
                SEQ_OUT <= '0';
                else
                SEQ_OUT <= '0';
                end if;
                Next_State <= S1;
        when S1 =>
                SEQ_OUT <= '0';
                if RESET = '1' then
                Next_State <= S0;
                elsif SEQ_OUT = '1' then
                Next_State <= S2;
                else
```

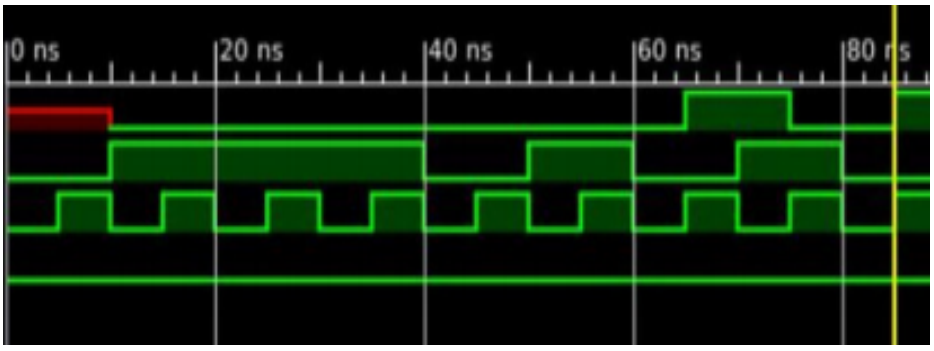
```
            Next_State <= S1;
            end if;
        when S2 =>
            SEQ_OUT <= '0';
            if SEQ_OUT = '0' then
            Next_State <= S1;
            else
            Next_State <= S3;
            end if;
        when S3 =>
            SEQ_OUT <= '0';
            if SEQ_OUT = '1' then
            Next_State <= S4;
            else
            Next_State <= S1;
            end if;
        when S4 =>
            SEQ_OUT <= '1';
            if SEQ_OUT = '0' then
            Next_State <= S1;
            else
            Next_State <= S4;
            end if;
        when others =>
            Next_State <= S0;
        end case;
        end process;
end Behavioral;
```

Test bench

```
-- Test bench for MOORE FSM 1010 SEQUENCE DETECTOR
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity test_Moore_SequenceDetector is
end test_Moore_SequenceDetector;

architecture testbench of test_Moore_SequenceDetector is
  -- Constants
  constant CLK_PERIOD: time := 10 ns;  -- Clock period (adjust as needed)
```

```
-- Signals
signal CLK: std_logic := '0';             -- Clock input
signal RESET: std_logic := '0';           -- Reset input
signal INPUT: std_logic := '0';           -- Input signal
signal OUTPUT: std_logic;                 -- Output signal

-- Instantiate the Moore Sequence Detector component
component MooreSequenceDetector
        Port (
        CLK : in  STD_LOGIC;
        RESET : in  STD_LOGIC;
        INPUT : in  STD_LOGIC;
        OUTPUT : out STD_LOGIC
        );
end component;

-- Clock generation process
process
begin
        while now < 1000 ns loop  -- Simulate for 1000 ns (adjust as needed)
        CLK <= '0';
        wait for CLK_PERIOD / 2;
        CLK <= '1';
        wait for CLK_PERIOD / 2;
        end loop;
        wait;
end process;

-- Stimulus process
process
begin
        RESET <= '1'; -- Assert reset initially

        wait for 20 ns; -- Wait for a few clock cycles

        RESET <= '0'; -- Deassert reset

        -- Test Case 1: Input sequence 1010
        INPUT <= '1';
        wait for CLK_PERIOD; -- Wait for one clock cycle
        INPUT <= '0';
        wait for CLK_PERIOD; -- Wait for one clock cycle
        INPUT <= '1';
        wait for CLK_PERIOD; -- Wait for one clock cycle
```

```
        INPUT <= '0';
        wait for CLK_PERIOD; -- Wait for one clock cycle

        -- Test Case 2: Input sequence 1101
        INPUT <= '1';
        wait for CLK_PERIOD; -- Wait for one clock cycle
        INPUT <= '1';
        wait for CLK_PERIOD; -- Wait for one clock cycle
        INPUT <= '0';
        wait for CLK_PERIOD; -- Wait for one clock cycle
        INPUT <= '1';
        wait for CLK_PERIOD; -- Wait for one clock cycle

        -- Add more test cases as needed...

        report "Simulation completed" severity note;
        wait;
  end process;

  -- Instantiate the Moore Sequence Detector component
  Sequence_Detector_inst : MooreSequenceDetector
        port map (
        CLK => CLK,
        RESET => RESET,
        INPUT => INPUT,
        OUTPUT => OUTPUT
        );

end testbench;
```



DAY 25: N:1 MUX

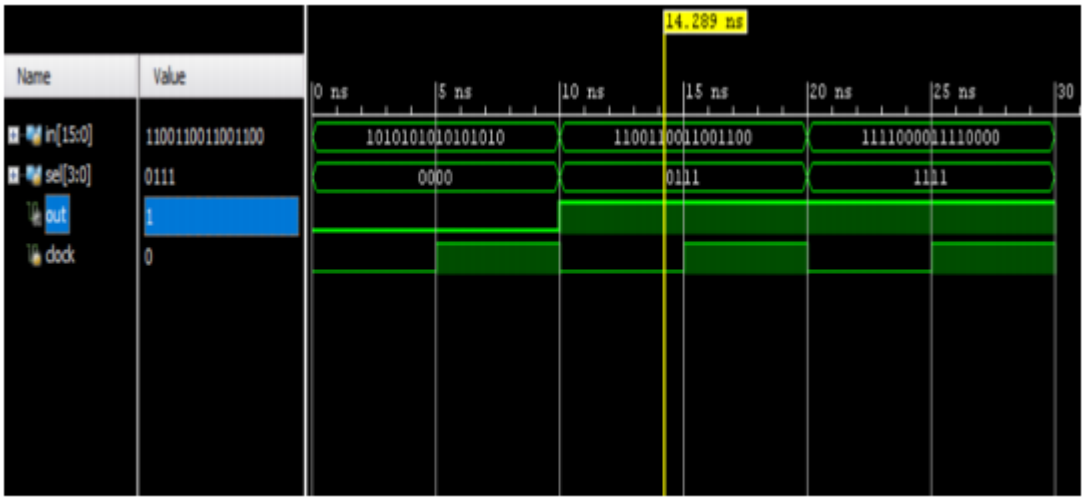
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity MUX_Nto1 is
        Port (
        A, B, C, D : in STD_LOGIC;
        SEL : in STD_LOGIC_VECTOR(1 downto 0);
        Y : out STD_LOGIC
        );
end MUX_Nto1;

architecture Behavioral of MUX_Nto1 is
```

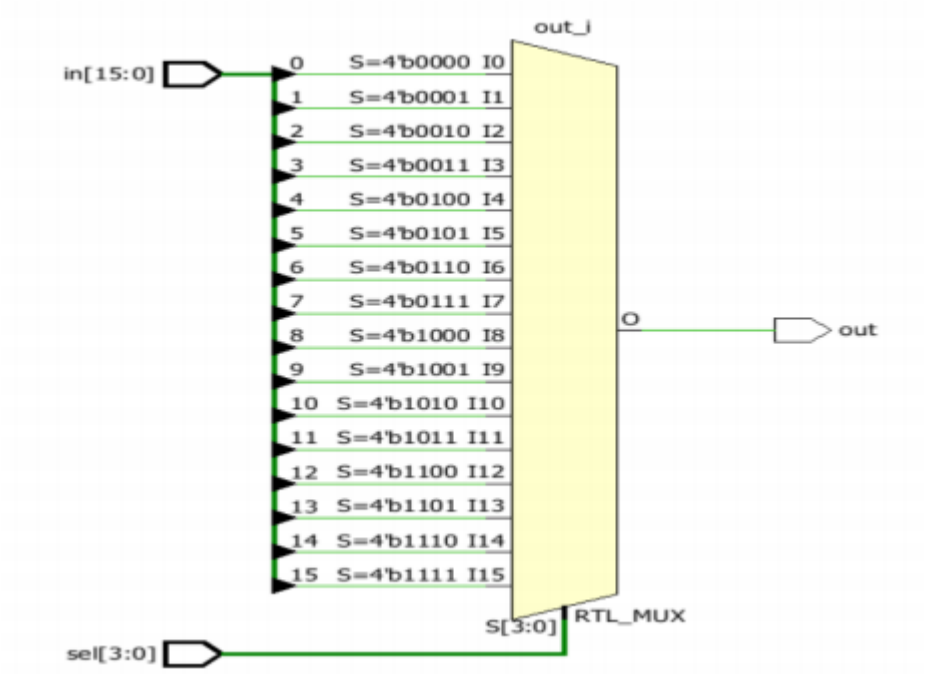
Test bench

```
-- Test bench for N:1 MUX
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity test_Nto1_MUX is
end test_Nto1_MUX;

architecture testbench of test_Nto1_MUX is
  -- Constants
  constant CLK_PERIOD: time := 10 ns;  -- Clock period (adjust as needed)

  -- Signals
  signal CLK: std_logic := '0';                -- Clock input
  signal RESET: std_logic := '0';              -- Reset input
  signal SELECT: std_logic_vector(1 downto 0);  -- Select inputs
  signal DATA_IN: std_logic_vector(3 downto 0); -- Data inputs
  signal DATA_OUT: std_logic;                  -- Data output

  -- Instantiate the N:1 MUX component
  component Nto1MUX
```

```
        Port (
        CLK : in  STD_LOGIC;
        RESET : in  STD_LOGIC;
        SELECT : in  STD_LOGIC_VECTOR(1 downto 0);
        DATA_IN : in  STD_LOGIC_VECTOR(3 downto 0);
        DATA_OUT : out STD_LOGIC
        );
end component;

-- Clock generation process
process
begin
        while now < 1000 ns loop  -- Simulate for 1000 ns (adjust as needed)
        CLK <= '0';
        wait for CLK_PERIOD / 2;
        CLK <= '1';
        wait for CLK_PERIOD / 2;
        end loop;
        wait;
end process;

-- Stimulus process
process
begin
        RESET <= '1'; -- Assert reset initially

        wait for 20 ns; -- Wait for a few clock cycles

        RESET <= '0'; -- Deassert reset

        -- Test Case 1: SELECT = "00", DATA_IN = "1010"
        SELECT <= "00";
        DATA_IN <= "1010";
        wait for CLK_PERIOD; -- Wait for one clock cycle

        -- Test Case 2: SELECT = "01", DATA_IN = "0011"
        SELECT <= "01";
        DATA_IN <= "0011";
        wait for CLK_PERIOD; -- Wait for one clock cycle

        -- Add more test cases as needed...

        report "Simulation completed" severity note;
        wait;
```

```
  end process;

  -- Instantiate the N:1 MUX component
  MUX_inst : Nto1MUX
          port map (
          CLK => CLK,
          RESET => RESET,
          SELECT => SELECT,
          DATA_IN => DATA_IN,
          DATA_OUT => DATA_OUT
          );

end testbench;
```




out_i
in[15:0]
0 S=4'b0000 I0
1 S=4'b0001 I1
2 S=4'b0010 I2
3 S=4'b0011 I3
4 S=4'b0100 I4
5 S=4'b0101 I5
6 S=4'b0110 I6
7 S=4'b0111 I7
8 S=4'b1000 I8
9 S=4'b1001 I9
10 S=4'b1010 I10
11 S=4'b1011 I11
12 S=4'b1100 I12
13 S=4'b1101 I13
14 S=4'b1110 I14
15 S=4'b1111 I15
O
out
S[3:0]
RTL_MUX
sel[3:0]