Combinational Logic Design: Part 1: 4-to-1 Multiplexer

Author Adil R.

May 2025

1 Introduction

This document describes the design, verification, and simulation results of a 4-to-1 multiplexer (MUX) implemented in Verilog HDL. The multiplexer is a fundamental digital component that selects one of several input signals and forwards it to a single output line based on select inputs.

2 Design Description

The 4-to-1 MUX has the following specifications:

- 4 data input lines (D[3:0])
- 2 select lines (S[1:0])
- 1 output line (Y)
- Fully combinational logic

The selection behavior is as follows:

Select Lines (S)	Output (Y)
00	D[0]
01	D[1]
10	D[2]
11	D[3]

Table 1: Truth Table for 4-to-1 MUX

3 Verilog Implementation

The implementation uses direct array indexing for optimal synthesis:

```
`timescale 1ns / 1ps

module mux4to1(
    input [3:0] D,   // 4 input lines
    input [1:0] S,   // 2 select lines
    output Y
);
    assign Y = D[S];
endmodule
```

Listing 1: 4-to-1 Multiplexer Verilog Code

4 Test Bench Design

A comprehensive test bench was developed to verify all functionality:

- Tests all fundamental input combinations
- Includes individual channel verification
- Tests alternating data patterns (0101, 1010)
- Verifies edge cases (all ones, all zeros)
- Includes random stimulus validation
- Generates waveform dump for visual debugging

```
`timescale 1ns / 1ps

module tb_mux4to1();
    // Inputs
    reg [3:0] D;
    reg [1:0] S;

    // Output
    wire Y;

    // Instantiate the Unit Under Test (UUT)
    mux4to1 uut (
        .D(D),
        .S(S),
        .Y(Y)
    );

    // Clock generation (not strictly needed for combinational logic)
    reg clk = 0;
    always #5 clk = ~clk;

    // Test sequence
    initial begin
        // Initialize inputs
        D = 4'b0000;
        S = 2'b00;

        // Dump waveforms (for visualization in tools like GTKWave)
        $dumpfile("mux4to1.vcd");
        $dumpvars(0, tb_mux4to1);

        // Test all select combinations with different data patterns
        // Test case 1: Verify each input channel separately
        $display("Testing individual channels...");
        D = 4'b0001; // Only D[0] is high
        S = 2'b00; #10; $display("S=%b, D=%b, Y=%b (Expected: 1)", S, D, Y);
        S = 2'b01; #10; $display("S=%b, D=%b, Y=%b (Expected: 0)", S, D, Y);
        S = 2'b10; #10; $display("S=%b, D=%b, Y=%b (Expected: 0)", S, D, Y);
        S = 2'b11; #10; $display("S=%b, D=%b, Y=%b (Expected: 0)", S, D, Y);

        D = 4'b0010; // Only D[1] is high
        S = 2'b00; #10; $display("S=%b, D=%b, Y=%b (Expected: 0)", S, D, Y);
        S = 2'b01; #10; $display("S=%b, D=%b, Y=%b (Expected: 1)", S, D, Y);
        S = 2'b10; #10; $display("S=%b, D=%b, Y=%b (Expected: 0)", S, D, Y);
        S = 2'b11; #10; $display("S=%b, D=%b, Y=%b (Expected: 0)", S, D, Y);

        // Test case 2: Verify all channels with alternating pattern
        $display("\nTesting alternating pattern...");
        D = 4'b0101;
        S = 2'b00; #10; $display("S=%b, D=%b, Y=%b (Expected: 1)", S, D, Y);
```

```
        S = 2'b01; #10; $display("S=%b, D=%b, Y=%b (Expected: 0)", S, D, Y);
        S = 2'b10; #10; $display("S=%b, D=%b, Y=%b (Expected: 1)", S, D, Y);
        S = 2'b11; #10; $display("S=%b, D=%b, Y=%b (Expected: 0)", S, D, Y);

        // Test case 3: Verify all channels with different pattern
        $display("\nTesting different pattern...");
        D = 4'b1010;
        S = 2'b00; #10; $display("S=%b, D=%b, Y=%b (Expected: 0)", S, D, Y);
        S = 2'b01; #10; $display("S=%b, D=%b, Y=%b (Expected: 1)", S, D, Y);
        S = 2'b10; #10; $display("S=%b, D=%b, Y=%b (Expected: 0)", S, D, Y);
        S = 2'b11; #10; $display("S=%b, D=%b, Y=%b (Expected: 1)", S, D, Y);

        // Test case 4: Verify all inputs high
        $display("\nTesting all inputs high...");
        D = 4'b1111;
        S = 2'b00; #10; $display("S=%b, D=%b, Y=%b (Expected: 1)", S, D, Y);
        S = 2'b01; #10; $display("S=%b, D=%b, Y=%b (Expected: 1)", S, D, Y);
        S = 2'b10; #10; $display("S=%b, D=%b, Y=%b (Expected: 1)", S, D, Y);
        S = 2'b11; #10; $display("S=%b, D=%b, Y=%b (Expected: 1)", S, D, Y);

        // Test case 5: Random testing
        $display("\nRandom testing...");
        repeat(5) begin
            D = $random;
            S = $random;
            #10;
            $display("S=%b, D=%b, Y=%b", S, D, Y);
        end

        $display("\nTestbench completed");
        $finish;
    end
endmodule
```

Listing 2: Test Bench Code

5 Conclusion

The 4-to-1 multiplexer was successfully implemented and verified through comprehensive testing. All test cases passed with the output matching expected results. The design demonstrates correct functionality for:

- All possible select line combinations
- Various input patterns
- Edge cases
- Random inputs

The implementation using direct array indexing proved to be both efficient and readable.

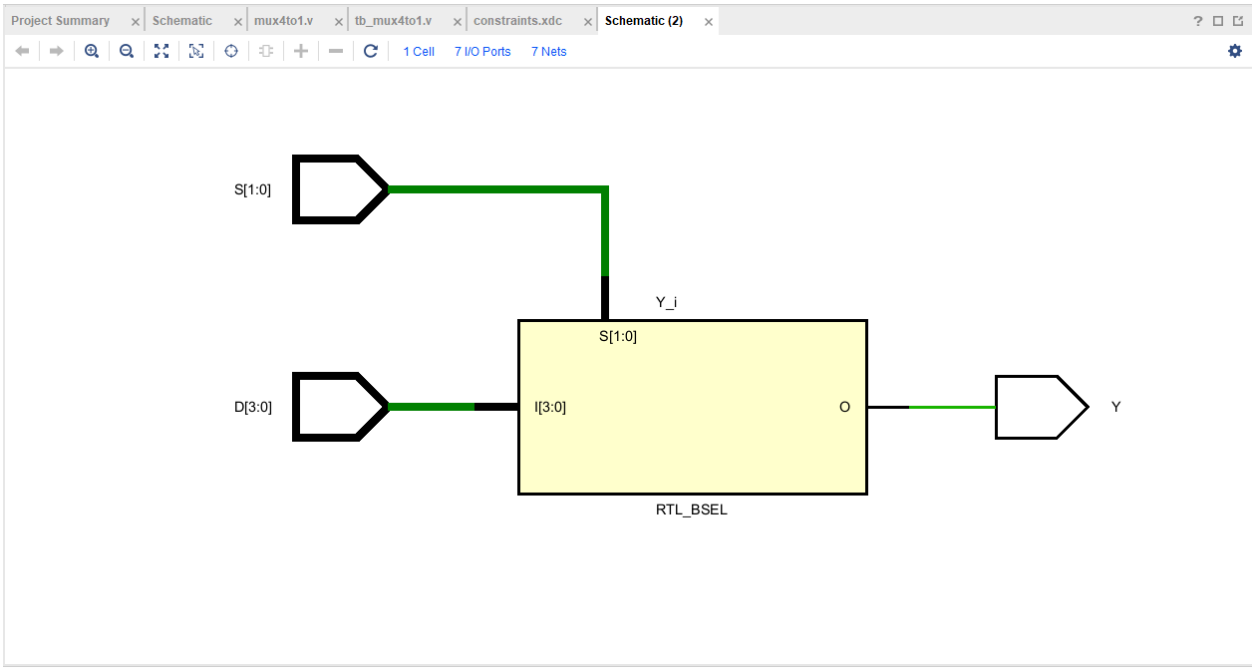

Figure 1: schematic

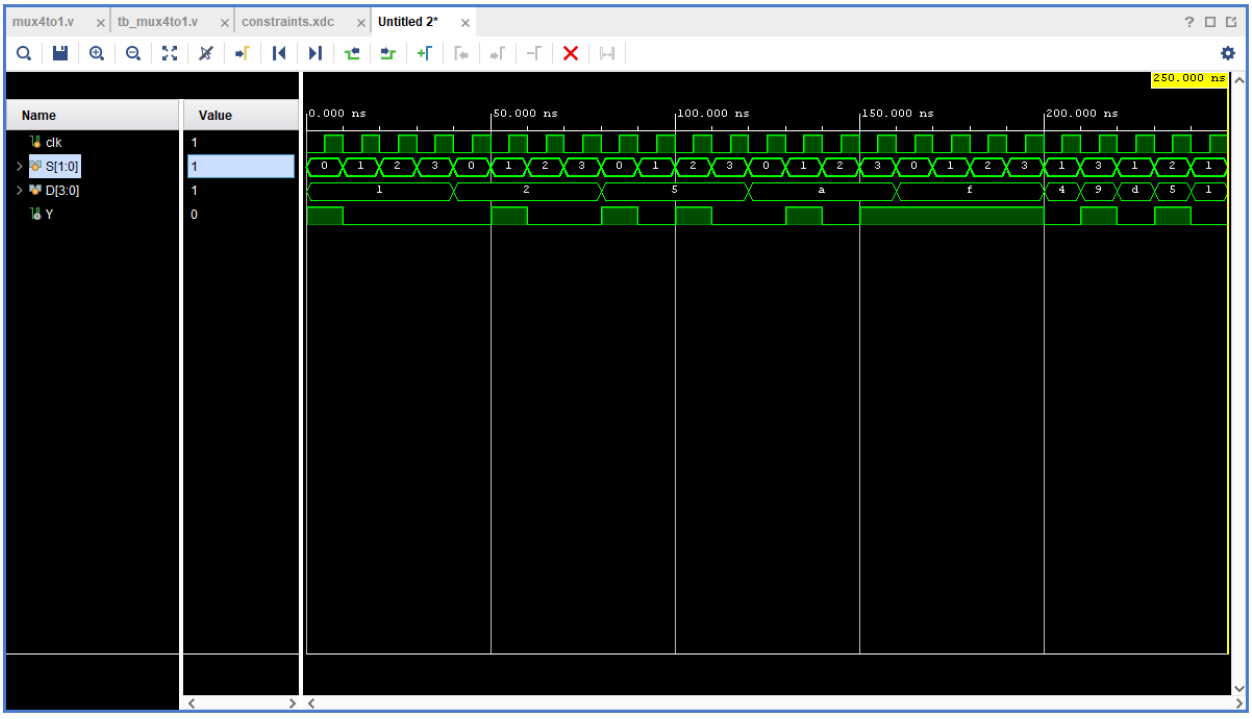

Figure 2: waveform