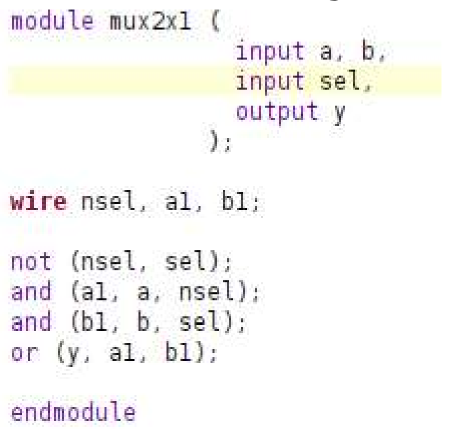
16x1 Multiplexer Using Nested 2x1 MUXes with Debounced Toggle Select Control

3300L

Dia Agrawal and Robert Lainez Torres

Design Code

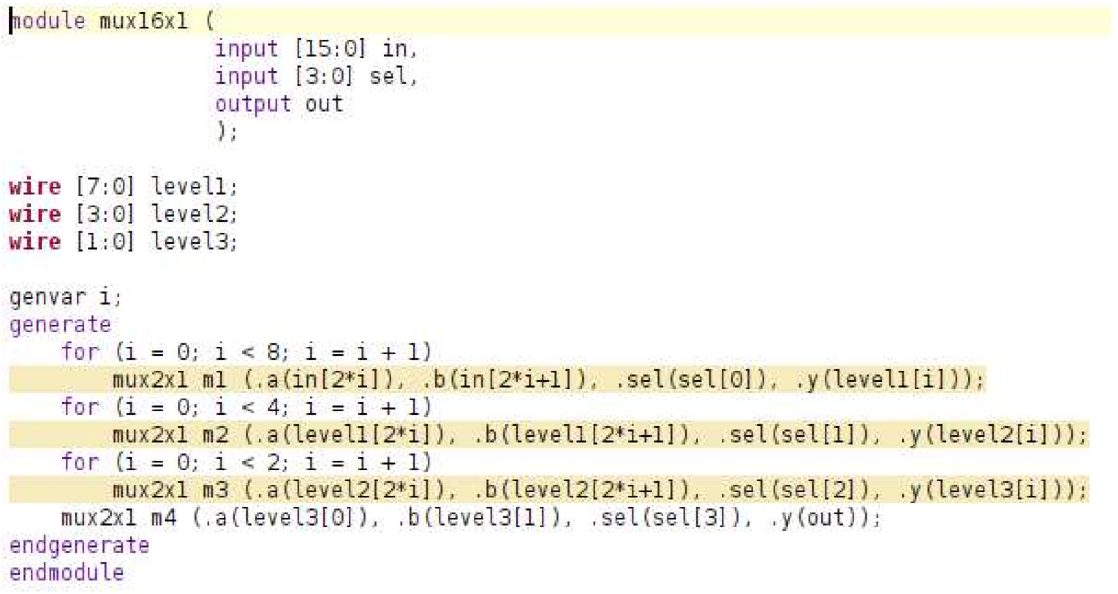
2x1 Gate Level Mux design



```
module mux2x1 (
                input a, b,
                input sel,
                output y
              );

wire nsel, a1, b1;

not (nsel, sel);
and (a1, a, nsel);
and (b1, b, sel);
or (y, a1, b1);

endmodule
```

This module implements a basic 2-to-1 multiplexer using logic gates. It selects input a when sel is 0 and input b when sel is 1.

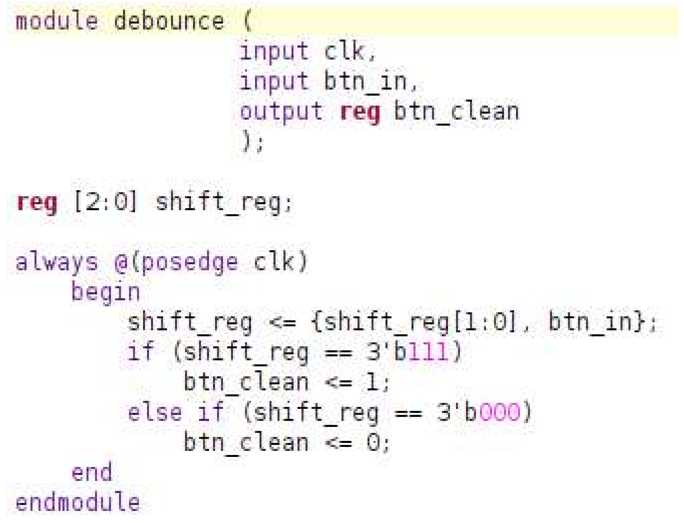
16x1 Mux implementation using 2x1 Muxes



```
module mux16x1 (
               input [15:0] in,
               input [3:0] sel,
               output out
               );

wire [7:0] level1;
wire [3:0] level2;
wire [1:0] level3;

genvar i;
generate
    for (i = 0; i < 8; i = i + 1)
        mux2x1 m1 (.a(in[2*i]), .b(in[2*i+1]), .sel(sel[0]), .y(level1[i]));
    for (i = 0; i < 4; i = i + 1)
        mux2x1 m2 (.a(level1[2*i]), .b(level1[2*i+1]), .sel(sel[1]), .y(level2[i]));
    for (i = 0; i < 2; i = i + 1)
        mux2x1 m3 (.a(level2[2*i]), .b(level2[2*i+1]), .sel(sel[2]), .y(level3[i]));
    mux2x1 m4 (.a(level3[0]), .b(level3[1]), .sel(sel[3]), .y(out));
endgenerate
endmodule
```

This module constructs a 16-to-1 multiplexer by hierarchically connecting multiple mux2x1 modules. The first layer uses 8 muxes to reduce 16 inputs to 8 using sel[0]. The second layer reduces 8 to 4 using sel[1], the third reduces 4 to 2 using sel[2], and the final mux selects the output using sel[3]. This layered approach enables selection of one out of 16 inputs using only 2x1 muxes.

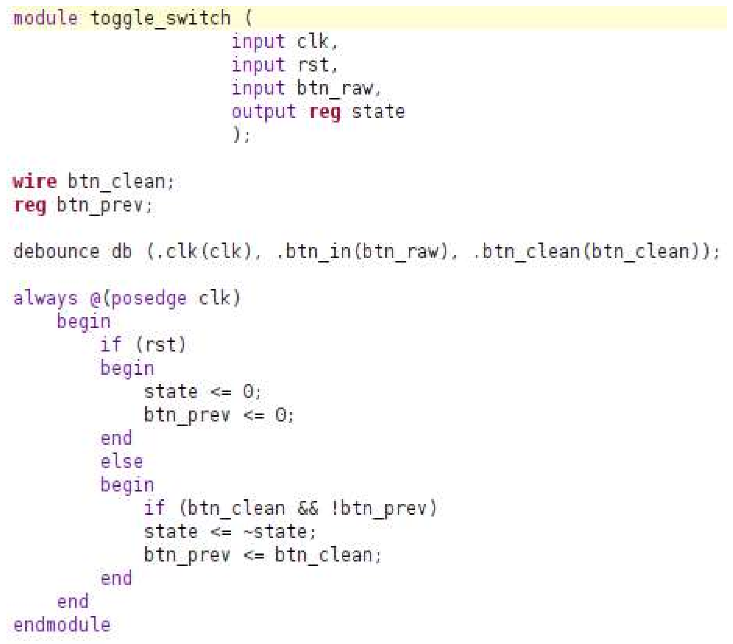
Debounce code for button



```
module debounce (
                input clk,
                input btn_in,
                output reg btn_clean
                );

reg [2:0] shift_reg;

always @(posedge clk)
    begin
        shift_reg <= {shift_reg[1:0], btn_in};
        if (shift_reg == 3'b111)
            btn_clean <= 1;
        else if (shift_reg == 3'b000)
            btn_clean <= 0;
    end
endmodule
```

This module filters out mechanical noise from a button input using a 3-bit shift register. When all three sampled bits are 1, btn_clean is set to 1; when all are 0, it's set to 0, helping detect only stable button presses/releases.

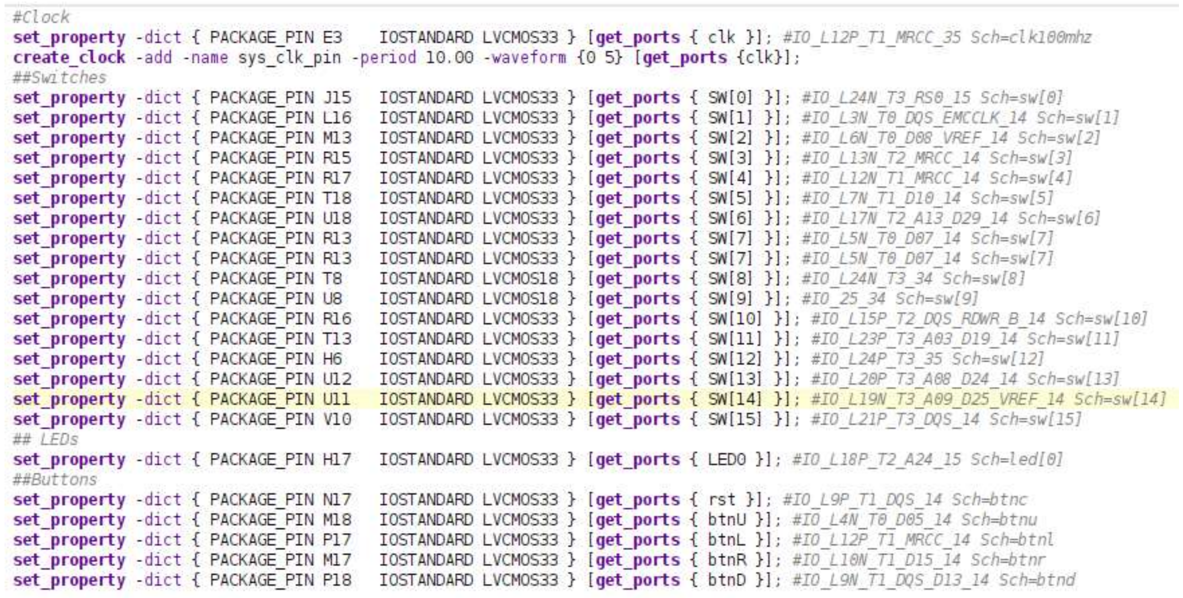
Toggle switch code using debounce code



```
module toggle_switch (
                    input clk,
                    input rst,
                    input btn_raw,
                    output reg state
                    );

wire btn_clean;
reg btn_prev;

debounce db (.clk(clk), .btn_in(btn_raw), .btn_clean(btn_clean));

always @(posedge clk)
    begin
        if (rst)
        begin
            state <= 0;
            btn_prev <= 0;
        end
        else
        begin
            if (btn_clean && !btn_prev)
            state <= ~state;
            btn_prev <= btn_clean;
        end
    end
endmodule
```

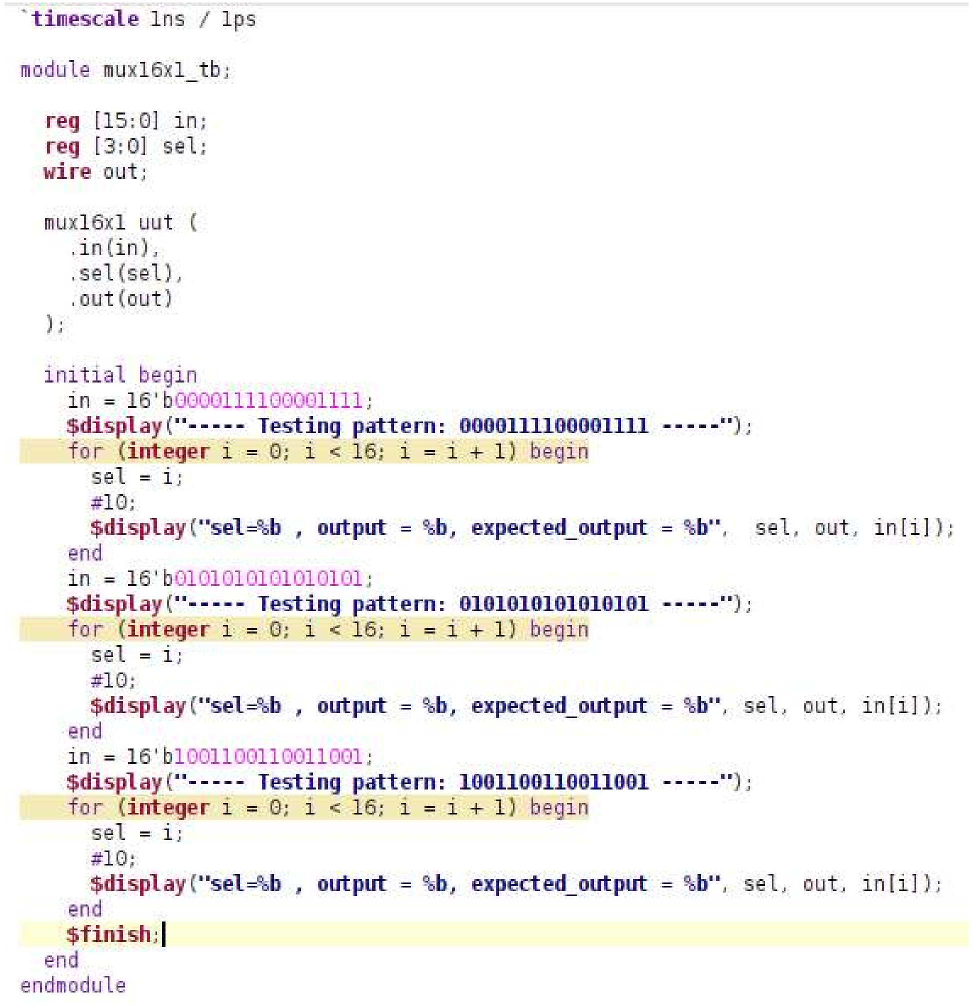
This module uses the debounce module to clean the button input (btn_raw). It then toggles the output state on each rising edge of the debounced button signal. A register btn_prev stores the previous clean state to detect rising edges. On reset, state and btn_prev are cleared to 0.

Constraint file



```
#Clock
set_property -dict { PACKAGE_PIN E3    IOSTANDARD LVCMOS33 } [get_ports { clk }]; #IO_L12P_T1_MRCC_35 Sch=clk100mhz
create_clock -add -name sys_clk_pin -period 10.00 -waveform {0 5} [get_ports {clk}];
##Switches
set_property -dict { PACKAGE_PIN J15   IOSTANDARD LVCMOS33 } [get_ports { SW[0] }]; #IO_L24N_T3_RS0_15 Sch=sw[0]
set_property -dict { PACKAGE_PIN L16   IOSTANDARD LVCMOS33 } [get_ports { SW[1] }]; #IO_L3N_T0_DQS_EMCCLK_14 Sch=sw[1]
set_property -dict { PACKAGE_PIN M13   IOSTANDARD LVCMOS33 } [get_ports { SW[2] }]; #IO_L6N_T0_D08_VREF_14 Sch=sw[2]
set_property -dict { PACKAGE_PIN R15   IOSTANDARD LVCMOS33 } [get_ports { SW[3] }]; #IO_L13N_T2_MRCC_14 Sch=sw[3]
set_property -dict { PACKAGE_PIN R17   IOSTANDARD LVCMOS33 } [get_ports { SW[4] }]; #IO_L12N_T1_MRCC_14 Sch=sw[4]
set_property -dict { PACKAGE_PIN T18   IOSTANDARD LVCMOS33 } [get_ports { SW[5] }]; #IO_L7N_T1_D10_14 Sch=sw[5]
set_property -dict { PACKAGE_PIN U18   IOSTANDARD LVCMOS33 } [get_ports { SW[6] }]; #IO_L17N_T2_A13_D29_14 Sch=sw[6]
set_property -dict { PACKAGE_PIN R13   IOSTANDARD LVCMOS33 } [get_ports { SW[7] }]; #IO_L5N_T0_D07_14 Sch=sw[7]
set_property -dict { PACKAGE_PIN R13   IOSTANDARD LVCMOS33 } [get_ports { SW[7] }]; #IO_L5N_T0_D07_14 Sch=sw[7]
set_property -dict { PACKAGE_PIN T8    IOSTANDARD LVCMOS18 } [get_ports { SW[8] }]; #IO_L24N_T3_34 Sch=sw[8]
set_property -dict { PACKAGE_PIN U8    IOSTANDARD LVCMOS18 } [get_ports { SW[9] }]; #IO_25_34 Sch=sw[9]
set_property -dict { PACKAGE_PIN R16   IOSTANDARD LVCMOS33 } [get_ports { SW[10] }]; #IO_L15P_T2_DQS_RDWR_B_14 Sch=sw[10]
set_property -dict { PACKAGE_PIN T13   IOSTANDARD LVCMOS33 } [get_ports { SW[11] }]; #IO_L23P_T3_A03_D19_14 Sch=sw[11]
set_property -dict { PACKAGE_PIN H6    IOSTANDARD LVCMOS33 } [get_ports { SW[12] }]; #IO_L24P_T3_35 Sch=sw[12]
set_property -dict { PACKAGE_PIN U12   IOSTANDARD LVCMOS33 } [get_ports { SW[13] }]; #IO_L20P_T3_A08_D24_14 Sch=sw[13]
set_property -dict { PACKAGE_PIN U11   IOSTANDARD LVCMOS33 } [get_ports { SW[14] }]; #IO_L19N_T3_A09_D25_VREF_14 Sch=sw[14]
set_property -dict { PACKAGE_PIN V10   IOSTANDARD LVCMOS33 } [get_ports { SW[15] }]; #IO_L21P_T3_DQS_14 Sch=sw[15]
## LEDs
set_property -dict { PACKAGE_PIN H17   IOSTANDARD LVCMOS33 } [get_ports { LED0 }]; #IO_L18P_T2_A24_15 Sch=led[0]
##Buttons
set_property -dict { PACKAGE_PIN N17   IOSTANDARD LVCMOS33 } [get_ports { rst }]; #IO_L9P_T1_DQS_14 Sch=btnc
set_property -dict { PACKAGE_PIN M18   IOSTANDARD LVCMOS33 } [get_ports { btnU }]; #IO_L4N_T0_D05_14 Sch=btnu
set_property -dict { PACKAGE_PIN P17   IOSTANDARD LVCMOS33 } [get_ports { btnL }]; #IO_L12P_T1_MRCC_14 Sch=btnl
set_property -dict { PACKAGE_PIN M17   IOSTANDARD LVCMOS33 } [get_ports { btnR }]; #IO_L10N_T1_D15_14 Sch=btnr
set_property -dict { PACKAGE_PIN P18   IOSTANDARD LVCMOS33 } [get_ports { btnD }]; #IO_L9N_T1_DQS_D13_14 Sch=btnd
```

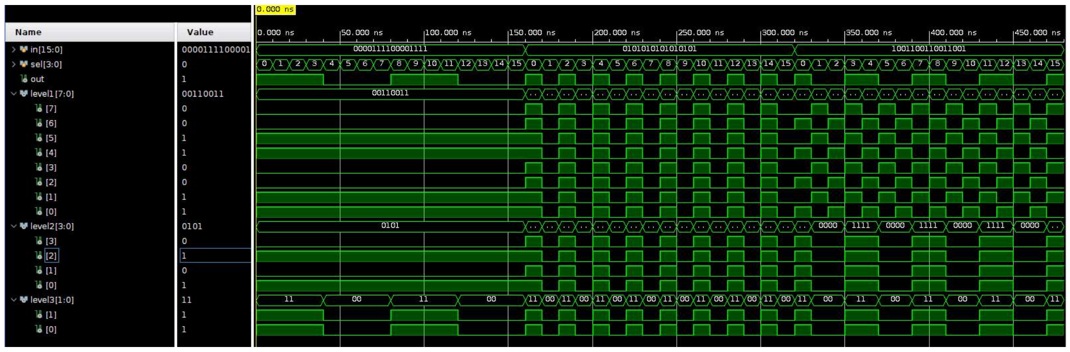
Our constraint files declare all the switches and buttons as well as one LED for output.
Testbench Code



```
`timescale 1ns / 1ps

module mux16x1_tb;

  reg [15:0] in;
  reg [3:0] sel;
  wire out;

  mux16x1 uut (
    .in(in),
    .sel(sel),
    .out(out)
  );

  initial begin
    in = 16'b0000111100001111;
    $display("----- Testing pattern: 0000111100001111 -----");
    for (integer i = 0; i < 16; i = i + 1) begin
      sel = i;
      #10;
      $display("sel=%b , output = %b, expected_output = %b",  sel, out, in[i]);
    end
    in = 16'b0101010101010101;
    $display("----- Testing pattern: 0101010101010101 -----");
    for (integer i = 0; i < 16; i = i + 1) begin
      sel = i;
      #10;
      $display("sel=%b , output = %b, expected_output = %b", sel, out, in[i]);
    end
    in = 16'b1001100110011001;
    $display("----- Testing pattern: 1001100110011001 -----");
    for (integer i = 0; i < 16; i = i + 1) begin
      sel = i;
      #10;
      $display("sel=%b , output = %b, expected_output = %b", sel, out, in[i]);
    end
    $finish;
  end
endmodule
```

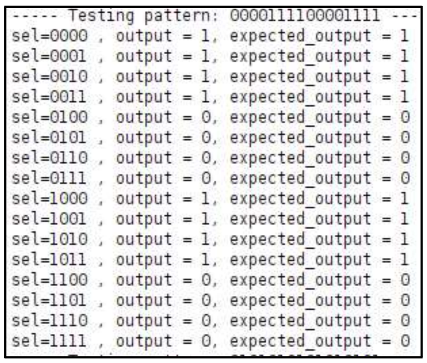
This testbench checks if the 16x1 multiplexer works correctly. It runs three different input patterns and, for each one, cycles through all 16 select values from 0 to 15. For each value of sel, it waits a short time, then prints out what the sel value is, what the multiplexer outputs, and what the correct expected output should be. This helps confirm that the mux is selecting the right input bit. After all tests are done, the simulation ends.

Behavioral Simulation

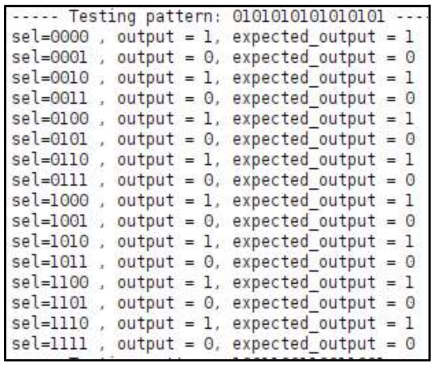


This waveform shows the actual behavior of the 16x1 multiplexer over time. As the sel value changes from 0 to 15, the out signal correctly reflects the corresponding input bit from in. It confirms the logic is working as expected.

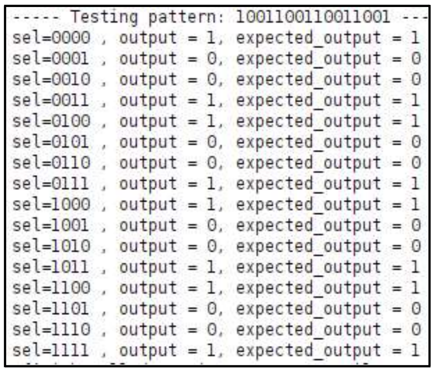
TCL Console for verification



```
----- Testing pattern: 0000111100001111 ---
sel=0000 , output = 1, expected_output = 1
sel=0001 , output = 1, expected_output = 1
sel=0010 , output = 1, expected_output = 1
sel=0011 , output = 1, expected_output = 1
sel=0100 , output = 0, expected_output = 0
sel=0101 , output = 0, expected_output = 0
sel=0110 , output = 0, expected_output = 0
sel=0111 , output = 0, expected_output = 0
sel=1000 , output = 1, expected_output = 1
sel=1001 , output = 1, expected_output = 1
sel=1010 , output = 1, expected_output = 1
sel=1011 , output = 1, expected_output = 1
sel=1100 , output = 0, expected_output = 0
sel=1101 , output = 0, expected_output = 0
sel=1110 , output = 0, expected_output = 0
sel=1111 , output = 0, expected_output = 0
```



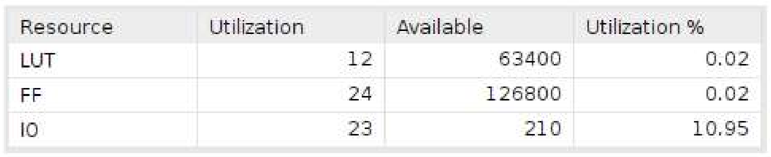
```
----- Testing pattern: 0101010101010101 ---
sel=0000 , output = 1, expected_output = 1
sel=0001 , output = 0, expected_output = 0
sel=0010 , output = 1, expected_output = 1
sel=0011 , output = 0, expected_output = 0
sel=0100 , output = 1, expected_output = 1
sel=0101 , output = 0, expected_output = 0
sel=0110 , output = 1, expected_output = 1
sel=0111 , output = 0, expected_output = 0
sel=1000 , output = 1, expected_output = 1
sel=1001 , output = 0, expected_output = 0
sel=1010 , output = 1, expected_output = 1
sel=1011 , output = 0, expected_output = 0
sel=1100 , output = 1, expected_output = 1
sel=1101 , output = 0, expected_output = 0
sel=1110 , output = 1, expected_output = 1
sel=1111 , output = 0, expected_output = 0
```



```
----- Testing pattern: 1001100110011001 ---
sel=0000 , output = 1, expected_output = 1
sel=0001 , output = 0, expected_output = 0
sel=0010 , output = 0, expected_output = 0
sel=0011 , output = 1, expected_output = 1
sel=0100 , output = 1, expected_output = 1
sel=0101 , output = 0, expected_output = 0
sel=0110 , output = 0, expected_output = 0
sel=0111 , output = 1, expected_output = 1
sel=1000 , output = 1, expected_output = 1
sel=1001 , output = 0, expected_output = 0
sel=1010 , output = 0, expected_output = 0
sel=1011 , output = 1, expected_output = 1
sel=1100 , output = 1, expected_output = 1
sel=1101 , output = 0, expected_output = 0
sel=1110 , output = 0, expected_output = 0
sel=1111 , output = 1, expected_output = 1
```

The console prints out each test case. It displays the select value, the actual output, and what the expected output should be. From the logs, you can see that the mux output always matches the expected value, proving correctness for all three test patterns

Utilization Report



Resource	Utilization	Available	Utilization %
LUT	12	63400	0.02
FF	24	126800	0.02
IO	23	210	10.95

This report shows that the design uses very minimal hardware resources — only 12 look-up tables (LUTs), 24 flip-flops (FFs), and 23 input/output pins (IOs), which is a tiny fraction of what's available on the FPGA. So, the design is both correct and efficient.

Partner Contribution Section: We both uploaded all the design code on our own and worked on the test bench together. Robert worked on gathering all the simulation, verification, and report summaries while I worked on downloading it to the board and presenting it.