Solution to the Assignment-1 [B.] answer (Answer For the mux related diagram):

Show how the function, f = w1!w3! + w1w3 + w2w3 + w1w2, can be realized using one or more instances of the circuit in Figure P4.1. Note that there are no NOT gates in the circuit; hence complements of signals have to be generated using the multiplexers in the logic block.

The below figure shows what can be the inputs for the corresponding unknown values of i1-i8
Only one instance of the below circuit can be used to implement the above ‘f’ equation.

In order to write the structural code the gate names need to be used in the form of gate_name instance_name (output, inp1,input) etc,.

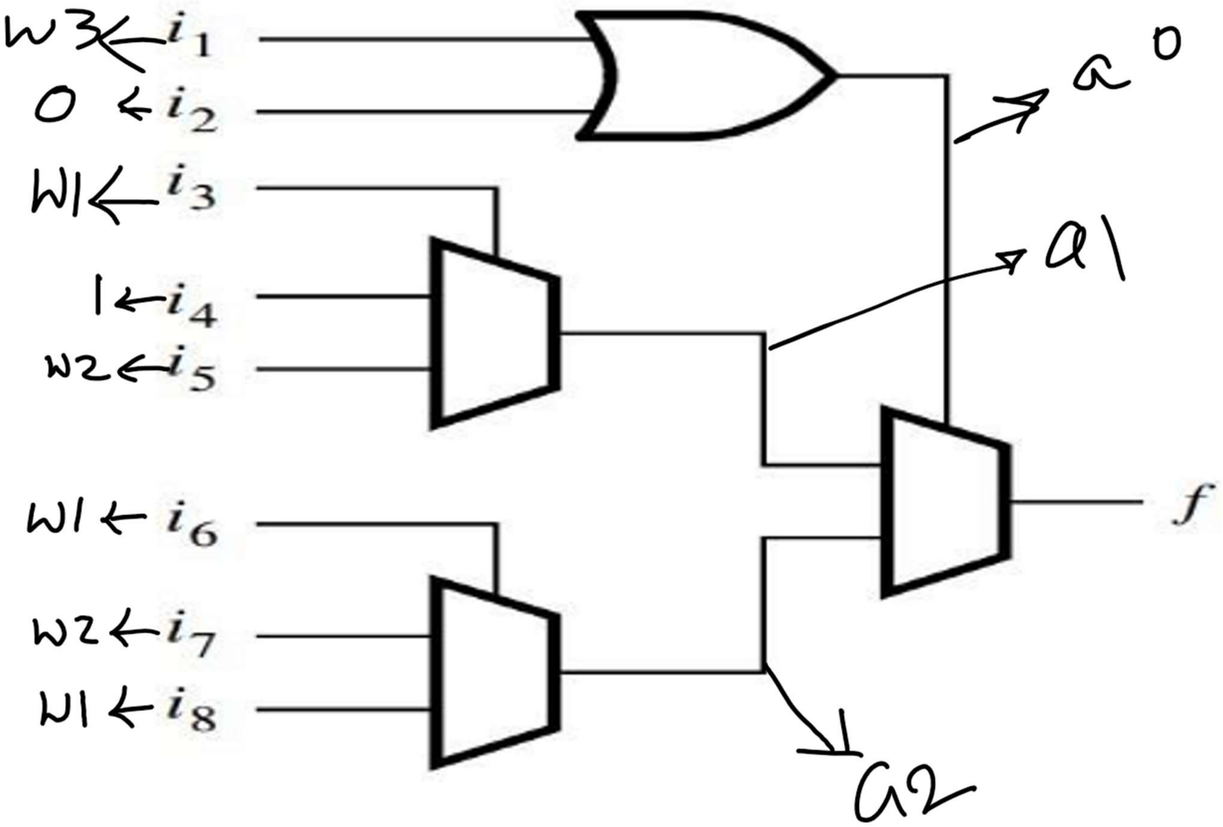


Figure 3. The FigureP4.1 with solved inputs

Design Description:

The design consists of:

- Inputs: `w1`, `w2`, `w3` (1-bit each)
- Output: `f` (1-bit)
- Internal registers: `a0`, `a1`, `a2`
- Constants: `b = 0`, `i0 = 1`

The Structural description of the Verilog using behavioural modelling style:

```
module top_module(
    input wire w3,   // Input signal for OR logic (used in or1 block)
    input wire w2,   // Input signal for mux logic
    input wire w1,   // Selector input used in both muxes
    output reg f       // Final output of the logic
);

    // Internal signals (intermediate outputs)
    reg a0;  // Output of OR logic (or1)
    reg a1;  // Output of first multiplexer (mx1)
    reg a2;  // Output of second multiplexer (mx2)

    // Constants
    wire b = 1'b0;      // Constant logic 0 (used in or1 logic)
    wire i0 = 1'b1;     // Constant logic 1 (used in mx1 logic)

    // ------------------------
    // or1 Logic Block
    // ------------------------
    // Implements: if both w3 and b are 0 -> a0 = 0
    // Otherwise, a0 = w3
    always @ (w3) begin
        if (w3 == 1'b0 && b == 1'b0)
            a0 = 1'b0;
        else
            a0 = w3;
    end

    // ------------------------
    // mx1 Logic Block
    // ------------------------
    // 2:1 Multiplexer:
    // if w1 == 1 -> a1 = w2
    // else       -> a1 = i0 (i.e., constant 1)
    always @ (w2 or w1) begin
        if (w1)
            a1 = w2;
        else
            a1 = i0;
    end

    // ------------------------
    // mx2 Logic Block
    // ------------------------
    // 2:1 Multiplexer:
    // if w1 == 1 -> a2 = w1
    // else       -> a2 = w2
    always @ (w2 or w1) begin
        if (w1)
            a2 = w1;
        else
            a2 = w2;
    end
    // ------------------------
    // mx3 Logic Block
    // ------------------------
    // 2:1 Multiplexer:
    // if a0 == 1 -> f = a2
    // else       -> f = a1
    always @ (a0 or a1 or a2) begin
        if (a0)
            f = a2;
        else
            f = a1;
    end
endmodule
```

Theory behind the testbench code:

The testbench tb_top_module is designed to verify the functionality of the top_module by applying all **8 possible input combinations** of the three inputs: w3, w2, and w1.

Key Features of the Testbench:

- **Inputs**: w1, w2, w3 (declared as reg)
- **Output**: f (observed as wire)
- **Design Under Test (DUT)**: The top_module is instantiated and connected with the testbench inputs and output.
- **Input Combinations**: A for loop iterates over values from 0 to 7 using a 3-bit vector ({w3, w2, w1}), automatically generating all combinations.
- **Timing**: A short delay #5 ensures proper signal propagation before sampling the output.
- **Display**: The simulation output is displayed in a tabular format using $display, allowing easy verification of the truth table.

Purpose of the Testbench:

To ensure that the logic inside `top_module` behaves correctly for **each combination** of `w1`, `w2`, and `w3`. It serves as an **automated test case generator** and output logger, reducing manual verification effort.

Output Interpretation:

- Each row printed by the testbench corresponds to a specific combination of inputs.
- The resulting output `f` is matched against expected behavior from the design logic.

Expected outputs:

w3	w2	w1	f
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

The testbench code for the above design the :

Since the design has w1,w2 & w3 as the input combinations and b=1'b0, i0=1'b1 already, so we need to go for the eight combinations of w1,w2 & w3.

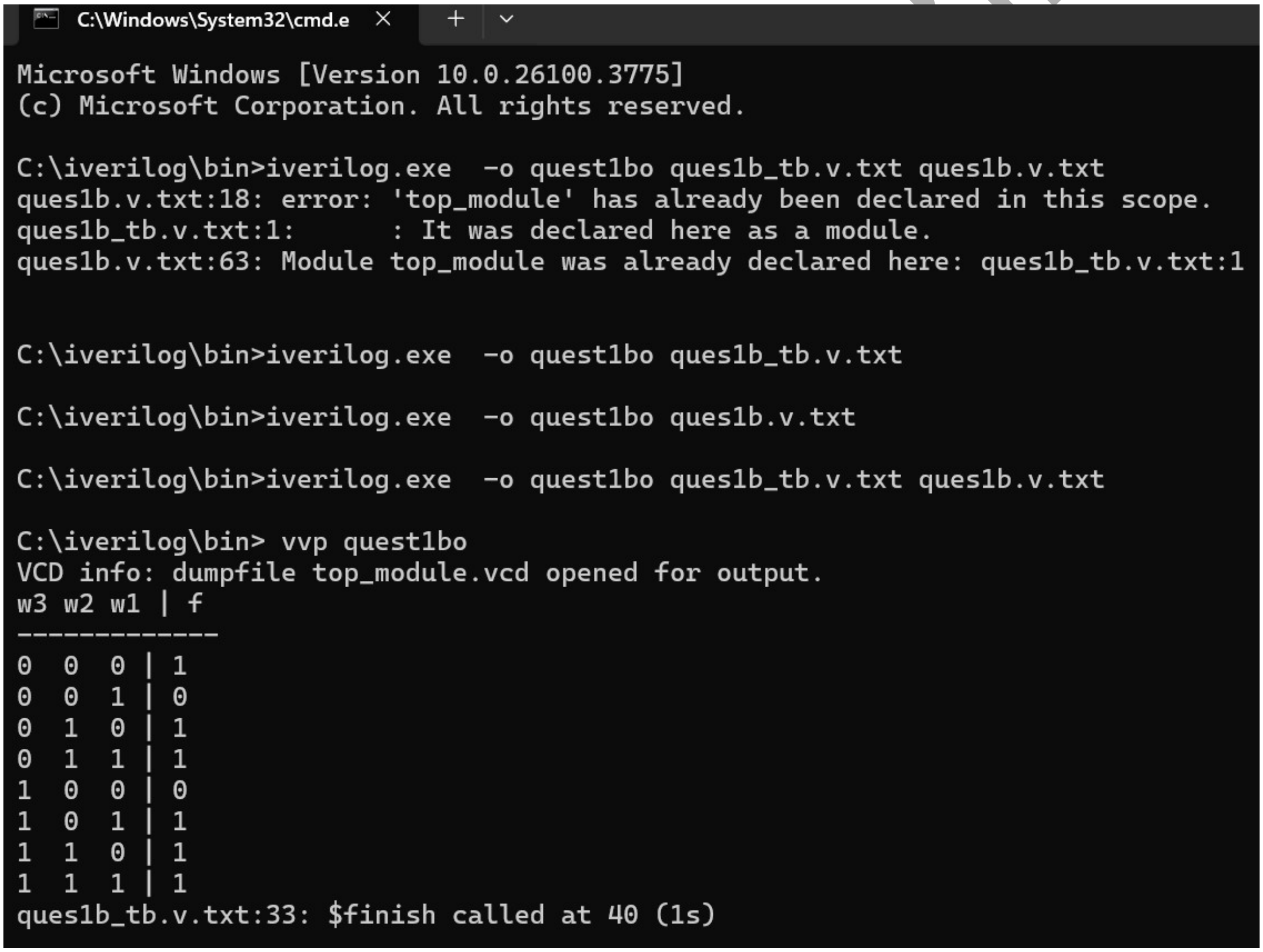
```
module tb_top_module;

    // Declare input signals for the DUT (Design Under Test)
    reg w1, w2, w3;
    wire f;  // Output from DUT
    // Instantiate the DUT (top_module)
    top_module dut (
        .w1(w1),
        .w2(w2),
        .w3(w3),
        .f(f)
    );
    // Create a VCD file for GTKWave
    initial begin
        $dumpfile("ques1b_dump.vcd"); // VCD file to store waveform data
        $dumpvars(0, tb_top_module); // Dump all signals in the testbench
    end
    // Apply all possible input combinations and generate outputs
    initial begin
        $display("w3 w2 w1 | f");
        $display("-------------");
        // Test all input combinations
        for (integer i = 0; i < 8; i = i + 1) begin
            {w3, w2, w1} = i[2:0];  // Generate inputs from 000 to 111
            #5;                     // Wait for signal propagation
            $display("%b  %b  %b | %b", w3, w2, w1, f); // Display output
        end

        $finish;  // End the simulation
    end

endmodule
```

Simulation Results:



```
Microsoft Windows [Version 10.0.26100.3775]
(c) Microsoft Corporation. All rights reserved.

C:\iverilog\bin>iverilog.exe  -o quest1bo ques1b_tb.v.txt ques1b.v.txt
ques1b.v.txt:18: error: 'top_module' has already been declared in this scope.
ques1b_tb.v.txt:1:      : It was declared here as a module.
ques1b.v.txt:63: Module top_module was already declared here: ques1b_tb.v.txt:1


C:\iverilog\bin>iverilog.exe  -o quest1bo ques1b_tb.v.txt

C:\iverilog\bin>iverilog.exe  -o quest1bo ques1b.v.txt

C:\iverilog\bin>iverilog.exe  -o quest1bo ques1b_tb.v.txt ques1b.v.txt

C:\iverilog\bin> vvp quest1bo
VCD info: dumpfile top_module.vcd opened for output.
w3 w2 w1 | f
-------------
0  0  0 | 1
0  0  1 | 0
0  1  0 | 1
0  1  1 | 1
1  0  0 | 0
1  0  1 | 1
1  1  0 | 1
1  1  1 | 1
ques1b_tb.v.txt:33: $finish called at 40 (1s)
```

Figure 4. obtained results using the Icarus Verilog tool

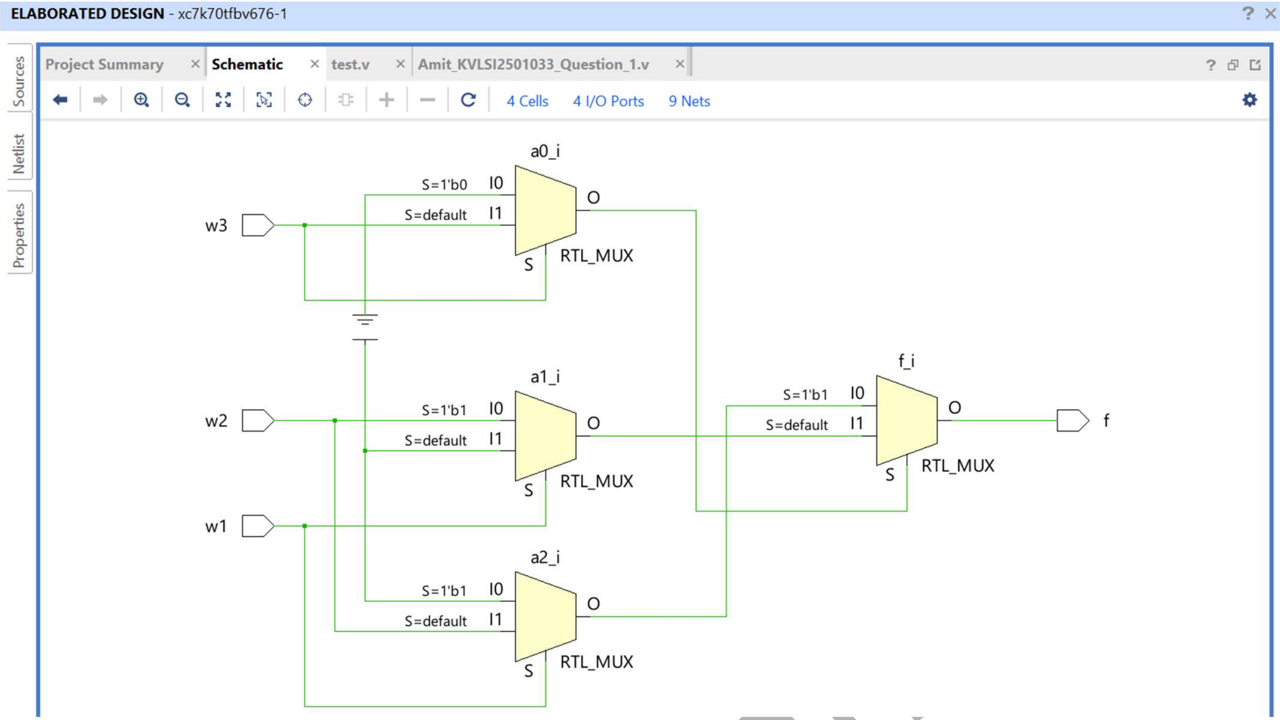


Figure 5. Vivado's output for the design (schematic)

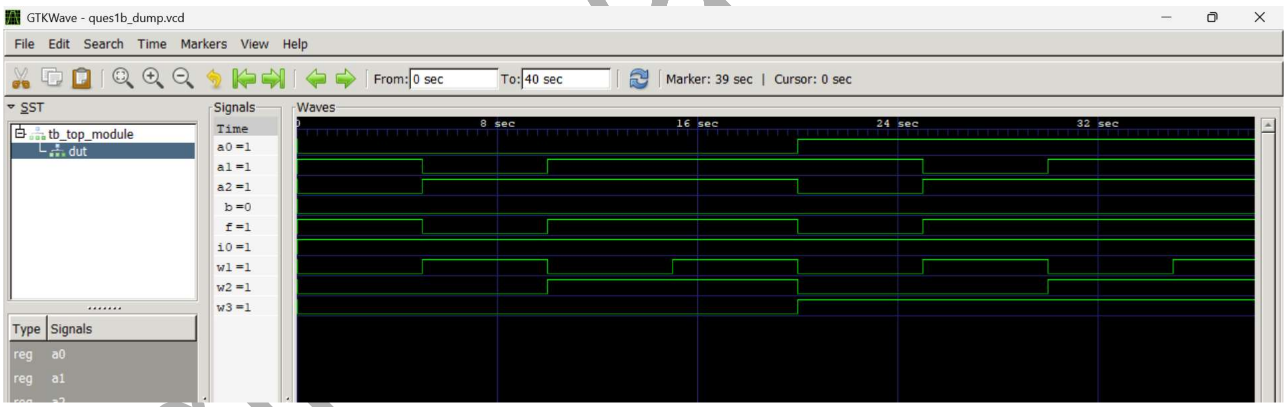


Figure 6. Waveform using GTKWave.

Links to the assignment and files :
Github:
https://github.com/amitvsuryavanshi04/amit_kvlsi_iiitb/tree/main/KVLS602-High%20Level%20Synthesis
EDA Playground:

https://edaplayground.com/x/Sj7X Question 1A

https://www.edaplayground.com/x/ri9T Question 1B