**ET-3206 DD LABORATORY**

**JOURNAL**

**EXPERIMENT - 5**

**Simulation of a combinational Circuit in Verilog HDL**

**(2:1 MUX)**

**Group No.: - *ETB-B3-G9***

**Group Members**

**1. Raghavi Narayanan(12311308)**

1. **Objectives:**
2. To write RTL code for a 2:1 Multiplexer using dataflow abstraction level.

(b) To verify the logic functionality and truth table through simulations.  
(c) To analyze the switching behavior through simulation waveforms.

1. **EDA Tool used:**

**Synopsys:**

* **VCS** – Compilation & simulation
* **Verdi** – Waveform viewing and logic analysis

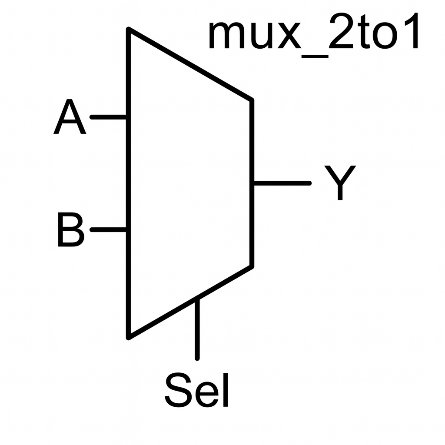
1. **Theory:**

A 2:1 Multiplexer (MUX) is a combinational circuit used to route one of two input signals to a single output line based on the value of a control signal called the select line. It functions as a digital switch: when the select input is low, the upper data input is chosen, and when the select input is high, the lower input is chosen. This selective routing allows multiplexers to be used in data selection, signal routing, arithmetic circuits, and memory access paths. Because only one input is passed to the output at a time, multiplexers help reduce hardware complexity by allowing multiple signals to share a single transmission line.

In a 2:1 multiplexer, the data inputs are labelled A and B, the select line is S, and the output is Y. The internal structure typically consists of AND gates controlled by the select line and its complement, followed by an OR gate that combines the selected data. Thus, the output always corresponds to one of the inputs based on the select signal.

**Logical Equation:**

**Symbol:**

****

**Dataflow Model:**

The behaviour of a 2:1 multiplexer maps directly to a conditional selection operation, making it ideal for dataflow modelling in Verilog. Dataflow level describes circuits that use continuous assignments and operators, rather than a gate-by-gate structure. Using the conditional operator (?:), the MUX logic can be expressed compactly and clearly as:

assign Y = (sel)? B: A;

This statement directly reflects the logical function of the multiplexer and is the most efficient way to model its behaviour in HDL.

1. **Steps in Synopsys:**
2. Create the RTL design file for the 2:1 multiplexer and save it in the working directory.
3. Create the testbench file for applying different combinations of A, B, and sel.
4. Enable FSDB waveform dumping inside the testbench so that Verdi could capture signals during simulation.
5. Open a terminal in the working directory and compile the design and testbench using Synopsys VCS.
6. Check the compilation log to ensure there are no syntax errors or unresolved modules.
7. Execute the simulation executable generated by VCS to run the testbench.
8. Confirm that the simulation is completed successfully and that an FSDB file is generated in the folder.
9. Launch Verdi and load the generated FSDB file.
10. In Verdi, add all relevant signals (A, B, sel, and Y) to the waveform viewer.
11. View the waveform transitions, zoom in on edges, and observe the switching behaviour of output Y when sel changes.
12. Verify functionality by checking that:
    * When sel = 0, the output Y follows A
    * When sel = 1, the output Y follows B
13. Compare the observed waveform with the truth table to confirm correct multiplexer behaviour.
14. Check for glitches or unexpected output transitions during signal changes.
15. Rerun the simulation if required after making corrections in the RTL or testbench files.
16. Capture observations and verify that the simulation matched the theoretical logic equation of the 2:1 MUX

**V. Simulation Analyses in Synopsys:**

**1. Functional Behaviour**

During simulation, the multiplexer output Y was observed under different combinations of inputs A, B, and the select line sel. The waveform clearly showed that the output always followed the selected data input. When sel = 0, the output tracked the value of A, and when sel = 1, the output tracked B. All transitions matched the expected behaviour of a 2:1 multiplexer without any deviation.

**2. Truth Table Verification**

By comparing the waveform with the theoretical truth table, all four possible input combinations were verified. Every simulated output value corresponded correctly to the logic equation

**Y = Sel·B + Sel'·A**,

confirming correct implementation and selection behaviour throughout the simulation run.

**3. Switching and Timing Behaviour**

The waveform displayed clean transitions for the output signal. Changes in Y occurred immediately after changes in sel or in the selected data input. No glitches or intermediate spikes were observed during the switching process. The output remained stable for the entire duration of each input condition, indicating proper combinational operation.

**4. Input-to-Output Relationship**

The propagation of values from A or B to Y was instant at the RTL level. The waveform clearly demonstrated that only the selected path was active at any given time. The unselected input did not influence the output, confirming correct multiplexer gating behaviour.

**5. Overall Analysis**

All waveform results aligned with the theoretical operation of a 2:1 multiplexer. The simulation successfully demonstrated expected logic selection, stable switching, and correct output values, validating the functionality of the RTL design.

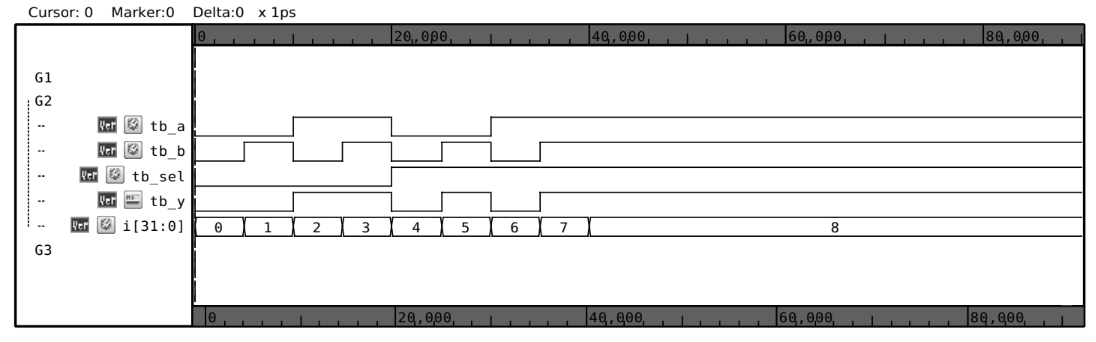
**Truth Table:**

| **S** | **A** | **B** | **Y** |
| --- | --- | --- | --- |
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 1 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 1 |

**RTL:**

****

**Waveform:**

****

**Testbench Code :**

`timescale 1ns/1ps

module mux2to1\_tb;

reg tb\_a, tb\_b, tb\_sel;

wire tb\_y;

integer i;

mux2to1 U1 (tb\_a,tb\_b,tb\_sel,tb\_y);

initial

begin

$fsdbDumpfile("mux2to1\_tb.fsdb");

$fsdbDumpvars(0,mux2to1\_tb);

for(i = 0; i<8; i = i+1)

begin

{tb\_sel,tb\_a,tb\_b} = i;

#5 $display("a = %1b, b = %1b, sel = %1b, y = %1b",tb\_a,tb\_b,tb\_sel,tb\_y);

end

#50 $finish;

end

endmodule

**Design Module Code:**

`timescale 1ns/1ps

`include "mux2to1.v"

module mux2to1(input a,b,sel, output y);

wire w1,w2;

assign w1 = (a & ~sel);

assign w2 = (b & sel);

assign y = w1 |w2 ;

endmodule

**Conclusion:**

The 2:1 Multiplexer was successfully designed and simulated using Synopsys VCS and Verdi.  
The testbench, hierarchical symbol view, and waveform analysis were performed. Simulation results matched the theoretical truth table, confirming correct MUX operation and selection behaviors.