GROUP 21 MINI PROJECT PHASE-I

PROJECT REPORT - DDS Lab (Code: CS203) - Dr. B. R. Bhowmik

Problem Statement

Design, synthesize a mux and test it for single stuck-at faults.

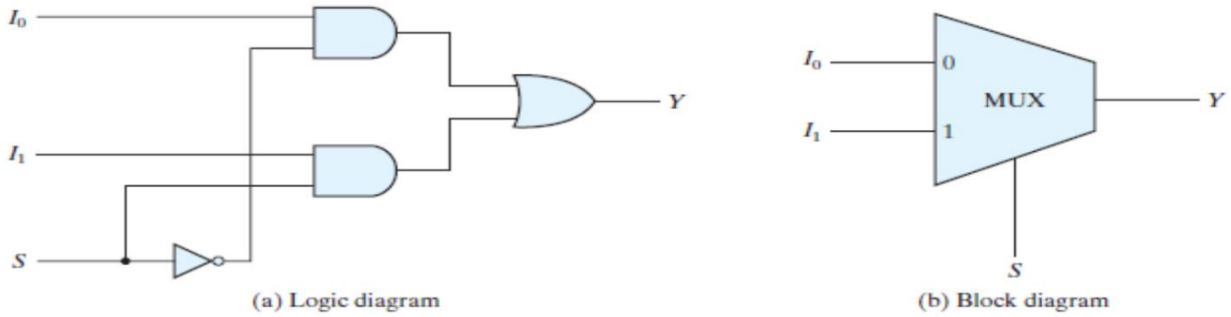
GROUP 21

TEAM MEMBERS

✓ **191CS124**	ISHAAN SINGH	**TEAM LEADER**
✓ **191CS123**	IKJOT SINGH DHODY	**TEAM MEMBER**
✓ **191CS161**	SWAPNIL GUDURU	**TEAM MEMBER**
✓ **191CS102**	AKARSHEE JAIN	**TEAM MEMBER**
✓ **191CS142**	PRACHI PRIYAM SINGH	**TEAM MEMBER**

PROBLEM STATEMENT

Suppose you have designed a logic circuit for an 2x1 mux. Logic and block diagram of a 2x1 mux are shown below.



(a) Logic diagram (b) Block diagram

Any of the input lines I0, I1, S, or the output lines from the gates (AND, NOT, OR) can be affected by a stuck-at faults. Design, synthesize this mux and test it for this fault.

INITIAL THOUGHTS

Any real-life circuit can be affected by faults. The key is to detect and correct these faults well in advance, so that faulty units are not available to the end user.

Faults that affect the functioning of the circuit can be detected and corrected with certainty and ease with simple testing methods that look for single stuck-at faults in the circuit. This technique covers almost 95% of the functional faults that affect a circuit. An opportunity to study the same, and design a mux with FPGA synthesis and analysis is the aim of the project.

Contents

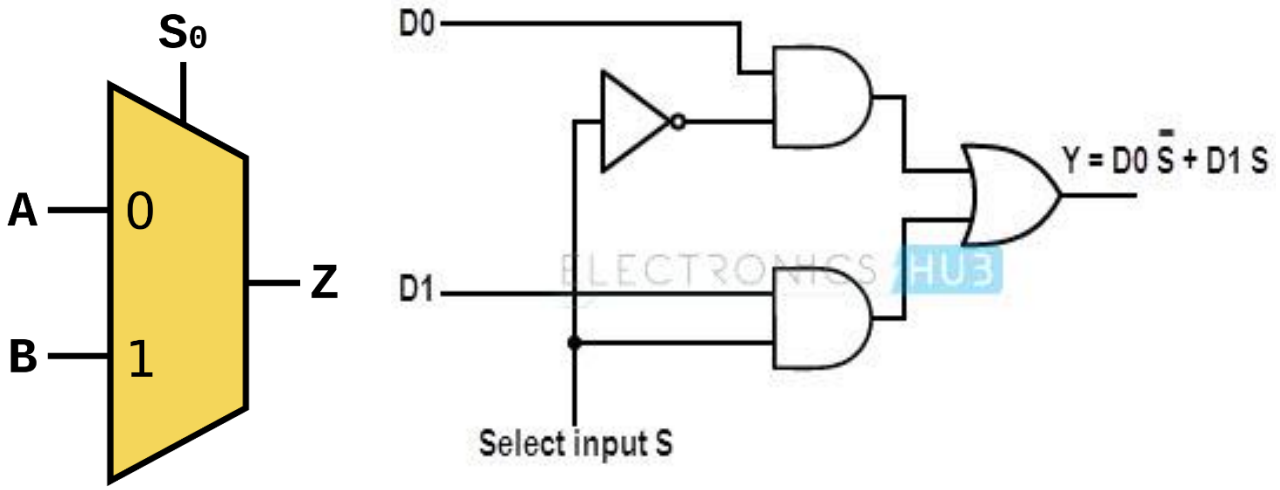
Following is the list of contents in the doc, hyperlinked to the corresponding section as well.

DESIGNING A MUX

In electronics, a multiplexer (or mux), also known as a data selector, is a device that selects between several analog or digital input signals and forwards the selected input to a single output line. The selection is directed a separate set of digital inputs known as select lines. A multiplexer of $\mathbf{2^n}$ inputs has **n** select lines, which are used to select which input line to send to the output.

One use for multiplexers is economizing connections over a single channel, by connecting the multiplexer's single output to the demultiplexer's single input. In analog circuit design, a multiplexer is a special type of analog switch that connects one signal selected from several inputs to a single output.

The schematic, logic diagram and truth table of a 2-to-1 mux is given below.

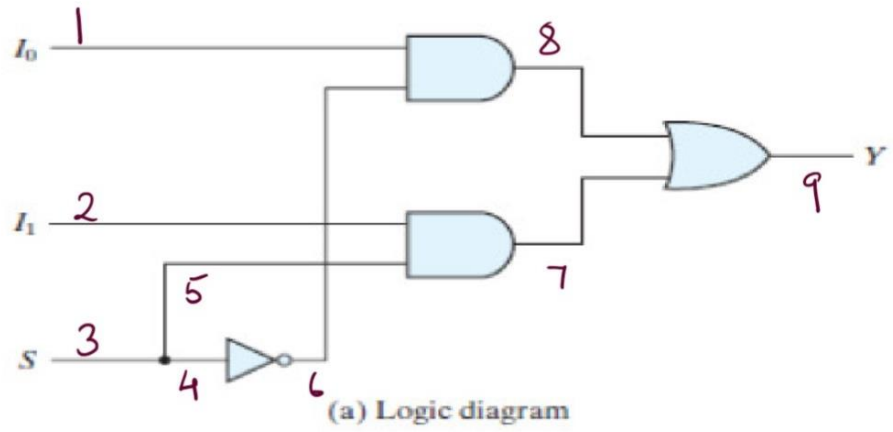


SELECT LINE (S)	INPUTS (I0 and I1)		OUTPUT (O)
0	**0**	0	***0***
0	**0**	1	***0***
0	**1**	0	***1***
0	**1**	1	***1***
1	0	**0**	***0***
1	0	**1**	***1***
1	1	**0**	***0***
1	1	**1**	***1***

IMPLEMENTING A MUX – Xilinx

Using Xilinx, the processing of developing a logic circuit is over-simplified. All we are required to do is write the Verilog HDL code for the module. The software designs and analyses the code to develop reports that provide key insights into the design and aesthetics of the design, its power utilisation, input-output port analysis, and so on.

In the problem given to us, we have a multiplexer circuit, which we have to test for the stuck at faults for either 0 or 1 at each and every wire, by designing a testbench for the same. We first examined the circuit's logic diagram. As you can see below, there were 9 wires.



(a) Logic diagram

Given below is the Verilog code to implement the 2-to-1 MUX.

```
module mux_proper (S, I0, I1, out);
   input S, I0, I1;
   output [8:0] out;
   assign out [0] = I0;
   assign out [1] = I1;
   assign out [2] = S;
   assign out [3] = S;
   assign out [4] = S;
   assign out [5] = ~S;
   assign out [6] = I1 & S;
   assign out [7] = I0 & (~S);
   assign out [8] = (I1 & S) | (I0 & (~S));
endmodule
```

In this module code above,

- ✓ **S** – Select line for the 2-to-1 MUX
- ✓ **I0 and I1** – Input Lines for the MUX
- ✓ **Out** – A 9-bit Output value that indicates the status of every wire

The key element of the circuit is the **out** element. This 9-bit output from the module contains 0-indexed voltage information about each of the 9 wires in the circuit. For example, **out [0] represents the voltage level (high / low) of Wire 1 in the diagram.**

Since the circuit has 9 wires, we need to create 18 + 1 modules to provide an exhaustive test for our testbench, which are

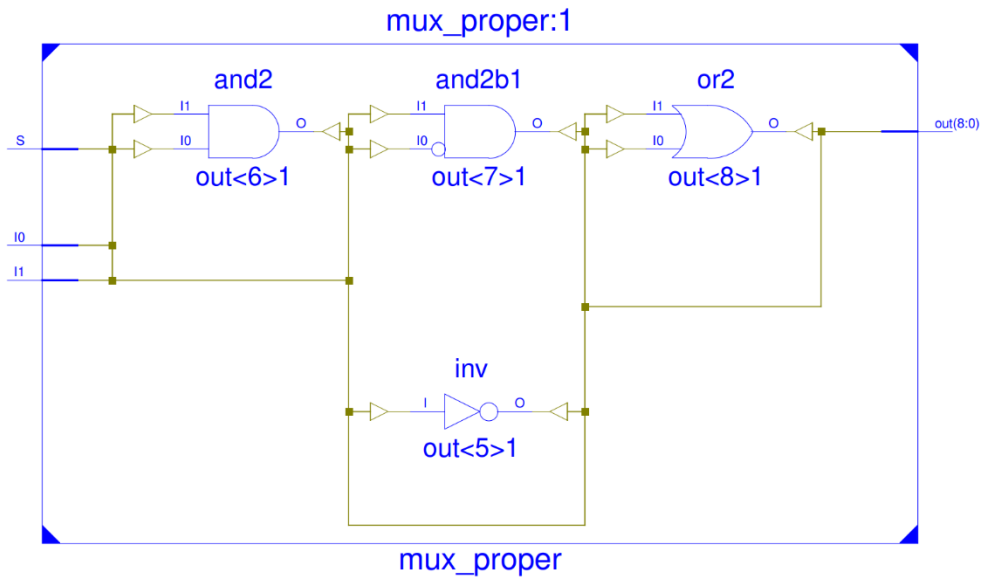
- ✓ **18 modules** : 9 wires each having 2 stuck at faults **(Stuck at 1, Stuck at 0)**
- ✓ **1 module** : Actual representation of a multiplexer **(mux_proper)**

If our testbench is able to conclusively tell exactly which wire and which fault is present in that particular wire of every testcase above, our project is complete and the study concludes.

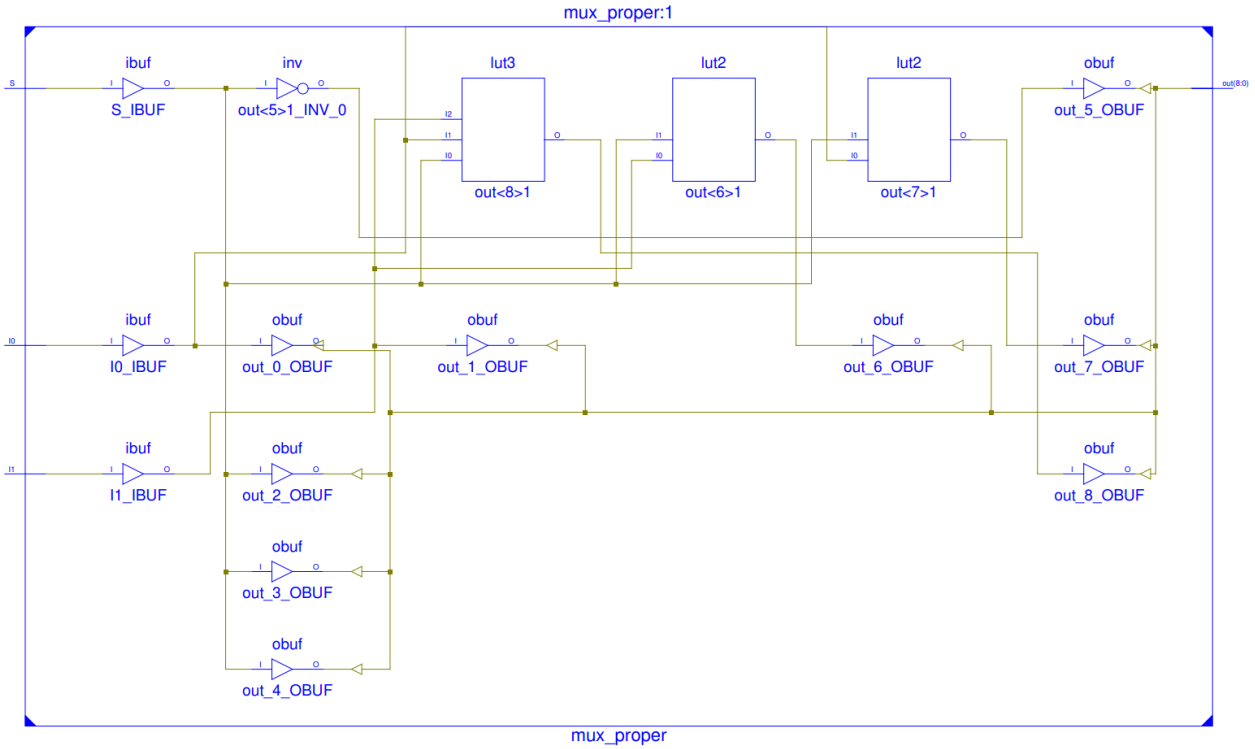
ANALYSIS OF THE MODULE

On generating the synthesis reports for the MUX module code shown above, we get some detailed files that comment on our design. All files can be found in the attached submission directory.

LOGIC DIAGRAM OF THE MODULE



LUT DIAGRAM FOR THE MODULE



DESIGN SUMMARY

- ✓ **Top Level Output File Name:** **mux_proper.ngc**
- ✓ **Selected Device:** **7a100tcsg324-3**
- ✓ **Slice Logic Utilization:**
  - **Number of Slice LUTs:** 4 out of 63400 0%
    - Number used as Logic: 4 out of 63400 0%
- ✓ **Slice Logic Distribution:**
  - **Number of LUT Flip Flop pairs used:** 4
    - Number with an unused Flip Flop: 4 out of 4 100%
    - Number with an unused LUT: 0 out of 4 0%
    - Number of fully used LUT-FF pairs: 0 out of 4 0%
    - Number of unique control sets: 0
- ✓ **IO Utilization:**
  - **Number of IOs:** 12
  - **Number of bonded IOBs:** 12 out of 210 5%

TIMING REPORT

- ✓ Speed Grade: -3
  - Minimum period: No path found
  - Minimum input arrival time before clock: No path found
  - Maximum output required time after clock: No path found
  - Maximum combinational path delay: 0.917ns
- ✓ Timing Details: All values displayed in nanoseconds (ns)
  - Timing constraint: Default path analysis
  - Total number of paths / destination ports: 13 / 9
- ✓ Delay: 0.917ns (Levels of Logic = 3)
  Source: S (PAD)
  Destination: out<8> (PAD)
  Data Path: S to out<8>

Cell:in->out	fanout	Gate Delay	Net Delay	Logical Name
IBUF:I->O	7	0.001	0.539	S_IBUF (out_2_OBUF)
LUT3:I0->O	1	0.097	0.279	out<8>1 (out_8_OBUF)
OBUF:I->O		0.000	out_8_OBUF (out<8>)	
Total		0.917ns	(0.098ns logic, 0.819ns route) (10.7% logic, 89.3% route)	

POWER REPORT

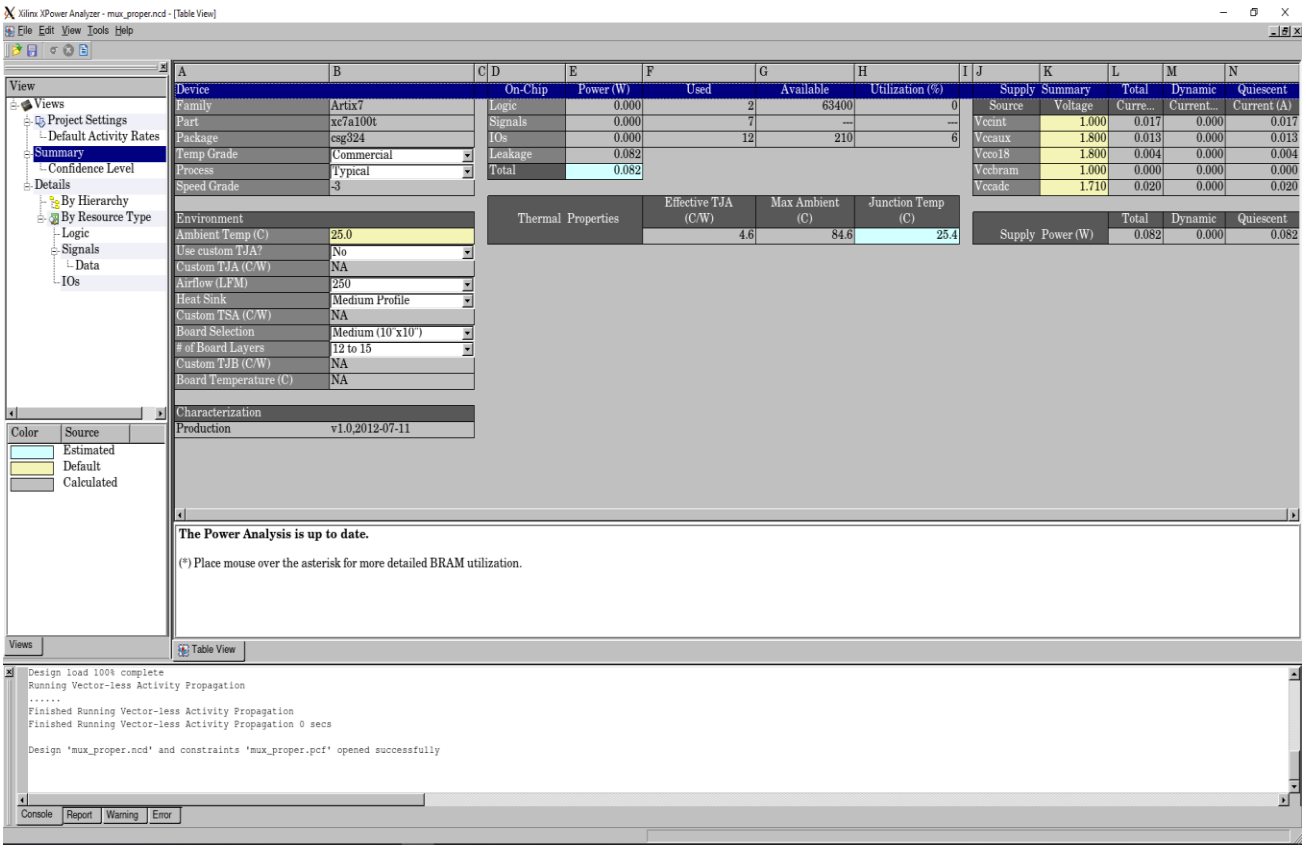
- ✓ **Environment**
  - Ambient Temp (C) 25.0
  - Use custom TJA? No
  - Custom TJA (C/W) NA
  - Airflow (LFM) 250
  - Heat Sink Medium Profile
  - Custom TSA (C/W) NA
  - Board Selection Medium (10"x10")
  - # of Board Layers 12 to 15
  - Custom TJB (C/W) NA
  - Board Temperature (C) NA

- ✓ **Default Activity Rates**
  - FF Toggle Rate (%) 12.5
  - I/O Toggle Rate (%) 12.5
  - Output Load (pF) 5.0
  - I/O Enable Rate (%) 100.0
  - BRAM Write Rate (%) 50.0
  - BRAM Enable Rate (%) 50.0
  - DSP Toggle Rate (%) 12.5

- ✓ **Thermal Summary**
  - Effective TJA (C/W) 4.6
  - Max Ambient (C) 84.6
  - Junction Temp (C) 25.4

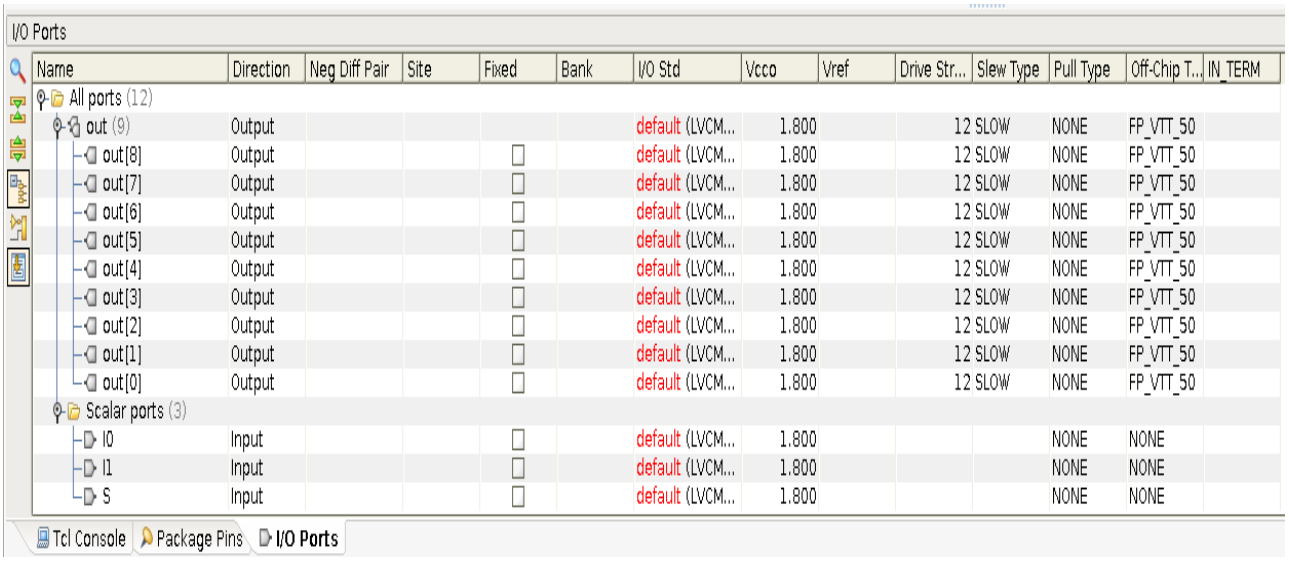
- ✓ **Power Supply Summary**

	Total	**Dynamic**	**Static Power**
Supply Power (mW)	82.16	0.00	82.16

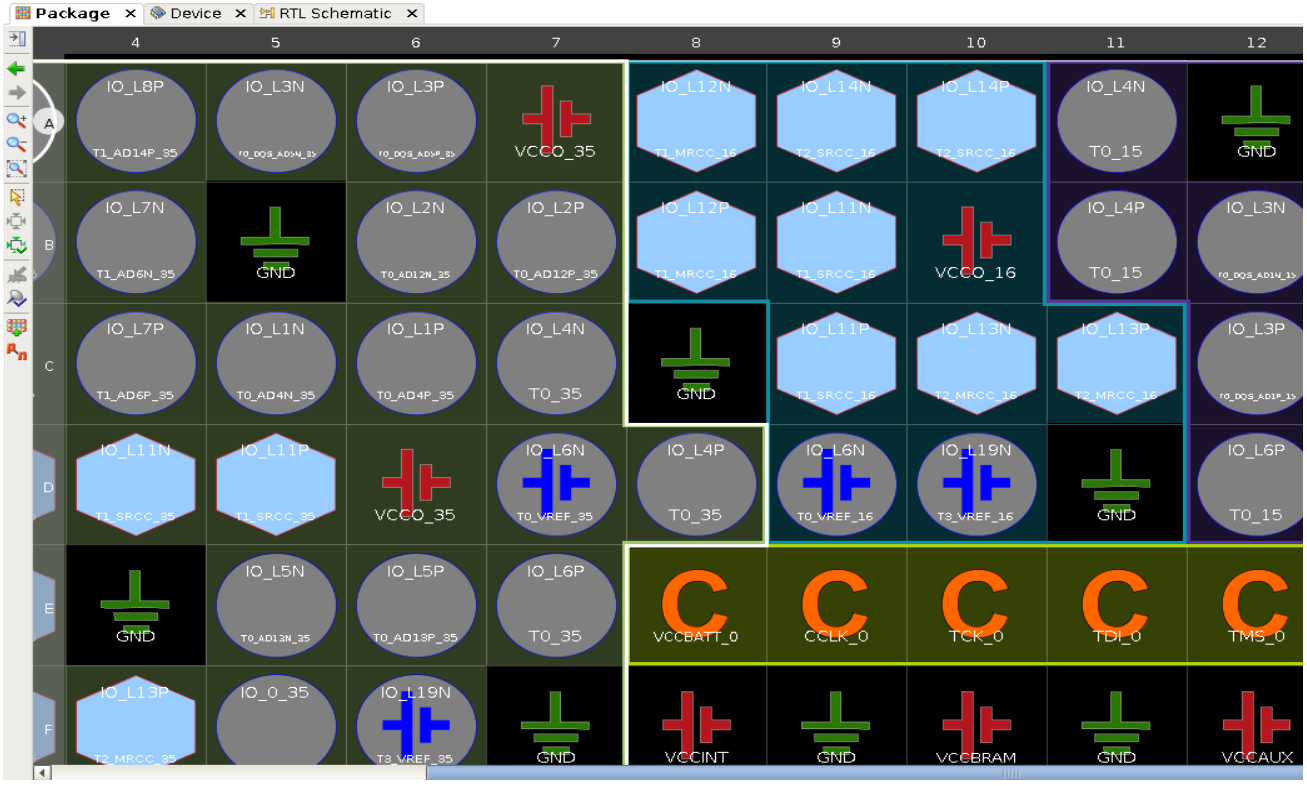


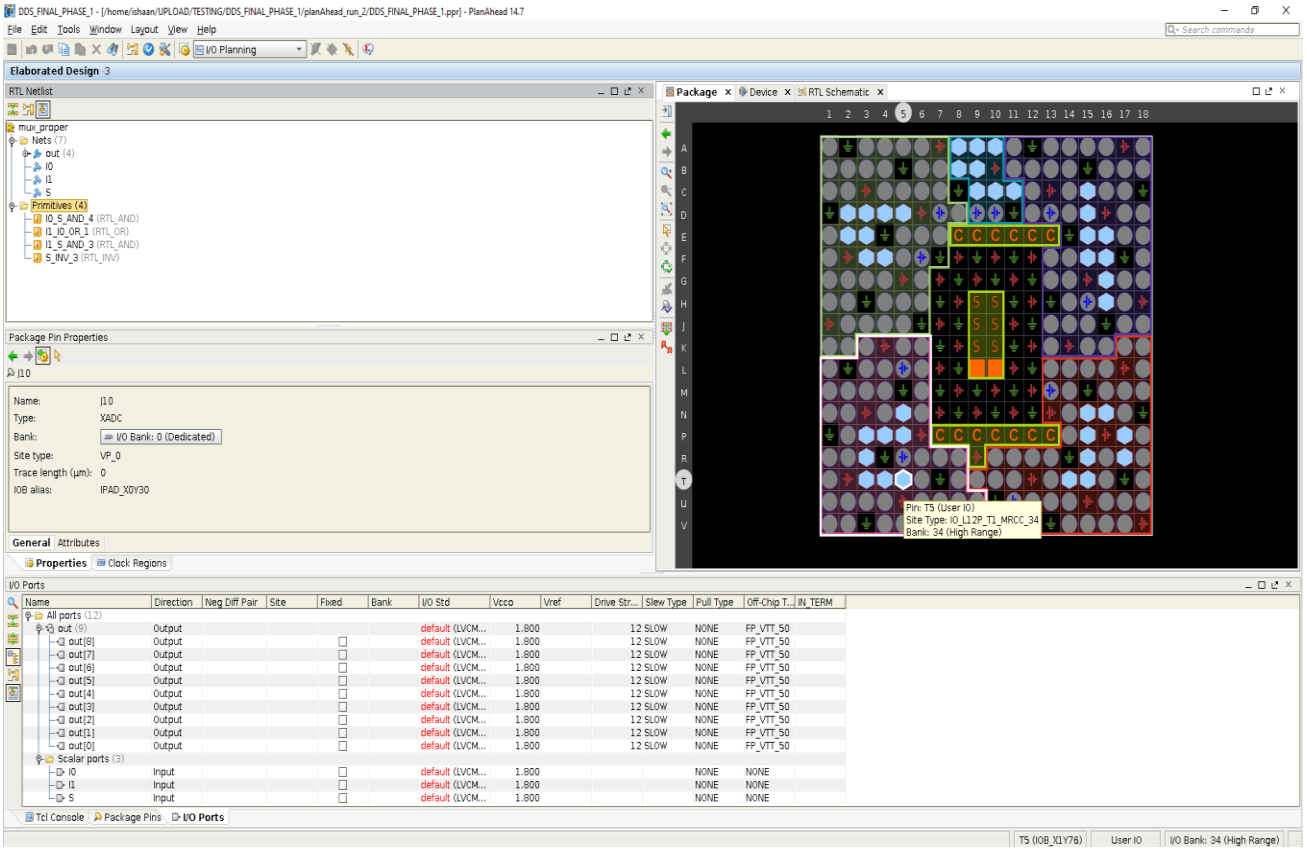
FPGA SYNTHESIS BOARD

To synthesise the Circuit using FPGA Library, we shall use the Plan Ahead 14.7 by Xilinx to produce FPGA Synthesis. We shall assign the I/O terminals of our design to different sites as shown in the picture below.

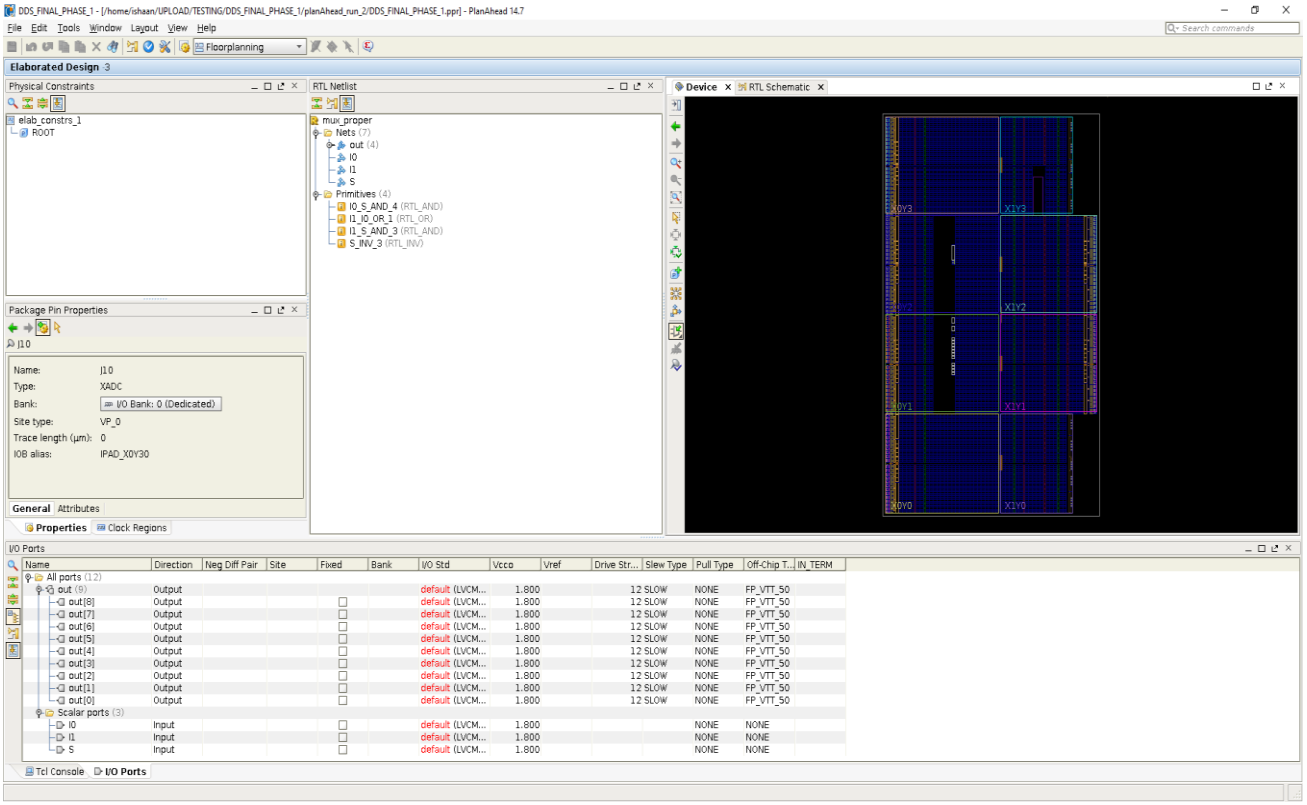


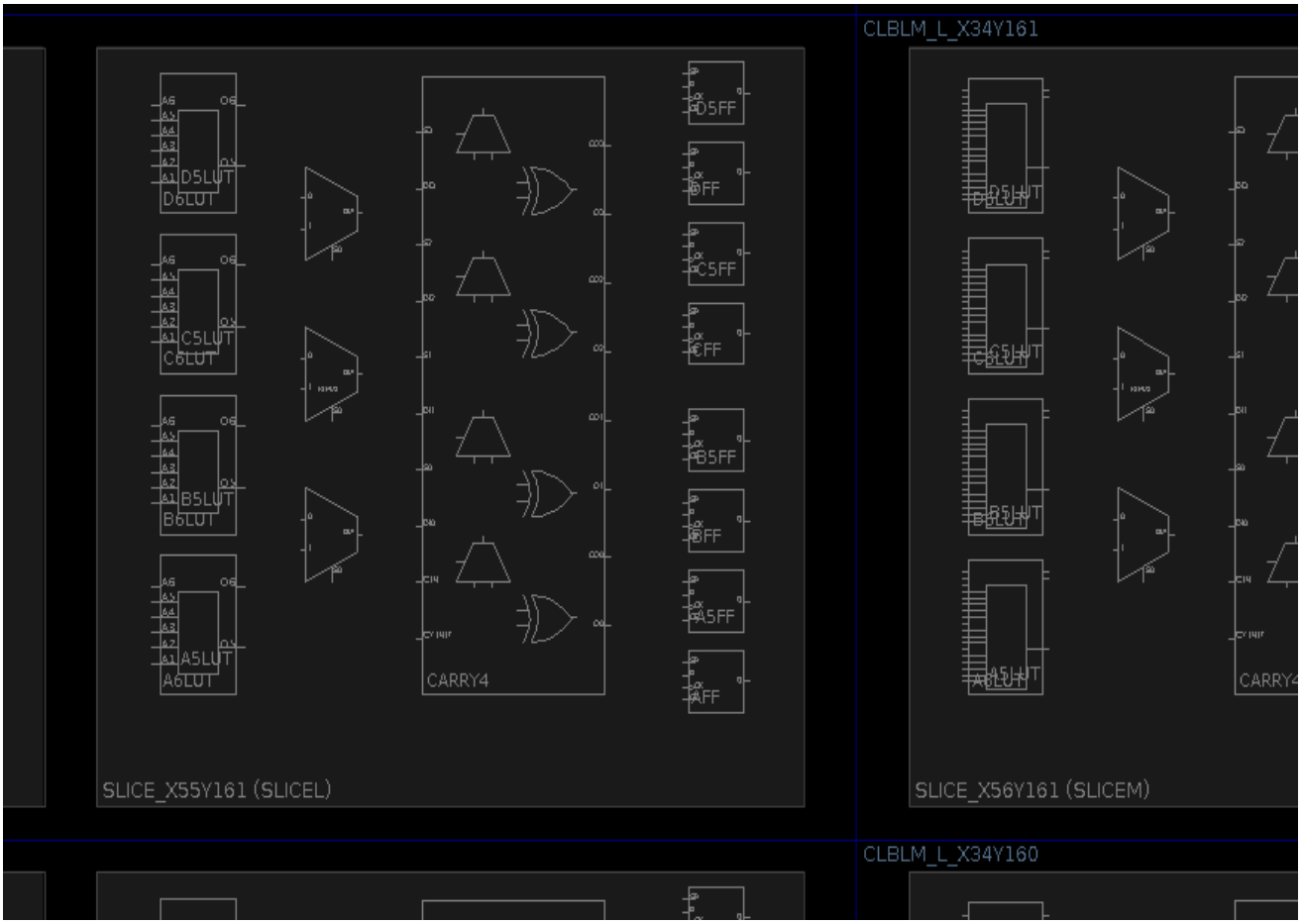
After assigning each input and output of our design to each port, the final chip would look like the following in the below two images. The first image is the magnified image of chip with the assigned sites to each input and output visible and the second image is a zoomed-out image.





Xilinx also provides us with the Floor Plan analysis of the chip, that looks as follows. The first image is the zoomed-out image of chip floor and the second image is a zoomed-in image of a part of the circuitry.





This design and robust analysis provided by Xilinx is of utmost use to a manufacturer as it helps the person in charge design and pre-test the circuit for faults and correctness.

APPROACH TO SOLUTION

We solve this problem in the following steps:

- ✓ We take an input set, i.e., the current values of I0, I1, S, and we generate a 9-bit array **(out)** from our **(mux_proper)** module. We call this our **'golden'** array because it consists of the correct values for each wire, for our current input.
- ✓ The 9-bit array **(out)** represents the state of the wire at each index as shown in the logic diagram earlier. We compare this with our golden array. If it is equal, then there is no issue. But if there is a mismatch, then we are looking at a fault in the circuit.
- ✓ When a fault is detected in this manner, we can conclusively tell which wire and what fault in the wire was detected.

The following section contains a more rigorous description of the same and proof of concept.

Here is the Verilog Testbench for our problem. It has been broken into parts with a small description about every piece of the code, highlighting the key features and usage of the testbench for testing 2-to-1 MUX modules for single stuck-at faults.

INITIALISATION

```
module test_mux;

    // Inputs
    reg S;
    reg I0;
    reg I1;

    // Outputs
    wire [8:0] out;
    reg [8:0] golden = 9'b111111111; //'
    integer index, i;

    // Instantiate the Unit Under Test (UUT)
    mux_proper uut (.S(S), .I0(I0), .I1(I1), .out(out));
```

In this section of the testbench, we declare our module net and instantiate the UUT we wish to test.

In this section of the code, the key element is the UUT being initialised. In this example, it says **mux_proper** which is the mux implementation without any stuck-at faults.

As part of the package provided with this submission, we have included Verilog modules that will generate a specific stuck-at fault, for all wires.

Hence, just changing the name of the UUT being instantiated, you could test our testbench against a module that has a particular fault.

For example, instantiating **mux_w1_0** instead of **mux_proper** will allow you to test an implementation of the MUX that has a stuck-at 0 fault in wire 0 as per our logic diagram, 0 indexed.

Feel free to play around with all the test cases for all 9 wires and observe the simulation and the output, as we explain in a later section.

THE TEST AND PRINT LOGIC

```
// add checks
task test_current_config; begin
    if (out != golden) begin
        update_task;
    end
  end
endtask

task display_task; begin
    $display("Error is in the wire - %d", index);
    $display("The wire is stuck at %b", (~golden[index]));
    $finish;
  end
endtask

task update_task; begin
    index = -1;
    for (i=0; i<9; i=i+1) begin
        if (out[i] != golden[i]) begin
            index = i;
            display_task;
            $finish;
        end
    end
  end
endtask

task update_golden; begin
    golden[0] = I0; golden[1] = I1;
    golden[2] = S;  golden[3] = S;
    golden[4] = S;  golden[5] = ~S;
    golden[6] = I1 & S; golden[7] = I0 & (~S);
    golden[8] = (I1 & S) | (I0 & (~S));
  end
endtask
```

This section of the testbench focuses on the main logic behind the test, simulation and the output generation.

Following is a run-through of the code.

- ✓ We get the output from a particular module in the 9-bit array – out. (This array contains state of wires from the module which we are testing currently).
- ✓ We then compare this 9-bit value with our golden array in the task called **test_current_config**.
- ✓ If it is equal, then there is no issue.
- ✓ But if there is a mismatch, then we go to the task **update_task**.
- ✓ In the **update_task** we traverse through the out and golden arrays, to get the index of the first mismatched bit in the out and golden arrays.
- ✓ Since we are checking every possible test case, i.e., all the 8 test cases for I0, I1 and S, we will encounter a case in which the fault is detected.
- ✓ Hence, we keep going through the test cases till such a case is encountered and then print the value of the index where the fault is detected and the nature of the fault in the **display_task**.
- ✓ If everything goes smoothly, that can only mean that our circuit and all its wires are free from single stuck-at-faults.

THE TESTCASES FOR THE TESTBENCH

```
  initial begin
      #0 S=0; I0=0; I1=0; update_golden;
      #5 test_current_config;
      #0 I0=0; I1=0; S=1; update_golden;
      #5 test_current_config;
      #0 I0=0; I1=1; S=0; update_golden;
      #5 test_current_config;
      #0 I0=0; I1=1; S=1; update_golden;
      #5 test_current_config;
      #0 I0=1; I1=0; S=0; update_golden;
      #5 test_current_config;
      #0 I0=1; I1=0; S=1; update_golden;
      #5 test_current_config;
      #0 I0=1; I1=1; S=0; update_golden;
      #5 test_current_config;
      #0 I0=1; I1=1; S=1; update_golden;
      #5 test_current_config;
      #0 $display("No errors in the module."); $finish;
  end
endmodule
```

ADVANTAGES AND DISADVANTAGES – APPROACH

Advantages:

The major advantage, as seen clearly is that the complexity is greatly reduced. Many different physical defects may be modelled by the same logical stuck-at fault. SSF is technology independent that has been successfully used on TTL, ECL, CMOS, etc.

Apart from this, stuck-at tests also cover a large percentage of multiple stuck-at faults (95%).

Stuck-at tests cover a large percentage of unmodeled physical defects.

- ✓ It has been proven that Stuck-at-fault is good for defects within a cell.
  - Any lower-level model is considered too complex and inaccurate.
- ✓ Good for defects outside of a cell.
  - Bridges easy to cover without explicit targeting.

Disadvantages:

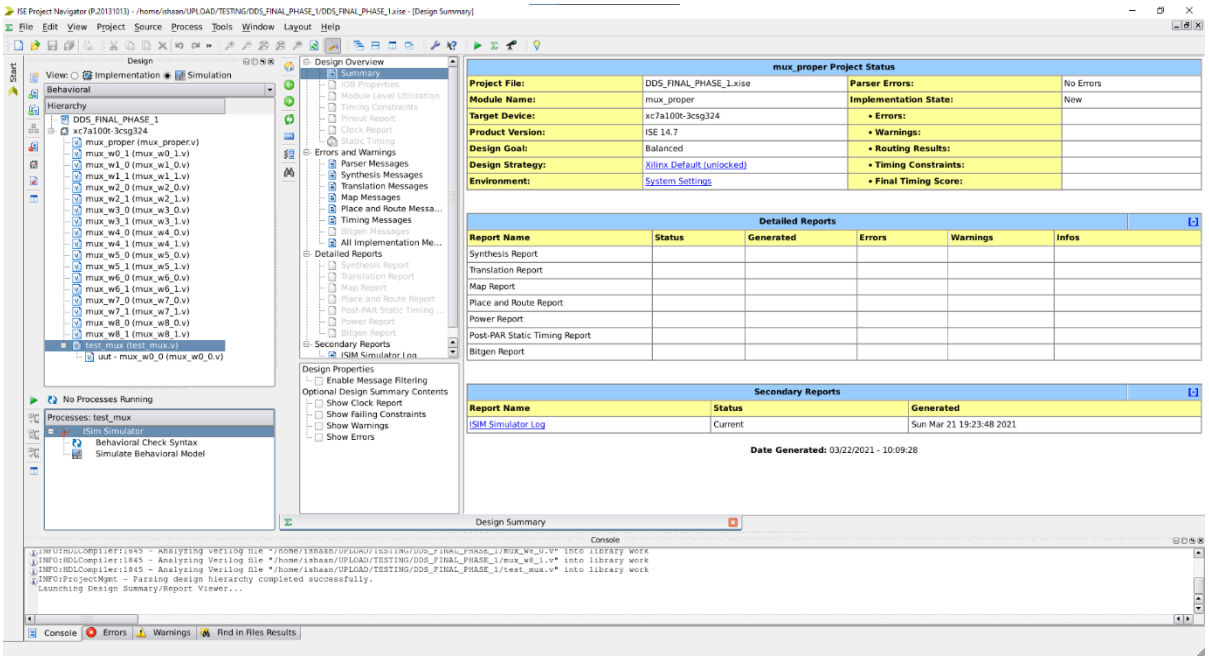
Though the stuck-at-fault has numerous advantages, it has few disadvantages as well which may occur due to various unforeseen reasons. Some of them have been listed here.

Combinational Redundancy – Sometimes a fault, say “f” on line l cannot be excited or cannot be propagated or both. Then the fault f is termed untestable. If the fault f is untestable, then the fault f is redundant, i.e., the line l or the associated gate can be removed from the circuit without changing the logic function.

Unintentional redundancies occur due to poorly optimized designs. This can happen for hand designed or synthesized circuits. Interconnect of synthesized individually non-redundant logic blocks can create global redundancies.

SIMULATION PROCEDURE

1. Unzip the **project_archive** and open it in Xilinx with **Open Project** option.
2. Initially, the test is set to run simulation for **mux_proper** module. To see the output waveform for the given test cases in the testbench code, the **Isim Simulator** is employed. For starting the simulation, clicking on **Behavioral Check Syntax and followed by Simulate Behavioral Model** starts the Isim Simulator and gives the output waveform. The location of the **Behavioral Check Syntax** and **Simulate Behavioral Model** on Xilinx Design Suite ISE 14.7 is on the left side of the screen as shown in the image below.



3. A green tick sign after the **Behavioral Check Syntax** is an indication that there aren't any errors on the testbench Verilog code written and now is ready for simulation to generate waveforms.
4. The console displays no errors as shown in diagram below as expected as this is the proper implementation of the mux.

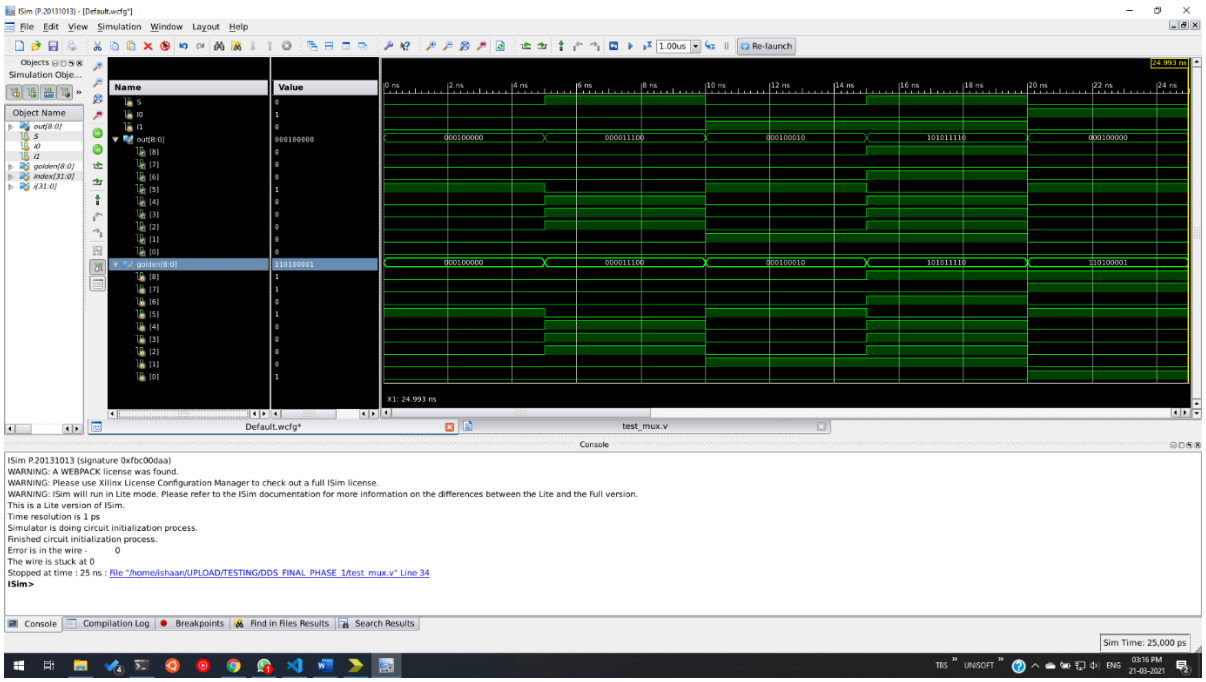
```
Started : "Behavioral Check Syntax".
Determining files marked for global include in the design...
Running vlogcomp...
Command Line: vlogcomp -work isim_temp -intstyle ise -prj /home/ishaan/UPLOAD/TESTING/DDS_FINAL_PHASE_1/test_mux_stx_beh.prj
Determining compilation order of HDL files
Analyzing Verilog file "/home/ishaan/UPLOAD/TESTING/DDS_FINAL_PHASE_1/mux_w0_0.v" into library isim_temp
Analyzing Verilog file "/home/ishaan/UPLOAD/TESTING/DDS_FINAL_PHASE_1/test_mux.v" into library isim_temp
Analyzing Verilog file "/opt/Xilinx/14.7/ISE_DS/ISE//verilog/src/glbl.v" into library isim_temp

Process "Behavioral Check Syntax" completed successfully
```

5. Now, to verify the stuck at faults at the remaining 9 sites, in line 14 of **test_mux**, change it to the required wire and corresponding stuck-at-fault as shown in

diagram below. In the given diagram, we test for stuck at fault of wire zero stuck at 0. Similarly, to test for a wire **i** stuck at fault **j** , initialize the **uut mux_wi_j**.

```
1  module test_mux;
2
3      // Inputs
4      reg S;
5      reg I0;
6      reg I1;
7
8      // Outputs
9      wire [8:0] out;
10     reg [8:0] golden = 9'b111111111; //'
11     integer index, i;
12
13     // Instantiate the Unit Under Test (UUT)
14     mux_w0_0 uut (
15         .S(S),
16         .I0(I0),
17         .I1(I1),
18         .out(out)
19     );
20
21     // add checks
22     task test_current_config;
23     begin
24         if (out != golden) begin
25             update_task;
26         end
```

6. Now, we click on **Simulate Behavioural Model** under processes toolbar again to verify output for required wire and corresponding stuck-at-fault. The console shows the wire with single stuck at fault and the corresponding fault.

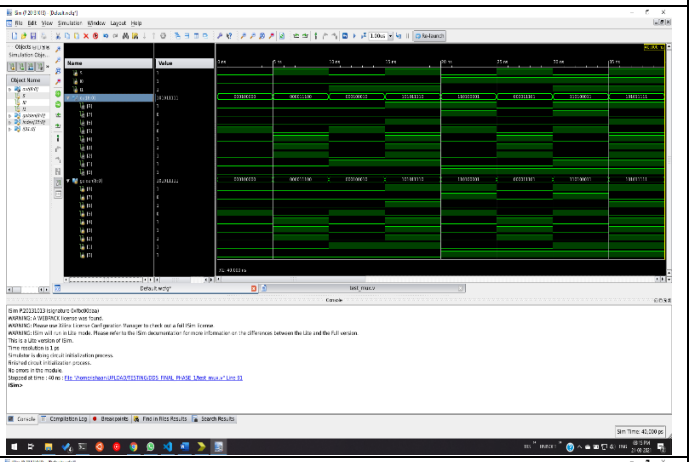


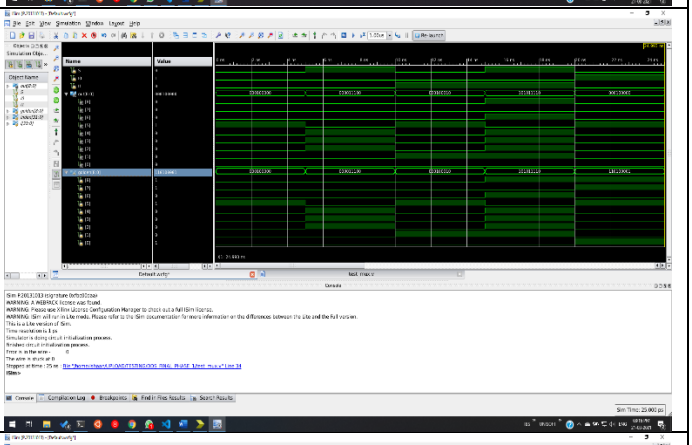
7. Test around with all 18 modules corresponding to 9 wires and 2 faults each.

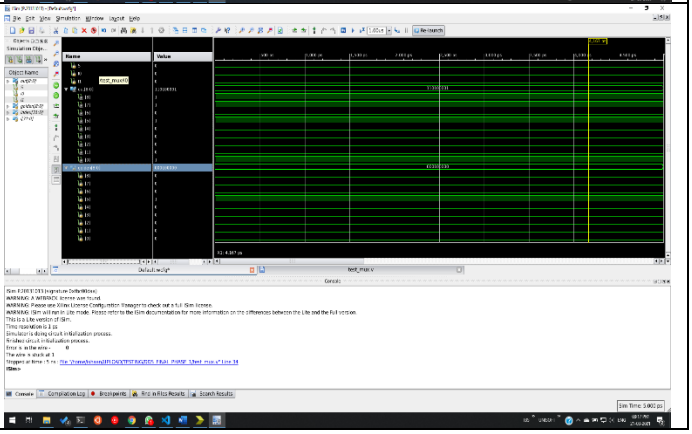
SIMULATION RESULTS

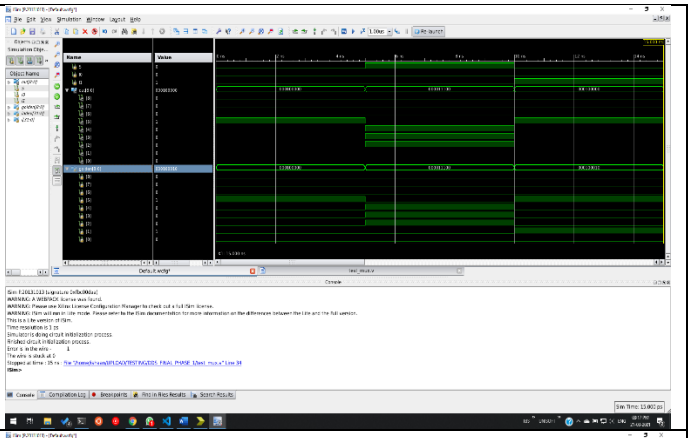
We ran out testbench on all possible faults in the circuit. The results for the same are attached below along with the code for the module alongside. The same results are expected on any system irrespective of architecture.

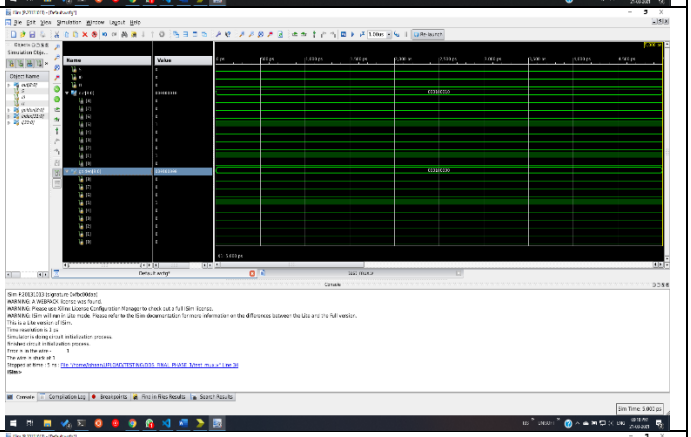
FAULT	MODULE CODE	OUTPUT

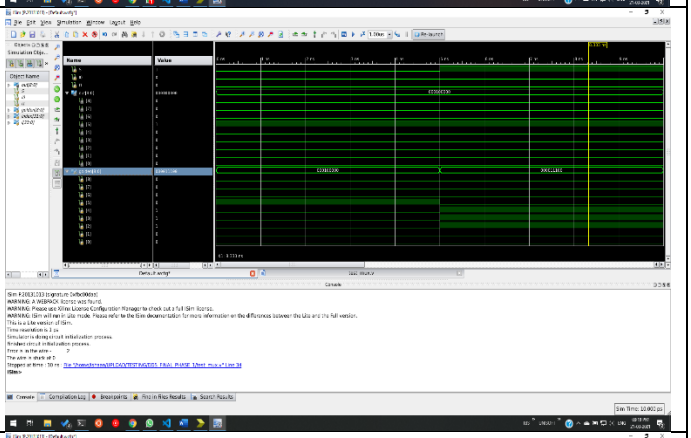
NO FAULTS	`module mux_proper(S,I0,I1,out);`
	`input S, I0, I1;`
	`output [8:0] out;`
	`assign out[0] = I0;`
	`assign out[1] = I1;`
	`assign out[2] = S;`
	`assign out[3] = S;`
	`assign out[4] = S;`
	`assign out[5] = ~S;`
	`assign out[6] = I1 & S;`
	`assign out[7] = I0 & (~S);`
	`assign out[8] =`
	`(I1 & S) | (I0 & (~S));`
	`endmodule`
		

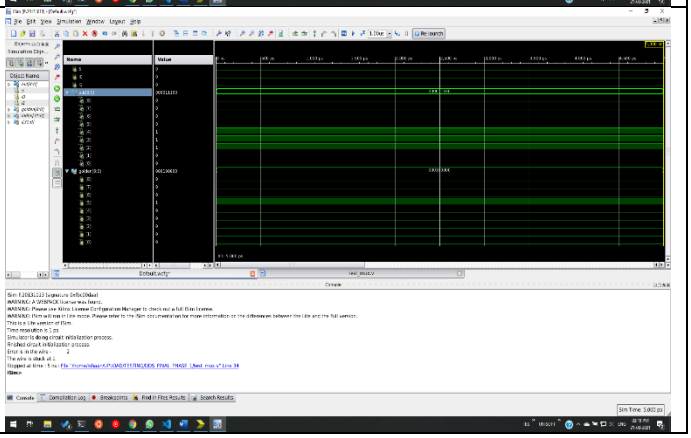
WIRE 0 SSA-0	`module mux_w0_0(S,I0,I1,out);`
	`input S, I0, I1;`
	`output [8:0] out;`
	`assign out[0] = 0;`
	`assign out[1] = I1;`
	`assign out[2] = S;`
	`assign out[3] = S;`
	`assign out[4] = S;`
	`assign out[5] = ~S;`
	`assign out[6] = I1 & S;`
	`assign out[7] = 0 & (~S);`
	`assign out[8] =`
	`(I1 & S) | (0 & (~S));`
	`endmodule`
		

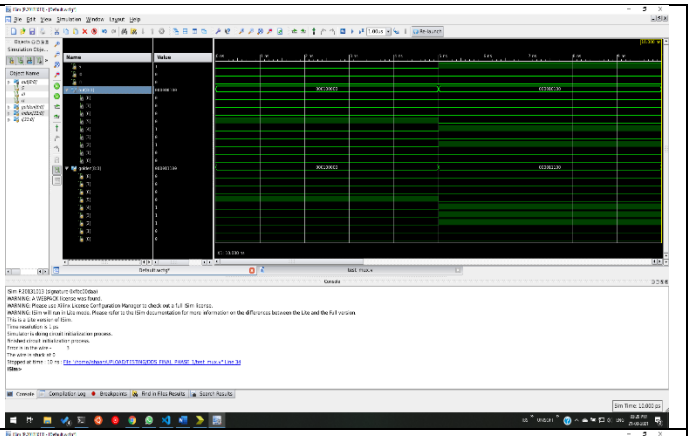
WIRE 0 SSA-1	`module mux_w0_1(S,I0,I1,out);`
	`input S, I0, I1;`
	`output [8:0] out;`
	`assign out[0] = 1;`
	`assign out[1] = I1;`
	`assign out[2] = S;`
	`assign out[3] = S;`
	`assign out[4] = S;`
	`assign out[5] = ~S;`
	`assign out[6] = I1 & S;`
	`assign out[7] = 1 & (~S);`
	`assign out[8] =`
	`(I1 & S) | (1 & (~S));`
	`endmodule`
		

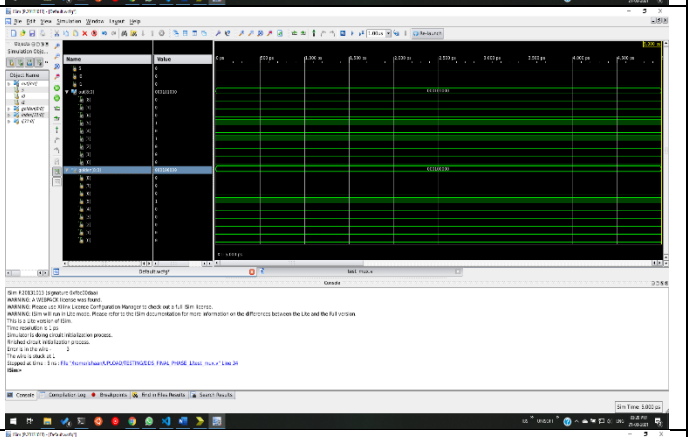
WIRE 1 SSA-0	`module mux_w1_0(S,I0,I1,out);`
	`  input S, I0, I1;`
	`  output [8:0] out;`
	`  assign out[0] = I0;`
	`  assign out[1] = 0;`
	`  assign out[2] = S;`
	`  assign out[3] = S;`
	`  assign out[4] = S;`
	`  assign out[5] = ~S;`
	`  assign out[6] = 0 & S;`
	`  assign out[7] = I0 & (~S);`
	`  assign out[8] =`
	`  (0 & S) | (I0 & (~S));`
	`endmodule`
		

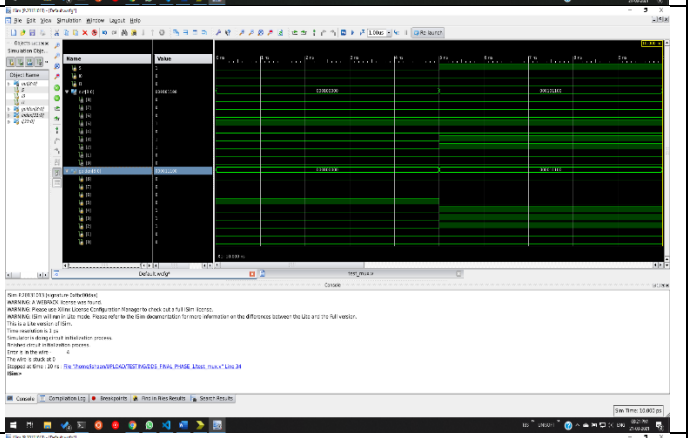
WIRE 1 SSA-1	`module mux_w1_1(S,I0,I1,out);`
	`  input S, I0, I1;`
	`  output [8:0] out;`
	`  assign out[0] = I0;`
	`  assign out[1] = 1;`
	`  assign out[2] = S;`
	`  assign out[3] = S;`
	`  assign out[4] = S;`
	`  assign out[5] = ~S;`
	`  assign out[6] = 1 & S;`
	`  assign out[7] = I0 & (~S);`
	`  assign out[8] =`
	`  (1 & S) | (I0 & (~S));`
	`endmodule`
		

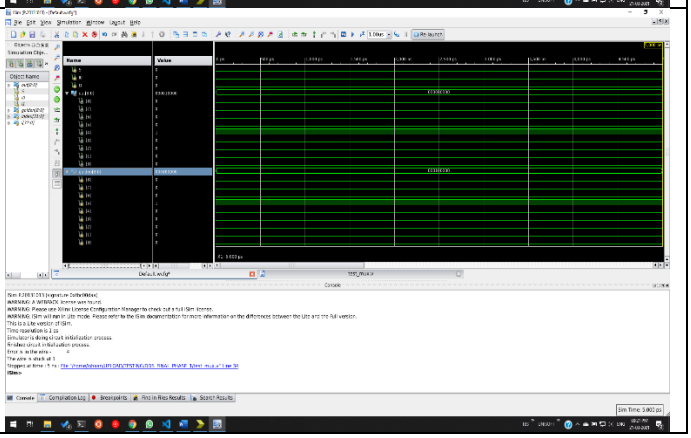
WIRE 2 SSA-0	`module mux_w2_0(S,I0,I1,out);`
	`  input S, I0, I1;`
	`  output [8:0] out;`
	`  assign out[0] = I0;`
	`  assign out[1] = I1;`
	`  assign out[2] = 0;`
	`  assign out[3] = 0;`
	`  assign out[4] = 0;`
	`  assign out[5] = ~0;`
	`  assign out[6] = I1 & 0;`
	`  assign out[7] = I0 & (~0);`
	`  assign out[8] =`
	`  (I1 & 0) | (I0 & (~0));`
	`endmodule`
		

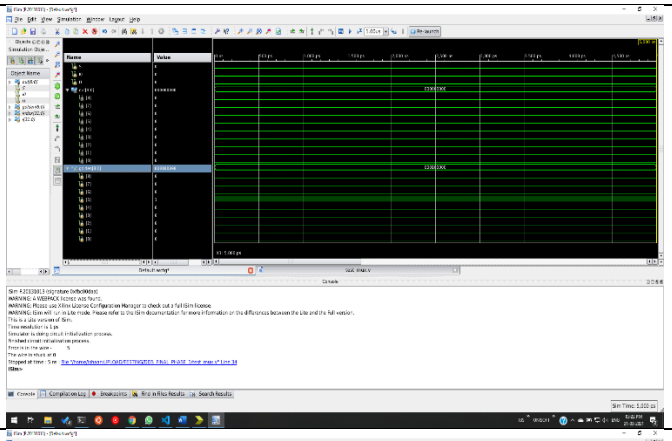
WIRE 2 SSA-1	`module mux_w2_1(S,I0,I1,out);`
	`  input S, I0, I1;`
	`  output [8:0] out;`
	`  assign out[0] = I0;`
	`  assign out[1] = I1;`
	`  assign out[2] = 1;`
	`  assign out[3] = 1;`
	`  assign out[4] = 1;`
	`  assign out[5] = ~1;`
	`  assign out[6] = I1 & 1;`
	`  assign out[7] = I0 & (~1);`
	`  assign out[8] =`
	`  (I1 & 1) | (I0 & (~1));`
	`endmodule`
		

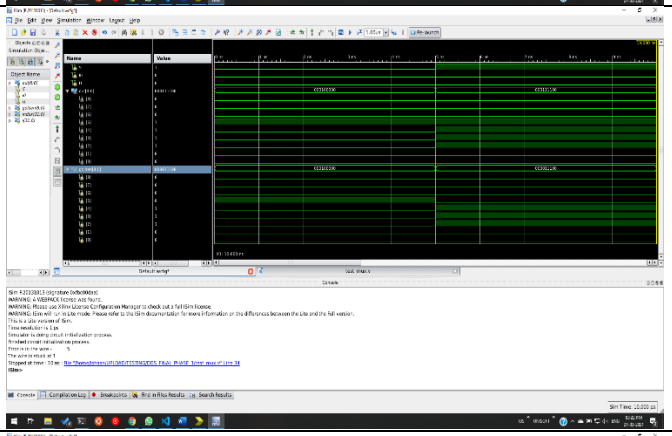
WIRE 3 SSA-0	`module mux_w3_0(S,I0,I1,out);`
	`  input S, I0, I1;`
	`  output [8:0] out;`
	`  assign out[0] = I0;`
	`  assign out[1] = I1;`
	`  assign out[2] = S;`
	`  assign out[3] = 0;`
	`  assign out[4] = S;`
	`  assign out[5] = ~S;`
	`  assign out[6] = I1 & 0;`
	`  assign out[7] = I0 & (~S);`
	`  assign out[8] =`
	`  (I1 & 0) | (I0 & (~S));`
	`endmodule`
		

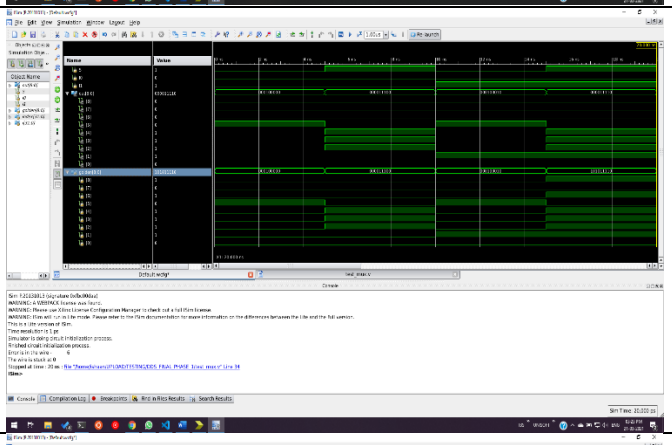
WIRE 3 SSA-1	`module mux_w3_1(S,I0,I1,out);`
	`  input S, I0, I1;`
	`  output [8:0] out;`
	`  assign out[0] = I0;`
	`  assign out[1] = I1;`
	`  assign out[2] = S;`
	`  assign out[3] = 1;`
	`  assign out[4] = S;`
	`  assign out[5] = ~S;`
	`  assign out[6] = I1 & 1;`
	`  assign out[7] = I0 & (~S);`
	`  assign out[8] =`
	`  (I1 & 1) | (I0 & (~S));`
	`endmodule`
		

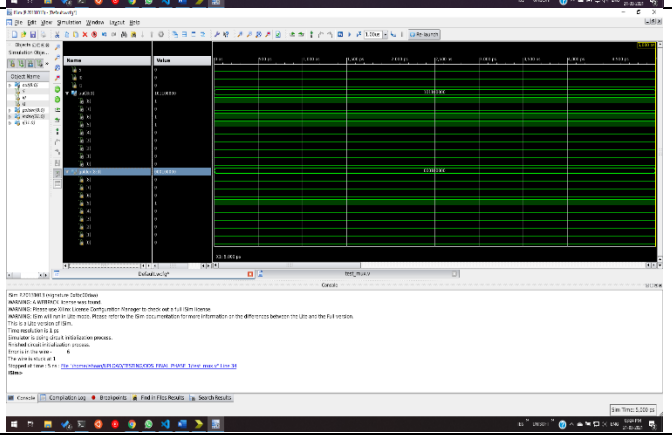
WIRE 4 SSA-0	`module mux_w4_0(S,I0,I1,out);`
	`  input S, I0, I1;`
	`  output [8:0] out;`
	`  assign out[0] = I0;`
	`  assign out[1] = I1;`
	`  assign out[2] = S;`
	`  assign out[3] = S;`
	`  assign out[4] = 0;`
	`  assign out[5] = ~0;`
	`  assign out[6] = I1 & S;`
	`  assign out[7] = I0 & (~0);`
	`  assign out[8] =`
	`  (I1 & S) | (I0 & (~0));`
	`endmodule`
		

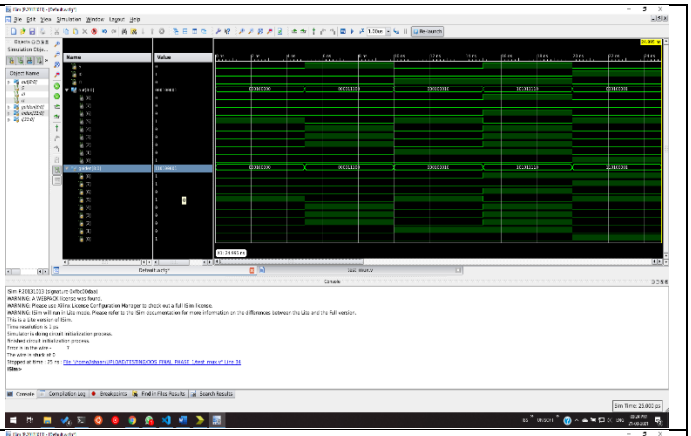
WIRE 4 SSA-1	`module mux_w4_1(S,I0,I1,out);`
	`  input S, I0, I1;`
	`  output [8:0] out;`
	`  assign out[0] = I0;`
	`  assign out[1] = I1;`
	`  assign out[2] = S;`
	`  assign out[3] = S;`
	`  assign out[4] = 1;`
	`  assign out[5] = ~1;`
	`  assign out[6] = I1 & S;`
	`  assign out[7] = I0 & (~1);`
	`  assign out[8] =`
	`  (I1 & S) | (I0 & (~1));`
	`endmodule`
		

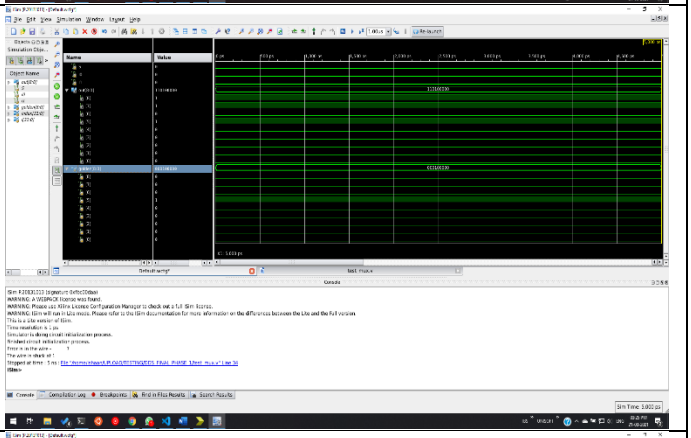
WIRE 5 SSA-0	`module mux_w5_0(S,I0,I1,out);`
	`  input S, I0, I1;`
	`  output [8:0] out;`
	`  assign out[0] = I0;`
	`  assign out[1] = I1;`
	`  assign out[2] = S;`
	`  assign out[3] = S;`
	`  assign out[4] = S;`
	`  assign out[5] = 0;`
	`  assign out[6] = I1 & S;`
	`  assign out[7] = I0 & (0);`
	`  assign out[8] =`
	`  (I1 & S) | (I0 & (0));`
	`endmodule`
		

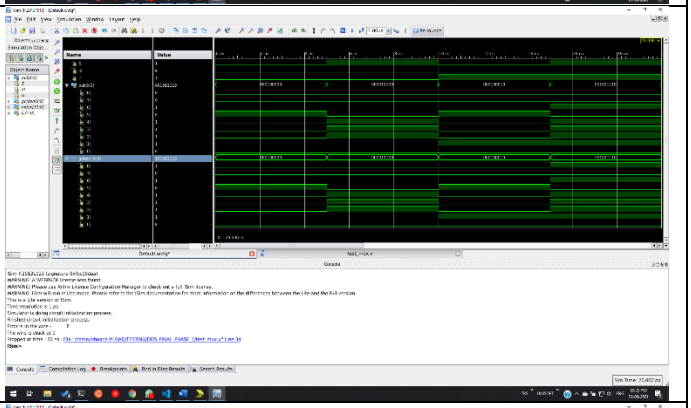
WIRE 5 SSA-1	`module mux_w5_1(S,I0,I1,out);`
	`  input S, I0, I1;`
	`  output [8:0] out;`
	`  assign out[0] = I0;`
	`  assign out[1] = I1;`
	`  assign out[2] = S;`
	`  assign out[3] = S;`
	`  assign out[4] = S;`
	`  assign out[5] = 1;`
	`  assign out[6] = I1 & S;`
	`  assign out[7] = I0 & (1);`
	`  assign out[8] =`
	`  (I1 & S) | (I0 & (1));`
	`endmodule`
		

WIRE 6 SSA-0	`module mux_w6_0(S,I0,I1,out);`
	`  input S, I0, I1;`
	`  output [8:0] out;`
	`  assign out[0] = I0;`
	`  assign out[1] = I1;`
	`  assign out[2] = S;`
	`  assign out[3] = S;`
	`  assign out[4] = S;`
	`  assign out[5] = ~S;`
	`  assign out[6] = 0;`
	`  assign out[7] = I0 & (~S);`
	`  assign out[8] =`
	`  (0) | (I0 & (~S));`
	`endmodule`
		

WIRE 6 SSA-1	`module mux_w6_1(S,I0,I1,out);`
	`  input S, I0, I1;`
	`  output [8:0] out;`
	`  assign out[0] = I0;`
	`  assign out[1] = I1;`
	`  assign out[2] = S;`
	`  assign out[3] = S;`
	`  assign out[4] = S;`
	`  assign out[5] = ~S;`
	`  assign out[6] = 1;`
	`  assign out[7] = I0 & (~S);`
	`  assign out[8] =`
	`  (1) | (I0 & (~S));`
	`endmodule`
		

WIRE 7 SSA-0	`module mux_w7_0(S,I0,I1,out);`
	`input S, I0, I1;`
	`output [8:0] out;`
	`assign out[0] = I0;`
	`assign out[1] = I1;`
	`assign out[2] = S;`
	`assign out[3] = S;`
	`assign out[4] = S;`
	`assign out[5] = ~S;`
	`assign out[6] = I1 & S;`
	`assign out[7] = 0;`
	`assign out[8] =`
	`(I1 & S) | (0);`
	`endmodule`
		

WIRE 7 SSA-1	`module mux_w7_1(S,I0,I1,out);`
	`input S, I0, I1;`
	`output [8:0] out;`
	`assign out[0] = I0;`
	`assign out[1] = I1;`
	`assign out[2] = S;`
	`assign out[3] = S;`
	`assign out[4] = S;`
	`assign out[5] = ~S;`
	`assign out[6] = I1 & S;`
	`assign out[7] = 1;`
	`assign out[8] =`
	`(I1 & S) | (1);`
	`endmodule`
		

WIRE 8 SSA-0	`module mux_w8_0(S,I0,I1,out);`
	`input S, I0, I1;`
	`output [8:0] out;`
	`assign out[0] = I0;`
	`assign out[1] = I1;`
	`assign out[2] = S;`
	`assign out[3] = S;`
	`assign out[4] = S;`
	`assign out[5] = ~S;`
	`assign out[6] = I1 & S;`
	`assign out[7] = I0 & (~S);`
	`assign out[8] = 0;`
	`endmodule`
		

WIRE 8 SSA-1	`module mux_w8_1(S,I0,I1,out);`
	`input S, I0, I1;`
	`output [8:0] out;`
	`assign out[0] = I0;`
	`assign out[1] = I1;`
	`assign out[2] = S;`
	`assign out[3] = S;`
	`assign out[4] = S;`
	`assign out[5] = ~S;`
	`assign out[6] = I1 & S;`
	`assign out[7] = I0 & (~S);`
	`assign out[8] = 1;`
	`endmodule`