

B.M.S. College of Engineering
(Autonomous Institution affiliated to VTU, Belagavi)

Department of Computer Science and Engineering



AAT
Verilog Laboratory
Report

19CS3PCLOD

(Autonomous Scheme 2020)

Submitted By:

Name: Ananya Setty B.A
USN: 1BM21CS

B.M.S. College of Engineering
Department of Computer Science and Engineering



Laboratory Certificate

This is to certify that **ANANYA SETTY B.A** has satisfactorily completed the course of Experiments in Practical Logic Design (Verilog) prescribed by the Department during the odd semester 2021-22.

Name of the Candidate: **ANANYA SETTY B.A**

USN No.: **1BM21CS**

Semester: **III**

Section: **A**

Marks	
Max. Marks	Obtained
10	
Marks in Words	

Signature of the staff in-charge

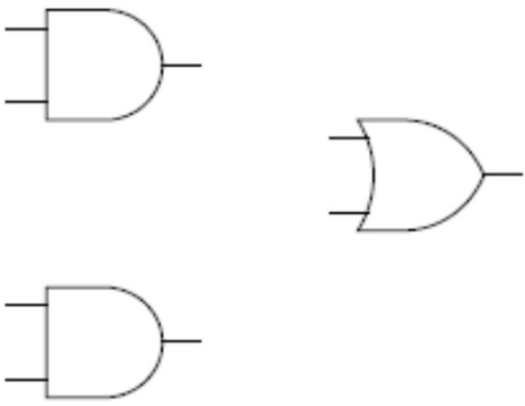
Head of the Department

Date:

Verilog Program List

19CS3PCLOD

Laboratory Experiments

Serial No.	Title
	CYCLE I Structural Modeling
•	<p>Write HDL implementation for the following Logic</p> <ul style="list-style-type: none"> • AND/OR/NOT <p>Simulate the same using structural model and depict the timing diagram for valid inputs.</p>
•	<p>Write HDL implementation for the following Logic</p> <ul style="list-style-type: none"> • NAND/NOR <p>Simulate the same using structural model and depict the timing diagram for valid inputs.</p>
•	<p>Write HDL implementation for the following Logic</p> <div style="text-align: center;">  </div> <p>Simulate the same using structural model and depict the timing diagram for valid inputs.</p>
•	Write HDL implementation for a 4:1 Multiplexer. Simulate the same using structural model and depict the timing diagram for valid inputs.
•	Write HDL implementation for a 2-to-4 decoder. Simulate the same using structural model and depict the timing diagram for valid inputs.
•	Write HDL implementation for a 4-to-2 encoder. Simulate the same using structural model and depict the timing diagram for valid inputs.

	<p style="text-align: center;">CYCLE II Behavior Modeling</p>
•	Write HDL implementation for a RS flip-flop using behavioral model. Simulate the same and depict the timing diagram for valid inputs.
•	Write HDL implementation for a JK flip-flop using behavioral model. Simulate the same and depict the timing diagram for valid inputs.
•	Write HDL implementation for a 4-bit right shift register using behavioral model. Simulate the same and depict the timing diagram for valid inputs.
•	Write HDL implementation for a 3-bit up-counter using behavioral model. Simulate the same and depict the timing diagram for valid inputs.
	<p style="text-align: center;">CYCLE III Dataflow Modeling</p>
•	Write HDL implementation for AND/OR/NOT gates using data flow model. Simulate the same and depict the timing diagram for valid inputs.
•	Write HDL implementation for a 3-bit full adder using data flow model. Simulate the same and depict the timing diagram for valid inputs.

Verilog Program List-19CS3PCLOD
SCHEME OF CONDUCT AND EVALUATION

CLASS: III SEMESTER
YEAR: 21-22

EVALUATION SCHEME Tutorial Test: 1 hour

Expt. No.	TITLE	Max. Marks	Marks Obtained	Signature
•	AND/OR/NOT	5		
•	NAND/NOR			
•	Logic diagram			
•	Multiplexer			
•	Decoder			
•	Encoder			
•	RS			
•	JK			
•	Shift right			
•	Counter			
•	AND/OR/NOT – data flow			
•	3-bit full adder			
	Test: Viva – 2 Marks + Write-up – 1 Mark + Execution – 2 Marks	5		
	TOTAL MARKS	10		

Icarus Verilog Installation (Windows)

- Download and install iverilog from here: <http://bleyer.org/icarus/>
- During installation, check the checkbox so that iverilog is added to the System PATH.
- If this is not done, one will need to manually locate their iverilog installation directory and copy the path to the bin folder situated within it.
- To do this, we will need to navigate to the control panel and access the environment variables section of the computer. Here, we add a new variable and set it to %PATH%;c:\iverilog\bin assuming that is the installation directory for iverilog.
- Open command prompt or any other preferred terminal and type `iverilog` and press enter. Trace back the steps for mistakes if version information is not displayed and re-install if necessary.

Compilation and viewing waveforms

- With iverilog, a third party text editor is necessary. Any editor would work (notepad, gedit, or vim, for example) but a modern text editor such as SublimeText 3 or Visual Studio Code is preferred as plug-ins can be installed to provide syntax highlighting for the Verilog HDL code being written.
- While writing the code, two lines must be added in the test bench module.

```
module testSR;
    reg [1:0]A;
    reg c;
    wire q,qb;
    SR_FF srff(A,c,q,qb);
    initial c = 1'b0;
    always #5 c = ~c;
    initial
    begin
        $dumpfile("test.vcd");
        $dumpvars(0,testSR);
    end
endmodule
```

The `$dumpfile()` and `$dumpvars()` functions should be passed, with the former containing the name of a file ending with `.vcd` and the latter containing the name of the testbench module itself. This will dump a vcd file that will allow us to view the waveform. This should always be right after the initial and begin statements.

- After this file is written and saved under a `.v` extension (example `VHDLcode.v`), the terminal must be opened and the user must navigate to the directory in which the file is saved.
- In this directory, running `iverilog -o outputFile VHDLcode.v` will output a `.vvp` file with the name `outputFile.vvp`.
- After generating the vvp file, run the `vvp outputFile` command to get the waveform dump with name of the given string under `$dumpfile()`.

- The next step is to run `gtkwave` in the command line and open File > Open New Tab > select the generated `.vcd` file. Afterward, click on the added element on the viewer and insert it. This will display the waveform.

References

- Zucker, M. (2019). *E15 - Installing and testing Icarus Verilog*. [online] Swarthmore.edu. Available at: https://www.swarthmore.edu/NatSci/mzucker1/e15_f2014/iverilog.html
- KONSTADELIAS, I. (n.d.). *Icarus Verilog + GTKWave Guide*. [ebook] http://inf-server.inf.uth.gr/~konstadel/resources/Icarus_Verilog_GTKWave_guide.pdf

Cycle 1: Structural Modelling

Experiment 1: Source Code

```
module gates(input a, b, output [2:0]y);
    assign y[2]= a & b; // AND gate
    assign y[1]= a | b; // OR gate
    assign y[0]= ~a; // NOT gate
endmodule

module gates_tb;
    wire [2:0]y;
    reg a, b;
    gates aon(a,b,y);
    initial
    begin
        $dumpfile("dumpg1.vcd");
        $dumpvars(0,gates_tb);
        a = 1'b0;
        b = 1'b0;
        #50;
        a = 1'b0;
        b = 1'b1;
        #50;
        a = 1'b1;
        b = 1'b0;
        #50;
        a = 1'b1;
        b = 1'b1;
        #50;
    end
endmodule
```


Compilation, Execution and Result of Simulation:

The image displays the workflow for Verilog simulation, including code editing, waveform analysis, and command-line execution.

Verilog Code (and_or_not1.v):

```
1 module gates(input a, b, output [2:0]y)
2     assign y[2]= a & b; // AND gate
3     assign y[1]= a | b; // OR gate
4     assign y[0]= ~a; // NOT gate
5 endmodule
6
7 module gates_tb;
8     wire [2:0]y;
9     reg a, b;
10    gates aon(a,b,y);
11    initial
12    begin
13        $dumpfile("dumpg1.vcd");
14        $dumpvars(0,gates_tb);
15        a = 1'b0;
16        b = 1'b0;
17        #50;
18        a = 1'b0;
19        b = 1'b1;
20        #50;
21        a = 1'b1;
22        b = 1'b0;
23        #50;
24        a = 1'b1;
25        b = 1'b1;
26        #50;
27    end
28 endmodule
```

GTKWave Simulation (dumpg1.vcd):

The waveform shows the signals `a`, `b`, and the 3-bit output `y[2:0]` over time. The output values are 001, 011, 010, and 110, corresponding to the input combinations (0,0), (0,1), (1,0), and (1,1) respectively.

Terminal Output:

```
Select C:\Windows\System32\cmd.exe - gtkwave dumpg1.vcd
C:\iverilog\bin\myverilog>iverilog -o pg1 and_or_not1.v
C:\iverilog\bin\myverilog>vvp pg1
VCD info: dumpfile dumpg1.vcd opened for output.
C:\iverilog\bin\myverilog>gtkwave dumpg1.vcd
GTKWave Analyzer v3.3.108 (w)1999-2020 BSI
[0] start time.
[200] end time.
```

Verilog file length: 451 lines: 29 Ln: 20 Col: 15 Pos: 361 Windows (CR LF) UTF-8 INVS 31°C Partly sunny 1852 26-03-2022

Experiment 2 :Compilation, Execution and Result of Simulation

The image shows a Verilog code editor (Notepad++) and a waveform viewer (GTKWave) side-by-side. The Verilog code defines an OR gate and a testbench. The testbench initializes inputs A and B, and uses delays to change their values. The simulation results in GTKWave show the waveforms for A, B, and the output x.

```
1 module and_gate(A,B,x); // OR gate used as a gate
2   input A,B;           // defines two input ports
3   output x;            // defines one output port
4   and g1(x,A,B);       //Gate declaration
5 endmodule
6
7 module andgate_tb;     //Simulation module for gate
8   reg A,B;             //Storage of data for passing
9   wire x;
10  and g1(A,B,x);        //circuit is instantiated
11  initial               //starts simulation
12  begin                 //Input is generated to test
13
14    $dumpfile("dump2.vcd");
15    $dumpvars(0,andgate_tb);
16    A=1'b0;B=1'b0; // 1'b0 signifies binary 0
17    #20             // Delay of 20 ns
18    A=1'b0;B=1'b1; // After 20ns AB=01
19    #20             // Another Delay of 20 ns
20    A=1'b1;B=1'b0; // After 40ns from start point AB=10
21    #20
22    A=1'b1;B=1'b1; // After 60ns from start point AB=11
23    #20 $finish;    // the simulation terminates after 80 ns
24  end
25 endmodule
```

The GTKWave window shows the simulation results for the testbench. The signals A, B, and x are displayed as waveforms over time. The time scale is from 0 to 80 ns. The signals A and B are shown as digital signals, and x is shown as a digital signal that changes state based on the inputs A and B.

GTKWave - dump2.vcd
Marker: 4 sec | Cursor: 3 sec
From: 0 sec To: 80 sec

Signals: A, B, x
Time: 0, 20 sec, 40 sec, 60 sec, 80 sec

Verilog file: length: 1,175 lines: 32 Ln: 22 Col: 70 Pos: 1,077 Windows (CR LF) UTF-8 BNS

29°C 19:37 26-03-2022

Experiment 3 :Compilation, Execution and Result of Simulation

The image displays the Verilog source code for a 2-to-1 multiplexer and its simulation results. The code is written in a text editor and includes two modules: `addor` and `test_andor`. The `addor` module implements a 2-to-1 multiplexer with inputs `A, B, C, D` and output `Y`. It uses two 2-input AND gates (`g1` and `g2`) and a 3-input OR gate (`g3`) to produce the output `Y`. The `test_andor` module is a testbench that initializes the inputs `a, b, c, d` and calls the `addor` module. It also sets up a Verilog dump file named `dumpg2.vcd` to capture the simulation results.

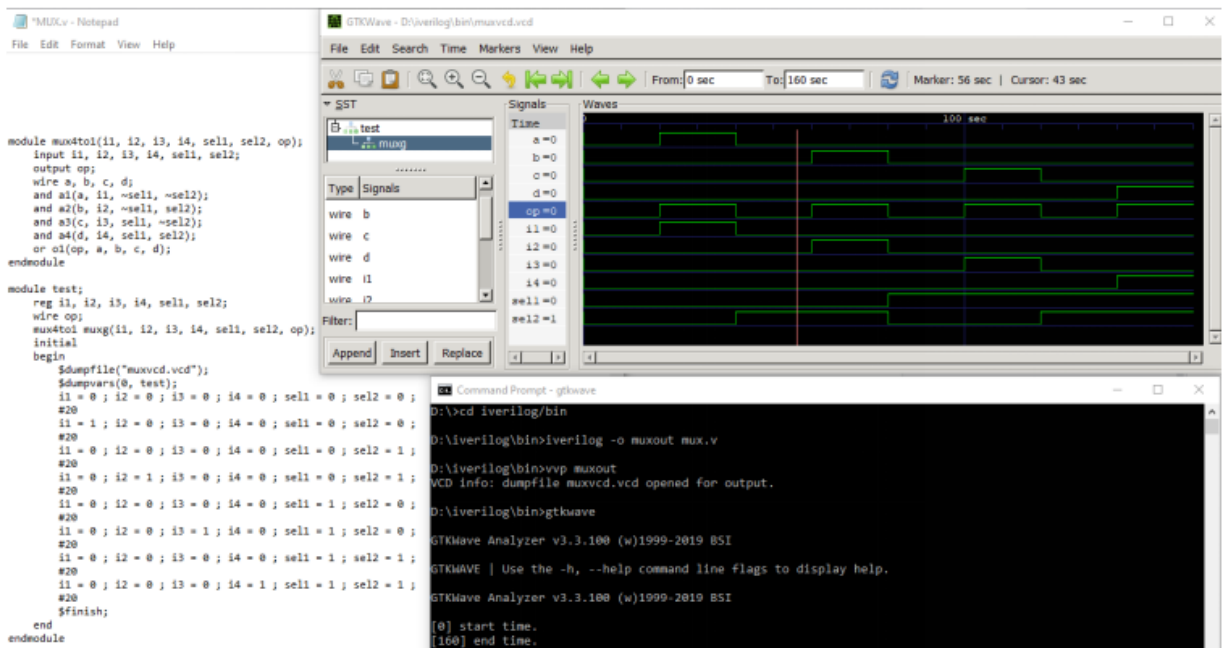
```
1 module addor(A,B,C,D,Y);
2   input A,B,C,D;
3   output Y;
4   wire and_op1, and_op2;
5   and g1(and_op1,A,B);
6   and g2(and_op2,C,D);
7   or g3(Y,and_op1,and_op2); // g3 represents 3 input OR gate
8 endmodule
9
10 module test_andor;
11   reg a,b,c,d;
12   wire y;
13   addor ao(a,b,c,d,y);
14   initial
15   begin
16     $dumpfile("dumpg2.vcd");
17     $dumpvars(0,test_andor);
18     a=0; b=1; c=1; d=1;
19     #10
20     a=0; b=0; c=1; d=0;
21     #10
22     $finish;
23   end
24 endmodule
25
```

The simulation results are shown in the GTKWave window, which displays a waveform for the signals `a, b, c, d, y`. The signals are represented as digital waveforms over time. The `y` signal is the output of the multiplexer, which is the logical OR of the two 2-input AND gates. The simulation results show that the output `y` is 1 for the first 10 ns and 0 for the next 10 ns, which corresponds to the testbench inputs.

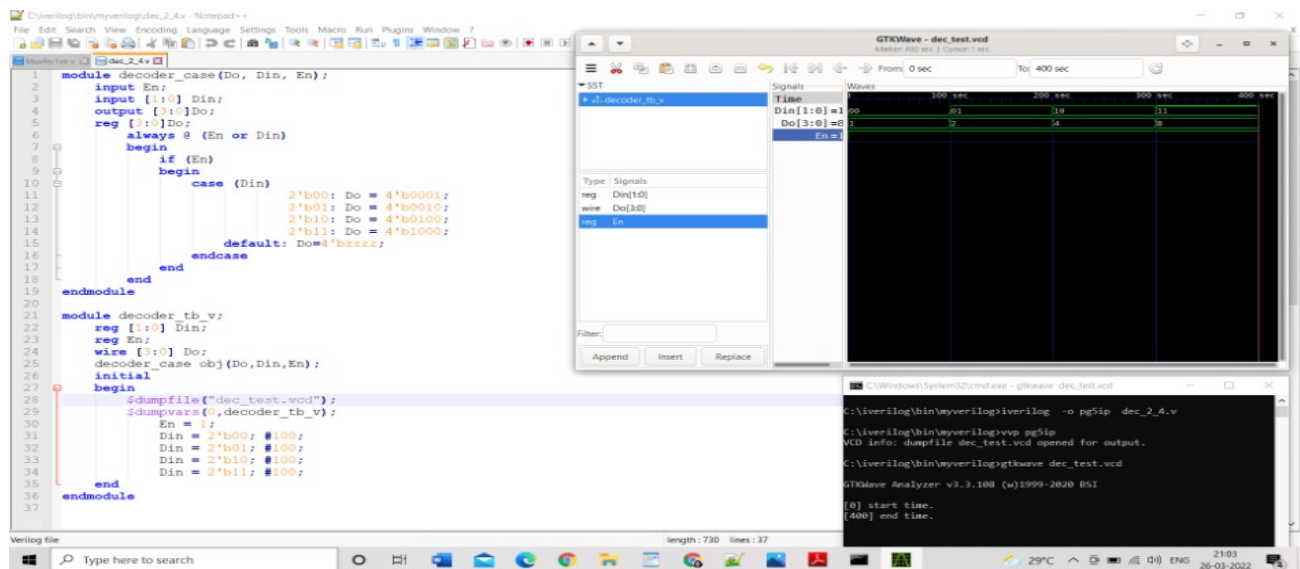
The bottom window shows the command prompt output, which includes the compilation and simulation commands:

```
C:\iverilog\bin\myverilog>iverilog -o nandnor nand_nor2.v
C:\iverilog\bin\myverilog>vvp nandnor
VCD info: dumpfile dumpg2.vcd opened for output.
nand_nor2.v:22: $finish called at 20 (1s)
C:\iverilog\bin\myverilog>gtkwave dumpg2.vcd
GTKWave Analyzer v1.3.108 (w)1999-2020 BSI
[0] start time.
[20] end time.
```

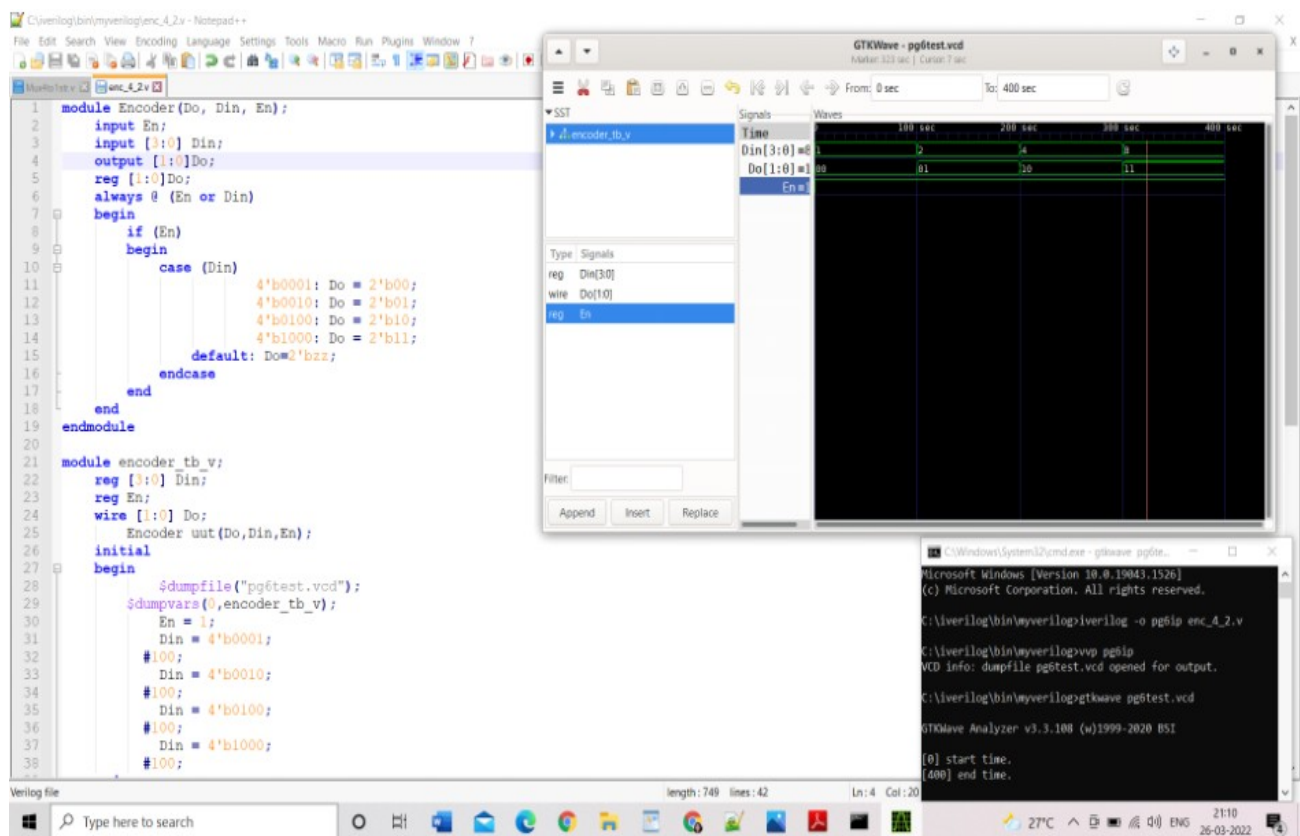
Experiment 4 :Compilation, Execution and Result of Simulation



Experiment 5 :Compilation, Execution and Result of Simulation

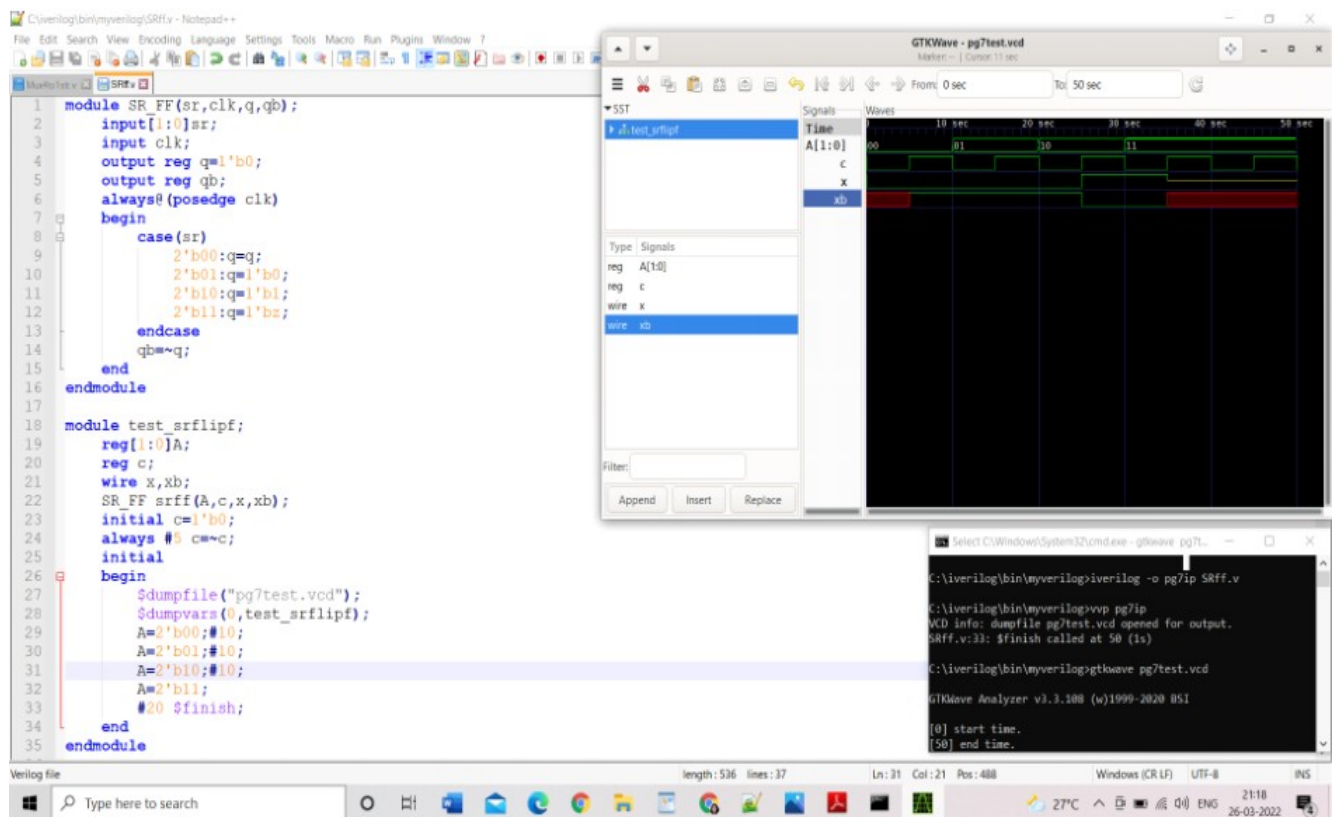


Experiment 6 :Compilation, Execution and Result of Simulation

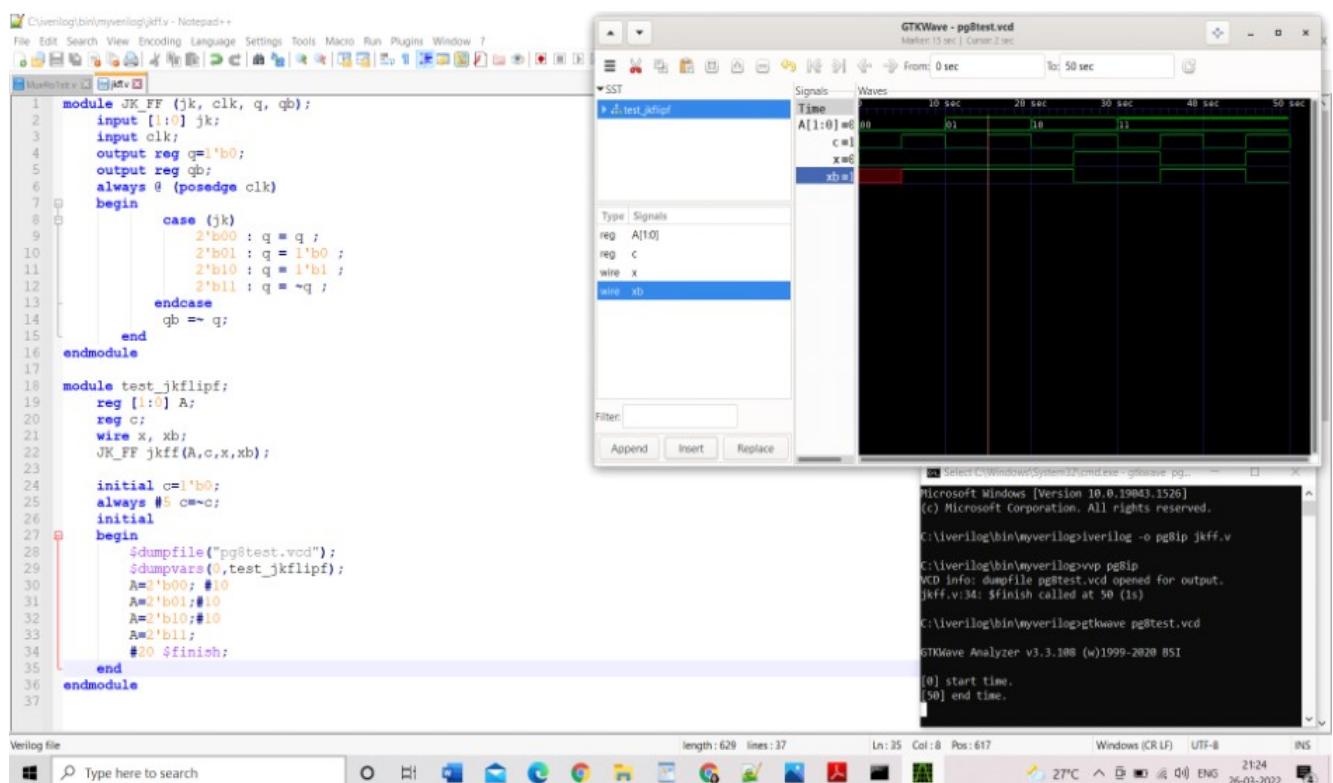


Cycle 2: Behavior Modelling

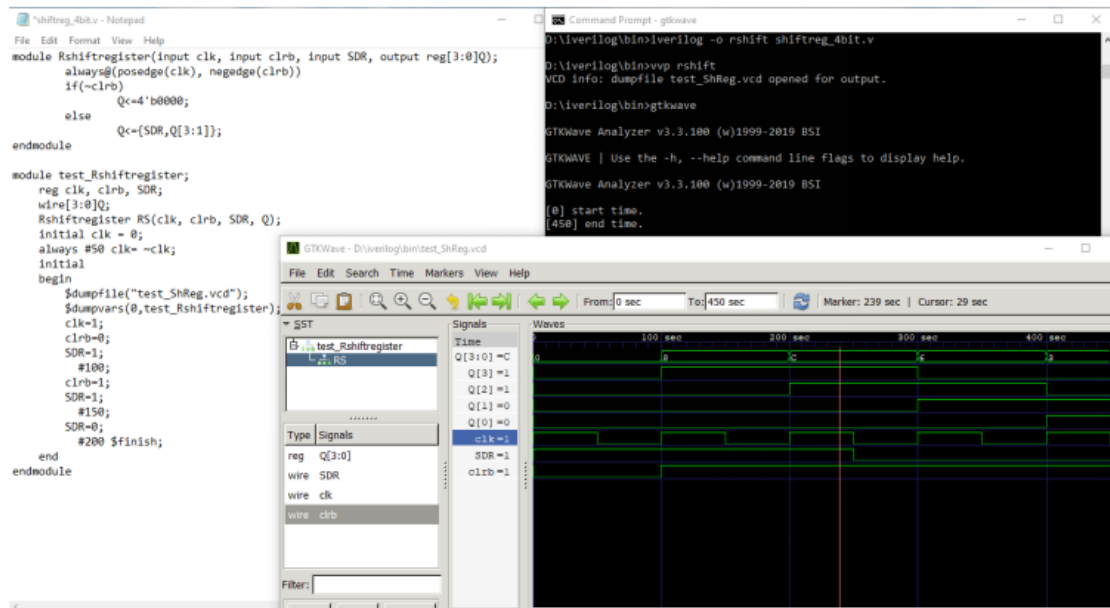
Experiment 7 :Compilation, Execution and Result of Simulation



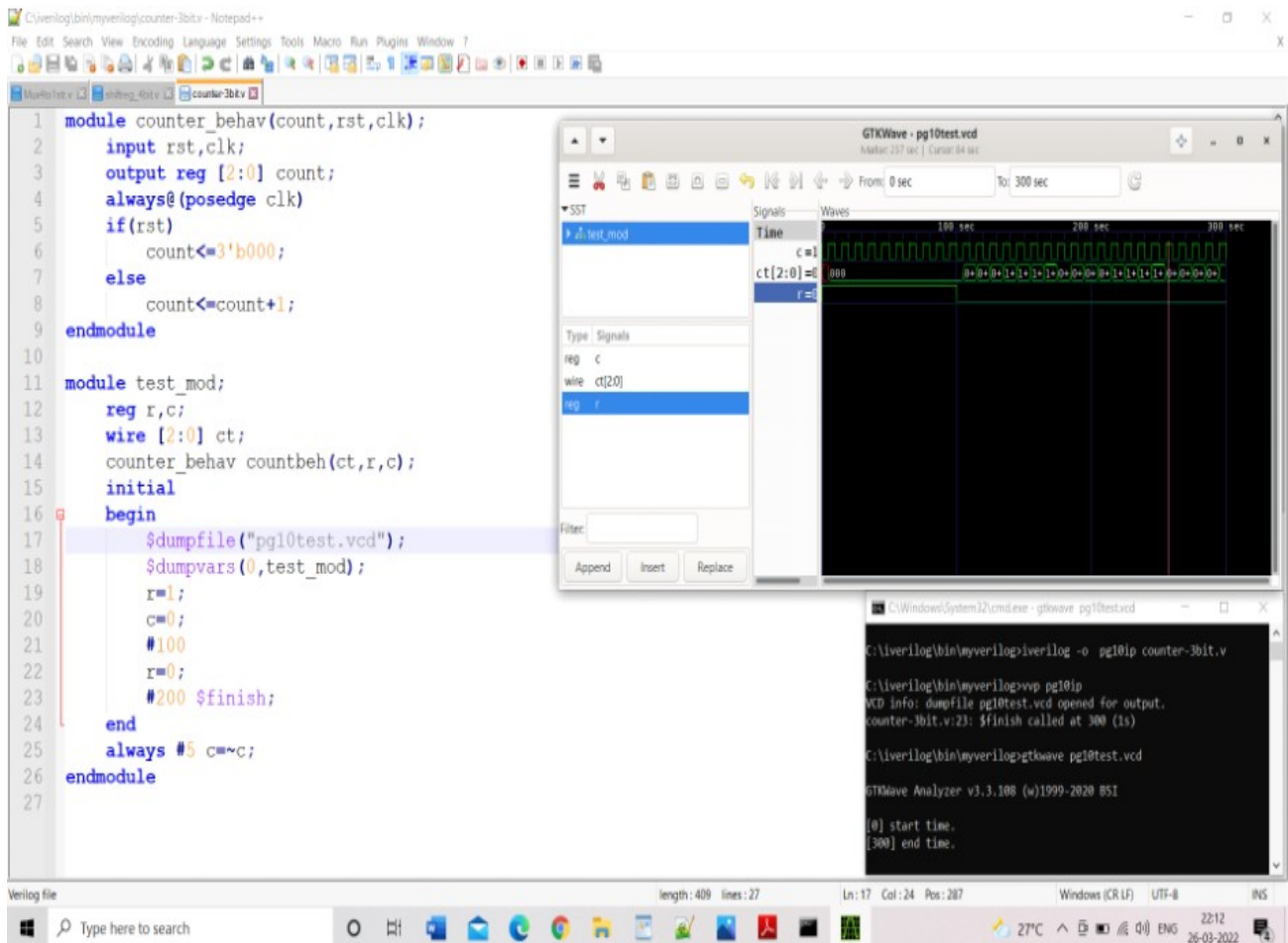
Experiment 8:Compilation, Execution and Result of Simulation



Experiment 9: Compilation, Execution and Result of Simulation

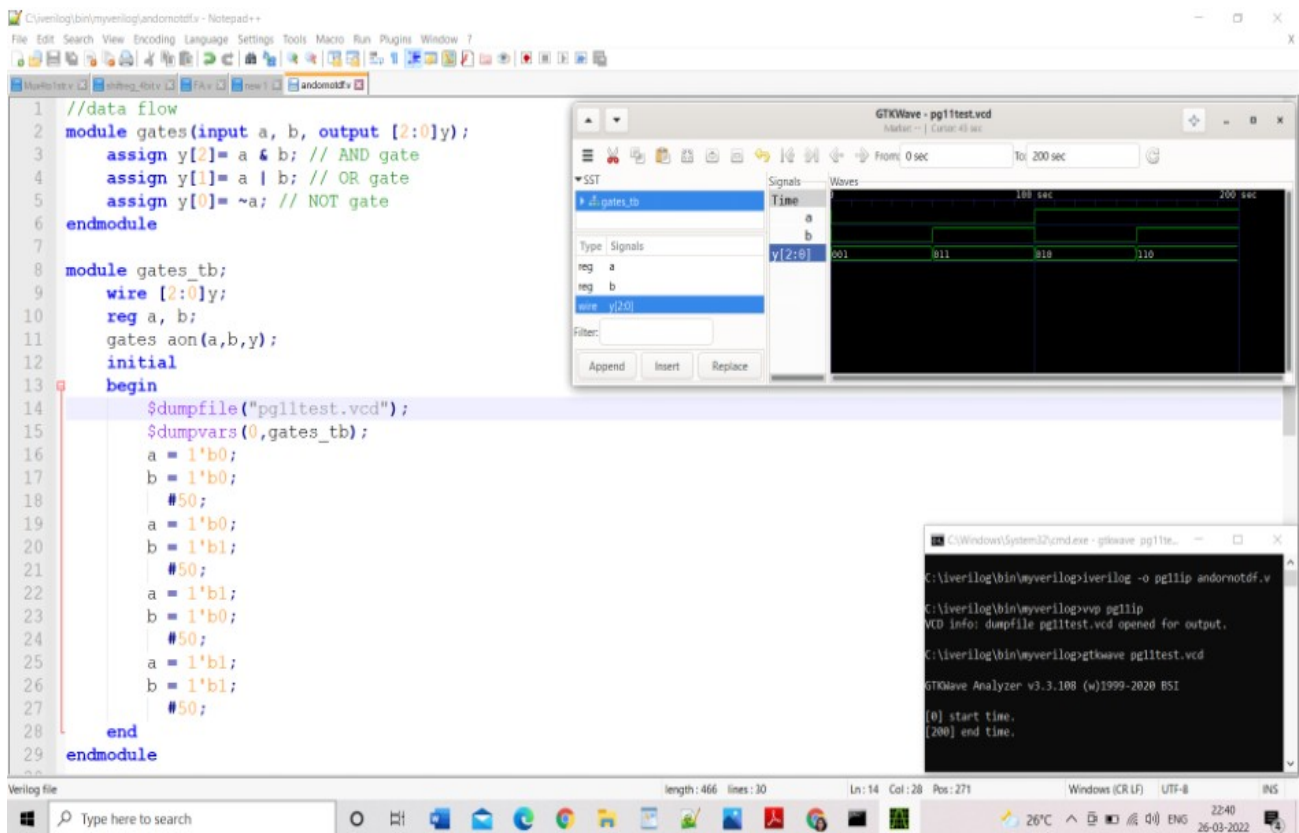


Experiment 10: Compilation, Execution and Result of Simulation

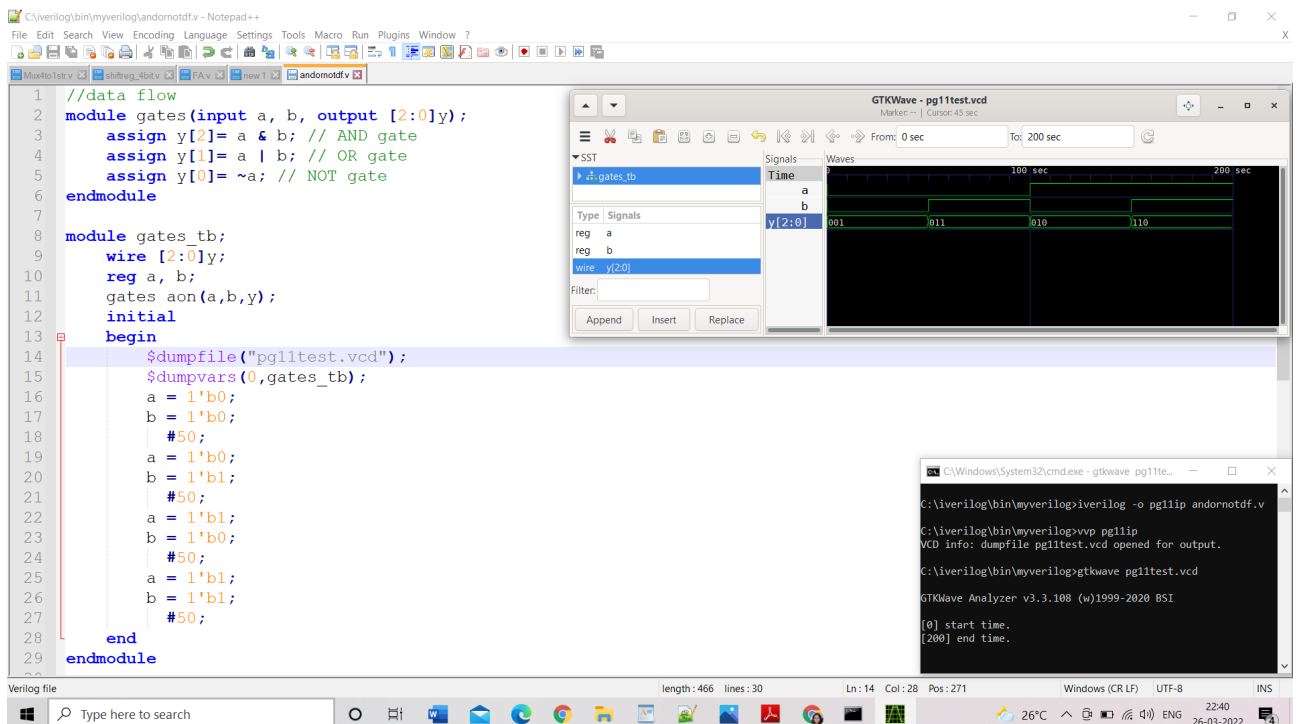


Cycle 3: Dataflow Modelling

Experiment 11: Compilation, Execution and Result of Simulation



Experiment 12: Compilation, Execution and Result of Simulation



Experiment 12:Compilation, Execution and Result of Simulation