

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

Department of Computer Science and Engineering



AAT

**Verilog Laboratory
Report**

23CS3PCLOD

(December 2023-March 2024)

Submitted by:

Yashraj Sinha

1BM22CS335

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



Laboratory Certificate

This is to certify that Yashraj Sinha has satisfactorily completed the course of Experiments in Practical Logic Design (Verilog) prescribed by the Department during the odd semester 2023-24.

Name of the Candidate: Yashraj Sinha

USN No.: 1BM22CS335 Semester: III Section: F

Marks	
Max. Marks	Obtained
10	
Marks in Words	

Signature of the staff in-charge

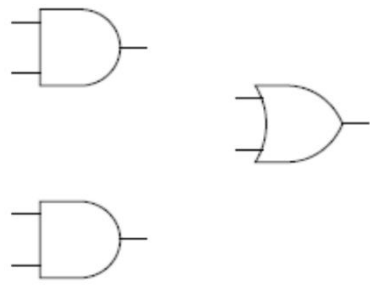
Head of the Department

Date:

Verilog Program List

23CS3PCLOD

Laboratory Experiments

Serial No.	Title
	CYCLE I Structural Modelling
1.	<p>Write HDL implementation for the following Logic</p> <p>a. AND/OR/NOT</p> <p>Simulate the same using a structural model and depict the timing diagram for valid inputs.</p>
2.	<p>Write HDL implementation for the following Logic</p> <p>a. NAND/NOR</p> <p>Simulate the same using a structural model and depict the timing diagram for valid inputs.</p>
3.	<p>Write HDL implementation for the following AND-OR Combinational Logic.</p> <div style="text-align: center;">  </div> <p>Simulate the same using a structural model and depict the timing diagram for valid inputs.</p>
4.	Write HDL implementation for a 4:1 Multiplexer. Simulate the same using a structural model and depict the timing diagram for valid inputs.
5.	Write HDL implementation for a 2-to-4 decoder. Simulate the same using a structural model and depict the timing diagram for valid inputs.
6.	Write HDL implementation for a 4-to-2 encoder. Simulate the same using a structural model and depict the timing diagram for valid inputs.

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

Verilog Program List-23CS3PCLOD

SCHEME OF CONDUCT AND EVALUATION

CLASS: III SEMESTER

YEAR: 23-24

EVALUATION SCHEME Tutorial Test: 1 hour

Expt. No.	TITLE	Max. Marks	Marks Obtained	Signature
1.	AND/OR/NOT	5		
2.	NAND/NOR			
3.	Logic diagram			
4.	Multiplexer			
5.	Decoder			
6.	Encoder			
7.	RS			
8.	JK			
9.	Shift Right			
10.	Counter			
11.	AND/OR/NOT – data flow			
12.	3-bit Full Adder			
	Test: Viva – 2 Marks + Writeup – 1 Mark + Execution – 2 Marks	5		
	TOTAL MARKS	10		

Icarus Verilog Installation (Windows)

1. Download and install iverilog from here: <http://bleyer.org/icarus/>
2. During installation, check the checkbox so that iverilog is added to the System PATH.
3. 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.
4. 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.
5. 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.

Please turn it over.

Compilation and viewing waveforms

1. 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.
2. 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);
            A = 2'b00; #10
        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.

3. 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.

4. In this directory, running `iverilog -o outputFile VHDLcode.v` will output a .vvp file with the name `outputFile.vvp`.
5. After generating the vvp file, run the `vvp outputFile` command to get the waveform dump with name of the given string under `$dumpfile()`.
6. 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

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

CYCLE 1: STRUCTURAL MODELLING

Experiment 1

```
module gates(a,b,c,t0,t1,t2);    // start of module gates with following variables
    input a,b,c;                // the inputs to the given circuit are a,b,c
    output t0,t1,t2;            //the outputs from the gates are t0,t1,t2
    and ag(t0,a,b);             // and gate ag takes input a,b and gives output t0
    or og(t1,a,b);              // or gate og takes input a,b and gives output t1
    not notg(t2,c);             // not gate notg takes c as input and gives output t2
endmodule                       //end of the module

module testbench;              //start of testbench
    reg a,b,c;                 //a,b,c are data storage elements , synthesized to combinational circuit
    wire t0,t1,t2;             //t0,t1,t2 are assigned to be connected
    gates g(a,b,c,t0,t1,t2);
    initial                    //initial processes execute only once
    begin
        $dumpfile("gates.vcd"vvp ); //Level set to 0 implies that all variables of module
        $dumpvars(0,testbench); into the gates.vcd file(to show output in gtkwave)
        a = 1'b0 ; b= 1'b0 ; c= 1'b0;    //for the first 20ns inputs a,b,c are given values
        #20                                0,0,1 represented by a single bit
        a = 1'b0 ; b= 1'b1 ; c= 1'b0;    //for the next 20ns inputs a,b,c are given values
        #20                                0,1,0(single bit representation)
        a = 1'b1 ; b= 1'b0 ; c= 1'b0;    //for next 20ns a,b,c have the values 1,0,0
        #20
        a = 1'b1 ; b= 1'b1 ; c= 1'b0;    //for next 20ns a,b,c have values 1,1,0
        #20
        a = 1'b0 ; b= 1'b0 ; c= 1'b1;    //for next 20ns a,b,c have values 0,0,1
        #20
        a = 1'b0 ; b= 1'b0 ; c= 1'b0;    //for next 20ns a,b,c have values 0,0,0
        #20
        $finish;                      // $finish denotes the end of time duration for
    end                                which input values are given to a,b,c
endmodule
```


Compilation, Execution and Result of Simulation

The image shows a screenshot of a Windows environment with three windows open: Notepad, Command Prompt, and GTKWave.

Notepad Window (p1 - Notepad): Contains the Verilog code for a module named `gates` and a testbench module named `testbench`.

```
module gates(a,b,c,t0,t1,t2);
    input a,b,c;
    output t0,t1,t2;
    and ag(t0,a,b);
    or og(t1,a,b);
    not ntg(t2,c);
endmodule

module testbench;
    reg a,b,c;
    wire t0,t1,t2;
    gates g(a,b,c,t0,t1,t2);
    initial
    begin
        $dumpfile("p1.vcd");
        $dumpvars(0,testbench);
        a = 1'b0 ; b = 1'b0 ; c = 1'b0;
        #20
        a = 1'b0 ; b = 1'b1 ; c = 1'b0;
        #20
        a = 1'b1 ; b = 1'b0 ; c = 1'b0;
        #20
        a = 1'b1 ; b = 1'b1 ; c = 1'b0;
        #20
        a = 1'b0 ; b = 1'b0 ; c = 1'b1;
        #20
        a = 1'b0 ; b = 1'b0 ; c = 1'b1;
        #20
        $finish;
    end
endmodule
```

Command Prompt - gtkwave: Shows the execution of the Verilog code using the `iverilog` and `gtkwave` commands.

```
Microsoft Windows [Version 10.0.19043.1526]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Arnav>cd desktop
C:\Users\Arnav\Desktop>cd logic design verilog
C:\Users\Arnav\Desktop\Logic Design Verilog>iverilog -o andornotout p1.v
C:\Users\Arnav\Desktop\Logic Design Verilog>vvp andornotout
VCD info: dumpfile p1.vcd opened for output.
p1.v:29: $finish called at 120 (1s)
C:\Users\Arnav\Desktop\Logic Design Verilog>gtkwave
GTKWave Analyzer v3.3.108 (w)1999-2020 BSI
GTKWAVE | Use the -h, --help command line flags to display help.
GTKWave Analyzer v3.3.108 (w)1999-2020 BSI

[0] start time.
[120] end time.
```

GTKWave - C:\Users\Arnav\Desktop\Logic Design Verilog\p1.vcd: Shows the simulation results in a waveform viewer. The signals `a`, `b`, `c`, `t0`, `t1`, and `t2` are displayed. The time scale is 100 ns. The signals `a`, `b`, and `c` are inputs, and `t0`, `t1`, and `t2` are outputs. The waveform shows the logic levels of these signals over time.

Experiment 2

```
module no(t0,a,b,t1);           // start of module no with following variables
    input a,b;                  //a and b are inputs for the circuit
    output t0,t1;               //t0 and t1 are outputs for the circuit
    and a1(x1,a,b);             //a1 represents and gate with a,b inputs and x1 output
    not n1(t0,x1);              //n1 represents not gate with x1 input and t0 output
    or o1(x2,a,b);              //o1 represents or gate with a,b inputs and x2 output
    not n2(t1,x2);              //n2 represents not gate with x2 input and t1 output
endmodule                       // x1,x2 are intermediate inputs/output for respective gates

module testbench;
    wire t0,t1;
    reg a,b;
    no g(t0,a,b,t1);
    initial                      //initial processes execute only once
    begin
        $dumpfile("gates12.vcd");//All variables of module dumped to gates12.vcd
        $dumpvars(0,testbench);
        a = 1'b0 ; b=1'b0;      //for the first 20ns inputs a,b have values 0,0
        #20
        a = 1'b0 ; b=1'b1;      //for the next 20ns inputs a,b have values 0,1
        #20
        a = 1'b1 ; b=1'b0;      //for the next 20ns inputs a,b have values 1,0
        #20
        a = 1'b1 ; b=1'b1;      //for the next 20ns inputs a,b have values 1,1
        #20
        $finish;                 //$finish denotes the end of time duration for
                                //which input values are given to a and b
    End
endmodule
```

Compilation, Execution and Result of Simulation

The image shows a Notepad window with a Verilog testbench named `p2`. The testbench defines a module `no` with inputs `a, b` and output `t1`. It includes logic for `and`, `not`, and `or` operations. The testbench module `testbench` sets initial values for `a` and `b` and uses `$dumpfile` and `$dumpvars` for simulation output.

```
module no(t0, a, b, t1);
    input a,b;
    output t0,t1;
    and a1(x1, a, b);
    not n1(t0,x1);
    or o1(x2,a,b);
    not n2(t1,x2);
endmodule

module testbench;
    wire t0,t1;
    reg a,b;
    no g(t0,a,b,t1);
    initial
    begin
        $dumpfile("p2.vcd");
        $dumpvars(0,testbench);
        a = 1'b0 ; b = 1'b0 ;
        #20
        a = 1'b0 ; b = 1'b1 ;
        #20
        a = 1'b1 ; b = 1'b0 ;
        #20
        a = 1'b1 ; b = 1'b1 ;
        #20
        $finish;
    end
endmodule
```

A Command Prompt window shows the execution of the Verilog code using `iverilog` and `gtkwave`. The output indicates that the VCD file `p2.vcd` was opened for output and the simulation finished at 80 seconds.

```
C:\Users\Arnav\Desktop\Logic Design Verilog>iverilog -o nandnorout p2.v
C:\Users\Arnav\Desktop\Logic Design Verilog>vvp nandnorout
VCD info: dumpfile p2.vcd opened for output.
p2.v:26: $finish called at 80 (1s)
C:\Users\Arnav\Desktop\Logic Design Verilog>gtkwave
GTKWave Analyzer v3.3.108 (w)1999-2020 BSI
GTKWAVE | Use the -h, --help command line flags to display help.
GTKWAVE | Use the -h, --help command line flags to display help.
[0] start time.
[120] end time.
WM Destroy
```

The GTKWave window displays the simulation results in a waveform viewer. The signals `a`, `b`, `t0`, `t1`, `x1`, and `x2` are shown. The waveform shows the values of these signals over time, with a marker at 35 seconds and a cursor at 80 seconds.

GTKWave - C:\Users\Arnav\Desktop\Logic Design Verilog\p2.vcd
Marker: 35 sec | Cursor: 80 sec

Signals: `a=0`, `b=1`, `t0=1`, `t1=0`, `x1=0`, `x2=1`

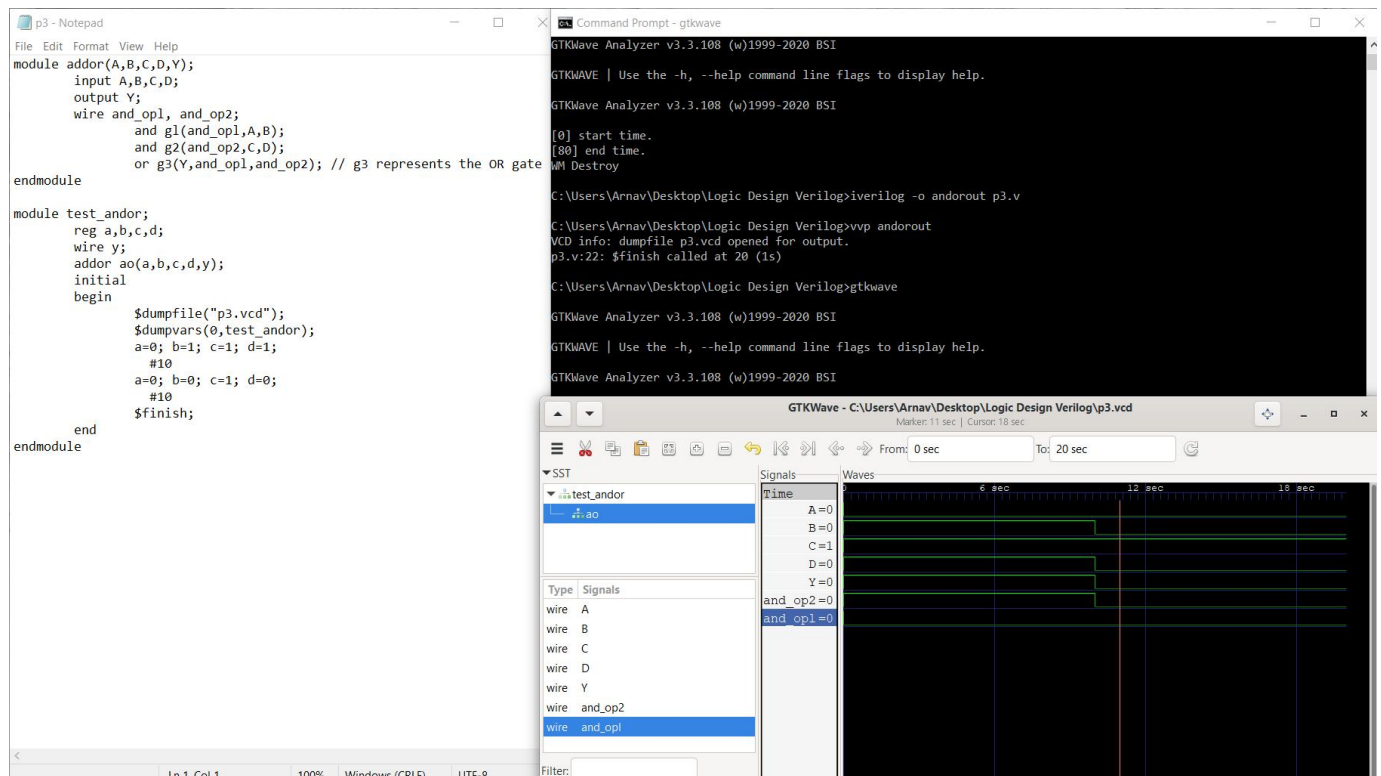
Waves: 10 sec, 20 sec, 30 sec, 40 sec, 50 sec, 60 sec, 70 sec, 80 sec

Experiment 3

```
module andor(A,B,C,D,Y);
    input A,B,C,D;           //inputs for the circuit are a,b,c,d
    output Y;                //y is the output of the circuit
    wire and_op1,and_op2;
        and g1(and_op1,A,B);    //g1 represents an and gate
        and g2(and_op2,C,D);    //g2 represents an and gate
        or g3(Y,and_op1,and_op2); //g3 represents the or gate
endmodule

module test_andor;
    reg a,b,c,d; //a,b,c,d treated as data storage elements
    wire y;
    andor ao(a,b,c,d,y);
    initial      //initial processes execute only once
    begin
        $dumpfile("diagram_test.vcd");
        $dumpvars(0,test_andor);
        a=0;b=1;c=1;d=1;    //for the first 10ns a,b,c,d have the values 0,1,1,1
        #10
        a=0;b=0;c=1;d=0;    //for the next 10ns a,b,c,d have the values 0,0,1,0
        #10
        $finish;            //$finish denotes the end of time duration for
                             //which input values are given to a,b,c
    end
endmodule
```

Compilation, Execution and Result of Simulation



Experiment 4

```
module Mux4to1(t1,t2,t3,t4, sel1, sel2,op); //Mux4to1 module declaration with all
    input t1,t2,t3,t4,sel1,sel2;           //involved variables
    output op;
    wire a,b,c,d;
    and a1(a,t1,-sel1,-sel2); //a1 is an and gate with a as output, t1,-sel1,-sel2 inputs
    and a2(b,t2,-sel1,sel2); //Similarly a2,a3,a4 are AND gates with respective
    and a3(c,t3,sel1,-sel2);           inputs and outputs
    and a4(d,t4,sel1,sel2);
    or o1(op,a,b,c,d); //o1 is an or gate with op as output,
endmodule
```

```
module test;
    reg t1,t2,t3,t4,sel1,sel2;
    wire op;
    Mux4to1 muxg(t1,t2,t3,t4,sel1,sel2,op);
    initial
    begin
        $dumpfile("muxout.vcd");
        $dumpvars(0,test);
        t1=0;t2=0;t3=0;t4=0;sel1=0;sel2=0;
        //inputs t1,t2,t3,t4 are given values 0,0,0,0 for the first 20ns and sel1=0,sel2=0
        #20
        t1=1;t2=0;t3=0;t4=0;sel1=0;sel2=0;
        //inputs t1,t2,t3,t4 are given values 1,0,0,0 for the next 20ns and sel1=0,sel2=0
        #20
        t1=0;t2=0;t3=0;t4=0;sel1=0;sel2=1;
        //inputs t1,t2,t3,t4 are given values 0,0,0,0 for the next 20ns and sel1=0,sel2=1
        #20
        t1=1;t2=1;t3=0;t4=0;sel1=0;sel2=1;
        //inputs t1,t2,t3,t4 are given values 0,0,0,0 for the next 20ns and sel1=1,sel2=1
        #20
        t1=0;t2=0;t3=1;t4=0;sel1=1;sel2=0;
        //inputs t1,t2,t3,t4 are given values 0,0,0,0 for the next 20ns and sel1=1,sel2=0
        #20
        t1=1;t2=1;t3=0;t4=1;sel1=1;sel2=0;
        //inputs t1,t2,t3,t4 are given values 0,0,0,0 for the next 20ns and sel1=1,sel2=0
        #20
        t1=1;t2=0;t3=0;t4=1;sel1=1;sel2=1;
        //inputs t1,t2,t3,t4 are given values 0,0,0,0 for the next 20ns and sel1=1,sel2=1
        #20
    end
endmodule
```

```

t1=0;t2=1;t3=1;t4=0;sel1=1;sel2=1;
//inputs t1,t2,t3,t4 are given values 0,0,0,0 for the next 20ns and sel1=1,sel2=1
#20
$finish;                                // $finish denotes the time duration for which
end                                     //inputs t1,t2,t3,t4 and select lines sel1,sel2
endmodule

```

Compilation, Execution and Result of Simulation

The image shows the Verilog code in Notepad and its simulation in GTKWave. The code defines a 4-to-1 multiplexer and a testbench. The testbench sets inputs t1, t2, t3, t4 to 0 for 20ns, then sets sel1 and sel2 to 1 for another 20ns, and finally calls \$finish.

The simulation in GTKWave shows the waveforms for the signals. The signals are: a=0, b=0, c=0, d=0, i1=0, i2=0, i3=0, i4=0, op=0, sel1=1, and sel2=0. The time scale is 100 ns. The simulation shows that the output op is 0 for the first 20ns and then becomes 1 for the next 20ns, which corresponds to the testbench's configuration.

```

p4 - Notepad
File Edit Format View Help
module mux4to1(i1,i2,i3,i4,sel1,sel2,op);
  input i1,i2,i3,i4,sel1,sel2;
  output op;
  wire a,b,c,d;
  and a1(a,i1,~sel1,~sel2);
  and a2(b,i2,~sel1,sel2);
  and a3(c,i3,sel1,~sel2);
  and a4(d,i4,sel1,sel2);
  or o1(op,a,b,c,d);
endmodule

module test;
  reg i1,i2,i3,i4,sel1,sel2;
  wire op;
  mux4to1 muxg(i1,i2,i3,i4,sel1,sel2,op);
  initial
  begin
    $dumpfile("p4.vcd");
    $dumpvars(0,test);
    i1 = 0; i2 = 0; i3 = 0; i4 = 0; sel1 = 0; sel2 = 0;
    #20
    i1 = 1; i2 = 0; i3 = 0; i4 = 0; sel1 = 0; sel2 = 0;
    #20
    i1 = 0; i2 = 0; i3 = 0; i4 = 0; sel1 = 0; sel2 = 1;
    #20
    i1 = 0; i2 = 1; i3 = 0; i4 = 0; sel1 = 0; sel2 = 1;
    #20
    i1 = 0; i2 = 0; i3 = 0; i4 = 0; sel1 = 1; sel2 = 0;
    #20
    i1 = 0; i2 = 0; i3 = 1; i4 = 0; sel1 = 1; sel2 = 0;
    #20
    i1 = 0; i2 = 0; i3 = 0; i4 = 0; sel1 = 1; sel2 = 1;
    #20
    i1 = 0; i2 = 0; i3 = 0; i4 = 1; sel1 = 1; sel2 = 1;
    #20
    $finish;
  end
endmodule

```

GTKWave Analyzer v3.3.108 (w)1999-2020 BSI
 GTKWAVE | Use the -h, --help command line flags to display help.
 GTKWave Analyzer v3.3.108 (w)1999-2020 BSI
 [0] start time.
 [20] end time.
 MM Destroy
 C:\Users\Arnav\Desktop\Logic Design Verilog>iverilog -o muxout p4.v
 C:\Users\Arnav\Desktop\Logic Design Verilog>vvp muxout
 VCD info: dumpfile p4.vcd opened for output.
 p4.v:36: \$finish called at 160 (1s)
 C:\Users\Arnav\Desktop\Logic Design Verilog>gtkwave
 GTKWave Analyzer v3.3.108 (w)1999-2020 BSI
 GTKWAVE | Use the -h, --help command line flags to display help.

GTKWave - C:\Users\Arnav\Desktop\Logic Design Verilog\p4.vcd
 Marker: 88 sec | Cursor: 139 sec
 From: 0 sec To: 160 sec
 Signals: a=0, b=0, c=0, d=0, i1=0, i2=0, i3=0, i4=0, op=0, sel1=1, sel2=0
 Waves: a=0, b=0, c=0, d=0, i1=0, i2=0, i3=0, i4=0, op=0, sel1=1, sel2=0

Experiment 5

```
module bcddecoder2to4(b0,b1,d0,d1,d2,d3);
    input b0,b1;          //inputs b0,b1 and outputs d0,d1,d2 and d3
    output d0,d1,d2,d3;
    wire t0,t1;
    not n1(t0,b0);        //n1,n2 represent the NOT gates
    not n2(t1,b1);
    and a0(d0,t0,t1);     //a0 represents an and gate , d0 output and t0,t1,inputs
    and a1(d1,t0,b1);     //Similarly a1,a2,a3 represent the other AND gates
    and a2(d2,b0,t1);
    and a3(d3,b0,t1);
endmodule

module test;
    reg b0,b1;
    wire d0,d1,d2,d3;
    bcddecoder2to4 bcdg(b0,b1,d0,d1,d2,d3);
    initial
    begin
        $dumpfile("bcd.vcd");
        $dumpvars(0,test);
        b0 = 0; b1 = 0;          //for the first 40ns b0,b1 have values 0,0
        #40
        b0 = 0; b1 = 1;          //for the next 40ns b0,b1 have values 0,1
        #40
        b0 = 1; b1 = 0;          //for the next 40ns b0,b1 have values 1,0
        #40
        b0 = 1; b1 = 1;          //for the next 40ns b0,b1 have values 1,1
        #40
        $finish;
    end
endmodule
```


Compilation, Execution and Result of Simulation

The image shows a workflow for Verilog simulation. On the left, a Notepad window displays the Verilog code for a 2-to-4 decoder. The code defines a module `bcddecoder2to4` with inputs `b0, b1` and outputs `d0, d1, d2, d3`. It uses logic gates (NOT, AND) to implement the decoder. A test module is also shown, which instantiates the decoder and sets initial values for `b0` and `b1` to 0 and 1 respectively, then calls `$finish`.

In the center, a Command Prompt window shows the execution of the simulation. The commands used are `gtkwave`, `iverilog -o decodout p5.v`, `vvp decodout`, and `gtkwave`. The output shows that the VCD file `p5.vcd` was opened for output and the simulation finished at 160 ns.

On the right, the GTKWave window displays the simulation results. The 'SST' (Signal Structure Tree) shows the test module and the `bcdg` instance. The 'Signals' list includes `b0`, `b1`, `d0`, `d1`, `d2`, `d3`, `t0`, and `t1`. The 'Waves' window shows the timing diagram for these signals. The signals `b0` and `b1` are shown as step functions, and the outputs `d0`, `d1`, `d2`, and `d3` are shown as step functions that change when the inputs change. The time scale is 100 ns.

```
module bcddecoder2to4(b0,b1,d0,d1,d2,d3);
    input b0,b1;
    output d0,d1,d2,d3;
    wire t0,t1;
    not n1(t0,b0);
    not n2(t1,b1);
    and a0(d0,t0,t1);
    and a1(d1,t0,b1);
    and a2(d2,b0,t1);
    and a3(d3,b0,b1);
endmodule

module test;
    reg b0,b1;
    wire d0,d1,d2,d3;
    bcddecoder2to4 bcdg(b0,b1,d0,d1,d2,d3);
    initial
    begin
        $dumpfile("p5.vcd");
        $dumpvars(0,test);
        b0 = 0; b1 = 0;
        #40
        b0 = 0; b1 = 1;
        #40
        b0 = 1; b1 = 0;
        #40
        b0 = 1; b1 = 1;
        #40
        $finish;
    end
endmodule
```

```
C:\Users\Arnav\Desktop\Logic Design Verilog>gtkwave
GTKWave Analyzer v3.3.108 (w)1999-2020 BSI
GTKWAVE | Use the -h, --help command line flags to display help.
GTKWave Analyzer v3.3.108 (w)1999-2020 BSI
[0] start time.
[160] end time.
MM Destroy

C:\Users\Arnav\Desktop\Logic Design Verilog>iverilog -o decodout p5.v
C:\Users\Arnav\Desktop\Logic Design Verilog>vvp decodout
VCD info: dumpfile p5.vcd opened for output.
p5.v:29: $finish called at 160 (1s)

C:\Users\Arnav\Desktop\Logic Design Verilog>gtkwave
GTKWave Analyzer v3.3.108 (w)1999-2020 BSI
GTKWAVE | Use the -h, --help command line flags to display help.
```

GTKWave - C:\Users\Arnav\Desktop\Logic Design Verilog\p5.vcd
Marker: 92 sec | Cursor: 159 sec
From: 0 sec To: 160 sec
SST: test, bcdg
Signals: b0=1, b1=0, d0=0, d1=0, d2=1, d3=0, t0=0, t1=1
Waves: 100 sec

Experiment 6

```
module encoder4to2( b0,b1,d0,d1,d2,d3 );
    output b0,b1;
    input d0,d1,d2,d3;
    not n0(t0,d0);      //n0,n1,n2,n3 represent the NOT gates
    not n1(t1,d1);
    not n2(t2,d2);
    not n3(t3,d3);
    and a0(t4,t0,t1,d2,t3); //a0,a1,a2,a3 represent the AND gates
    and a1(t5,t0,t1,t2,d3);
    and a2(t6,t0,d1,t2,t3);
    and a3(t7,t0,t1,t2,t3);
    or o0(b0,t4,t3);      //o0,o1 represent the OR gates
    or o1(b1,t6,d3);
endmodule

module test;
    wire b0,b1;
    reg d0,d1,d2,d3;
    encoder4to2 encg(b0,b1,d0,d1,d2,d3);
    initial
    begin
        $dumpfile("enc4to2.vcd");
        $dumpvars(0,test);
        d0 = 1;d1 = 0;d2 = 0;d3 = 0; //for the first 40ns inputs d0,d1,d2,d3 have
        #40;                          the values 0,0,0,0
        d0 = 0;d1 = 1;d2 = 0;d3 = 0; //for the next 40ns inputs d0,d1,d2,d3 have
        #40;                          the values 0,1,0,0
        d0 = 0;d1 = 0;d2 = 1;d3 = 0; //for the next 40ns inputs d0,d1,d2,d3 have
        #40;                          the values 0,0,1,0
        d0 = 0;d1 = 0;d2 = 0;d3 = 1; //for the next 40ns inputs d0,d1,d2,d3 have
        #40;                          the values 0,0,0,1
    end
endmodule
```

Compilation, Execution and Result of Simulation

The image shows a Verilog project setup and simulation. On the left, a Notepad window displays the Verilog code for a 4-to-2 encoder. The code defines a module `encoder4to2` with inputs `d0, d1, d2, d3` and outputs `b0, b1`. It uses intermediate wires `t0` through `t7` and logic gates (NOT, AND, OR) to implement the encoding logic. A test module is also provided to initialize the inputs and dump the waveforms to a file named `p6.vcd`.

In the center, a Command Prompt window shows the execution of the Verilog code using the `iverilog` and `vvp` tools. The commands executed are:

```
C:\Users\Arnav\Desktop\Logic Design Verilog>gtkwave
GTKWave Analyzer v3.3.108 (w)1999-2020 BSI
GTKWAVE | Use the -h, --help command line flags to display help.
GTKWave Analyzer v3.3.108 (w)1999-2020 BSI
[0] start time.
[160] end time.
WM Destroy

C:\Users\Arnav\Desktop\Logic Design Verilog>iverilog -o encodout p6.v
C:\Users\Arnav\Desktop\Logic Design Verilog>vvp encodout
VCD info: dumpfile p6.vcd opened for output.
p6.v:33: $finish called at 160 (1s)

C:\Users\Arnav\Desktop\Logic Design Verilog>gtkwave
GTKWave Analyzer v3.3.108 (w)1999-2020 BSI
```

On the right, the GTKWave window displays the simulation results. The 'SST' (Signal Selection Table) lists the signals: `b0`, `b1`, `d0`, `d1`, `d2`, `d3`, `t0`, and `t1`. The 'Waves' window shows the timing diagram for these signals over a 100-second period. The signals are represented as digital waveforms, with `b0` and `b1` showing the output of the encoder for the given input sequence.

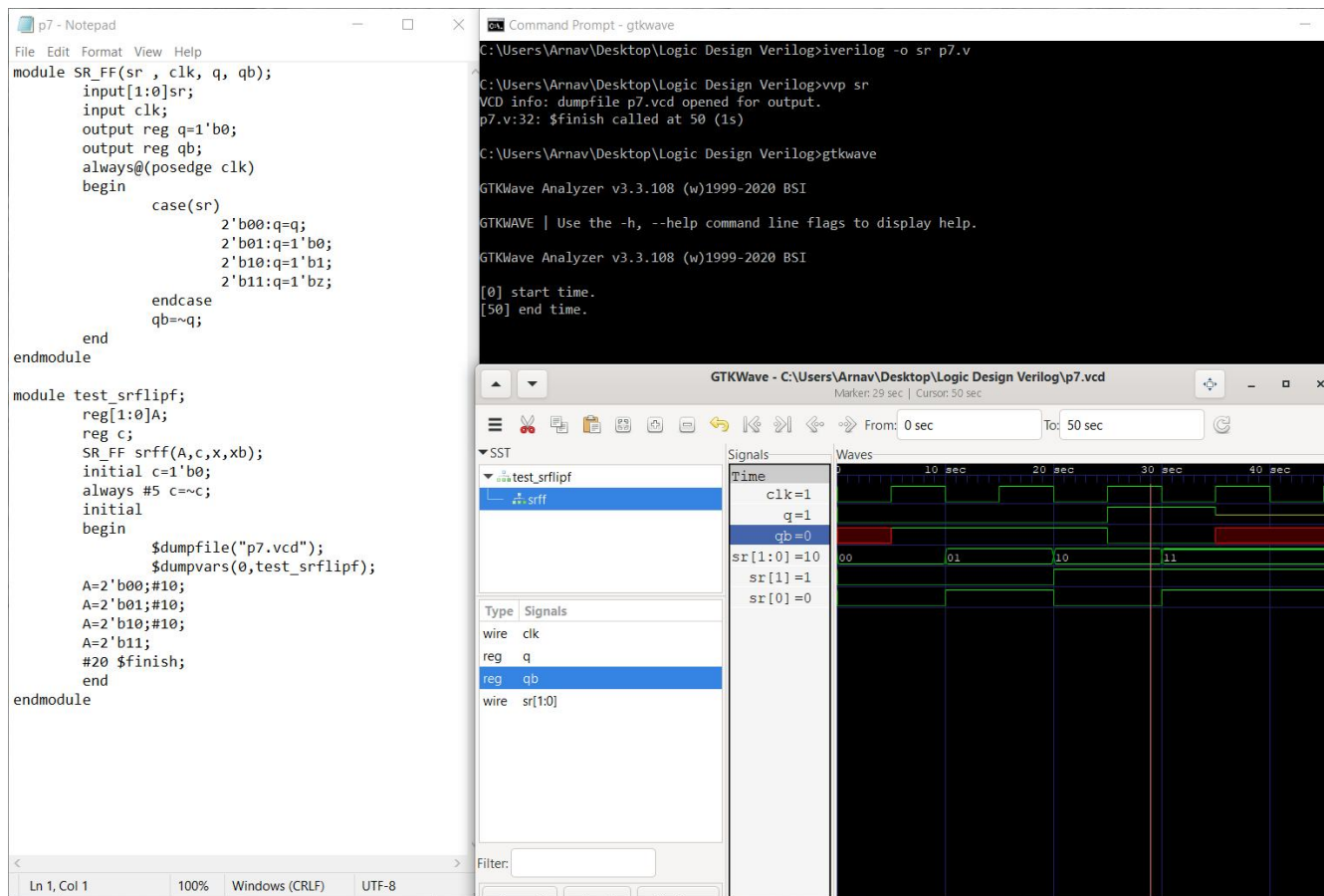
CYCLE 2: BEHAVIOR MODELLING

Experiment 7

```
module SR_FF(sr , clk, q, qb);
    input[1:0]sr;          //[1:0]sr represents the RS Flip-flop input
    input clk;             //clk represents the clock of the circuit
    output reg q=1'b0;      //q and qb represent data storage elements synthesized
    output reg qb;          //for sequential logic circuit
    always@(posedge clk)    //code inside block executed at every positive edge
    begin
        case(sr)
            2'b00:q=q;      //statements which demonstrate the flip-flops behavior
            2'b01:q=1'b0;    //based on the inputs
            2'b10:q=1'b1;
            2'b11:q=1'bz;    //forbidden state
        endcase
        qb=~q;
    end
endmodule

module test_srflipf;
    reg[1:0]A;
    reg c;
    SR_FF srff(A,c,x,xb);
    initial c=1'b0;
    always #5 c=~c;
    initial
    begin
        $dumpfile("sr_test1.vcd");
        $dumpvars(0,test_srflipf);
        A=2'b00;#10;        //For first 10ns, value of A is 00, in the next 10ns the value is 01
        A=2'b01;#10;        //Then 10 and later 11.(This stage is allowed for 20ns)
        A=2'b10;#10;
        A=2'b11;
        #20 $finish; // $finish denotes end of the time duration for which values are given
    end
endmodule
```

Compilation, Execution and Result of Simulation

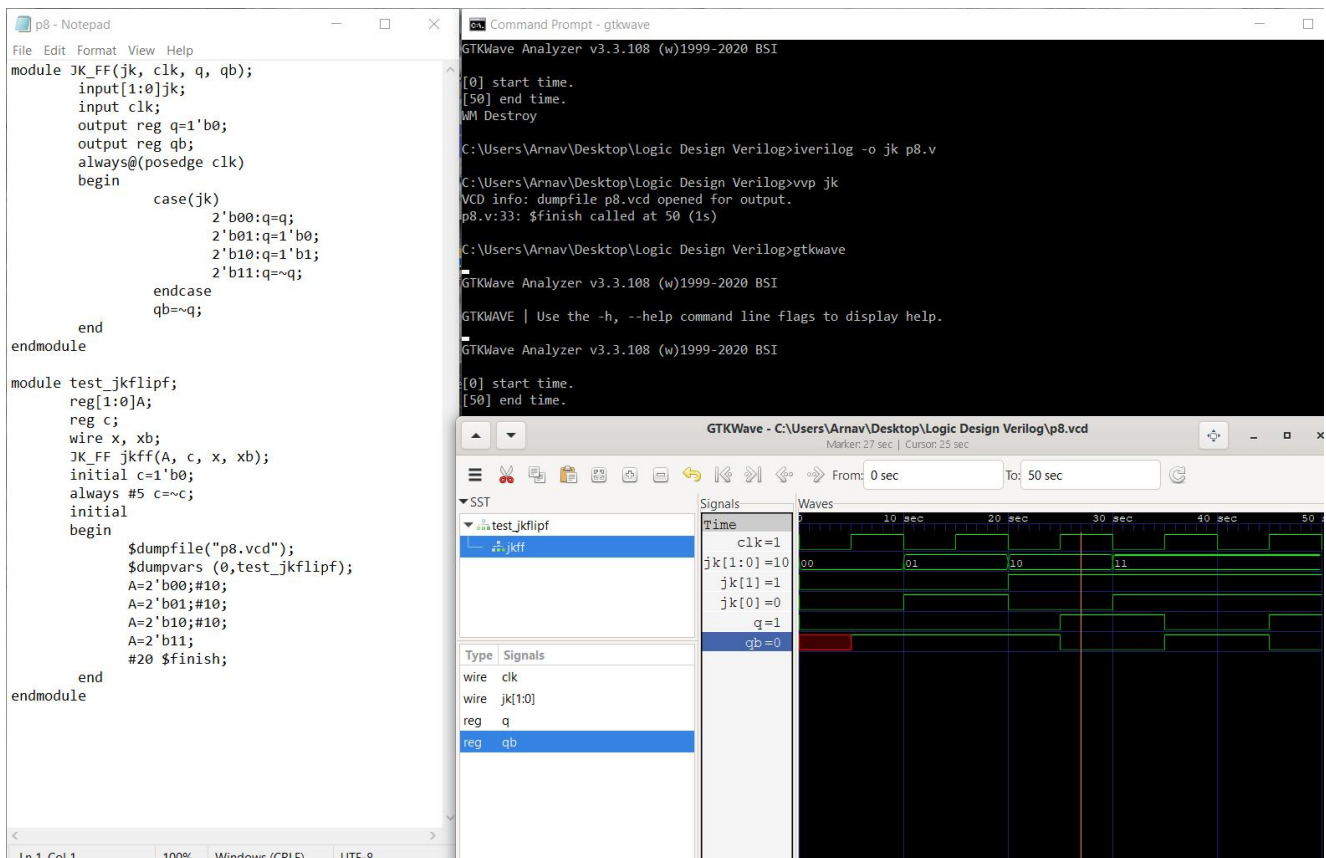


Experiment 8

```
module JK_FF(jk , clk, q, qb);
    input[1:0]jk;
    input clk;
    output reg q=1'b0; //storage elements synthesized for sequential circuit
    output reg qb;
    always@(posedge clk)
    begin
        case(jk)
            2'b00:q=q;           //Q(n+1) = Qn , 0 , 1, (Qn)'
            2'b01:q=1'b0;
            2'b10:q=1'b1;
            2'b11:q=~q;
        endcase
        qb=~q;
    end
endmodule

module test_jkflipf;
    reg[1:0]A;
    reg c;
    wire x,xb;
    JK_FF srff(A,c,x,xb);
    initial c=1'b0;
    always #5 c=~c;    //for every 5ns c is complemented forever
    initial
    begin
        $dumpfile("jk_test1.vcd"); //all variables of module dumped to jk_test1.vcd
        $dumpvars(0,test_jkflipf);
        A=2'b00;#10;
        A=2'b01;#10;
        A=2'b10;#10;
        A=2'b11;
        #20 $finish;
    end
endmodule
```

Compilation, Execution and Result of Simulation

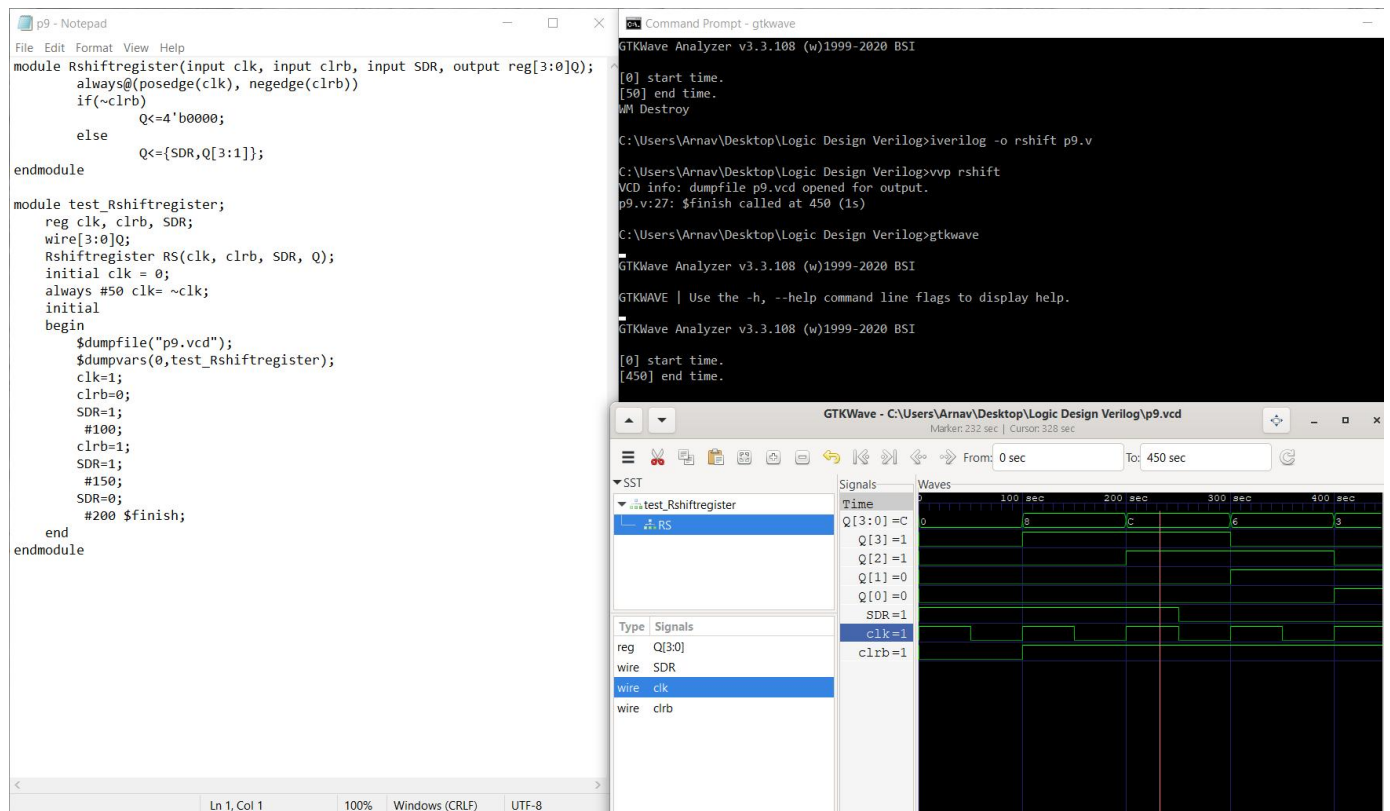


Experiment 9

```
module Rshiftregister(input clk, input clrb, input SDR, output reg[3:0]Q);
    always@(posedge (clk),negedge(clrb))
        if(~clrb)
            Q<=4'b0000; //4-bit shift register
        else
            Q<={SDR,Q[3:1]};
endmodule
```

```
module test_Rshiftregister;
    reg clk,clrb,SDR;
    wire[3:0]Q;
    Rshiftregister RS(clk,clrb,SDR,Q);
    initial clk = 0;
    always #50 clk=~clk; //for every 50ns clk is complemented
    initial
        begin
            $dumpfile("test_Shreg1.vcd");
            $dumpvars(0,test_Rshiftregister);
            clk=1;
            clrb=0;
            SDR = 1;
            #100;
            clrb=1;
            SDR = 1;
            #150;
            SDR=0;
            #200 $finish;
        end
endmodule
```


Compilation, Execution and Result of Simulation

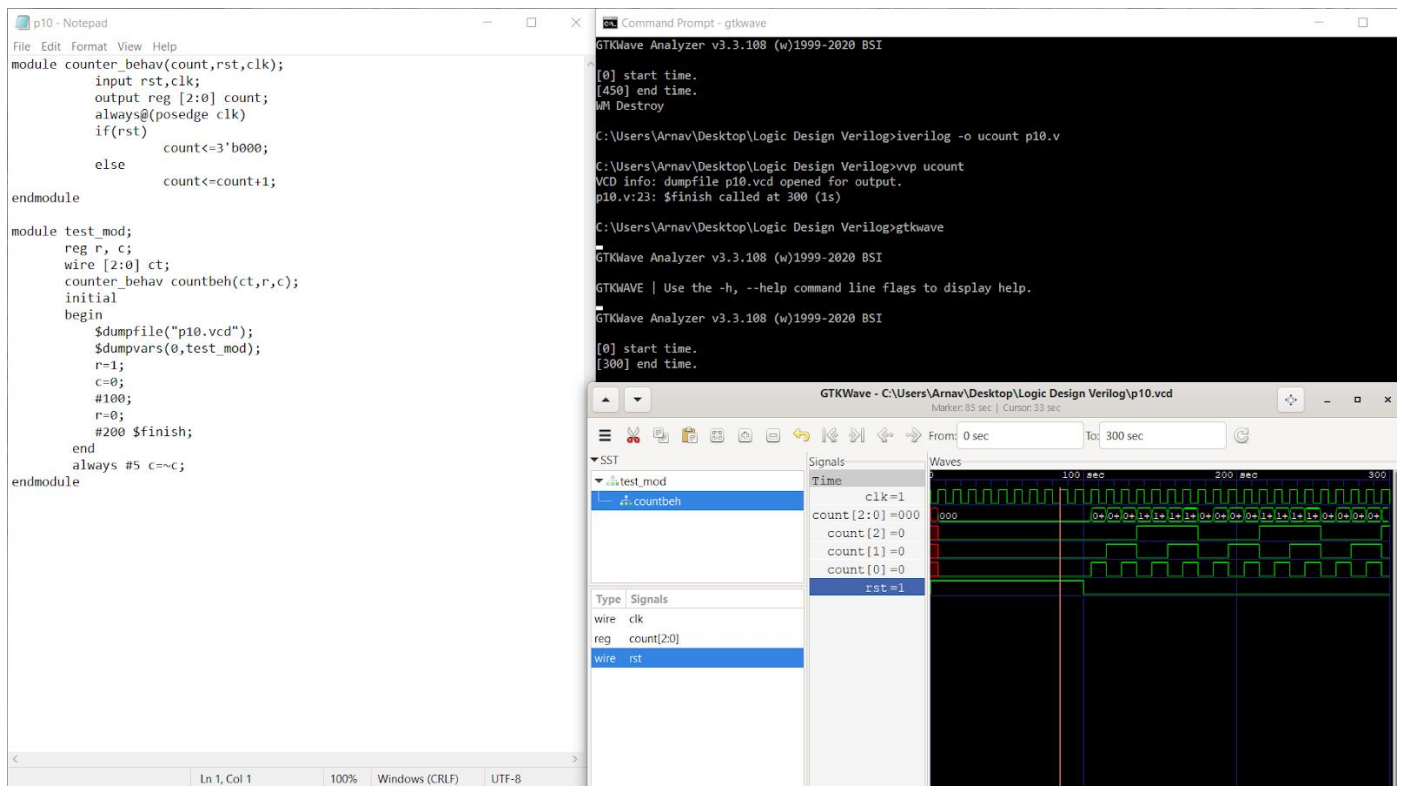


Experiment 10

```
module counter_behav(count,rst,clk);
    input rst,clk;
    output reg [2:0] count;
    always@(posedge clk) //code executed for every positive edge
    if(rst)
        count<=3'b000;
    else
        count<=count+1; //clock pulse count incremented
endmodule

module test_mod;
    reg r,c;
    wire [2:0] ct;
    counter_behav countbeh(ct,r,c);
    initial
    begin
        $dumpfile("cnt_test1.vcd");
        $dumpvars(0,test_mod);
        r=1; //for first 100 ns reset r has value 1, clock c has value 0
        c=0;
        #100; //for next 200ns reset r has value 0, clock c
        r=0;
        #200 $finish;
    end
    always #5 c=~c;
endmodule
```

Compilation, Execution and Result of Simulation



CYCLE 3: DATAFLOW MODELLING

Experiment 11

```
module gates(input a,b,output[2:0]y); // Circuit is described by means of their function.
    assign y[2]= a & b;           //The first output y[0] is defined by 'AND' of a,b
    assign y[1]= a | b;           //The output y[1] is defined by 'OR' off a,b
    assign y[0]= ~a;              //The output y[2] is defined by and NOT of a
endmodule
```

```
module gates_tb;
    wire [2:0]y;
    reg a, b;
    gates aon(a,b,y);
    initial
    begin
        $dumpfile("gates_test.vcd");
        $dumpvars(0,gates_tb);
        a = 1'b0;                //for first 50ns inputs a,b have values 0,0(represented by
        b = 1'b0;                single bit)
        #50;
        a = 1'b0;                //for next 50ns inputs a,b have values 0,1(represented by
        b = 1'b1;                single bit)
        #50;
        a = 1'b1;                //for next 50ns inputs a,b have values 1,0(represented by
        b = 1'b0;                single bit)
        #50;
        a = 1'b1;                //for next 50ns inputs a,b have values 1,1(represented by
        b = 1'b1;                single bit)
        #50;
    end
endmodule
```

Compilation, Execution and Result of Simulation

The image displays the process of compiling, executing, and simulating a Verilog code. It consists of three main windows:

- Notepad (p11 - Notepad):** Contains the Verilog code for a data flow module named `gates`. The code defines a module `gates(input a, b, output [2:0]y);` with three assignments: `y[2] = a & b;` (AND gate), `y[1] = a | b;` (OR gate), and `y[0] = ~a;` (NOT gate). A testbench module `gates_tb` is also defined, which initializes `a` and `b` to various values and uses `$dumpfile` and `$dumpvars` for simulation output.
- Command Prompt - gtkwave:** Shows the commands used to compile and simulate the code:

```
C:\Users\Arnav\Desktop\Logic Design Verilog>iverilog -o andornotout p11.v
C:\Users\Arnav\Desktop\Logic Design Verilog>vvp andornotout
VCD info: dumpfile p11.vcd opened for output.
C:\Users\Arnav\Desktop\Logic Design Verilog>gtkwave
GTKWave Analyzer v3.3.108 (w)1999-2020 BSI
GTKWAVE | Use the -h, --help command line flags to display help.
GTKWave Analyzer v3.3.108 (w)1999-2020 BSI
[0] start time.
[200] end time.
```
- GTKWave - C:\Users\Arnav\Desktop\Logic Design Verilog\p11.vcd:** Displays the simulation waveform. The signals shown are `a`, `b`, and `y[2:0]`. The waveform shows the values of `a` and `b` changing over time, and the resulting output `y[2:0]` for each combination of inputs.

Experiment 12

```
module adder(input a, input b,input cin, output cout);
    assign s = a^b^cin;
    assign cout= (a&b) |(a& cin) |(b & cin);
endmodule

module test1;
    reg a, b,cin;
    wire s , cout;
    adder FA (a,b,cin,cout);
    initial
    begin
        $dumpfile("FA1.vcd");
        $dumpvars(0,test1);
        a = 1;           //for first 5ns, a and b have values 1,1 and carry-in
        b = 1;           has value 0
        cin = 0; #5
        a = 1;           //for next 5ns, a and b have values 1,1 and carry-in
        b = 1;           has value 1
        cin = 1; #5
        a = 0;           //for next 5ns, a and b have values 0,1 and carry-in
        b = 1;           has value 0
        cin = 0; #5
        #10 $finish;    //the state is continued for 10ns ore after which time
                        duration is ended using $finish command
    end
endmodule
```

Compilation, Execution and Result of Simulation

