**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** |
|  | Write HDL implementation for the following Logic   1. AND/OR/NOT   Simulate the same using structural model and depict the timing diagram for valid inputs. |
| 2. | Write HDL implementation for the following Logic   1. NAND/NOR   Simulate the same using structural model and depict the timing diagram for valid inputs. |
| 3. | Write HDL implementation for the following AND-OR Combinational Logichttps://lh6.googleusercontent.com/D5cn5vNbGmGGFP-mcEuU8rVvLB1eOeW3VAu5MlFoBxIN-oEmkroBpXhXVdIQWnrv6H9aYrXWlAVwjHHN34szMTcftahgXYvtDv135jJIOlh4e03NvuUsxjGHX0zJcb8wWxx-_aQ    Simulate the same using structural model and depict the timing diagram for valid inputs. |
| 4. | Write HDL implementation for a 4:1 Multiplexer. Simulate the same using structural model and depict the timing diagram for valid inputs. |
| 5. | Write HDL implementation for a 2-to-4 decoder. Simulate the same using structural model and depict the timing diagram for valid inputs. |
| 6. | Write HDL implementation for a 4-to-2 encoder. Simulate the same using structural model and depict the timing diagram for valid inputs. |
|  | **CYCLE II**  **Behavior Modeling** |
| 7. | Write HDL implementation for a RS flip-flop using behavioral model. Simulate the same using structural model and depict the timing diagram for valid inputs. |
| 8. | Write HDL implementation for a JK flip-flop using behavioral model. Simulate the same using structural model and depict the timing diagram for valid inputs. |
| 9. | Write HDL implementation for a 3-bit up-counter using behavioral model. Simulate the same using structural model and depict the timing diagram for valid inputs. |
|  | **CYCLE III**  **Dataflow Modeling** |
| 10. | Write HDL implementation for AND/OR/NOT gates using data flow model. Simulate the same using 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**

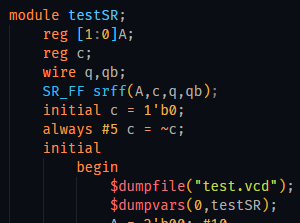
**EVALUATION SCHEME Tutorial Test: 1 hour**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Expt. No.** | **TITLE** | Max. Marks | **Marks Obtained** | **Signature** |
|  | AND/OR/NOT | 5 |  |  |
| 2. | NAND/NOR |  |
| 3. | Logic diagram |  |
| 4. | Multiplexer |  |
| 5. | Decoder |  |
| 6. | Encoder |  |
| 7. | RS |  |
| 8. | JK |  |
| 9. | Counter |  |
| 10. | AND/OR/NOT – data flow |  |
|  | **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 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.    
   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

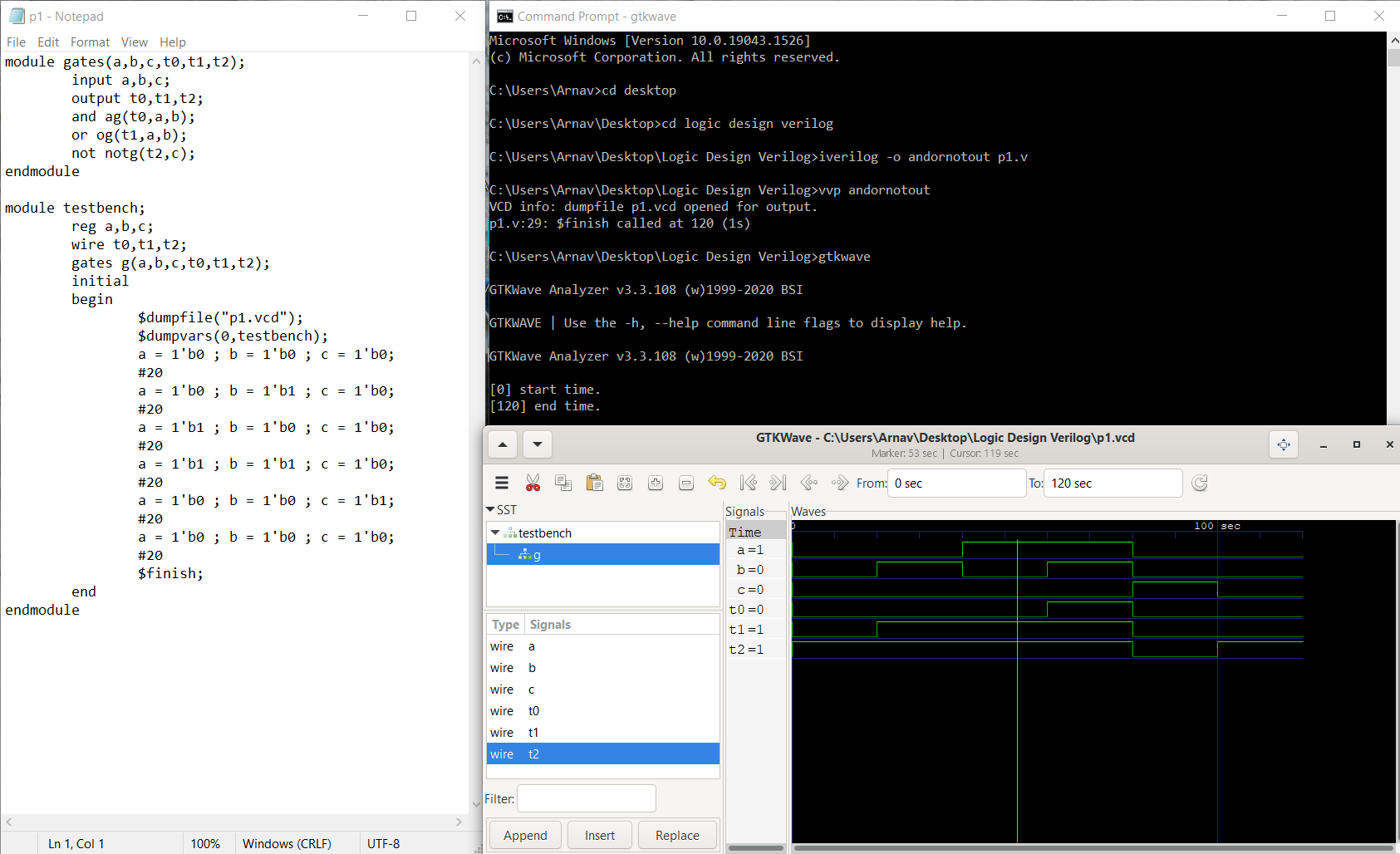
#20

$finish; //$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**

****

**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

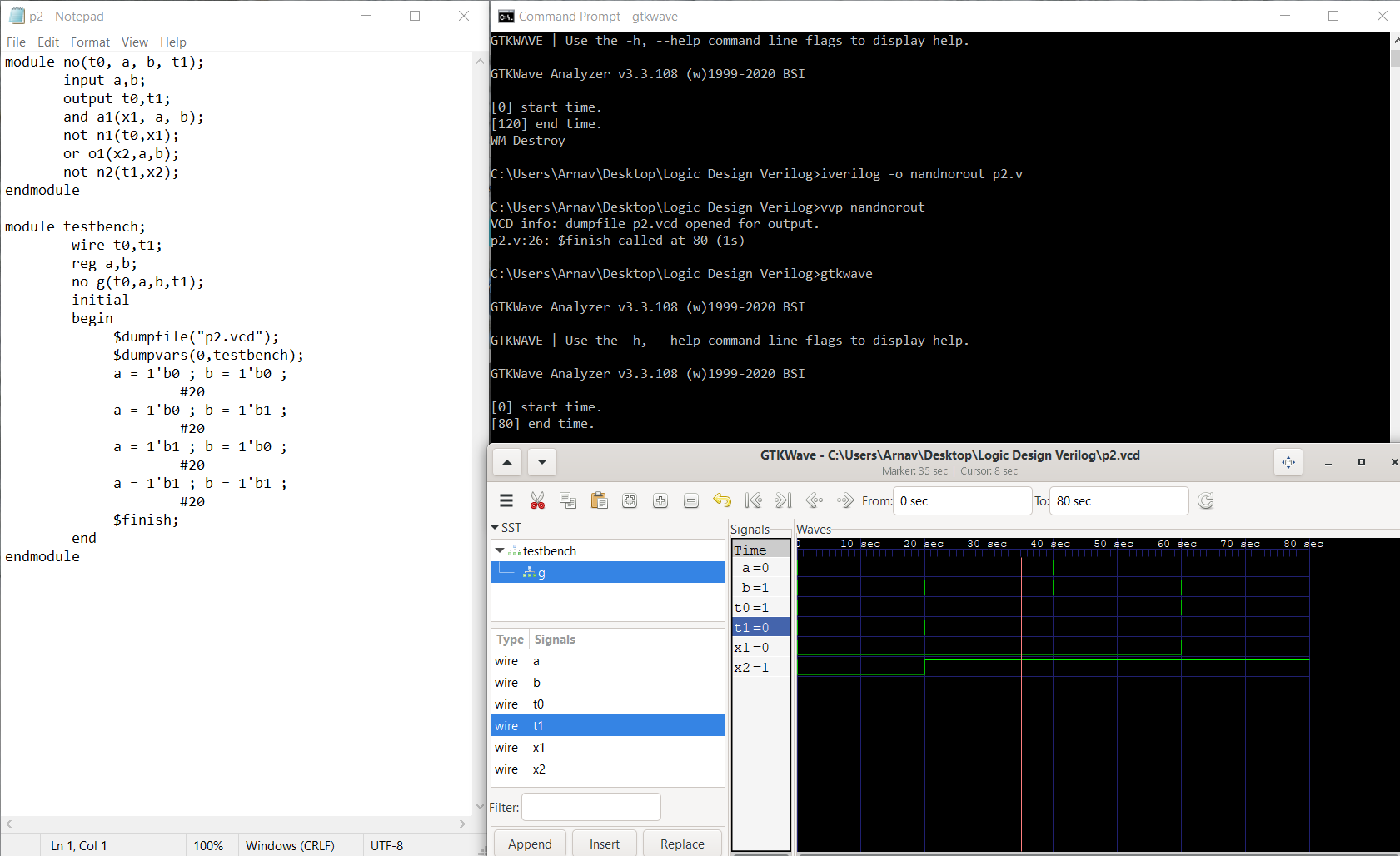
#20

$finish; //$finish denotes the end of time duration for

End which input values are given to a and b

endmodule

**Compilation, Execution and Result of Simulation**



**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

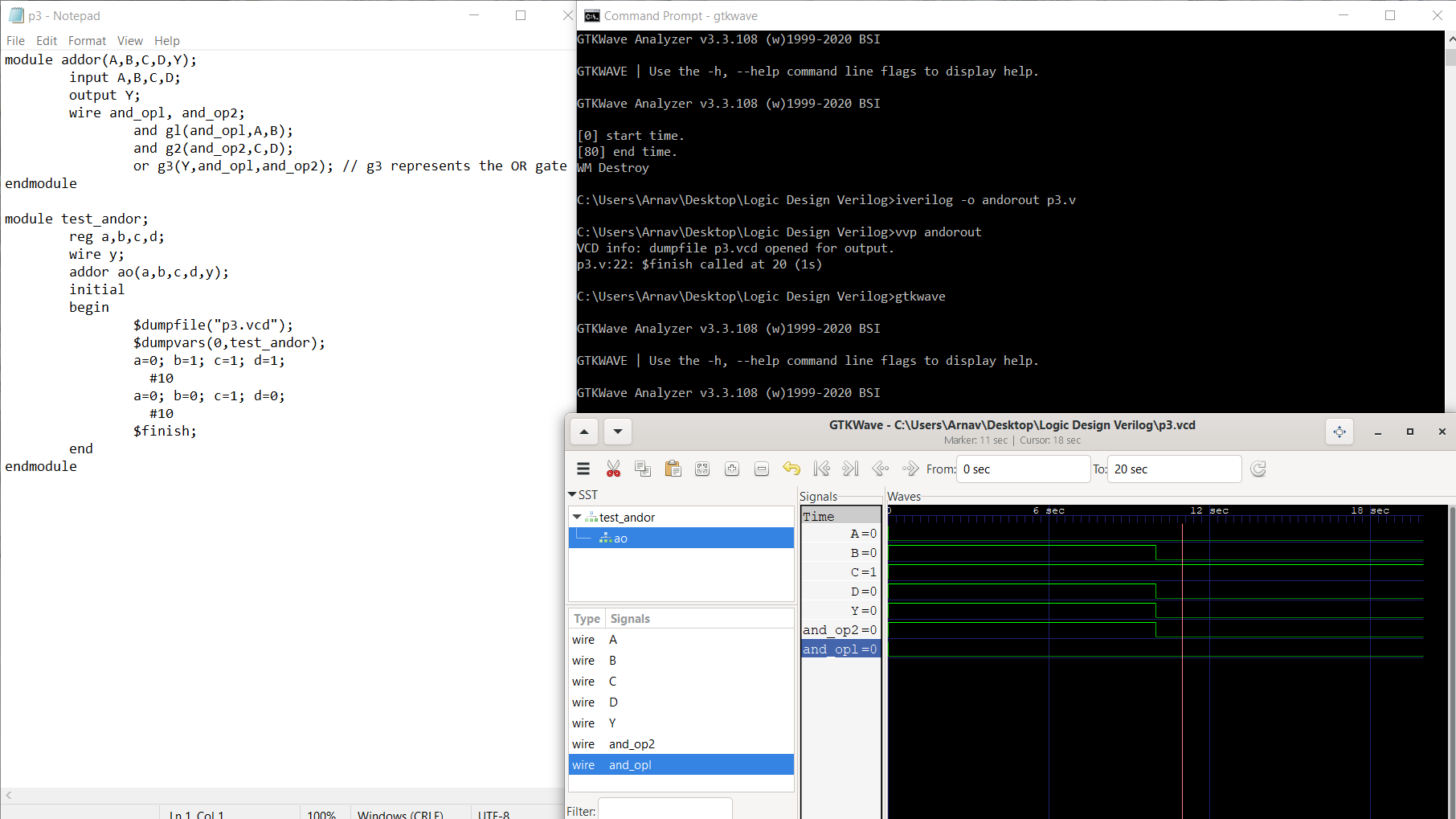
#10

$finish; //$finish denotes the end of time duration for

end which input values are given to a,b,c

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

#20 0,0,0,0 for the first 20ns and sel1=0,sel2=0

t1=1;t2=0;t3=0;t4=0;sel1=0;sel2=0; //inputs t1,t2,t3,t4 are given values

#20 1,0,0,0 for the next 20ns and sel1=0,sel2=0

t1=0;t2=0;t3=0;t4=0;sel1=0;sel2=1; //inputs t1,t2,t3,t4 are given values

#20 0,0,0,0 for the next 20ns and sel1=0,sel2=1

t1=1;t2=1;t3=0;t4=0;sel1=0;sel2=1; //inputs t1,t2,t3,t4 are given values

#20 0,0,0,0 for the next 20ns and sel1=1,sel2=1

t1=0;t2=0;t3=1;t4=0;sel1=1;sel2=0; //inputs t1,t2,t3,t4 are given values

#20 0,0,0,0 for the next 20ns and sel1=1,sel2=0

t1=1;t2=1;t3=0;t4=1;sel1=1;sel2=0; //inputs t1,t2,t3,t4 are given values

#20 0,0,0,0 for the next 20ns and sel1=1,sel2=0

t1=1;t2=0;t3=0;t4=1;sel1=1;sel2=1; //inputs t1,t2,t3,t4 are given values

#20 0,0,0,0 for the next 20ns and sel1=1,sel2=1

t1=0;t2=1;t3=1;t4=0;sel1=1;sel2=1; //inputs t1,t2,t3,t4 are given values

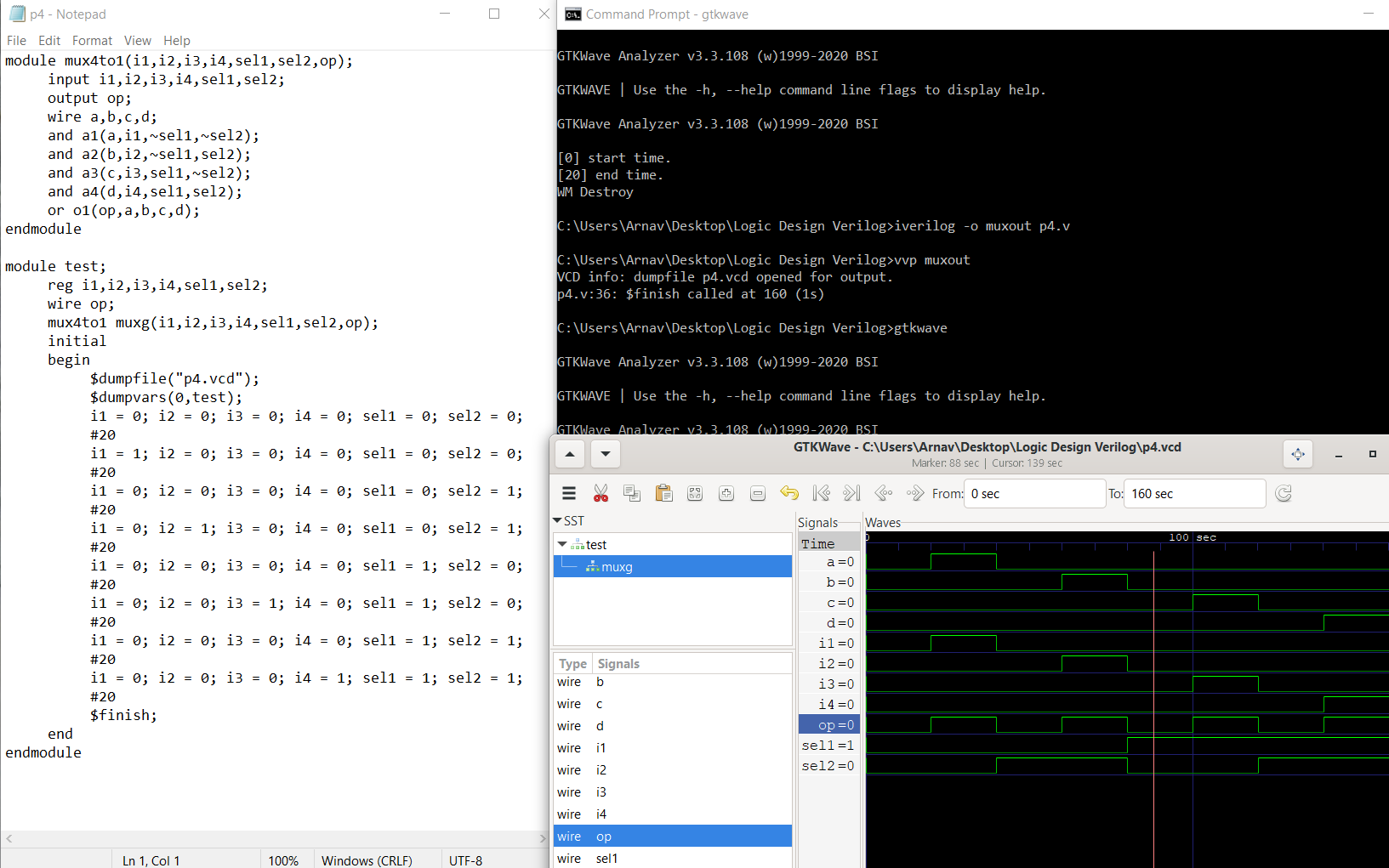
#20 0,0,0,0 for the next 20ns and sel1=1,sel2=1

$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**



**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

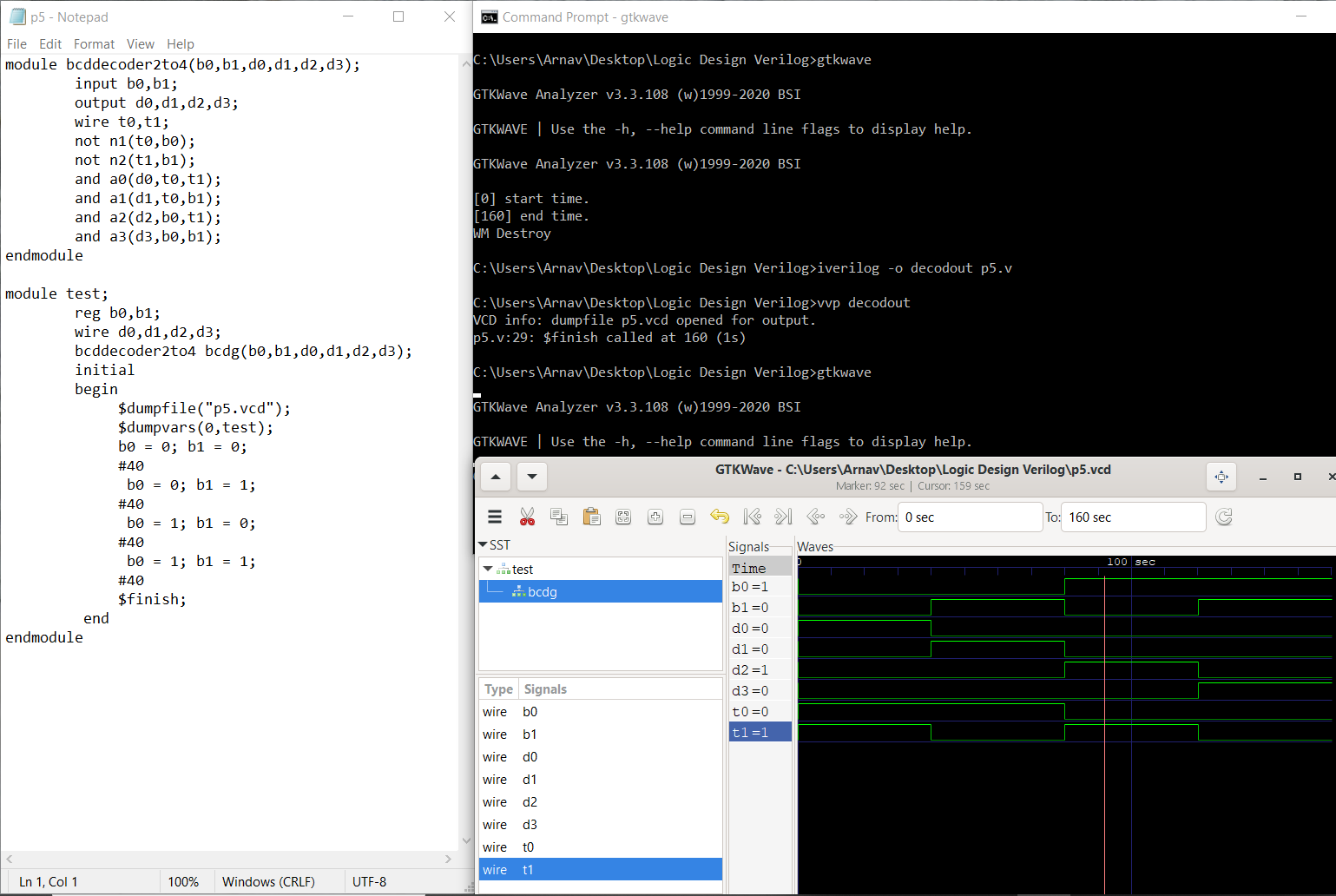
#40

$finish;

end

endmodule

**Compilation, Execution and Result of Simulation**

****

**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

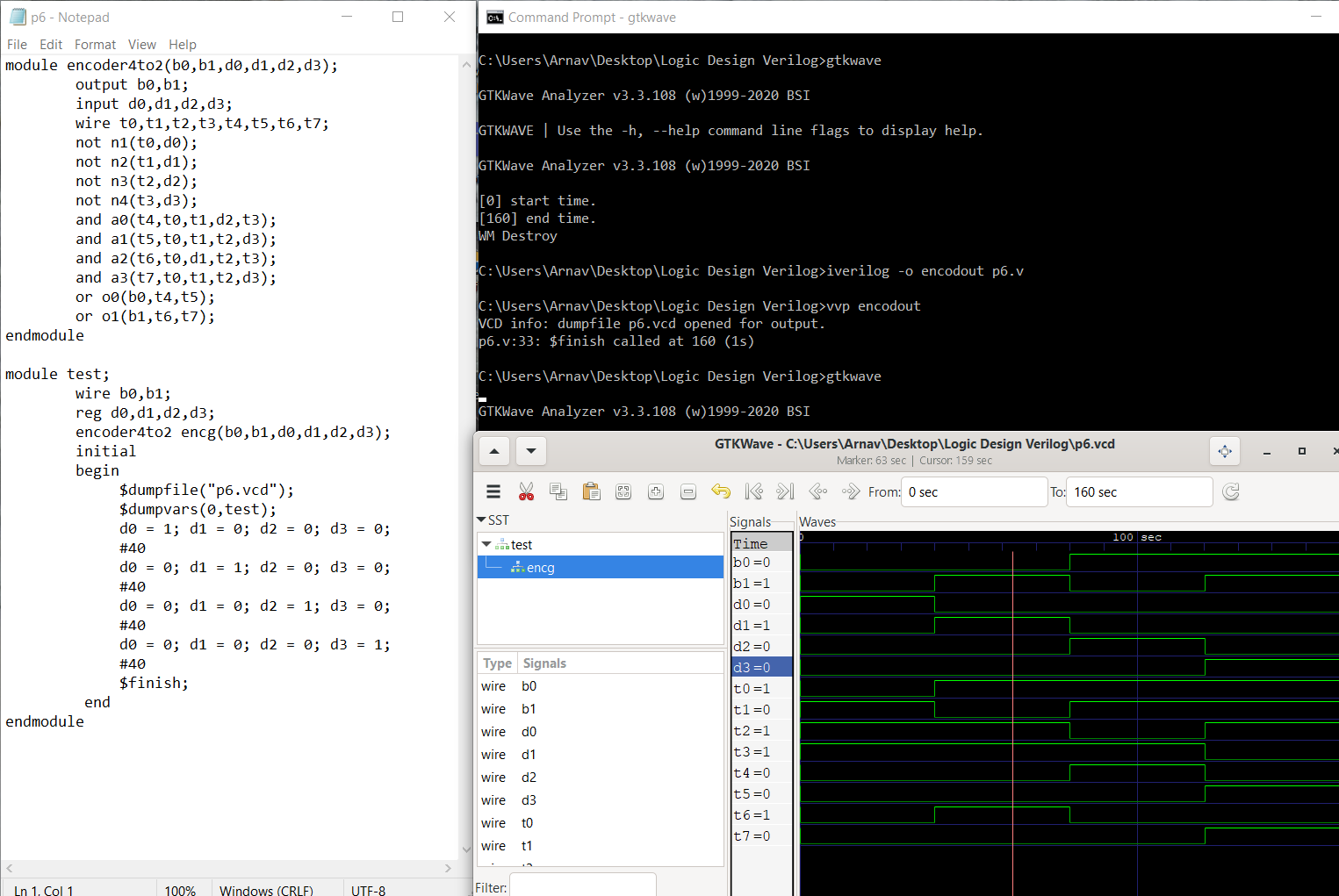
d0 = 0;d1 = 0;d2 = 0;d3 = 1; //for the next 40ns inputs d0,d1,d2,d3 have

#40; the values 0,0,0,1

end

endmodule

**Compilation, Execution and Result of Simulation**



**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;

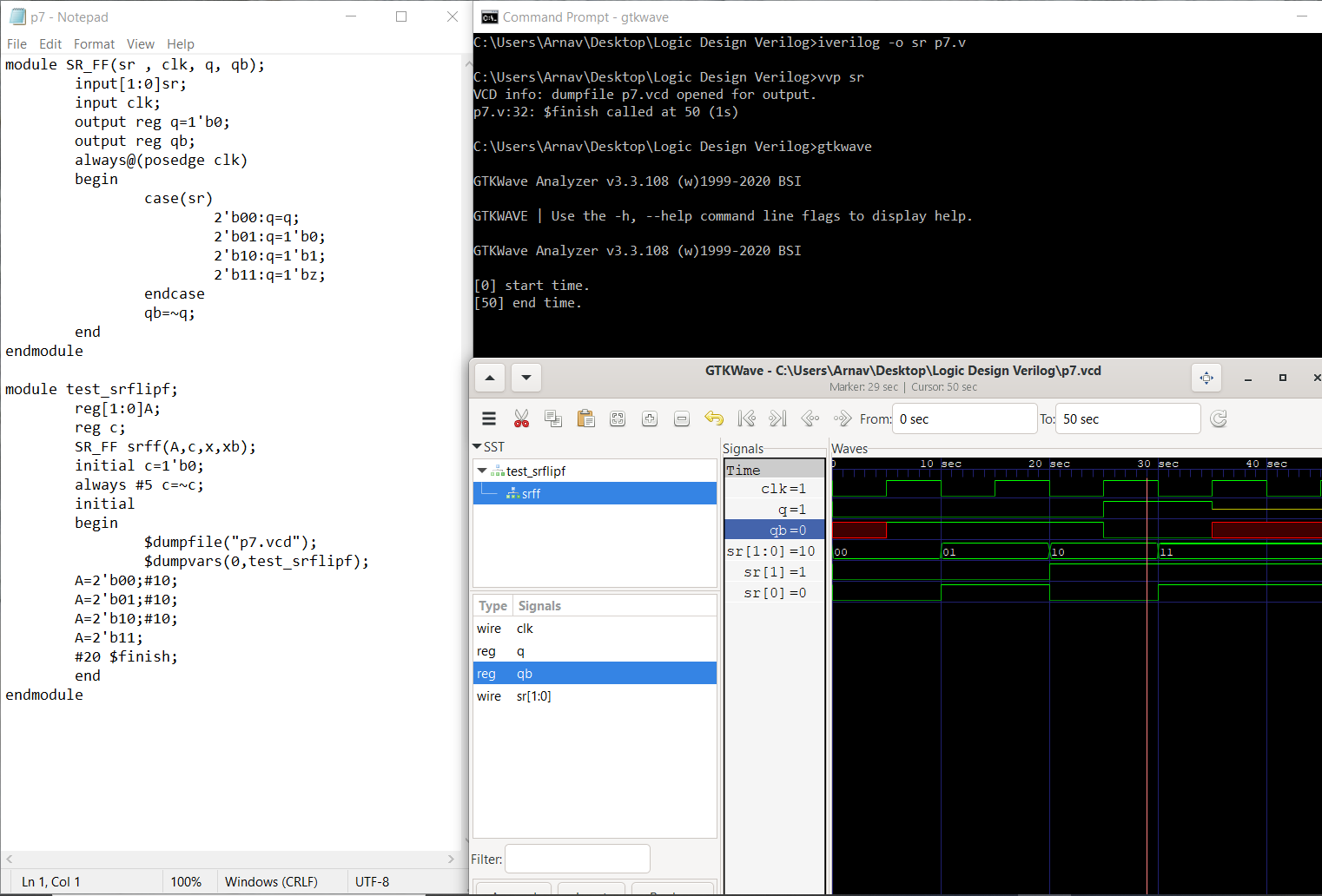
A=2'b11;

#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 elemnts synthesized for sequential ciruit

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;

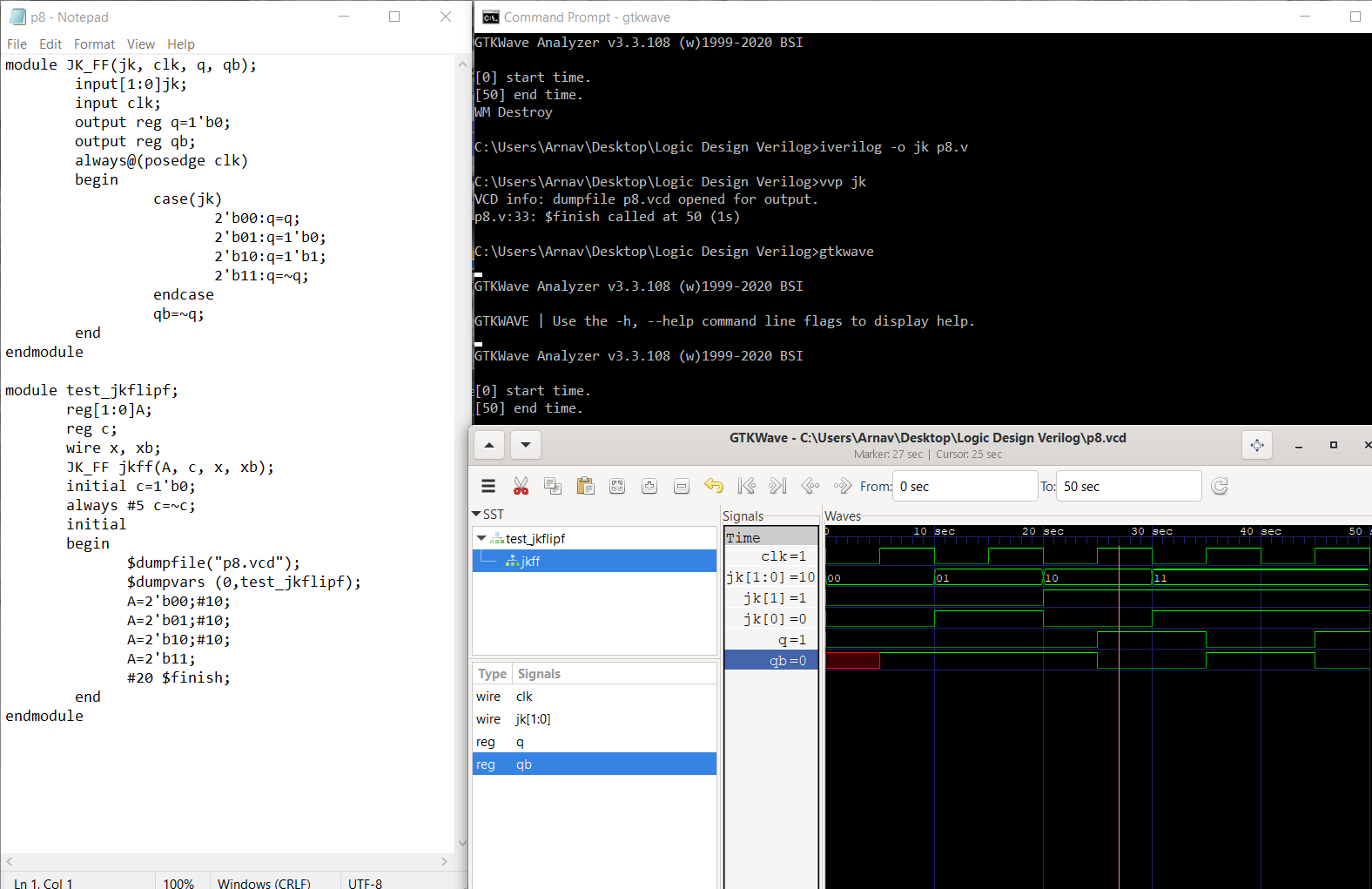
A=2'b11;

#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;

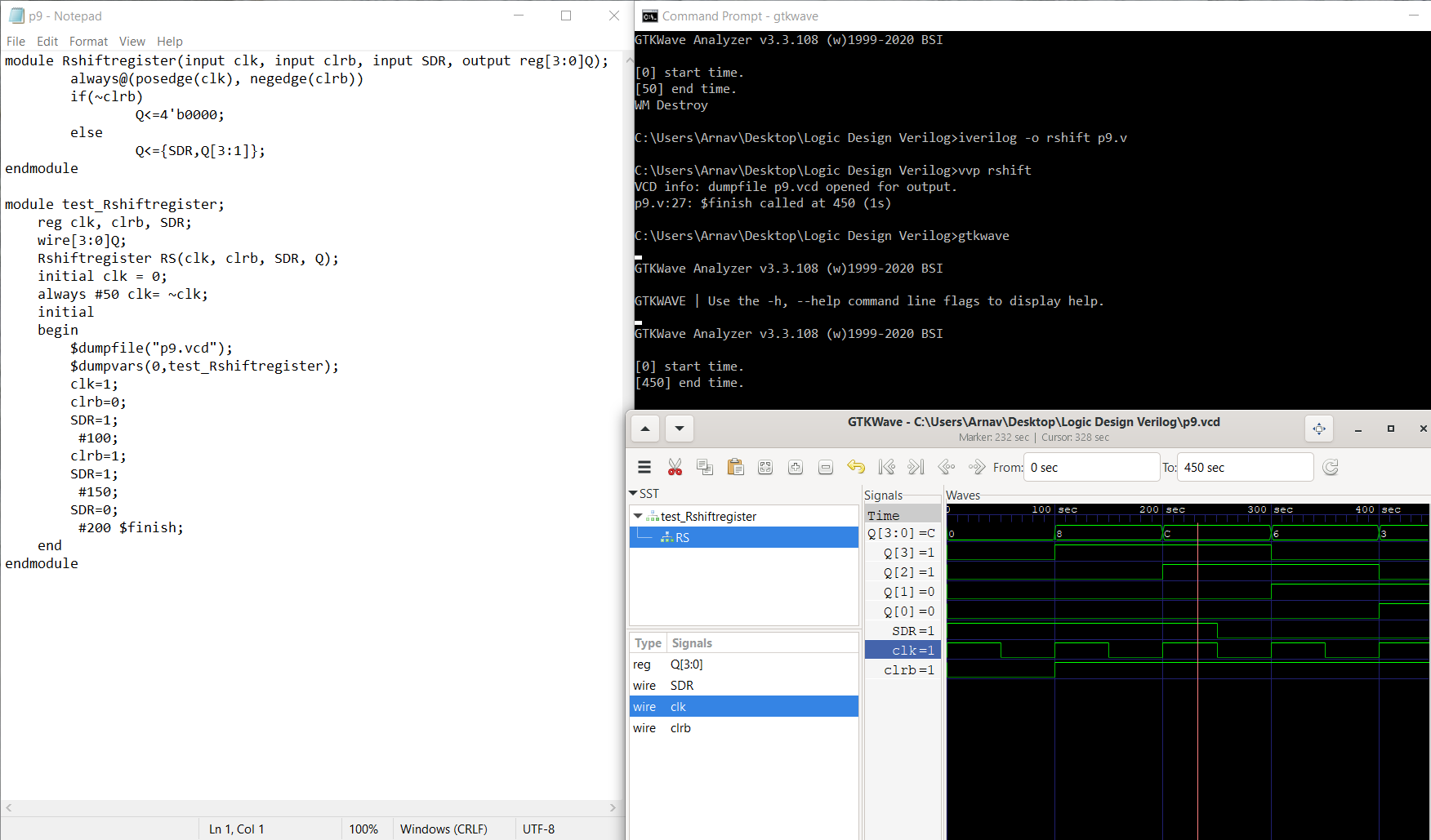
SDR=0;

#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 100ns 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;

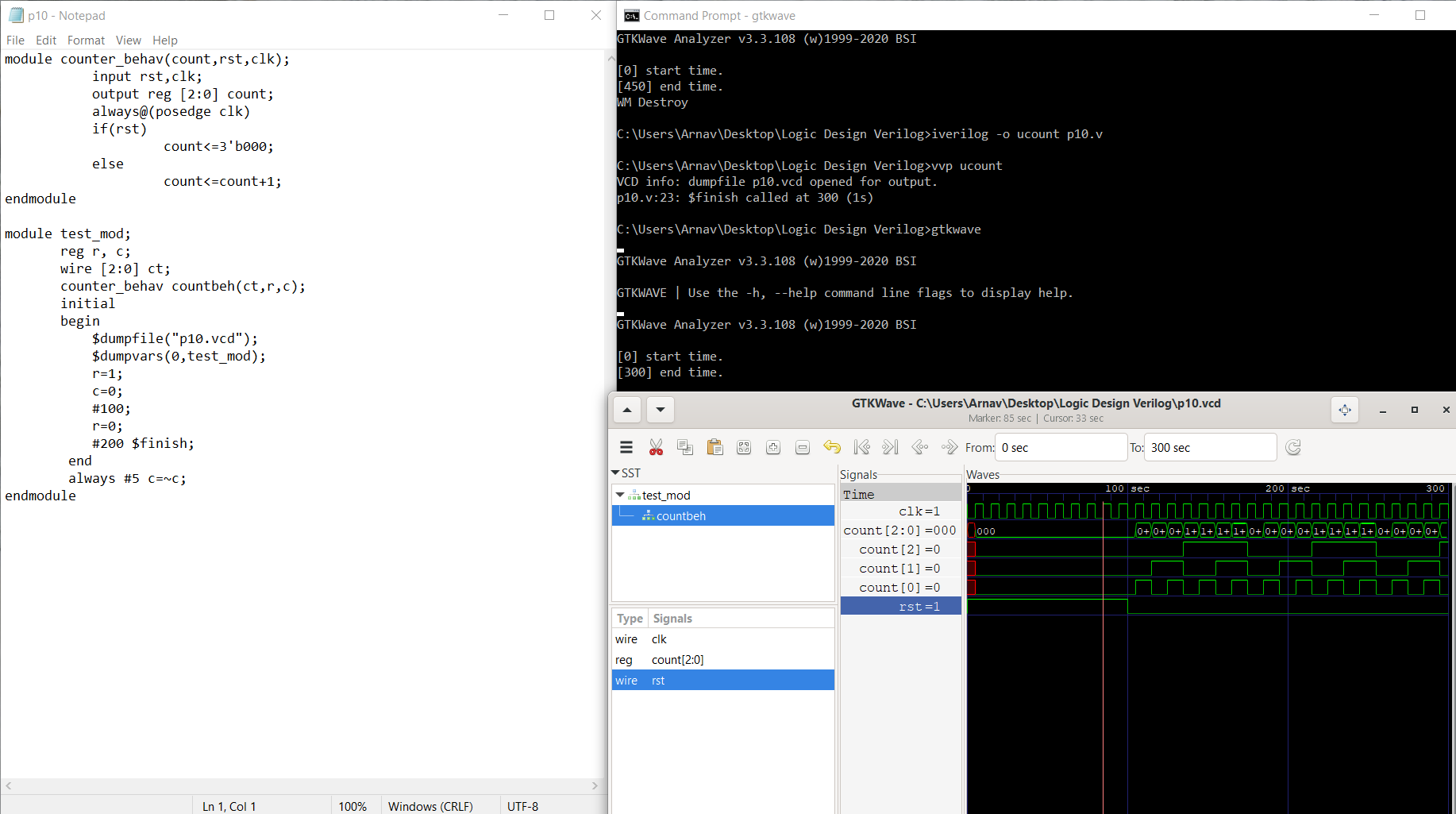
#200 $finish;

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

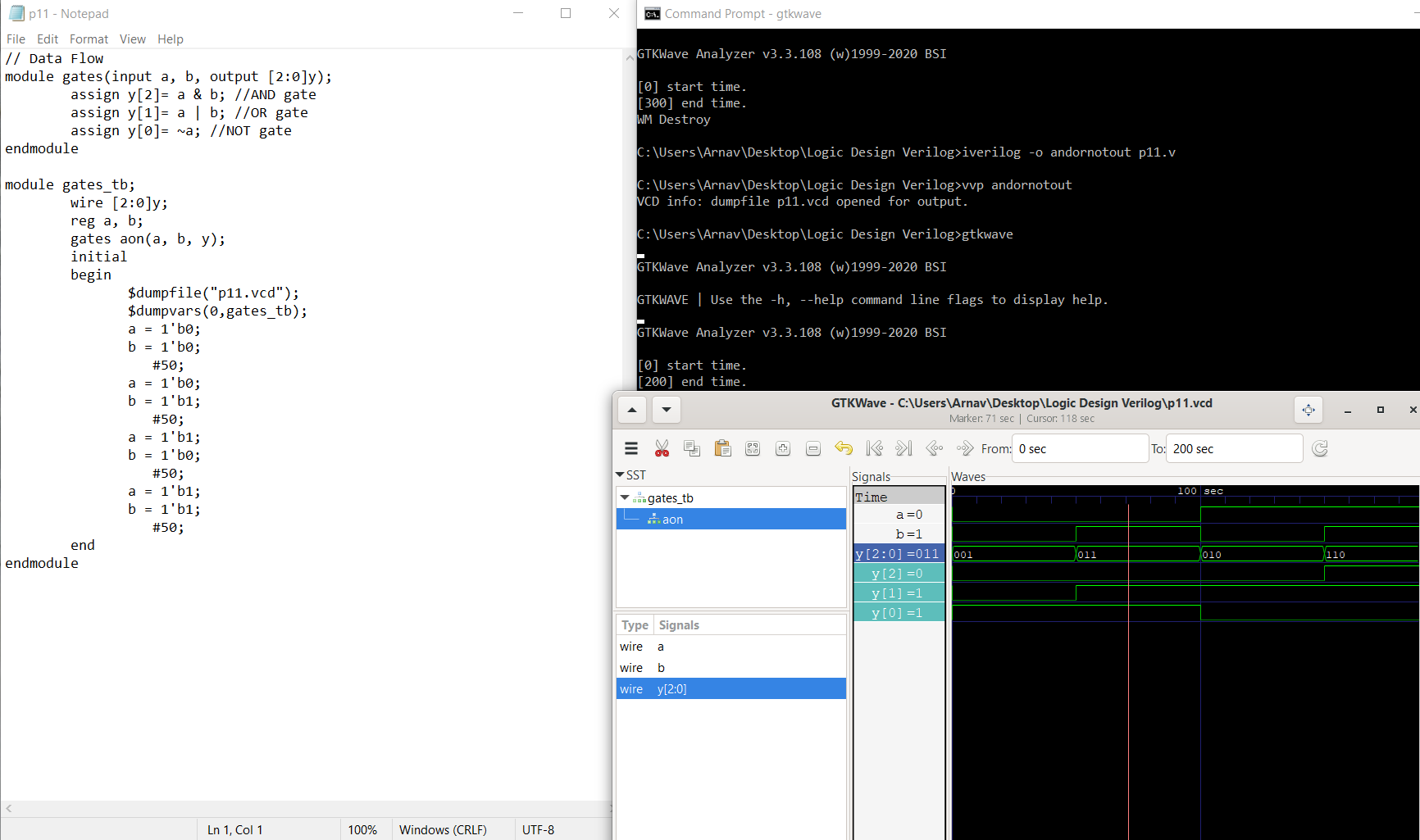
b = 1'b1; single bit)

#50;

end

endmodule

**Compilation, Execution and Result of Simulation**

****

**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

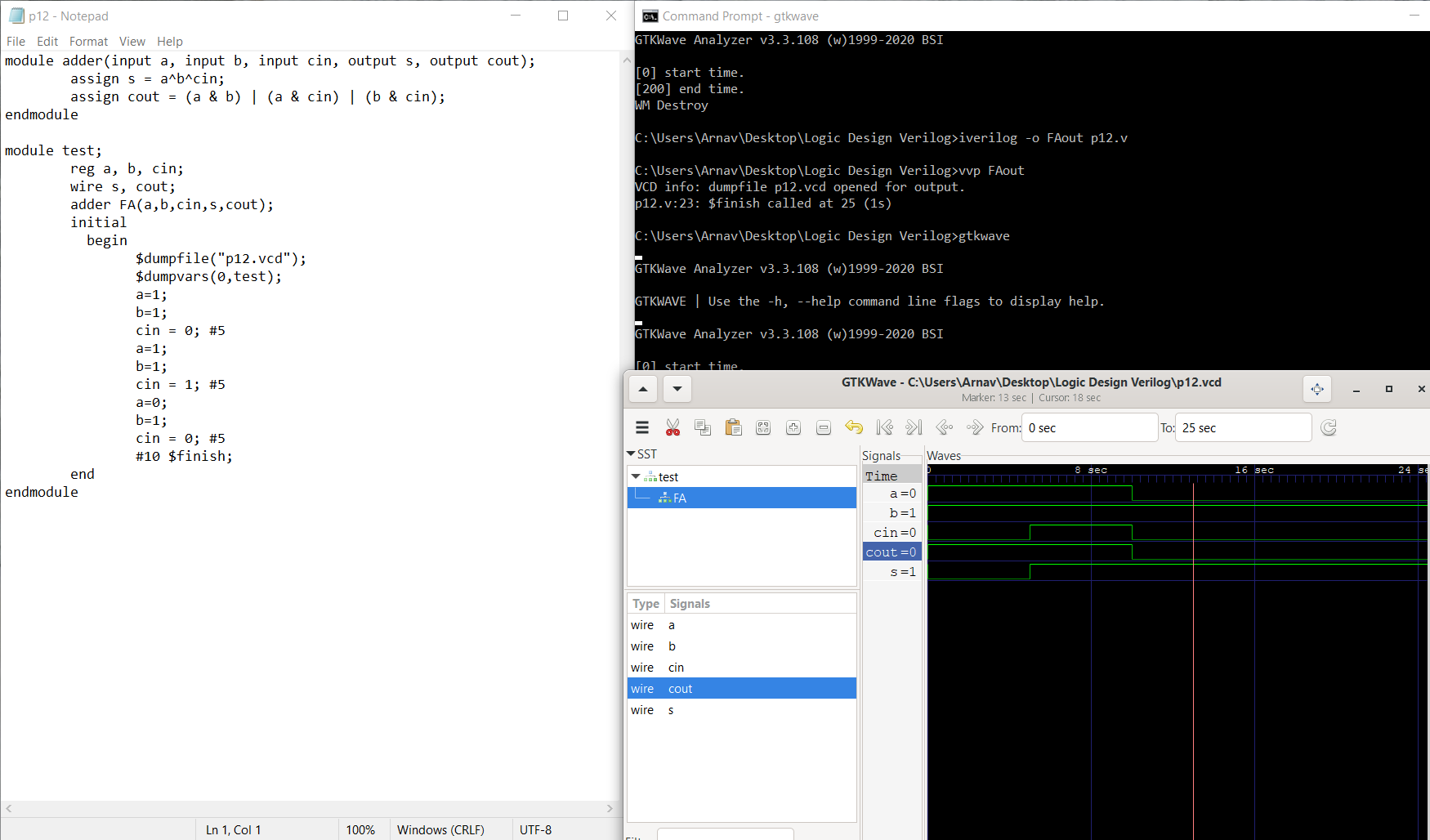
cin = 0; #5

#10 $finish; //the state is continued for 10ns ore after which time

end duration is ended using $finish command

endmodule

**Compilation, Execution and Result of Simulation**

****