B.M.S. College of Engineering

(Autonomous Institution affiliated to VTU, Belagavi)

Department of Computer Science and Engineering

AAT

Verilog Laboratory Report

23CS3PCLOD

(September 2025-January 2026)

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

Laboratory Certificate

This is to certify that Chetan Barakki Satish Kumar, Chinmay G Hegde, D Murali Satya Suhas, Darshan R Palrecha satisfactorily completed the course of Experiments in Practical Logic Design (Verilog) prescribed by the Department during the odd semester 2025-26.

Name of the Candidate: Chetan Barakki Satish Kumar, Chinmay G Hegde, D Murali Satya Suhas, Darshan R Palrecha

USN No.: **1BM24CS081, 1BM24CS083, 1BM24CS086, 1BM24CS091**

Semester: **III** Section: **G**

Marks	
Max. Marks	Obtained
10	
Marks in Words	

Signature of the staff in-charge **Head of the Department**

Date:

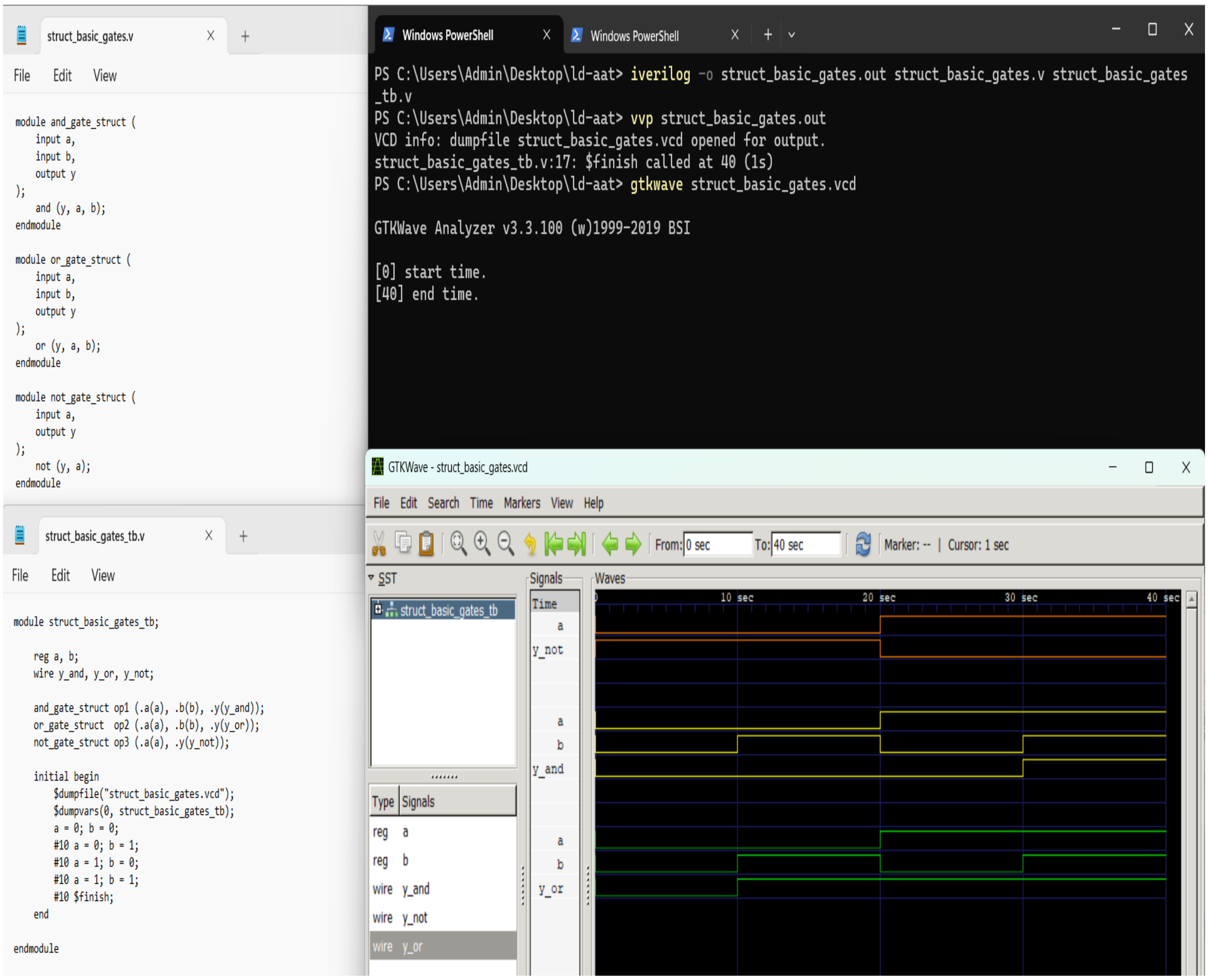
CYCLE 1: STRUCTURAL MODELLING

Experiment 1: Write HDL implementation for the following Logic AND/OR/NOT. Simulate the same using structural model and depict the timing diagram for valid inputs.

```
module and_gate_struct (
   input a,
   input b,
   output y
);
   and (y, a, b);
endmodule

module or_gate_struct (
   input a,
   input b,
   output y
);
   or (y, a, b);
endmodule

module not_gate_struct (
   input a,
   output y
);
   not (y, a);
endmodule

// testbench
module struct_basic_gates_tb;

   reg a, b;
   wire y_and, y_or, y_not;

   and_gate_struct op1 (.a(a), .b(b), .y(y_and));
   or_gate_struct  op2 (.a(a), .b(b), .y(y_or));
   not_gate_struct op3 (.a(a), .y(y_not));

   initial begin
      $dumpfile("struct_basic_gates.vcd");
```

```
        $dumpvars(0, struct_basic_gates_tb);
        a = 0; b = 0;
        #10 a = 0; b = 1;
        #10 a = 1; b = 0;
        #10 a = 1; b = 1;
        #10 $finish;
    end
endmodule
```

Compilation, Execution and Result of Simulation



struct_basic_gates.v

```
module and_gate_struct (
    input a,
    input b,
    output y
);
    and (y, a, b);
endmodule

module or_gate_struct (
    input a,
    input b,
    output y
);
    or (y, a, b);
endmodule

module not_gate_struct (
    input a,
    output y
);
    not (y, a);
endmodule
```

struct_basic_gates_tb.v

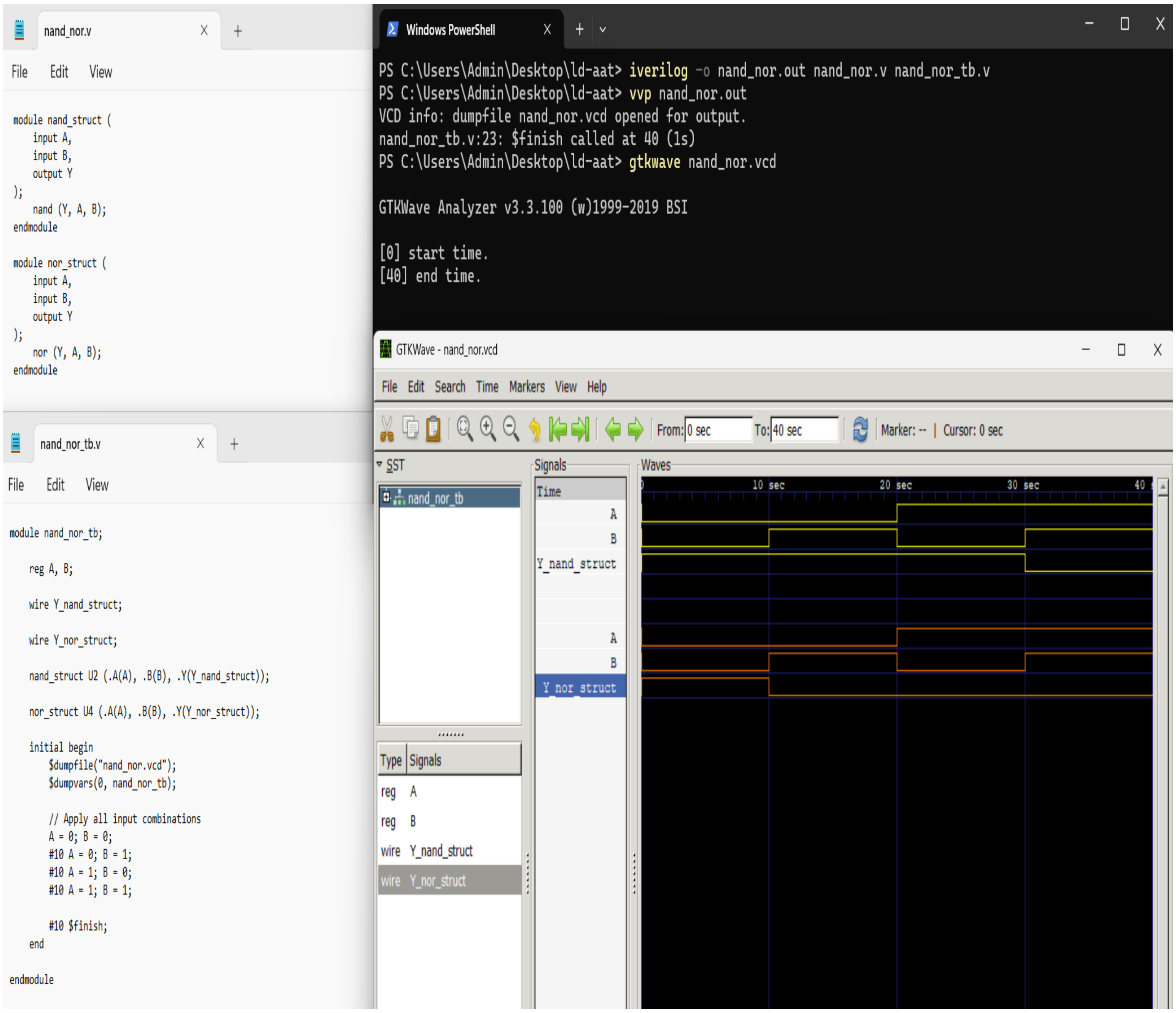
```
module struct_basic_gates_tb;

    reg a, b;
    wire y_and, y_or, y_not;

    and_gate_struct op1 (.a(a), .b(b), .y(y_and));
    or_gate_struct  op2 (.a(a), .b(b), .y(y_or));
    not_gate_struct op3 (.a(a), .y(y_not));

    initial begin
        $dumpfile("struct_basic_gates.vcd");
        $dumpvars(0, struct_basic_gates_tb);
        a = 0; b = 0;
        #10 a = 0; b = 1;
        #10 a = 1; b = 0;
        #10 a = 1; b = 1;
        #10 $finish;
    end

endmodule
```

```
PS C:\Users\Admin\Desktop\ld-aat> iverilog -o struct_basic_gates.out struct_basic_gates.v struct_basic_gates
_tb.v
PS C:\Users\Admin\Desktop\ld-aat> vvp struct_basic_gates.out
VCD info: dumpfile struct_basic_gates.vcd opened for output.
struct_basic_gates_tb.v:17: $finish called at 40 (1s)
PS C:\Users\Admin\Desktop\ld-aat> gtkwave struct_basic_gates.vcd

GTKWave Analyzer v3.3.100 (w)1999-2019 BSI

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

Experiment 2: Write HDL implementation for the following Logic NAND/NOR. Simulate the same using structural model and depict the timing diagram for valid inputs.

```
module nand_struct (
   input A,
   input B,
   output Y
);
   nand (Y, A, B);
endmodule

module nor_struct (
   input A,
   input B,
   output Y
);
   nor (Y, A, B);
endmodule


// testbench
module nand_nor_tb;

   reg A, B;

   wire Y_nand_struct;

   wire Y_nor_struct;

   nand_struct U2 (.A(A), .B(B), .Y(Y_nand_struct));

   nor_struct U4 (.A(A), .B(B), .Y(Y_nor_struct));

   initial begin
      $dumpfile("nand_nor.vcd");
      $dumpvars(0, nand_nor_tb);

      // Apply all input combinations
      A = 0; B = 0;
      #10 A = 0; B = 1;
      #10 A = 1; B = 0;
      #10 A = 1; B = 1;

      #10 $finish;
   end

endmodule
```

Compilation, Execution and Result of Simulation

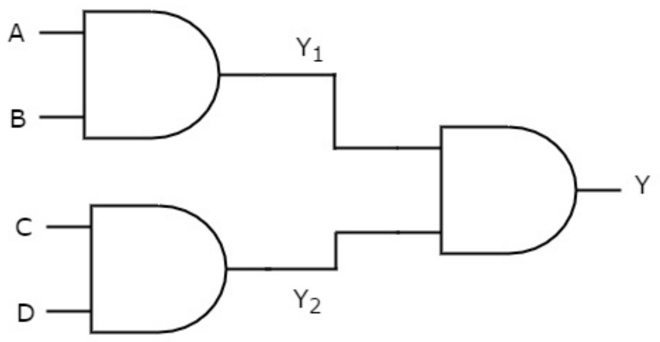


```
module nand_struct (
    input A,
    input B,
    output Y
);
    nand (Y, A, B);
endmodule

module nor_struct (
    input A,
    input B,
    output Y
);
    nor (Y, A, B);
endmodule
```

```
module nand_nor_tb;

    reg A, B;

    wire Y_nand_struct;

    wire Y_nor_struct;

    nand_struct U2 (.A(A), .B(B), .Y(Y_nand_struct));

    nor_struct U4 (.A(A), .B(B), .Y(Y_nor_struct));

    initial begin
        $dumpfile("nand_nor.vcd");
        $dumpvars(0, nand_nor_tb);

        // Apply all input combinations
        A = 0; B = 0;
        #10 A = 0; B = 1;
        #10 A = 1; B = 0;
        #10 A = 1; B = 1;

        #10 $finish;
    end

endmodule
```

```
PS C:\Users\Admin\Desktop\ld-aat> iverilog -o nand_nor.out nand_nor.v nand_nor_tb.v
PS C:\Users\Admin\Desktop\ld-aat> vvp nand_nor.out
VCD info: dumpfile nand_nor.vcd opened for output.
nand_nor_tb.v:23: $finish called at 40 (1s)
PS C:\Users\Admin\Desktop\ld-aat> gtkwave nand_nor.vcd

GTKWave Analyzer v3.3.100 (w)1999-2019 BSI

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

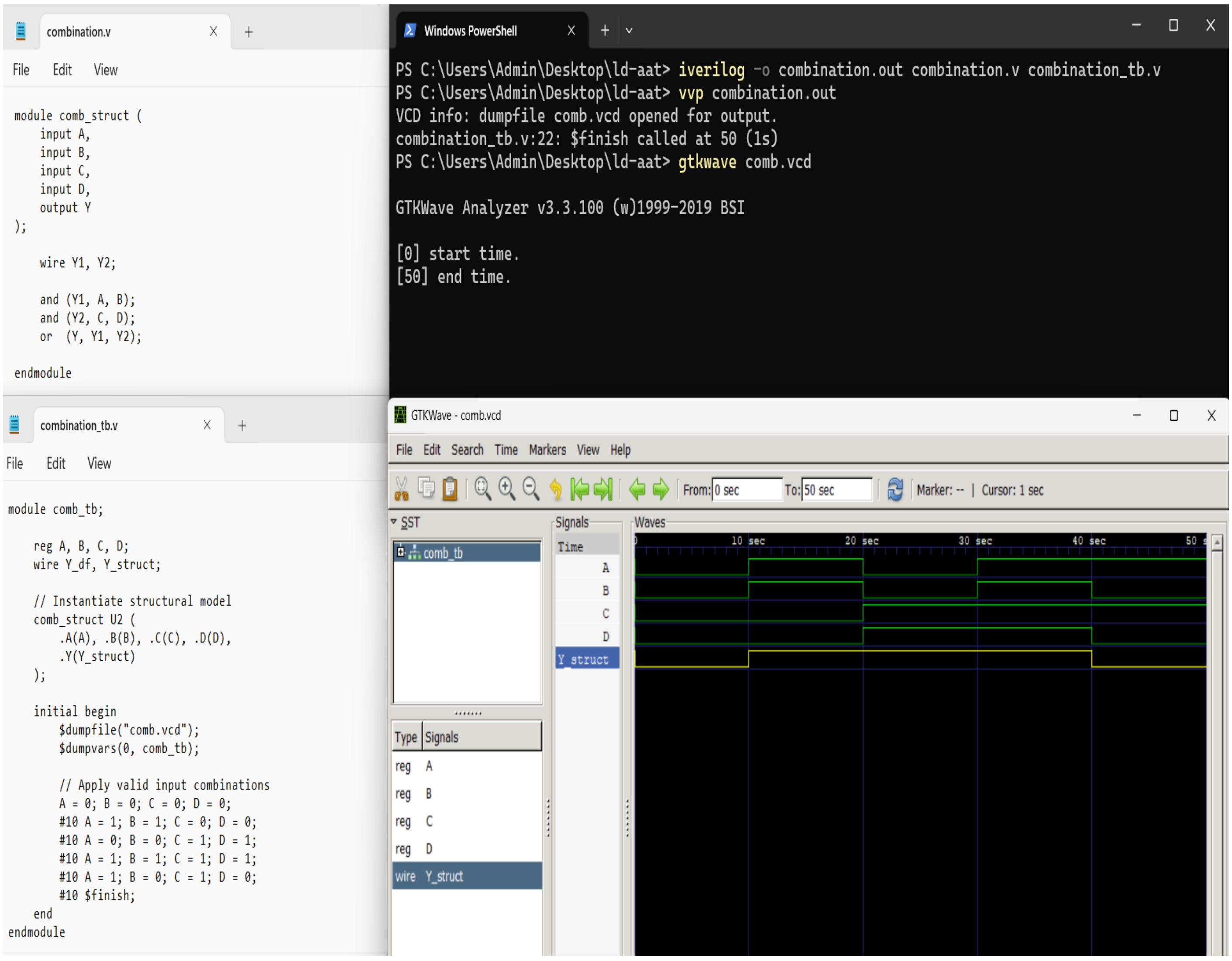
Experiment 3: Write HDL implementation for the following AND-OR Combinational Logic Y = (AB)(CD). Simulate the same using structural model and depict the timing diagram for valid inputs.



```
module comb_struct (
   input A,
   input B,
   input C,
   input D,
   output Y
);

   wire Y1, Y2;

   and (Y1, A, B);
   and (Y2, C, D);
   or  (Y, Y1, Y2);

endmodule

// testbench
module comb_tb;

   reg A, B, C, D;
   wire Y_df, Y_struct;

   // Instantiate structural model
   comb_struct U2 (
      .A(A), .B(B), .C(C), .D(D),
      .Y(Y_struct)
   );

   initial begin
      $dumpfile("comb.vcd");
      $dumpvars(0, comb_tb);

      // Apply valid input combinations
      A = 0; B = 0; C = 0; D = 0;
```

```
        #10 A = 1; B = 1; C = 0; D = 0;
        #10 A = 0; B = 0; C = 1; D = 1;
        #10 A = 1; B = 1; C = 1; D = 1;
        #10 A = 1; B = 0; C = 1; D = 0;
        #10 $finish;
    end
endmodule
```

Compilation, Execution and Result of Simulation



```
module comb_struct (
    input A,
    input B,
    input C,
    input D,
    output Y
);

    wire Y1, Y2;

    and (Y1, A, B);
    and (Y2, C, D);
    or  (Y, Y1, Y2);

endmodule
```

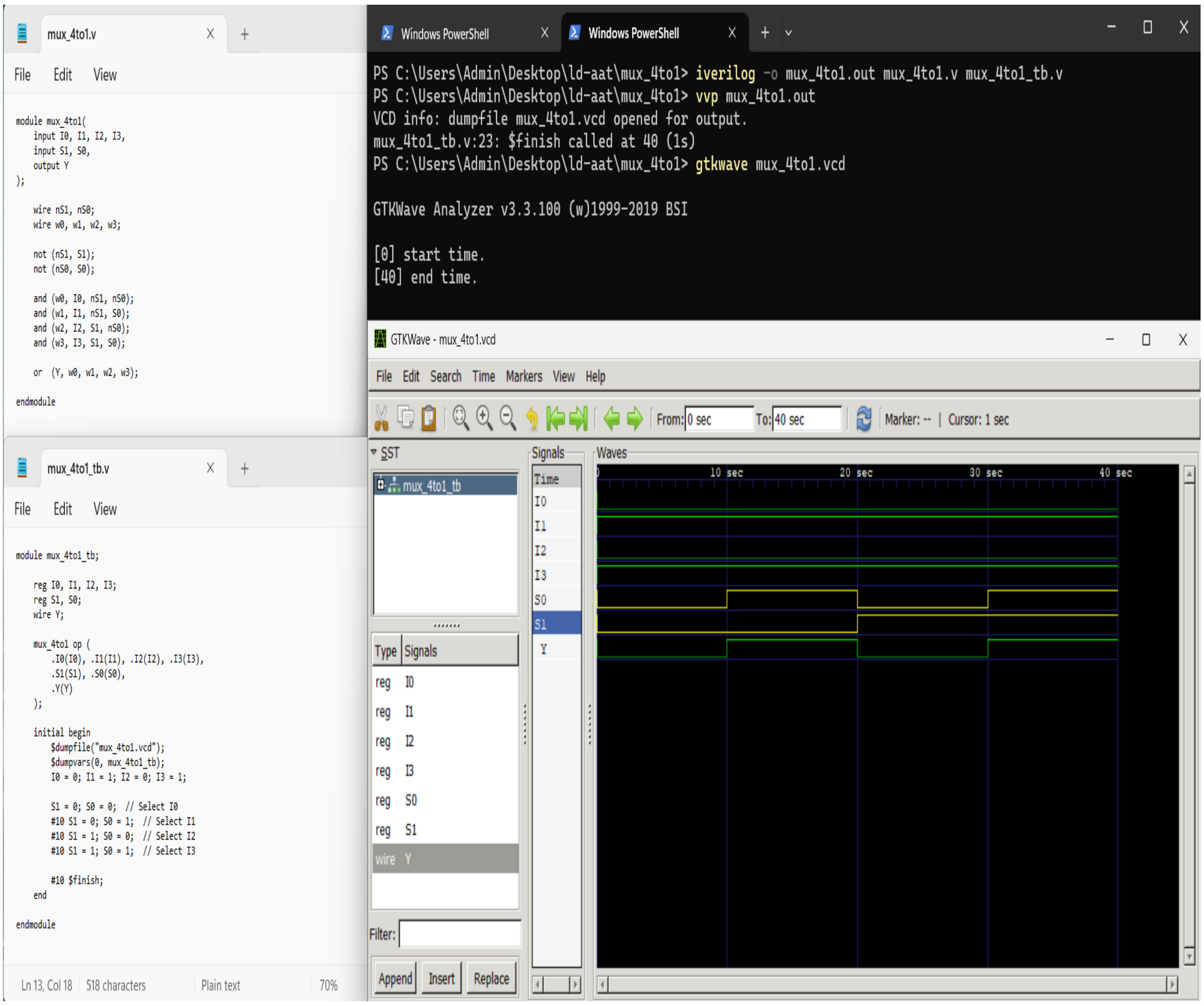
```
module comb_tb;

    reg A, B, C, D;
    wire Y_df, Y_struct;

    // Instantiate structural model
    comb_struct U2 (
        .A(A), .B(B), .C(C), .D(D),
        .Y(Y_struct)
    );

    initial begin
        $dumpfile("comb.vcd");
        $dumpvars(0, comb_tb);

        // Apply valid input combinations
        A = 0; B = 0; C = 0; D = 0;
        #10 A = 1; B = 1; C = 0; D = 0;
        #10 A = 0; B = 0; C = 1; D = 1;
        #10 A = 1; B = 1; C = 1; D = 1;
        #10 A = 1; B = 0; C = 1; D = 0;
        #10 $finish;
    end
endmodule
```

```
PS C:\Users\Admin\Desktop\ld-aat> iverilog -o combination.out combination.v combination_tb.v
PS C:\Users\Admin\Desktop\ld-aat> vvp combination.out
VCD info: dumpfile comb.vcd opened for output.
combination_tb.v:22: $finish called at 50 (1s)
PS C:\Users\Admin\Desktop\ld-aat> gtkwave comb.vcd

GTKWave Analyzer v3.3.100 (w)1999-2019 BSI

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

Experiment 4: Write HDL implementation for a 4:1 Multiplexer. Simulate the same using

```
module mux_4to1(
    input I0, I1, I2, I3,
    input S1, S0,
    output Y
);
    wire nS1, nS0;
    wire w0, w1, w2, w3;

    not (nS1, S1);
    not (nS0, S0);
    and (w0, I0, nS1, nS0);
    and (w1, I1, nS1, S0);
    and (w2, I2, S1, nS0);
    and (w3, I3, S1, S0);

    or  (Y, w0, w1, w2, w3);
endmodule

// testbench
module mux_4to1_tb;
    reg I0, I1, I2, I3;
    reg S1, S0;
    wire Y;

    mux_4to1 op (
        .I0(I0), .I1(I1), .I2(I2), .I3(I3),
        .S1(S1), .S0(S0),
        .Y(Y)
    );
    initial begin
        $dumpfile("mux_4to1.vcd");
        $dumpvars(0, mux_4to1_tb);
        I0 = 0; I1 = 1; I2 = 0; I3 = 1;
        S1 = 0; S0 = 0;  // Select I0
        #10 S1 = 0; S0 = 1;  // Select I1
        #10 S1 = 1; S0 = 0;  // Select I2
        #10 S1 = 1; S0 = 1;  // Select I3

        #10 $finish;
    end
endmodule
```

Compilation, Execution and Result of Simulation



mux_4to1.v

```
module mux_4to1(
    input I0, I1, I2, I3,
    input S1, S0,
    output Y
);

    wire nS1, nS0;
    wire w0, w1, w2, w3;

    not (nS1, S1);
    not (nS0, S0);

    and (w0, I0, nS1, nS0);
    and (w1, I1, nS1, S0);
    and (w2, I2, S1, nS0);
    and (w3, I3, S1, S0);

    or  (Y, w0, w1, w2, w3);

endmodule
```

mux_4to1_tb.v

```
module mux_4to1_tb;

    reg I0, I1, I2, I3;
    reg S1, S0;
    wire Y;

    mux_4to1 op (
        .I0(I0), .I1(I1), .I2(I2), .I3(I3),
        .S1(S1), .S0(S0),
        .Y(Y)
    );

    initial begin
        $dumpfile("mux_4to1.vcd");
        $dumpvars(0, mux_4to1_tb);
        I0 = 0; I1 = 1; I2 = 0; I3 = 1;

        S1 = 0; S0 = 0;  // Select I0
        #10 S1 = 0; S0 = 1;  // Select I1
        #10 S1 = 1; S0 = 0;  // Select I2
        #10 S1 = 1; S0 = 1;  // Select I3

        #10 $finish;
    end

endmodule
```

Ln 13, Col 18 | 518 characters | Plain text | 70%

Windows PowerShell

```
PS C:\Users\Admin\Desktop\ld-aat\mux_4to1> iverilog -o mux_4to1.out mux_4to1.v mux_4to1_tb.v
PS C:\Users\Admin\Desktop\ld-aat\mux_4to1> vvp mux_4to1.out
VCD info: dumpfile mux_4to1.vcd opened for output.
mux_4to1_tb.v:23: $finish called at 40 (1s)
PS C:\Users\Admin\Desktop\ld-aat\mux_4to1> gtkwave mux_4to1.vcd

GTKWave Analyzer v3.3.100 (w)1999-2019 BSI

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

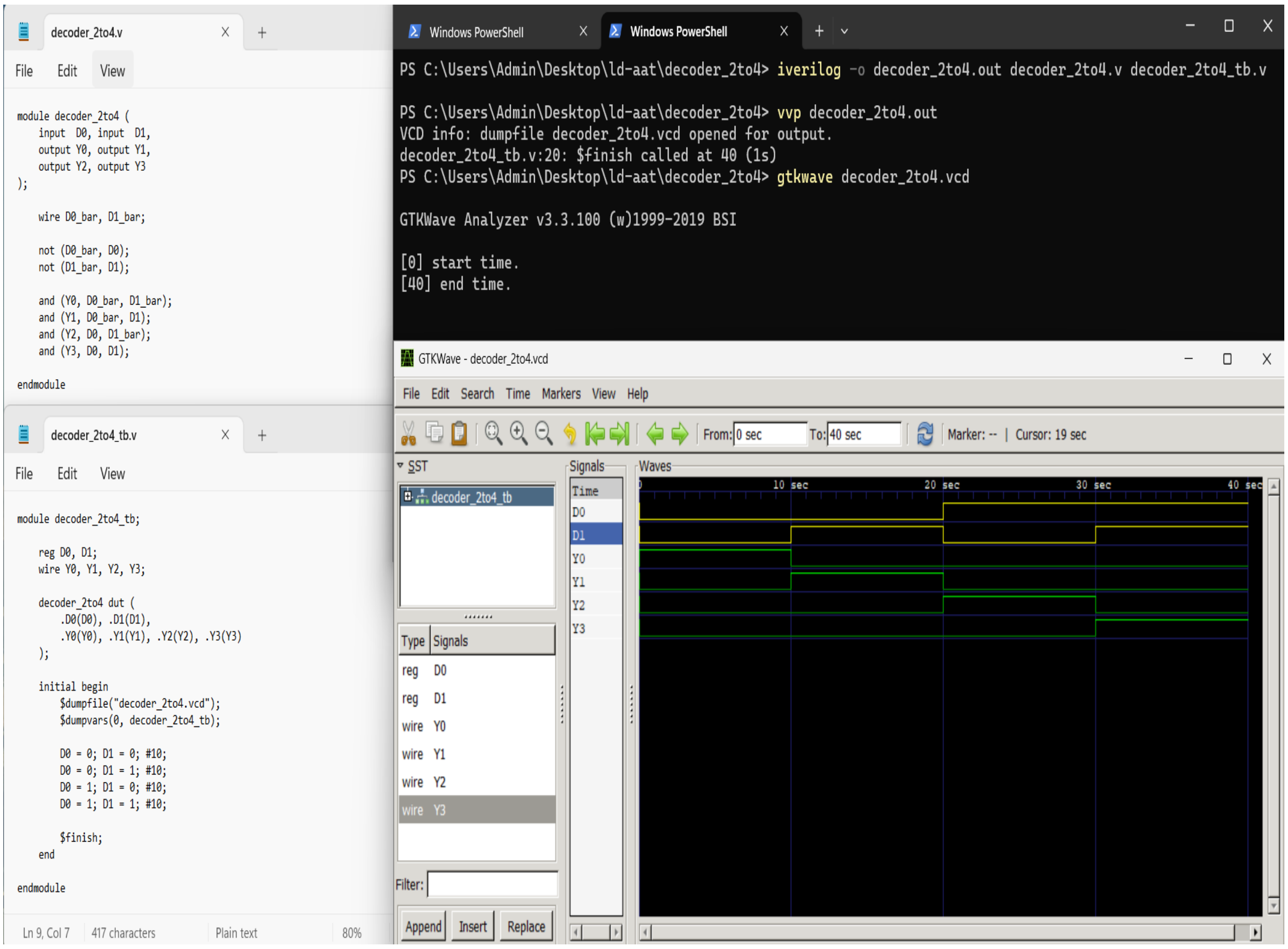
GTKWave - mux_4to1.vcd

From: 0 sec | To: 40 sec | Marker: -- | Cursor: 1 sec

Experiment 5: Write HDL implementation for a 2-to-4 decoder. Simulate the same using structural model and depict the timing diagram for valid inputs.

```
module decoder_2to4 (
   input  D0, input  D1,
   output Y0, output Y1,
   output Y2, output Y3
);

   wire D0_bar, D1_bar;

   not (D0_bar, D0);
   not (D1_bar, D1);

   and (Y0, D0_bar, D1_bar);
   and (Y1, D0_bar, D1);
   and (Y2, D0, D1_bar);
   and (Y3, D0, D1);
endmodule


// testbench
module decoder_2to4_tb;

   reg D0, D1;
   wire Y0, Y1, Y2, Y3;

   decoder_2to4 dut (
      .D0(D0), .D1(D1),
      .Y0(Y0), .Y1(Y1), .Y2(Y2), .Y3(Y3)
   );

   initial begin
      $dumpfile("decoder_2to4.vcd");
      $dumpvars(0, decoder_2to4_tb);

      D0 = 0; D1 = 0; #10;
      D0 = 0; D1 = 1; #10;
      D0 = 1; D1 = 0; #10;
      D0 = 1; D1 = 1; #10;

      $finish;
   end
endmodule
```

Compilation, Execution and Result of Simulation



```
module decoder_2to4 (
    input  D0, input  D1,
    output Y0, output Y1,
    output Y2, output Y3
);

    wire D0_bar, D1_bar;

    not (D0_bar, D0);
    not (D1_bar, D1);

    and (Y0, D0_bar, D1_bar);
    and (Y1, D0_bar, D1);
    and (Y2, D0, D1_bar);
    and (Y3, D0, D1);

endmodule
```

```
module decoder_2to4_tb;

    reg D0, D1;
    wire Y0, Y1, Y2, Y3;

    decoder_2to4 dut (
        .D0(D0), .D1(D1),
        .Y0(Y0), .Y1(Y1), .Y2(Y2), .Y3(Y3)
    );

    initial begin
        $dumpfile("decoder_2to4.vcd");
        $dumpvars(0, decoder_2to4_tb);

        D0 = 0; D1 = 0; #10;
        D0 = 0; D1 = 1; #10;
        D0 = 1; D1 = 0; #10;
        D0 = 1; D1 = 1; #10;

        $finish;
    end

endmodule
```

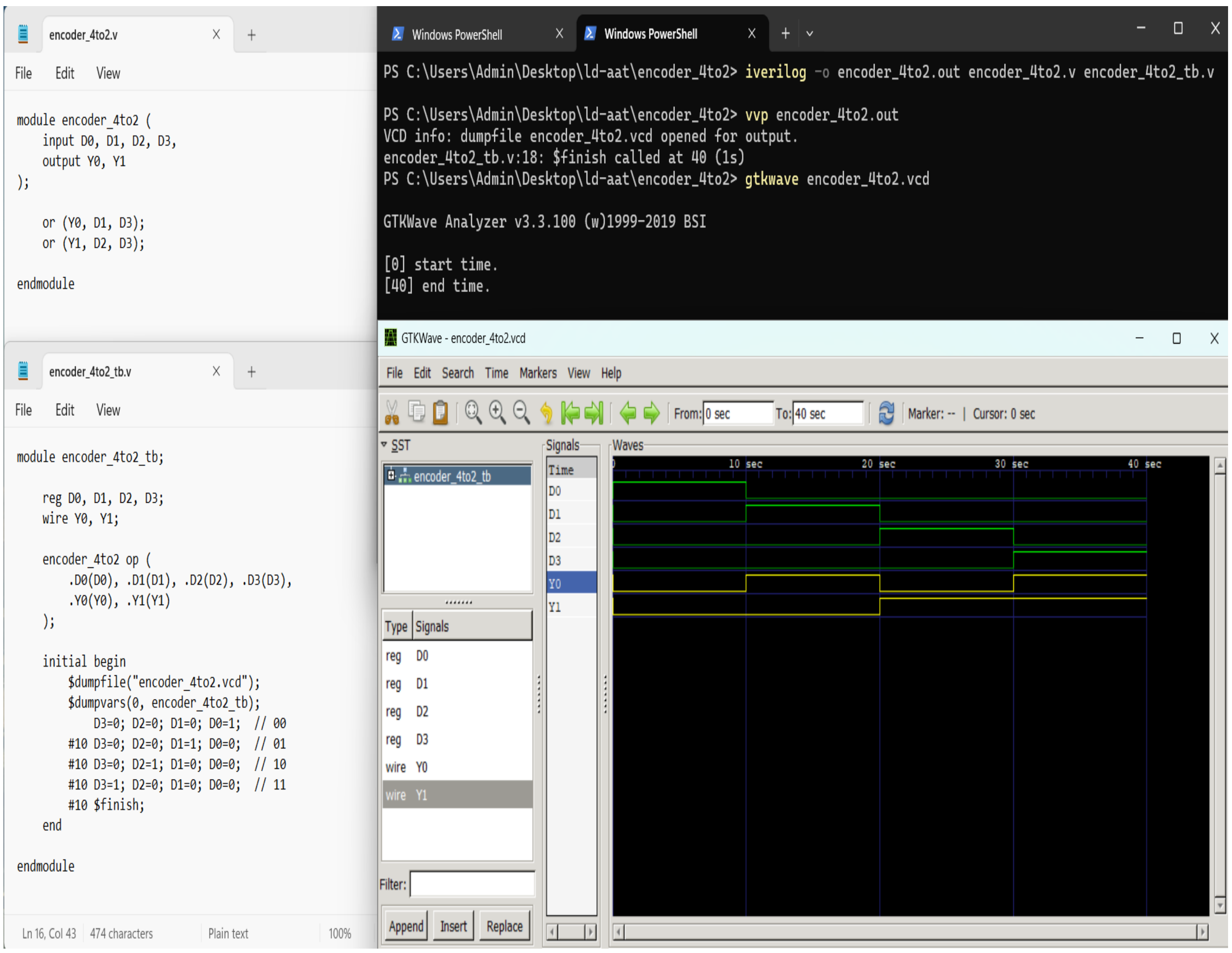
```
PS C:\Users\Admin\Desktop\ld-aat\decoder_2to4> iverilog -o decoder_2to4.out decoder_2to4.v decoder_2to4_tb.v

PS C:\Users\Admin\Desktop\ld-aat\decoder_2to4> vvp decoder_2to4.out
VCD info: dumpfile decoder_2to4.vcd opened for output.
decoder_2to4_tb.v:20: $finish called at 40 (1s)
PS C:\Users\Admin\Desktop\ld-aat\decoder_2to4> gtkwave decoder_2to4.vcd

GTKWave Analyzer v3.3.100 (w)1999-2019 BSI

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

Type	Signals
reg	D0
reg	D1
wire	Y0
wire	Y1
wire	Y2
wire	Y3

Experiment 6: Write HDL implementation for a 4-to-2 encoder. Simulate the same using structural model and depict the timing diagram for valid inputs.

```
module encoder_4to2 (
   input D0, D1, D2, D3,
   output Y0, Y1
);

   or (Y0, D1, D3);
   or (Y1, D2, D3);

endmodule

// testbench
module encoder_4to2_tb;

   reg D0, D1, D2, D3;
   wire Y0, Y1;

   encoder_4to2 op (
      .D0(D0), .D1(D1), .D2(D2), .D3(D3),
      .Y0(Y0), .Y1(Y1)
   );

   initial begin
      $dumpfile("encoder_4to2.vcd");
      $dumpvars(0, encoder_4to2_tb);
         D3=0; D2=0; D1=0; D0=1;  // 00
      #10 D3=0; D2=0; D1=1; D0=0;  // 01
      #10 D3=0; D2=1; D1=0; D0=0;  // 10
      #10 D3=1; D2=0; D1=0; D0=0;  // 11
      #10 $finish;
   end

endmodule
```

<ins>**Compilation, Execution and Result of Simulation**</ins>



```
module encoder_4to2 (
    input D0, D1, D2, D3,
    output Y0, Y1
);

    or (Y0, D1, D3);
    or (Y1, D2, D3);

endmodule
```

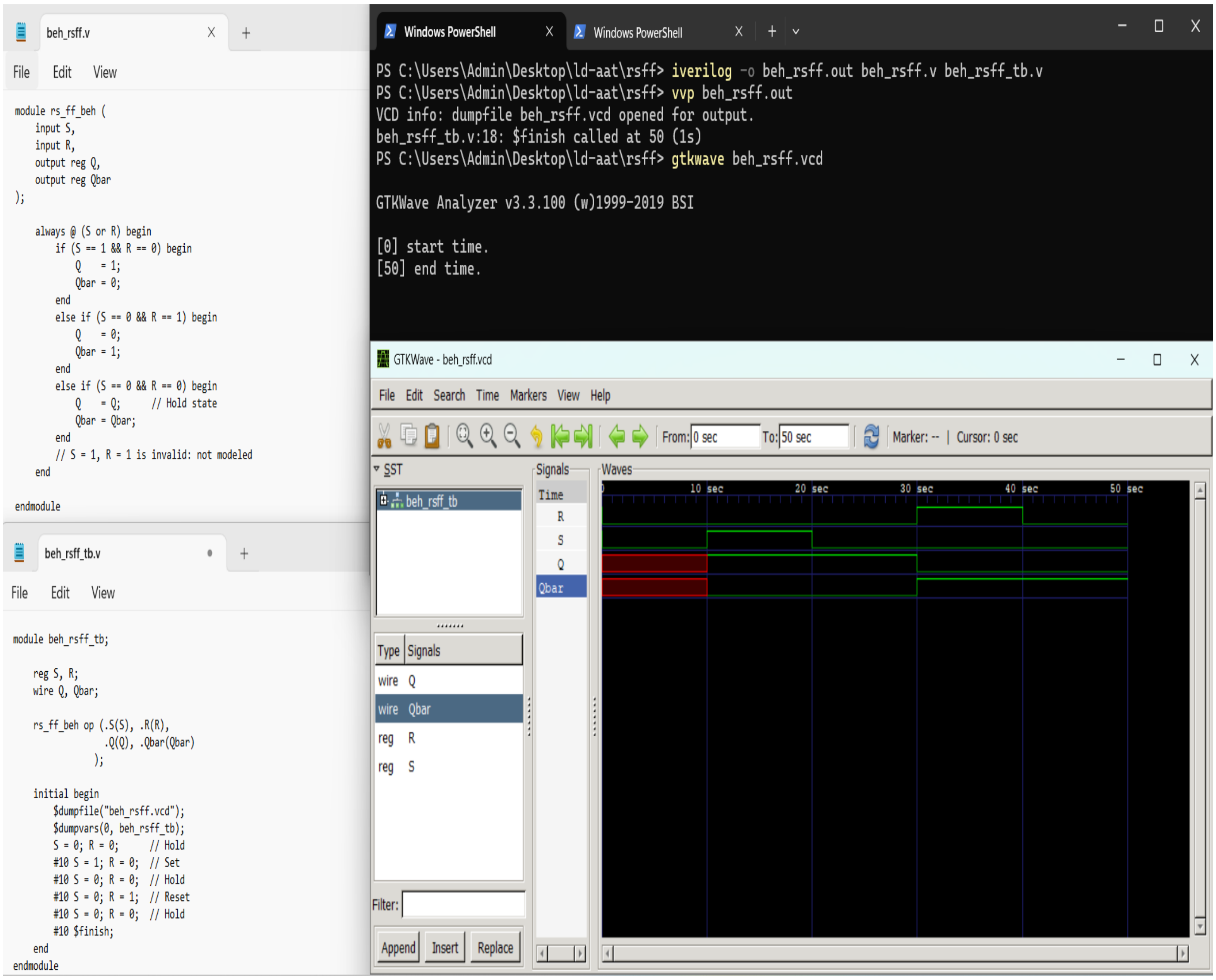
```
module encoder_4to2_tb;

    reg D0, D1, D2, D3;
    wire Y0, Y1;

    encoder_4to2 op (
        .D0(D0), .D1(D1), .D2(D2), .D3(D3),
        .Y0(Y0), .Y1(Y1)
    );

    initial begin
        $dumpfile("encoder_4to2.vcd");
        $dumpvars(0, encoder_4to2_tb);
            D3=0; D2=0; D1=0; D0=1;  // 00
        #10 D3=0; D2=0; D1=1; D0=0;  // 01
        #10 D3=0; D2=1; D1=0; D0=0;  // 10
        #10 D3=1; D2=0; D1=0; D0=0;  // 11
        #10 $finish;
    end

endmodule
```

```
PS C:\Users\Admin\Desktop\ld-aat\encoder_4to2> iverilog -o encoder_4to2.out encoder_4to2.v encoder_4to2_tb.v

PS C:\Users\Admin\Desktop\ld-aat\encoder_4to2> vvp encoder_4to2.out
VCD info: dumpfile encoder_4to2.vcd opened for output.
encoder_4to2_tb.v:18: $finish called at 40 (1s)
PS C:\Users\Admin\Desktop\ld-aat\encoder_4to2> gtkwave encoder_4to2.vcd

GTKWave Analyzer v3.3.100 (w)1999-2019 BSI

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

Experiment 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.

```
module rs_ff_beh (
   input S,
   input R,
   output reg Q,
   output reg Qbar
);
   always @ (S or R) begin
      if (S == 1 && R == 0) begin
         Q    = 1;
         Qbar = 0;
      end
      else if (S == 0 && R == 1) begin
         Q    = 0;
         Qbar = 1;
      end
      else if (S == 0 && R == 0) begin
         Q    = Q;      // Hold state
         Qbar = Qbar;
      end
      // S = 1, R = 1 is invalid: not modeled
   end
endmodule


// testbench
module beh_rsff_tb;
   reg S, R;
   wire Q, Qbar;
   rs_ff_beh op (.S(S), .R(R), .Q(Q), .Qbar(Qbar));
   initial begin
      $dumpfile("beh_rsff.vcd");
      $dumpvars(0, beh_rsff_tb);
      S = 0; R = 0;      // Hold
      #10 S = 1; R = 0;  // Set
      #10 S = 0; R = 0;  // Hold
      #10 S = 0; R = 1;  // Reset
      #10 S = 0; R = 0;  // Hold
      #10 $finish;
   end
endmodule
```

Compilation, Execution and Result of Simulation



```
module rs_ff_beh (
    input S,
    input R,
    output reg Q,
    output reg Qbar
);

    always @ (S or R) begin
        if (S == 1 && R == 0) begin
            Q    = 1;
            Qbar = 0;
        end
        else if (S == 0 && R == 1) begin
            Q    = 0;
            Qbar = 1;
        end
        else if (S == 0 && R == 0) begin
            Q    = Q;      // Hold state
            Qbar = Qbar;
        end
        // S = 1, R = 1 is invalid: not modeled
    end

endmodule
```

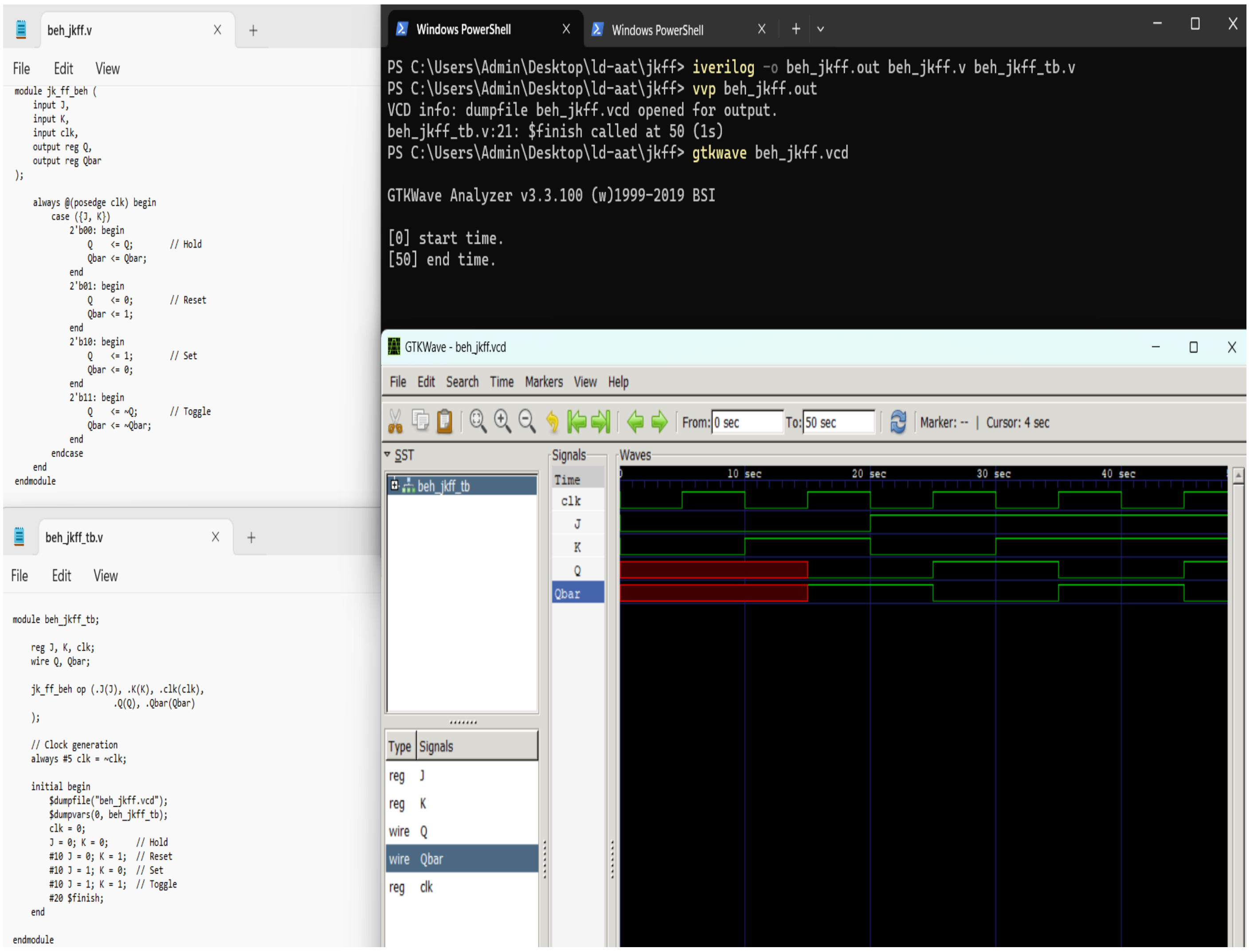
```
module beh_rsff_tb;

    reg S, R;
    wire Q, Qbar;

    rs_ff_beh op (.S(S), .R(R),
                  .Q(Q), .Qbar(Qbar)
                );

    initial begin
        $dumpfile("beh_rsff.vcd");
        $dumpvars(0, beh_rsff_tb);
        S = 0; R = 0;      // Hold
        #10 S = 1; R = 0;  // Set
        #10 S = 0; R = 0;  // Hold
        #10 S = 0; R = 1;  // Reset
        #10 S = 0; R = 0;  // Hold
        #10 $finish;
    end
endmodule
```

```
PS C:\Users\Admin\Desktop\ld-aat\rsff> iverilog -o beh_rsff.out beh_rsff.v beh_rsff_tb.v
PS C:\Users\Admin\Desktop\ld-aat\rsff> vvp beh_rsff.out
VCD info: dumpfile beh_rsff.vcd opened for output.
beh_rsff_tb.v:18: $finish called at 50 (1s)
PS C:\Users\Admin\Desktop\ld-aat\rsff> gtkwave beh_rsff.vcd

GTKWave Analyzer v3.3.100 (w)1999-2019 BSI

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

Experiment 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.

```
module jk_ff_beh ( input J, input K, input clk, output reg Q, output reg Qbar );
   initial begin
        Q = 0;
        Qbar = 1;
    end

    always @(posedge clk) begin
          Q = 1;
          Qbar = 0;
        case ({J, K})
          2'b00: begin
             Q    <= Q;        // Hold
             Qbar <= Qbar;
          end
          2'b01: begin
             Q    <= 0;        // Reset
             Qbar <= 1;
          end
          2'b10: begin
             Q    <= 1;        // Set
             Qbar <= 0;
          end
          2'b11: begin
             Q    <= ~Q;       // Toggle
             Qbar <= ~Qbar;
          end
        endcase
    end
endmodule

// testbench
module beh_jkff_tb;

    reg J, K, clk;
    wire Q, Qbar;

    jk_ff_beh op (.J(J), .K(K), .clk(clk), .Q(Q), .Qbar(Qbar));

    // Clock generation
    always #5 clk = ~clk;

    initial begin
        $dumpfile("beh_jkff.vcd");
        $dumpvars(0, beh_jkff_tb);
        clk = 0;
        J = 0; K = 0;       // Hold
        #10 J = 0; K = 1;  // Reset
        #10 J = 1; K = 0;  // Set
        #10 J = 1; K = 1;  // Toggle
        #20 $finish;
    end
endmodule
```

Compilation, Execution and Result of Simulation



beh_jkff.v

```
module jk_ff_beh (
    input J,
    input K,
    input clk,
    output reg Q,
    output reg Qbar
);

    always @(posedge clk) begin
        case ({J, K})
            2'b00: begin
                Q    <= Q;        // Hold
                Qbar <= Qbar;
            end
            2'b01: begin
                Q    <= 0;        // Reset
                Qbar <= 1;
            end
            2'b10: begin
                Q    <= 1;        // Set
                Qbar <= 0;
            end
            2'b11: begin
                Q    <= ~Q;       // Toggle
                Qbar <= ~Qbar;
            end
        endcase
    end
endmodule
```

beh_jkff_tb.v

```
module beh_jkff_tb;

    reg J, K, clk;
    wire Q, Qbar;

    jk_ff_beh op (.J(J), .K(K), .clk(clk),
                  .Q(Q), .Qbar(Qbar)
    );

    // Clock generation
    always #5 clk = ~clk;

    initial begin
        $dumpfile("beh_jkff.vcd");
        $dumpvars(0, beh_jkff_tb);
        clk = 0;
        J = 0; K = 0;      // Hold
        #10 J = 0; K = 1;  // Reset
        #10 J = 1; K = 0;  // Set
        #10 J = 1; K = 1;  // Toggle
        #20 $finish;
    end

endmodule
```

Windows PowerShell

```
PS C:\Users\Admin\Desktop\ld-aat\jkff> iverilog -o beh_jkff.out beh_jkff.v beh_jkff_tb.v
PS C:\Users\Admin\Desktop\ld-aat\jkff> vvp beh_jkff.out
VCD info: dumpfile beh_jkff.vcd opened for output.
beh_jkff_tb.v:21: $finish called at 50 (1s)
PS C:\Users\Admin\Desktop\ld-aat\jkff> gtkwave beh_jkff.vcd

GTKWave Analyzer v3.3.100 (w)1999-2019 BSI

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

GTKWave - beh_jkff.vcd

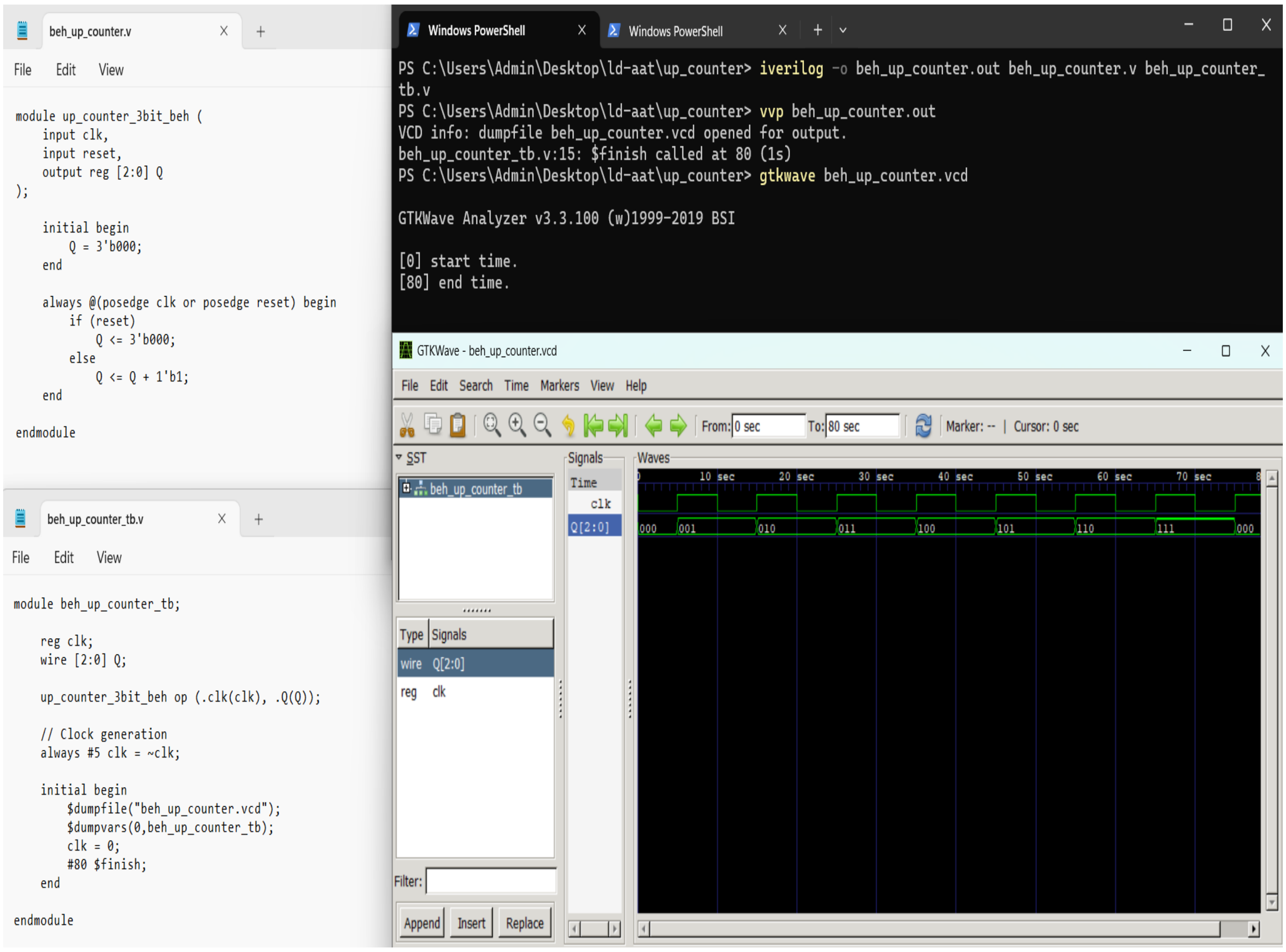
From: 0 sec To: 50 sec Marker: -- Cursor: 4 sec

Type	Signals
reg	J
reg	K
wire	Q
wire	Qbar
reg	clk

Experiment 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.

```
module up_counter_3bit_beh (
   input clk,
   input reset,
   output reg [2:0] Q
);

   initial begin
      Q = 3'b000;
   end

   always @(posedge clk or posedge reset) begin
      if (reset)
         Q <= 3'b000;
      else
         Q <= Q + 1'b1;
   end

endmodule

// testbench
module beh_up_counter_tb;

   reg clk;
   wire [2:0] Q;

   up_counter_3bit_beh op (.clk(clk), .Q(Q));

   // Clock generation
   always #5 clk = ~clk;

   initial begin
      $dumpfile("beh_up_counter.vcd");
      $dumpvars(0,beh_up_counter_tb);
      clk = 0;
      #80 $finish;
   end

endmodule
```

Compilation, Execution and Result of Simulation



beh_up_counter.v

```
module up_counter_3bit_beh (
    input clk,
    input reset,
    output reg [2:0] Q
);

    initial begin
        Q = 3'b000;
    end

    always @(posedge clk or posedge reset) begin
        if (reset)
            Q <= 3'b000;
        else
            Q <= Q + 1'b1;
    end

endmodule
```

beh_up_counter_tb.v

```
module beh_up_counter_tb;

    reg clk;
    wire [2:0] Q;

    up_counter_3bit_beh op (.clk(clk), .Q(Q));

    // Clock generation
    always #5 clk = ~clk;

    initial begin
        $dumpfile("beh_up_counter.vcd");
        $dumpvars(0,beh_up_counter_tb);
        clk = 0;
        #80 $finish;
    end

endmodule
```

Windows PowerShell

```
PS C:\Users\Admin\Desktop\ld-aat\up_counter> iverilog -o beh_up_counter.out beh_up_counter.v beh_up_counter_
tb.v
PS C:\Users\Admin\Desktop\ld-aat\up_counter> vvp beh_up_counter.out
VCD info: dumpfile beh_up_counter.vcd opened for output.
beh_up_counter_tb.v:15: $finish called at 80 (1s)
PS C:\Users\Admin\Desktop\ld-aat\up_counter> gtkwave beh_up_counter.vcd

GTKWave Analyzer v3.3.100 (w)1999-2019 BSI

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

GTKWave - beh_up_counter.vcd

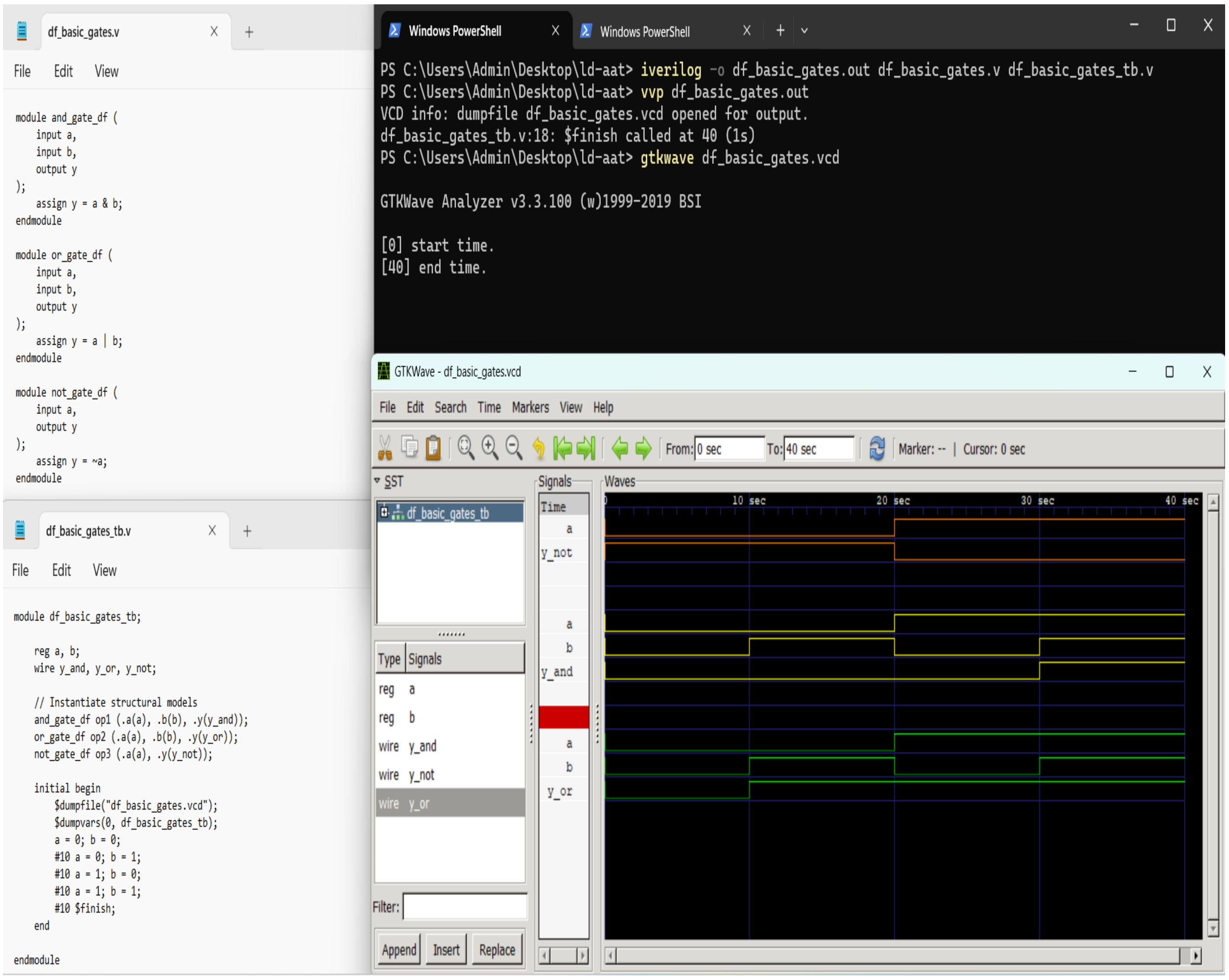
From: 0 sec To: 80 sec Marker: -- | Cursor: 0 sec

Experiment 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.

```
module and_gate_df (
   input a,
   input b,
   output y
);
   assign y = a & b;
endmodule

module or_gate_df (
   input a,
   input b,
   output y
);
   assign y = a | b;
endmodule

module not_gate_df (
   input a,
   output y
);
   assign y = ~a;
endmodule


// testbench
module df_basic_gates_tb;

   reg a, b;
   wire y_and, y_or, y_not;

   // Instantiate structural models
   and_gate_df op1 (.a(a), .b(b), .y(y_and));
   or_gate_df op2 (.a(a), .b(b), .y(y_or));
   not_gate_df op3 (.a(a), .y(y_not));

   initial begin
      $dumpfile("df_basic_gates.vcd");
      $dumpvars(0, df_basic_gates_tb);
      a = 0; b = 0;
```

```
        #10 a = 0; b = 1;
        #10 a = 1; b = 0;
        #10 a = 1; b = 1;
        #10 $finish;
    end
endmodule
```

Compilation, Execution and Result of Simulation



df_basic_gates.v

```
module and_gate_df (
    input a,
    input b,
    output y
);
    assign y = a & b;
endmodule

module or_gate_df (
    input a,
    input b,
    output y
);
    assign y = a | b;
endmodule

module not_gate_df (
    input a,
    output y
);
    assign y = ~a;
endmodule
```

df_basic_gates_tb.v

```
module df_basic_gates_tb;

    reg a, b;
    wire y_and, y_or, y_not;

    // Instantiate structural models
    and_gate_df op1 (.a(a), .b(b), .y(y_and));
    or_gate_df op2 (.a(a), .b(b), .y(y_or));
    not_gate_df op3 (.a(a), .y(y_not));

    initial begin
        $dumpfile("df_basic_gates.vcd");
        $dumpvars(0, df_basic_gates_tb);
        a = 0; b = 0;
        #10 a = 0; b = 1;
        #10 a = 1; b = 0;
        #10 a = 1; b = 1;
        #10 $finish;
    end

endmodule
```

Windows PowerShell

```
PS C:\Users\Admin\Desktop\ld-aat> iverilog -o df_basic_gates.out df_basic_gates.v df_basic_gates_tb.v
PS C:\Users\Admin\Desktop\ld-aat> vvp df_basic_gates.out
VCD info: dumpfile df_basic_gates.vcd opened for output.
df_basic_gates_tb.v:18: $finish called at 40 (1s)
PS C:\Users\Admin\Desktop\ld-aat> gtkwave df_basic_gates.vcd

GTKWave Analyzer v3.3.100 (w)1999-2019 BSI

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

GTKWave - df_basic_gates.vcd

From: 0 sec To: 40 sec Marker: -- | Cursor: 0 sec