ECEN 651 Lab Exercise 2 Report

Behavioral, Dataflow, and Structural Verilog

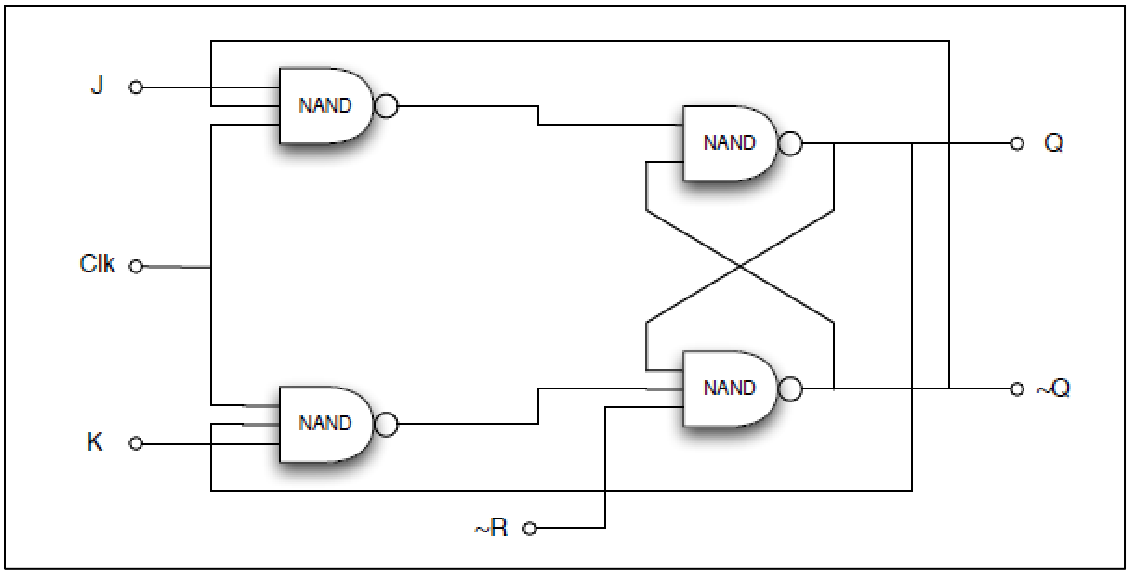
Name: Pranav Anantharam

UIN: 734003886

Date: 09/22/2023

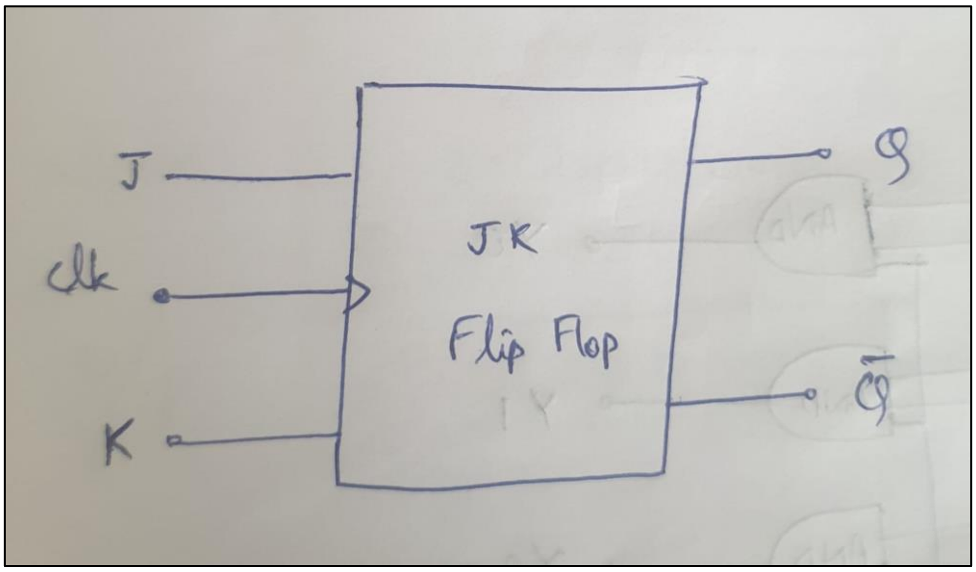
1. Design and simulate a JK Flip-flop using structural Verilog.

Schematic:

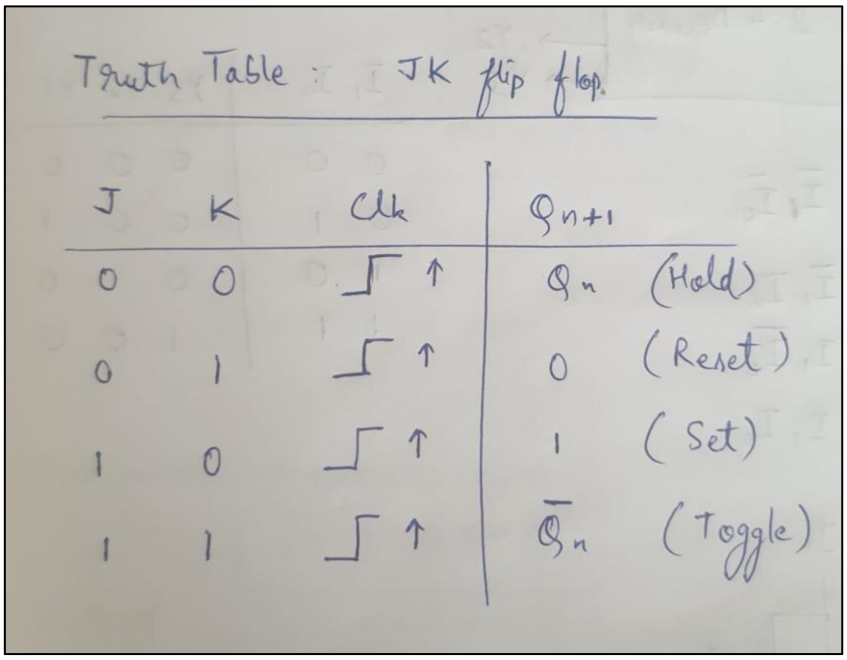


JK Flip-Flop constructed with NAND gates

Block Diagram:



Truth Table:



Truth Table : JK flip flop.

J	K	Clk	Q_{n+1}	
0	0	↑	Q_n	(Hold)
0	1	↑	0	(Reset)
1	0	↑	1	(Set)
1	1	↑	$\bar{Q}_n$	(Toggle)

Design Source code filename: JK.v

```
`timescale 1ns / 1ps
module JK(j, k, clk, reset, out);
        // Inputs
        input j, k, clk, reset;
        // Output
        output out;
        wire w1, w2, w3, qbar;
        // If reset is high, reset flip flop value to zero
        not not1(w3, reset);
        nand #2 nand_1(w1, j, clk, qbar);
        nand #2 nand_2(w2, k, clk, out);
        nand nand_3(out, qbar, w1);
        nand nand_4(qbar, out, w2, w3);
endmodule
```

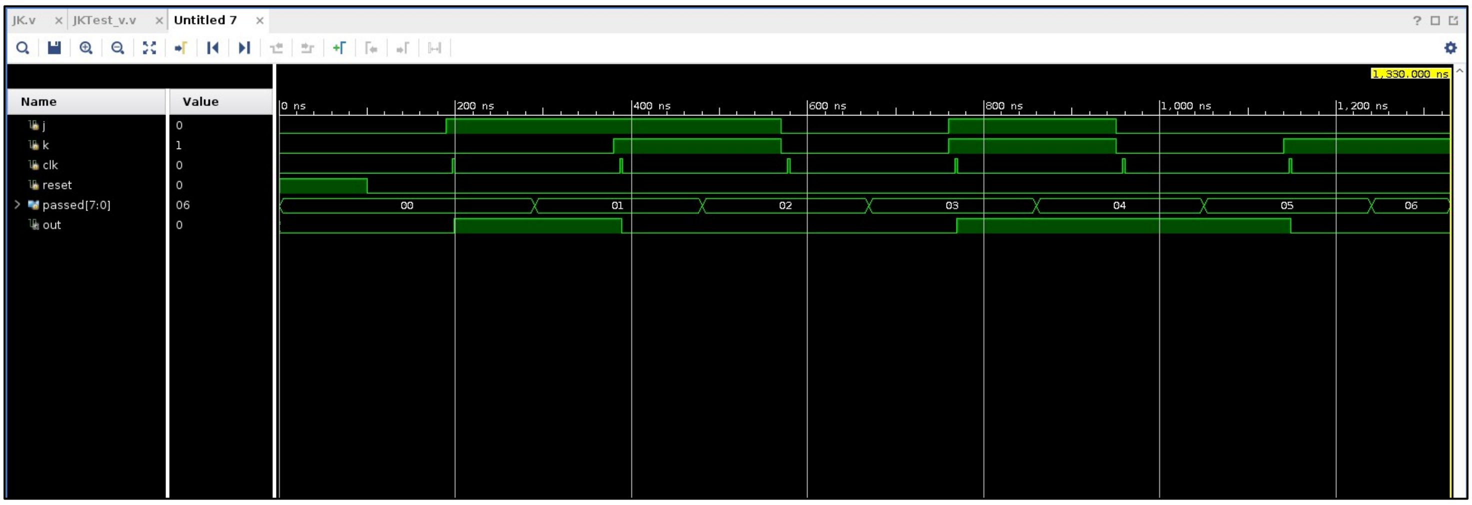
Testbench Source code filename: JKTest_v.v

```
`timescale 1ns / 1ps

`define STRLEN 15

module JKTest_v ;

    task passTest;

        input actualOut, expectedOut;

        input [`STRLEN*8:0] testType;

        inout [7:0] passed;


        if(actualOut == expectedOut)

        begin

            $display("%t : %s passed", $time, testType);

            passed = passed + 1;

        end

        else begin

            $display("%t : %s failed: %d should be %d", $time, testType, actualOut,
expectedOut);

        end

    endtask


    task allPassed;

        input [7:0] passed;

        input [7:0] numTests;

        if(passed == numTests)

            $display ("All tests passed");

        else

            $display ("some tests failed");

    endtask


    // Inputs

    reg j;

    reg k;

    reg clk;

    reg reset;

    reg [7:0] passed;


    //Outputs

    wire out;
```

```
//Instantiate the Unit Under Test (UUT)
JK uut (
    .out(out) ,
    .j(j) ,
    .k(k) ,
    .clk(clk) ,
    .reset(reset)
);

initial begin
    //Initialize Inputs
    j = 0;
    k = 0;
    clk = 0;
    reset = 1;
    passed = 0;

    //Wait 100ns for global reset to finish
    #100;

    //Add stimulus here
    reset = 0;


    #90; j =1; k =0; # 7 ; clk = 1 ;
    # 3 ; clk = 0 ; #90;
    passTest(out, 1, "Set", passed);
    #90; j =1; k =1; # 7 ; clk = 1 ;
    #3 ; clk = 0 ; #90;
    passTest (out, 0, "Toggle 1", passed) ;
    #90; j =0; k =0; # 7 ; clk = 1 ;
    #3 ; clk = 0 ; #90;
    passTest (out, 0, "Hold 1", passed) ;
    #90; j =1; k =1; # 7 ; clk = 1 ;
    #3 ; clk = 0 ; #90;
    passTest (out, 1, "Toggle 2", passed) ;
    #90; j =0; k =0; # 7 ; clk = 1 ;
    #3 ; clk = 0 ; #90;
    passTest (out, 1, "Hold 2", passed) ;
    #90; j =0; k =1; # 7 ; clk = 1 ;
```

```
        #3 ; clk = 0 ; #90;
        passTest (out, 0, "Reset", passed) ;
        #90; allPassed(passed , 6) ;
    end
endmodule
```

Simulation Output Waveform: (JK Flip Flop Structural)



Output Console Logs: (JK Flip Flop Structural)

```
Vivado Simulator 2017.4
Time resolution is 1 ps
source JKTest_v.tcl
# set curr_wave [current_wave_config]
# if { [string length $curr_wave] == 0 } {
#   if { [llength [get_objects]] > 0} {
#     add_wave /
#     set_property needs_save false [current_wave_config]
#   } else {
#      send_msg_id Add_Wave-1 WARNING "No top level signals found. Simulator will start without
a wave window. If you want to open a wave window go to 'File->New Waveform Configuration' or
type 'create_wave_config' in the TCL console."
#   }
# }
restart
INFO: [Simtcl 6-17] Simulation restarted
run all
              290000 :              Set passed
```

```
            480000 :         Toggle 1 passed
            670000 :           Hold 1 passed
            860000 :         Toggle 2 passed
           1050000 :           Hold 2 passed
           1240000 :            Reset passed
All tests passed
```

---

2. Design and simulate a JK and D Flip-flop using behavioral Verilog.

(a) In behavioral Verilog, create a JK Flip-flop module. Be sure to use non-blocking assignment statements.

Design Source code filename: JK.v

```
`timescale 1ns / 1ps
module JK(j, k, clk, reset, out);
      // Inputs
      input j, k, clk, reset;
      // Output
      output reg out;
      // Trigger flip flop only on positive edge of clock or when reset is high
      always @ (posedge clk or posedge reset) begin
      if(reset)
            out <= 0;
      else begin
            case({j,k})
                  2'b00 : out <= out;      // Hold
                  2'b01 : out <= 0;        // Reset
                  2'b10 : out <= 1;        // Set
                  2'b11 : out <= ~out;     // Toggle
            endcase
      end
end
endmodule
```

(b) Test your module against the test bench provided. The operation should be identical to that of your structural version.

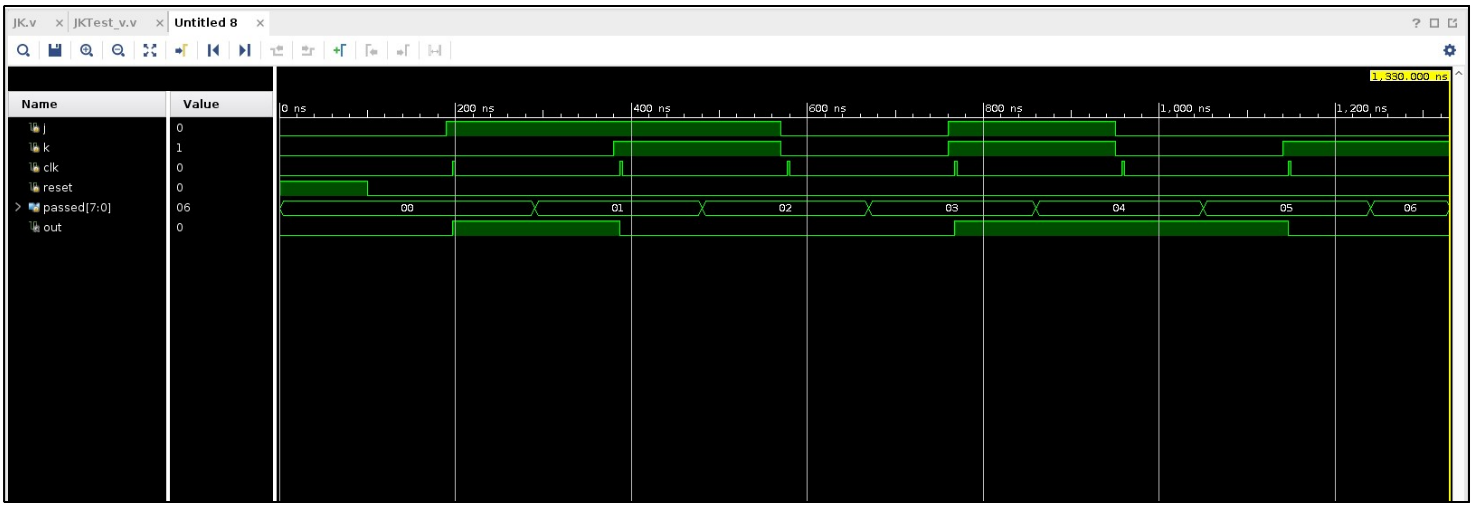
Testbench code filename: JKTest_v.v

```
`timescale 1ns / 1ps
`define STRLEN 15
module JKTest_v ;
    task passTest;
        input actualOut, expectedOut;
        input [`STRLEN*8:0] testType;
        inout [7:0] passed;

        if(actualOut == expectedOut)
        begin
            $display("%t : %s passed", $time, testType);
            passed = passed + 1;
        end
        else begin
            $display("%t : %s failed: %d should be %d", $time, testType, actualOut,
expectedOut);
        end
    endtask

    task allPassed;
        input [7:0] passed;
        input [7:0] numTests;

        if(passed == numTests)
            $display ("All tests passed");
        else
            $display ("some tests failed");
    endtask

    // Inputs
    reg j;
    reg k;
    reg clk;
    reg reset;

    reg [7:0] passed;
```

```
//Outputs
wire out;

//Instantiate the Unit Under Test (UUT)
JK uut (
    .out(out) ,
    .j(j) ,
    .k(k) ,
    .clk(clk) ,
    .reset(reset)
);

initial begin
    //Initialize Inputs
    j = 0;
    k = 0;
    clk = 0;
    reset = 1;
    passed = 0;

    //Wait 100ns for global reset to finish
    #100;

    //Add stimulus here
    reset = 0;
    #90; j =1; k =0; # 7 ; clk = 1 ;
    # 3 ; clk = 0 ; #90;
    passTest(out, 1, "Set", passed);
    #90; j =1; k =1; # 7 ; clk = 1 ;
    #3 ; clk = 0 ; #90;
    passTest (out, 0, "Toggle 1", passed) ;
    #90; j =0; k =0; # 7 ; clk = 1 ;
    #3 ; clk = 0 ; #90;
    passTest (out, 0, "Hold 1", passed) ;
    #90; j =1; k =1; # 7 ; clk = 1 ;
    #3 ; clk = 0 ; #90;
    passTest (out, 1, "Toggle 2", passed) ;
    #90; j =0; k =0; # 7 ; clk = 1 ;
```

```
        #3 ; clk = 0 ; #90;
        passTest (out, 1, "Hold 2", passed) ;
        #90; j =0; k =1; # 7 ; clk = 1 ;
        #3 ; clk = 0 ; #90;
        passTest (out, 0, "Reset", passed) ;
        #90; allPassed(passed , 6) ;
    end
endmodule
```

Simulation Output Waveform: (JK Flip Flop Behavioral)

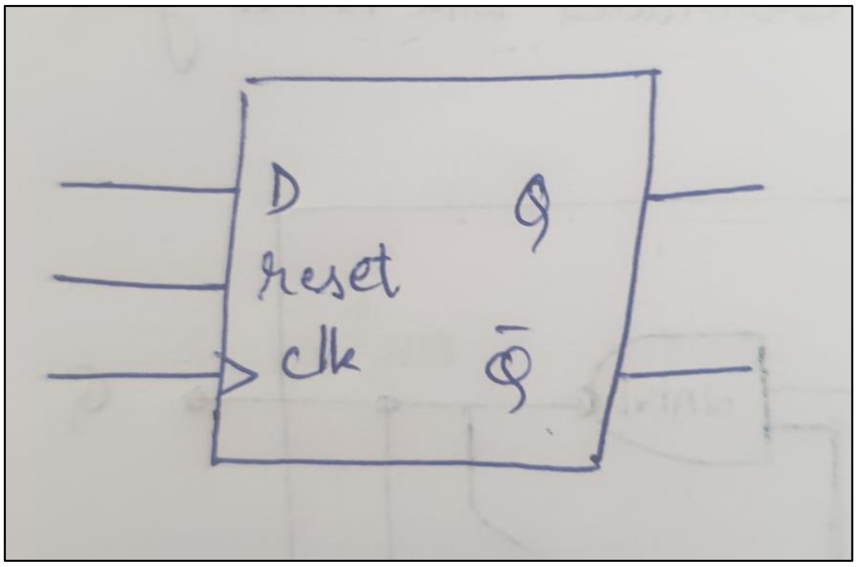


Output Console Logs: (JK Flip Flop Behavioral)

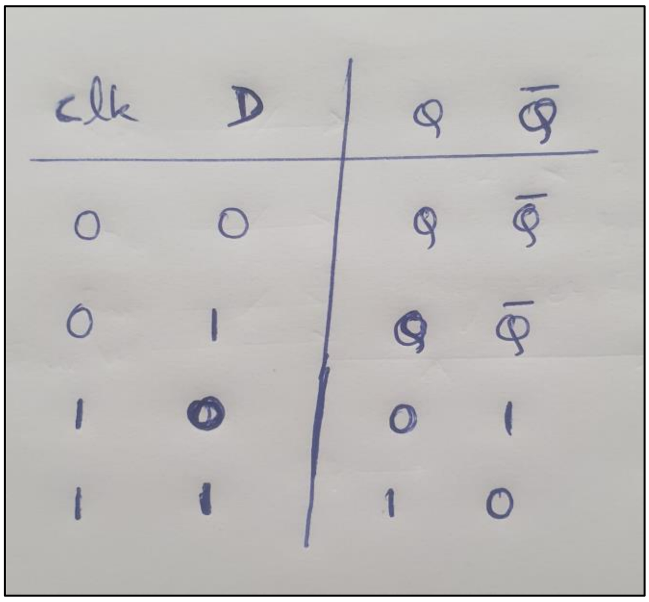
```
INFO: [USF-XSim-8] Loading simulator feature
Vivado Simulator 2017.4
Time resolution is 1 ps
source JKTest_v.tcl
# set curr_wave [current_wave_config]
# if { [string length $curr_wave] == 0 } {
#   if { [llength [get_objects]] > 0} {
#     add_wave /
#     set_property needs_save false [current_wave_config]
#   } else {
#      send_msg_id Add_Wave-1 WARNING "No top level signals found. Simulator will start without
a wave window. If you want to open a wave window go to 'File->New Waveform Configuration' or
type 'create_wave_config' in the TCL console."
#   }
# }
# run 1000ns
              290000 :              Set passed
              480000 :         Toggle 1 passed
              670000 :           Hold 1 passed
              860000 :         Toggle 2 passed
INFO: [USF-XSim-96] XSim completed. Design snapshot 'JKTest_v_behav' loaded.
INFO: [USF-XSim-97] XSim simulation ran for 1000ns
restart
INFO: [Simtcl 6-17] Simulation restarted
run all
              290000 :             Set passed
              480000 :         Toggle 1 passed
              670000 :           Hold 1 passed
              860000 :         Toggle 2 passed
             1050000 :           Hold 2 passed
             1240000 :            Reset passed
All tests passed
```

(c) Now create a D Flip-flop using behavioral Verilog and provide the truth table for a D flip-flop.

Block Diagram:



Truth Table:



clk	D	Q	$\bar{Q}$
0	0	Q	$\bar{Q}$
0	1	Q	$\bar{Q}$
1	0	0	1
1	1	1	0

Design Source code filename: D.v

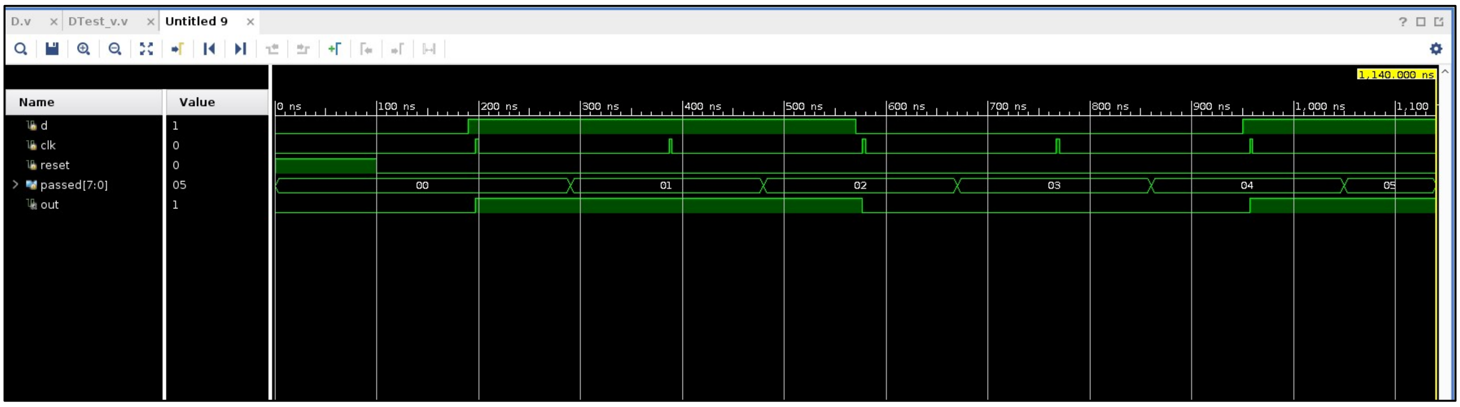
```
`timescale 1ns / 1ps

module D(d, clk, reset, out);

	// Inputs

	input d, clk, reset;

	// Output

	output reg out;

	// Trigger flip flop on positive edge of clock or when reset is high

	always @ (posedge clk or posedge reset) begin

		if(reset)

			out <= 0;	// Reset output to zero

		else

			out <= d;	// Output is set to d input

	end

endmodule
```

(d) Using the above testbench as a starting point, create a testbench to test the operation of your D Flip-flop.

Testbench code filename: DTest_v.v

```
`timescale 1ns / 1ps

`define STRLEN 15

module DTest_v ;

    task passTest;

        input actualOut, expectedOut;

        input [`STRLEN*8:0] testType;

        inout [7:0] passed;


        if(actualOut == expectedOut)

        begin

            $display("%t : %s passed", $time, testType);

            passed = passed + 1;

        end

        else begin

            $display("%t : %s failed: %d should be %d", $time, testType, actualOut,
expectedOut);

        end

    endtask

    task allPassed;

        input [7:0] passed;

        input [7:0] numTests;

        if(passed == numTests)

            $display ("All tests passed");

        else

            $display ("some tests failed");

    endtask


    // Inputs

    reg d;

    reg clk;

    reg reset;

    reg [7:0] passed;


    //Outputs

    wire out;
    //Instantiate the Unit Under Test (UUT)
    D uut (
        .out(out) ,
        .d(d) ,
        .clk(clk) ,
        .reset(reset)
    );

    initial begin
        //Initialize Inputs
        d = 0;
        clk = 0;
        reset = 1;
        passed = 0;
        //Wait 100ns for global reset to finish
        #100;

        //Add stimulus here
        reset = 0;
        #90; d = 1; #7; clk = 1;
        #3; clk = 0; #90;
        passTest(out, 1, "Set 1", passed);
        #90; d = 1; #7; clk = 1;
        #3; clk = 0; #90;
        passTest(out, 1, "Hold 1", passed);
        #90; d = 0; #7; clk = 1;
        #3; clk = 0; #90;
        passTest(out, 0, "Reset", passed);
        #90; d = 0; #7; clk = 1;
        #3; clk = 0; #90;
        passTest(out, 0, "Hold 2", passed);
        #90; d = 1; #7; clk = 1;
        #3; clk = 0; #90;
        passTest(out, 1, "Set 2", passed);
        #90; allPassed(passed , 5) ;
    end
endmodule
```

Simulation Output Waveform: (D Flip Flop)



Output Console Logs: (D Flip Flop)

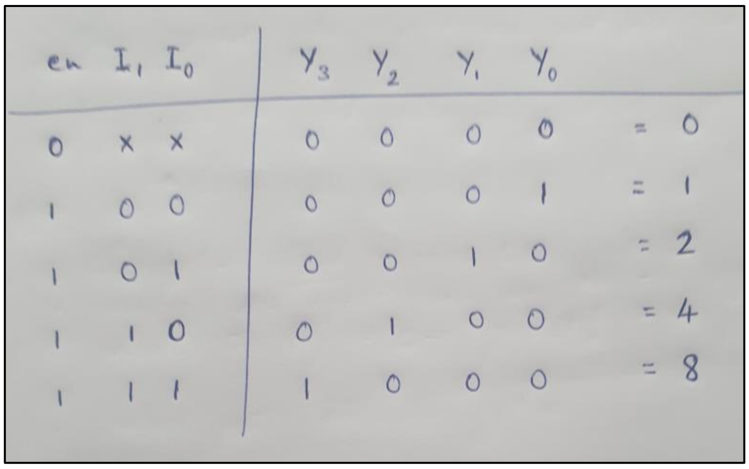
```
INFO: [USF-XSim-8] Loading simulator feature

Vivado Simulator 2017.4

Time resolution is 1 ps

source DTest_v.tcl

# set curr_wave [current_wave_config]

# if { [string length $curr_wave] == 0 } {

#   if { [llength [get_objects]] > 0} {

#     add_wave /

#     set_property needs_save false [current_wave_config]

#   } else {

#      send_msg_id Add_Wave-1 WARNING "No top level signals found. Simulator will start without
a wave window. If you want to open a wave window go to 'File->New Waveform Configuration' or
type 'create_wave_config' in the TCL console."

#   }

# }

# run 1000ns

              290000 :             Set 1 passed

              480000 :            Hold 1 passed

              670000 :             Reset passed

              860000 :            Hold 2 passed

INFO: [USF-XSim-96] XSim completed. Design snapshot 'DTest_v_behav' loaded.

INFO: [USF-XSim-97] XSim simulation ran for 1000ns

restart

INFO: [Simtcl 6-17] Simulation restarted

run all
```

```
            290000 :             Set 1 passed
            480000 :            Hold 1 passed
            670000 :             Reset passed
            860000 :            Hold 2 passed
           1050000 :             Set 2 passed
All tests passed
```

---

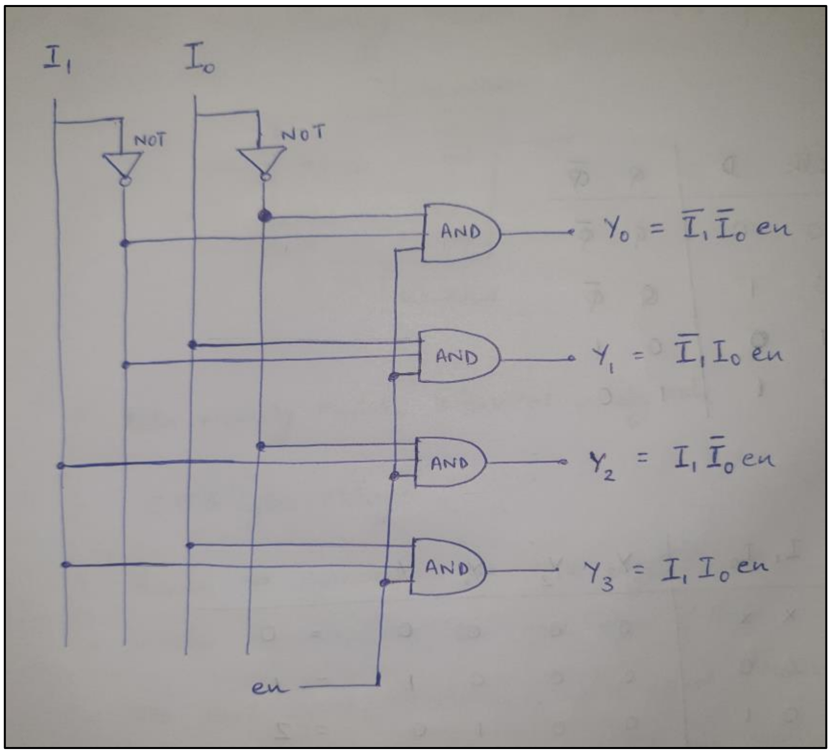
3. Design and simulate a 2-to-4 decoder using dataflow Verilog.

(a) Construct the truth table for a 2:4 decoder with an enable signal.



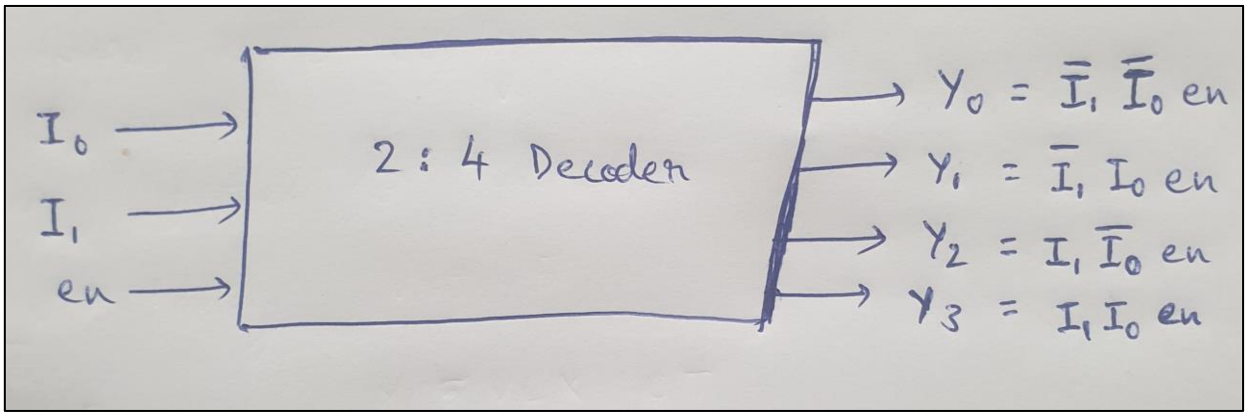
en	I_1	I_0	Y_3	Y_2	Y_1	Y_0	
0	x	x	0	0	0	0	= 0
1	0	0	0	0	0	1	= 1
1	0	1	0	0	1	0	= 2
1	1	0	0	1	0	0	= 4
1	1	1	1	0	0	0	= 8

(b) Draw the gate level schematic for a 2:4 decoder.



(c) Describe the decoder in Verilog using dataflow level modeling.

Block Diagram:


I_0
I_1
en
2 : 4 Decoder
$Y_0 = \bar{I_1}\,\bar{I_0}\,en$
$Y_1 = \bar{I_1}\,I_0\,en$
$Y_2 = I_1\,\bar{I_0}\,en$
$Y_3 = I_1\,I_0\,en$

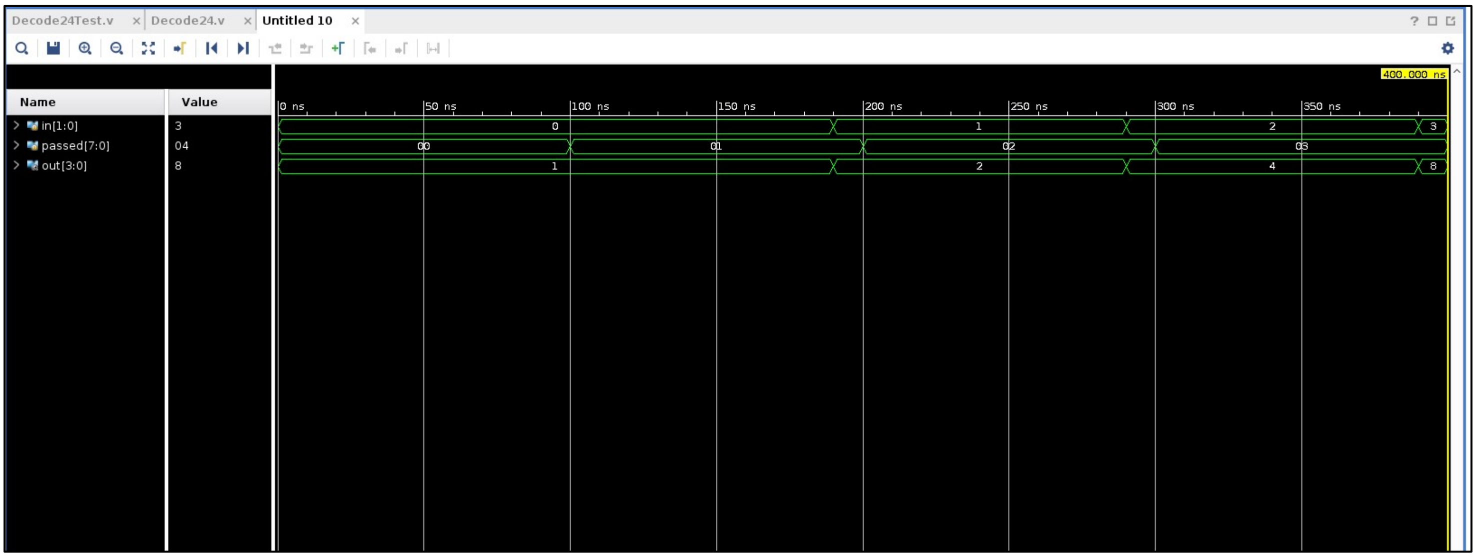
Source code filename: Decode24.v

```
module Decode24(in,out);
       // Input
       input  [1:0] in;
       // Output
       output [3:0] out;
       // Assign outputs according to truth table
       assign out[0] = ~in[1] & ~in[0];
       assign out[1] = ~in[1] & in[0];
       assign out[2] = in[1] & ~in[0];
       assign out[3] = in[1] & in[0];
endmodule
```

Testbench code filename: Decode24Test_v.v

```
`timescale 1ns / 1ps

`define STRLEN 15

module Decode24Test_v ;

    task passTest;

        input [3:0] actualOut;

        input [3:0] expectedOut;

        input [`STRLEN*8:0] testType;

        inout [7:0] passed;


        if(actualOut == expectedOut)

        begin

            $display("%t : %s passed", $time, testType);

            passed = passed + 1;

        end

        else

            $display("%s failed: %d should be %d", testType, actualOut, expectedOut);

    endtask


    task allPassed;

        input [7:0] passed;

        input [7:0] numTests;


        if(passed == numTests)

            $display ("All tests passed");

        else

            $display ("some tests failed");

    endtask


    // Inputs

    reg [1:0] in;

    reg [7:0] passed;


    //Outputs

    wire [3:0] out;
    //Instantiate the Unit Under Test (UUT)
    Decode24 uut (
        .in(in) ,
        .out(out)
    );
    initial begin
        //Initialize Inputs
        in = 0;
        passed = 0;
        //Add stimulus here
        #90; in = 0; #10;
        passTest(out, 1, "Input 0", passed);
        #90; in = 1; #10;
        passTest(out, 2, "Input 1", passed);
        #90; in = 2; #10;
        passTest(out, 4, "Input 2", passed);
        #90; in = 3; #10;
        passTest(out, 8, "Input 3", passed);
        allPassed(passed , 4) ;
    end
endmodule
```

Simulation Output Waveform: (2-to-4 Decoder)



Decode24Test.v × Decode24.v × Untitled 10 ×

```
Name           Value
in[1:0]        3
passed[7:0]    04
out[3:0]       8
```

Output Console Logs: (2-to-4 Decoder)

```
INFO: [USF-XSim-8] Loading simulator feature

Vivado Simulator 2017.4

Time resolution is 1 ps

source Decode24Test_v.tcl

# set curr_wave [current_wave_config]

# if { [string length $curr_wave] == 0 } {

#   if { [llength [get_objects]] > 0} {

#     add_wave /

#     set_property needs_save false [current_wave_config]

#   } else {

#      send_msg_id Add_Wave-1 WARNING "No top level signals found. Simulator will start without
a wave window. If you want to open a wave window go to 'File->New Waveform Configuration' or
type 'create_wave_config' in the TCL console."

#   }

# }

# run 1000ns

              100000 :          Input 0 passed

              200000 :          Input 1 passed

              300000 :          Input 2 passed

              400000 :          Input 3 passed

All tests passed

INFO: [USF-XSim-96] XSim completed. Design snapshot 'Decode24Test_v_behav' loaded.

INFO: [USF-XSim-97] XSim simulation ran for 1000ns

restart

INFO: [Simtcl 6-17] Simulation restarted

run all

              100000 :          Input 0 passed

              200000 :          Input 1 passed

              300000 :          Input 2 passed

              400000 :          Input 3 passed

All tests passed
```

---

4. Answer the following review questions:

(a) In behavioral Verilog, two types of assignment statements exist. What are they, and when would you use one over the other?

Answer:

In behavioral Verilog, two types of assignment statements exist. They are blocking assignments and non-blocking assignments.

Blocking Assignments: (Denoted by ‘ = ’)
Blocking assignment statements are executed one after another in a procedural block, i.e., these assignments are executed in series order. They block the execution of the next statement until the current assignment is completed. Blocking assignment statements are used when we wish to model combinational logic or when the order of execution is important.

Non-blocking Assignments: (Denoted by ‘ <= ’)
Non-blocking assignment statements allows assignments to be scheduled without blocking the execution of the following statements, i.e., these assignments do not block the execution of the next statements. The right side of all statements are determined first, then left sides are assigned together in parallel. Non-blocking assignment statements are used when we wish to model sequential logic elements like flip flops. Non-blocking assignments represent the behavior of registers where data is stored and updated on clock edges.

(b) Compare and contrast the structural versus behavioral implementation of the JK flip-flop. Which level of abstraction might you use for a processor design and why?

Answer:

Structural Implementation:
The structural implementation of the JK flip-flop describes the flip flop using an interconnected network of lower-level components and gates. It describes how the flip flop is built internally using logic gates and describes the propagation of signals through gates. Hence, it provides a low level of abstraction.

Behavioral Implementation:
The behavioral implementation describes the functionality of the JK flip-flop **without** detailing its internal structure of logic gates. It describes the behavior of the JK flip flop to various input signals and the outputs generated. Hence, it provides a high level of abstraction.

For the design of a processor, a behavioral implementation would be preferred. Since processors are complex circuits with many interacting components, designing a processor using a structural implementation would be time-consuming and tedious. Moreover, behavioral implementations are easier to read, modify and maintain since we only need to describe the behavior of the processor to inputs and the desired outputs. Also, the design process can be made simpler if we use synthesis tools to generate the gate-level structural details from our behavioral implementation; rather than pursue the structural implementation for a processor.

(c) Based on the gate level diagram you created for the 2:4 decoder, how would the structural implementation compare to the dataflow implementation? Could you use behavioral Verilog to create the 2:4 decoder? If so, provide a snippet of code to do so.

Answer:

In the structural implementation of a 2:4 decoder, we describe the circuit using lower-level components (AND, OR, NOT gates). It provides a lower level of abstraction, wherein we have full control over the structure of the circuit, logic gates, inputs.

In the dataflow implementation of a 2:4 decoder, we describe the functionality of the circuit using logical expressions. Dataflow implementations are more abstract, easier to read and understand. It provides a higher level of abstraction.

2:4 Decoder Structural Implementation:

```
module decoder_2to4_struct(en, a, b, y);

// Inputs

input en, a, b;

// Output

output [3:0]y;

// Intermediate outputs of NOT gates

wire wa, wb, wen;

// Three NOT gates

not n_1(wa,a);

not n_2(wb,b);

not n_3(wen, en);


// Four NAND gates

// Assign outputs based on schematic

nand o_0(y[0], wen, wa, wb);

nand o_1(y[1], wen, a, wb);

nand o_2(y[2], wen, wa, b);

nand o_3(y[3], wen, a, b);

endmodule
```

2:4 Decoder Behavioral Implementation:

```
module decoder_2to4_behav(input [1:0]I, output [3:0]Y);

// Assign outputs based on truth table

always @* begin

       case(I)

              2'b00: Y = 4'b0001;

              2'b01: Y = 4'b0010;

              2'b10: Y = 4'b0100;

              2'b11: Y = 4'b1000;

              default: Y = 4'b0000;

       endcase

end

endmodule
```