ECEN 651

LAB NO. 2

Behavioural, Dataflow, and Structural Verilog

TA: Mr. Ye Wang

DATE: 09/11/2019

Student Name: Dhiraj Dinesh Kudva

UIN: 829009538

Objective: The objectives of this lab are

1. To learn levels of abstraction to model or program the digital circuit: This lab will help to understand different level of abstraction like Dataflow, Behavioural and Structural Programming in Verilog.

2. Implementation of JK flipflop and D flipflop using Verilog: This lab will also provide an insight of coding in Verilog. With the help of this, I will learn to implement JK and D flipflop in Structural and Behavioural Verilog.

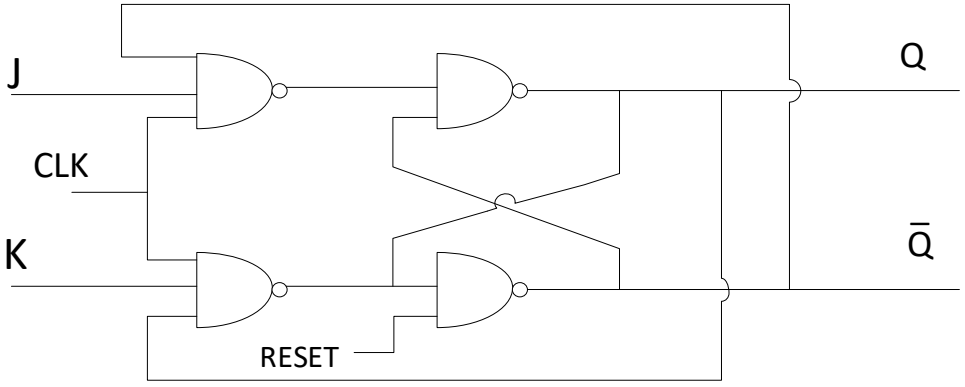
3. Implementation of 2:4 Decoder using Verilog: This lab will also focus on implementation of 2:4 decoder using dataflow level modelling.

Design:

1. JK Flipflop using Structural Level Modelling

The JK flipflop has two inputs: J and K, using which many other types of flipflops can be generated. Hence it is also called as universal flipflop.

The below figure is gate level representation of JK flipflop.



JK FLIPFLOP

STEPS-

JK flipflop using Structural level modelling and Behavioural Modelling:

1. Open the Xilinx Vivado software.
2. Create new project and in that create new source files in that project.
3. Create a separate module for JK flipflop.
4. Define the inputs and outputs.
5. Create source code for JK flipflop using Structural level of modelling. The structural level modelling represents the gate level description.
6. After this step, write test bench for JK flipflop.
7. Mention the delay and clock cycle in this test bench. Initialization of the variables should be done in the test bench.
8. Run the testbench and verify the output.

9. Similarly, create source code for JK Flipflop using Behavioural Modelling. The same testbench can be used and again verify the output.

Truth Table:

Clk	J	K	Q	Qbar
0	X	X	No change	No change
1	0	0	No change	No change
1	0	1	0	1
1	1	0	1	0
1	1	1	Toggle	Toggle

Verilog Code: JK Flipflop Structural

```
`timescale 1ns / 1ps

module JK ( out,j,k,clk,reset); //Module initialization
output out; //Defining outputs
input j, k, clk, reset ; // Defining inputs
wire a,b,q1,q2;
wire Qbar; // Defining internal connections
//gate level modelling

nand #2 N1 (a, clk, j, q2); //providing delay of 2ns for nand gate output
nand #2 N2 (b, clk, k, q1);
nand #2 N3 (q1, a, q2);
nand #2 N4 (q2, b, q1, ~reset); //
assign out=q1; // connecting internal connections to output
assign Qbar=q2;
assign out=~Qbar;                    // Qbar is complement of Q
endmodule          // end of module definition
```

Verilog Code: JK Flipflop Behavioural

```
module JK( out,j,k,clk,reset); // module definition
output out;                          // defining outputs
input j,k, clk, reset ;
reg Qbar;                    //defining inputs
reg out;                             //declaring outputs as reg type
always @ (posedge clk or posedge reset)    //output changes if either clock edge is thereor
reset is high
begin
if (reset == 1) begin                // if reset is high then Q+ =0 irrespective
of clock
```

```
out <= 1'b0;
Qbar <= 1'b1;
end

else if ( j==0 & k==0) begin // J=0,K=0 => Q+ =Q
out <= out;
Qbar <= Qbar;
end

else if ( j==1 & k==0) begin // J=1,K=0 => Q+ =1
out <= 1'b1;
Qbar <= 1'b0;
end

else if ( j==0 & k==1) begin // J=0,K=1 => Q+ =0
out <= 1'b0;
Qbar <= 1'b1;
end

else if ( j==1 & k==1) begin // J=1,K=1 => Q+ =Qbar
out <= Qbar;
Qbar <= out;
end
end
endmodule //module definition ends
```

Testbench for JK Flipflop

```
`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 ("%s passed", 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 j; reg k; reg clk;  reg reset;
reg [7:0] passed;
//outputs
wire out;

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

// initialize inputs
j=0; k=0; clk=0; reset=1; passed=0;

//wait 100 ns for global reset to finish
#100

//Add stimuls 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);
```

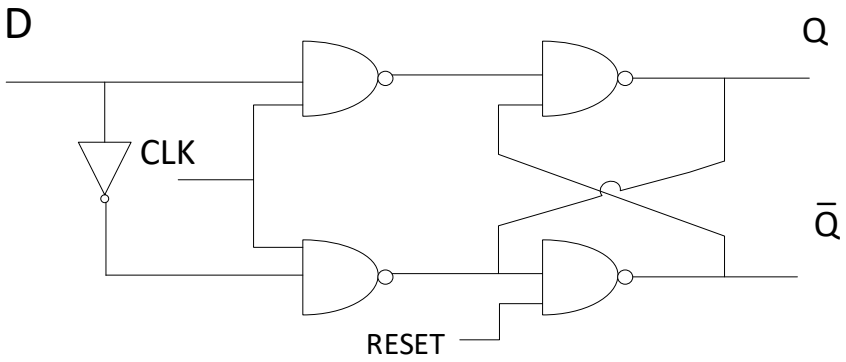
#90; j=0; k=1; #7; clk=1; #3; clk=0;
#90; passTest(out,0,"Reset ",passed);
#90;
allPassed(passed,6);
end
endmodule

Output of JK Flipflop:

2. D Flipflop Using Behavioural Verilog

The D flip-flop tracks the input, making transitions with match those of the input D. The D stands for "data"; this flip-flop stores the value that is on the data line. It can be thought of as a basic memory cell.

The below figure is gate level representation of JK flipflop



D Flipflop using gates.

Steps:

1. Open the Xilinx Vivado software.
2. Create new project and in that create new source files in that project.
3. Create a separate module for D flipflop
4. Define the inputs and outputs.
5. Create source code for D flipflop using Behavioural Verilog.
6. After this step, write test bench for D flipflop.
7. Mention the delay and clock cycle in this test bench. Initialization of the variables should be done in the test bench.
8. Run the testbench and verify the output.

Truth Table

Clk	D	Q	Qbar
0	0	No change	No change
0	1	No change	No change
1	0	0	1
1	1	1	0

Verilog Code for D flipflop

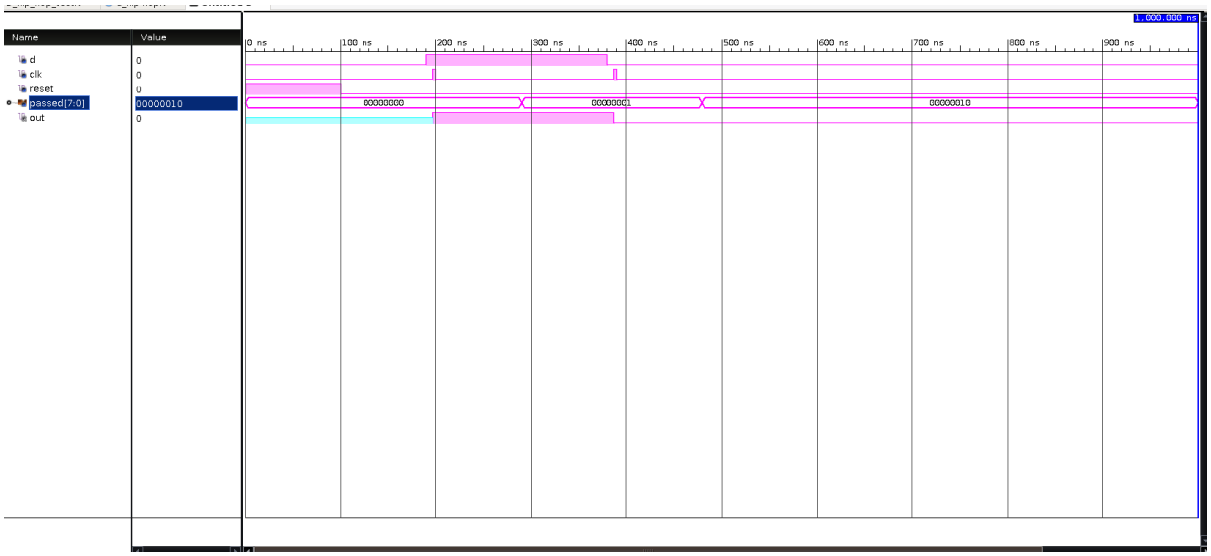
```
`timescale 1ns/ 1ps

module d_flip_flop ( out,d ,clk ,reset );
output out ;
reg out ;

input d ; wire d ; input clk ; wire clk ; input reset ; wire reset ;

always @ (posedge (clk)) begin
 if (reset)
  out <= 0;
 else
  out <= d ;
end
endmodule
```

Testbench for Dflipflop

```
`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 ("%s passed", 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 d; reg clk; reg reset;
reg[7:0] passed;

//outputs
wire out;

//instantiate the Unit Under Test (UUT)
d_flip_flop uut ( out,d ,clk ,reset);
initial
begin
                //initializing 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", passed);

                #90; d=0; #7; clk=1; #3; clk=0;
                #90;
                passTest (out, 0, "Reset", passed);

                #90;
                allPassed(passed,2);
end
endmodule
```

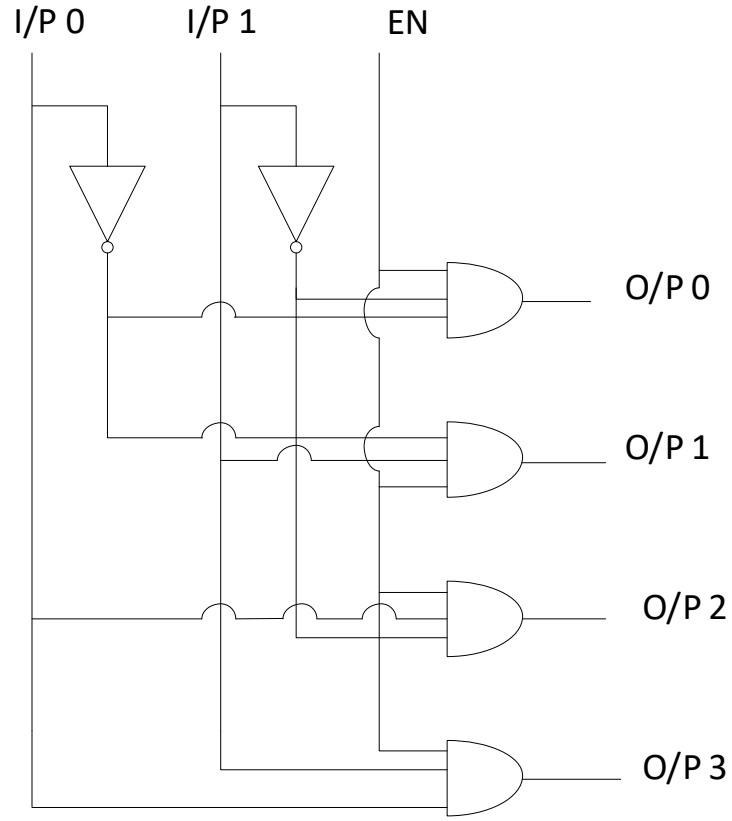
Output for Dflipflop




```
Vivado Simulator 2015.2
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
        input 0 passed
        input 1 passed
        input 2 passed
        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
run all
```

3. **2 to 4 decoder**

The 2:4 decoder have 2 inputs and 4 outputs. The outputs are selected based on the input. The below diagram is a gate level representation of 2:4 decoder.



Steps:

1. Open the Xilinx Vivado software.
2. Create new project and in that create new source files in that project.
3. Create a separate module for 2:4 decoder
4. Define the inputs and outputs.
5. Create source code for 2:4 decoder using dataflow Verilog.
6. After this step, write test bench for 2:4 decoder.
7. Mention the delay and clock cycle in this test bench. Initialization of the variables should be done in the test bench.
8. Run the testbench and verify the output.

Truth Table:

Enable	Input 1	Input 0	Output 3	Output 2	Output 1	Output 0
0	X	X	0	0	0	0
1	0	0	0	0	0	1
1	0	1	0	0	1	0
1	1	0	0	1	0	0
1	1	1	1	0	0	0

Verilog code of 2:4 decoder

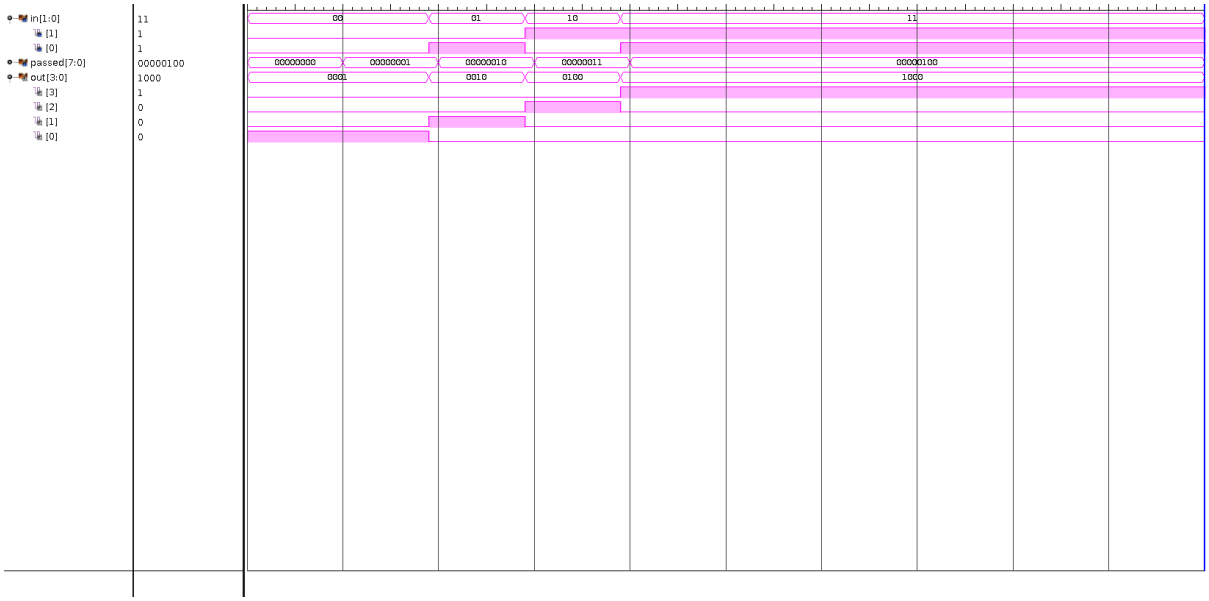
```
module decoder2_4 ( a,x );

input [1:0] a; //input initialization

output[3:0]x; //output initialization

//based on the truth table, the outputs are connected with the inputs.
assign x[0] = (~a[0]) & (~a[1]);
assign x[2] = (~a[0]) & a[1];
assign x[1] = a[0] & (~a[1]);
assign x[3] = a[0] & a[1];

endmodule
```

Testbench of 2:4 decoder

```
`define STRLEN 15
module Decode24Test_v;
task passTest;
input actualOut, expectedOut;
input [`STRLEN*8:0] testType;
inout [7:0] passed;

if (actualOut==expectedOut)
begin
        $display ("%s passed", 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)

decoder2_4 uut ( .a(in), .x(out));

initial begin

// initialize inputs
in = 0;
passed=0;

//Add stimuls 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
```

Output of 2:4 decoder




```
#   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
          input 0 passed
          input 1 passed
          input 2 passed
          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
run all
```

Questions:

a. In behavioural Verilog, two types of assignment statements exists. What are they, and when would you use one over the other?

Ans. The two types of assign statements in behavioural Verilog are blocking and nonblocking assignments. In blocking assignment, the right side value is assigned to the left side variable at that instant itself, whereas for non blocking it assigns the value after that particular clock cycle (mentioned in the always block of which the nonblocking assignment is a part). The blocking assignments are used in case of combinational circuits whereas the nonblocking assignments are preferred for sequential circuits.

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

Ans: The structural level modelling implements in terms of logic gates and interconnection between them. It also uses some of the predefined set of logic gates called as primitives. Whereas the behavioural is the highest level of abstraction. It is implemented in terms of the design algorithm without concern for the hardware implementation. The level of abstraction needed for a processor design mostly depends on the application. If the gate level design is known and the circuit is less complex, then we can prefer structural modelling. However, if the gate level design is

not clear, then based on the algorithm we can proceed with behavioural level modelling.

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 behavioural Verilog to create the 2:4 decoder? If so, provide a snippet of code to do so.

Ans: The dataflow is more related to flow of data in the circuit. It is related to data level implementation of the digital circuit. The dataflow level modelling is similar to logical equations. The structural implementation on the other hand, follows gate level implementation of 2:4 decoder. It uses the predefined modules for gates in Verilog, and programs accordingly. The dataflow level modelling is at a higher level of abstraction than structural modelling.

2:4 decoder Behavioural Verilog Code

```
module decoder2_4(a,x);
input [1:0] a;
output [3:0] x;
reg x;
always @( posedge (a))
begin
if (a[0]==0 && a[1]==0)
        x=4'b0001;
else if (a[0]==1 && a[1]==0)
        x=4'b0010
else if (a[0]==0 && a[1]==1)
        x=4'b0100
elseif  (a[0]==1 && a[1]==1)
        x=4'b1000
end
endmodule
```