ECEN 651: Microprogrammed Control of Digital Systems
Department of Electrical and Computer Engineering
Texas A&M University

Prof. Mi Lu
TA: Ye Wang

Laboratory Exercise #2

Behavioral, Dataflow, and Structural Verilog

Objective

The objective of lab this week is to explain the various levels of abstraction used to model digital logic in Verilog, while reviewing a few architectural level digital components. In Verilog, digital logic can be described in three distinct ways. At one end of the spectrum is structural modeling such that digital logic is conveyed using primitive logic gates. At the other end of the spectrum is behavioral in which a particular circuit behavior is simply described. Somewhere between these two is a method which allows the designer to express how data should flow through the digital circuitry using logic and arithmetic expressions. Typically, a design will consist of a mixture of the aforementioned design description levels. Experience developing hardware in Verilog will reveal which level is more appropriate for a given circumstance.

Procedure

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

   (a) Using Figure 1 as a reference, describe a JK Flip-flop in structural Verilog.

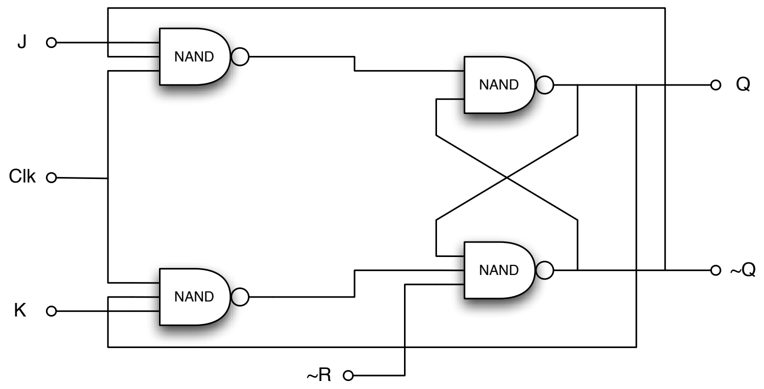

Figure 1: A JK Flip-Flop constructed with NAND gates

(b) Use the code provide below to test your structural model of the JK flip-flop. Demonstrate your progress to the TA.
Note the use of blocking assignment statements in the code below.

```
  `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(out),
                .j(j),
                .k(k),
                .clk(clk),
                .reset(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 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
```

(c) Provide a truth table to describe the operation of a JK Flip-flop.

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.

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

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

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

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.

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

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

   Hint: You can do so succinctly using the ternary operator and the << operator.

   (d) Use the code provided below to test the operation of your decoder. Demonstrate your progress to the TA when complete.

```
`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)
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
```

1 Deliverables

1. Submit a lab report that captures your efforts in lab.

2. Include all Verilog source files with appropriate comments.

   Note: Code submitted without adequate comments will not be graded!

3. Be sure to include all material requested in lab.

4. Answer the following review questions:

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

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

   (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.