**Verilog Lab 3 (ECS1005)**

**Objectives:**

For this lab as well, please download the Verilog HDL book on the local machines desktop for easy accessibility when needed. The lab establishes better concept development for the continuous assignments using operators. The objectives of this lab are as follows.

* Understanding the use of operators in Verilog.
* How to use operators for designing Multiplexers and decoders

**Task 1: Understanding Verilog operators**

Let’s understand the behaviour of various Verilog operators. You may read more about it in section 5.4.3 Verilog Operators.

Open the following repl.

<https://replit.com/@AyeshaKhalid5/ECS1005-Verilog-Lab3-Task1#Testbench.v>

This project has no DUT but only a testbench!. This testbench also produces no waveform as the $dumpvars is commented. This testbench takes you over several operators with their test values, one type at a time. Feel free to change these values as you please to enhance your understanding. When you are comfortable with one type of operators, comment that section and uncomment the next one to run it (group comments have been used here /\* multi-line comment\*/). We have also used $display that is used to write text to the standard output or a file. The syntax is as follows.

$display("text", signal, signal, ...);

We use the %d or %b to represent decimal and binary numbers in the text section, respectively. We represent %d to accommodate the largest possible value for the expression being displayed. We use %0d to display its minimum or non-zero part, suppressing any leading 0's.

Let’s now test your understanding on various Verilog operators. First fill up the expected output and operator type in the middle column (after reading about it from the Verilog Book) and then try to experimentally fill them up using Verilog Repl simulator.

Given two 4 bit numbers

A = 4'b1100; B = 4'b0010;

|  |  |  |
| --- | --- | --- |
| **Expression** | **Expected Output**  **(and operator type)** | **Actual Output**  **(After Verilog simulation)** |
| **A%B** | **A%B = 0**  **It’s the modulus (remainder) operator out of the 5.4.3.9 Numerical operators** | 0 |
| A>B |  |  |
| A!=B |  |  |
| ~B |  |  |
| A^B |  |  |
| | B |  |  |
| A >> 2 |  |  |
| A[3]?B[3]:B[2] |  |  |
| {A[0], A[1], A[2], A[3]} |  |  |
| A-B |  |  |

**Task 2: A Decoder using continuous assignment and bitwise logical operators**

Refer back to the lecture notes if needed when we studied Decoders. A 2 x 4 Decoder will have two input select lines, let’s call them S0,S1 and 4 outputs let’s call them D0,D1,D2 and D3. We also have an enable as an input called en. The truth table and logic diagram for such a decoder is given below.

A white sheet with numbers and letters

Description automatically generated A diagram of a circuit

Description automatically generated

We know that Decoders can be realized by AND gates. The number of gates needed is the same as the number of outputs of the decoder. Open the following repl that has a 2 x 4 Decoder and its testbench using gate level modelling.

<https://replit.com/@AyeshaKhalid5/ECS1005-Verilog-Lab3-Task2#Decoder2to4.v>

Understand and run it. Organize the signals as the following waveform, does your waveform match the following?

A screenshot of a computer screen

Description automatically generated

Open the Verilog file *Decoder2to4.v*. Convert the gate level modelling to the data flow modelling using the assign statements and check if you get the same results. Remove all gates, you will need no more than one assign statement per output of the decoder, total 4 assign statements in the body of the module with no internal wires needed. The assign statement for D0 is provided below.



**Task 3: A multiplexer using continuous assignment and conditional operator**

In this task, we will start with a multiplexer (2:1), written in gate level modelling. We will first extend or scale it to a 4:1 MUX and then convert it to data flow modelling (using assign statement and operators). We will first use the bitwise logical operators to model it (5.4.3.2 Bitwise Logical Operators) and then use the conditional operator (5.4.3.6 Conditional Operators). The variable input table, symbol, and the gate level logic diagram for a 2:1 mux is given below.A diagram of a circuit

Description automatically generated

Fork up the following code

<https://replit.com/@AyeshaKhalid5/ECS1005-Verilog-Lab3-Task3#MUX2.v>

1. Understand the *MUX2.v*. Inorder to make a 4:1 MUX, there is another file specified called *MUX4.v.* Lets try to extend the 2:1 MUX to a 4:1 MUX using 3 instances of the 2:1 MUX in the files *MUX4.v* (as shown in the figure below). You will need two internal wires, tmp0 and tmp1. Let’s call the three instances of MUX2 as M0 (top left mux at level 0), M1 (bottom left mux at level 0) and as M2 (mux at level 1 at the right). Make sure you carefully use the explicit port mapping convention.

A diagram of a circuit

Description automatically generated

Instantiation for M0 is provided below with explicit port mapping.



1. Once done, open the *testbench.v*. Our MUX4.v is instantiated with the name mymux, we have used explicit port mapping. Run the simulation and observe the waveform. Is it working as expected? In the *testbench.v*, try chancing the values of multiplexer inputs (IN0=1; IN1=0; IN2=1; IN3=0;) in the first **initial** block and see if the changes are reflected in the waveform or not. Paste the waveform below.
2. Let’s now change the gate level modelling style of *MUX2.v* to data-flow modelling using assign statements. Comment the line 7-9 in *MUX2.v* and use assign statements with the bitwise Logical operators (~,&,|). You could come up with a single compact expression by simply writing.



Does it match the diagram provided above? Do you need any internal wires?

1. Let’s now try using the conditional operator in our MUX2.v. The conditional operator (?:) takes three operands. Usage: *condition\_expr ? true\_expr : false\_expr ;*

Replace the MUX2.v code body with the following.



Re-Run the simulation and observe the waveform. Is it working as expected?