**Verilog Lab 4 (ECS1005)**

**Objectives:**

The objectives of this lab are as follows.

* Understand the use of two procedural blocks in Verilog (initial, always).
* To practice behavioural modelling constructs like case, if else in Verilog.

**Task0: Let’s fix back the waveforms first.**

In the last weeks lab, most of us faced the problem of not being able to run the waveforms. Let’s fix that first by running the compilation scripts in the following manner. Open and fork up the lab3 task 2 link below.

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

When you click the Run button, it will compile. If it opens up the waveform, then you can simply skip the next step. If the wave form does not open, then simply click the shell screen, right next to the console on the right side of the screen as shown below.

A screen shot of a computer

Description automatically generated

On the shell, copy and paste the following command and enter:

chmod +x run.sh && ./run.sh

As you run this command, the project will compile again and the waveform will launch as well.

Try doing the same with the lab3 task 3 link below.

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

**Task 1: Multiplexer Design using case and If-else structures in behavioural modelling**

The behavioural modelling style in Verilog is the highest level of abstraction and uses constructs that are very similar to C language. Behavioural modelling represents digital circuits at a functional and algorithmic level. It can be used to describe sequential and combinational circuits. In today’s lab, we will use behavioural modelling concept for combinational circuits only.

In this task we will use **case** statement and **If-else** chaining to model 4:1 mux that we have already modelled using gate level modelling in the last lab and the data flow modelling (assign statement with conditional operator). Open and fork up the following project.

<https://replit.com/@AyeshaKhalid5/ECS1005-Verilog-Lab4-Task1#tb_MUX4.v>

Diagram, schematic

Description automatically generated

1. Try to understand the code. There are two instantiations of 4:1 multiplexers, one is undertaken via gate level modelling and instantiation (as we saw in last week’s lab, *MUX4\_gatelevel.v*) while the other is undertaken only via the continuous assignments and conditional operators (*MUX4\_dataflow.v*) as shown above in the figure above. In the testbench (tb\_MUX4), we can compare the outputs of these two multiplexers to be the same. Try changing the values of multiplexer inputs (IN0=1; IN1=0; IN2=1; IN3=0;) in the **initial** block and see if the changes are reflected in the waveform or not after running it again.

Fill up the following table for the given 4:1 mux with inputs w0,w1,w2 and w3.

|  |  |  |  |
| --- | --- | --- | --- |
| **Input {s1,s0}** | **tmp0** | **tmp1** | **Output (f=?)** |
| 00 |  |  |  |
| 01 | w1 | w3 | w1 |
| 10 |  |  |  |
| 11 |  |  |  |

1. Let’s now write the Verilog code for a 4:1 multiplexer using the behavioural modelling. Simply duplicate the MUX4\_dataflow.v file in the project.

Graphical user interface, text, application, chat or text message

Description automatically generated

1. Call the new file *MUX4\_beh\_case.v*. Remove the internal logic to have the following clean interface. Since we are going to use an always block now, the output *f* will be assigned using procedural assignments for which the LHS cannot be a wire. Lets redeclare f as a reg. Simply add a line reg f; under the output f; in the module

Text

Description automatically generated

1. The always block will have a sensitivity list with the names of all the inputs that can cause this procedural block to trigger into action. This will include the select lines and the data lines of the multiplexer. Here is a skeleton code given.

A picture containing chart

Description automatically generated

1. The case statement has the following structure. (Refer to the section 8.2.2 case Statements of the reference book)

Text, letter

Description automatically generated

HINT: the input\_name will be a concatenation of s1 and s0 signals. Complete the case statement, don’t forget the default statement.

1. In the testbench, We now need to observe if the output of the these three modules (*MUX4\_gatelevel.v MUX4\_dataflow* and *MUX4\_beh\_case*) is same or not. We will now have 3 installations in the testbench with same inputs but different outputs. We first declare a new wire as output of the third module we just wrote. Let’s call it out\_MUX4\_beh\_case (add line in tb\_MUX4.v wire out\_MUX4\_beh\_case;). Instantiate the MUX4\_beh\_case in testbench, tie the ports. The three muxes now have the same inputs with different outputs.

Text

Description automatically generated

1. Re-Run the simulation and observe the waveform. Are the outputs of the three modules the same?
2. **DIY**: The 4:1 multiplexer can also be written using an **if- else if** chain. Duplicate the MUX4\_beh\_case.v and lets call the new module MUX4\_beh\_if. (Note it will have almost the same structure as the module MUX4\_beh\_case, except the use of CASE statement)
3. Use the following structure in the module MUX4\_beh\_if

A picture containing text, indoor

Description automatically generated

1. In the testbench, We now need to observe if the output of the four modules *(MUX4\_gatelevel, MUX4\_dataflow, MUX4\_beh\_case* and *MUX4\_beh\_if)* is same or not. We now declare a new wire as output of the new module we just wrote. Let’s call it *out\_MUX4\_beh\_if* (add line in testbench wire out\_MUX4\_beh\_if;). Instantiate the MUX4\_beh\_if in testbench, tie the ports. The four muxes have the same inputs with different outputs.
2. Re-Run the simulation and observe the waveform. Are the outputs of the four modules the same? Paste the waveform here.

**Task 2: A Decoder using procedural assignments with both if-else and case statements**

We worked on a 2 x 4 Decoder in the last lab in gate level modelling. In this task we will write the procedural block equivalent of the code (using always block, with if else and case statement). Lets fork up the following repl.

<https://replit.com/@AyeshaKhalid5/ECS1005-Verilog-Lab4-Task2#Decoder2to4.v>

The decoder will has two bits wide select line, let’s call it S and 4 bits wide data output let’s call it D. Both S and D are arrays here declared as



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

In order to mimic the function of en input, we can use an if else statement and for the select lines, we simply use a case statement as shown below. Don’t forget to redeclare the output D as a reg since it is now assigned within an procedural block (always block).

A screenshot of a computer code

Description automatically generated

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

A screenshot of a computer

Description automatically generated

**Task 3 (DIY-mini project): BCD to seven segment display decoder**

Open up the following link for an article on BCD to seven segment display.

<https://www.electrical4u.com/bcd-to-seven-segment-decoder/>

Diagram

Description automatically generated

We have to write a code for the decoder as shown above. Use the following black project as your starting point

<https://replit.com/@AyeshaKhalid5/ECS1005-Verilog-Lab4-Task3>

Hint: You are free to use case statement or an if- else if- else chain. For every input, you may need begin-end to assign multiple outputs.

Once you are done, run the simulation. Match the output waveform with the truth table below.

A screenshot of a graph

Description automatically generated

Does it match? Paste the waveform below.