ECE275: Lab 5 Prelab
Verilog Functionality Beyond Simple Gates

Pascal Francis-Mezger

October 4, 2021

Assignment Overview

So far for labs we have focused on recreating combinational logic by using direct boolean equations from your boolean algebra with the AND, OR, and NOT operators. This has been to help you solidify that a hardware description language is used to programmatically create a representation of the hardware.

Verilog also has many higher level features that can help you quickly create more advanced hardware designs. Keep in mind when utilizing these features, the synthesizer is still going to create your hardware on the FPGA using simple logic constructs (AND, OR, NOT, etc). So while these tools are powerful they also make it easy to create designs that are very wasteful of hardware.

The goal of this assignment is for you to learn about the more complex features available and how to utilize them in Verilog code. In the lab you will look at the synthesized logic from these designs, and compare the hardware to your expectations.

Assignment

Each question asks you to utilize a simple Verilog feature in code. You do not need to write a full Verilog program with top level for each question. If the question asks for a module, create a working module implementation using the feature. Otherwise, a code ”snippet” just showing the relevant lines of programming is adequate.

Number Representation

Sometimes it is necessary to represent a numeric value in Verilog. It is important when doing so to also specify how many bits should represent that value, especially when it is a signed value (can be positive or negative). Read section 3.5: Number Representation at `https://veriloggguide.readthedocs.io/en/latest/verilog/datatype.html`.

Write Verilog code to create a 4 bit wire variable X, and a 6 bit wire variable Y. Assign X the binary value 0101. Assign Y the decimal value 8.

Arithmetic

In Lab 4 you created a 4 bit adder by utilizing full adders in a ripple configuration. Instead, we could have made this much simpler by just using the addition operator, +.

Modify your 4 bit adder module from lab 4 to use the + operator instead of instantiating your full adder.

Concatenation and Replication

Concatenation is useful to combine values into a single vector in their relevant position. For example, if we want to combine two separate 4 bit values to represent one 8 bit value. Utilize 3.8.5 from the same website in the number representation section to answer the following question.

Utilize concatenation and replication to complete the following code to assign C[9:0] the pattern AABBA

```
module LED_Pattern(
    input [1:0] A,
    input [1:0] B,
    output [9:0] C
);
    assign C = //YOUR CODE HERE
endmodule
```

Conditional Operator

Utilize the same website from the number representation question. Read section 3.8.6 on conditional operators.

Modify your 4 bit multiplexer module from Lab 2 to instead use the conditional operator ?. This should only require 1 assign line.

Always Block and Sequential Logic

The following URL, `https://verilogguide.readthedocs.io/en/latest/verilog/procedure.html` is a great description of the differences in implementing combinational logic versus sequential logic in Verilog. Read sections 4.2-4.5 the answer the following questions.

If you have multiple assign lines in a Verilog program, are they executed in parallel or sequentially? If you have multiple lines inside an always block, are they executed in parallel, or sequentially? If you have multiple always blocks (block A and block B) are the lines of code in block A executed sequentially or in parallel with lines in block B?

Optional

The section below is optional, but covers a useful feature there were questions from students about last class.

Module Generation and For Loops

When instantiating a module multiple times, especially when the module needs to be instantiated a variable amount of times, you can utilize a for loop and generate block. The following URL has a useful example: `https://www.chipverify.com/verilog/verilog-generate-block`

Modify the 4-bit adder module you created in Lab 4 to instead use a generate block to instantiate the full adder modules.