**Lab 3: 8-bit Adder**

Author: Seth Ricks

Date: 10/1/2025

Class: ECEN 340

**Purpose:**

1. To reinforce structural Verilog coding style (figures 3.18 and 3.19 of the textbook).

2. To gain exposure to the Verilog Data Flow programming style (continuous assignment statements found in figures 3.20 and 3.21 of the textbook).

3. To learn how to instantiate modules into a higher-level module.

**Procedure:**

Part 1 – Implement a full adder module called for a 1-bit adder.

Part 2 – Implement an 8-bit adder module by instantiating the full adder circuit 8 time.

**Results/Report:**

Part 1:

In this part of the lab, we created a 1-bit adder module using structural style code. This can be seen in **Figure 1** below.

A screenshot of a computer program

AI-generated content may be incorrect.

Figure

Part 2:

In the second part of the lab, we used the 1-bit adder 8 times to create an 8-bit adder, which can be seen in our code in **Figure** **2** below. The 8 bit adder is the module named ‘adder8,’ while our 1-bit adder remains ‘fulladd.’ The generated schematic for the 8-bit adder can also be seen in **Figure 3** below.

A screenshot of a computer program

AI-generated content may be incorrect.

Figure

A diagram of a computer

AI-generated content may be incorrect.

Figure

Part 3:

As offered, my lab partner and I decided to do the extra credit. This involved re-writing our 1-bit adder to use behavioral code instead of structural. We were able to do this successfully, as seen in **Figure 4** below.

A screenshot of a computer program

AI-generated content may be incorrect.

Figure

We were also offered more extra credit if we re-wrote our 8-bit adder using a generate statement and a for loop. My partner and I were also able to do this successfully, as seen in **Figure 5** below. We renamed our 8-bit adder ‘addern,’ while our 1-bit adder remains ‘fulladd.’

A screenshot of a computer program

AI-generated content may be incorrect.

Figure

**Conclusion:**

This lab was fun and challenging. I used the generate and parameter statements, which were new tools that I haven’t implemented before. I learned that in Verilog, ‘generate’ is needed when instantiating other modules. ‘Parameter’ is useful in loops, as a constraint for the repeated code. It can be easily changed, so if we wanted to do a 5 or 9 bit adder, it would be easier to change this code than to rewrite the instantiations of the adder modules as seen in the structural style code. Overall I enjoyed and learned from this lab.