Temple University

College of Engineering

ECE 4612: Advanced Processor Systems

A red square with white letters

Description automatically generated

Adam Love

64-Bit Adder Design

10/26/2023

**Objective**

This report details the design and implementation of a 64-bit adder component, programmed in Verilog. The adder that is designed will be used as one of the components to create a single-cycle processor. The adder will be used as its own component, as well as within the arithmetic logic unit (ALU).

**Tools/Equipment**

The adder and all test benches were programmed using Verilog and were programmed within VS code. The adder was tested using Vivado to generate output waveforms.

**Procedure**

The adder was specified to be built from the bottom up. Starting at a half adder; using the half adder to build a full (1-bit) adder; using the full adder to build a 4-bit adder; using the 4-bit adder to build a 16-bit adder; using the 16-bit adder to build a 32-bit adder; then finally using the 32-bit adder to build a 64-bit adder. Designing in this way meant that, after the half and full adders, very little new code had to be written. The code from the previous iteration could be reused with the main changes being to the input and output register size. Upon reaching the 64-bit adder, an additional overflow flag had to be added. To detect overflow, the most significant bits of both input values and the output summation are compared. If there is a difference between

**Testing**

To test that the adder was designed correctly, multiple test benches had to be created. Test benches were created that tested every possible input state for the half adder, full adder, and 4-bit adder. After the 4-bit adder, only specific cases were tested for the 16-bit, 32-bit, and 64-bit adders, since testing every case would take too long and too much processing power. For the 16-bit and 32-bit adders, the tests were not very rigorous as they were mainly to make sure that nothing was broken at that level or below that level. The 64-bit adder had 7 test values run on it. The tests were testing 2 random inputs that would not cause a carry or overflow, testing those same inputs with the carry in, testing the overflow by using the highest and lowest positive signed integers, testing the two highest signed integers, testing the lowest signed integers, testing the highest and lowest signed integers, and using the carry in to generate an overflow.

**Results/Observations**

The 64-bit adder was able to successfully complete all the tests that were run on it. It correctly generated an overflow flag when expected, as well as the carry out flag. The waveforms from each test are shown below:

A screenshot of a computer

Description automatically generated

Figure : Addition of two random numbers (expecting no carry and no overflow)

A screenshot of a computer

Description automatically generated

Figure : Addition of the same random numbers, with carry in

A screenshot of a computer

Description automatically generated

Figure : Testing for overflow (highest and lowest positive signed integers)

A screenshot of a computer

Description automatically generated

Figure : Addition of two highest signed integers

A screenshot of a computer

Description automatically generated

Figure : Addition of two lowest signed integers

A screenshot of a computer

Description automatically generated

Figure : Highest and lowest signed integers

A screenshot of a computer

Description automatically generated

Figure : Using carry in to generate overflow

**Conclusion**

Starting this final project with the adder was a good reintroduction to Verilog programming. It allowed me to remind myself about hierarchical design and was a good way to remember how to create test benches. The 64-bit adder can now be used in the rest of the design without fear that it is working incorrectly. The adder will be used twice on its own in the overall design and will also be used to build the ALU.