

**Table of Contents**

|  |  |
| --- | --- |
| Title | Page Number |
| Table of Contents | 1 |
| Objective | 2 |
| Multiplier | 2-3 |
| Adder | 4-5 |
| Conclusion | 5 |

**Objective**

The objective of this lab is to explore the advantages of using lower precision floating point numbers, such as 16-bit or 8-bit, in machine learning applications. The ultimate aim of this project is to develop a tailored floating point unit suitable for embedded systems utilized in machine learning.

**Floating Point Multiplication**

Graphical user interface, text, application

Description automatically generated

Figure 1.1 32-bit Multiplication Simulation results

The photo above displays the results for the 32-bit multiplication of our 100 randomly generated inputs. Given that the exponents of the product are in range, our multiplier shows the A and B values in their hex and the IEEE values converted to decimal. It then displays the expected product and our actual product to compare. It then spits out the percent error between the two products. If the exponents of the product are smaller than the range, our multiplier will display “underflow” and if the exponents are greater than the range, our multiplier displays “overflow.”

Graphical user interface, application

Description automatically generated

Figure 1.2 16-bit Multiplication Simulation results

The photo above displays the results for the 16-bit multiplication of our 100 randomly generated inputs. Given that the exponents of the product are in range, our multiplier shows the A and B values in their hex and the IEEE values converted to decimal. It then displays the expected product and our actual product to compare. It then spits out the percent error between the two products. If the exponents of the product are smaller than the range, our multiplier will display “underflow” and if the exponents are greater than the range, our multiplier displays “overflow.” The percent error in this case is bigger as more errors occur when working with smaller bits.

**Floating Point Addition**

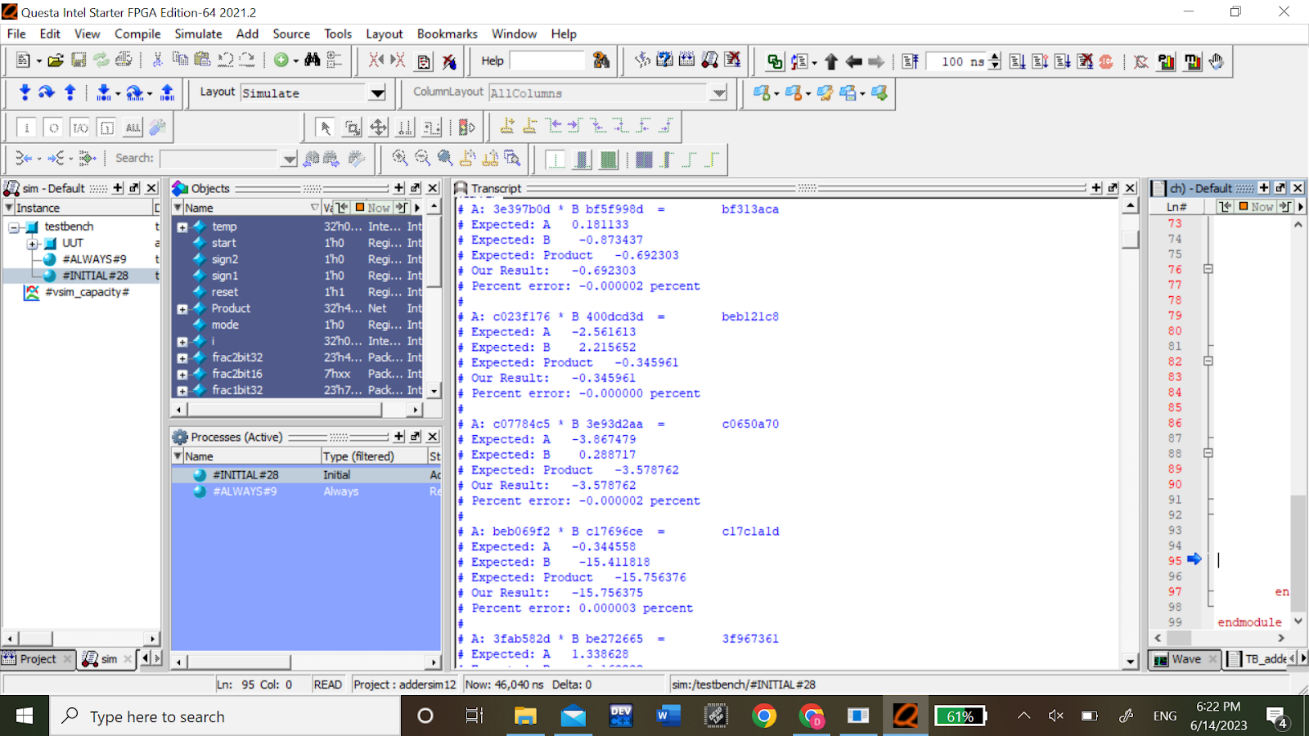


Figure 1.3 32-bit Addition Simulation results

Figure 1.3 contains the inputs and outputs of the addition and subtraction module, their decimal counterparts, and a percent error between the expected and our calculated. In this 32 bit calculation it can be seen that the percent error is relatively small compared to the expected and confirms the validity of our module.

A screenshot of a computer

Description automatically generated

Figure 1.4 16-bit Addition Simulation results

Figure 1.4 contains the inputs and outputs of the addition and subtraction module, their decimal counterparts, and a percent error between the expected and our calculated. In this 16 bit calculation it can be seen that the percent error is relatively small compared to the expected and confirms the validity of our module. When compared to the percent error of the 32-bit, it can be seen that the error is magnitudes larger(while still showcasing validity), due to less bits being available in the fraction portion.

**Conclusion**

In this lab, we learned how to design and implement a custom floating-point unit for machine learning applications in embedded systems. We learned that the use of FPGA implementation of floating-point units offered two key benefits: the ability to instantiate multiple units in parallel on the FPGA, enhancing the performance for operations like matrix multiplication, and the customization of the floating-point unit to match application-specific accuracy, power, and performance constraints. Overall, the construction of the 16-bit and 32-bit multiplier and adder/subtractor was a success as we were able to get a minimum error percentage for each outcome that was not an overflow or underflow.