Floating Point Addition and Subtraction

List of Authors (Saher Al-Khrayyef, Raymon Benjamin, Nathan Sketch)
Electrical and Computer Engineering Department
School of Engineering and Computer Science
Oakland University, Rochester, MI
smalkhra@oakland.edu, rbenjam2@oakland.edu, nksketch@oakland.edu

Abstract— **Addition, Subtraction, and Multiplication floating point numbers are a critical requirement for Digital Signal Processing (DSP) applications involving large dynamic range. This paper focuses on a generic precision floating point normalized addition, subtraction, and multiplication targeted for ZYBO Zynq-7000 FPGA based on VHDL. It handles the overflow and underflow cases. Truncation is implemented to give more precision.**

I. Introduction

Floating point numbers are one possible way of representing real numbers in binary format, the IEEE 754[1] standard presents two different floating point formats, Binary interchange format and Decimal interchange format.

We used the ZYBO Zynq-7000 FPGA board to implement our project. We created a custom IP and connected it to the AXI4Lite interface to communicate to the ARM processor. The computing language that we used to write the program is VHDL, which works well to program the FPGA board that we used. Another computing language that we used is C. Our project focused on addition, subtraction, or multiplies two floating point numbers. We used a generic precision floating point normalized addition subtraction, and multiplication and implement it on the Zyng-7000 FPGA board.

This project will improve our teamwork, our skills in VHDL and C programing language. Also, it will help us to get more experience working with FPGA boards. The class lecture was really helpful to our project because it covered important topics that we needed to understand for addition and subtraction method. The thing that we did a lot of research about it to implement and understand was the leading zero detector.

II. Methodology

In this section, we will provide a background on the operation of our floating point addition and subtraction project.

A. *Floating Point Addition and Subtraction Algorithem*

The precision of the floating point number was used as shown in the figure (1).

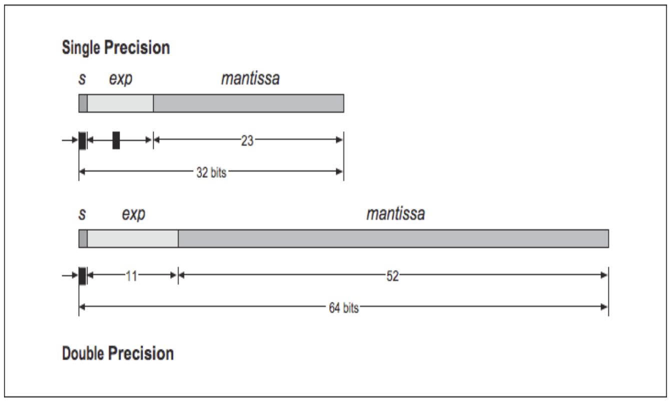


Figure 1: Single and Double Precision Floating Point

Single and double precision floating point represent the format of the floating point number. Where s is the sign bit, and exp is the exponent, which depends on the binary number size. Mantissa, also it depends on the binary number size. Bias for single is 127 and for double is 1023.

We used IEEE-754 STANDARD to calculate the floating point number. This represents as follow:

$$X = \pm 1.f \times 2e$$

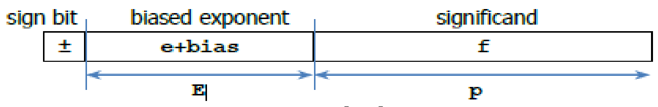


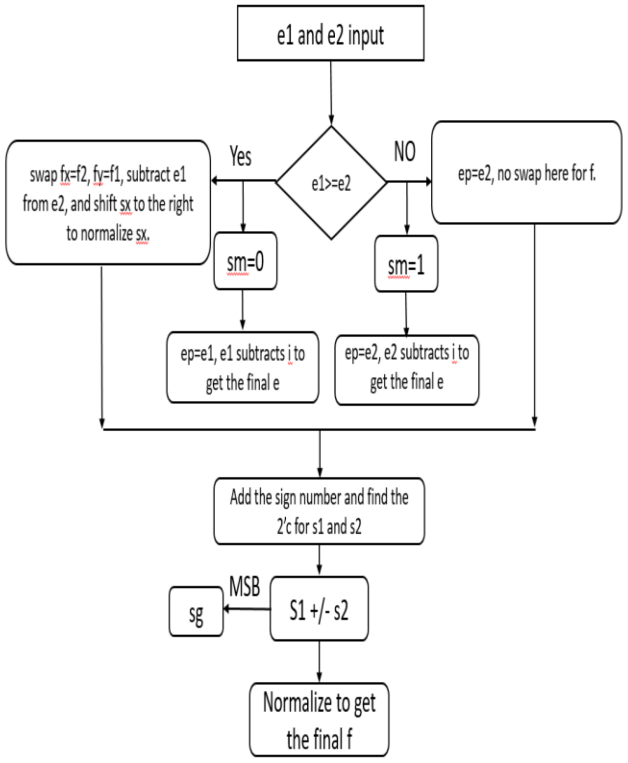
Figure 2: IEEE-754 Standard Representation

Significand: f is the mantissa. We should add 1 to the beginning of the mantissa before start addition/subtraction.

Significand range should be $[1, 2 - 2 - p] = [1, 2)$. Biased exponent E is the number of the bits. Bias = 127 or 1023.
e= exp – bias.

B. Design Floating Point Addition and Subtraction

Floating point addition and subtraction can be done by inputting e1 and e2, and then we compare them together. If e1 greater than or equal e2, we swapped the mantissas to be like this fx=f2 and fy=f1. Moreover, e1 subtracted e2 and the output would shift the sx number to the right to normalize sx. Also, the selector would be a zero that would make ep=e1, and e1 subtracted i that obtained from the LZD to get the final e. On the other hand, if e1 less than e2, there was no swapping for the mantissas. The selector was a one that would make ep=e2, and e2 subtracted i to get the final e. Then we added the sign number for s1 and s2, and we found the 2'C for them before we added them or subtracted them. Once we did that, we could do the addition or subtraction operation on the numbers to get the final f. The MSB is the sign number. The following figure is the flow chart diagram for addition and subtraction.



e1 and e2 input
e1>=e2
Yes
NO
swap fx=f2, fy=f1, subtract e1 from e2, and shift sx to the right to normalize sx.
ep=e2, no swap here for f.
sm=0
sm=1
ep=e1, e1 subtracts i to get the final e
ep=e2, e2 subtracts i to get the final e
Add the sign number and find the 2'c for s1 and s2
MSB
sg
S1 +/- s2
Normalize to get the final f

Figure 3: Flow Chart Floating Point Addition and Subtraction

Normalized Left Shift the normalized left shift in the post-normalization step removed leading zeros in order to follow floating point format. This did not work when the result of the adder-subtractor was all 0's. This case was handled by the special cases matrix.

Below is the full circuit for the addition and subtraction of floating point.

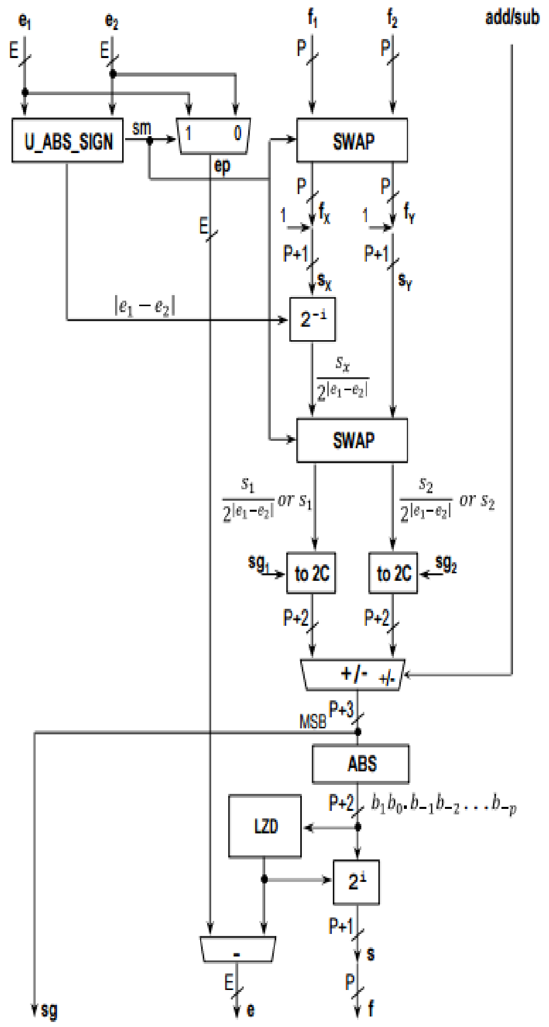


Figure 4: Floating Point Addition and Subtraction Circuit

C. Floating Piont Multiplication Algorithem

In order to find the floating point multiplication, we needed to add the two exponents from each number together, and then subtract their sum from the bias. Bias could be 127 or 1023. Also, we could obtain the sign bit by xor-ing the MSB from each number, and then we normalized the result. We used truncation for the final result to get the right format for the final result.

D. Design Floating Point Multiplication

To multiply the two numbers, we checked if the number one or number two was zero. If one of them was a zero, so the result should be zero. If none of them was zero, so we added the exponents together. And then we subtracted them from the bias. After that, we multiplied the significands. And then we normalized the result to get the final product in the right format. The figure 5 below shows the flow chart of the floating point multiplication.

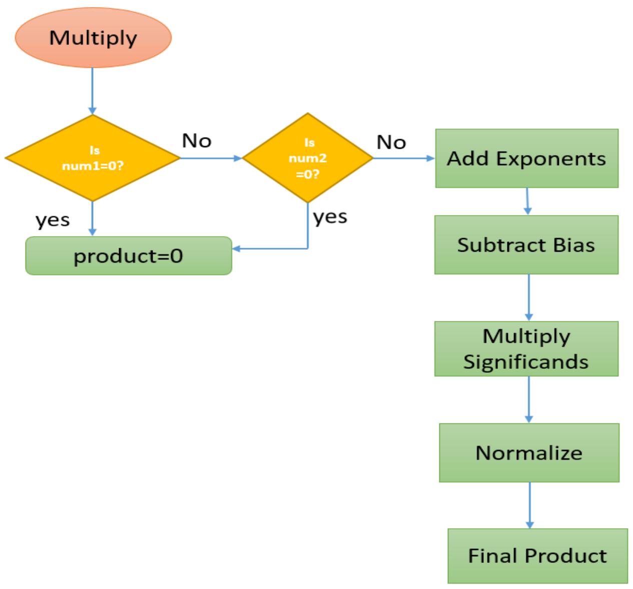
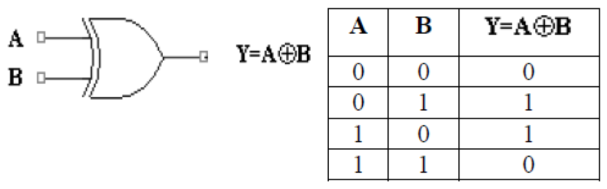


Figure 5: Flow Chart of Floating Point Multiplication

Sign Bit Calculation was calculated by multiplying two number's result is a negative sign if one of the multiplied numbers is of a negative value. By the aid of a truth table we find that this can be obtained by XOR-ing the sign of two inputs. The figure 6 shows the XOR gate and the truth table.

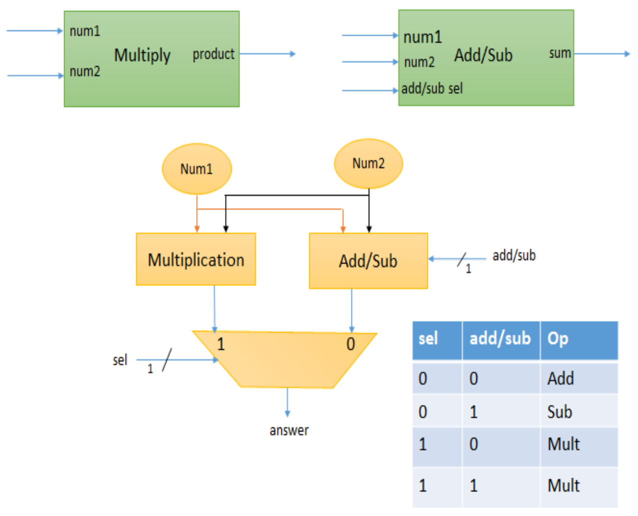


A	B	Y=A⊕B
0	0	0
0	1	1
1	0	1
1	1	0

Figure 6: Sign Bit Calculator XOR Gate

E. *Floating Point Top Circuit*

After we created the files for addition, subtraction, and multiplication, we tested them individually to see if we got the right result or not. After we got the right result, we combined all of them in one file to have them working in one circuit, and then we tested the final circuit.
The figure7 below shows the top floating point circuit.

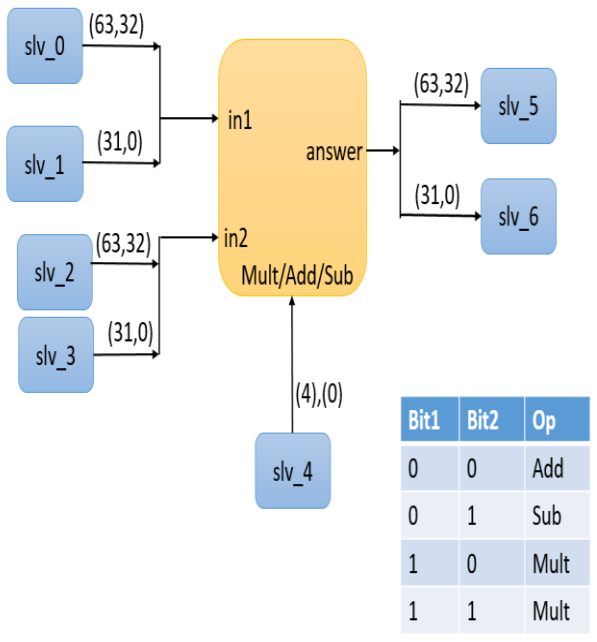


sel	add/sub	Op
0	0	Add
0	1	Sub
1	0	Mult
1	1	Mult

Figure 7: Top Floating Point Circuit

F. *Custom IP*

We created a custom IP using the Vivado Create and Package IP tool. In order to connect with the system on chip (SoC) an IP needs to be created. The interface used to create the IP was the AXI4Lite. A total of seven slave registers were used. Five registers were dedicated to the inputs and two were dedicated for the outputs. Below is a figure AXI4Lite Slave Register Interface. A block design was created to connect the IP to the ZYNQ arm processor. Below is a figure of the IP block diagram.



Bit1	Bit2	Op
0	0	Add
0	1	Sub
1	0	Mult
1	1	Mult

Figure 8: AXI4Lite Slave Register Interface

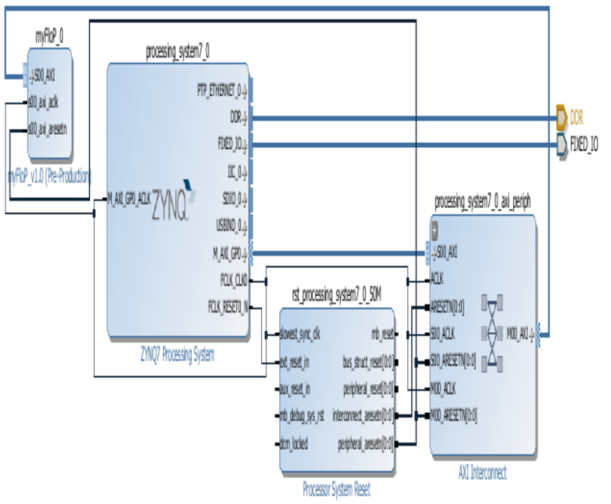


Figure 9: IP Block Diagram

III. Experimental Setup

To verify our project we simulated different test cases in Vivado 2015.2. We picked different test cases that will give us an overview to view if the circuit is working. The tests included all of addition, subtraction, and multiplication. One test is zero added/subtracted to zero or another number. A second is a very large exponent added/subtracted to a very small exponent. Another test is an addition and subtraction that will result in a very large mantissa shift, for example a 52 bit shift for a double floating point arithmetic. We simulated the multiplication circuit in a similar fashion. We tested the top circuit and IP. The hardware tools used were Vivado 2015.2. The software tools used were Xilinx SDK.

IV. Results

The results were as expected for the addition, subtraction, multiplication, and top circuits. The results are displayed in the figures below.

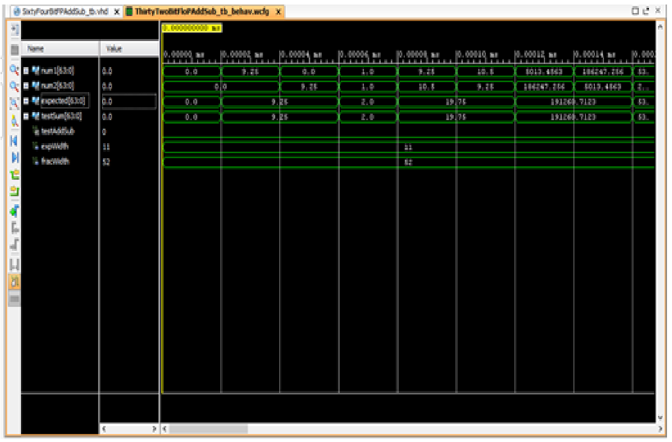


Figure 10: Addition Test bench

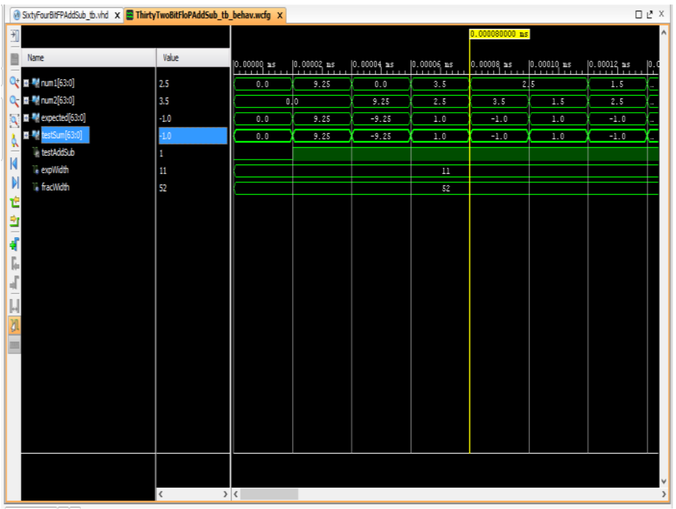


Figure 11: Subtraction Test bench

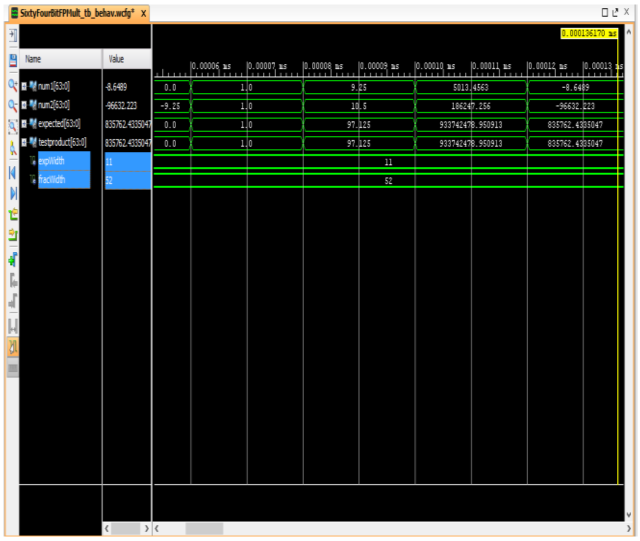


Figure 12: Multiplication Test Bench

After we simulated the top circuit we got the right result. The following simulation shows the top floating point circuit simulation.

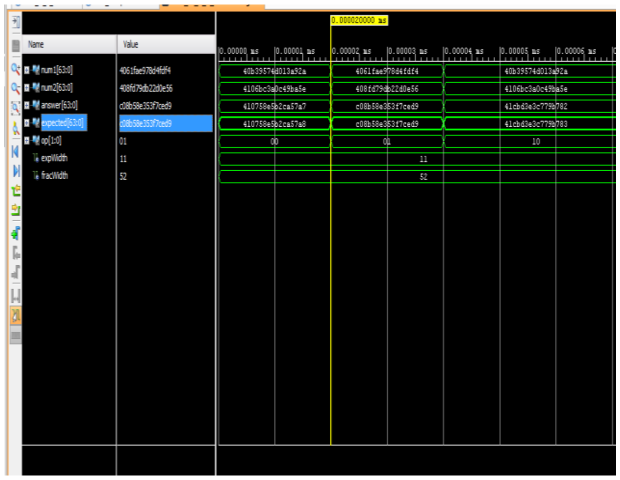


Figure 13: Top Floating Point Arithmetic Circuit Simulation

CONCLUSIONS

We have learned a lot from this project and are thankful we decided to do this project. We have learned about the process and flow chart to the addition, subtraction, and multiplication floating point circuit. We learned about the leading zero detector and the barrel shifter. We learned about the AXI interface. Also, how to create and connect custom built IPs to an ARM processor. Improvements that can be made are to create a functional divider circuit. Also, to test more cases, by generating a list of test cases and using Vivado to run the simulation. Another possibility is to use Berkeley TestFloat to test the circuits.

REFERENCES

[1] IEEE 754-2008, IEEE Standard for Floating-Point Arithmetic, 2008.

[2] Llamocca, Daniel. "Reconfigurable Computing Research Laboratory." Web. 13 Dec. 2015. <http://www.secs.oakland.edu/~llamocca/Fall2015_ece495.html>.