Adder/Subtraction/Multiplier Complex Floating Point Number Implementation over FPGA

W. Saad, N. El-Fishawy, S. EL-Rabaie, and M. Shokair
Dep. of Electronic and Communication Eng., Faculty of Electronic Engineering, El-Menufiya University, Egypt.
Email: waleedsaad100@yahoo.com, nelfishawy@hotmail.com, srabie1@yahoo.com, i_shokair@yahoo.com

***Abstract*—Field Programmable Gate Array (FPGA) devices have become the technology of choice in small volume modern systems and their capabilities will continue to improve in the future. In general, FPGAs are the best at tasks that use short word length integer, and exhibit a high degree of parallelism. However they are not so good at high precision floating point arithmetic. That is because of the inefficient routing and resource utilization for long word length integers. In this paper, a proposed pipelined combinational circuit design is introduced to add, subtract, and multiply two complex floating point numbers. Three different synthesize Electronic Design Automation (EDA) tools are used to compare between them, in order to achieve the best synthesize result. Post simulation is done to show the timing effect of the designed circuit.**

***Index Terms*—FPGA, EDA tools, Floating point, and Synthesize.**

I. Introduction

Field programmable gate arrays are digital integrated circuits (ICs) that contain configurable (programmable) blocks of logic along with configurable interconnects between these blocks. Design engineers can configure (program) such devices to perform a tremendous variety of tasks. A recent trend has been to take the coarse-grained architectural approach a step further by combining the logic blocks and interconnects of traditional FPGAs with embedded microprocessors and related peripherals to form a complete system on a programmable chip. High performance and scientific computing applications are typically compute-intensive and require high throughput rates. Applications of FPGAs include digital signal processing, software-defined radio, aerospace and defense systems, Application-Specific Integrated Circuit (ASIC) prototyping, medical imaging, computer vision, speech recognition, computer hardware emulation, and a growing range of other areas [1].

FPGAs are programmed using a logic circuit diagram or a source code in a hardware description language (HDL) to specify how the chip will work. VHDL stands for VHSIC (very high-speed integrated circuit) Hardware Description Language. It is commonly used as a design entry language for FPGA and ASIC devices.

This paper is organized as follows; Section II, gives an overview of the programmable devices evolution. Section III, introduces the concept of the floating point representation and the applied format for the designed circuit. The behavioral discussion of the proposed circuit implementation is discussed in Section IV. The circuit implementation and the comparison between the synthesize EDA tools are made in Section V. Section VI, shows the timing simulation for the designed circuit. Finally, conclusions are made in Section VII.

II. The Programmable Devices

One of the first techniques that allowed users to program their own devices were (and still are) known as fusible-link technology and antifuse counterparts. These technologies are one-time programmable ones.
Programmable Read Only Memory (PROM) was the first programmable devices and it was a one-time programmable one. One alternative is a technology known as erasable programmable read-only memory (EPROM) that could be programmed, erased, and reprogrammed with new data several times. The next rung up the technology ladder appeared in the form of electrically erasable programmable read-only memories (EEPROMs) that could be electrically reprogrammed instead of using ultraviolet radiation as in EPROMs. The new development was the Flash based technology which was a rapid technology compared with EPROM. Static random-access memory (SRAM) was the next stage in the programmable devices evolution. SRAM will remain unchanged unless it is specifically altered or until power is removed from the system [2], [3], [4].

In digital IC fabrication, there are two ways for design which are general purpose or commercial IC and personal purpose or programmable IC. Personal purpose IC can be categorized into two branches; ASIC and programmable devices. ASIC could be full custom or mask programmable gate arrays while the programmable devices can be PLD or FPGA. PLD devices are SPLD and CPLD. CPLD consists of more than one SPLD. It is faster and cheaper. Programmable interconnect matrix is used to interconnect between SPLDs [4].

The most common FPGA architecture consists of an array of configurable logic blocks (CLBs), I/O pads, and routing channels, as shown in Figure 1. Figure 2 shows a classic FPGA logic block which consists of a 4-input lookup table (LUT), and a flip-flop [3], [5], [6]. In recent years, manufacturers have started moving to 6-input LUTs in their high performance parts, claiming increased performance.

III. Floating Point Numbers Representation

Floating point is another format to represent a number. With the same number of bits, the range in floating-point format is much larger than that in signed integer format. Although

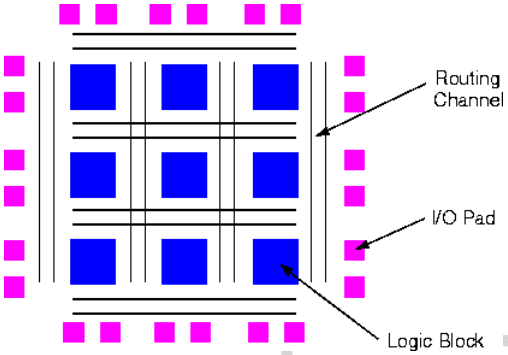

Figure 1. FPGA architecture.

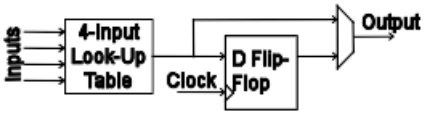

Figure 2. A classic FPGA logic block.

VHDL has a built-in floating-point data type, it is too complex to be synthesized automatically.

The Institute of Electrical and Electronics Engineers (IEEE) standardizes floating-point representation in IEEE 754. Floating-point representation is similar to scientific notation in that there is a number multiplied by a base number raised to some power. For example, 4.325 is represented in scientific notation as 1.078×2^2 or 1.081×2^2 if a high precision is required.

A float consists of three parts: the sign bit, the exponent, and the mantissa (fraction). The sign bit is 0 if the number is positive and 1 if the number is negative. The exponent represents the power factor of the base 2. The mantissa (fraction) is the normalized binary representation of the number to be multiplied by 2 raised to the power defined by the exponent.

For most of the previous implementations of the floating point units, the precision of the mantissa and exponent was usually assumed to be the critical parameter [7]. Much work in the past has been done on designing floating point units on FPGAs [8]. However designing floating point units nowadays has become much easier with resources like DSP blocks, fast embedded multipliers, etc....

In this paper, the area and resource utilization are analyzed for a pipelined floating point adder/subtractor and multiplier. In our representation for the floating number, the exponent is a 5-bit number that ranges in value from 0 to 31 unsigned numbers, and the fraction is a 6-bit number precision. Therefore, my floating number is 12 bits. Then 4.325 can be represented as 010001000101 which is 1.078×2^2.

IV. Designed Circuit Functional Explanation

A. Computation of complex value

Suppose two real or floating point values a and b, complex value can be a+jb. Then two complex values a+jb and c+jd can be added/subtracted as follows
(a+jb) $\pm$ (c+jd) = (a$\pm$c) + j(b$\pm$d). Then, complex addition/substraction just needs two floating point adders/substractors as shown in Figure 3.

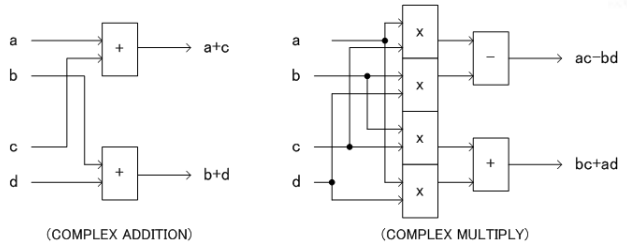

Figure 3. Complex addition/substraction/multiplication.

The two complex values can be multiplied as follows
(a+jb) $\times$ (c+jd) = (ac$-$bd) + j(ad+bc). Then, complex multiply needs 4 floating point multipliers and 1 subtraction and 1 addition as shown in Figure 3.

B. Floating point adder/substractor circuit design

Much work in the past has been done on designing floating point using matrix and vector operations for multiplication which are suitable for pipelined implementations [8]. However, in this paper, a combination circuit is designed using embedded DSP multiplier blocks.

A simplified 12-bit floating point format is used in this circuit design. The representation consists of a sign bit (s), a 5-bit exponent field (e), and an 6-bit fraction field (f). In this format, the value of a floating-point number is $(-1)^s \times 1.f \times 2^e$. Both exponent and fraction fields are in unsigned format. Therefore, under these conditions, the smallest nonzero number can be represented in this format is $(-1)^s \times 1.000000 \times 2^{00000}$ which is 0.00003, and the largest one can be represented as $(-1)^s \times 1.111111 \times 2^{11111}$ which is 130048.

The computation of add/sub two floating point numbers is done in four major steps as shown from Figure 4:

- Sorting: puts the number with the larger magnitude on the top and the number with the smaller magnitude on the bottom.
- Alignment: aligns the two numbers so they have the same exponent. This can be done by adjusting the exponent of the small number to match the exponent of the big one. This can be done by shifting right the smaller number fraction according to the difference between the two exponents.
- Addition/subtraction: adds or subtracts the fractions of two aligned numbers. The correct operation can be defined by Xored the sign bits of the two numbers with the desired operation (add/sub).
- Normalization: adjusts the result to normalized format. This is done by rounding the result of add/sub till the most significant bit (MSB) is 1 and subtract the leading zeros from the exponent. Two types of special normalization may be required:
  - After an addition, the result may generate a carry-out bit. This is resolved by adding one to the exponent and do not round the result fraction.
  - After a subtraction, the result may be too small to be normalized. This is resolved by approximated the result to zero.

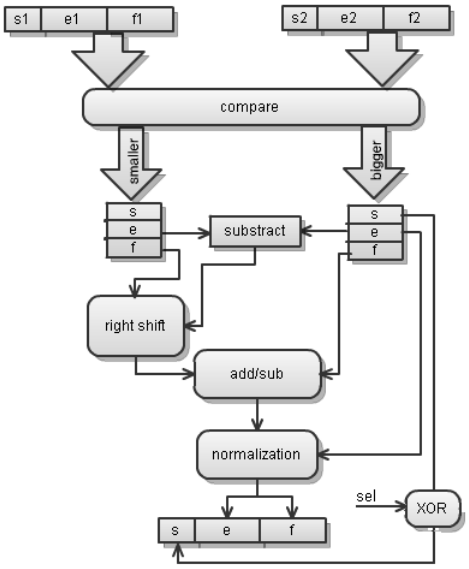

Figure 4. Floating point Adder/Subtractor.

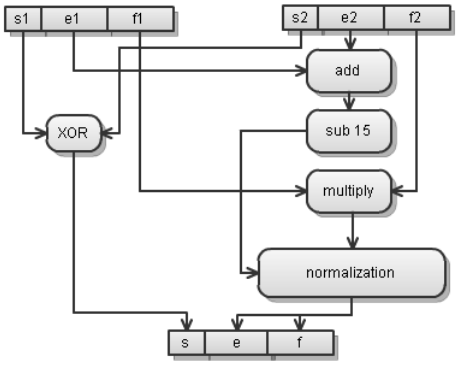

Figure 5. Floating point Multiplier.

C. Floating point multiplier circuit design

In the computation of multiply two floating point numbers, sorting step can be eliminated as it will be useless. Also the alignment step, which is pursuant to the sorting step, can be neglected. Therefore, the multiplication process is done by these steps, as shown from Figure 5.

- Addition: add the two exponents and then substract 15 from the result to let the result exponent in unsigned format.
- Multiplication: multiply the two fractions.
- Normalization: the result from 7-bits multiplication (the 6-bit fraction field after adding the one MSB) is 14-bits. These 14-bits are rounded till be normalized then neglect the lower significant bits to have 6-bit fraction output.

V. CIRCUIT IMPLEMENTATION

The complex floating point number Adder/ Substractor/ Multiplier circuit needs only some floating point Adders/ Subtraction/ Multipliers circuits as described in the previous section. As shown in Figure 6, the two input complex numbers can be represented as the real and imaginary parts (I1, I2, Q1,

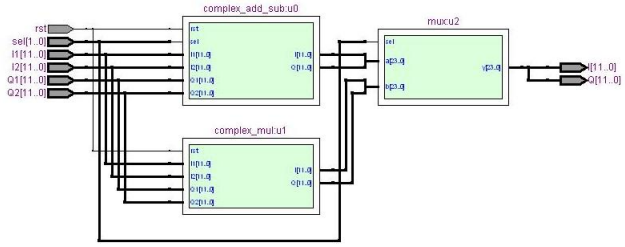

Figure 6. Adder/Substractor/Multiplier circuit.

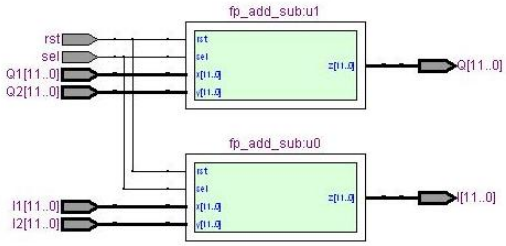

Figure 7. Complex add/sub circuit.

Q2) each is 12-bit floating point. The 2-bit input sel port is used to indicate the desired operation; "00" for addion, "01" for subtraction, and "10" for multiplication. This is done by using a simple multiplexer. The rst input port is used to reset the output (I,Q) when it is activated high.

The complex add/sub circuit consists of two floating point adders/subtraction as shown in Figure 7. Also inside the complex multiplication circuit, 4 floating point multipliers and 1 subtraction and 1 addition are found as illustrated from Figure 8.

This circuit is designed using VHDL and synthesized using three EDA tools which are ISE 10.1 [9], Quartus II [10], and Precision [11] (which is introduced by Mentor Graphics FPGA Advantage 7.0 PS) to compare their results and choose the best one. The design can be implemented on any FPGA target. The analysis and synthesis summary for the ISE and Precision tools for Virtex2p family, xc2vp30 device, is displayed in Table I, while the comparison between Quartus II, and Precision, for Stratix II family, EP2S15484 device, is discussed in Table II.

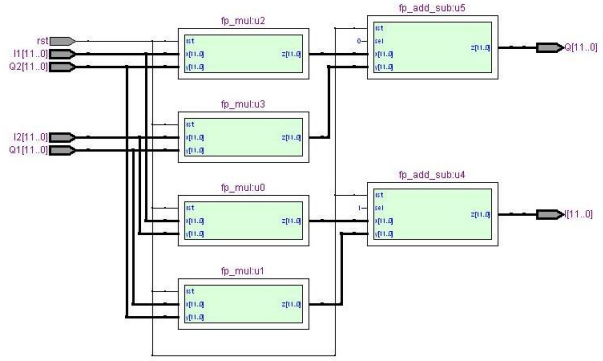

Figure 8. Complex multiplication circuit.

Table I
COMPARISON BETWEEN ISE AND PRECISION.

ISE 10.1
Function Generators 1031 out of 27392 —- 3.76%
CLB Slices 494 out of 13696 —- 3.61%
Block Multipliers 4 out of 136 —- 2.94%
IOs 75 out of 556 —- 13.49%
Precision
Function Generators 1092 out of 27392 —- 3.99%
CLB Slices 546 out of 13696 —- 3.99%
Block Multipliers 4 out of 136 —- 2.94%
IOs 75 out of 556 —- 13.49%

Table II
COMPARISON BETWEEN QUARTUS II AND PRECISION.

Quartus II
Combination LUT 669 out of 12480 —- 5.36%
DSP block 9-bit elems: 4 out of 96 —- 4.17%
IOs 75 out of 343 —- 21.87%
Precision
Combination LUT 822 out of 12480 —- 6.59%
DSP block 9-bit elems: 4 out of 96 —- 4.17%
IOs 75 out of 343 —- 21.87%

From tables, ISE 10.1 and Quartus II are slightly better than Precision EDA synthesis tool. In ISE tool, 3.76% resource utilization is achieved from the number of used LUTs, while in Precision the resource utilization is 3.99%. Therefore, the circuit takes a lower hardware area by ISE EDA tool.
By the same way, Quartus II gives a lower hardware area than Precision does to implement the circuit.

VI. SIMULATION RESULTS

Post synthesis simulation is done to show the effect of the delay timing constraints for the designed circuit. In this simulation, an example is taken to add, substract, and multiply two complex floating point numbers (X and Y) and the result will be Z.
X= 1.2 + j5.1
Y= 3.7 - j4.3

The floating point representation of the two complex values (X and Y) is expounded in Table III. Therefore, the inputs values of the designed circuit "I1, I2, Q1,Q2" are 1.2, 3.7, 5.1, and 4.3 respectively in floating formats. While the input "sel" has value of "00" to add X and Y, and the value "01" for subtraction, and "10" for multiplication process. The output Z is rested to zeros when the input "rst" is activated high.

The simulation is done using "Modelsim SE" EDA tool which belongs Mentor Graphics company [11]. The results from simulation after applying the inputs are shown in Figure 9. From Quartus II, the worst case for time delay is

Table III
THE FLOATING POINT REPRESENTATION OF X AND Y.

	X		Y	
representation	Real	Floating point	Real	Floating point
I	1.2	001111001100	3.7	010000110110
Q	5.1	010001010001	-4.3	110001000100

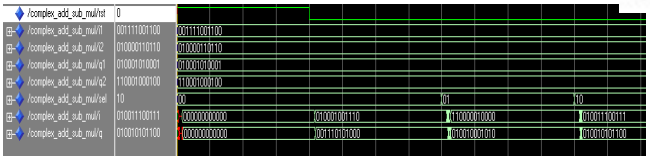

Figure 9. Post synthesis simulation results.

Table IV
RESULTS ANALYZATION.

operation	correct answer	designed circuit output
X+Y	4.9+j0.8	4.875+j0.8125
X-Y	-2.5+j9.4	-2.5+j9.25
X×Y	26.37+j13.71	25.75+j13.5

33.316 ns. Therefore, the frequency achieved by this circuit is about 30 MHz. The results analyzation are discussed in Table IV. The output values are obtained after converting the floating point representation to real representation.

From Table IV, the designed circuit behaves good in all operations. The circuit can be enhanced to improve the output value if a high precision is required. This can be done by using more bits in the mantissa (fraction) field of the floating point representation.

VII. CONCLUSIONS

Reprogrammable systems have provided significant performance improvements for many types of applications. A circuit has been designed to add, substract, and multiply two complex floating numbers. The floating point representation was done using 12 bits with 6 bits precision. Comparison was done between three synthesize EDA tools to choose the best one. Timing simulations were made to show the effect of gate delay. The designed circuit behaved well in all operations. If a high precision is required, more bits in the fraction field should be added.

REFERENCES

[1] S. Brown and J. Rose, "Fpga and cpld architectures: A tutorial," *IEEE Design and Test of Computers*, vol. 13, no. 2, pp. 42–57, 1996.
[2] J. Rose, A. E. Gamal, and A. S. Vincentelli, "Architecture of field-programmable gate arrays," *Proceedings of the IEEE*, vol. 81, no. 7, pp. 1013–1029, July 1993.
[3] S. Hauck, "The roles of fpgas in reprogrammable systems," *Proceedings of the IEEE*, vol. 86, Issue 4, pp. 615–638, 1998.
[4] C. M. Maxfield, *The Design Warrior's Guide to FPGAs*. Elsevier, 2004.
[5] J. Rose, R. J. Francis, D. Lewis, and P. Chow, "Architecture of programmable gate arrays: The effect of logic block functionality on area efficiency," *IEEE Journal of Solid State Circuits*, pp. 1217 – 1225, Oct. 1990.
[6] V. Betz and J. Rose, "Cluster-based logic blocks for fpgas: Area-efficiency vs. input sharing and size," in *IEEE Custom Integrated Circuits conf.*, 1997, pp. 551 - 554.
[7] G. Govindu, L. Zhuo, S. Choi, and V. Prasanna, "Analysis of high-performance floating-point arithmetic on fpgas," *Parallel and Distributed Processing Symposium*, pp. 149–156, April 2004.
[8] N. Shirazi, A. Walters, and P. Athanas, "Quantitative analysis of floating point arithmetic on fpga based custom computing machines," in *IEEE Symposium on Field Programmable Custom Computing Machines (FCCM 95)*, April 1995.
[9] http://www.xilinx.com.
[10] http://www.altera.com.
[11] http://www.mentor.com.