FPGA Implementation of Multiplier for Floating-Point Numbers Based on IEEE 754-2008 Standard

M. Shyamsi, M. I. Ibrahimy, S. M. A. Motakabber and M. R. Ahsan
Dept. of Electrical and Computer Engineering
International Islamic University Malaysia
Kuala Lumpur, Malaysia

***Abstract*—This paper illustrates designing and implementation process of floating point multiplier on Field Programmable Gate Array (FPGA). Floating-point operations are used in many fields like, digital signal processing, digital image processing, multimedia data analysis, etc. Implementation of floating-point multiplication is handy and easy for high level language. However, it is a challenging task to implement a floating-point multiplication in hardware level or in low level language due to the complexity of algorithms. A top-down approach has been applied for the prototyping of IEEE 754-2008 standard floating-point multiplier module using Verilog Hardware Description Language (HDL). Electronic Design Automation (EDA) tool of Altera Quartus II has been used for floating-point multiplier. The hardware implementation has been done by downloading the Verilog code onto Altera DE2 FPGA development board and found a satisfactory performance.**

***Keywords*—Floating-point; Verilog HDL; Carry Look Ahead Adder; Ripple Carry Adder; FPGA**

I. Introduction

Many application areas and scientific problems such as modern radar system, communication system, digital signal processing, image and video processing, etc. require floating-point arithmetic with a high degree of precision and real time operation. Before the 1980, the floating-point operations were largely emulated by software due to the high cost of hardware equipment. Recent development of technological advancement reduces the cost of hardware considerably. Thus, the requirements of developing high-speed, high precision and real-time hardware based floating-point multipliers arise. The development of the Very Large Scale Integration (VLSI) technology brings out reconfigurable and reusable logic devices like FPGAs. The physical FPGA devices become the best candidate for hardware implementation of floating-point multipliers because of its high integration density, low price, high performance, portability and flexibility in design.

The implementation of floating-point multiplier in FPGAs is not a new research in FPGAs application. An ample number of research works have been found related to hardware implementation of floating-point. However, the differences in techniques/algorithms, arithmetic, type of floating-point, the multiplier block can be observed to achieve the best trade off in the area and performance. Digit-Serial Floating-point Multiplier has been presented by Louca et. al., [1]. They have used the IEEE 754 standard for binary floating point arithmetic (single precision) to implement a floating-point multiplier on Altera FLEX 8000 series FPGAs. The authors have commented that floating-point operations are hard to implement on FPGAs due to the complexity of algorithms. In 2009, pipeline architecture based hardware implementation of floating point has been designed by Renxi et. al., [2]. A five level of pipeline scheme and Booth's encoder have been utilized for the implementation. A modified partial product compressor along with a carry look-ahead adder is also included in their structure for requirement of the high speed in the design. They have proposed this pipeline technique to ensure floating-point multiplier running stably at a frequency of 80MHz. Another notable work associated with single/double precision floating-point multiplier is proposed by Ozbilen and Mustafa [3]. The idea of this literature is the proposed floating-point multiplier can perform a double precision floating-point multiplication or two simultaneous single precision floating point multiplications. Since two results are generated in parallel in single precision floating-point multiplication, the multiplier's performance is almost doubled compared to a conventional floating point multiplier. From the synthesis results that were achieved, it showed that their proposed design is 10% larger than conventional multiplier and critical path increment is only one or two gate delay. Since modern floating-point multiplier designs have significantly larger area than the standard floating-point multiplier, the percentage of the extra hardware is less for those units.

The core problem in this research is the complicated implementation of floating-point multiplier in FPGAs. This is due to the complexity of their algorithms. It requires an excessive chip area and a resource which is always limited in FPGAs. This problem becomes even difficult if 32-bit floating point operations are required [1]. This research work has been done with IEEE-754 standard for floating-point arithmetic in order to design floating-point multiplier. The physical FPGA device has been chosen as a realization platform for the implementation of floating-point multiplier. After designing the multiplier model according to the algorithm, it has been coded in Verilog HDL using Altera Quartus II EDA tool. The synthesis of the designed model is performed before applying to the physical FPGA device. Finally, this Verilog code has been downloaded into FPGAs and tested by assigning pins in the Altera DE2 board.

II. Floating-Point Numbers and IEEE 754-2008 Standard

The scientific notation based floating-point number system is capable of representing very large and very small numbers without an increase in the number of bits and it is also capable of representing numbers that have both integer and fractional components [4]. A floating-point number (also known as a real number) consist of two parts and a sign. A simple example of floating point and its binary representation is given in Fig. 1 [5]. The IEEE 754-2008 standard [6] for floating-point numbers has been established to provide a method for computation that yields the same result whether the processing is done in hardware, software, or a combination of the two. Error, and error conditions in the mathematical processing are reported in a consistent manner regardless of implementation.

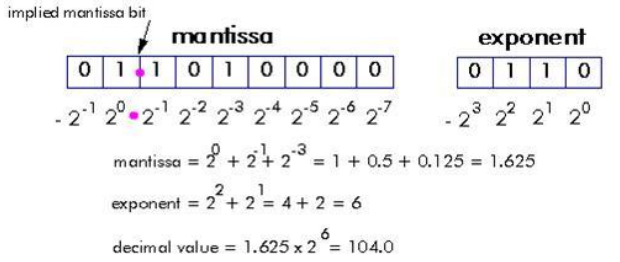

Fig. 1. Example of BORES signal processing [5]

The IEEE standard defines five basic formats which consist of three binary floating-point basic formats (which can be encoded using 32, 64, or 128 bits) and two decimal floating-point basic formats (which can encode using 64 or 128 bits). Following Table 1 shows the differences of these five basic formats.

TABLE I. Parameters Defining Basic Format of Floating-Point Numbers

Parameter	Binary Format (b=2)			Decimal Format (b=10)	
	Binary 32	*Binary 64*	*Binary 128*	*Decimal 64*	*Decimal 128*
p*	24	53	113	16	34
emax**	+127	+1023	+16383	+384	+6144

* *p,* number of significant digit
** *emax,* exponent parameter

In order to build smaller and faster implementations, single precision floating-point format are used. However, this format will result in less accurate in the calculation. The single precision format is shown in Fig. 2.

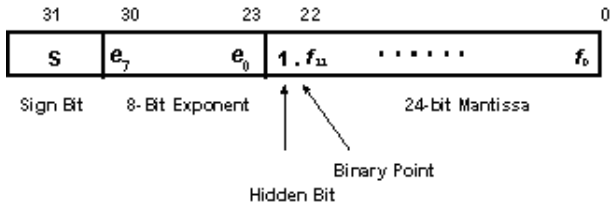

Fig. 2. IEEE floating-point format

Numbers in this format are composed of the following three fields:

1. 1-bit sign, S: A value of '1' indicates that the number is negative, and a '0' indicates a positive number.
2. Bias-127 exponent, e = E + bias: This gives us an exponent range from Emin = -126 to Emax = 127.
3. Fraction, f: The fractional part of the number.
4. Hidden bit: A bit that is typically not store in the fraction field as it is always 1 for all nonzero floating-point normalized numbers [8]. Hidden bit will be taken into account when doing operations such as multiplication of binary.

III. Methodology

The floating - point multiplier is hard to be implemented on FPGAs due to the ramification of their algorithms. Other than complexes of arithmetic or algorithms, it is also difficult to implement owing to the limitation of FPGAs in the area-speed trade-off. The key solution in improving the floating multiplier is its algorithms. Algorithms or arithmetic, determine how much area is needed to yield the required speed. This section states IEEE standard floating point multiplier design which is based on digit-serial floating point multiplier.

A. Floating-point Multiplier Main Block

The main block of floating-point multiplier was inspired from digital-serial floating point multiplier by Louca et. al., [1]. But this design has been modified by changing digit-serial multiplier with add-shift multiplier. Furthermore, 8-bit hierarchical adder has been used instead of 8-bit adder /subtractor.

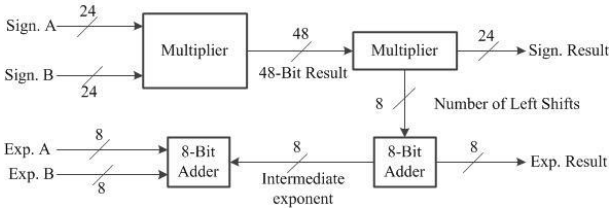

Fig. 3. Floating-point multiplier

Fig. 3 shows the main structure of floating-point multiplier. It composes a 24-bit multiplier, two 8-bit adders and a normalized unit. The sign result is not shown in the main block because it is yielded by XORing two sign input. Fraction A (Sign A) and fraction B (Sign B) yields 48-bit result. A normalized unit normalizes the 48-bit result of multiplication into 24-bit as fraction result (Sign. Result). There are two 8-bit adder to complete the floating point multiplier design. One of these adder adds two external exponents (Exp. A and Exp. B) while another adder adds result of two exponent (Intermediate Exponent) with a number of left shift that come from normalized unit. The last 8-bit adder returns a true result of the exponential addition (Exp. Result).

B. Shift-add Multiplier Block

Shift-Add Multiplier is influenced by the traditional method of multiplying two binary numbers. This approach utilizes shifter in combination with adder [7]. Fig. 4(a) illustrates the manual process of multiplying two binary numbers. The product is yielded by a series of addition operations. For each bit 'i' in the multiplier that is 1, it adds to the value of the multiplicand shifted to the left 'i' times. This algorithm has been simplified as pseudo-code and illustrated in Fig. 4(b).

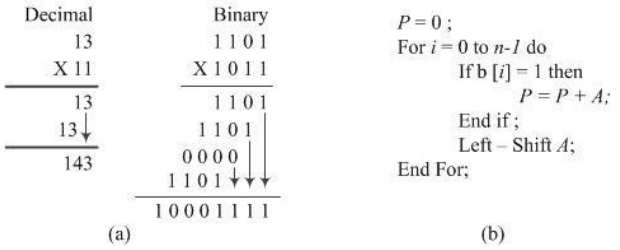

Fig. 4. (a) Handy method for multiplication (b) pseudo-code of multiplier

C. Hierarchical 8-bit Adder Block

As mentioned earlier, there are two adders which are performed for floating point multiplier. One of these Adders produces intermediate exponent output and another one yields exponent result for floating point multiplier. 8-bit Adder is designed by utilized two 4-bit Carry look-ahead adder (CLA) as basic components (Fig. 5). CLA improves the speed by reducing the amount of time required to determine the carry bits. It is different than ripple carry adder where the carry bit is calculated along with the sum bit; and before starting the calculation of its own result and carry, each bit must wait for the previous carry to be calculated. The CLA calculates one or more carry bits before the sum. This reduces the wait time for calculating the result of the large numbers of bits. Thus, to build 8-bit Adder, hierarchical model by Lee8 is employed to combine two 4-bit CLAs.

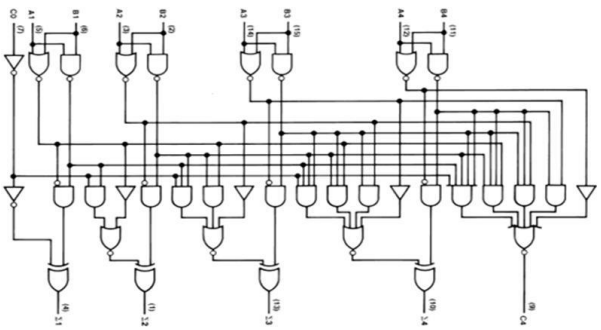

Fig. 5. 4-bit carry look-ahead adder

Fig. 6 shows 8-bit hierarchical Adder. Adder4 is 4 bit CLA. Input is divided into 4-bit for each input (A and B in this case). These 4 bit divided inputs are inserted to 4-bit CLA. To make the connection between first adder and second adder, chain of second CLA is connected to Cout of first CLA. Thus, the output of the Adder (SUM [7:0]) is a combination of two sum outputs of two CLAs. Because it wants to generate for biased 127-exponent, the result of adder subtracts by 127-based component. A normalize unit is also employed whose function is to convert 48-bit fraction output into 24-bit fraction for the floating point multiplier. It also produces a number of shifts left in 127-biased form.

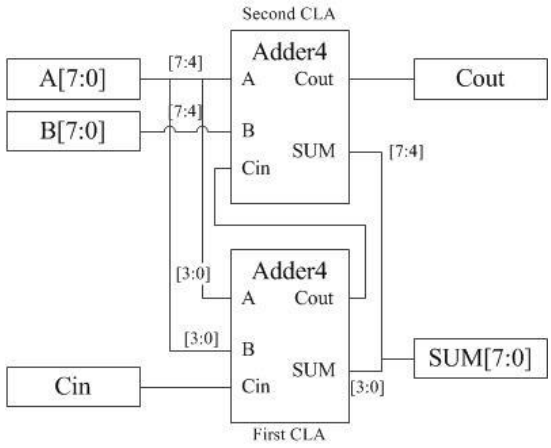

Fig. 6. 8-bit hierarchical adder

D. Analysis of the Design

The Design Entry comprises of the description of the design using graphical tools and/or high level languages. The Synthesis process involves with the conversion of this description into a device independent RTL netlist. The Implementation process generates a suitable circuit description from the RTL netlist for device programming. The correctness of the design representation can be verified through validating each one of these steps. The Design Entry process involves capture of schematics and Verilog HDL coding in this case. Verilog HDL is a hardware description language used for the designing and documenting electronic systems. Verilog HDL permits the designers to design the electronic system at various levels of abstraction. From the main structure of floating-point multiplier (Fig. 3), 6 modules of Verilog design have been coded which are fpmul.v, multiplier.v, normalize.v, hadder.v, cla.v and s7d.v. The floating point multiplier module will be performed in a top-down approach.

E. Normalization

Normalize module is used to perform normalization on 48-bit product of add-shift multiplier to yield a 24-bit fraction. It begins with initializing the base with 1'b01111111 (12710). Bias is used later to add with '0' or '1' according to two conditions.

Condition 1: If the MSB of 48-bit product of add-shift multiplier is '1', thus shift is equal to 1 + bias. For example, 1.100 x 1.100 = 10.010000. To normalize 10.010000, dot will be shifted to the left by 1 thus it yields 1.0010000 x 21. Thus 1 will be added to the exponent in second adder. After that, the product is not required to shift to the left for normalization.

Condition 2: If the MSB of 48-bit product of add-shift multiplier is '0', thus shift is equal to 0 + bias. For example, 1.100 x 1.000 = 01.100000. In this case, dot will not be shifted to the left to perform normalization, since first bit 1 is already in left after the dot. After that, the product is required to be shifted to the left so that bit is in MSB position. This will result in 1.1000000. Finally, normalized product will be gained from

concatenate operation. The remaining unused bit will be removed from p.

IV. Results and Discussion

The simulation and hardware implementation have been done using electronic design automation (EDA) tools. Altera Quartus II, Sypnopsys, and ModelSim are used as EDA tools to design logic circuits, simulate and implement into the Altera DE2 board. To deal with the complexity in algorithm, modular programming method applied. This involves with the partitioning of the circuit into smaller blocks according to their operations and then designing each block separately. Breaking down a large and complex task into convenient and manageable smaller part is known as the top-down design approach. The floating point multiplier design is divided into some sub modules. Fig. 7 shows the RTL view of the top module of floating point multiplier that consist of 3 sub modules (header, multiplier and normalize module). A separate module s7d is used for sending the output to 7-segment display. The subsequent figures (Fig. 8 to Fig. 11) illustrate the RTL diagram of each module. Table 2 shows the summary of the analysis and synthesis resource usage of the top module.

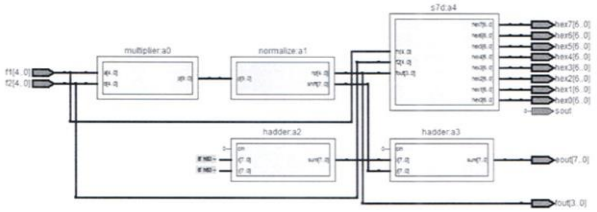

Fig. 7. RTL view of the top module of floating point multiplier with s7d module

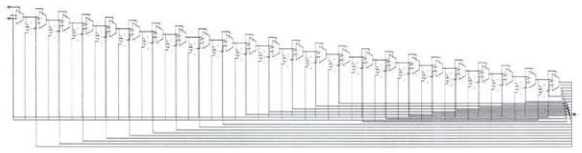

Fig. 8. RTL view multiplier module

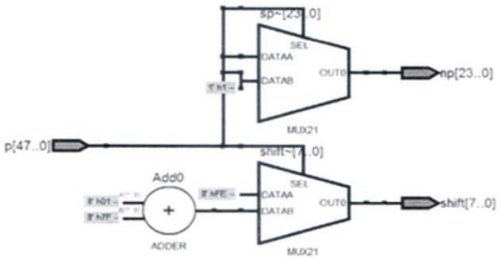

Fig. 9. RTL view normalize module

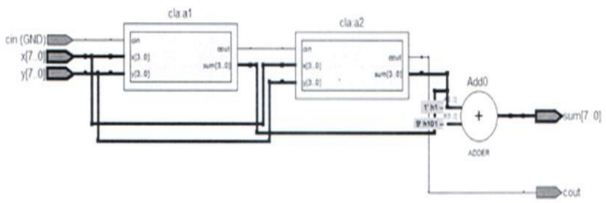

Fig. 10. RTL view adder module

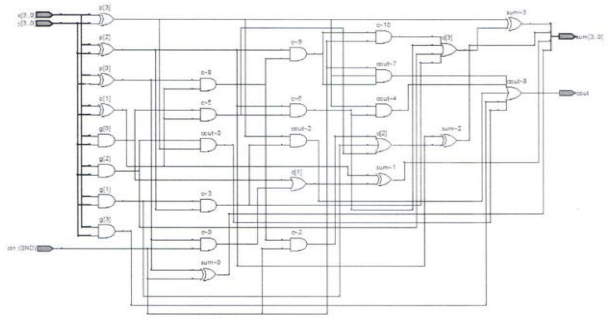

Fig. 11. RTL view CLA module

TABLE II. Top Module Analysis and Synthesis Resource Usage Summary

Resource	**Usage**
Estimated total logic elements	1199
Total combinational functions Logic element usage by number of LUT inputs 4 input functions 3 input functions <= 2 input functions	1199 62 1027 110
Logic elements by mode Normal mode Arithmetic mode	 634 565
Total registers Dedicated registers I/O registers	0 0 0
I/O pins Maximum fan-out node Maximum fan-out Total fan-out Average fan-out	96 f2[0] 45 3532 2.73

The Functional simulation of the designed multiplier model has been done using Altera ModelSim and Sypnopsys. Two multiplication tests have been performed for this top module of floating point multiplier. The tested parameters for the floating point multiplier are as follows.

First Test
s1=1'b0;
s2=1'b1;
e1=8'b10000100;
e2=8'b00111100;
f1=23'b01000000000000000000000;
f2=23'b11000000000000000000000;

Second Test
s1=1'b0;
s2=1'b1;
e1=8'b10000100;
e2=8'b00111100;
f1=23'b00100000000000000000000;
f2=23'b01000000000000000000000;

For the first test, the result for the sign is 1 since sign 1 and sign 2 is '0' and '1' respectively. First, hierarchical adder adds 132 and 60 and subtract with 127-based component to yield 66. Then this input goes to second hierarchical adder and add with 128 inputs from normalize module. After that, it is subtracted by 127 to gain exponent value of 65 of floating point multiplier. Fraction output is 23'b00011000000000000000000 which is yielded from 1.00011000000000000000000.

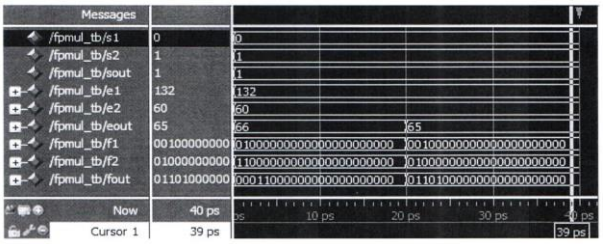

Fig. 12. fpmul testbench using Altera ModelSim

For the second test, the result for the sign is 1 since sign 1 and sign 2 is '0' and '1' respectively. First, hierarchical adder will add 132 and 60 and subtracts with 127-based component to yield 65. Then this input will go to second hierarchical adder and add with 127 inputs from normalize module since it satisfies the second condition. After that, it is subtracted by 127 to gain exponent value of 65 of floating point multiplier. Fraction output is 23110110010000000000000000 which is yielded from 1.01101000000000000000000. The simulation output for the first test is presented in Fig. 12 (using Altera ModelSim) and Fig. 13 (using Synopsys).

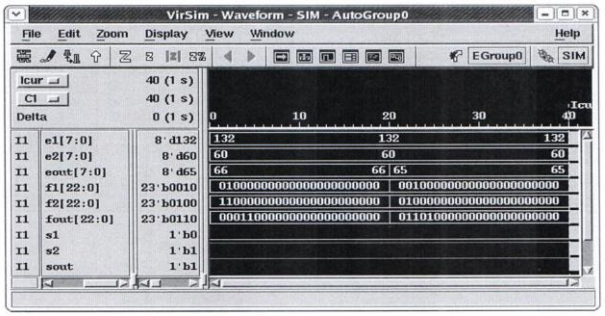

Fig. 13. fpmul testbench using Synopsys

After successful verification of functional simulation, the timing simulation has been done using Altera Quartus II EDA tool. Fig. 14 shows the timing simulation of the floating point multiplier. The timing simulation is different from functional simulation as some glitches appear. The glitch occurs because the output has to be performed for a specific period of time. Nevertheless, this small of glitch gives small effect of delay to output of the simulation.

The functional and timing simulation indicates that the designed multiplier model has fulfilled all the requirements of the specification. The model is then finally implemented onto the physical FPGA chip. All I/O pins of the modules are assigned as it is required to specify which pin to use for which type of the signal in a circuit. The code is implemented and downloaded into the FPGA on Altera DE2 board.

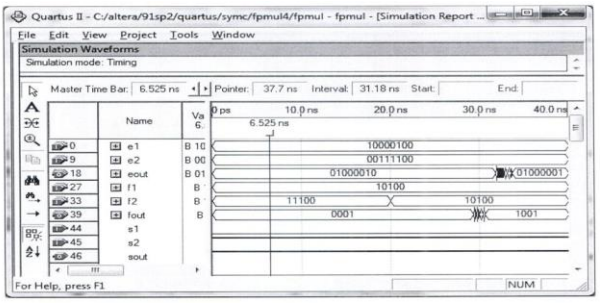

Fig. 14. Timing simulation of floating-point multiplier

Figure 15 represents the top view of the pin configuration for FPGA chip Cyclone II-EP2C35F672C6 which has total 672 pins. Fig. 16 shows the Altera DE2 board with the output for the following test inputs.

28 = 1.1100 x 24
20 = 1.0100 x 24
Mathematically 1.1100 x 24 x 1.0100 x 24 = 1.000110000 x 29

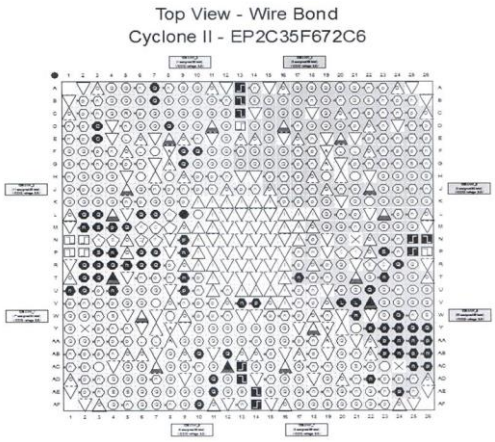

Fig. 15. Pin configuration for the FPGA implementation onto the Altera DE2 board

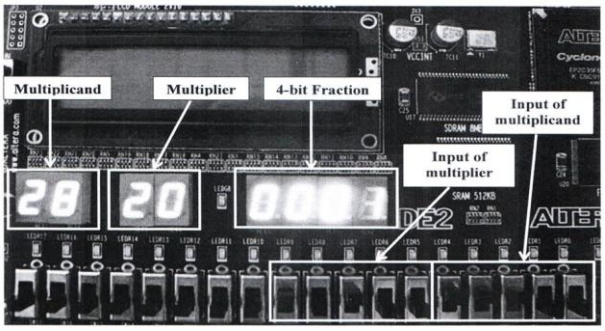

Fig. 16. FPGA implementation of IEEE standard multiplier on Altera DE2 board

V. Conclusion

The FPGA based hardware realization of IEEE standard floating-point multiplier has been successfully implemented. The synthesis result shows that a total of 1199 logic cells is utilized in Cyclone-II FPGA with a speed of 50 MHz and 96 numbers of pin assignment. During this work, a comprehensive study on the IEEE floating-point standard has been carried out.

The IEEE 754-2008 standard ensures that the computation of floating-point numbers yields an identical result, whether the processing has been done with hardware, software or with a combination of both. The multiplication algorithm using IEEE standard has been implemented in Verilog-HDL and verification have been done through simulation and synthesis. After validating the simulation result of the floating point multiplier, FPGA based hardware realization has been successfully performed on Alter DE2 development board. The future work may include increasing the number of input bits (binary 64 and binary 128) and implementing different multiplier algorithms through comparing their performances.

REFERENCES

[1] L. Louca, T. A. Cook, and W. H. Johnson, "Implementation of IEEE single precision floating point addition and multiplication on FPGAs," IEEE Symp. on FPGAs for Custom Computing Machines, Napa Valley, CA, pp. 107-116, 1996.

[2] G. Renxi, Z. Shangjun, Z. Hainan, M. Xiaobi, G. Wenying, and X. Lingling, "Hardware Implementation of a high speed floating point multiplier based on FPGA," 4th Int. Conf. on Computer Sci. & Edu., Nanning, China, pp. 1902–1906, 2009.

[3] M. M. ÖZBİLEN, and G. A. Mustafa. Single/Double Precision Floating-Point Multiplier Design for Multimedia Applications. J of E. & E Eng. vol. 9. Istanbul University, 2009, pp. 827–831.

[4] T. L. Floyd, Digital Fundamentals. Upper Saddle River-NJ: Prentice Hall, 2003.

[5] Introduction to DSP – DSP Processors: data formats, Retrieve from http://www.bores.com/courses/intro/chips/6_data.htm.

[6] IEEE Standard for Floating-Point Arithmetic, IEEE Standard 754-2008, pp. 1-58.

[7] S. Brown and Z. Vranesic, Fundamentals of Digital Logic with Verilog Design, New York : McGraw-Hill Sci./Eng./Math, 2007.

[8] J. M. Lee, Verilog Quickstart: A Practical Guide to Simulation and Synthesis in Verilog. NY: Springer-Verlag, 2002.