Scientific Journal Impact Factor (SJIF) : 1.711
International Journal of Modern Trends in Engineering and Research
ISSN (ONLINE) : 2349 - 9745
ISSN (PRINT) : 2393 - 8161

Computation of Floating Point Operations and Functions in FPGA

Sruthi.R.S
Department of Electronics and Communication
Mohandas College of Engineering and Technology

Abstract —Floating point describes a method of representing an approximation of a real number in a way that can support a wide range of values. Many scientific applications require floating point arithmetic because of high accuracy in their calculations. FPGAs are becoming more suitable for supporting high speed floating point arithmetic units. Floating point units are widely used in digital applications such as digital signal processing, digital image processing and multimedia. Many algorithms depend on floating point arithmetic because floating point representation supports wide range. This paper proposes a floating point unit with Arithmetic and Trigonometric operations for floating point numbers based on IEEE 754 standard and to enhance the performance of the system. The arithmetic unit allows various arithmetic operations such as, Addition, Subtraction, Multiplication and Division on floating point numbers. It is synthesized and simulated on the Virtex-6 FPGA board using Xilinx ISE Design Suite 14.5 and Modelsim SE 6.5b.

Keywords– *FPGA, Floating point, IEEE754 standard, Virtex-6*

I. INTRODUCTION

The floating point operations have found intensive applications in the various fields for the requirements for high precious operation due to its great dynamic range, high precision and easy operation rules. High attention has been paid on the design and research of the floating point processing units. With the increasing requirements for the floating point operations for the high-speed data signal processing and the scientific operation, the requirements for the high-speed hardware floating point arithmetic units have become more and more exigent. The implementation of the floating point arithmetic has been very easy and convenient in the floating point high level languages, but the implementation of the arithmetic by hardware has been very difficult. With the development of the very large scale integration (VLSI) technology, a kind of devices like Field Programmable Gate Arrays (FPGAs) have become the best options for implementing floating hardware arithmetic units because of their high integration density, low price, high performance and flexible applications requirements for high precious operation. The recent advancements in the area of Field Programmable Gate Array (FPGAs) has provided many useful techniques and tools for the development of dedicated and reconfigurable hardware employing complex digital circuits at the chip level. Therefore, FPGA technology can be gainfully utilized in order to develop digital circuits so that the problem of floating-point representation of numbers and the computational resources required while performing the arithmetic and logical operations during execution of the algorithm could be solved at the hardware level. Floating point describes a method of representing an approximation of a real number in a Way that can support a wide range of values. The numbers are in general, represented approximately to a fixed number of significant digits (the mantissa) and scaled using an exponent. Floating point operations are very complex operations because they require many algorithms. Floating point units are widely used in digital applications such as digital signal processing, digital image processing and multimedia. Digital arithmetic operations are very important in the design of digital processors and application specific systems. Many of the algorithms used in DSP and matrix arithmetic require elementary functions such as trigonometric, inverse trigonometric, logarithm, exponential, multiplication, and division functions. Often trigonometric functions are used

in embedded applications. Arithmetic circuits form an important class of circuits in digital systems. With the remarkable progress in the Very Large Scale Integration (VLSI) circuit technology, many complex circuits, unthinkable yesterday have become easily realizable today. Algorithms that seemed impossible to implement now have attractive implementation possibilities for the future. This means that not only the conventional computer arithmetic methods, but also the unconventional ones are worth investigation in new designs. In Conventional floating point units, the most frequently used floating point operations are addition/subtraction counting for more than 94% of all floating point instructions. Hence the employment of highly performing divider, multiplier, adder and subtractor modules is of high importance. The floating point unit is one of the most important custom applications needed in most hardware designs, as it adds accuracy, robustness to quantization errors and ease of use. This paper includes the different floating point operations like Addition, Subtraction, Multiplication and Division into a single unit. Also including a Trigonometric unit for the floating point numbers. There by increasing the flexibility and performance of the system.

II. FLOATING-POINT REPRESENTATION

The IEEE754 standard floating-point format consists of three fields—a sign bit, a biased exponent, and a mantissa. Single-precision numbers have a 1-bit sign, 8-bit exponent, and 23-bit mantissa as shown in Fig. 1(a). Double-precision numbers have a 1-bit sign, 11-bit exponent, and 52-bit mantissa as shown in Fig. 1(b).

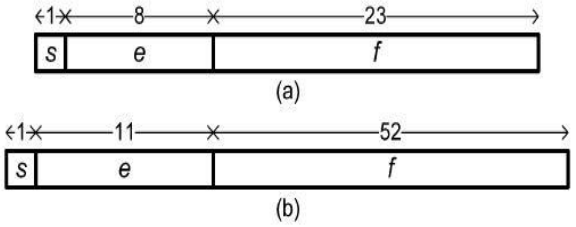

Fig. 1. (a) Single-precision and (b) double-precision IEEE754 floating-point numbers.

There is an implied "1" to the left of the binary point (except in the special case of denormal numbers)[20]

$$X = (-1)^s \times (1:f) \times (2^{e-127}) \quad (1)$$

$$X = (-1)^s \times (1:f) \times (2^{e-1023}) \quad (2)$$

Floating-point numbers have an advantage of being able to cover a much larger dynamic range compared to fixed-point numbers. The disadvantage is that floating-point computations are much more complex to implement in hardware.

III. FLOATING POINT ARITHMETIC

The floating point unit consist of Arithmetic unit and Trigonometric unit. The arithmetic operation consists up of Addition, Subtraction, Multiplication and Division. The proposed block diagram for the floating point unit is given in figure 2.

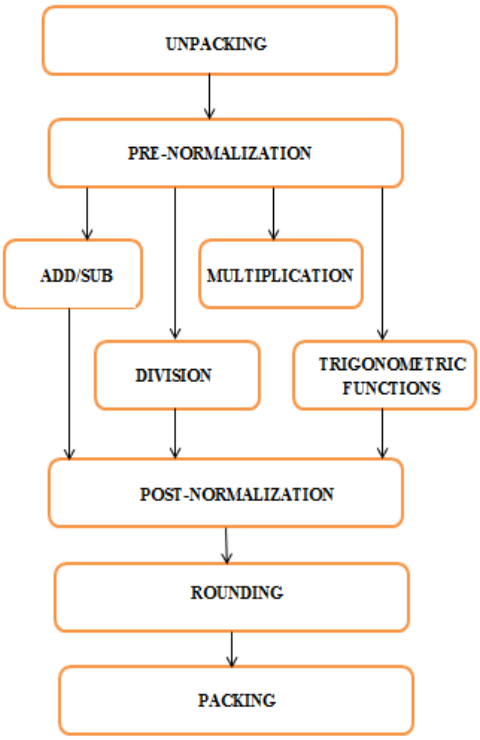

Fig.2: Block diagram of Floating point unit

All arithmetic operations have these five stages:
1. Unpacking
2. Pre-normalize: the operands are transformed into formats that makes them easy and efficient to handle internally.
3. Arithmetic core: the basic arithmetic operations are done here.
4. Post-normalize: the result will be normalized if possible (leading bit before decimal point will be 1, if normalized) and then transformed into the format specified by the IEEE standard.
5. Rounding: The IEEE standard specifies four rounding modes round to nearest, round to zero, round to positive infinity, and round to negative infinity. Table 1 shows the rounding modes selected for various bit combinations of rounding mode[2]. Based on the rounding changes to the mantissa corresponding changes has to be made in the exponent part also.

TABLE 1: ROUNDING MODES SELECTED FOR VARIOUS BIT COMBINATIONS OF ROUNDING MODE

Bit combination	Rounding Mode
00	round_nearest_even
01	round_to_zero
10	round_up
11	round_down

A) Floating Point Addition/Subtraction

The conventional floating-point addition algorithm consists of five stages exponent difference, pre alignment, addition, normalization and rounding [6]. Given floating point numbers X1 and X2, X1 = (s1, e1, f1) and X2 = (s2, e2, f2), X1+X2, the stages for computing are described as follows. The equations are given by,

Two floating point numbers are added as:

- $(F1\times2^{E1}) + (F2\times2^{E2}) = F\times2^{E}$ (3)

Where E=E1+E2

Two floating point numbers are subtracted as:
$(F1\times2^{E1}) - (F2\times2^{E2}) = F\times2^{E}$ (4)
Where E=E1-E2
The floating point addition/subtraction algorithm is as follows:

1) Find exponent difference d= e1-e2. If e1< e2, swap position of mantissas. Set larger exponent as tentative exponent of result.
2) Pre-align mantissas by shifting smaller mantissa right by bits.
3) Add or subtract mantissas to get tentative result for mantissa.
4) Normalization. If there are leading-zeros in the tentative result, shift result left and decrement exponent by the number of leading zeros. If tentative result overflows, shift right and increment exponent by 1 bit.
5) Round mantissa result. If it overflows due to rounding, shift right and increment exponent by 1 bit.

B) Floating Point Multiplication
Algorithmically, floating-point multiplication is much simpler than floating-point addition. However, a very wide integer multiplier is required. Given floating-point numbers X1 & X2, X1 = (s1, e1, f1) and X2 = (s2, e2, f2), X1 × X2, can be computed using

$S_p = s1 \text{ XOR } s2$ (5)
$e_p = e1+e2 – bias$ (6)
$1.fp = 1.f1 \times 1.f2$ (7)

Only the main parts of the data path are shown for clarity. If the result from the multiplier has two bits left of the binary point, the mantissa has to be shifted right to compensate and the exponent is incremented. If the rounding of the mantissa results in an overflow, the mantissa is shifted right by one and the exponent is incremented [9]. The floating point multiplication algorithm is as follows :
1) Multiply the mantissas, M1 x M2.
2) Placing the decimal point in the result.
3) Adding the exponent.
4) Obtaining the sign bit, S1 XOR S2.
5) Normalizing the result.
6) Rounding the result.
7) Check underflow or overflow.

C) Floating Point Division
Algorithmically, floating point multiplication is much simpler than floating point addition. The floating point division algorithm is similar to that of the floating point multiplier algorithm [3][16]. Given floating point numbers X1 & X2, X1 = (s1, e1, f1) and X2 = (s2, e2, f2), X1 / X2, can be computed using:
S= s1 XOR s2 (8)
e = e1-e2+bias (9)
f= f1 / f2 (10)
The floating point division algorithm is as follows
1) Divide the mantissas, f1 / f2

2) Placing the decimal point in the result.
3) Subtracting the exponent.
4) Obtaining the sign bit, s1 XOR s2.
5) Normalizing the result.
6) Rounding the result.
7) Check underflow or overflow.

IV. FLOATING POINT TRIGONOMETRIC FUNCTIONS

In this version all the trigonometric modules are created as look up table (LUT). To all the input values there is an equivalent double precision floating point unit value, to the input an un-signed value is given. The input port bits can be configured to any number of bits. It also supports all quadrants i.e. sine and cosecant is positive in first and second quadrants and negative in third and fourth quadrants, tangent and cotangent are positive in first and third quadrants and negative in second and fourth quadrants, cosine and secant are positive in first and fourth quadrants and negative in second and third quadrants(Fig 3).

If the value of the degrees is greater than 360 then the value is passed into the divider circuit, in this circuit the total value is divided by the value 360 and returns the remainder value that is less than 360, then the value is given as degrees to the top module and checks for the positive or negative quadrants. "ACTV" value decides which block to activate and passes the corresponding input value to that block and the output value is the same value as in the LUT or changed according to the corresponding quadrants.

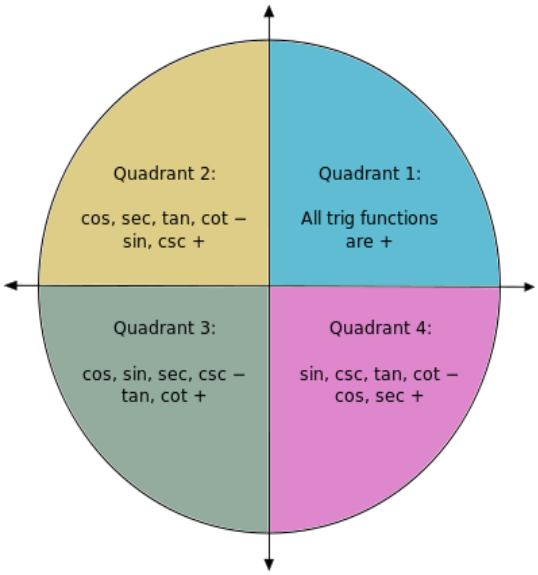

Fig.3: Trigonometric quadrants

This architecture is created using look up table. But instead of creating the table for all the values it is created only for the first 90 values and all the remaining values are derived using these 90 values. From the top module using "ACTV" Input we can choose which function to activate.

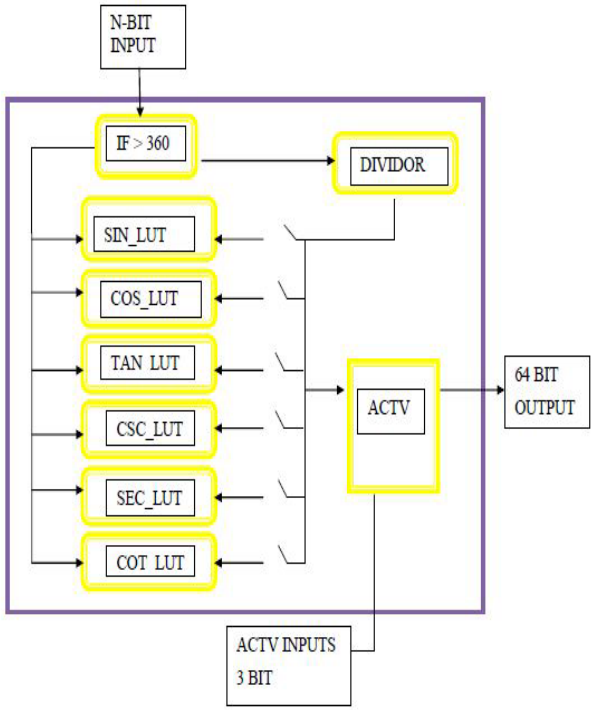

Fig.4: Block diagram of Trigonometric unit

V. SIMULATION RESULTS

The arithmetic and trigonometric blocks in the floating point unit explained has been coded in HDL and simulated using modelsim simulator. The simulation results for all operations performed by the unit are shown below.

a) Addition

Opa=90=0100000001010110100000000000000000000000 0000000000000000000000
Opb=0.07=0011111110110001111010111000010100011110 1011100001010001111011100
Opa+Opb=rounded_out=90.07=0100000001010110100001000111101011100001010001111010111
0 00010100

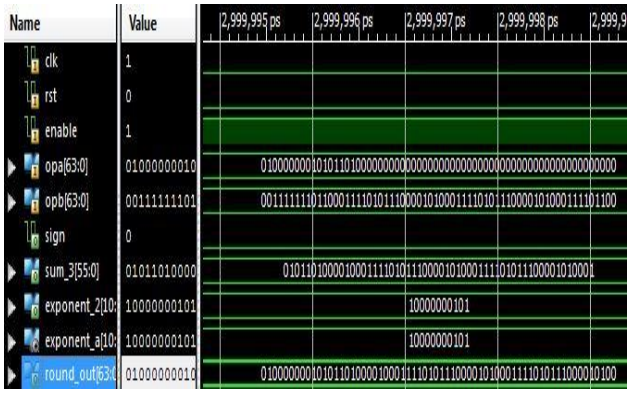

Fig.5: Simulation results of double precision floating point addition

b) Subtraction

Opa=90=0100000001010110100000000000000000000000 0000000000000000000000
Opb=0.07=0011111110110001111010111000010100011110 1011100001010001111011100
Opa+Opb=rounded_out=89.93=0100000000010100000000000000000000000000000000000000000
000000000

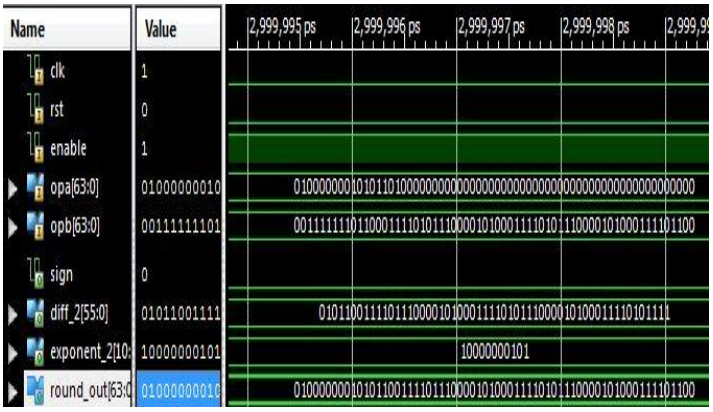

Fig.6: Simulation results of double precision floating point subtraction

c) Multiplication

Opa=5=0100000000010100000000000000000000000000000000000000000000000000

Opb=4=0100000000010000000000000000000000000000000000000000000000000000

Opa×Opb=rounded_out=20=0100000000110100000000000000000000000000000000000000000000000000

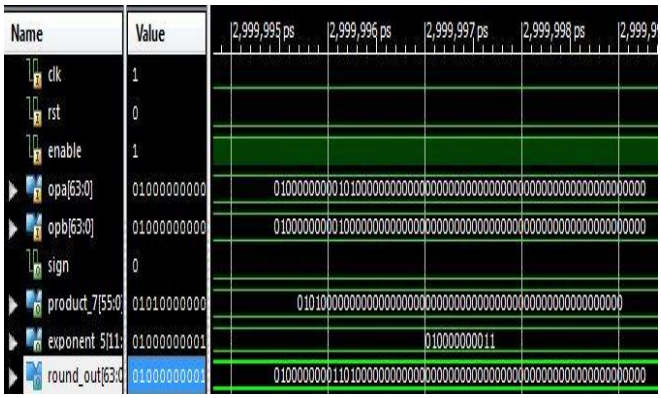

Fig.7: Simulation results of double precision floating point Multiplication

d) Division

Opa=20=0100000000110100000000000000000000000000000000000000000000000000

Opb=5=0100000000010100000000000000000000000000000000000000000000000000

Opa/Opb=rounded_out=4=0100000000010000000000000000000000000000000000000000000000000000

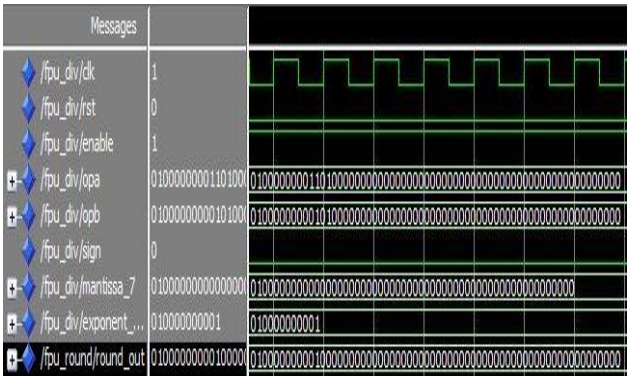

Fig.8: Simulation results of double precision floating point Division

e) Trigonometric functions

i. Sine

Input: degree=120
Output: data_sin=3febb67ae8584caa=0.866025

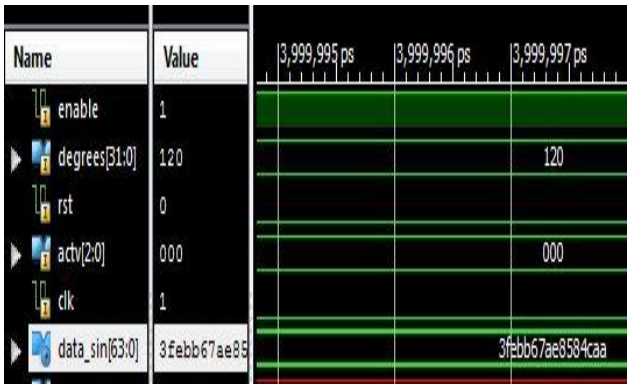

Fig.9: Simulation results of double precision floating point Sine function

ii. Cosine
Input: degree=60
Output: data_cos=3fe0000000000001=0.5

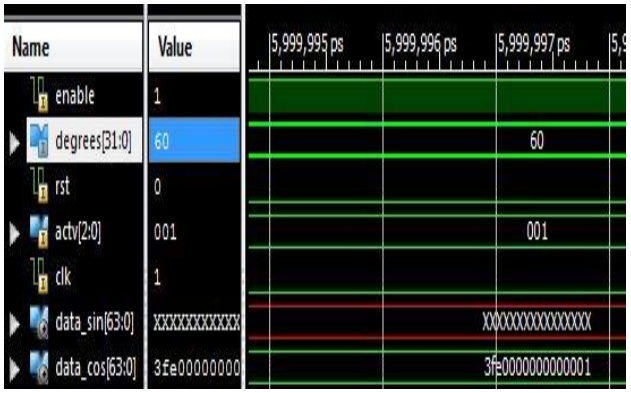

Fig.10: Simulation results of double precision floating point Cosine function

iii. Tangent
Input: degree=60
Output: data_tan=3ffbb67ae8584ca8=1.73205

Fig.11: Simulation results of double precision floating point Tangent function

iv. Cosecant
Input: degree=60
Output: data_csc=3ff279a74590331d=1.1547

Fig.12: Simulation results of double precision floating point Cosecant function

v. Secant

Input: degree=200

Output: data_sec=bff106df459ea072= -1.0641

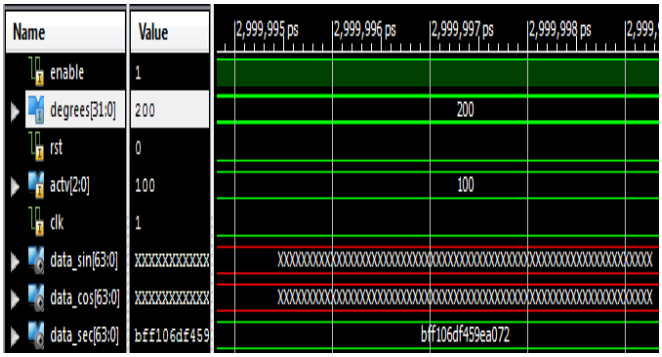

Fig.13: Simulation results of double precision floating point Secant function

vi. Cotangent

Input: degree=300

Output: data_cot=bfe279a74590331e= -0.5735

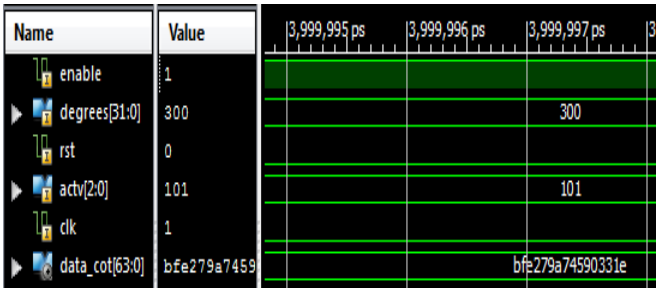

Fig.14: Simulation results of double precision floating point Cotangent function

TABLE 2 DEVICE UTILIZATION AND TIMING SUMMARY

Device Utilization Summary		
	Previous work[1]	Proposed work
Number of slice registers	4710	4341
Number of Slice LUTs	7681	6568
Timing Analysis		
	Previous work[1]	Proposed work
Minimum Period(ns)	9.392	3.526
Maximum frequency(MHz)	106.474	283.599

Table 2 shows the comparison of device utilization summary and timing summary reports of the proposed work with a previous work [1] on Virtex-6 FPGA. The results shows a better device utilization in the proposed work and have a minimum period of 3.526ns and maximum frequency of 283.599MHz which shows an increase in performance of the floating point arithmetic unit compared with the previous work.

VI. CONCLUSION

This paper presents a Double precision floating point arithmetic and trigonometric unit in FPGA. The arithmetic operations include addition, subtraction, multiplication and division. The main advantage of this FPU is that it can compute different trigonometric functions that supports double precision IEEE754 standard floating point format. The whole unit is synthesised on the Virtex 6 FPGA board using Xilinx ISE Design Suite 14.5 and simulated using Isim and Modelsim SE 6.5b.

REFERENCES

[1] Chaitanya A. Kshirsagar, P.M. Palsodkar,' An FPGA Implementation of IEEE - 754 Double Precision Floating Point Unit Using Verilog', International Journal of Electrical, Electronics and Data Communication,Volume-2, Issue-6,pp 684-688, June-2014
[2] Nikhil S. S Sheela Devi Aswathy Chandran,'Floating Point Unit (FPU) for FPGA', IJSRD - International Journal for Scientific Research and Development, Vol. 2, Issue 01, 2014 .
[3] Jitendra Soni, Ravi Mohan. ECE, SRIT, Jabalpur, India' Floating Point Single Precision Division in VHDL Environment', International Journal of Emerging Trends in Electronics and Computer Science (IJETECS) Volume 2,Issue 10,Oct.2013.
[4] Riya Saini, Galani Tina G. and R.D. Daruwala, 'Efficient Implementation of Pipelined Double Precision Floating Point Unit on FPGA',International Journal of Emerging Trends in Electrical and Electronics (IJETEE – ISSN: 2320-9569) Vol. 5, Issue. 1, July-2013.
[5] Yedukondala Rao Veeranki, R. Nakkeeran, 'Spartan 3E Synthesizable FPGA Based Floating-Point Arithmetic Unit', International Journal of Computer Trends and Technology (IJCTT),volume4, Issue4,April 2013.
[6] Yee Jern Chong and Sri Parameswaram, 'Configurable Multimode Embedded units floating-point for FPGAs', IEEE Transactions on VLSI systems, pp. 2033-2044,Vol.19, No.11, November 2012.
[7] ChiWai Yu, Alastair M. Smith, Wayne Luk, et.al,'Optimizing Floating Point Units in Hybrid FPGAs', IEEE Trans On VLSI Systems, Vol. 20, No. 7, July 2012.
[8] Shikha Khurana, Kanika Kaur,'Implementation of ALU using FPGA', International Journal of Emerging Trends and Technology in Computer Science (IJETTCS) Volume 1, Issue 2, July-August 2012
[9] Mohamed Al-Asrafy, Asraf Salem, WagdyAnis,'An Efficient Implementation of Floating Point Multiplier', Saudi International Electronics, Communications and Photonics Conference (SIECPC), pp. 1-5, 24-26 April 2011.
[10]Chun Hok Ho, ChiWai Yu, Philip Leong, Senior Member, IEEE, et.al,'Floating-Point FPGA: Architecture and Modeling', IEEE Transactions On Very Large Scale Integration (VLSI) Systems, Vol. 17, No. 12, December 2009
[11]Michael J. Beauchamp, Scott Hauck, Keith D. Underwood, and K. Scott Hemmert,'Architectural Modifications to Enhance the Floating-Point Performance of FPGAs', IEEE Transactions On Very Large Scale Integration (VLSI) Systems,Vol. 16, No. 2, February 2008
[12]J. Detrey and F. de Dinechin., 'Floating-point trigonometric functions for FPGA', In Field-Programmable Logic and Applications,pages29-IEEE, 2007.
[13]M.J.Beauchamp, S.Hauck, and K.S.Hemmert,'Embedded floating-point units in FPGAs',in Proc. IEEE Symp.Field Program. GateArrays(FPGA), 2006,pp.12-20.
[14]P. C. Diniz and G. Govindu,'Design of a field-programmable dual-precision floating-point arithmetic unit', in Proc. Int. Conf. Field Program. Logic Appl.(FPL), 2006, pp. 1-4.
[15]P. M. Seidel, G. Even, 'Delay-Optimization Implementation of IEEE Floating Point Addition', IEEE Trans on computers, pp. 97-113, vol. 53, no. 2 February 2004.
[16]S. Paschalakis and P. Lee,'Double precision Floating Point Arithmetic on FPGAs', Proc. of IEEE Conference on Field Programmable Technology, 2003, pp. 352-358.
[17]J. Liang, R. Tessier and O. Mencer, 'Floating Point Unit Generation and Evaluation for FPGAs', IEEE Symp. On Field-Programmable Custom Computing Machines, pp. 185-194, April 2003.
[18]J. Bhasker, 'A VHDL primer', Third edition, Pearson education, 1999.
[19]Dr. S. Ramachandran,'Digital VLSI Systems Design',Hanbook published by Springer.
[20]IEEE Standard for Binary Floating-Point Arithmetic, ANSI/IEEE Std 754, 1985.