Hardware Implementation of Single Precision Floating Point Arithmetic Unit

Noorjabeen[1], Vikas Pathak[1], Rahul Vijay[2]

[1] Department of Electronics & Communication Engineering, Swami Keshvanand Institute of Technology, Management and Gramothan, Jaipur, India

[2] Department of Electronics & Communication Engineering, Banasthali Vidyapith, Jaipur, India

Email: noorkhan984914@gmail.com, vikas.pathak@skit.ac.in, vijay.rahul1986@gmail.com

Received 10.09.2022, received in revised form 24.09.2022, accepted 12.10.2022

DOI: ***10.47904/IJSKIT.12.2.2030.40-45***

Abstract: This research paper represents the floating-point arithmetic calculations. In this work floating point arithmetic unit multiplication and addition have been performed. In this manuscript, IEEE 754 standard of 32-bit single precision floating point arithmetic is used. Floating-point arithmetic is useful for high dynamic range arithmetic but is more resource intensive than integer arithmetic. Floating point calculations occur on system has a wide range of values that require fast processing times. This research paper represents the redesign of floating-point arithmetic unit. It deals with the ALU similar structure which performs addition/subtraction and multiplication with the selection line. If the selection line is '1' then it performs addition / subtraction otherwise for selection line '0' multiplication of floating-point numbers is done. The proposed design is simulated using Xilinx Isim Simulator and synthesized and impleneted using Xilinx ISE on Nexys-4 DDR FPGA device.

Keyboard - Floating point Arithmetic, hardware implementation, Single precision, FPGA, ALU

1. INTRODUCTION

Designing digital processors and application systems is crucial for numerical calculations. In digital systems, arithmetic circuits play a significant role. Many complex circuits have evolved as a result of thorough development of very large-scale integration (VLSI), and they are now simple to build [1-2].

In arithmmetic circuits (like digital filters, Digital signal processors) many times we have to operate on real numbers. For representing real numbers in binary form two optins are available: Fixed point and floating-point arithmetic. Due to the advantages (like its high precision, wide range, and simple regulations) floating point technique is particularly valuable over fixed point. However, arithmetic units have been around for many years, and their common representation over the past decade has been the IEEE standard for floating point arithmetic [IEEE 754-2008]. In addition to their standard implementations, floating-point operators need additional space (or time). The system designer takes into account the throughput limitation and area for floating point representation [3-4].

A method of representing numbers is referred to as floating point either too large or too insufficient to be an integer. Resolution can be maintained with floating point representation and accuracy Unlike Fixed Point Representation [5]. Many challenging fields are discovered using floating point arithmetic applications. High research and design emphasis are paid on floating point processors units [6]. Floating point equivalents are unquestionably a precise trade-off as compared to using fewer data bits to describe the fixed point. It is recommended to choose an architecture that can execute arithmetic operations directly on complex numbers represented using a 32-bit subset of the IEEE floating-point format in order to avoid this compromise and minimise the number of lookup tables obtained [7]. The term floating point indicates that the radixpoint or decimal point is floating and can be positioned at any place.

The floating-point format for hardware implementations of floating-point units depends on variables like: Maximum accessible arithmetic error, dynamic range requirements, power consumption, and more [8]. Due to its floating point is used for many applications as Neural Network Efficiency via Post-Training Quantization with Adaptive Floating-Point [9].

This paper presents an FPGA implementation of single precision of floating-point arithmetic unit. The floating-point arithmetic implentation have been done in hardware description language (VHDL) and are based on the IEEE-754 standard.

In section 2, floating point algorithm section describes the basic floating-point format for addition /subtraction and multiplication with the steps involved for the algorithm are discussed briefly. Section 3 is the proposed design of floating-point arithmetic unit where we discuss in detail using the flow diagram about the design, we have implemented using VHDL. Section 4 presents the simulation section, in which the simulation result of Floating-point adder/subtractor, multiplier and ALU waveform generated through Simulator is shown. Here synthesis result of the ALU is also shown.

Finally, the manuscript is concluded, and future scope research directions are discussed in section 5.

2. FLOATING POINT ALGORITHM

Real numbers can be represented in binary format in a variety of ways, like fixed point and floating-point numbers. But floating-point arithmetic has several advantages like high precision and wider range compared to fixed point arithmetic. This paper focuses on IEEE 754 single precision binary format. The IEEE 754 [10] standard represents floating point numbers in two forms single precision and double precision binary format. Its representation is shown in Table 1. In this manuscript, the focus is on single precision floating point format due to its requirement of lesser hardware resources. It consists of sign bit, exponent and Mantissa. 32-bit Single Precision Floating-Point Numbers IEEE standard have one sign bit (S), eight bit exponent (E) and Mantissa (M) have twenty-three-bit fraction [11].

Table 1: IEEE 754 floating point binary format representation

TYPE	SIGN	EXPONENT	MANTISSA
32-bit single precision	1bit	8 bit	23bit fraction
64 bit-double precision	1bit	11bit	52bit fraction

The algorithms for floating point addition/ substraction and multiplication have been described in this section, which become the base for drawing the architecture and then writing the VHDL code for implementation of 32-bit floating point arithmetic unit.

2.1 Steps for Floating point addition

For floating point addition, the exponent should be same. So if they are not same we have to make them same. Further it can be explained in two cases might take place when two floating point numbers are added .

Case 1: - When both numbers are of same sign

Step 1: - Two number X and Y having Sx, Sy as their sign bit and M_x, M_y, and E_x, E_y as their Mantissa and exponent respectively.
Step 2: - If E_x or E_y = ‘0’. If yes; the hidden bit of X and Y is 0. If not, then $E_y > E_x$ is checked if yes X and Y are swapped and if $E_x > E_y$; then the content of X and Y are not swapped.
Step 3: - The difference in exponent is calculated d = E_x - E_y. If d = ‘0’ then the sifting of significand is not done. But if the value of d is more than ‘0’ than Y is assigned with ‘a’ and then shifted to the right by an amount ‘a’ and load left bit by ‘0’. The hidden bit is also included in shifting.
Step 4: - Exponent of Y is added with the ‘a’. Now the new value of Ey is equal to the addition of previous Y and ‘a’ and after addition X becomes equal to Y.
Step 5: - X and Y signed are checked, if it doesn’t match, then switch to step 6.
Step 6: - Add significant of X and Y as well as hidden bit.
Step 7: - 1 is added to the exponent value of both the equivalent exponent either X or new Y, if carry is generated in addition of significant. After addition, the result of significand is shifted towards right by one.
Step 8: - Check if there is no carry in step 6, then previous exponent is the real exponent.
Step 9: - Sign is taken as MSB of either X or Y.
Step 10: - Concatenate the sign bit, exponent value and mantissa into 32-bit format excluding 24th bit of significand with is the hidden bit.

Case 2: - When both numbers are of different sign.

Steps 1, 2, 3 and **4** will remain same as in the above case [12].
Step 5: - Sign of X and Y are examined whether they have different sign or not, if ‘yes’.
Step 6: - Take 2’s compliment of S_y, which is then added to S_x ($S = S_x$ + 2’s compliment of S_y).
Step 7: - Significant is added if there is generation of carry bit. If yes; then generated carry will be discarded and the result is also sifted towards left till there is ‘1’ in the SB and the amount of shifting is counted which is denoted by ‘b’.
Step 8: - Then ‘b’ is subtracted from exponent value. Actual exponent is calculated by subtraction of ‘b’ from exponent of first number.
Step 9: - If there is no carry in step 6 then MSB = 1 and in that case, S is replaced by 2’s complement.
Step 10: - Taking sign of larger number sign of final result is computed.
Step 11: - Concatenate the result into 32-bit format excluding 24th bit of significand (hidden bit).

2.2 Steps for Floating Point Multiplication

Assume A and B are the numbers taken for multiplication. It consists of sign bit (S_a, S_b), mantissa (M_a, M_b) and exponent (E_a, E_b) [6]. Multiplication of Floating point is performed through following steps:
$M = M_a * M_b$
$E = E_a + E_b - Bias$
$S = S_a$ XOR S_b

- ➢ Calculation of Exponent: The exponent is added to both binary adders. The input and offset are subtracted from the estimates [13]. The result received as the intermediate exponent.

- Calculation of significant: Both unsigned signifant are multiplied and the decimal point is inserted in the multiplication product [14]. The result received is known as transitional product.
- Calculation of sign bit: The sign bit depends on the numbers. If one of the numbers have negative, then result of multiplication is negative which can be achieved by Xoring the sign of two inputs.

3. PROPOSED ARCHITECTURE OF FLOATING-POINT ARITHMETIC UNIT

The proposed architecture for floating point arithmetic unit is discussed below. It contains the flow diagram which describes the various components (which can be further used for writing the VHDL code) for adder / subtractor and multiplier blocks of floating point aritmetic unit.

3.1 Architecture of floating-point Adder / subtractor

It is well known that floating point numbers have three main components i.e., sign bit, mantissa and exponent. For two number X and Y, S_x, S_y are considered as their sign bit, whereas M_x, M_y, and E_x, E_y are there Mantissa and exponent respectively.

First of all, we check for sign bit which is S_x and S_y and then exoring is done. For the mantissa part the M_x, M_y are compared in mantissa comparator, from where we get M_x-gt-M_y. The exponent of both the number E_y and E_x are compared in Exponet comparator and from where we have two output E_x equal to E_y (E_x-eq-E_y) and E_x greater than E_y (E_x-gt-E_y). In next step, the denormalization which is addition of 1 bit to the exponent and mantissa is done. And we get greater exponent (g-e) and lower exponent (l-e). Then for mantissa we get greater mantissa (g-m) and lower mantissa (l-m). Exponent portion which is "g-e", and "l-e" goes to subtractor, and we get d - e at outcome which goes to barrel shifter. From right shifter/ barrel shifter we get shift-m which goes to adder/ subtractor block. The sign bit decides for the adder/ subtractor where we have the mantissa g-m and l-m and the shift-m from where we get sum - m as a output. The output of barrel shifter (l-m) and adder/ subtractor (sum - m) is done on normalization i.e., 1 is subtractor from mantissa & exponent using priority Encoder and left shifter is done. And at last, by combing all the three exponents, mantissa and sign we get the output "z" which is the addition of the two floating point numbers.

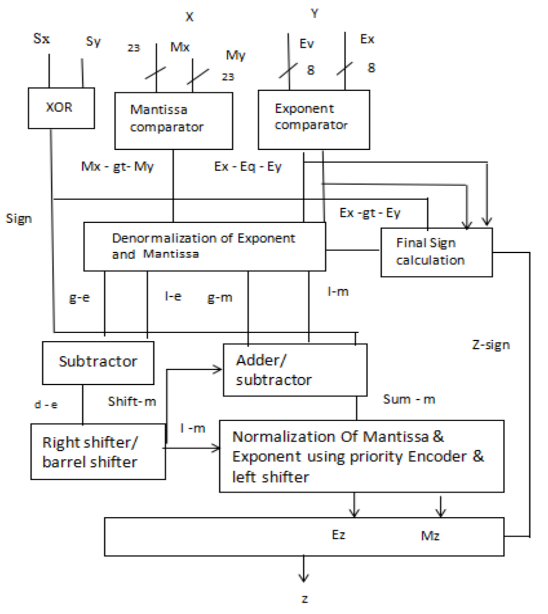


Figure 1: Proposed architecture of floating-point Adder / subtractor**.**

3.2 Proposed architecture of floating-point Multiplier

Assume X and Y are the numbers taken for multiplication. It consists of sign bit (S_x, S_y), mantissa (M_x, M_y) and exponent (E_x, E_y). Multiplication of Floating-point numbers is explained in following paragraph.
First of all, the sign bit of both the number is checked. If both are not same, then Xoring is done. Then 23-bits mantissas M_x and M_y are denormalized (addition of 1) and send to 24 bit multiplier, which results in 47th bit output. Then 47th bit of multiplied mantissa output is sent to adder as a flag bit. The exponent of both the number E_x and E_y (8 bit) is added along with the flag. When the exponents are added, the 127 value (inherit compoenent of exponent) is added two times. So, to get the exact exponent 127 value is subtracted from the output of the exponent adder. Starting 23-bits (select either 46-24 (for flag = 1) or 45-23 (for flag = 0) depending in 47th bit) are selected from multiplier output for getting the exact normalized mantissa output. The mantissa (normalized output) (p-m) received from 24- bit multiplier and the output from the subtractor which is the exponent portion and also the sign bit after Xoring is combined together to form the product of two floating point number.

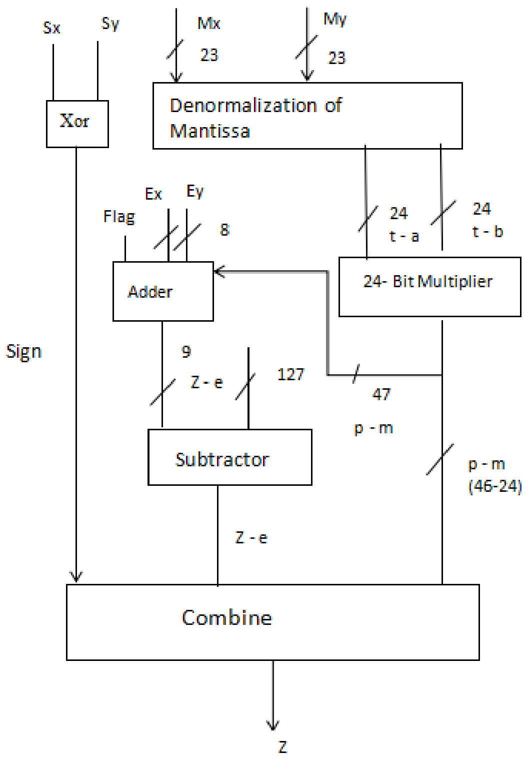


Figure 2: Proposed architecture of floating-point multiplier

3.3 Combined architecture of floating-point adder/subtractor and multiplier

Figure 3 shows the propsed combined architecture of floating-point adder/ subtractor and multiplier which consist of an adder, a multiplier, 2:1 MUX with a selection line. It shows that the proposed architecture will work as adder/Subtractor for selection Input s=1, whereas it will work as multiplier for s=0.

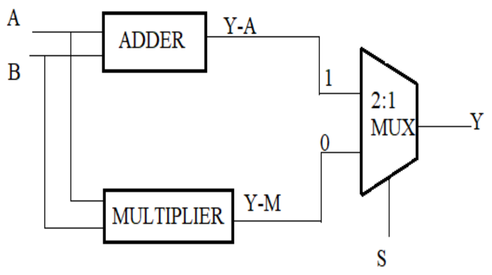
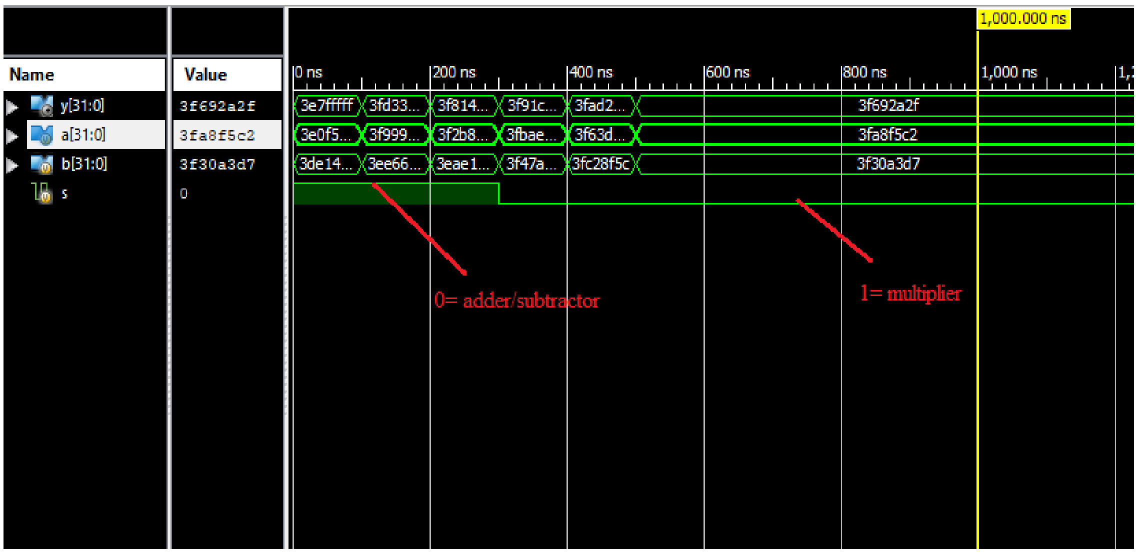


Figure 3: Proposed architecture of combined floating-point architecture of adder/subtractor and multipier

4. RESULT AND DISCUSSION

4.1 Simulation Results

The proposed design is simulated and verified using Xilinx Isim simulator. For this a VHDL test-bench is written. The proposed architecture is verified for both floating point adder/subtractor and multiplier. The timing wave forms (with hexadecimal format) of simulated design is shown in Figure 4. These wave forms clearly indicates that proposed architecture will work as adder/Subtractor for selection input s=1, whereas it will work as multiplier for s=0. For S=0 (Adder/Subtractor), the inputs taken are a=0.14, 1.20, 0.67 and b=0.11, 0.45, 0.34 then we get the output y = 0.25, 1.65, 1.01. Similarly for S=1 (Multiplier), the inputs taken are a=1.46, 0.89, 1.32 and b=0.78, 1.52, 0.69 then we get the output y=2.24, 2.41, 2.01. These result clearlry indicates that we got the similar practical simulation outputs as compared to theoretical values for adder / multipler.



```
Name      Value
y[31:0]   3f692a2f   3e7fffff 3fd33... 3f814... 3f91c... 3fad2... 3f692a2f
a[31:0]   3fa8f5c2   3e0f5... 3f999... 3f2b8... 3fbae... 3f63d... 3fa8f5c2
b[31:0]   3f30a3d7   3de14... 3ee66... 3eae1... 3f47a... 3fc28f5c 3f30a3d7
s         0
```

Figure 4: Simulation Waveform of proposed floating-point arithmetic unit

4.2 Synthesis Results:

The proposed design is syntheised using Xilinx ISE 2014.4 and implemented on Nexys – 4 DDR FPGA (Artix – 7 FPGA Family) Trainer Kit (with actual device named as “xc7a100t-3csg324”). VHDL Language is used for writing the HDL code of proposed deign. The RTL schematic (top and internal view) is shown in Figures 5 and 6 respectively. The top-view clearly indicates the various inputs and outputs of the proposed design. The internal view represents the different internal components like adder, multipler, multiplexer etc. Further if we enhance the multiplier/ adder blocks, then the detailed gate level netlist components of respective block can be visualized.

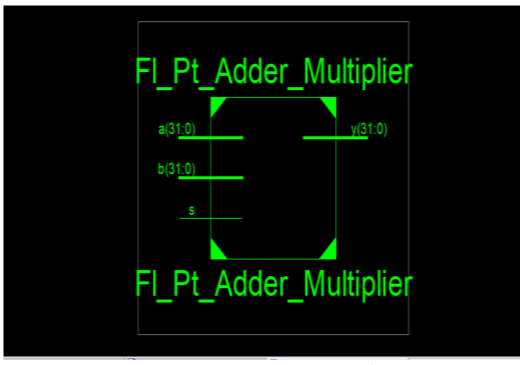


Figure 5: - RTL Schematic (top view) of Floating-Point arithmetic unit

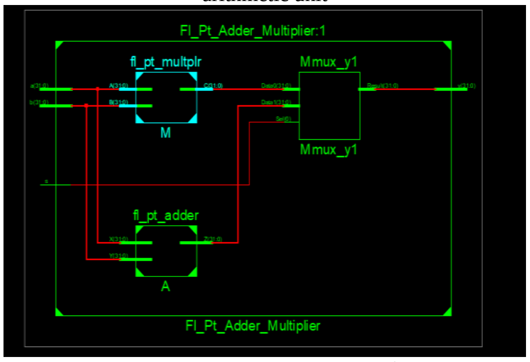
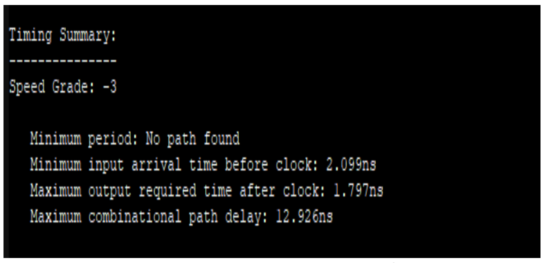


Figure 6: - RTL Schematic (internal view) of Floating-Point arithmetic unit

Table 2 shows the synthesis report, which indicates the hardware units used in floating point arithmetic units. Device Utilization summary is represented in table 3, which indicates that how many components (out of the available FPGA components) are used. Theire percentage utilization are also shown in this table. The synthesis timing summary (as shown in figure 7) indicates the maximum combinational path delay of proposed design is 12.926 ns.



```
Timing Summary:
---------------
Speed Grade: -3

   Minimum period: No path found
   Minimum input arrival time before clock: 2.099ns
   Maximum output required time after clock: 1.797ns
   Maximum combinational path delay: 12.926ns
```

Figure 7: - Synthesis timing summary of Floating-Point arithmetic unit

Table 2: Hardware blocks used in proposed design implemented for Artix-7 FPGA Kit.

S. No.	Device used	No. of Devices Used
1	Multiplier	1
2	Adder/ Subtractor	7
3	Latches	1
4	Comparator	3
5	Multiplexer	47
6	Xors	2

Table 3: Devices utilization summary of proposed design implemented for Artix-7 FPGA Kit.

Logic utilization of devices	Used	Available	Utilization
Slice register	1	126800	0%
Slices LUTs	379	63400	0%
Fully used LUT - FF Pairs	1	379	0%
Bonded IOBs	97	210	46%
Number of DSP48E1s	2	240	0%

5. CONCLUSION

Floating point calculations occur on system has a wide range of values that require fast processing times. In general, it may be assumed that fixed point implementations are faster and less expensive, although floating-point implementations offer a higher dynamic range and don't require scaling, which may be appealing for algorithms with more intricate logic. In this manuscript, an hardware efficient architecture of combined floating point arithmetic unit is proposed. In this hardware unit, Addition, subtraction and multiplication operations are executed according to 32-bit single precision floating point algorithm (IEEE-754 standard). The VHDL code is written for describing the hardware of proposed design. The hardware architecture is synthesized using Xilinx ISE Tool and implemented on Nexys- 4 DDR- FPGA (Artix - 7 family FPGA) Board. Synthesis results indicates that, the propesed design has 12.926 ns of prpopogation delay with efficient use of limited hardwrae resources.

REFERENCE

[1] Muller, J.-M., Elementary Functions: Algorithms and Implementation, 2nd edition, Chapter 10, ISBN 0-8176-4372-9, Birkhäuser, 2006.

[2] Sasidharan, Anjana, and P. Nagarajan. "VHDL Implementation of IEEE 754 floating point unit."International Conference on Information

Communication and Embedded Systems (ICICES2014). IEEE, 2014.

[3] Paschalakis, Stavros, and Peter Lee. "Double precision floating-point arithmetic on FPGAs."Proceedings. 2003 IEEE International Conference on Field-Programmable Technology (FPT) (IEEE Cat. No. 03EX798). IEEE, 2003.

[4] Sunesh, N. V., and P. Sathishkumar. "Design and implementation of fast floating point multiplier unit."2015 International Conference on VLSI Systems, Architecture, Technology and Applications (VLSI-SATA). IEEE, 2015.

[5] Purnima, Shrivastava, et al. "VHDL Environment for Floating point Arithmetic Logic Unit-ALU Design and Simulation. "Research Journal of Engineering Sciences. ISSN 2278 (2012): 9472.

[6] Grover, Naresh, and M. K. Soni. "Design of FPGA based 32-bit Floating Point Arithmetic Unit and verification of its VHDL code using MATLAB."International Journal of Information Engineering & Electronic Business 6.1 (2014).

[7] Rajesh, GOLLA Srinivasulu Mr G., and V. Trimurthulu. "Optimized Design and Implementation of IEEE-754 Floating Point Processor."

[8] Khushbu Naik and Tarun Lad, “Implementation of IEEE 32 Bit Single Precision Floating Point Addition and Subtraction”, International Journal of Computer Application, Volume 5, No.3, pp. 107-111, April 2015

[9] Liu, Fangxin, et al. "Improving neural network efficiency via post-training quantization with adaptive floating-point. "Proceedings of the IEEE/CVF International Conference on Computer Vision. 2021.

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

[11] Grover, Naresh, and M. K. Soni. "Design of FPGA based 32-bit Floating Point Arithmetic Unit and verification of its VHDL code using MATLAB."International Journal of Information Engineering & Electronic Business 6.1 (2014).

[12] Karan Gumber and Sharmelee Thangjam, “Performance Analysis of Floating Point Adderusing VHDL on Reconfigurable Hardware”, International Journal of Computer Applications, Volume 46, No.9, pp. 1-5, May 2012

[13] Paulo S. R. Diniz, Eduardo A.B. da Silva and Sergio L. Netto, “Digital Signal Processing: System Analysis and Design”, ISBN: 9780511781667, Cambridge University Press

[14] Pardeep Sharma and Gurpreet Singh, “Analysing Single Precision Floating Point Multiplieron Virtex 2P Hardware Module”, International Journal of Engineering Research andApplications (IJERA), Vol. 2, Issue 5, pp. 2016-2020, September- October 2012