International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering (IJAREEIE)

| e-ISSN: 2278 – 8875, p-ISSN: 2320 – 3765| www.ijareeie.com | Impact Factor: 7.122|

||Volume 9, Issue 6, June 2020||

VLSI Implementation of Floating Point Hardware Using VHDL

M.Chattaerjee[1], D.Ghosh[2], R.Nayek[3]

Lecturer, Department of ECE, Shree Ramkrishna Institute of Science and Technology, Kolkata, West Bengal, India[1]

PG Student, Department of ECE, RCC Institute of Information Technology, Kolkata, West Bengal, India[2, 3]

ABSTRACT: A floating point CORDIC coprocessor is designed which accepts inputs in the IEEE 754 double precision format. These are internally converted into 128 bit fixed point representation by a preprocessor. The CORDIC module operates in either the rotation or vectoring modes for circular, linear or hyperbolic systems to produce the result. The results after completion of the CORDIC iterations are reconverted into the IEEE 754 format by a postprocessor. Field Programmable Gate Arrays are increasingly being used to design high- end computationally intense microprocessors capable of handling both fixed and floating- point mathematical operations. Addition is the most complex operation in a floating-point unit and offers major delay while taking significant area. The Objective of this paper to implement the 32 bit binary floating point adder with minimum time. Floating point numbers are used in various applications such as medical imaging, radar, telecommunications Etc. Here pipelined architecture is used in order to increase the performance and the design is achieved to increase the operating frequency. This paper discusses in detail the best possible FPGA implementation will act as an important design resource.

KEYWORDS: IEEE 754, Double precision, Floating point unit (FPU), logarithmic approach, Adder, Multiplier

I.INTRODUCTION

In the recent years, there has been a great development in the area of computer systems which are built based on general purpose micro- processors. Arithmetic and Logic Unit (ALU) is a common component in these processors in order to execute all arithmetic operations needed. The speed of an ALU may be enhanced by supporting floating point operations along with fixed point operations which in turn contributes to overall performance of the system. Floating Point Unit (FPU) can act as a co-processor by effectively taking up the intensive tasks from main processor and decreases the overhead on main processor [6]. The main functionalities of FPUs include addition, subtraction, multiplication, division, finding square root etc. Adder is the basic building block of any ALU and other operations like multiplications and divisions are built based on adder block. FPU acts as crucial block in DSP applications like filter design where addition and multiplication operations are called frequently. Floating point numbers are basically a real numbers in binary format [8]. The scientific applications are basically depends on the multimode computation. Multimode based computation are used to avoid underflow by removing multiplication by addition. Recent FPGA have a large number of look up tables (LUTs), registers, hardware multipliers and microprocessors. Field Programmable Gate Array (FPGA) is a silicon chip with unconnected logic blocks, these logic blocks can be defined and redefined by user at any time. FPGAs are increasingly being used for applications which require high numerical stability and accuracy. By using this method the time is save and the method is easier, in current situation, this is unattainable, because within this adder/subtraction, input ought to lean in IEEE 754 format. The floating point arithmetic unit present in the processor can compute addition, subtraction, multiplication, division, as well as square root of the numbers that are represented in IEEE754 Standard single precision floating point format [2]. The complex floating point arithmetic unit present in the processor can compute the addition, subtraction, multiplication on the complex numbers that are represented in a subset of IEEE754 standard format with 16-bits (1-sign bit, 5 exponent bits, 10-mantissa bits) for real part and 16-bits for imaginary part (1-sign bit, 5-exponent bits, 10-mantissa bits) [3]. Digital computer arithmetic is an aspect of logic design with the objective of developing appropriate algorithms in order to achieve an efficient utilization of the available hardware [1-4]. Given That the hardware can only perform a relatively simple and primitive set of Boolean operations, arithmetic operations are based on a hierarchy of operations that are built upon the simple ones. Since ultimately, speed, power and chip area are the most often used measures of the efficiency of an algorithm, there is a strong link between the algorithms and technology used for its implementation.

| e-ISSN: 2278 – 8875, p-ISSN: 2320 – 3765| www.ijareeie.com | Impact Factor: 7.122|

||Volume 9, Issue 6, June 2020||

II.BASICS OF TESTABLE FLOATING POINT ALU

In a computer processor, arithmetic-logic unit (ALU) is a hardware block that does all arithmetic and logic operations based on the instruction code (instruction code tells the operation that needs to be executed and the operands upon which the operation needs to be executed). Typical operations of arithmetic unit (AU) in ALU are addition, subtraction, multiplication and division where as in Logic Unit (LU) are AND, OR, shifting, bitwise operations. In some other processors, ALU may be split into dedicated arithmetic unit (AU) and a logic unit (LU). Different parts of the processor have access to ALU unit to perform necessary operations. The parts of the processor that have access to ALU include memory unit, processor's controller unit (CPU) and input/output devices. Inputs required for the ALU (called operands) are nothing but CPU registers. Similarly, the output of ALU goes again to one of the CPU registers. Any instruction that comes to ALU contains information about 'op code' and 'operands'. 'op code' defines the operations that need to be performed on the 'operands'. Depending on the instruction's 'op code', the number of operands can be one or more. If bitwise operations are being performed, then only one operand is enough but in case of addition and multiplication, two operands are needed. In few cases, 'op code' indicates the operation that needs to be performed, whether a floating point operation or fixed point operation. Based on this information, ALU reads the operands as either floating point or fixed point. Floating-point processing utilizes a format defined in IEEE 754, and is supported by microprocessor architectures. However, the IEEE 754 format is inefficient to implement in hardware, and floating-point processing is not supported in VHDL.

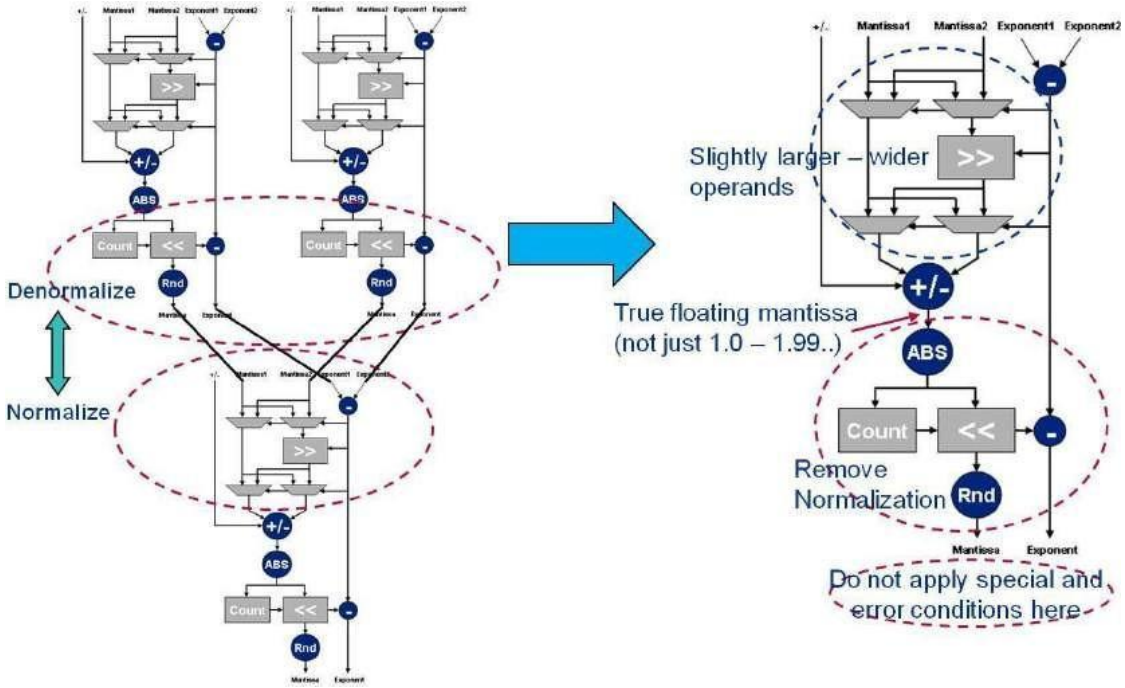


Figure.2 New Floating-Point Approach

These techniques are used to build a high-performance floating-point data path within the FPGA. Because IEEE 754 representation is still necessary to comply with floating-point processing, the floating-point circuits must support this interface at the boundaries of each data path, such as a fast Fourier transform (FFT), a matrix inversion, or sine function. This floating-point approach has been found to yield more accurate results than if IEEE 754 compliance is performed at each operator.

In general, Floating-Point ALU executes the floating point operations needed by different parts of the processor. Semiconductor industry follows IEEE754 standard to represent a floating point number in floating point operations. There are two different floating point number formats

| e-ISSN: 2278 – 8875, p-ISSN: 2320 – 3765| www.ijareeie.com | Impact Factor: 7.122|

||Volume 9, Issue 6, June 2020||

1.Single-precision floating point number format
2.Double-precision floating point number format

According to IEEE754 standard, floating point number has three fields:

1) Sign bit (s)
2) Biased exponent (e)
3) Mantissa (m).

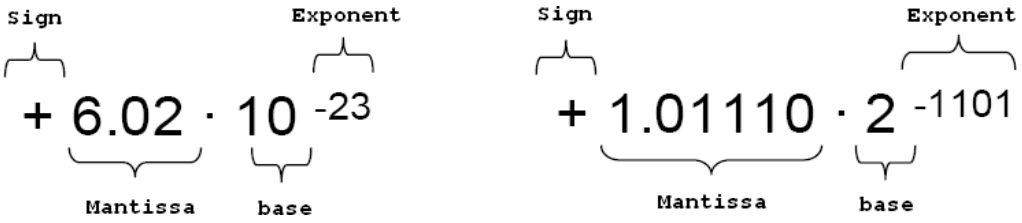


Fig.3 Overview of floating point number

Based on the format, the lengths of these three different fields vary. As shown in Fig 2.1(a), single-precision numbers have 1-bit sign, 8-bit exponent, and 23-bit mantissa. Double- precision numbers have 1-bit sign, 11-bit exponent, and 52-bit mantissa as shown in Fig.
2.1(b). Equations 2.1 and 2.2 represent a floating point number 'X' in single and double precision formats respectively. The symbol {*} represents multiplication here.

$X = (-1)s * 1.m * 2(e-127)$ (1)

$X = (-1)s * 1. m * 2(e-1023)$ (2)

The major advantage of floating-point numbers over fixed-point numbers is that floating- point numbers cover wide range of values. However, the complexity in implementing the floating point operations is more compared to fixed point which is the main disadvantage. Following sections describe floating point addition, multiplication, normalization and rounding off functions in detail.

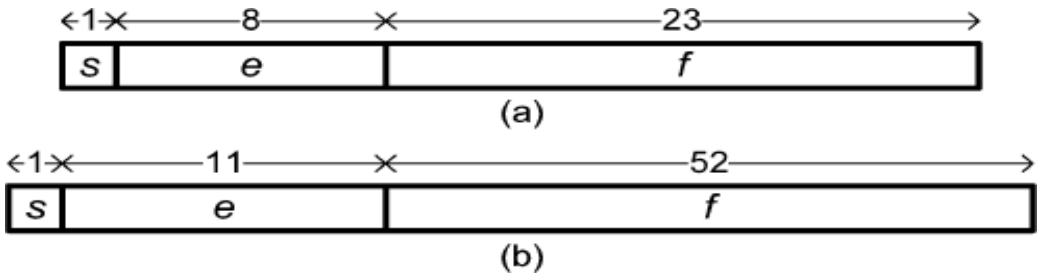


Fig.4 (a) Single-precision and (b) Double-Precision IEEE 754 floating-point numbers

III.FLOATING POINT VERIFICATION

Floating-point results cannot be verified by comparing bit for bit, as is typical in fixed- point arithmetic. The reason is that floating-point operations are not associative, which can be proved easily by writing a program in C or MATLAB to sum up a selection of floating-point numbers. Summing the same set of numbers in the opposite order will result in a few different LSBs. To verify the floating-point designs, the designer must replace the bit-by-bit matching of results typically used in fixed-point data processing with a tolerance-based method that Comparex the hardware results to the simulation model results.

| e-ISSN: 2278 – 8875, p-ISSN: 2320 – 3765| www.ijareeie.com | Impact Factor: 7.122|

||Volume 9, Issue 6, June 2020||

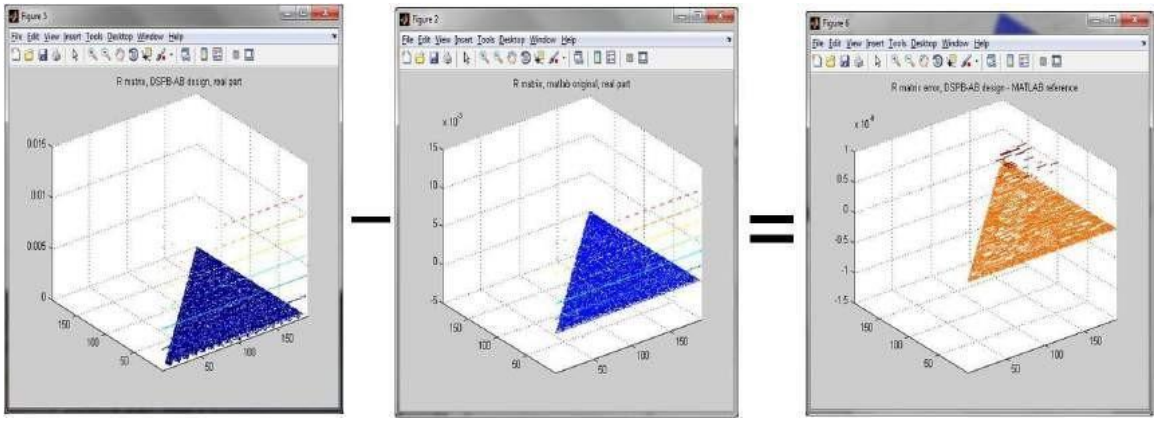


Fig.5 Matrix Error Plot

To verify the accuracy of the non-IEEE 754 approach, matrix inversion was performed using single-precision floating-point processing. These results were also computed using single-precision with an IEEE 754-based Pentium processor. Then a reference result was computed on the processor, using IEEE 754 double-precision floating-point processing, which provides near-perfect results relative to single-precision. Comparing both the IEEE 754 single-precision results and the single- precision hardware results, and computing norm and the differences, shows that the hardware implementation gives a more accurate result than the IEEE 754 approach, due to the extra mantissa precision used in the intermediate calculations.

IV.THE STANDARD IEEE 754

Standard IEEE 754 specifies formats and methods in order to operate with floating point arithmetic. These methods for computational with floating point numbers will yield the same result regardless the processing is done in hardware, software or a combination for the two or the implementation.
The standard specifies:

- Formats for binary and decimal floating point data for computation and data interchange
- Different operations as addition, subtraction, multiplication and other operations
- Conversion between integer-floating point formats and the other way around
- Different properties to be satisfied when rounding numbers during arithmetic and conversions
- Floating point exceptions and their handling (NaN, ±∞ or zero)

IEEE 754 specifies four different formats to representing the floating point values:

- Simple Precision (32 bits)
- Double precision (64 bits)

| e-ISSN: 2278 – 8875, p-ISSN: 2320 – 3765| www.ijareeie.com | Impact Factor: 7.122|

||Volume 9, Issue 6, June 2020||

- Simple Extended Precision (≥43 bits but not too used)
- Double Extended Precision (≥79 bits, usually represented by 80

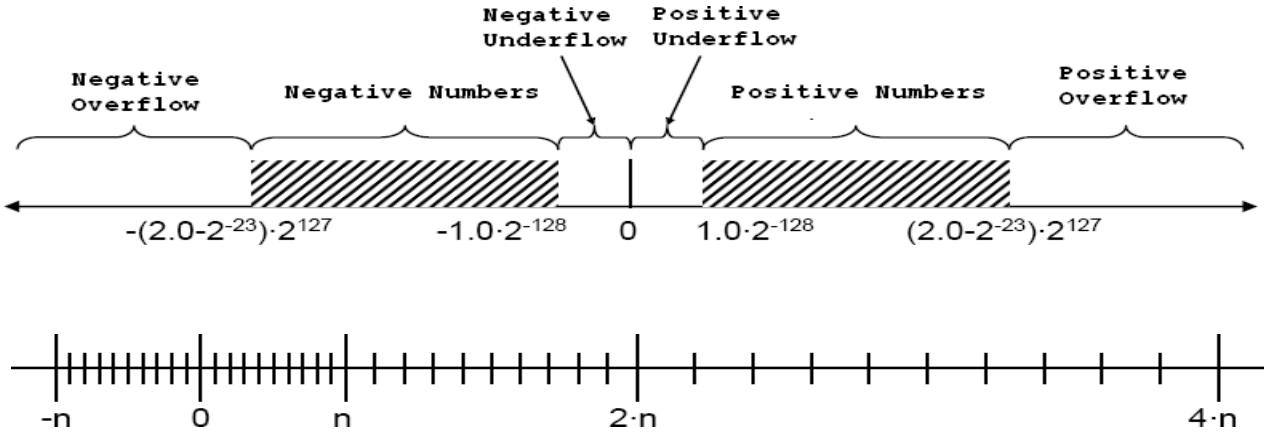


Fig.6 Standard IEE 754 specification

V.ADDER

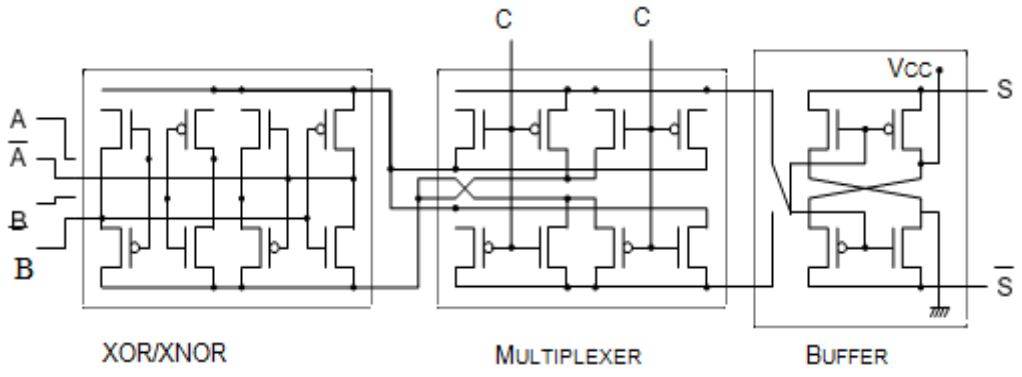
The adder is a fundamental piece of the design because it implements the addition/subtraction operation, main purpose of the 32 bit Floating Point Adder. The Adder block is composed by two entities: sign-out and adder. Sign-out is responsible for the sign operation and the adder is the adder strictly speaking. First we will examine a realization of a one-bit adder which represents a basic building block for all the more elaborate addition schemes.

A. Full Adder:

The Carry function is further rewritten defining the Carry-Propagate p_i and Carry-Generate g_i terms:

$pi = ai \oplus bi$, $gi = ai \cdot bi$

Some technologies, such as CMOS, implement the functions more efficiently by using pass-transistor circuits. For example, the critical path of the carry-in to *carry- out* uses a fast pass-transistor multiplexer in an alternative implementation of the Full Adder shown in Fig. The ability of pass-transistor logic to provide an efficient multiplexer implementation has been exploited in CPL and DPL logic families. Even an XOR gate is more efficiently implemented using multiplexer topology. A Full-Adder cell which is entirely multiplexer based as published by Hitachi is shown in Fig Such a Full-Adder realization contains only two transistors in the *Input-to-Sum* path and only one transistor in the Cin-to-Cout path. The short critical path is a factor that contributes to a remarkable speed of this implementation



| e-ISSN: 2278 – 8875, p-ISSN: 2320 – 3765| www.ijareeie.com | Impact Factor: 7.122|

||Volume 9, Issue 6, June 2020||

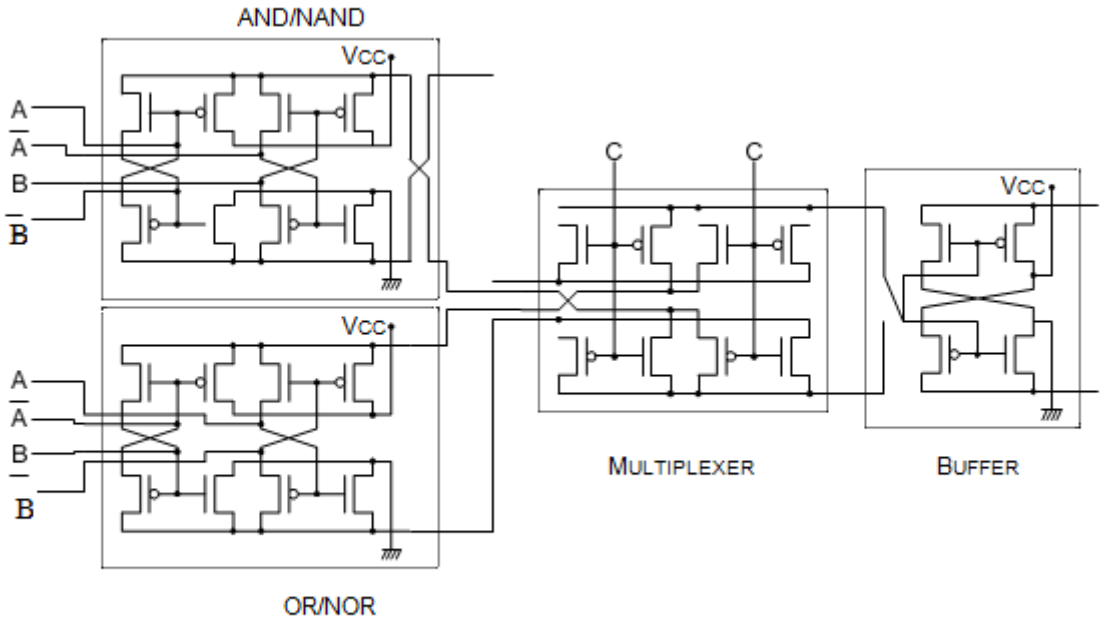


Fig.7 Pass-Transistor realization of a Full-Adder in DPL

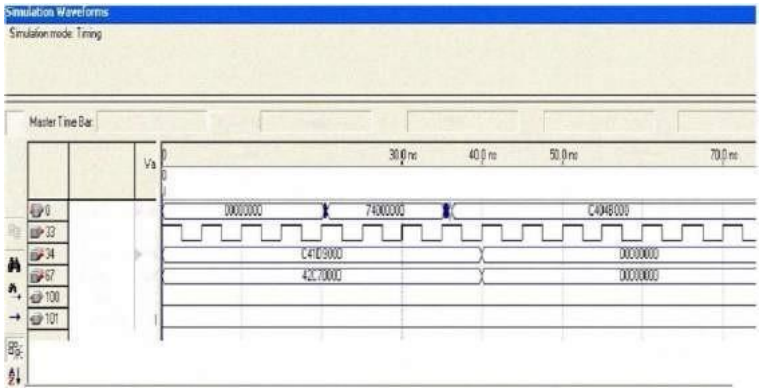


Fig.8 Simulation of Adder

VI.MULTIPLICATION

In microprocessors multiplication operation is performed in a variety of forms in hardware and software depending on the cost and transistor budget allocated for this particular operation. In the beginning stages of computer development any complex operation was usually programmed in software or coded in the micro-code of the machine. Some limited hardware assistance was provided. Today it is more likely to find full hardware implementation of the multiplication in order to satisfy growing demand for speed and due to the decreasing cost of hardware [2-5]. For simplicity, we will describe a basic multiplication algorithm which operates on positive n-bit long integers X and Y resulting in the product P which is *2n* bit long:

$$P=XY=X * \sum_{i=0}^{n-1} \quad yiri$$

This expression indicates that the multiplication process is performed by summing n terms of a partial product Pi. This product indicates that i-th term Pi is obtained by simple arithmetic left shift of X for i positions and multiplication by the single digit yi. For the binary radix (r=2), yi is 0 or 1 and multiplication by the digit yi is very simple to perform. The addition of n terms can be performed at once, by passing the partial products through a network of adders or

| e-ISSN: 2278 – 8875, p-ISSN: 2320 – 3765| www.ijareeie.com | Impact Factor: 7.122|

||Volume 9, Issue 6, June 2020||

sequentially, by adding partial products using an adder n times.

Fig.10 Generation of the Partial Product Reduction Tree in TDM multiplier

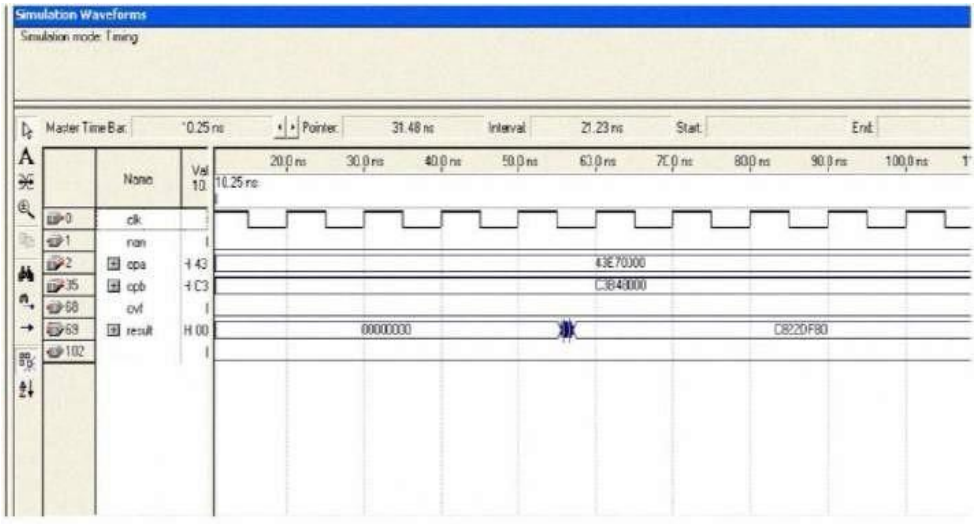


Fig 11 Simulation of Multiplier

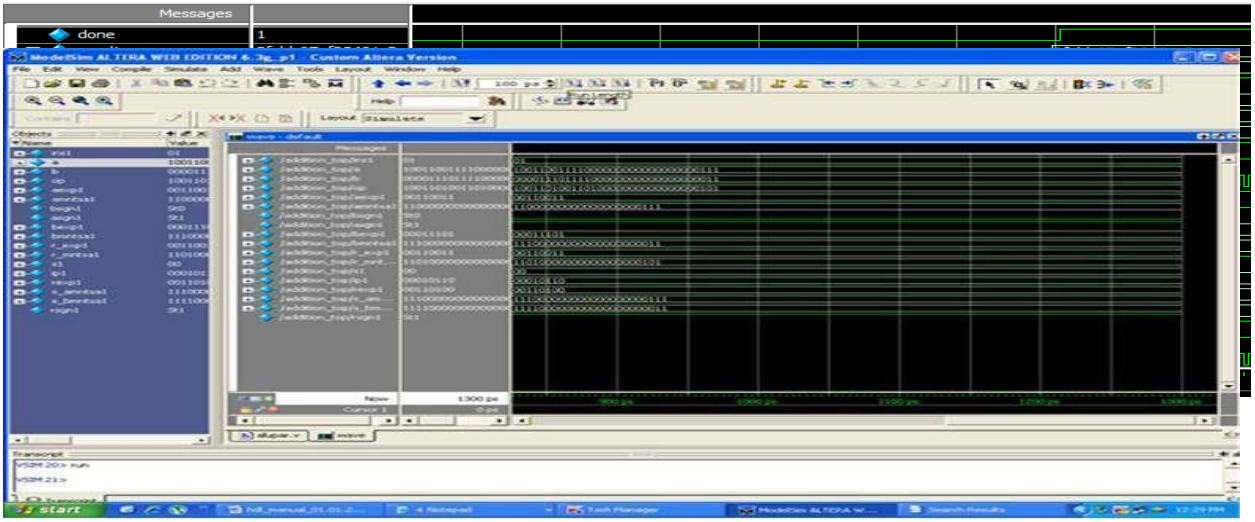
Device Utilization Memory			
Logic Utilization	Used	Available	Utilization
Number of Slices	395	4656	8%
Number of Slices FLIPFLOP	285	9312	3%
Number of 4 Input LUTs	731	9312	7%
Number of bonded IOBs	104	190	54%
Number of GCLKs	1	24	4%

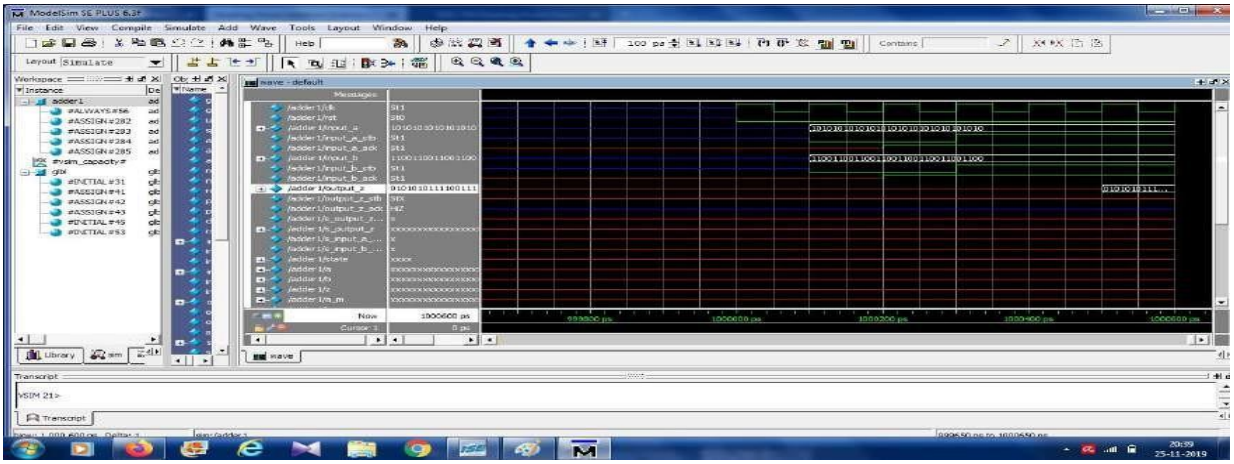
Table.2 Device Utilization Memory (estimated value)

| e-ISSN: 2278 – 8875, p-ISSN: 2320 – 3765| www.ijareeie.com | Impact Factor: 7.122|

||Volume 9, Issue 6, June 2020||

VII. RESULTS





VIII.CONCLUSION

The importance and usefulness of floating point format nowadays does not allow any discussion. Any computer or electronic device which operates with real numbers implements this type of representation and operation. During this report I have tried to explain the operation and benefits of using this notation against the fixed point. The main feature is that it can represent a very large or small numbers with a relatively small and finite quantity of memory. The VHDL code has been implemented so that all the operations are carried out with combinational logic which reaches a faster response because there are not any sequential devices as flip-flops which delays the execution time. The first one deals with type of architecture used. For example, the adder or the shifter is implemented with a known structure. Predetermined operations as addition (+) or shifting (SLL or SLR) are allow but I decided using a generate function and designing my own device which improves the time response. Finally the mixed numbers option. IEEE 754 does not say anything about the operations between subnormal and normal numbers. Obviously the design is not perfect. The accuracy is not optimal. In the future using a double precision format would be an improvement. The execution time would increase but the accuracy also would be better. Maybe a complete FPU design would be another good improvement. Multiplication and division have an easier implementation than the addition/subtraction and it would do the project more complete. Finally using the code over a FPGA and testing it physically over a board would be the last aim which would leave the design completely finished.

| e-ISSN: 2278 – 8875, p-ISSN: 2320 – 3765| www.ijareeie.com | Impact Factor: 7.122|

||Volume 9, Issue 6, June 2020||

IX.FUTURE SCOPE

[1] The present work on the adder and multiplier architecture can be extended in various directions as to enhance the performance higher order compressors 7:2, 9:2, can be used to accumulate partial products.
[2] A double precision IEEE Floating point Multiplier can be designed for multiplication intensive applications, such as DSP or graphics, could benefit several high performance multipliers on same chip. A high throughput multiplier or several multipliers working on same chip can be used for single chip video signal processing.

REFERENCES

[1] Jaenicke and W. Luk, "Parameterized Floating-Point Arithmetic on FPGAs", Proc. of IEEE ICASSP, 2001, vol. 2, pp. 897-900.
[2] “DesignChecker User Guide”, HDL Designer Series 2010.2a, Mentor Graphics, 2010
[3] “Precision® Synthesis User‟s Manual”, Precision RTL plus 2010a update 2, Mentor Graphics, 2010.
[4] Patterson, D. & Hennessy, J. (2005), Computer Organization and Design: The Hardware/software Interface, Morgan Kaufmann, John G. Proakis and Dimitris G. Manolakis (1996), “Digital Signal Processing: Principles, Algorithms and Applications”, Third Edition.
[5] R.V.K Pillai, D. Al-Khalili, and A.J. Al-Khalili. A Low Power Approach to Floating Point Adder Design. In Proceedings of ICCD '97, Computer Design: VLSI in Computers and Processors, pages 178{185, Austin, TX, October 1997.ICCD.
[6] Jinwoo Suh, Dong-In Kang, and Stephen P. Crago, “Efficient Algorithms for Fixed-Point Arithmetic Operations In An Embedded PIM”, 2005, University of Southern California/Information Sciences Institute
[7] R Dhanabal, Ushashree, "Implementation of a High Speed Single Precision Floating Point Unit using Verilog" ,International Journal of Computer Applications (0975 – 8887),2013.
[8] Steven Smith, (2003), *Digital Signal Processing-A Practical guide for Engineers and Scientists*, 3rd Edition, Elsevier Science, USA
[9] Ryan, P.G., Rawat, S., and Fuchs, W.K., “Automated diagnosis of VLSI failures,” in Digest of Papers of the 1991 IEEE VLSI Test Symposium, Atlantic City, NJ, pp.187-192, April, 1991.
[10] P.H. Bardell, W.H. McAnney, J. Savir, Built-In Test for VLSI: Pseudorandom Techniques, pp. 285-289.
[11] Cheol-Ho Jeong, Woo-Chan Park, Tack-Don Han, Sang-Woo Kim, Moon-Key Lee, "In order issue out-of-order execution floating- point coprocessor for CalmRISC32”, 15th IEEE Symposium on Computer Arithmetic, 2001
[12] Haibing Hu, Tianjun Jin, Xianmiao Zhang; Zhengyu Lu, Zhaoming Qian,“A Floating point Coprocessor Configured by a FPGA in a Digital Platform Based on Fixed-point DSP for power Electronics” IEEE 5th conference on Power Electronics and Motion Control Conference, 2006.
[13]J.M.Rabaey, A.Chandrakasan, and B.Nicolic, “Digital Integrated Circuits”, (2nd Edition) Prentice Hall, 2002.
[14] H.T. Vergos, C. Efstathiou, and D. Nikolos, “Diminished-One Modulo 2n+1 Adder Design,” IEEE Trans. Computers, vol. 51, no. 12, pp. 1389-1399, Dec. 2002.