International Journal of Current Engineering and Technology

ISSN 2277 – 4106

Available at http://inpressco.com/category/ijcet

Research Article

Design and Simulation of Floating Point Pipelined ALU Using HDL and IP Core Generator

Itagi Mahi P.[a*] and S. S. Kerur[b]

[a]Department of Electronics and Communication Engineering, SDMCET, Dharwad, India

Abstract

Short for Arithmetic Logic Unit, ALU is one of the important components within a computer processor. It performs arithmetic functions like addition, subtraction, multiplication, division etc along with logical functions. Pipelining allows execution of multiple instructions simultaneously. Pipelined ALU gives better performance which will evaluated in terms of number of clock cycles required in performing each arithmetic operation. Floating point representation is based on IEEE standard 754. In this paper a pipelined ALU is proposed simulating five arithmetic operations namely addition, subtraction, multiplication, division and square root in the HDL environment. Simulation Results is also obtained in IP Core Generator supported by Xilinx. Synthesis is carried out on the Xilinx 13.2 platform and ISim is used for the simulation process.

Keywords: *ALU, IEEE standard 754, Single Precision, Pipelining, Hardware Description Language (HDL), VHDL.*

1. Introduction

The idea of floating-point representation over intrinsically integer fixed-point numbers, which consist purely of significand, is that expanding it with the exponent component achieves greater range. For instance, to represent large values, e.g. distances between galaxies, there is no need to keep all 39 decimal places down to femtometre-resolution. Assuming that the best resolution is in light years, only the 9 most significant decimal digits matter, whereas the remaining 30 digits carry pure noise, and thus can be safely dropped.

In computing, 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 term floating point refers to the fact that a number's radix point (decimal point, or, more commonly in computers, binary point) can "float"; that is, it can be placed anywhere relative to the significant digits of the number. Over the years, a variety of floating-point representations have been used in computers. However, since the 1990s, the most commonly encountered representation is that defined by the IEEE 754 Standard.

The Xilinx CORE Generator System delivers a library of both parameterizable and point solution LogiCORE IP cores with detailed data sheet specifications. LogiCORE IP are designed and supported by Xilinx.

1.1 Single Precision Floating Point Format

The IEEE has standardized the computer representation for binary floating-point numbers in IEEE 754. This standard is followed by almost all modern machines. It is an effort for the computer manufacturers to conform to a common representation and arithmetic convention for floating point data.

The standard defines:

- Arithmeticformats: sets of binary and decimal floating-point data, which consist of finite numbers (including signed zeros and subnormal numbers), infinities, and special "not a number" values (NaNs)
- Interchange formats: encodings (bit strings) that may be used to exchange floating-point data in an efficient and compact form
- Rounding rules: properties to be satisfied when rounding numbers during arithmetic and conversions
- Operations: arithmetic and other operations on arithmetic formats
- Exception handling: indications of exceptional conditions (such as division by zero, overflow, etc.)

The IEEE single precision floating point standard representation requires a 32 bit word whose bits may be represented as numbered from 0 to 31, left to right. The format is as follows:

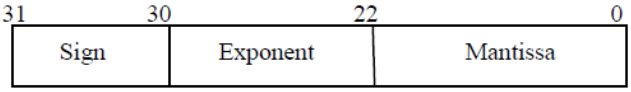

Fig1: Single Precision Floating Point Format

1-bit “Sign” signifies whether the number is positive or negative. “1” indicates negative number and “0” indicates

*Corresponding author: **Itagi Mahi P**

positive number. 8-bit "Exponent" provides the exponent range from E (min) =-126 to E (max) =127. 23-bit "Mantissa" signifies the fractional part of a number. The mantissa must not be confused with the significand. The leading "1" in the significand is made implicit.

1.2 Conversion of Decimal to Floating Point Number

Conversion of Decimal to Floating point 32 bit format is explained with an example. Suppose the decimal number considered is 129.85. Before converting into floating format this number is converted into binary value which is 10000001.110111. After conversion the radix point is moved to the left such that there will be only one bit towards the left of the radix point and this bit must be 1. This bit is known as "hidden bit". The above binary value now can be represented as 1.00000011101110000000000. The number which is after the radix point is called the mantissa which is of 23 bits and the whole number is called significand which is of 24 bits(including the hidden bit). Now the number of times the radix point is shifted (say x) is counted. In above case there is 7 times shifting of radix point to the left. This value must be added to 127 to get the exponent value i.e. original exponent value is 127 + "x". Thus the exponent becomes 127 + 7 = 134 which is 10000110. Sign bit i.e. MSB is "0" because number is + ve. Now the result is assembled into 32 bit format which is sign, exponent, mantissa: 01000011000000011101110000000000.

1.3 Xilinx Core Generator

Xilinx CORE Generator System accelerates design time by providing access to highly parameterized Intellectual Properties (IP) for Xilinx FPGAs and is included in the ISE® Design Suite. CORE Generator provides a catalog of architecture specific, domain-specific (embedded, connectivity and DSP), and market specific IP (Automotive, Consumer, Mil/Aero, Communications, Broadcast etc.). These user-customizable IP functions range in complexity from commonly used functions, such as memories and FIFOs, to system-level building blocks, such as filters and transforms. Using these IP blocks can save days to months of design time. The highly optimized IP allows FPGA designers to focus efforts on building designs quicker while helping bring products to market faster. Through its seamless integration with the ISE development environment, the CORE Generator system streamlines the design process and improves design quality.

2. Literature Survey

In recent years, Floating-point numbers are widely adopted in many applications due to its high dynamic range and good robustness against quantization errors, capabilities. Floating-point representation is able to retain its resolution and accuracy. IEEE specified standard for floating-point representation is known as IEEE 754 standard. This standard specifies interchange and arithmetic formats and methods for binary and decimal floating-point arithmetic in computer programming environments. (IEEE 754-2008)

The main objective of implementation of floating point operation on reconfigurable hardware is to utilize less chip area with less combinational delay (Karan Gumber et.al, May 2012) which means less latency i.e. faster speed. A parameterizable floating point adder and multiplier implemented using the software-like language Handel-C, using the Xilinx XCV1000 FPGA, a five stages pipelined multiplier achieved 28MFlops (A. Jaenicke et. Al ,2001). The hardware needed for the parallel 32-bit multiplier is approximately 3 times that of serial.

A single precision floating point multiplier that doesn't support rounding modes can be implemented using a digit-serial multiplier (L. Louca et. al, 1996). The ALU is a fundamental building block of the central processing unit of a computer, and even the simplest microprocessors contain one for purposes such as maintaining timers. By using pipeline with ALU design, ALU provides a high performance. With pipelining concept ALU execute multiple instructions simultaneously (Suchita Pare et. al, 2012)

3. Methodology

The block diagram of the designed arithmetic unit is shown in the following figure. It supports five arithmetic operations: addition, subtraction, multiplication, division and square root. It has 4 inputs: a start bit, two 32-bit operand in IEEE format, one 3-bit operation code, one 2 bit code for rounding operation; and has outputs: 32-bit output in IEEE format, one ready bit, eight exceptions (Inexact, Overflow, Underflow, Divide-by-zero, Infinity, Zero, QNaN, SNaN). All arithmetic operations have been carried out in four separate modules one for addition and subtraction and one each for multiplication, division and square root as shown in figure 3.1. In this unit one can select operation to be performed on the 32-bit operands by a 3-bit op-code and the same op-code selects the output from that particular module and connects it to the final output of the unit. Ready bit will be high when result will be available to be taken from the output. Particular exception signal will be high whenever that type of exception will occur. Since the result precision is not infinite, sometimes rounding is necessary. In this paper the rounding mode considered is the 'Round to Nearest Even', wherein the value is rounded up or down to the nearest infinitely precise result. If the value is exactly halfway between two infinitely precise results, then it should be rounded up to the nearest infinitely precise even.

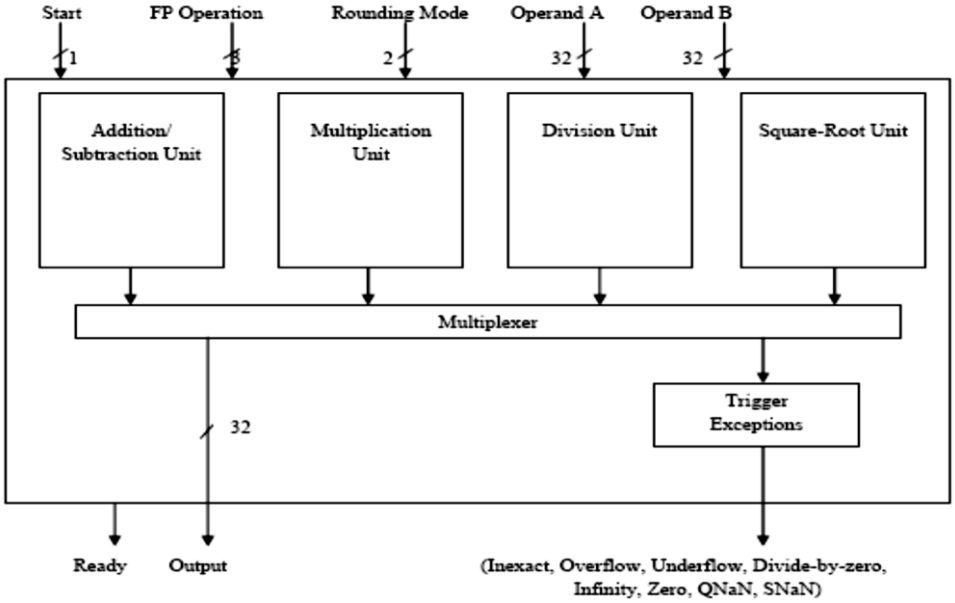

Fig 2: Floating Point ALU Basic Architecture

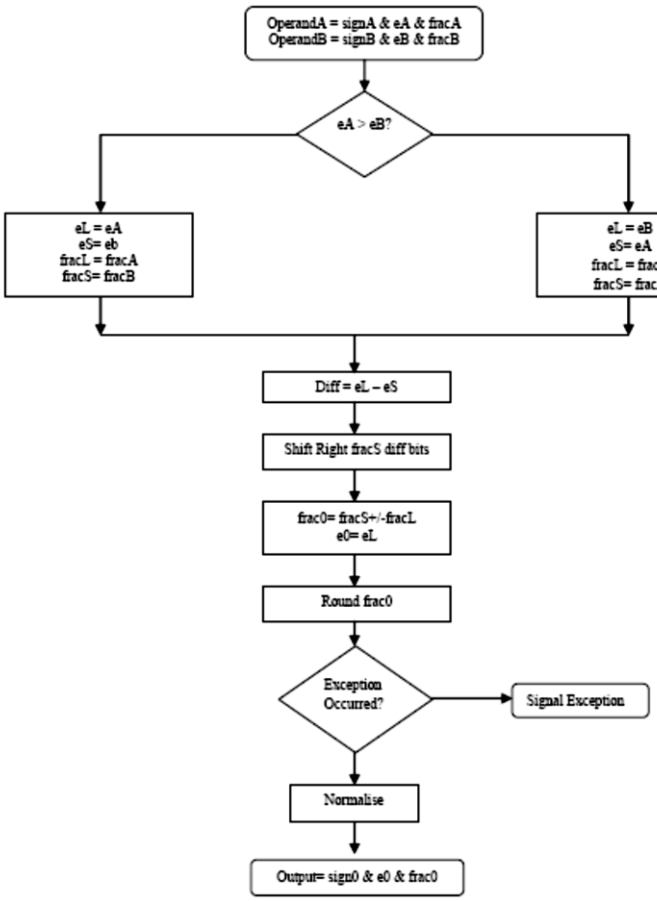

Fig3: Flowchart for floating point addition/subtraction

All arithmetic operations have these three stages:

- Pre-normalize: The operands are transformed into formats that makes them easy and efficient to handle internally.
- Arithmetic core: The basic arithmetic operations are done here.
- 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.

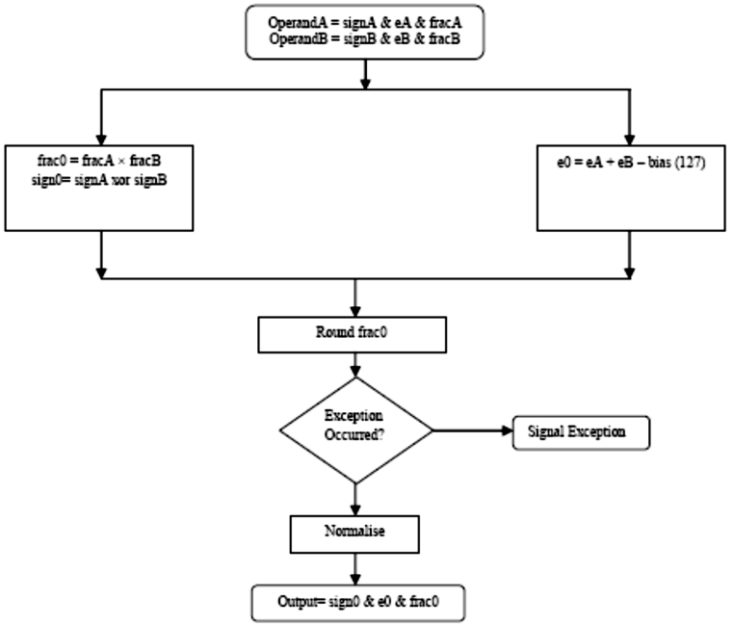

Fig 4: Flowchart for floating point multiplication

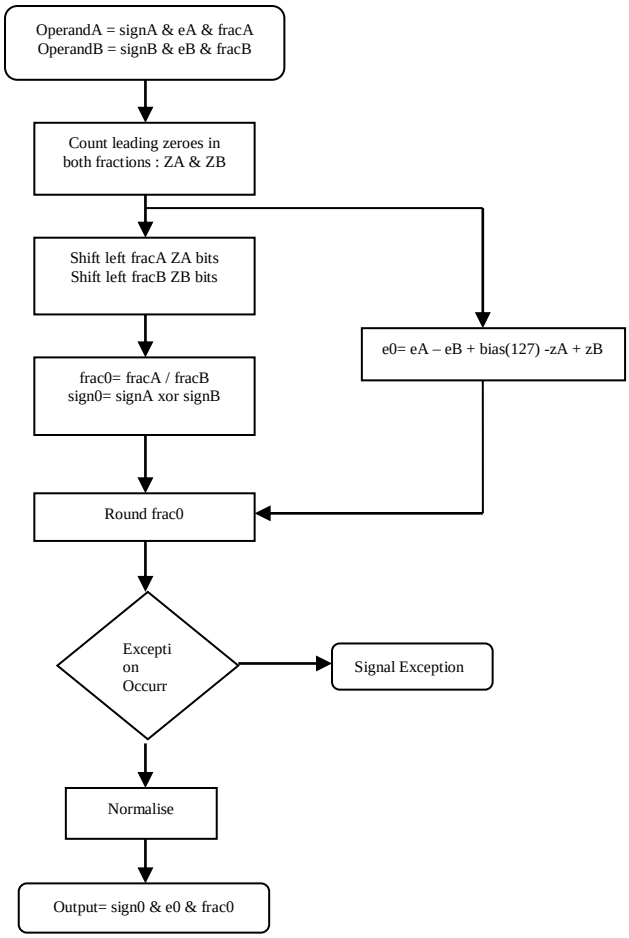

Fig5: Flowchart for floating point division

3.1 Addition and Subtraction

The conventional floating-point addition algorithm consists of five stages - exponent difference, pre-alignment, addition, normalization and rounding. The algorithm used for adding or subtracting floating point numbers is shown in the following flowchart.

3.2 Multiplication Algorithm

In floating point multiplication, the two mantissas are to be multiplied, and the two exponents are to be added. In order to perform floating-point multiplication, a simple algorithm is realized:

- Add the exponents and subtract 127 (bias)
- Multiply the mantissas and determine the sign of the result
- Normalize the resulting value, if necessary

3.3 Division Algorithm

It is shown in Fig 5.

3.4 Square Root

In the square-root algorithm used, multiplications were replaced with left-shifts and all divisions with right-shifts. This makes the algorithm very efficient and fast for hardware implementations. The algorithm used is similar to division algorithm except for only one operand will be considered. An iterative algorithm is used [2]. The equation for calculating the output exponent value is:

$e_O = (e_A + \text{bias}\ (127) - z_A) / 2$

3.5 Xilinx IP Core Generator

The IP provides all the necessary IEEE compliant, highly

parameterizable floating-point arithmetic operators, allowing us control the fraction and exponent word lengths, as well as the latency and implementation specifics for the required algorithm. The core chosen here is floating point addition core as shown in the figure below:

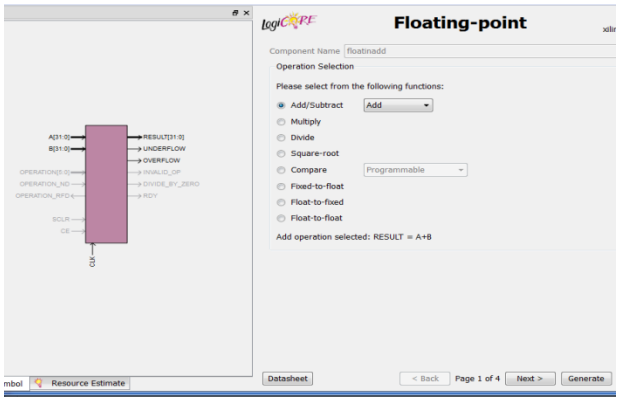

Fig 6: Snapshot of the Floating Point Core Generator window

The following files are generated after the generation of the core:

- XCO
- EDN/NGC
- SYM
- VHO or VEO Template Files
- VHD or V Simulation Wrapper Files
- VHD or V Source Code Files

4. Simulation Results

4.1 Addition

Operand A= 01000000101101000000000000000000
Operand B= 00111111100011100011010100111111

Output = **01000000 11010111 10001101 01010000**

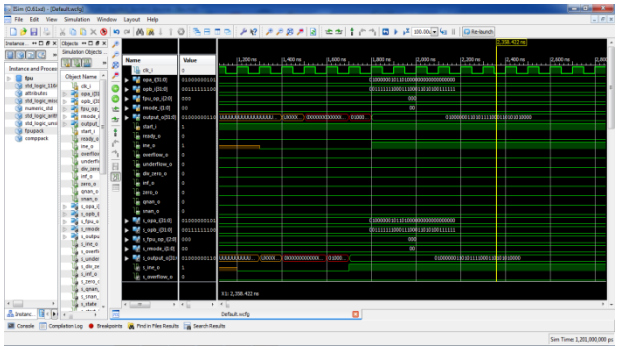

Fig7: Simulation results for floating point addition

4.2 Subtraction

Operand A= 01000000101101000000000000000000
Operand B = 00111111100011100011010100111111

Output = **01000000 10010000 01110010 10110000**

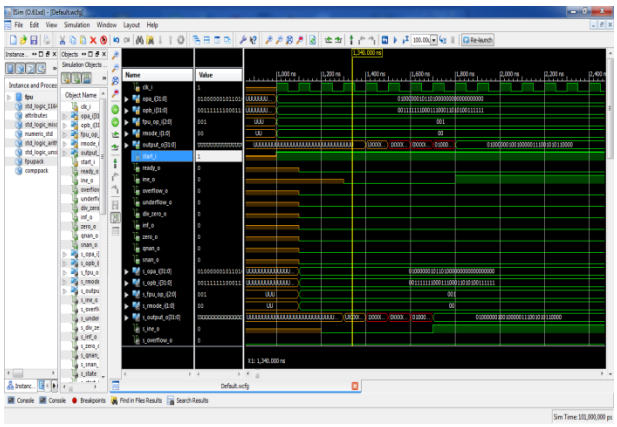

Fig8: Simulation results for floating point subtraction

4.3 Multiplication

Operand A= 00111111100000000000000000000000
Operand B =00111111100000000000000000000000
Output = **00111111 10000000 00000000000000000**

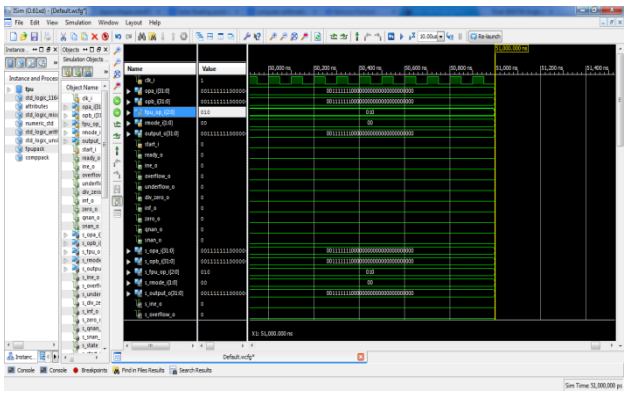

Fig 9: Simulation results for floating point multiplication

4.4 Division

Operand A= 00111111100000000000000000000000
Operand B =00111111100000000000000000000000
Output = **00111111 10000000 00000000000000000**

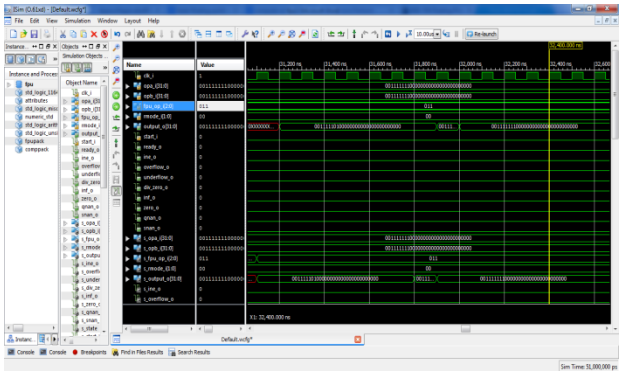

Fig10: Simulation results for floating point division

4.5 Square Root

Operand A= 00111111100000000000000000000000
Output = **00111111 10000000 0000000000000000**

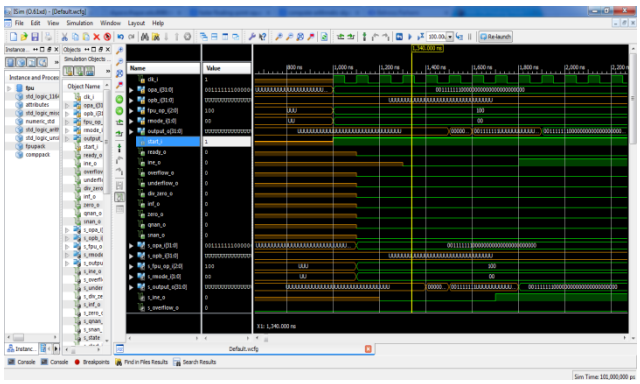

Fig11: Simulation results for floating point square root

4.6 Floating point Addition Using IP Core Generator

Simulation results for for addition is shown in the following figure obtained by using the Xilinx IP Core generator. The same operands have been used as used in the addition done in the VHDL environment.

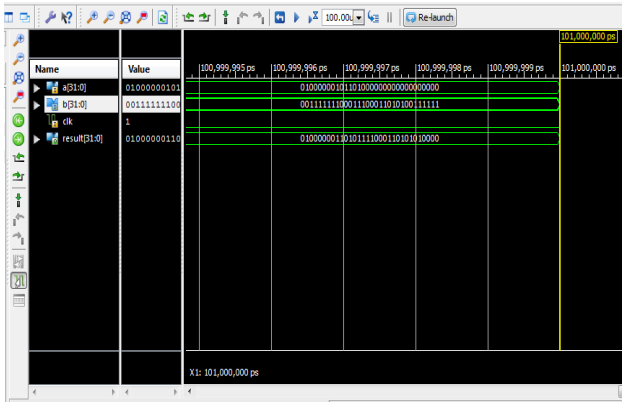

Fig12: Simulation results for floating point addition using IP Core Generator

The simulation results of only floating point addition have been shown for convenience and the other arithmetic operations follow in the similar way.

5. Conclusion

The pipelined Floating point Arithmetic unit has been designed to perform five arithmetic operations, addition, subtraction, multiplication, division and square root, on floating point numbers. IEEE 754 standard based floating point representation has been used. The unit has been coded in VHDL. The same arithmetic operations have also been simulated in Xilinx IP Core Generator.

Table1: Result in terms of no. of clock cycles

OPERATION	NO. OF CYCLES
Addition	4
Subtraction	4
Multiplication	8
Division	8
Square Root	10

Device Utilization Summary:

Number of Slice Registers	2%
Number of Slice LUTs	10%
Number used as Logic	10%
Number used as Memory	0%
Number of LUT Flip Flop pairs used	8070
Number with an unused Flip Flop	80%
Number with an unused LUT	8%
Number of fully used LUT-FF pairs	10%
Number of IOs	112
Number of bonded IOBs	17%

Acknowledgment

The authors thank the authorities of Sri Dharmasthala Manjunatheshwara College of Engineering and Technology, Dhavalagiri, Dharwad, Karnataka, India for encouraging them for this research work.

Reference

IEEE 754-2008 (2008), IEEE Standard for Floating-Point Arithmetic.

Behrooz Parhami (2000), Computer Arithmetic Algorithms and Hardware Designs, Oxford University Press.

Karan Gumber, Sharmelee Thangjam (May 2012), Performance Analysis of Floating Point Adder using VHDL on Reconfigurable Hardware, International Journal of Computer Applications, Volume 46– No.9.

A. Jaenicke and W. Luk (2001), Parameterized Floating-Point Arithmetic on FPGAs, Proc. of IEEE ICASSP, Vol. 2, pp.897-900.

L. Louca, T. A. Cook, and W. H. Johnson (1996), Implementation of IEEE Single Precision Floating Point Addition and Multiplication on FPGAs Proceedings of 83 the IEEE Symposium on FPGAs for Custom Computing Machines (FCCM'96), pp. 107–116.

Suchita Pare, Dr. Rita Jain (December 2012), 32 Bit Floating point Arithmetic Logic Unit ALU Design and Simulation, International Journal of Emerging Trends in Electronics and Computer Science (IJETECS) Volume 1, Issue 8.