www.ijecs.in
International Journal Of Engineering And Computer Science ISSN: 2319-7242
Volume 4 Issue 9 Sep 2015, Page No. 14523-14528

FPGA Implementation of Double Precision Floating Point Arithmetic Unit

Prabhjot Kaur[1], Ankur Sharma[2], Raminder Preet Pal Singh[3]

[1]Arni University, Kathgarh, Indora, HP
prabhjotgoli@gmail.com

[2]Arni University, Kathgarh, Indora, HP
Ankur.sharma.ind@gmail.com

[3]Arni University, Kathgarh, Indora, HP
raminder_212003@rediffmail.com

Abstract: ***Every computer has a floating point processor or a dedicated accelerator that fulfils the requirements of precision using detailed floating point arithmetic. The main applications of floating points today are in the field of medical imaging, biometrics, motion capture and audio applications, including broadcast, conferencing, musical instruments and professional audio. Floating point representation can support a much wider range of values than fixed point representation. In this design the complex logic operations which consist of various multiple numbers of stages are converted into single stage implementation. Once the inputs are applied to the input terminals the final output is obtained at the output terminals there are no intermediate stages. So now the input take less time to reach at output due to single stage implementation and the number of flip flops and other intermediate required circuits are less as a result the area require is less in the presented design called high throughput design. All four individual units addition, subtraction, multiplication and division are designed using Verilog and than combined into one single unit. The code is dumped into low cost Spartan 6 FPGA.***

Keywords: Floating Point, Throughput, IEEE, FPGA, Double Precision, Verilog, Arithmetic Unit

1. Introduction

A floating-point arithmetic unit designed to carry out operations on floating point numbers. Floating point arithmetic unit is widely used in scientific, commerce and in signal processing applications. Floating point representation can support a much wider range of values than fixed point representation. To represent very large or small values, large range is required as the integer representation is no longer appropriate. These values can be represented using the IEEE-754 standard based floating point representation. The overall throughput of the design can be increased by using a single stage implementation.

2. High Throughput Design

In this design the complex logic operations which consist of various multiple numbers of stages are converted into single stage implementation as shown in Fig 2. Time taken by data to reach at D_{out} from D_{in} is less in comparison to the multiple stages design shown in Fig 1. In multiple stages design there are many combinational delay and also the same number of flip flop delays but in single stage implementation there is only one combinational delay that is slightly bigger than combinational delay in multiple stage design and also less flip flops is used in single stage implementation so overall throughput is increased in single stage implementation

2.1 Design with Divided Smaller Combo Delays - This design represents a multiple stage implementation having several small combo delays, propagation delay of in between flip flops and set up delay of in between flip flops. Due to multiple stage implementations it has several delays due to flip flop. The clock time period must be greater than the sum of time period of propagation delay due to input and in between flip flops, time period of set up time of output and in between flip flops and small combinational delay of each component. Most of the work till time is based on design with divided smaller combo delays.

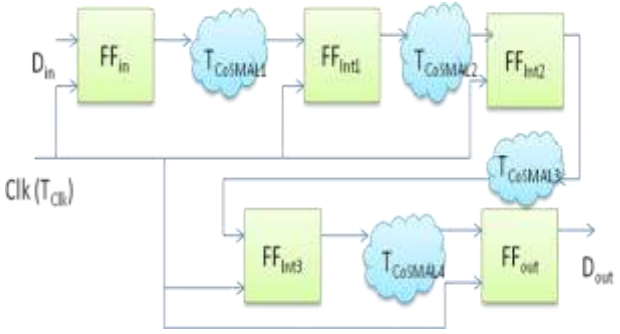

Figure 1: Design with Low Throughput (Divided into Smaller Combo Delays)

$$T_{Clk} \geq (T_{propFFin} + T_{coSMAL1} + T_{setupFFInt1}) + (T_{propFFInt1} + T_{coSMAL2} + T_{setupFFInt2}) + (T_{propFFInt2} + T_{coSMAL3} + T_{setupFFInt3}) + (T_{propFFInt3} + T_{coSMAL4} + T_{setupFFout})$$

Where

T_{Ckl}= Time period of clock

$T_{propFFin}$=Propagation delay of input flip flop and in between flip flops

$T_{comboSMAL}$=Small Combinational delay due to in between circuits

$T_{setupFFout}$=Setup time of output flip Flop and in between flip flops

2.2 Design with one Bigger Combo Delay- This design represents a single stage implementation having a bigger combo delay, propagation delay of input flip flop and set up delay of output flip flop. Due to single stage implementation it has only two delays due to flip flop. The clock time period must be greater than the sum of time period of propagation delay due to input flip flop, time period of set up time of output flip flop and combinational delay in between circuits. Our design is based on this single stage design. So it has high throughput.

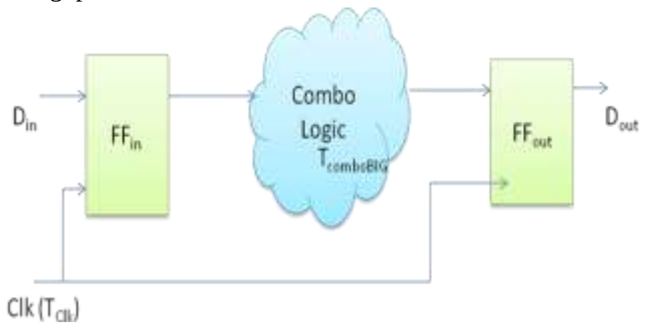

Figure 2: Design with High Throughput (Bigger Combo Delay)

$\mathbf{T_{Clk}} \geq T_{propFFin} + T_{comboBIG} + T_{setupFFout}$

Where

T_{Ckl}= Time period of clock

$T_{propFFin}$=Propagation delay of input flip flop

$T_{comboBIG}$=Combinational delay due to in between circuits

$T_{setupFFout}$=Setup time of output flip Flop

3. Implementation of Double Precision Floating Point Arithmetic Unit

The block diagram of the proposed arithmetic unit is given in figure. The unit supports four arithmetic operations: Add, Subtract, Multiply and Divide. In this design the complex logic operations segmented and implemented into various multiple numbers of stages are converted into single stage implementation in simple words we can say that the multiple stages are converted into single stage. Once the inputs are applied to the input terminals the final output is obtained at the output terminals there are no intermediate stages. So now the inputs take less time to reach at output terminals and due to single stage implementation the number of flip flops and other intermediate required circuits are less as a result the area require is less in the presented design.

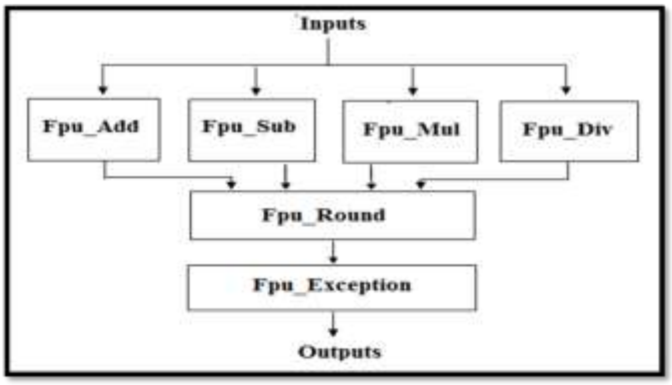

Figure 3: Block diagram of double precision floating point arithmetic unit

All arithmetic operations have been carried out in four separate modules one for addition, one for subtraction, one for multiplication and one for division as shown in figure 3. In this unit one can select operation to be performed on the 64-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. Particular exception signal will be high whenever that type of exception will occur.

3.1 Fpu_Add- Floating Point adder

In order to add two fractions, the associated exponents must be equal. Thus, if the two exponents are different, we must un normalize one of the fractions and adjust the exponents accordingly. The smaller number is the one that should adjusted so that if significant digits are lost, the effect is not significant.

3.2 Floating Point Subtraction

In order to subtract two fractions, the associated exponents must be equal. Thus, if the two exponents are different, we must un normalize one of the fractions and adjust the exponents accordingly. The smaller number is the one that should adjusted so that if significant digits are lost, the effect is not significant.

3.3 Floating Point Multiplication:

There are two operand named operand A and operand B to be multiplied. The mantissa of operand A and the leading ‗1' (for normalized numbers) are stored in the 53-bit register (mul_a). The mantissa of operand B and the leading ‗1' (for normalized numbers) are stored in the 53-bit register (mul_b). Multiplying all 53 bits of mul_a by 53 bits of mul_b would result in a 106-bit product. Depending on the synthesis tool used, this might be synthesized in different ways that would not take efficient advantage of the multiplier resources in the target device. 53 bit by 53 bit multipliers are not available in the most popular Xilinx and Altera FPGAs, so the multiply would be broken down into smaller multiplies and the results would be added together to give the final 106-bit product. Instead of relying on the synthesis tool to break down the multiply, which might result in a slow and inefficient layout of FPGA resources, the module (fpu_mul) breaks up the multiply into smaller 24-bit by 17-bit multiplies.

3.4. Floating Point Division

DOI: 10.18535/ijecs/v4i9.76

The leading '1' (if normalized) and mantissa of operand A is the dividend, and the leading '1' (if normalized) and mantissa of operand B is the divisor. The divide is executed long hand style, with one bit of the quotient calculated each clock cycle based on a comparison between the dividend register (dividend_reg) and the divisor register (divisor_reg). If the dividend is greater than the divisor, the quotient bit is '1', and then the divisor is subtracted from the dividend, this difference is shifted one bit to the left, and it becomes the dividend for the next clock cycle. If the dividend is less than the divisor, the dividend is shifted one bit to the left, and then this shifted value becomes the dividend for the next clock cycle. The exponent for the divide operation is calculated from the exponent fields of operands A and B. The exponent of operand A is added to 1023, and then the exponent of operand B is subtracted from this sum. The result is the exponent value of the output of the divide operation. If the result is less than 0, the quotient will be right shifted by the amount.

3.5 Rounding

Rounding takes a number regarded as infinitely precise and, if necessary, modifies it to fit in the destination's format while signalling the inexact exception, underflow, or overflow when appropriate.

Round to nearest even: This is the standard default rounding. 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.

Round-to-Zero: Basically in this mode the number will not be rounded. The excess bits will simply get truncated, e.g. 3.47 will be truncated to 3.5

Round-Up: In this mode the number will be rounded up towards +∞, e.g. 5.2 will be rounded to 6, while -4.2 to -4

Round-Down: The opposite of round-up, the number will be rounded up towards -∞, e.g. 5.2 will be rounded to 5, while -4.2 to -5

3.6 Exceptions

Exception is an event that occurs when an operation on some particular operands has no outcome suitable for a reasonable application.

The five possible exceptions are:

Invalid: Operation are like square root of a negative number, returning of NaN by default, etc., output of which does not exist.

Division by zero: It is an operation on a finite operand which gives an exact infinite result for e.g., 1/0 or log (0) that returns positive or negative infinity by default.

Overflow: It occurs when an operation results a very large number that can't be represented correctly i.e. which returns ±infinity by default (for round-to-nearest mode).

Underflow: It occurs when an operation results very small i.e. outside the normal range and inexact by default.

Inexact: It occurs whenever the result of an arithmetic operation is not exact due to the restricted exponent or precision range.

4. Synthesis, Timing and Simulation Result

4.1. Synthesis Result

Logic Utilization	Used
Number of Slice Registers	4463
Number of Slice LUTs	6204
Number of fully used LUT-FF pairs	2999
Number of bonded IOBs	206
Number of BUFG/BUFGCTRLs	1
Number of DSP48Es	12

4.2 Timing Result

Minimum Period	8.906ns (Max frequency: **112.284** MHz)
Minimum Input arrival time before clocks	5.729ns
Maximum output required time after clocks	3.597ns

4.3 . Simulation Result

The simulation results of double precision floating point arithmetic unit (Addition, Subtraction, Multiplication and Division) is shown in fig 3, fig 4, fig 5, fig 6 respectively. In the waveforms clock defines the applied frequency to the signals. Fpu_op defines the operation to be preformed that is 0=addition,1=subtraction, 2=multiplication and 3=division. Opa1 and Opa1 defines the input operand one and input operand two respectively. The r_mode signal defines the various rounding modes (00=Round to nearest even, 01=Round-to-Zero, 10=Round-Up, 11=Round-Down. Fpu_out defines the final output of the signals.

Simulation Result of floating point addition- It is calculated for the two operands of 64 bits each. The reset signal is kept low throughout the simulation, so that operands are initialised all at once. With always combinational block all registers values are initialised first, then at the high of enable signal, addition of the two operands are calculated. First, after calculating the result, the result goes into fpu_round and then goes into fpu_exceptions. From fpu_exceptions the out signal gives the output. This module passes the sign, exponent and mantissa to the rounding module. As frequency is 112.284 MHz so one clock cycle completes 8.906 ns and 20 clock cycles completes in 8.906ns x 20 =178.12ns. Therefore the addition process completes in 178.12ns.

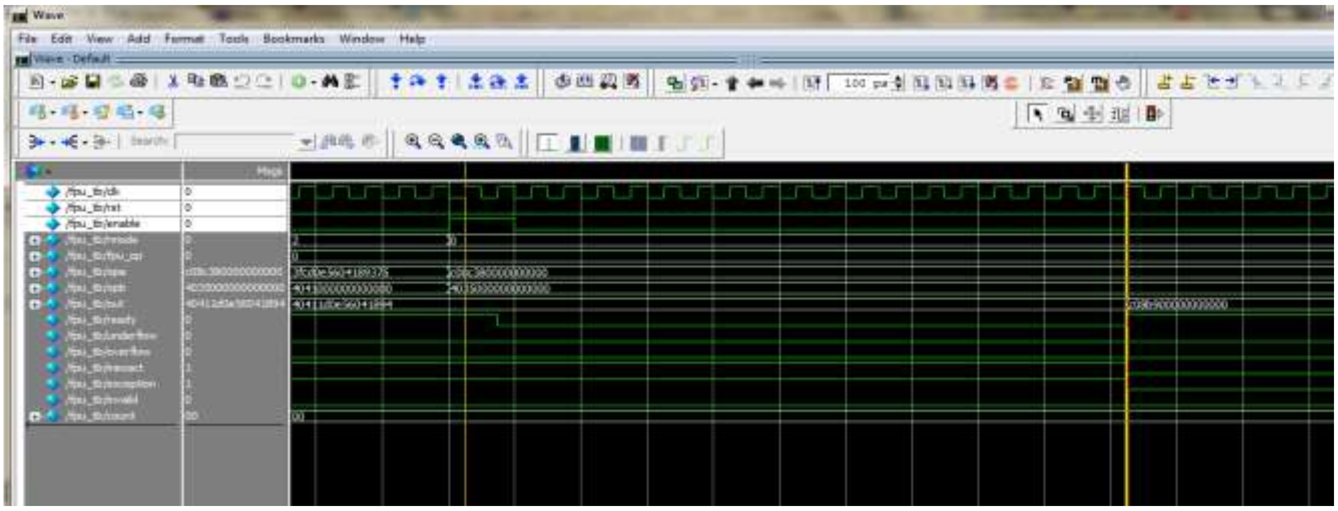
Figure 3: Simulation Result of Floating Point Addition

Simulation result of floating point subtraction- In case of subtraction, if operand b is bigger than operand a, then the result is routed to the following. Take the smaller number and subtract it from bigger number and invert the sign of the result. At the clk, the reset is provided and singals are made to zero. With the help of combinational logic is applied and all the register values are initialised to zero and after that at the high of enable signal the subtraction is performed. As with the fpu_add module, the fpu_sub module passes on the sign, exponent, and mantissa signals to the rounding module. There are 2 extra remainder bits at the end of the mantissa that determine if rounding will be performed, with the least significant remainder bit calculated by performing an OR on any bits that were shifted out of the mantissa due to the difference in exponents. As frequency is 112.284 MHz so one clock cycle completes 8.906 ns and 20 clock cycles completes in 8.906ns x 20 =178.12ns. Therefore the addition process completes in 178.12ns.

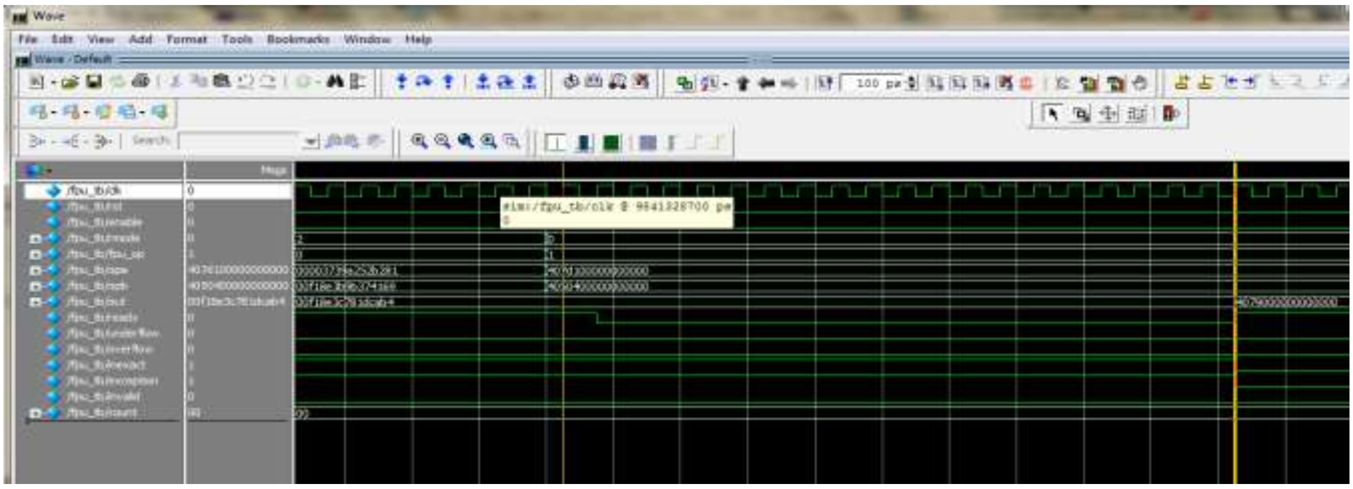
Figure 4: Simulation Result of Floating Point Subtraction

Simulation result of floating point multiplication- It is calculated for the two operands of 64 bits each. The reset signal is kept low throughout the simulation, so that operands are initialised all at once. With always combinational block all registers values are initialised first, then at the high of enable signal, multiplication of the two operands are calculated first, after calculating the result, the result goes into fpu_round and then goes into fpu_exceptions. From fpu_exceptions the out signal gives the output. As frequency is 112.284 MHz so one clock cycle completes 8.906 ns and 24 clock cycles completes in 8.906ns x 24 =213.75ns. Therefore the addition process completes in 213.75ns.

Simulation result of floating point division- It is calculated for the two operands of 64 bits each. The reset signal is kept low throughout the simulation, so that operands are initialised all at once. With always combinational block all registers values are initialised first, then at the high of enable signal, division of the two operands are calculated first, after calculating the result, the result goes into fpu_round and then goes into fpu_exceptions. From fpu_exceptions the out signal gives the output. As frequency is 112.284 MHz so one clock cycle completes 8.906 ns and 74 clock cycles completes in 8.906ns x 74 =610ns. Therefore the addition process completes in 610ns.

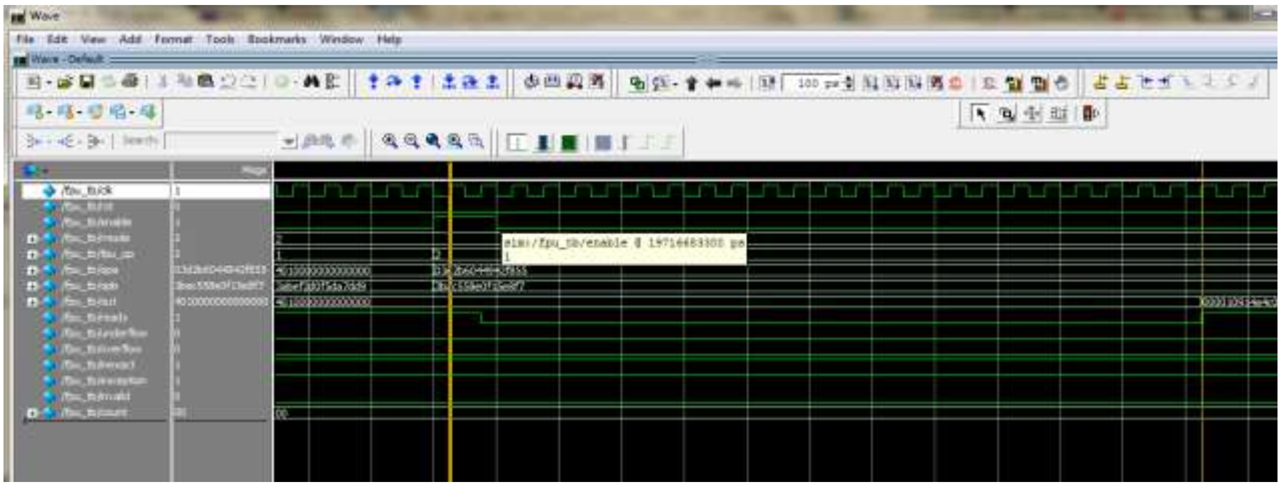
Wave
File Edit View Add Format Tools Bookmarks Window Help
Wave - Default
/fpu_tb/clk
sim:/fpu_tb/enable @ 19716663300 ps
1

Figure 5: Simulation Result of Floating Point Multiplication

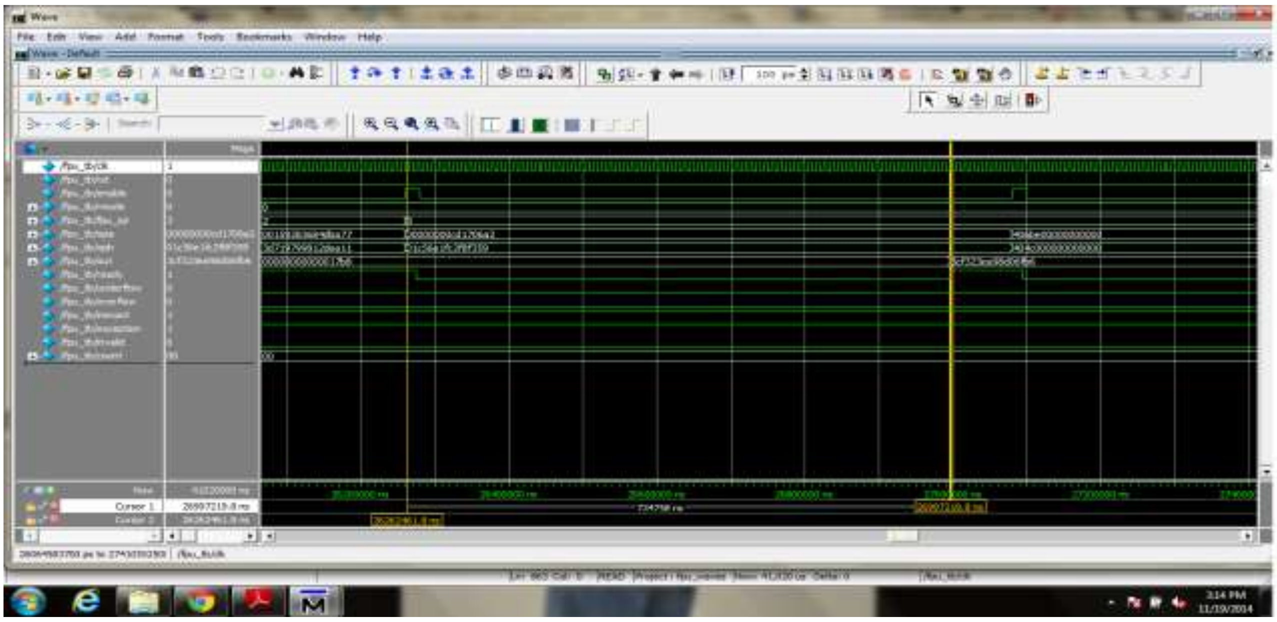

Figure 6: Simulation Result of Floating Point Divisio

5. Conclusion

This paper presents a single stage implementation of double precision floating point arithmetic unit. The complete design is captured in Verilog Hardware description language (HDL), tested in simulation using Questa Sim, placed and routed on a Spartan 6 FPGA from Xilinx. When synthesized, this module used 4463 number of slice registers and 6204 number of slice LUTs. The overall throughput is increased in this design.

References

[1] Deepa Saini , Bijender M'dia "Floating Point Unit Implementation on FPGA" International Journal Of Computational Engineering Research(IJCER), Vol. 2 , pp.972-976, Issue No.3, May-June 2012

[2] Paschalakis, S., Lee, P., "Double Precision Floating-Point Arithmetic on FPGAs", In Proc. 2003 2nd IEEE International Conference on Field Programmable Technology (FPT '03), Tokyo, Japan, pp. 352-358, 2003

[3] Addanki Puma Ramesh, A. V. N. Tilak, A.M.Prasad "An FPGA Based High Speed IEEE-754 Double Precision Floating Point Multiplier Using Verilog" 2013 International Conference on Emerging Trends in VLSI, Embedded System, Nano Electronics and Telecommunication System (ICEVENT), pp. 1-5, 7-9 Jan. 2013

[4] Ushasree G, R Dhanabal, Sarat Kumar Sahoo "Implementation of a High Speed Single Precision Floating Point Unit using Verilog" International Journal of Computer Applications National conference on VSLI and Embedded systems, pp.32-36, 2013

[5] Pramod Kumar Jain, Hemant Ghayvat , D.S Ajnar "Double Precision Optimized Arithmetic Hardware Design For Binary & Floating Point Operands" International Journal of Power Control Signal and Computation (IJPCSC) Vol. 2 No. 2 ISSN : 0976-268X

[6] Basit Riaz Sheikh and Rajit Manohar "An Operand-Optimized Asynchronous IEEE 754 Double-Precision

Floating-Point Adder", IEEE Symposium on Asynchronous Circuits and Systems (ASYNC), pp. 151 – 162, 3-6 May 2010

[7] Ms. Anjana Sasidharan, Mr. M.K. Arun" Vhdl Implementation Of Ieee 754 Floating Point Unit" IJAICT, ISSN 2348 – 9928Volume 1, Issue 2, June 2014

[8] Dhiraj Sangwan , Mahesh K. Yadav "Design and Implementation of Adder/Subtractor and Multiplication Units for Floating-Point Arithmetic" International Journal of Electronics Engineering, 2(1), pp. 197-203, 2010

[9] Tarek Ould Bachir, Jean-Pierre David "Performing Floating-Point Accumulation on a modern FPGA in Single and Double Precision" 18th IEEE Annual International Symposium on Field-Programmable Custom Computing Machines, pp.105-108, 2010

[10] Geetanjali Wasson "IEEE-754 compliant Algorithms for Fast Multiplication of Double Precision Floating Point Numbers" International Journal of Research in Computer Science, Volume 1, Issue 1, pp. 1-7, 2011

[11] KavithaSravanthi, Addula Saikumar "An FPGA Based Double Precision Floating Point Arithmetic Unit using Verilog" International Journal of Engineering Research & Technology ISSN: 2278-0181, Vol. 2 Issue 10, October - 2013

[12] Rathindra Nath Giri, M.K.Pandit "Pipelined Floating-Point Arithmetic Unit (FPU) for Advanced Computing Systems using FPGA" International Journal of Engineering and Advanced Technology (IJEAT), Volume-1, Issue-4, pp. 168-174, April 2012

[13] H. Yamada, T. Hottat, T. Nishiyama, F. Murabayashi, T. Yamauchi, and H. Sawamoto "A 13.3ns Double-precision Floating-point ALU and Multiplier", IEEE International Conference on Computer Design: VLSI in Computers and Processors, pp**.** 466 – 470, 2-4 Oct 1995

[14] Shrivastava Purnima, Tiwari Mukesh, Singh Jaikaran and Rathore Sanjay "VHDL Environment for Floating point Arithmetic Logic Unit - ALU Design and Simulation" Research Journal of Engineering Sciences, Vol. 1(2), pp.1-6, August - 2012

[15] Hwa-Joon Oh, Silvia M. Mueller, Christian Jacobi, Kevin D. Tran, Scott R. Cottier "A Fully Pipelined Single-Precision Floating-Point Unit in the Synergistic Processor Element of a CELL Processor" IEEE Journal of Solid-State Circuits, Vol. 41, No. 4, pp. 759-771, April 2006

[16] Jongwook Sohn, Earl E. Swartzlander "Improved Architectures for a Fused Floating Point Add-Subtract Unit" IEEE Transactions on Circuits and Systems—I: regular papers, Vol. 59, No. 10, pp. 2285-2291, October 2012