RESEARCH ARTICLE OPEN ACCESS

Design of 32-bit Floating Point Unit for Advanced Processors

Amana Yadav* Ila Chaudhary**
Electronics and Communication Department, Faculty of Engineering and Technology,
Manav Rachna International University, Faridabad

ABSTRACT
Floating Point Unit is one of the integral unit in the Advanced Processors. The arithmetic operations on floating point unit are quite complicated. They are represented in IEEE 754 format in either 32-bit format (single precision) or 64-bit format (double precision). They are extensively used in high end processors for various applications such as mathematical analysis and formulation, signal processing etc. This paper describes the detailed process for the computation of addition, subtraction and multiplication operations on floating point numbers. It has been designed using VHDL. The design has been simulated and synthesized to identify the area occupied and its performance in terms of delay.
Keywords: Arithmetic operations, Floating point, IEEE standard 754, VHDL

I. INTRODUCTION

The real numbers may be described informally as numbers that can be given by an infinite decimal representation, such as 2.48717733398724433.... The real numbers include both rational numbers, such as 56 and −23/129, and irrational numbers, such as π and the square root of 2, and can be represented as points along an infinitely long number line. They can have fixed point as well as floating point representation. Computation of floating point numbers needs advanced processing techniques. Advanced processors have dedicated floating point processor unit which is capable of performing arithmetic operations on real numbers with single precision (32-bits format)[1] or double precision(64-bits format).
Floating point notation is represented in the form as follows[2]:
$n = b^e * m$
where,
n = the number to be represented
b = base
m = mantissa

Value of b is
'2' for binary numbers
'8' for octal numbers
'10' for decimal numbers
'16' for hexadecimal numbers

In floating point arithmetic[3] user can round off the results of the computations as per his requirement but IEEE standard 754 defines the rules that lead to the same result of computation by rounding off. It prevents the existence of different results in different computations for the same input.

Single precision 32 – bit floating point format
32- bit floating point representation as per IEEE follows the standard shown in fig. 1

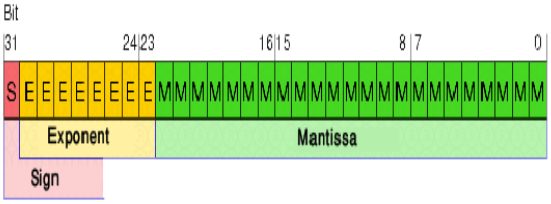

Fig. 1: IEEE 754 standard for single precision representation

The most significant bit starts from the left.
The number represented in the single precision format is
Value = (-1)s 2 e × 1.f (normalized) when E > 0 else
= (-1)s 2 -126 × 0.f (denormalized)
where,
f = (b23 -1 +b22 -2 + bi n +…+b0 -23) where bi n =1 or 0
s = sign (0 is positive; 1 is negative)
E = biased exponent; Emax=255 , Emin=0. E=255 and E=0 are used to represent special values.
e = unbiased exponent; e = E – 127(bias)

A bias of 127 is added to the actual exponent to make negative exponents possible without using a sign bit. So for example if the value 105 is stored in the exponent placeholder, the exponent is actually -22 (105 – 127). Also, the leading fraction bit before the decimal point is actually implicit and can be 1 or 0 depending on the exponent and therefore saving one bit. After the arithmetic computation of a number it is required to

normalize the result which means that MSB 1 is in the most left bit of the fractional part. It is done for representation of the number in only one way else same number can be written in hundreds of ways if kept in denormalized form.

Exceptions in floating point Unit

Various exceptions are defined by IEEE standard 754 which helps in implementing the arithmetic at the hardware level[5]. These exceptions are listed below:

Invalid operations: Some arithmetic operations are invalid, such as a division by zero or square root of a negative number. The result of an invalid operation shall be a NaN. There are two types of NaN, quiet NaN (QNaN) and signaling NaN (SNaN). They have the following format, where s is the sign bit:

QNaN = s 11111111 10000000000000000000000
SNaN = s 11111111 00000000000000000000001

Division by Zero

The division of a number (except zero) by zero gives infinity as a result. However, other arithmetic operations such as addition or multiplication may also give infinity as a result. The addition or multiplication of two numbers may also give infinity as a result. Therefore, to differentiate between the two cases, a divide-by-zero exception was implemented. Other exceptions that are defined by the IEEE standard are listed as **Inexact, underflow, overflow, infinity and zero.** Different rounding modes used are Round to nearest even, Round-to-zero, Round-up and Round-Down.

II. ARITHMETIC OPERATIONS

Addition / Subtractions[4]: A similar procedure is to be followed for the implementation of addition and subtraction. Hence, a single unit is used for these operations.

Table 1 shows an example of two operands considered for the computation.

Table1. IEEE standard 754 representation of the operands considered for the inputs

	Operand A	Operand B
Decimal Values	2.5	4.75
Sign bit	0	0
Exponent	10000000	10000001
Fraction	**1.**010…….0_2	**1.**00110…..0_2

Addition: Following are the steps followed for the addition of two floating point operands:

1. Finding the difference of exponents

Diff = 10000001 - 10000000 = 00000001

Exponent A is smaller than exponent B by 1-bit

2. Shifting the fraction of the operand A (smaller one) to the right

fraction A = 0.1010…..0_2

3. Incrementing the exponent A by '1' to equalize it to the larger one.
4. Adding the two fraction parts

$Fract_{out}$ = 0.1010…..0_2 + 1.00110….0_2 = 1.11010….0_2

5. Normalizing the result if necessary
6. Writing the result

Output = 0 10000001 11010……0_2

Subtraction:- For subtraction similar procedure is followed accept that the fraction part is subtracted. Fig. 2. Shows the flow chart for addition and subtraction of floating point numbers using floating point arithmetic.

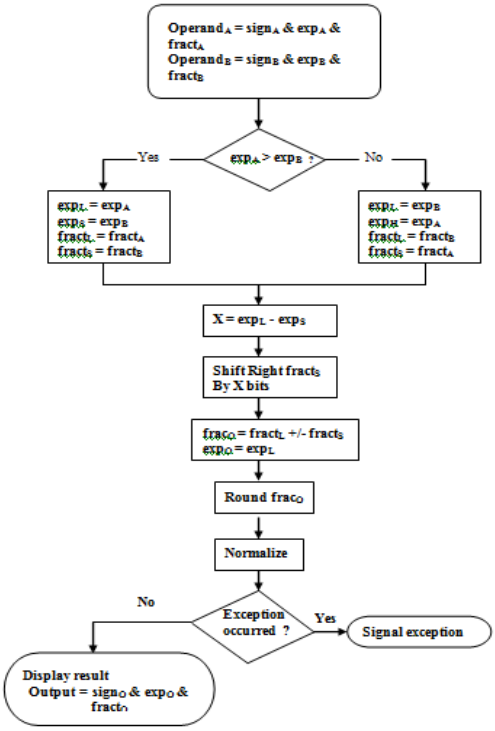

Fig. 2: Flow chart for addition / subtraction

Multiplication:

A separate block is to be provided for the multiplication of the floating point numbers. Steps followed for the process of multiplication are as follows:

1. Adding two exponents and subtracting the bias 127

Exponent out = 10000000 + 10000001 - 10000010

2. Multiplying the fraction out = 1.0011 * 1.01 by standard multiplication algorithm.

Result = 1.0111111

3. Normalize the result if necessary and give the output

Output = 0 10000010 0111110……..0_2

Fig. 3 shows the flow chart for the process of multiplication.

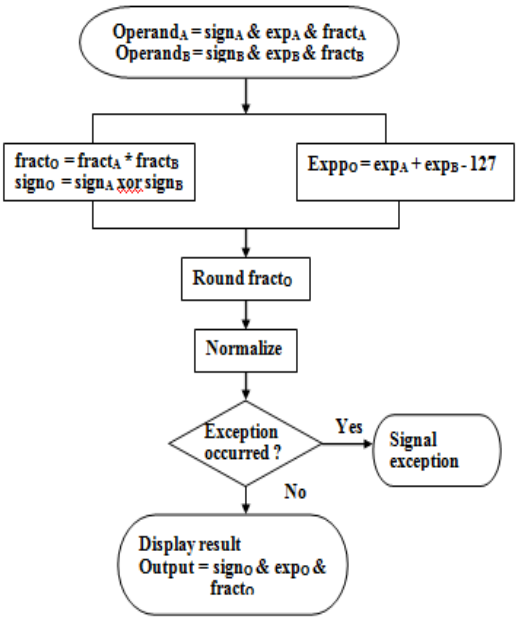

Fig. 3.: Flow chart for Multiplication

Fig. 4 shows the architecture of Floating Point Unit.

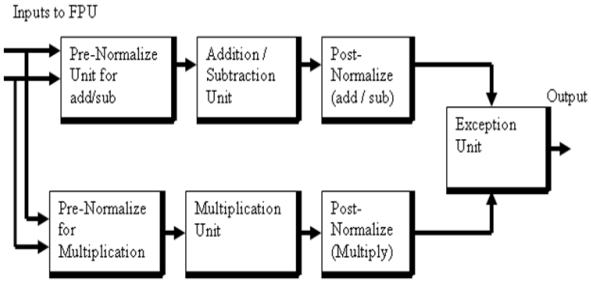

Fig. 4: Architecture of floating point Unit

III. IMPLEMENTATION

Fig. 5 shows main entity of Floating Point Unit.

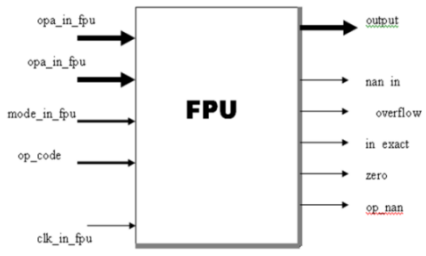

Fig. 5: Top level entity of FPU

Two input operands are forwarded to the pre-normalization unit. Also, other inputs like opcode telling the operation to be performed and rounding mode inputs are also send to the internal unit.

Pre-normalization unit for addition / subtraction

Function:- Input operands to the FPU are send to pre-normalization unit for add-sub. It performs the following functions.

1. It separates the exponent and the fraction part with 8-bits of exponent and 23 bits of fraction.
2. Fraction is expanded as:- Carry(1) & Hidden(1) & Fraction(23) & Guard(1) & Round(1) & Sticky(1). Hidden bit is '1' and three zeros added at the end that help to prevent the loss of data during rounding and shifting.
3. Checks which exponent is larger and finds there difference.
4. Sends larger exponent as the output.
5. Shifts the fraction part of the exponent to the right with the smaller exponent.
6. Sends the fractions to add / sub unit.

Fig. 6 shows the pre-normalize unit. Also, inputs/outputs are described in Table 2.

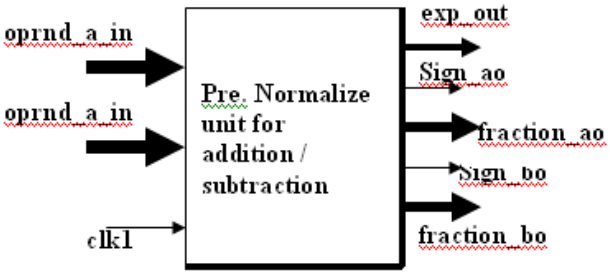

Fig.6: Pre-normalization unit

Table 2: Inputs / Outputs of Pre-Normalize Unit

Input / Output	No. of bits	Description
clk1		Acts as clock for this unit
oprnd_a_in	32	Input operand A to FPU by user for computation
oprnd_b_in	32	Input operand B to FPU by user for computation
Fraction_ao	28	Output fraction A to add / sub unit
Fraction_bo	28	Output fraction B to add / sub unit
exp_out	8	Larger exponent as output to post-normalize unit
Sign_ao	1	Sign bit of operand A
Sign_ao	1	Sign bit of operand B

Addition / Subtraction Unit:

Inputs / Outputs:- This unit takes the output of the pre-normalize unit as the input, performs addition or subtraction depending on the opcode. Table 3 describes the inputs / outputs of the unit and Fig. 7 shows them.

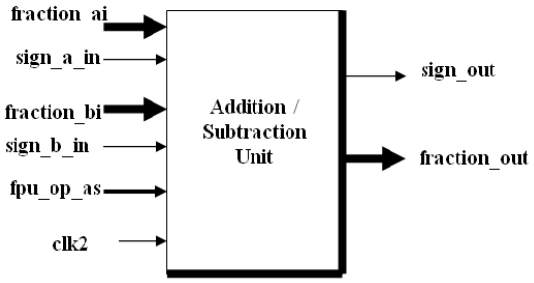

Fig. 7: Addition / Subtraction Unit

Table 3 Inputs / outputs of Addition / Subtraction Unit

Input / Output	No. of bits	Description
clk2		Act as clock for this unit
fpu_op_as	2	Opcode telling whether addition or subtraction to be performed
fraction_ai	28	Output fraction A from pre-normalize unit
fraction_bi	28	Output fraction B from pre-normalize unit
sign_a_in	1	MSB of the operand A i.e. sign bit
sign_b_in	1	MSB of the operand B i.e. sign bit
sign_out	1	Output sign of the result after computation
fraction_out	28	Fraction result

Post Normalization Unit:

Function:- The result obtained from the addition / subtraction unit is fed to the post-normalize unit shown in Fig. 8. Table 4 describes the inputs/ outputs of post-normalization unit.

It performs the normalization as follows

1. It counts the leading number of zeros in the fraction part starting from the hidden bit.
2. Decrements the exponent by the same number of bits.
3. Left shift the fraction by the same number of bits
4. Makes the hidden bit finally '1'
5. Takes the rounding mode decision depending on mode_in_fpu input to the main entity and performs rounding off of the fraction part.
6. The fraction part is truncated.
7. Sends the outputs to the exception unit.
8. Checks if any data has been lost during rounding.

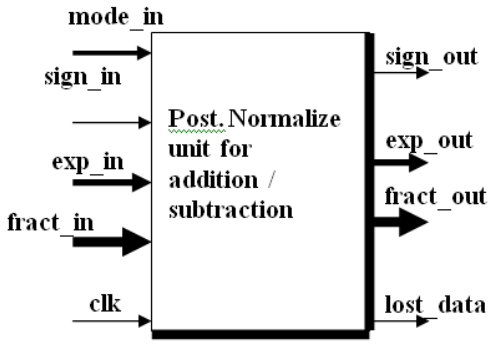

Fig. 8: Post Normalization Unit

Table 4: Inputs / Outputs Post-Normalization Unit

Input	No. of bits	Description
Clk		Clock input to this block
sign_in	1	Resulting sign bit from the addition subtraction unit
exp_in	8	Resulting exponent from pre-normalize unit
fract_in	28	Resulting fraction after computation from addition / subtraction
mode_in	2	Mode_in_fpu input given to it to take rounding decesions
fract_out	23	Normalized fraction result
exp_out	9	Exponent output (9 th bit is carry to check if the result is overflowing after incrementing
sign_out	1	Sign bit of the result
lost_data	1	Tells if any data has been lost during normalization

Pre-Normalize unit for multiplication:

The input operands to the FPU main entity are fed to the pre-normalization unit shown in Fig. 9 and are described in Table 5. It performs the following functions:-

1. Takes 32-bit input operands and separates the sign bit, exponent and fraction part as per IEEE-754 standard.
2. It recovers the hidden bit.
3. Changes the length of the exponents and adds them.
4. Subtract the bias of 127 from the exponent so that only 127 is the bias added to the result but not 254
5. Calculates the sign bit and sends the results to the multiplication unit.

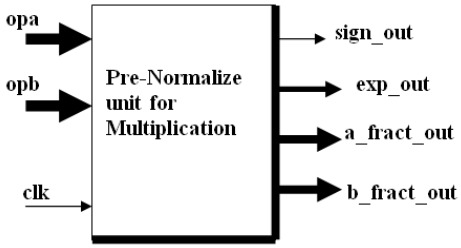

Fig. 9: Pre-Normalize unit for multiplication

Table 5: Pre-Normalize unit for multiplication

Input / Output	No. of bits	Description
clk3		Clock signal for this block
Opa	32	Input operand A of the main entity
Opb	32	Input operand B of the main entity
a_fract_out	24	Fraction A (with hidden bit)
b_fract_out	24	Fraction B (with hidden bit)
exp_out	10	Output exponent after adding them
sign_out	1	Resulting sign of two inputs

Multiplication Unit

It takes the fraction part from pre-normalize unit for multiplication unit as the output and gives the product as the output as shown in Fig. 10 and is described in Table 6.

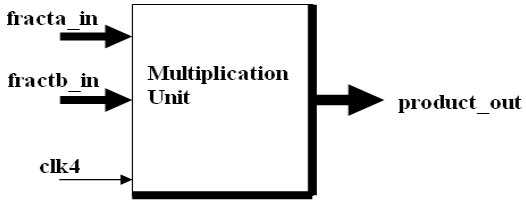

Fig. 10: Multiplication Unit

Table 6: Inputs / Outputs for Multiplication

Input / output	No. of bits	Description
clk4		Act as clock for this block
fracta_in	24	Fraction of operand A i.e. output of pre-normalize unit
fractb_in	24	Fraction of operand A i.e. output of pre-normalize unit
product_out	49	Product output of two fraction inputs

Post-normalization unit for multiplication

It is shown in Fig. 11 and is described in Table 7. It performs the following functions:-

1. Count the number of zeros starting from the left.
2. Decrements the value of exponent accordingly
3. Shifting the fraction part to the left by the number of zeros.
4. Rounding the result depending on the mode_in_fpu signal of the FPU unit.
5. Truncates the fraction part
6. Also, checks if there is any loss of data.
7. Sends the sign bit, exponent and fraction part and information about the loss of data to the exception unit.

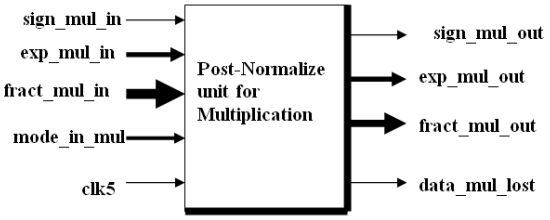

Fig. 11 Post-Normalize unit for Multiplication

Table 7: Inputs / Outputs for post-normalization of multiplication

Input / Output	No. of bits	Description
clk5		Act as clock to this unit
sign_mul_in	1	Output from the pre-normalization unit denoting the sign of the result
exp_mul_in	10	Exponent output from pre-normalize unit.
fract_mul_in	49	Product result of the multiplication unit
mode_in_mul	2	Rounding mode decisions are taken depending on its value.
sign_mul_out	1	Sign bit of the result
exp_mul_out	9	Exponent part of the result
fract_mul_out	23	Fraction part of the result
data_mul_lost	1	Data lost or not during truncation of the fraction part

Exception Unit

This unit shown in Fig 12 allows implementing the special values in the floating point unit and signals them whenever necessary i.e. invalid input or invalid operation. Foe example ∞ +/- ∞. The interface of the unit is described in Table 8.

Table 8 Inputs / Outputs of Exception Unit

Input / Output	No. of bits	Meaning
clk_in_exc		Act as clock for this unit
opa_in_exc	32	Operand A to check exception foe input
opb_in_exc	32	Operand B to check exception foe input
sign_in_exc	1	Sign bit of the result
exp_in_exc	9	Exponent of the result
fract_exc	23	Fraction part of the result
data_lost_exc	1	Data lost (output from post-normalization unit
fpu_op_exc	2	Opcode input to FPU
output_exc	32	Combined result after computatin
nan_in_exc	1	If the input is
zero_exc	1	If an input is NaN
in_exact_exc	1	If there is a loss of data
overflow_exc	1	If the result is exceeding the maximum limit
op_nan_exc	1	If an invalid operation is performed

Inputs to FPU Two 32-bits operands in IEEE-754 floating point format along with the opcode, rounding mode select and clock are given as inputs to FPU. Table below shows the inputs and their functioning.

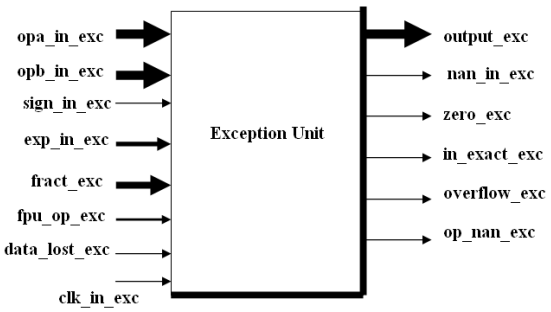

Fig. 12 Exception Unit

FPU Unit

Interface of top level entity is shown in Fig. 12 and are described in Table 9.1 and Table 9.2.

Table 9.1 Inputs of FPU

Input	No. of bits	Description	Values
clk_in_fpu		Give clock signal to FPU and all its internal blocks	
opa_in_fpu	32	Operand input A to the unit	
opb_in_fpu	32	Operand input B to the unit	
mode_in_fpu	2	Select rounding mode	"00" – Round up "01" – Round down "10" – Round to zero "11" – Round to even
op_code	2	Selects the operation	"00" – Addition "01" – Subtraction "10" – Multiplication "11" – Left for future use

Table 9.2 Output of FPU

Outputs	No. of bits	Meaning
Output	32	Result of arithmetic operation
nan_in	1	Either of the input is NaN
Overflow	1	Result exceeding maximum limit
in_exact	1	Loss of data while rounding
Zero	1	Result is zero in fraction part
op_nan	1	Invalid operation performed

IV. SIMULATION RESULTS

The design has been simulated and synthesized on Xilinx 13.1 ISE Design Suite. It has been synthesized on Vitex 5 FPGA module. Fig. 13.a, b, c, d shows the simulated waveform for the pre-normalized, addition / subtraction and post-normalized units, output of addition from FPU top module.

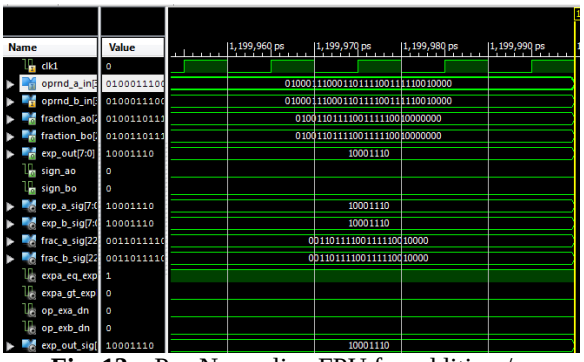

Fig. 13a. Pre-Normalize FPU for addition / subtraction

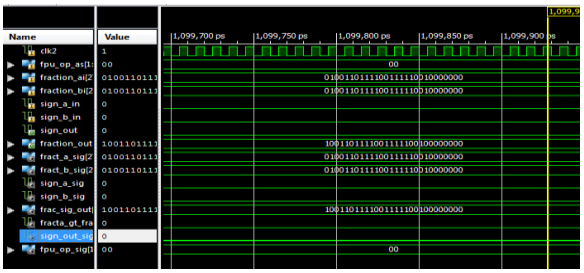

Fig. 13b: Addition / Subtraction Unit

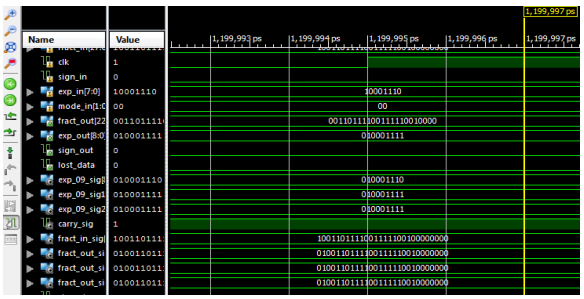

Fig. 13c Post-Normalize Unit for addition / Subtraction

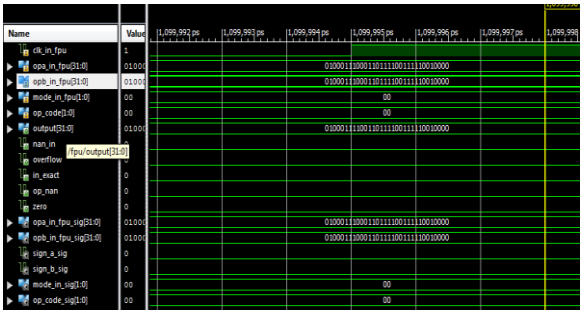

Fig. 13d: Output of FPU Top for addition

Similarly pre-normalize unit for multiplication, multiplication and post-normalize unit for multiplication have been implemented and final output of multiplication from FPU top have been simulated. Product output of two floating point numbers by FPU is shown in Fig. 14.

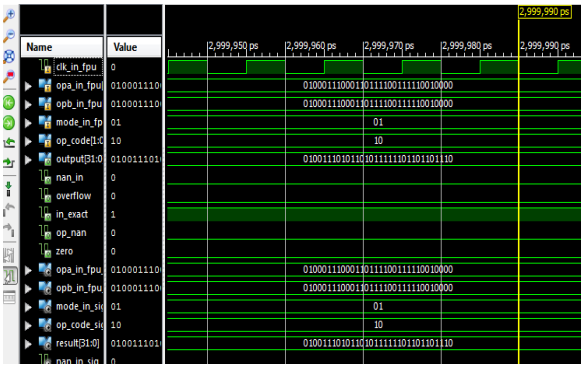

Fig. 14: Product output of Multiplier

Fig. 15a, b, c shows the schematic, synthesis report for device utilization summary and timing report respectively.

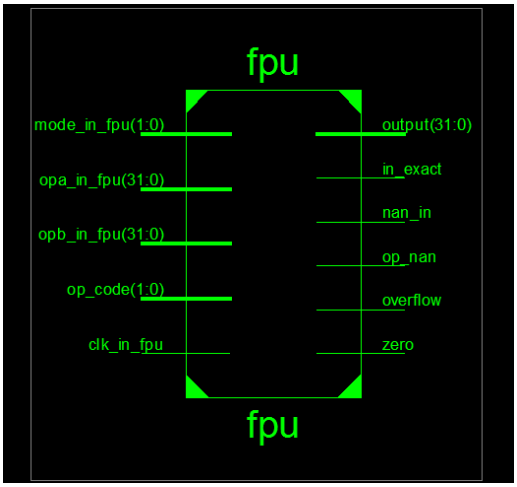

Fig.15a: Schematic of the FPU Unit

```
Device utilization summary:
---------------------------

Selected Device : 5vlx50tff1136-1


Slice Logic Utilization:
 Number of Slice Registers:             845  out of  28800     2%
 Number of Slice LUTs:                 2683  out of  28800     9%
    Number used as Logic:              2532  out of  28800     8%
    Number used as Memory:              151  out of   7680     1%
       Number used as SRL:              151

Slice Logic Distribution:
 Number of LUT Flip Flop pairs used:   2964
   Number with an unused Flip Flop:    2119  out of   2964    71%
   Number with an unused LUT:           281  out of   2964     9%
   Number of fully used LUT-FF pairs:   564  out of   2964    19%
   Number of unique control sets:        39

IO Utilization:
 Number of IOs:                        106
 Number of bonded IOBs:                106  out of    480    22%

Specific Feature Utilization:
 Number of BUFG/BUFGCTRLs:               1  out of     32     3%
```

Fig.15b: Device Utilization Summary of FPU

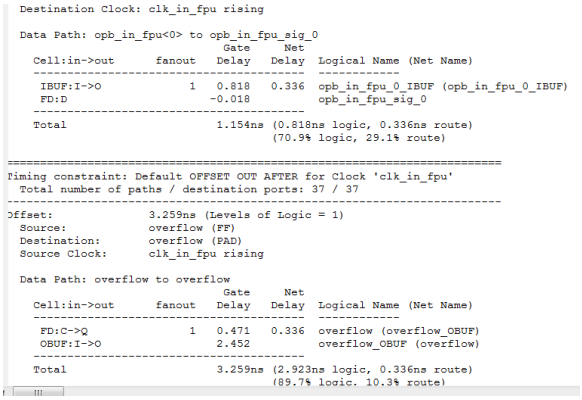

Fig. 15c: Timing Summary of FPU

Table 10 shows the hardware requirement of the design.

Table 10: Hardware Utilization Summary

S.No.	Slice Logic	Utilization
1	Number of Slice Registers	845 out of 28800 2%
2	Number of Slice LUTs	2683 out of 28800 9%
3	Number used as Logic	2532 out of 28800 8%
4	Number used as Memory	**151** out of 7680 1%
6	Number of IOs:	**106**
7	Number of BUFG/BUFGCTRLs	**1**

V. CONCLUSION

In this paper, floating point unit has been designed, simulated and then synthesized in order to obtain its performance in terms of the area occupied and delay on Vitex 5 FPGA Module. For the data path opb_in_fpu to opb_in_sig_0 total combinational logic delay and routing delay is 1.154ns and total overflow to overflow delay is 3.259ns. Hardware requirements have also been specified in the paper. Prenormalization and post-normalization units of the FPU can be further optimized to reduce the hardware requirement as well as delay.

REFERENCES

[1]. K. K. Lasith, Anoop Thomas "Efficient implementation of single precision floating point processor in FPGA", *Proceedings Emerging Research Areas: Magnetics, Machines and Drives (AICERA/iCMMD), 2014 Annual International Conference on* 4-26 July 2014, Kottayam, India

[2]. Seungchul Kim, Yongjoo Lee, Wookyeong Jeong "Low cost floating point arithmetic unit design", *ASIC, 2002. Proceedings. 2002 IEEE Asia-Pacific Conferenc*e on 8-8 Aug. 2002, Taipei, Taiwan, Taiwan.

[3]. Naresh Grover, M. K. Soni, "Design of FPGA based 32-bit Floating Point Arithmetic Unit and verification of its VHDL code 9 using MATLAB", *I. J. Information Engineering and Electronics Business, Jan. 2014, pp 1-14*

[4]. A. Malik, "Seok-Bum Ko, Effective implementation of floating-point adder using pipelined LOP in FPGAs," *Proceedings Electrical and Computer Engineering, 2005. Canadian Conference* on, vol., no., pp. 706–709, 1-4 May 2005.

[5]. Sayali A. Bawankar, Prof. G. D. Korde, "Review on 32 bit single precision Floating point unit (FPU) Based on IEEE 754 Standard using VHDL", International Research General of Engineering and Technology, Vol. 4, Issue 02, Feb. 2017, pp 1077-1082.