RESEARCH ARTICLE OPEN ACCESS

Design of Single Precision Floating Point Arithmetic Logic Unit

M.Bhavani*, G.Sirisha**, K.Siva Kumari***, B.Lalitha Rani****, A.Charishma*****
**(Assistant Professor, Department of Electronics and Communication Engineering, Bapatla Women's Engineering College, Bapatla)*
*** (Department of Electronics and Communication Engineering, Bapatla Women's Engineering College, Bapatla,)*
****(Department of Electronics and Communication Engineering, Bapatla Women's Engineering College, Bapatla,)*
*****(Department of Electronics and Communication Engineering, Bapatla Women's Engineering College, Bapatla,)*
******(Department of Electronics and Communication Engineering, Bapatla Women's Engineering College, Bapatla,)*

ABSTRACT
The main aim of Floating point Arithmetic logic unit (ALU) is presented that in stepwise design, all arithmetic operations like Addition, Subtraction, Multiplication and Division are combined to form a Floating point ALU unit.These operations are executed on 32-bit floating point numbers. Each operation is executes individual to each other.This unit uses the IEEE-754 single precision format.This paper presents the Design of 32-Bit floating point Arithmetic logic unit. The methods of Addition,Subtraction,Multiplication and Division are simulated Verilog HDL using Xilinx Software,14.7 Version.The logical method for Addition and Subtraction operation is expanded in order to decrease the no.of gates used.The results shows that the RTL view and Synthesis reports.

Keywords: Delay, Floating point number, no.of LUTS, Verilog, Xilinx.

Date of Submission: 24-07-2021 Date of Acceptance: 09-08-2021

I. INTRODUCTION

In the newest technology, Precision plays a major part in more applications like Digital signal processing. Floating point numbers [1] are used to represent noninteger fractional numbers and are used in most engineering and technical calculations.The most commonly used floating point standard is the IEEE Standard.According to this standard, floating point numbers are represented with 32bits(single precision). Floating point numbers are used in more applications are such as telecommunications, medical, imagining, radars etc.In this paper the operations are executed on 32-bit floating point numbers and the The logical method for Addition and Subtraction operation is designed to getting better performance which is required in signal computation applications. The design of floating point ALU is used to get the aim of small area. Then we use Verilog hardware description language (VHDL) [2].It is a user defined language .In VHDL we use two approaches to get better performance .In the two approaches, we use top down approach. Topdown approach means stepwise design. The HDL is used to characterize the performance of overall circuit with respect to speed and area. In the ALU, The main block of central processing unit (CPU) that handles all format have 32 bits to represent a floating arithmetic operations, logical operations etc.

The IEEE 754 floating point standard format has dividing into three main parts. They are Sign(1 bit), Mantissa(8 bits), exponent(23 bits).

IEEE 754 floating point format:

The IEEE 754 floating point format have 32 bits for representing a floating point number.

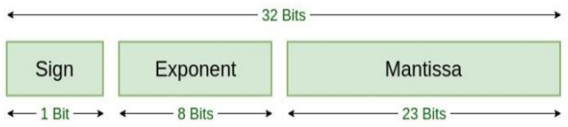


Fig 1: IEEE 754 floating point format

The standard format to represent floating point numbers which has three major parts as shown in figure1. They are sign, mantissa and exponent. The sign bit carries 1-bit where '1' and '0'represents a positive and negative number [2]. The mantissa is also known as floating point number or significant, it carries much no.of bits. which carries 23 bits. It represents precision bits of numbers [2]. The

II. OLDER DESIGNS

1. Floating point fused Add-Subtract unit

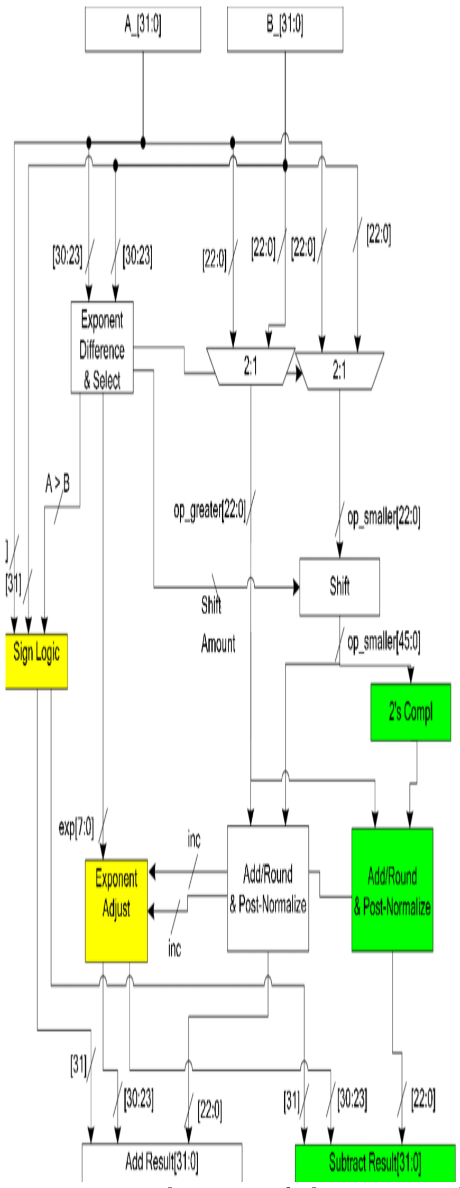


Fig 1: Computer Architecture of Floating point fused add subtract unit

The architecture of the fused add-subtract unit is shown in figure 1. It is derived from the floating-point add unit. The exponent difference, significant shift and exponent adjustment functions can be performed once with a single set In Fig 1 shows the architecture of the fused add-subtract unit, exponent carries 8 bits it represents both positive and negative [2]. Floating point numbers are of four types of exceptions. They are Overflow, Underflow, Division by zero, Invalid operation [3]. the blocks with white background are the same blocks used for a single floating-point add operation. The blocks with green background are additional blocks used to perform the subtract operation, and the blocks with yellow background are similar to the floating point add blocks, but with extended functionality to calculate the sign and exponent for the new subtract operation.

It detects the effective operation based on the signs of the two operands and the intended operation [4]. It also generates guard and pre-sticky bits that aid in the proper rounding of the final results. In a parallel conventional implementation of the fused add-subtract such as that two floating-point adders are used to perform the operation [4]. This approach is fast, however, the area and power overhead is large because two floating point add/subtract units are used.

In a conventional implementation of the fused add-subtract one floating-point adder/Subtractor is used to perform the operation in addition to a storage element to store the addition or subtraction result. This approach is very efficient in terms of area. However, due to the serial execution of both operations, the time needed to get both results is twice the time needed by the parallel approach. Also since a storage element is used, it adds slightly to the area and power overhead, through two floating-point adders operating.

Two inputs of 32 bits are applied at the input of the unit. This unit passes 23 bit mantissa of both the inputs to a multiplexer. The 8 bit exponents are also given to an Exponent Comparison block. This comparison block generates a control signal to pass one of the mantissa of lower exponent value to the shift register. Therefore for normalizing, the result is given to a Rounding/Normalizing block. The circuit has Rounding/Normalizing block for normalizing the resulting number. In addition/subtraction block, the result may overflows and generates one carry/borrow bit. The rounding is done by shifting the result to 1 bit right and raising the exponent by 1. The exponent is raised in the exponent adjustment block. The final rounded result, adjusted exponent and the final sign bit are passed to the output of the add-subtract unit i.e. sum and difference.

2. Floating point fused Dot-Product unit

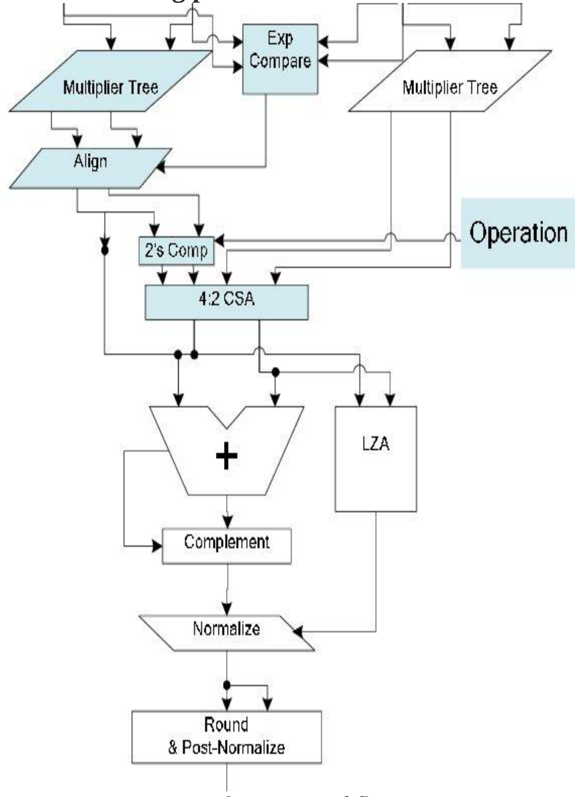


Fig 2: Computer architecture of floating point Dot-Product unit

The architecture of the fused dot-product unit is shown in figure 2. It is derived from the floating-point add unit. The exponent difference, significand shift and exponent adjustment functions can be performed once with a single set of hardware, with the results shared by both the add and the subtract operations in fig.3. New add and normalize blocks are needed for the new subtract operation. It shows the architecture of the fused add-subtract unit, the blocks with white background are the same blocks used for a single floating-point add operation. The blocks with green background are additional blocks used to perform the subtract operation, are similar to the floating point add blocks, but with extended functionality to calculate the sign and exponent for the new subtract operation. Since two operations are explicitly performed for sum and difference results (e.g., if the addition is used for the sum, the subtraction is used for the difference), the addition and subtraction are separately placed and only one LZA and normalization (for the subtraction) is required.

Assuming both sign bits are positive, the addition and subtraction are performed separately. Then, two multiplexers select the sum and difference with the operation decision bit, which is the XOR of the two sign bits. This will realize their Dot-product format of multiplication and sum them again to make as FDP for better than serial implementation. This FDP will increase the efficiency of FFT implementation.

In the FDP unit that is shown in above figure, a multiplier tree, an aligner in addition to 4:2 reduction tree are added to a conventional FPM to perform the dot-product operation. The remaining components of the FPM are used as is which results in a significant area reduction compared to the conventional implementation [5]. Although it is not especially attractive, a system could use this unit to replace a floating-point adder and a floating-point multiplier. If operands B and D are set to one, then the unit will perform addition only with simple data forwarding multiplexers for operands A and C to skip the multiplication trees. The speed of the addition will be one multiplexer delay more than a discrete floating-point adder [5].

In these both architectures cannot peform all arithmetic and logical operations. So we can perform all arithmetic operations, We can use Arithmetic logic unit (ALU).

III. PROPOSED METHOD

1. Architecture

Arithmetic unit performs Arithmetic operations on floating point numbers consist of addition, subtraction, multiplication and division. The operations are done with algorithms similar to those used on sign magnitude integers (because of the similarity of representation) — example, only add numbers of the same sign. If the numbers are of opposite sign, must do subtraction.

The Addition, Subtraction, Multiplication and division have been executed by the 32-bit floating point ALU as shown in figure 1. Pre Normalization blocks have been used. First block is used for the addition/subtraction and other for multiplication/division operations [6] . The Mantissa part has been normalised by the post normalization unit [6]. Then the final result is obtained. Two IEEE 754 32 bit operands have been taken to do the arithmetic operations. Eventually the exceptions occurred have been detected and handled by using handling. The Exception handling are of two types. They are one is Over flow and another one is Underflow. First one is overflow that occurs in Addition and underflow occurs in Subtraction.

Overflow is said to occur when the true result of an arithmetic operation is finite is said to occur when the true result of an arithmetic operation is finite but larger in magnitude than the largest floating point number which can be stored using the given precision. Underflow is said to occur when the true result of an arithmetic operation is smaller in magnitude (infinitesimal) than the smallest normalized floating point number which can be stored. Overflow can't be ignored in calculations whereas underflow can effectively be replaced by zero. These exceptions are handled by using exception handling block. If there are no exceptions then we get final result. In such cases, the result must be *rounded* to fit into the available number of M positions. The extra bits that are used in intermediate calculations to improve the precision of the result are called *guard bits*. It is only a tradeoff of hardware cost (keeping extra bits) and speed versus accumulated rounding error, because finally these extra bits have to be rounded off to conform to the IEEE standard.

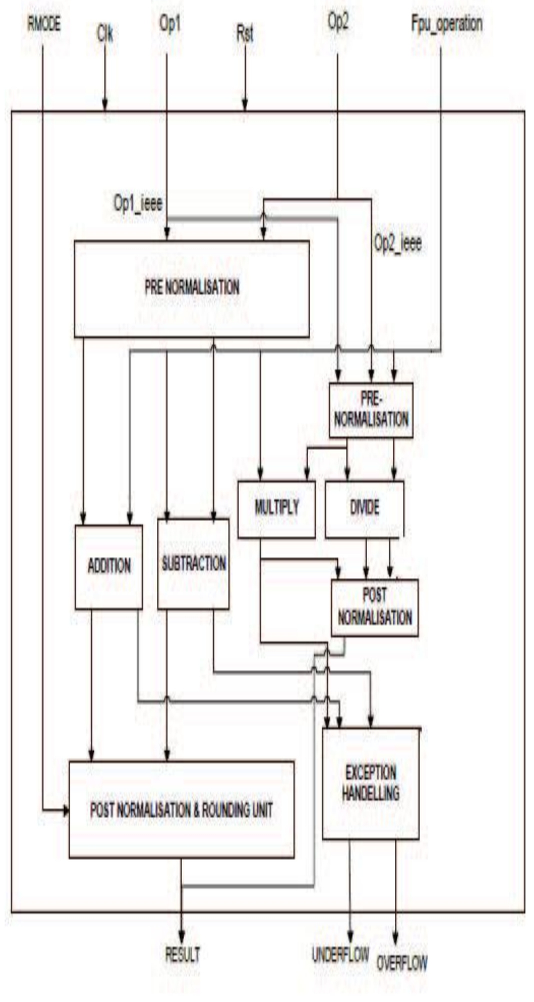


Fig 1: Floating point ALU Architecture

2. Flowcharts

2.1 Addition and Subtraction

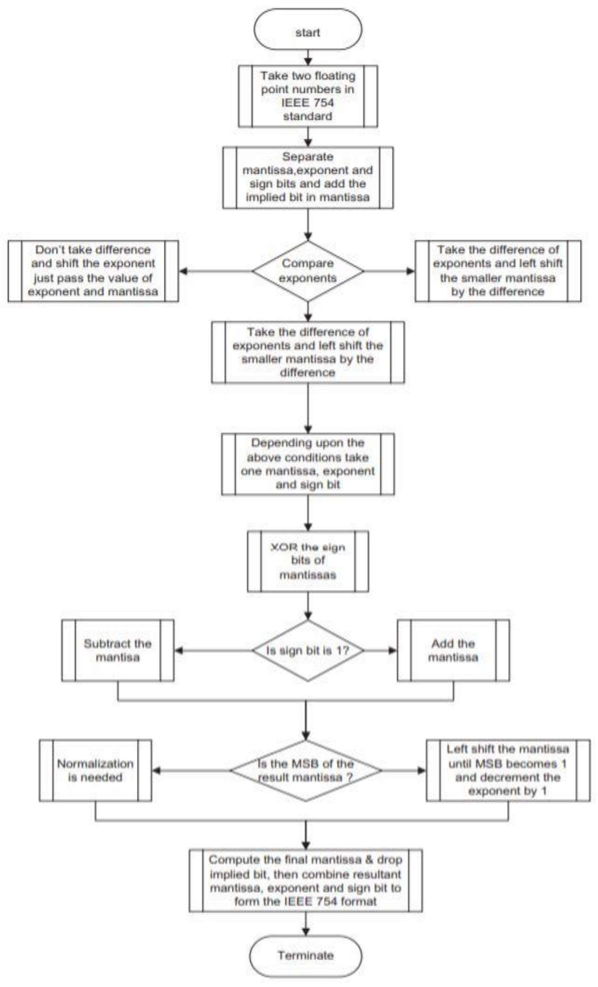

start
Take two floating point numbers in IEEE 754 standard
Separate mantissa,exponent and sign bits and add the implied bit in mantissa
Don't take difference and shift the exponent just pass the value of exponent and mantissa
Compare exponents
Take the difference of exponents and left shift the smaller mantissa by the difference
Take the difference of exponents and left shift the smaller mantissa by the difference
Depending upon the above conditions take one mantissa, exponent and sign bit
XOR the sign bits of mantissas
Subtract the mantissa
Is sign bit is 1?
Add the mantissa
Normalization is needed
Is the MSB of the result mantissa ?
Left shift the mantissa until MSB becomes 1 and decrement the exponent by 1
Compute the final mantissa & drop implied bit, then combine resultant mantissa, exponent and sign bit to form the IEEE 754 format
Terminate

Fig 2.1: Flow chart for Addition and Subtraction

The flow chart for Addition and Subtraction as shown in figure 2.1. Here duplet scopes be allowed transpire occupy measure. One and other ivalues exist matching indication, Both MSB bit perhaps to '1' or '0' (Balnace of contrasy forces).Although the pair of ivalues exist non identical indications, especially be MSB of only integer added is '1' (positive) and the MSB of other is '0' (negative).In the first place require to inspect effective indication of twain numbers if the signal of pair digits possess are non identical execute two's complement for the MSB number having '1'.Behind Adding performance takes place carry out one and the other digits.

First step is to take two floating point numbers using IEEE 754 standards. Now we separate mantissa, exponent and sign bits. Next we go to exponent compare circuit. It performs the comparision operation of two exponents. If we have a exponent difference we perform left shift operation, Otherwise just pass the values of exponent and mantissa. There are two possibilities that may occur while computing two 32-bit floating point numbers. By depending on sign bit, we perform arithmetic operation. If both operands are same sign, Their MSB could be either ‘1’ or ‘0’ (Positive or Negative). When both operands are of different signs, that is MSB of one operand is ‘1’ (Positive) and the MSB of other is ‘0’ (negative).We need to check the sign of two floating point numbers. If the sign of both numbers are different, then we perform one’s complement plus one operation for having MSB is ‘1’. After Addition operation is executed. If we get any exceptions like overflow or underflow .Then overcome the exceptions by left shift the mantissa until MSB becomes ‘1’ and decrement the exponent by 1. Finally then combine the resultant mantissa, exponent and sign bit to from the IEEE 754 format.

2.2 Multiplication

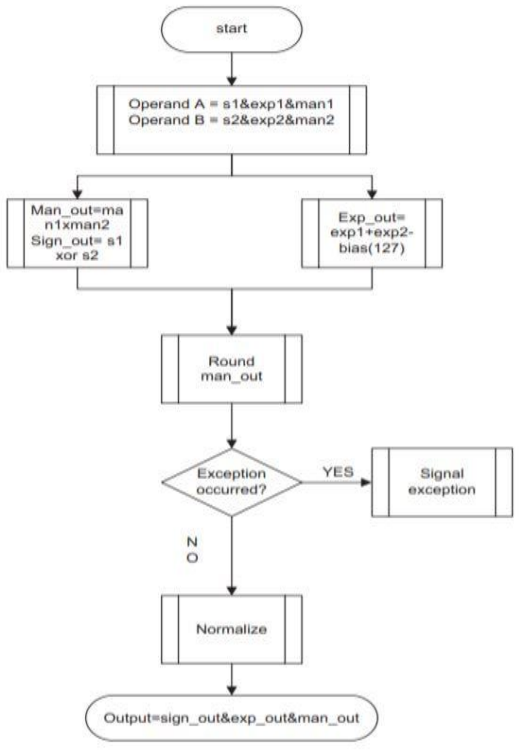


Fig 2.2: Flowchart for Multiplication

The algorithm of floating point multiplication is shown in fig 2.2. First step is to take the two floating point numbers using IEEE 754 standard. We have to check whether the multiplicand, multipliers and or zeros or not. We perform Addition and Subtraction operations, and then align the mantissas. In such a way that both the exponents are equal. In multiplication algorithm, there is no need to align the mantissas. To get exponent output, add the two exponents and the bias 127 is subtracted. The ex-or operation is performed on the MSB bits to get the sign bit. Multiply the mantissa parts of the two numbers. We used fixed point signed magnitude multiplication algorithm. Check the exception flags are overflow, underflow have been determined. Overflow means the corresponding register cannot store the additional bit. After normalisation we got the final output.

2.3 Division

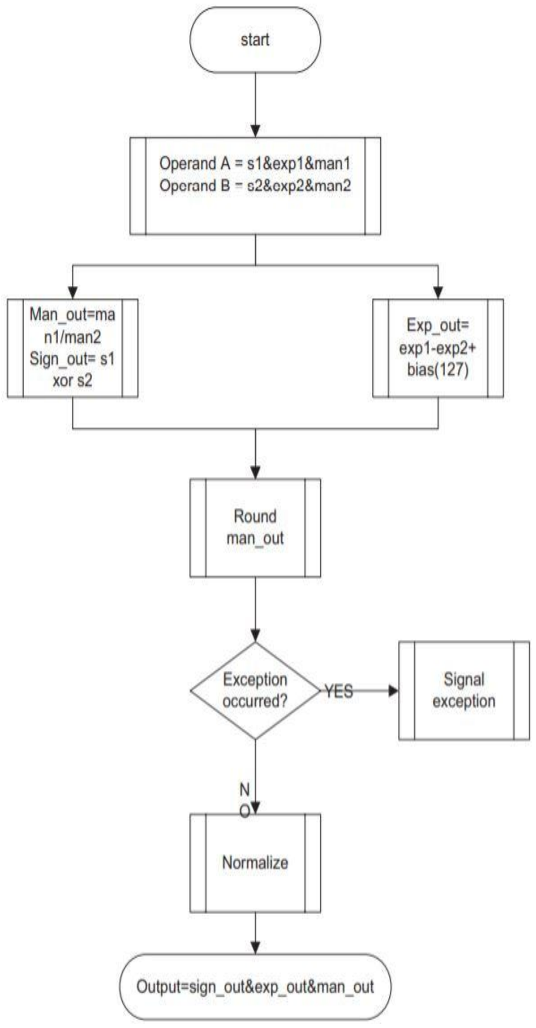


Fig 2.3 :Flowchart for division

The flowchart for division as shown in above figure 2.3.First step is to take the two floating point numbers using IEEE 754 standard. We check zeros or not. If BR is zero then we cannot possible to perform division. If divisor is zero, we get division by zero problem. Report the error message as divide by zero. If BR is not equal to zero, that means divisor is zero. Now check the value of dividend. It is present in Ac register. Suppose dividend value is zero, then result is also zero. No need to continue the process. Otherwise perform the operation. Here QS is nothing but the size of quotient and depends upon dividend sign and divisor sign as same or different. It performs exclusive or operation on dividend sign and divisor sign. If they are same then the result is zero and they are different, result will be one. Next we have to check any overflow or underflow. So for that purpose we have to subtract division from the dividend. In order to perform the subtraction operation, we need to add two's complement of the division to dividend. To overcome the overflow, we perform shift right operation. To get exponent output, Subtract the two exponents and the bias 127 is added. The ex-or operation is performed on the MSB bits to get the sign bit. Check the exception flags and overflow or underflow have been determined. Divide the mantissa parts of the two numbers. We have to follow restoring algorithm. After normalisation we got the final output.

IV. SIMULATION RESULTS AND PERFORMANCE ANALYSIS

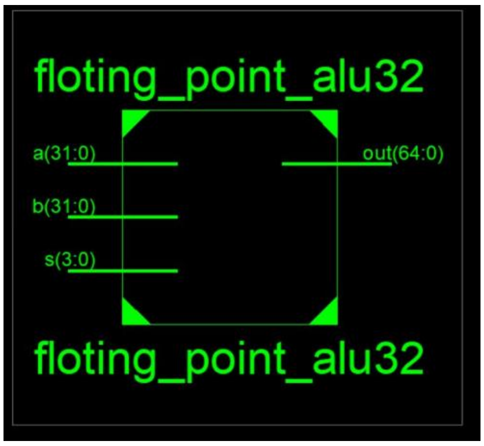


Fig 1: RTL view of 32-bit floating point ALU

The above figure 1 show the Register Transfer Level (RTL) of Floating point ALU. It describes how data is transformed as it is passed from register to register .The transforming of the data is performed by the combinational logic that exists between the registers.

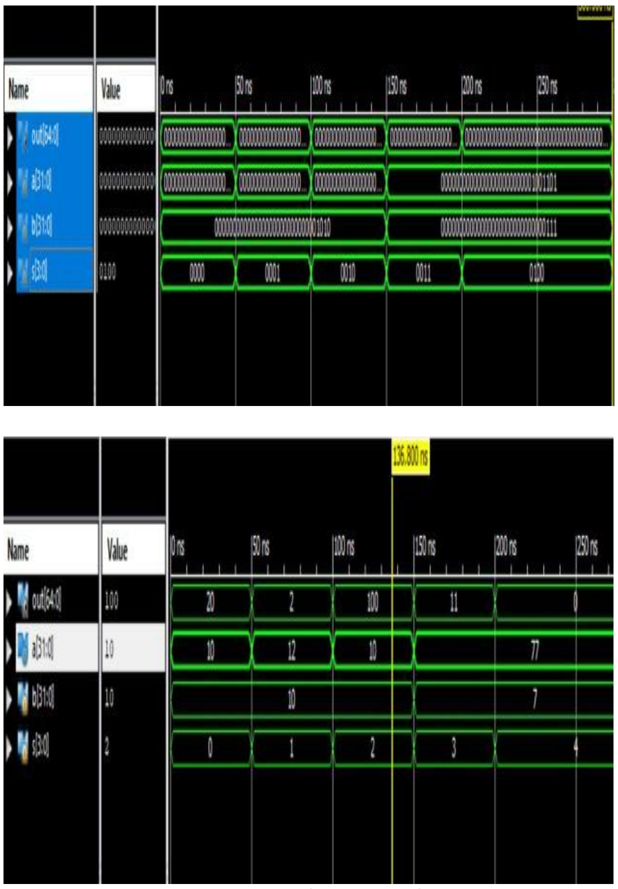
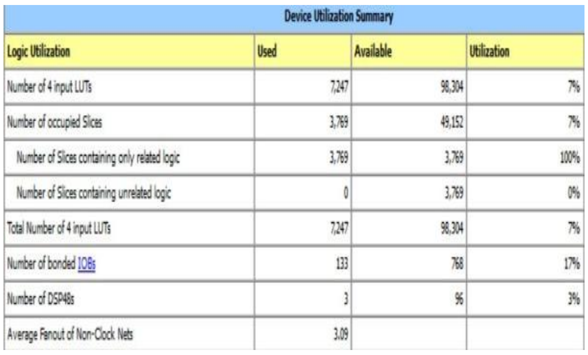


Fig 2: Simulation Outputs for 32-bit floating point ALU

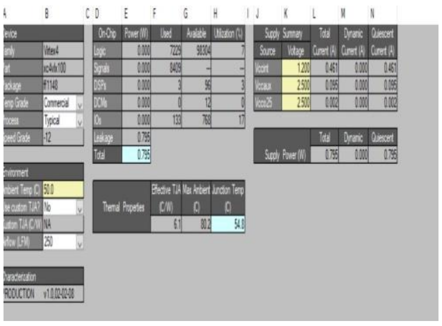
The above figures 2 shows the simulation output for 32- bit flaoting point ALU, is represented in binary and unsigned decimal form. By depending on selection lines to find the output.



Device Utilization Summary			
Logic Utilization	Used	Available	Utilization
Number of 4 input LUTs	7,247	98,304	7%
Number of occupied Slices	3,769	49,152	7%
Number of Slices containing only related logic	3,769	3,769	100%
Number of Slices containing unrelated logic	0	3,769	0%
Total Number of 4 input LUTs	7,247	98,304	7%
Number of bonded IOBs	133	768	17%
Number of DSP48s	3	96	3%
Average Fanout of Non-Clock Nets	3.09		

Fig 4: Device utilisation of 32-bit floating point ALU

The above figure 4 shows the Device utilisation of 32-bit floating point ALU, it is presented that how much area is occupied . It takes lesser area compared to older designs. Mainly this figure 4 shows the no.of LUTs are 7,247. It is the main aim to use Floating point ALU.



evice	
amily	Virtex4
art	xc4vlx100
ackage	ff1148
emp Grade	Commercial
rocess	Typical
peed Grade	-12

On-Chip	Power (W)	Used	Available	Utilization (%)
Logic	0.000	7229	98304	7
Signals	0.000	8409	--	--
DSPs	0.000	3	96	3
DCMs	0.000	0	12	0
IOs	0.000	133	768	17
Leakage	0.795			
Total	0.795			

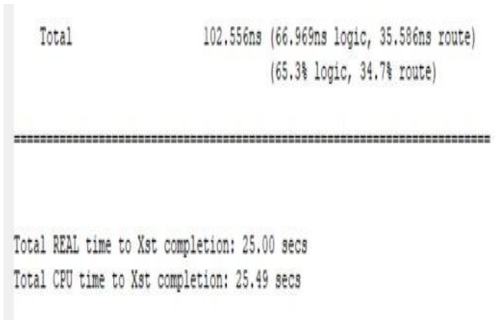
Supply Summary		Total	Dynamic	Quiescent
Source	Voltage	Current (A)	Current (A)	Current (A)
Vccint	1.200	0.461	0.000	0.461
Vccaux	2.500	0.095	0.000	0.095
Vcco25	2.500	0.002	0.000	0.002

	Total	Dynamic	Quiescent
Supply Power (W)	0.795	0.000	0.795

nvironment	
mbient Temp (C)	50.0
se custom TJA?	No
ustom TJA (C/W)	NA
irflow (LFM)	250

Thermal Properties	Effective TJA (C/W)	Max Ambient (C)	Junction Temp (C)
	6.1	80.2	54.8

haracterization	
RODUCTION	v1.0,02-02-08



```
Total                     102.556ns (66.969ns logic, 35.586ns route)
                                    (65.3% logic, 34.7% route)

=========================================================================


Total REAL time to Xst completion: 25.00 secs
Total CPU time to Xst completion: 25.49 secs
```

Fig 5: Power and Delay analysis of 32 bit floating point ALU.

The above figure 5 shows the power and Delay analysis of 32- bit floating point ALU, is presented that power is 0.795W and delay is 102.556ns.

V. CONCLUSION

The implementation of a 32-bit floating point arithmetic logic unit is done and efficient algorithm for addition and subtraction is implemented in order to reduce the no.of gate used. For future enhancement, we use pipelining technique**.**

ACKNOWLEDGEMENTS

We sincerely thank to my project guide, who helped me in all aspects of my project to complete in short term. We also thank Bapatla Women's Engineering College for providing necessary facilities towards carrying out this work.

REFERENCES

[1]. Shanthala. N1, Nayana. M, Chandrashekar.C, Dr. Siva Yella mp al li "Basic operation performed on Arithmetic Logic Unit (ALU) For 32-Bit Floating Point Numbers", *International Journal of Applied Engineering Research* ISSN 0973-4562 Volume 12, Number 12 (2017) pp. 3248-3252 Research india Publications. http://www.ripublication.com

[2]. Swathi.A, G.Srinivasulu "ASIC implementation of a High speed double Presicion(64) floating point unit using verilog", International journal and magazine of engineering, technology, management and research ISSN 2348-4845

[3]. Dave Omkar R, Aarthy M, "ASIC implementation of 32 and 64 Bit floating point ALU using Pipelining", International Journal of computer applications(0975-8887) volume 94-No. 17,May 2014

[4]. H.H. Saleh, ―H.Fused Floating-Point Arithmetic for DSP,‖ PhD dissertation,Univ. of Texas, 2008.

[5]. Jorge Tonfat, Ricardo Reis, ―Improved Fused Floating Point Add-Subtract and Multiply-Add Unit for FFT Implementation‖, in 2014 2nd International Conference on Devices, Circuits and Systems (ICDCS).

[6]. Ushasree G, R Dhanabal, Dr Srat Kumar sahoo, "VLSI implementation of a High Speed Single Precision Floating Point Unit Using Verilog", Proceedings of 2013 IEEE conference on information and communication technologies(ICT 2013)

[7]. Naresh Grover, M,K Soni, "Design of FPGA based 32-bit Floating Point Arithmetic Unit And verification of its VHDL code using MATLAB", I.J Information Engineering and Electronics Buisiness, 2014,1,1-14 published Online February in MECS

[8]. Sayali A. Bawankar, Prof. Girish. D. Korde," Design and Simulation of Floating Point Adder,Subtractor & 24-bit Vedic Multiplier", International Jornal for Research in Applied Science &Multiplier", International Journal for Research in Applied Science & Engineering Technology (IJRASET) ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor:6.887 Volume 5 Issue V11, July 2017-Available at www.ijraset.com

[9]. Prashanth B, P.Anil Kumari, G Sreenivasulu," Design & Implementation of Floating point ALU on a FPGA Processor", 2012 International Conference on Computing on Computing, Electronics and Electrical Technologies[ICCEET]

[10]. Manisha Sangwan, A Anita Angeline, "Design and Implementation of Single Precision Pipelined Floating Point Co-Processor", 2013 International Conference on Advanced Electronic Systems(ICAES).