International Journal of Engineering Technology and Management Sciences
Website: ijetms.in Issue: 1 Volume No.6 January – 2022 DOI: 10.46647/ijetms.2022.v06i01.003
ISSN: 2581-4621

VLSI IMPLEMENTATION OF HIGH SPEED SINGLE PRECESSION FLOATING POINT UNIT USING VERILOG

NARAHARI BHARGAVI [1], B NAGA RAJESH [2]
[1] 192T1D3805 M.Tech DECS, Dr.KVSRECW, JNTUA,Affiliated, Kurnool, Andhra Pradesh India
[2]Assistant Professor, Dept of ECE, Dr.KVSRECW, JNTUA,Affiliated, Kurnool, Andhra Pradesh India
[1]naraharibhargavi555@gmail.com
[2]nagarajesh87@gmail.com

Abstract— **Single-precision floating-point format is a computer number format that is used to represent a wide dynamic range of values. Floating point numbers representation has widespread dominance over fixed point numbers. Since the recent years, researchers are putting a lot of efforts in interfacing complex modules which are used in signal processing with processors for increasing the speed. In this work implementation of a floating point arithmetic unit which can perform addition, subtraction, multiplication, and division functions on 32-bit operands that use the IEEE 754-2008 standard is done using Verilog. The FPU of this work is a single precision IEEE754 compliant integrated unit. Pre-normalization of operands is employed for addition and subtraction, multiplication using bit pair recoding and division using non restoring division. It can handle not only basic floating point operations like addition, subtraction, multiplication and division but can also handle operations like transcendental functions like sine, cosine and tangential function. The logical method for Addition and Subtraction operation is expanded in order to decrease the no. of gates used.**

Keywords: Floating Point Unit, IEEE 754, Pre-Normalization, Bit Pair Recoding, Non Restoring Division Arithmetic Unit.

I INTRODUCTION

Floating-point calculation is considered to be an esoteric subject in the field of Computer Science. This is obviously surprising, because floating-point is omnipresent in computer systems. Floating-point (FP) data type is almost present in every language. From PCs to supercomputers, all have FP accelerators in them. Most compilers are called from time to time to compile the floating-point algorithms and virtually every OS have to respond to all FP exceptions during operations such as overflow. Also FP operations have a direct effect on designs as well as designers of computer systems. So it is very important to design an efficient FPU such that the computer system becomes efficient. Further, FPU can be improvised by using efficient algorithm for the basic as well as transcendental functions, which can be handled by any FPU, with reduced complexity of the logic used. This FPU further can be worked upon to improvise further complex operations-viz. exponent, etc. It can be designed so that it can handle different data types like character, strings etc, can serve as a backbone for designing a fault tolerant IEEE754 compliant FPU on higher grounds and such that pipeline can be implemented. When a CPU executes a program that is calling for a floating-point (FP) operation, there are three ways by which it can carry out the operation. Firstly, it may call a floating-point unit emulator, which is a floating-point library, using a series of simple fixed-point arithmetic operations which can run on the integer ALU. These emulators can save the added hardware cost of a FPU but are significantly slow. Secondly, it may use an add-on FPUs that are entirely separate from the CPU, and are typically sold as an optional add-ons which are purchased only when they are needed to speed up math-intensive operations. Else it may use integrated FPU present in the system. IEEE754 standard is a technical standard established by IEEE and the most widely used standard for floating-point computation, followed by many hardware CPU and FPU and software implementations. Single-precision floating-point format is a computer number format that occupies 32 bits in a computer memory and represents a wide dynamic range of values by using a floating point. In IEEE 754-2008, the 32-bit with base 2 format is officially referred to as single precision or binary32. It was called single in IEEE 754-1985. The IEEE 754 standard specifies a single

International Journal of Engineering Technology and Management Sciences
Website: ijetms.in Issue: 1 Volume No.6 January – 2022 DOI: 10.46647/ijetms.2022.v06i01.003
ISSN: 2581-4621

precision number as having sign bit which is of 1 bit length, an exponent of width 8 bits and a significant precision of 24 bits out of which 23 bits are explicitly stored and 1 bit is implicit 1.

.

II PROBLEM DEFINITION

As the efficiency of the FP operation carried out by the FPU is very much responsible for the efficiency of the Computer System, It is very much necessary to implement not only efficient algorithms, but to reduce the memory requirement, reduce the clock cycles for any operations, and to reduce the complexity of the logic used. In the path to make a better and efficient FPU, we have tried to use the preexisting efficient algorithms and incorporate few changes in them or combine different positive aspects of already existing algorithms. This has resulted in positive and better or at least comparable results than that of preexisting FPUs results of which has been provided in the last chapter.

32 bits		
sign	exponent	mantissa
1-bit	8-bits	23-bits

Figure 1 : IEEE 754 single precision format

The Floating Point Arithmetic unit consists the blocks mentioned in below figure.

.

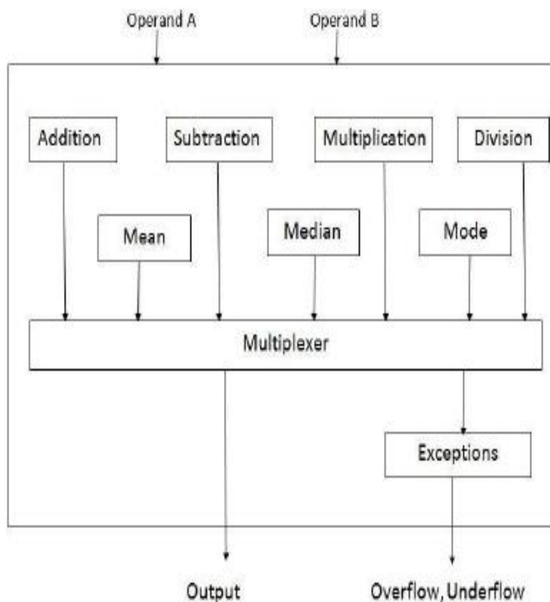


Figure 2: Floating Point Arithmetic Unit

The flowchart representation of Addition algorithm is shown in figure3, we had implemented block CLA where output carry of one block is input to the other adder block. Subtraction can be interpreted as addition of a positive and a negative number. So using the same algorithm as that of addition, we can complete the subtraction operation by taking complement of the negative number and adding 1 to the complement. This is same as taking the 2"s complement of the negative number. Doing this we interpreted the negative number as positive and carry the addition operation.

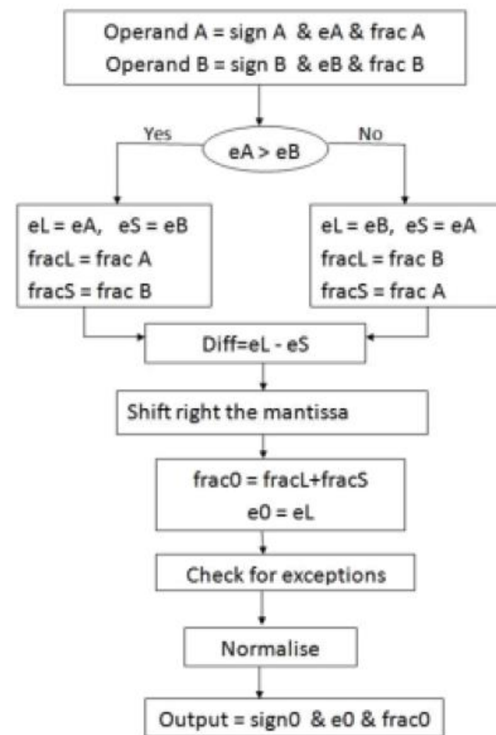


Figure 3: Addition algorithm flow chart

The flowchart representation of Multiplication algorithm is shown in figure4 Multiplication of negative number using 2"s complement is more complicated than multiplication of a positive number. This is because performing a straightforward unsigned multiplication of the 2's complement representations of the inputs does not give the correct result. Multiplication can be designed in such that it first converts all their negative inputs to positive quantities and use the sign bit of the original inputs to determine the sign bit of the result. But this increases the time required to perform a multiplication, hence decreasing the efficiency of the whole FPU. Here initially we have used Bit Pair Recoding algorithm which increases the efficiency of multiplication by pairing. To further

Website: ijetms.in Issue: 1 Volume No.6 January – 2022 DOI: 10.46647/ijetms.2022.v06i01.003
ISSN: 2581-4621

increase the efficiency of the algorithm and decrease the time complexity, we have combined the Karatsuba algorithm with the bit pair recoding algorithm

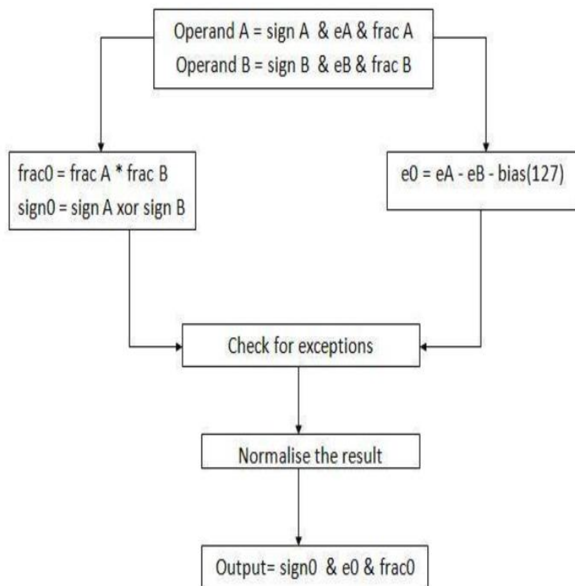


Figure 4: Multiplication algorithm flow chart

The flowchart for division algorithm is shown in figure 5.

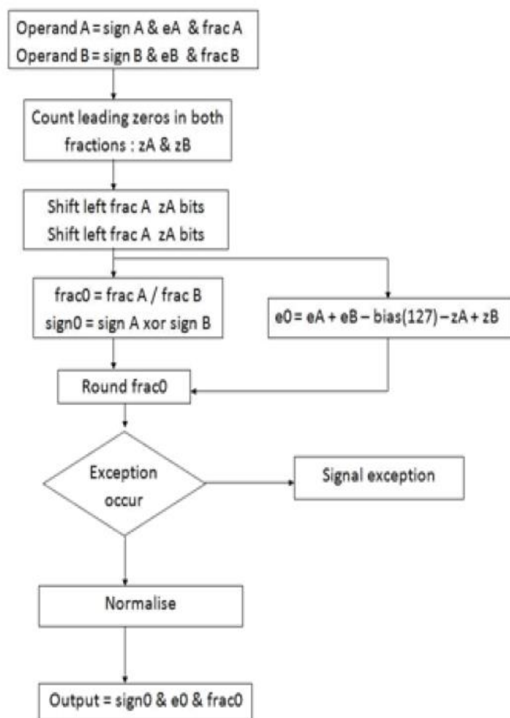


Figure 5: Division algorithm flow chart

The division that has been used in our FPU is based on the Non-restoring division algorithm. It is considered as a sequence of addition or subtraction and shifting operations. Here, correction of the quotient bit, when final remainder and the dividend has different sign, and restoration of the remainder are postponed to later steps of the algorithm, unlike restoration division. In this algorithm, restoration of the operation is totally avoided. Main advantage of this NRD algorithm is the compatibility with the 2"s complement notation used for the division of negative numbers.

III IMPLEMENTATION

The architecture and methodology implemented is shown in figure6.

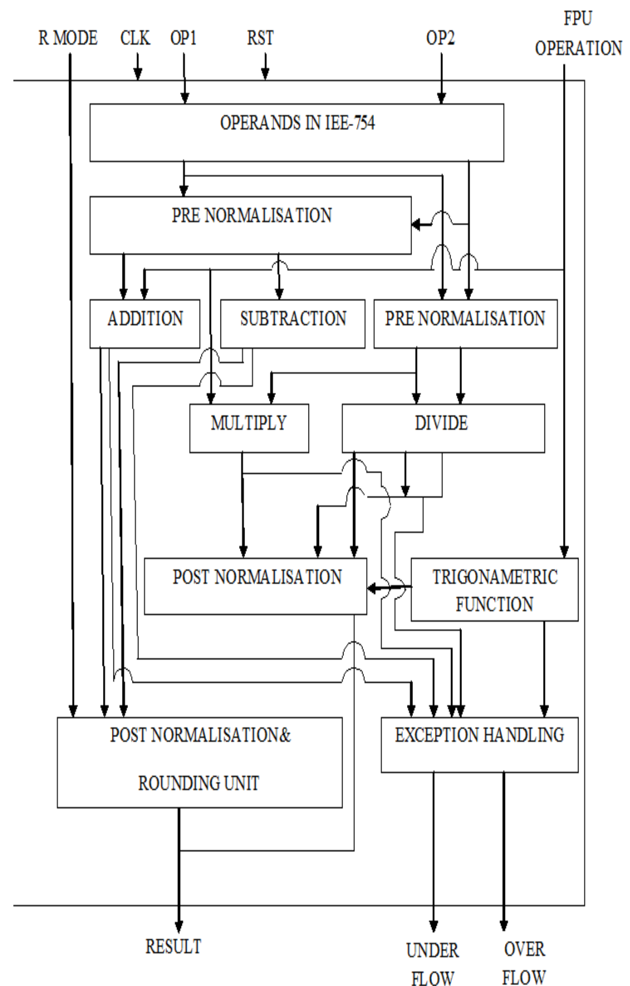


Figure 6: Block Diagram of Proposed Floating Point Arithmetic Unit.

The FPU of a single precision floating point unit that performs add, subtract, multiply, divide functions is shown in figure6 Two pre-normalization units for addition/subtraction and multiplication/division operations has been given

International Journal of Engineering Technology and Management Sciences
Website: ijetms.in Issue: 1 Volume No.6 January – 2022 DOI: 10.46647/ijetms.2022.v06i01.003
ISSN: 2581-4621

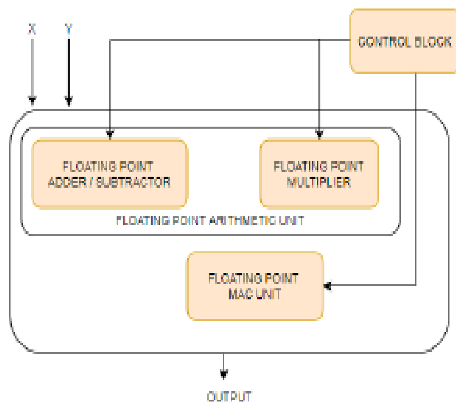


Figure 7: Basic block diagram of the existing system

In the proposed method Post normalization unit has been given that normalizes the mantissa part The final result can be obtained after post-normalization. To carry out the arithmetic operations, two IEEE-754 format single precision operands are considered. Pre-normalization of the operands is done. Then the selected operation is performed followed by post-normalizing the output obtained .Finally the exceptions occurred are detected and handled using exceptional handling. The executed operation depends on a three bit control signal (z) which will determine the arithmetic operation.

IV Modules Implementation

As our FPU works with floating point numbers, the operations, intermediate calculations and output are conventionally in the same floating point structure. But this invariably increases the complexity of calculation and the number of adjustments required at each level to obtain the correct result. Our proposal is to convert the floating point number into a simple yet quite precise integral representation and perform the calculations on the same, followed by the final conversion of the output into its expected floating point result format.

The floating point data is inputted in two parts. The first part is a 32 bit binary value of the **integer part** of the floating point operand and other is a 32 bit binary value of **fractional part** of the floating point operand. This is done because Verilog cannot deal with floatingpoint numbers. So we need to consolidate the two parts (integral and fractional) of the operand into a single 32 bit **effective operand.** This is done by the following algorithm explained in below.

Step 1: The sign bit (31stbit) of the input **integer part** becomes the sign bit of the **effective operand**.

Step 2: Then the position of 1stsignificant 1 is searched in the input **integer part** from RHS. This **position** is stored.

Step 3: All the bits from this position to the end of the input **integer part** (i.e. till the 0th bit) is taken and inserted into the **effective operand** from its 30th bit onward.(This step stores the actual useful bits of the **integer part** as not all the 32 bits are used to accommodate the **integer part**.)

Step 4: If there are still positions in the **effective operand** that are not filled, then it is filled with the bits from the input **fractional part** from its MSB down to the number of bits equal to places left to be filled.(This step stores the just requisite number of bits from the fractional part to complete the 32 bit representation)

This can be explained with the help of an example.

Float_op_int = 32‘‘b00000010101000110101000011100000

Float_op_frc = 32‘‘b11111111111111000000000000111111111

Step 1: Assign output[31] = Float_op_int[31]

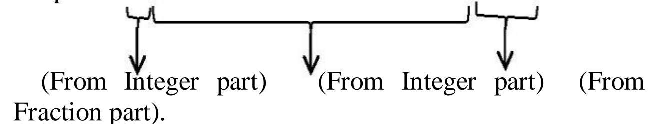
Step 2: Pos of 1st 1 from LHS of Float_op_int = 25(pos counted from RHS)

Step 3: Assign output = Float_op_int[25:0]

Step 4: Remaining bits left to be assigned inremaning = 32-26-1 = 5

Step 5: output[4:0] = Float_op_frc[31:27]

Output = 0 10101000110101000011100000 11111

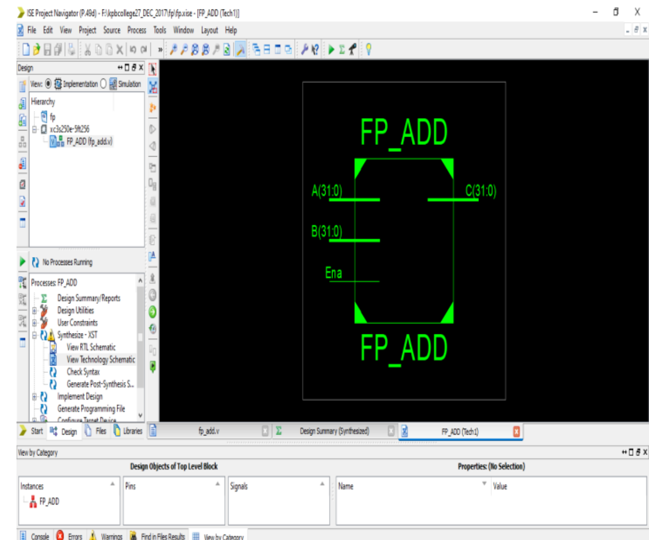


(From Integer part) (From Integer part) (From Fraction part).

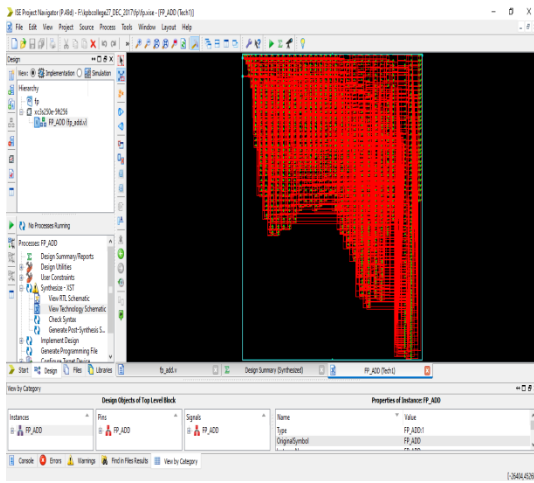
So, basically our technique gives preference to the fractional part for smaller numbers and the integer part for larger ones thus keeping intact the effective precision of the floating point number.

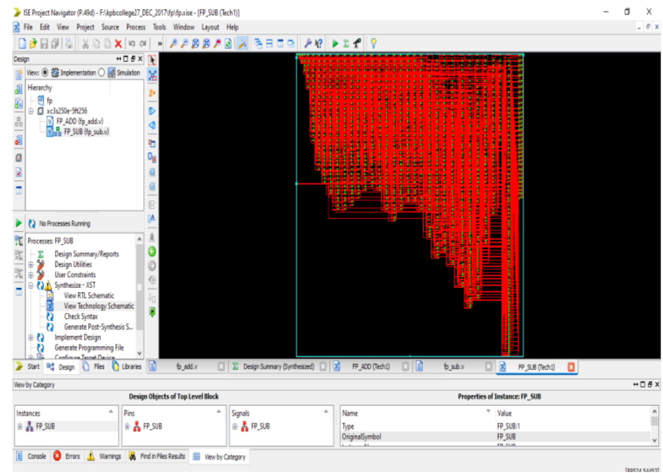
V RESULTS

Below figures represents the individual RTL view of each module and the simulation of three main modules.



Website: ijetms.in Issue: 1 Volume No.6 January – 2022 DOI: 10.46647/ijetms.2022.v06i01.003
ISSN: 2581-4621





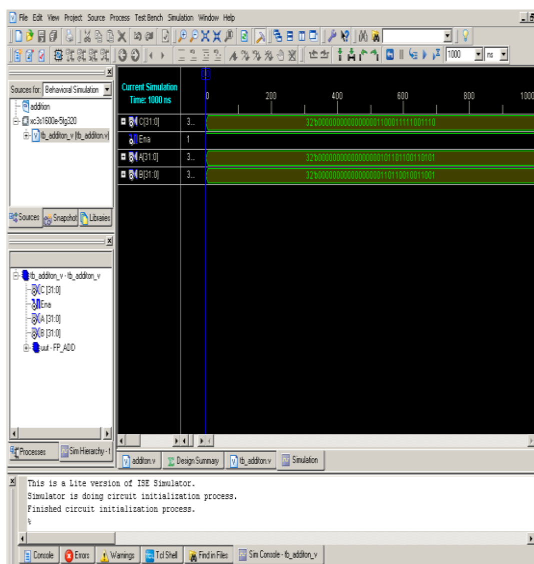


Figure 8: Simulation Result for Addition

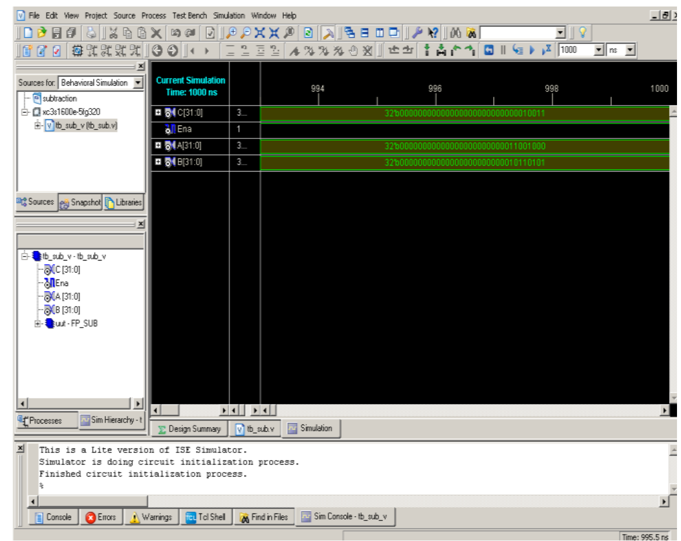
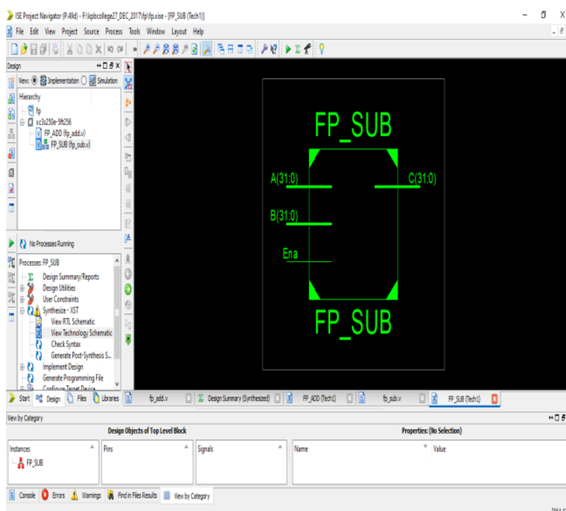
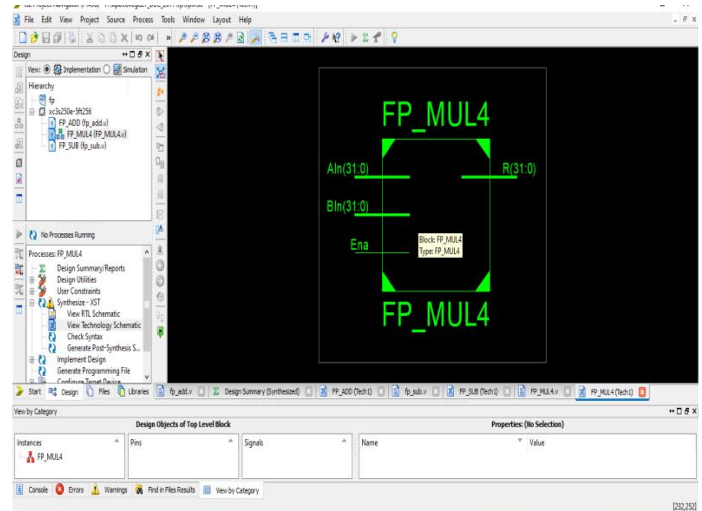
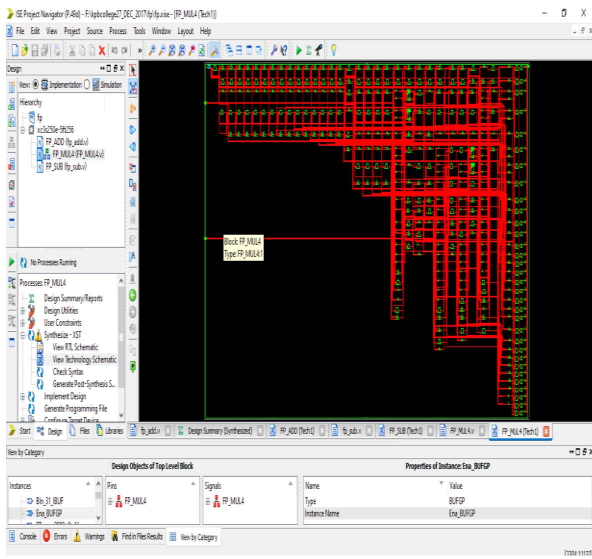


Figure 9: Simulation Result for Subtraction





International Journal of Engineering Technology and Management Sciences
Website: ijetms.in Issue: 1 Volume No.6 January – 2022 DOI: 10.46647/ijetms.2022.v06i01.003
ISSN: 2581-4621



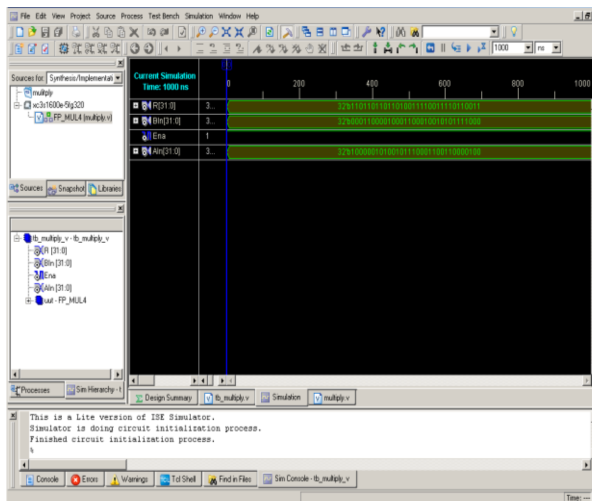
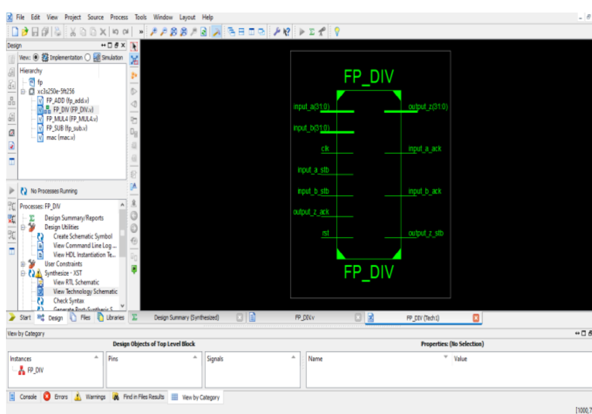
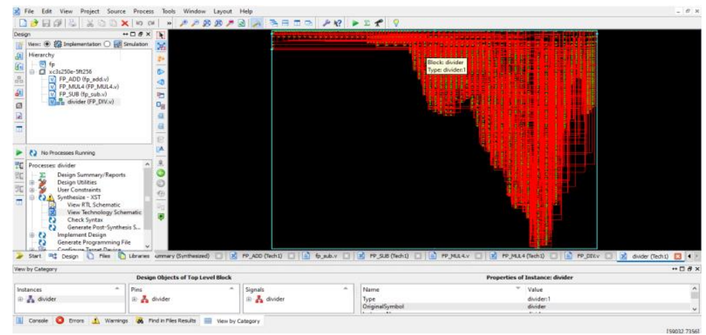


Figure 10: Simulation Result for Multiplication.





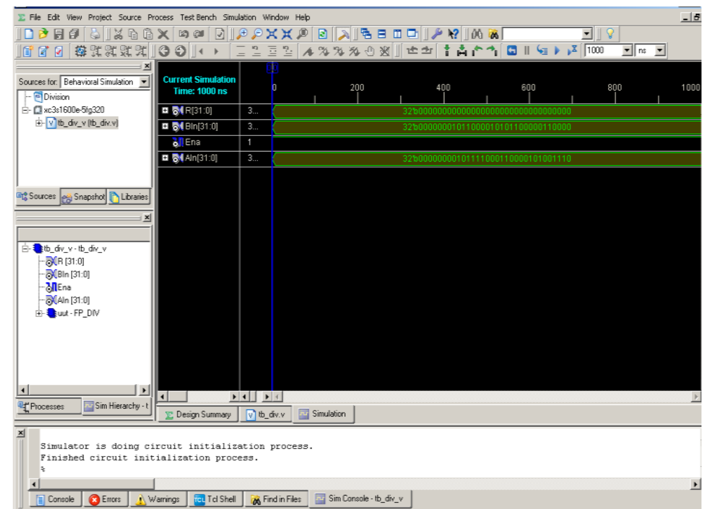
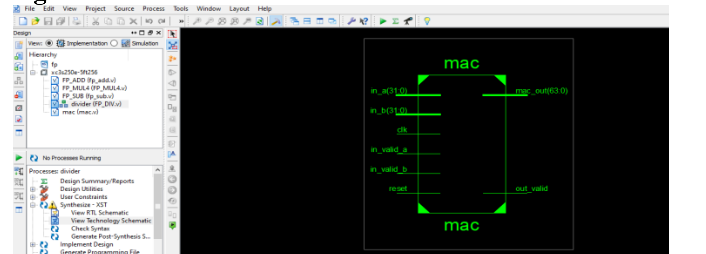


Figure 11: Simulation Result for Division



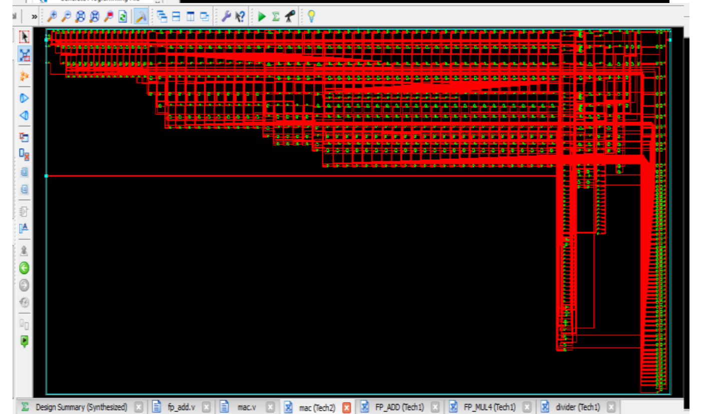


Figure 12: Simulation Result for MAC

Table 1: Existing System Results.

Addition/subtraction-Device Utilization Summary			
Logic Utilization	Used	Avai lable	Utilization
Number of slice registers	224	4800	4%
Number of slice LUT's	376	2400	15%
Number of bounded IOBs	60	102	58%
Number of fully used LUT-FF pairs	142	458	31%
Number of BUFG/BUFGCTRLs	1	16	6%
Multiplication -Device Utilization Summary			
Logic Utilization	Used	Avai lable	Utilization
Number of slice registers	427	4800	8%
Number of slice LUT's	1734	2400	72%
Number of bounded IOBs	106	102	102%
Number of fully used LUT-FF pairs	262	1899	13%
Number of BUFG/BUFGCTRLs	2	16	12%

Table 2: Proposed System Results.

Addition/subtraction-Device Utilization Summary			
Logic Utilization	Used	Avai lable	Utilization
Number of slice registers	412	2448	16%
Number of slice LUT's	764	4896	15%
Number of bounded IOBs	97	172	56%
Number of fully used LUT-FF pairs	89	4896	1%
Number of BUFG/BUFGCTRLs	1	24	4%
Multiplication -Device Utilization Summary			
Logic Utilization	Used	Avai lable	Utilization
Number of slice registers	65	2448	2%
Number of slice LUT's	122	4896	2%
Number of bounded IOBs	97	172	56%
Number of Mult 18x18SIo's	4	12	33%
Number of BUFG/BUFGCTRLs	1	24	4%
Division -Device Utilization Summary			
Logic Utilization	Used	Avai lable	Utilization
Number of slice registers	424	4800	17%
Number of slice LUT's	797	4896	16%
Number of bounded IOBs	104	102	101%
Number of fully used LUT-FF pairs	386	4896	7%
Number of BUFG/BUFGCTRLs	1	24	4%

Table 3: Existing Implementation

Unit	Minimum input arrival time before clock (ns)	Maximum output necessary time after the clock (ns)	Minimum period necessary (ns)
Adder/ subtractor	14.091	4.118	7.843
Multiplier	6.717	4.118	9.263
MAC unit	5.505	5.525	9.061

Table 4: Proposed Implementation

Unit	**Minimum input arrival time before clock (ns)**	**Maximum output necessary time after the clock (ns)**	**Minimum period necessary (ns)**
Adder/SuB	**12.814**	**3.928**	**7.52**
Multiplier	**4.189**	**4.182**	**8.415**
Mac Unit	**4.266**	**4.04**	**27.568**

VI CONCLUSIONS

The algorithm that we have used for the final FPU was comparable or even better in some case than the already existing efficient algorithms like in the case of block CLA and CLA with reduced fan-in in terms of memory used, delay, and device utilization. Because we have built the FPU using possible efficient algorithms with several changes incorporated at our ends as far as the scope permitted, all the unit functions are unique in certain aspects and given the right environment (in terms of higher memory or clock speed or data width better than the FPGA Spartan 3E synthesizing environment), these functions will tend to show comparable efficiency and speed and if pipelined then higher throughput may be obtained. Minimum period: 8.415ns (Maximum Frequency: 118.842MHz), Minimum input arrival time before clock: 4.189ns, Maximum output required time after clock: 4.182ns. Tough we have succeeded to achieve small amount of success in improvising the FPU, i.e. as per the results of synthesis and simulation, we have proved that our FPU have less memory requirement, less delay, comparable clock cycle and low code complexity, but still we have a vast amount of work that can be put on this FPU to further improvise the efficiency of the FPU. We can

further implement operations like Exponential functions and Logarithmic functions. Further implementing Pipelining for the above operations can further increase the efficiency of the FPU.

REFERENCES

1) Dr. Ravindra P. Rajput Srujana B Malkapur "Design of Generic Floating Point Pipeline Based Arithmetic Operation for DSP Processor" IEEE Xplore, 978-1-7281-5374-2/20/$31.00 ©2020 IEEE, pg.no 1059-1064
2) Manisha Sangwan, A Anita Angeline,Design and Implementation of Single Precision Pipelined Floating Point Co-Processor, 2013 InternationalConference on Advanced Electronic Systems (ICAES).
3) Ushasree G, R Dhanabal, Dr Sarat Kumar Sahoo, VLSI Implementationof a HighSpeed Single Precison Floating Point Unit using Verilog, proceedings of 2013 IEEE Conference on Information and Communication Technologies(ICT 2013).
4) F. Mhaboobkhan, K. Kokila, R. Jothikha, and K. L. Preethikha, " Design of P ipelined P arity Preserving Double P recision Reversible Float ing P oint Multiplier Using 90 nm Technology," *2020 6th Int. Conf. Adv. Comput. Commun. Syst. ICACCS 2020*, no. 2, pp. 739-744,2020,doi:10.1109/ICACCS48705.2020.9074209.
5) A. Yadav and I. Chaudhary, "Design of 32 -bit Float ing Point Unit for Advanced P rocessors," *Int. J. Eng. Res. Appl.*, vol. 07, no. 06, pp. 39–46, 2017, doi: 10.9790/9622-0706053946.
6) 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
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) H.H. Saleh, ―H.Fused Floating-Point Arithmetic for DSP,‖ PhD dissertation,Univ. of Texas, 2008.
9) 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
10) 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]