International Journal of Research Publication and Reviews

Journal homepage: www.ijrpr.com ISSN 2582-7421

Area and Speed Efficient Floating Point Unit Implementation on Hybrid FPGAs

Sateesh Kourav[a*], *Sunil Saha*[b],

[a]*MTech. Scholar, Department of Electronics and Communication, GGITS, Jabalpur*
[b]*Asst. Professor, Department of Electronics and Communication, GGITS, Jabalpur*

A B S T R A C T

Proposed design an Area and Speed optimized FALU for modern computing of advance processors, as we aware that in most of the DSP (Digital Signal Processors) FALU is a key element for real time computation of signals and real time data to meet real time scenario of signal processing it is highly required to make computation faster as possible so we have come up with idea to design fast FALU. The chip area is another requirement to design a compact module and less power. A FALU module has three sub-modules FM, FA & WB, optimizing these we can optimized overall design we have gone through multiple approaches for Floating multiplication and floating addition we plan to use Mitchell algorithm for multiplication and Wallace for addition and we will use coarse grain of Vertex -4 for all logical operations. We have proposal to merge Mitchell & Wallace techniques for designing FM & FA sub-modules and to design top module of design hierarchy including coarse grain logic modules WB's along with FA & FM. There top module is a 32-bit FALU module.

Keywords: FPGA, Floating Point Arithmetic Logic Unit (FALU), Floating Multiplier, Word Block, VHDL, VLSI

1.Introduction

Floating Multiplication is especially relevant since other arithmetic operators, such as division or exponentiation, which they usually utilize multipliers as building blocks. Various FPGAs are available now a day's which give us good hands-on research work in the field of ASIC designing. And as we knew a hard-core ASIC will gives us a better throughput over the software-based library routine if our application is specific. If our application is defined then there is nothing to trade off, for better performance Hybrid FPGA based IP (Intellectual Property) is batter choice.

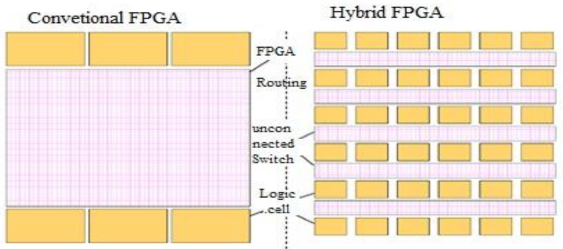


Fig 1 Conventional (Fine grain) FPGA v/s Hybrid (Coarse grain) FPGA

A hybrid FPGA consists of a combination of coarse-grained and fine-grained reconfigurable elements. It provides a high-throughput and cost-effective platform for designers to develop applications. Coarse-grained units are more efficient than fine-grained programmable logic for implementing specific word-level operations. For example, an application which demands high-performance floating-point computation can achieve better speed and density by incorporating embedded Floating arithmetic Logic Units (FALUs). Floating point adders/subtracters (FAs) and floating-point multipliers (FMs) contain several basic functional elements such as barrel shifters, adders and multipliers. Word blocks (WBs) are used for the bitwise operation of the floating-point number such as comparison, shifting, latch and logical operation.A coarse-grained FALU is composed of FAs, FMs, and WBs.The FAs and FMs are double precision and fully IEEE754b compatible including all four rounding modes, de-normalizednumbers and status signals.

This work uses IEEE754b Floating-point Format as Computer number format that occupies 4 bytes (32 bits) in computer memory and represents a

* *Corresponding author.* Tel.: +91-8120632029
E-mail address: kourav530@gmail.com

wide dynamic range of values by using a floating point. Proposed work is a design of 32-bit FALU hence the as per IEEE754b format the it would be:

1bit Signed	23 bits Exponent	8-bit fraction

The range will be -16777216.00390625 To 16777215. 00390625, And the least precision in number presentation will be 1/256 =0.00390625. Let say if we want to write as per the floating precision adopted

$(+23.45)_{10} = (000000000000000010010111.01110011)_2$And $(-38.78)_{10} = (111111111111111111011001.00111001)_2$

2. Methodology

As we have proposed to design a FALU unit for that we have to aware about the floating numbers and must keep in mind that computation for floating number is different the computation for non-floating numbers, as our FALU design is for 32-bit numbers so we have taken the upper 24 bit out of 32-bit number as integer and remains lower 8 bit of 32bit number as float. An example is below: -

13.25 => 000000000000000000001101.01000000
35.6875 => 000000000000000000100101.10110000

Proposed FALU has arithmetic and logical operations, arithmetic addition, subtraction, multiplication, & division. Logical operation includes logical AND, OR, XOR, XNOR, NAND, AND, NOR & NOT. There are three blocks need to be design in FALU module as shown in figure below, FA/FS, WB, FM & FD.

- FA/ FS => floating addition and subtraction
- FM => floating multiplication
- FD=> floating division

Module FA/ FS is design using coregent of Hybrid FPGA which is a coarse grain.

- FM design using our area optimized Mitchell algorithm
- FD design using our design & proposed technique.

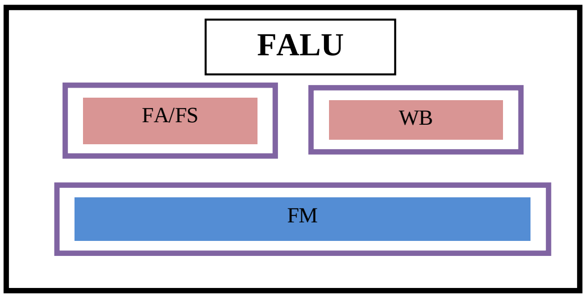


Fig 2: FALU internal Modules

2.1 FA/FS: FA is floating addition and FS is floating subtraction, we just need to select a hybrid FPGA and as we have discussed hybrid FPGA has coarse boundary which includes defined coregen. Of addition and subtraction. We have used them in our programming.

2.2 FM: FM is Floating multiplication and we have used Mitchell algorithm with necessary modification done by us in it. Mitchell algorithm is one of the most significant multiplication methods in LNS is Mitchell's algorithm. An approximation of the logarithm and the antilogarithm is essential, and it is derived from a binary representation of the numbers.

The logarithm of the product is

$$log_2(N_1.N_2) = k_1 + k_2 + log_2(1+m_1) + log_2(1+m_2) \quad (1)$$

The expression $log_2(1+m)$ is approximated with m and the logarithm of the two number's product is expressed as the sum of their characteristic numbers and mantissas:

$$log_2(N_1.N_2) \approx k_1 + k_2 + m_1 + m_2 \quad (2)$$

The characteristic numbers k_1 and k_2 represent the places of the most significant operands' bits with the value of '1'. For 32-bit numbers, the range for characteristic numbers is from 0 to 15. The fractions m_1 and $m_2$are in range [0, 1). The final MA approximation for the multiplication where$P_{true} = N_1.N_2$ depends on the carry bit from the sum of the mantissas and is given by:

$$P_{MA} = (N_1.N_2)_{MA} = \begin{cases} 2^{k_1+k_2}(1+m_1+m_2), & m_1+m_2 < 1 \\ 2^{k_1+k_2+1}(m_1+m_2), & m_1+m_2 \geq 1 \end{cases} \quad (3)$$

The final approximation for the product (3.29) requires the comparison of the sum of the mantissas with '1'. The sum of the characteristic numbers determines the most significant bit of the product. The sum of the mantissas is then scaled (shifted left) by $2^{k_1+k_2}$ or by$2^{k_1+k_2+1}$, depending on the $m_1 + m_2$.

If $m_1 + m_2 < 1$, the sum of mantissas is added to the most significant bit of product to complete the final result. Otherwise, the product is approximated only with the scaled sum of mantissas. The proposed MA-based multiplication is given in Algorithm discuss below:

1.N_1, N_2: n-bits binary multiplicands, $P^{(0)}_{approx} = 0$: 2n-bits first approximation, $C^{(1)} = 0$: 2n-bits i correction terms, $P_{approx} = 0$: 2n-bits product
2. Calculate k_1: leading one position of N_1
3. Calculatek_2: leading one position of N_2

4. Calculate$(N_1 - 2^{k_1})2^{k_2}$: shift $(N_1 - 2^{k_1})$to the left by $k_2$bits
5. Calculate$(N_2 - 2^{k_2})2^{k_1}$: shift $(N_2 - 2^{k_2})$to the left by $k_1$bits
6. Calculate $k_{12} = k_1 + k_2$
7. Calculate $2^{(k_1+k_2)}$: decode k_{12}
8. Calculate$P_{approx}^{(0)}$: add $2^{(k_1+k_2)}$, $(N_1 - 2^{k_1})2^{k_2}$and $(N_2 - 2^{k_2})2^{k_1}$
9. Repeat i-times or until.$N_1 = 0$, or $N_2 = 0$:
   (a) Set:$N_1 = (N_1 - 2^{k_1})$, $N_2 = (N_2 - 2^{k_2})$
   (b) Calculate k_1: leading one position of N_1
   (c) Calculatek_2: leading one position of N_2
   (d) Calculate$(N_1 - 2^{k_1})2^{k_2}$: shift $(N_1 - 2^{k_1})$to the left by $k_2$bits
   (e) Calculate$(N_2 - 2^{k_2})2^{k_1}$: shift $(N_2 - 2^{k_2})$to the left by $k_1$bits
   (f) Calculate $k_{12} = k_1 + k_2$
   (g) Calculate $2^{(k_1+k_2)}$: decode k_{12}
   (h) Calculate$C^{(i)}$: add $2^{(k_1+k_2)}$, $(N_1 - 2^{k_1})2^{k_2}$and $(N_2 - 2^{k_2})2^{k_1}$

$$P_{approx}^{(i)} = P_{approx}^{(0)} + \sum_i C^{(i)} \quad (4)$$

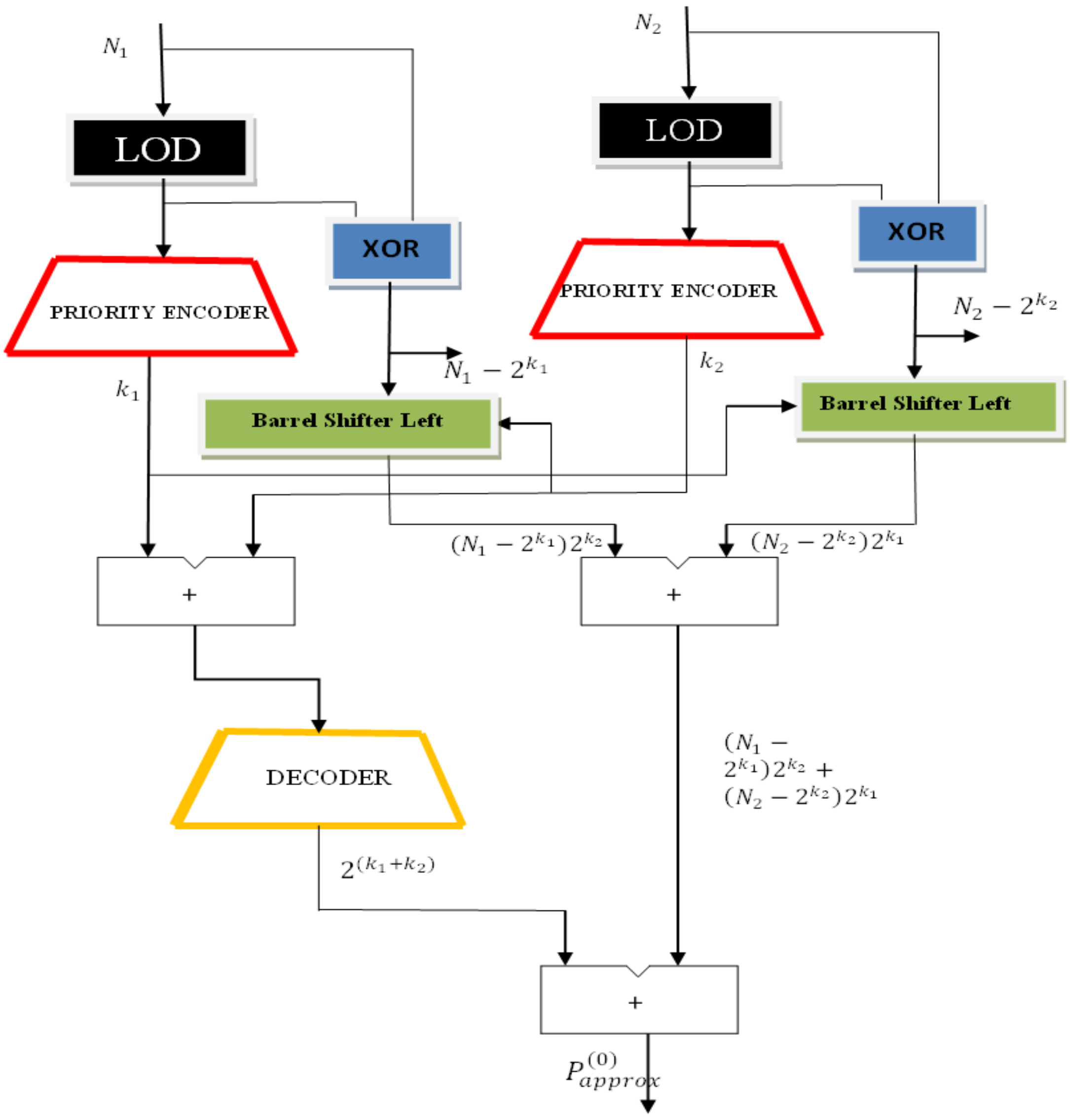


Figure 3 basic block of FM

2.2.1 Leading One Detector (LOD): The LOD is used in logarithmic converters to find the leading-one position, which is used to determine the integer portion and the fractional part of the binary logarithm. The characteristic value of a logarithmic number can be determined by the position of the leading one of the input numbers. The approach proposed by Mitchell uses a shift and count method to compute the desired characteristic value. This is the simplest way to implement the system; however, it will take too long to compute the output. The worst-case scenario, where the leading one is at the least significant bit position of the input, must be considered in the design. This will significantly affect the speed of the converter if the number of input bits is large.

2.2.2 Priority Encoder: The priority encoder will encode the position of the most significant input with a value of 1 and disregard the values of all the inputs that are in less significant positions. It gets the input from the output of the LOD circuit. For the case of 32-bit priority encoder, it will analyze the input and generate 5-bit outputs.

2.2.3 Barrel Shifter: the shifter is a main block in binary-to-binary logarithm converters because it provides the fractional part or the mantissa. The barrel shifters are widely used to shift or rotate large size binary words. A barrel is a combinational logic circuit with data inputs, data outputs, and control inputs. The output word is similar to the input word rotated or shifted by a number of bit positions specified by the control input. Here the barrel shifter gets the control inputs from the priority encoder, the data input of the barrel shifter is the input operand with most significant bit (MSB) removed from it.

2.2.4 Decoder: The decoder used is a 6:64bit decoder. The decoder unit gets the input as the sum of the characteristic numbers. The decode unit decodes 2^{k1} and 2^{k2} i.e. it puts the leading one in the product.

Adder: The design uses one 5-bit and two 64-bit adders. The 5-bit adder adds up the encoded value from the priority encoders, and gives the output to the decoder unit. The 64-bit adder is used to sum together the outputs of the barrel shifters; its output goes to the second 64-bit adder. The second 64-bit adder adds together the output of the decoder unit and the first 64-bit adder.

2.2.5 Implementation of Basic Block (BB) with correction circuits:To increase the accuracy of the multiplier, we implemented multipliers with error-correction circuits (ECC). The error-correction circuit is used to calculate the term ${n_1}^{(1)}$in figure 3 and thus approximates the term $({N_1}^{(1)} - 2^{k_1})2^{k_2} + ({N_2}^{(2)} - 2^{k_2})2^{k_1}$.

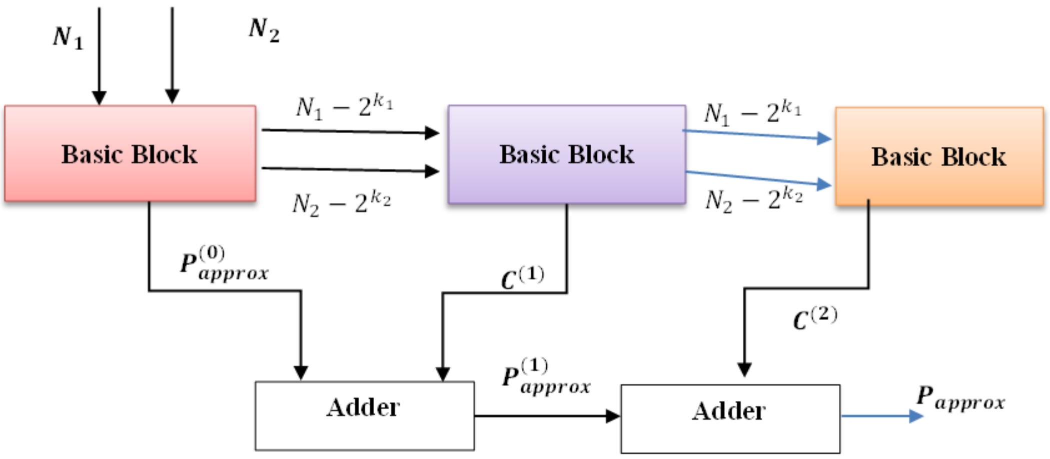


Figure 4: Logarithmic two-level iteration

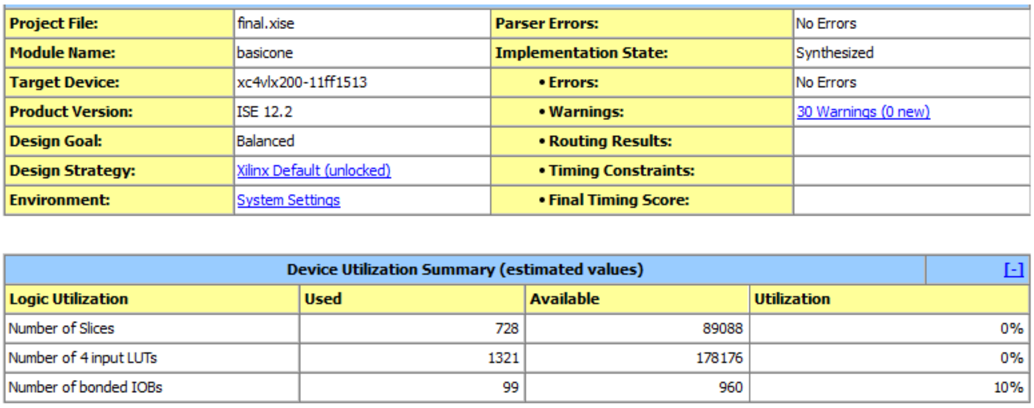
2.3 FALU OPCODE: As we discuss FALU will perform arithmetic as well as logical operation so here we need OPCODE according which we select operation.

TABLE 1 FALU OPCODE'S

S. N.	OPCODE	OPERATION
1	000	Addition
2	001	Subtraction
3	010	Multiplication
4	011	Logical Xor
5	100	Logical not
6	101	Logical and
7	110	Logical or
8	111	NOP

3.Results

Figure 5 shows the Synthesis *report of* proposed modified Mitchell multiplication after complete RTL entry on Xilinx EDA, it shows the number of LUT, number of slices used for designing proposed FALU also it shows the time taken by proposed design to generate the FALU output.



Project File:	final.xise	**Parser Errors:**	No Errors
Module Name:	basicone	**Implementation State:**	Synthesized
Target Device:	xc4vlx200-11ff1513	**• Errors:**	No Errors
Product Version:	ISE 12.2	**• Warnings:**	30 Warnings (0 new)
Design Goal:	Balanced	**• Routing Results:**	
Design Strategy:	Xilinx Default (unlocked)	**• Timing Constraints:**	
Environment:	System Settings	**• Final Timing Score:**	

Device Utilization Summary (estimated values)			[-]
Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slices	728	89088	0%
Number of 4 input LUTs	1321	178176	0%
Number of bonded IOBs	99	960	10%

Figure 5: Synthesis report of Proposed FALU

Figure 6 shown below shows the simulation of proposed FALU for test input signal : a= 04.A0 and b = 03.F0

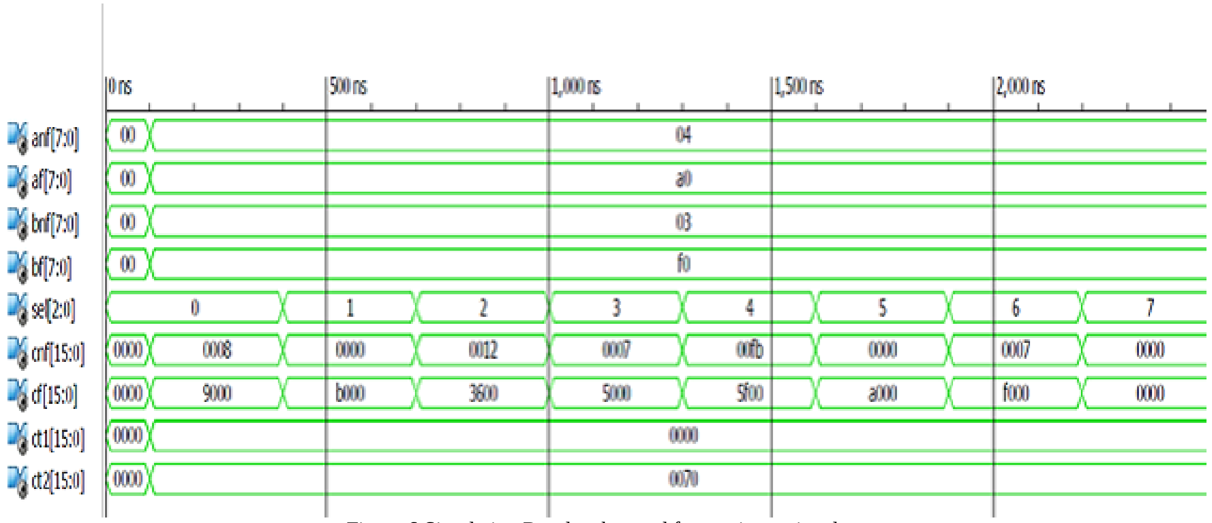


Figure 6 Simulation Results observed for test input signals

TABLE 2 SIMULATION VERIFICATIONS

Input1	Input2	operation	Expected output	Output found	Test
$(04.A0)_{16}$ $(4.625)_{10}$	$(03F0)_{16}$ $(3.9375)_{10}$	000-Addition	$(0008.9000)_{16}$ $(8.5625_{)10}$	$(0008.9000)_{16}$ $(8.5625_{)10}$	Tested OK
$(04.A0)_{16}$ $(4.625)_{10}$	$(03F0)_{16}$ $(3.9375)_{10}$	001-Subtraction	$(0000.B000)_{16}$ $(0.6875_{)10}$	$(0000.B000)_{16}$ $(0.6875_{)10}$	Tested OK
$(04.A0)_{16}$ $(4.625)_{10}$	$(03F0)_{16}$ $(3.9375)_{10}$	010-Multiplication	$(0012.3600)_{16}$ $(18.2109375_{)10}$	$(0012.3600)_{16}$ $(18.2109375_{)10}$	Tested OK
$(04.A0)_{16}$ $(4.625)_{10}$	$(03F0)_{16}$ $(3.9375)_{10}$	011-logical XOR	$(0007.5000)_{16}$ $(7.3125)_{)10}$	$(0007.5000)_{16}$ $(7.3125)_{)10}$	Tested OK
$(04.A0)_{16}$ $(4.625)_{10}$	$(03F0)_{16}$ $(3.9375)_{10}$	100-logical NOT of Input1	$(00FB.5F00)_{16}$ $(251.37109375_{)10}$	$(00FB.5F00)_{16}$ $(251.37109375_{)10}$	Tested OK
$(04.A0)_{16}$ $(4.625)_{10}$	$(03F0)_{16}$ $(3.9375)_{10}$	101 logical AND	$(0000.A000)_{16}$ $(0.625_{)10}$	$(0000.A000)_{16}$ $(0.625_{)10}$	Tested OK
$(04.A0)_{16}$ $(4.625)_{10}$	$(03F0)_{16}$ $(3.9375)_{10}$	110 logical OR	$(0007.F000)_{16}$ $(7.9375)_{)10}$	$(0007.F000)_{16}$ $(7.9375)_{)10}$	Tested OK
$(04.A0)_{16}$ $(4.625)_{10}$	$(03F0)_{16}$ $(3.9375)_{10}$	111-no operation	$(0000.0000)_{16}$ $(0.0_{)10}$	$(0000.0000)_{16}$ $(0.0)_{10}$	Tested OK

TABLE 3: PROPOSED MULTIPLIER RESULTS

No. of Slices	**215**
No. of 4 bit LUT	383
IOBs	95
Delay	8.894 ns
Max Freq.	112.4353 Mhz

TABLE 4: PROPOSED FALU RESULTS

No. of Slices	**728**
No. of 4 bit LUT	1321
IOBs	99
Delay	12.778
Max Freq.	78.25 Mhz

TABLE 5 COMPARATIVE RESULTS

	Base 1	**Base 2**	**Base 3**	**Proposed**
No. of Slices	739	762	751	728
No. of 4 bit LUT	--	--	--	1321
IOBs	--	--	--	99
Delay	35.7 ns	--	--	12.778
Max Freq.	--	66.18 Mhz	--	78.25 Mhz

As shown in above table-5 our proposed multiplier is batter in Area as our number of slices required less then base papers, and our speed is also batter as our Maximum frequency is higher than base papers.

4. Conclusion

Proposed work estimated the number of slices and 4 input LUTs is in FALU decreased by using Mitchell Multiplication Algorithm. It can be concluded that proposed design for Floating arithmetic Logic Unit in Hybrid FPGA in which logarithmic Multiplier used for multiplication purpose will requires lesser amount of area (gates) & delay (ns) as compared to the reference base papers. These techniques will be able to decrease the area up to a great extent for small numbers however for big number one more iterative block needs to be add and that will increase the total area. This paper proposed fast multiplier architecture for signed multiplications in FALU using Mitchell method of logarithmic mathematics. Since FALU is widely used in Digital Signal Processors the proposed multiplier can substantially speed up the multiplication operation which is the basic hardware block. Proposed work occupies less area and are faster than the other FALU.

References

[1] M. G. Arnold, V. Paliouras and I. Kouretas, "Implementing the Residue Logarithmic Number System Using Interpolation and Cotransformation," in IEEE Transactions on Computers, vol. 69, no. 12, pp. 1719-1732, 1 Dec. 2020, doi: 10.1109/TC.2019.2930514.

[2] R. P. Rao, N. D. Rao, K. Naveen and P. Ramya, "IMPLEMENTATION OF THE STANDARD FLOATING POINT MAC USING IEEE 754 FLOATING POINT ADDER," 2018 Second International Conference on Computing Methodologies and Communication (ICCMC), 2018, pp. 717-722, doi: 10.1109/ICCMC.2018.8487626.

[3] J. Varunkumar, R. Anusha, S. Shreenidhi and S. Vishwas, "Advanced verification of Single precision floating point ALU," 2019 1st International Conference on Advanced Technologies in Intelligent Control, Environment, Computing & Communication Engineering (ICATIECE), 2019, pp. 365-368, doi: 10.1109/ICATIECE45860.2019.9063823.

[4] Burud and P. Bhaskar, "Design and Implementation of FPGA Based 32-Bit Floating-Point Processor for DSP Application," 2018 Fourth International Conference on Computing Communication Control and Automation (ICCUBEA), 2018, pp. 1-5, doi: 10.1109/ICCUBEA.2018.8697775.

[5] Mark G. Arnold, Sylvain Collange publish their research paper title “Options for denormal representation in logarithmic arithmetic”, at J sign Process system, DOI 10.1007/s11265-014-0874-03, Springer link, year 2014

[6] Mr. Sandesh S. Saokar, R. M. Banakar and Saroja Siddamal proposed their research work entitle “High Speed Signed Multiplier for Digital Signal Processing Applications” at IEEE 2012

[7] D. Naresh, Dr. Giri Babu Kande publish their research paper entitle High Speed Signed multiplier for Digital Signal Processing Applications at IOSR Journal of Electrical and Electronics Engineering (IOSR-JEEE)

[8] D. Naresh, Dr. Giri Babu Kande High Speed Signed multiplier for Digital Signal Processing Applications, IOSR Journal of Electrical and Electronics Engineering (IOSR-JEEE) e-ISSN: 2278-1676, p-ISSN: 2320–3331, Volume 8, Issue 2 (Nov.-Dec.), PP 57-61

[9] Manish Chaudhary, Mandeep Singh Narula, High Speed Modified Booth’s Multiplier for Signed and Unsigned Numbers,International Journal of Science and Engineering Investigations vol. 2, issue 12, January 2013

[10] E. da Costa, J. Monteiro, S. Bampi, A new array architecture for signed multiplication using Gray encoded radix-2m operands, INTEGRATION, the VLSI journal 40 (2007) 118–132, 2006 Elsevier B.V. All rights reserved