International Journal of Computing and Digital Systems
ISSN (2210-142X)
Int. J. Com. Dig. Sys. 7, No.2 (Mar-2018)
http://dx.doi.org/10.12785/ijcds/070204

A Carry-Look Ahead Adder Based Floating-Point Multiplier for Adaptive Filter Applications

Aneela Pathan[1], Tayab D Memon[2] and Sheeraz Memon[3]

[1] *Institute of Information and Communication Technology, Mehran University of Engineering and Technology, Jamshoro, Pakistan*
[2]*Department of Electronic Engineering, Mehran University of Engineering and Technology, Jamshoro, Pakistan*
[3]*Department of Computer System Engineering, Mehran University of Engineering and Technology, Jamshoro, Pakistan*

Received 29 Sep.2017, Revised 26 Jan. 2018, Accepted 13 Feb. 2018, Published 1 Mar. 2018

Abstract: Floating-point arithmetic has various applications in the field of Science and Engineering. Specially, need of high precision floating-point multipliers is observed in Digital Signal Processing- like in filtering and transformations . High speed signal processing demands for high speed hardware. Though, various high level languages based implementations of floating-point multiplier are observed so far , but the hardware based implementation has still remained a bottleneck. With the development of Very Large Scale Integration (VLSI) technology, Field Programmable Gate Array (FPGA) has become the best candidate for implementing floating-point multipliers (due to their high integration density, low price, high performance and flexible applications). In this work, we have shown the implementation of IEEE-754 single precision floating-point multiplier on FPGA using carry-look ahead adder (for exponent addition). The multiplier may be used in adaptive filters for multiplying the fractional step size (mue) to update the filter weights. This paper also presents the comparative analysis of proposed design with Spartan 6 FPGA's built-in IPcore for floating-point multiplier. The results are compared in terms of recourse utilization, power consumption, observed delay, logic levels and maximum achieved frequency. It is shown that our design is better in terms of achieved frequency with a small increase in resource utilization.

Keywords: Floating-point multipier,Carry-look ahead adder, FPGA,IPcore ,Adaptive filter

1. INTRODUCTION

In adaptive filter design, the step size (mue) used to set the weights of adaptive filters is kept small, as too large step size gives a fast response to weight changes but results in a large excess mean square error (MSE)[1].Mostly, it is taken in fractions. Fractional or floating-point arithmetic is one of the key areas in adaptive filters, dealing not with the filter coefficients in factional values but also the intermediate arithmetic operations. Beside scientific computations, many DSP applications need floating -point arithmetic[2] .

The fractional or floating-point arithmetic has always remained a bottleneck to be implemented on hardware. It is required to convert floating-point values to large precision fixed-point representation for resulting in large dynamic range, but the hardness of this conversion and the quantization error reduces usage of this float to fixed point arithmetic in high precision embedded systems[3] .

Though, various high level languages (Like C,C++) based implementations of floating-point multiplier are observed so far, but the hardware based implementation has still remained a bottleneck. With the development of Very Large Scale Integration (VLSI) technology, Field Programmable Gate Array (FPGA) has become the best candidate for implementing floating-point multipliers (due to their high integration density, low price, high performance and flexible applications).

The FPGAs come with various built-in primitives and IPcores (Intellectual Property), optimized in terms of area, speed or power[4]. Like, the IPcore for floating-point multiplier is optimized in terms of area, but results in reduced performance in terms of frequency[5]. Hence, not restricting the design based on built-in IPcore, one can have the opportunity to get customized optimization in terms of area, power as well as frequency . This approach tends to more adaptable and flexible circuits .

In this work, we have shown the implementation of IEEE-754 single precision floating-point multiplier on FPGA using carry-look ahead adder (for exponent addition). The multiplier may be used in adaptive filters for multiplying the fractional step size (mue) to update the filter weights. Besides the comparative analysis of proposed design with FPGA's built-in IPcore for floating-

Email address: pathan_aneela@quest.edu.pk, tayabuddin.memon@faculty.edu.pk, sheeraz.memon@faculty.edu.pk

point multiplier is also carried out. The results are compared in terms of recourse utilization, power consumption, observed delay, logic levels and maximum achieved frequency. It is shown that our design is better in terms of achieved frequency with a small increase in resource utilization.

Several works are reported in area of floating-point multiplier design and other floating-point arithmetic units on a range of FPGA architectures. Few of them are discussed below.

In[6] the hardware based (FPGA) implementation of a high speed floating-point multiplier with pipeline architecture is presented . In the design ,Radix-4 Booth's encoding algorithm based on improved 4:2 compression structure is implemented as floating-point multiplier. While, the sum and carry vectors are added by a carry look-ahead adder. The timing simulation results show that the floating- point multiplier can be steadily run at the frequency of 80 MHz.

In[7] authors have reported a fast and area efficient carry select adder, that is implemented for exponent addition in floating-point multiplier along with the parallel processing of various units used in the architecture. The result shows a decrement of 27 % in the combinational path delay with an increment of around 8% in the number of LUTs used in comparison to other works discussed in paper. Whereas, the maximum frequency achieved in that design is 24.41 MHz causing the bottleneck for design to be used in high speed circuits.

In[8]the floating-point multiplier is proposed by utilizing reduced complexity Wallace multiplier for mantissa multiplication to minimize the latency. Normalization and Alignment Shifter has also been designed using barrel shifter to obtain the higher precision and lower latency. The total delay for this shifter is found to be 5.845 ns. While, the average delay of proposed approach is 37.673ns resulting the average frequency to restrict up to 26.54 MHz.

The work referred in[9] presents multi-functional, multiple precision floating-point Multiply-add Fused (MAF) unit. The mentioned multiply-add fused unit is accomplished to perform a quadruple accuracy. The design is done on a 65 nm silicon process attaining highest operating frequency of 293.5 MHz at 381 mW power.

In[10]FPGA based implementation of single precision floating-point multiplier and adders using the different adder approaches is discussed. The design shows that the carry-select adder based circuit offers best performance of all candidates including full carry-look ahead. Whiles, the frequency achieved at maximum does not exceed from 14.29 MHz.

A 16/18 bit pipelined multiplier following IEEE-754 standard is presented in[11].The design lacks in rounding mode support. The maximum frequency achieved is 19.0MHz.

A latency optimized floating-point built-in primitive of Vertex II FFPGA is instantiated in [12]. The latency of design is observed to be 4 clock cycles; whereas, the maximum frequency is100 MHz.

In[13]Handle-C along with Xilinx XCV1000 FPGA is used for designing of a parameterizable floating-point pipelined multiplier. With five stages of pipelining the design could reach to 28MFlops.

The implementation of IEEE-754 multiplier is observed in[14] targeted on Xilinx Virtex-5 FPGA. The pipelined approach is used to maximize the efficiency. It is also seen that over flow and underflow cases are tackled, but the rounding is not implemented. The design achieves 301 MFLOPs with the latency of three clock cycles.

With the idea of increasing the speed by reducing the delay, with incorporating an optimal adder like carry look ahead, the author in[15] has deigned IEEE-754 floating-point multiplier. The beauty of the design is its flexibility of being interfaced with any 32 bit processor.

In[16] authors describe a proficient implementation of an IEEE-754 single precision floating-point multiplier using Vedic mathematics-as with using Vedic mathematics partial products can be reduced, so that the area and power constraints of the floating- point multiplier can be reduced efficiently. The carried out work can be further improved by using high speed adders and substructures.

All the architectures mentioned above results in optimization in one way or other, but still some space is there to optimize the FPFA based floating-point multiplier in terms of area as well as frequency. This idea of customized optimization is carried out in this work and the results achieved show the succession of idea.

This paper further proceeds as follows: adaptive filter followed by carry-look ahead adder and IEEE-754 single precision floating- point multiplier is discussed in section 2. Section 3 describes the system design in FPGA. While, results and conclusion is given in section 4 and 5 respectively.

2. DESIGN COMPONENTS

In this section we have briefly described the basic components separately.

A. Adaptive Filter

The system that manages to produce the relationship of two signals using iterative manner in real time environment is known as an adaptive filter[17] .

The theme of adaptive filter is the adaptation in order to adjust the characteristics of the filter through an interaction with the environment [18].

The basic components of adaptive filter shown in Figure1 include:

- The signals being processed by the filter.
- The structure that defines how the output signal of the filter is computed from its input signal.
- The parameters within this structure that can be iteratively changed to alter the filter's input-output relationship.
- The adaptive algorithm that describes how the parameters are adjusted from one time instant to the next [19].

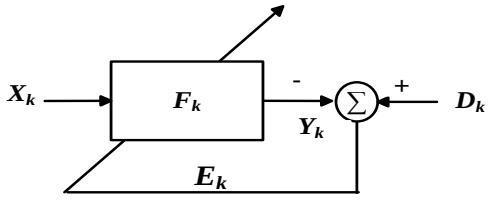

Figure 1. Block diagram of adaptive filter

Most of the common adaptive algorithm is Least Mean Square (LMS).

The LMS is the adaptive algorithm, used to drive the desired filter coefficients with stochastic gradient descent method. The filter is adapted based on the current time error (difference between the desired signal and the actual signal) [20].

B. Carry -look ahead adder

One of the major bottlenecks of ripple carry adder is the time of carry propagation (where the next output sum bit is dependent upon the previous carry). This can be handled once the next stage carry is calculated at prior. This approach is adapted by carry-look ahead adder, that is one of the fast digital adders [21].

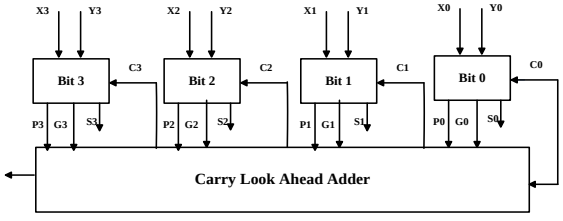

Figure 2. Block diagram of carry-look ahead adder

In comparison to ripple carry adder, carry-look ahead adder has improved speed, as it pre-calculates the carry bit.

Hence, using this adder in the design will result in overall reduced delay [22].

The carry- look ahead approach works on the basis of two terms the Propagate and the Generate. Those may be prior calculated as follow:

$P_i = A_i$ xor B_i (Carry propagate) (1)
$G_i = A_i$ and B_i (Carry generate) (2)

The S_i and C_{i+1} represent the sum and carry-out. Respectively, the S_i and C_{i+1} are expressed as:

$S_i = P_i$ xor C_{i-1} (3)
$C_{i+1} = G_i$ or (P_i and C_i) = G_i+ (P_i C_i)) (4) [18].

C. IEEE -754 Single precision floating- point multiplier

The IEEE-754 standard defines number representations and operations for floating-point arithmetic. The three types of floating- point formats given in IEEE-754 standard are Single, Double and Double-Extended[23] .

The floating- point representation of any number consists of three fields known as Sign, Exponent and Significant or Mantissa part. The bit width distribution of three IEEE formats for floating- point representation is given in figure 3, figure 4 and figure 5 respectively.

Sign	Exponent	Significant/Mantissa
1-bit	8-bits	23-bits

Figure 3. Single precession (32−bit)

Sign	Exponent	Significant/Mantissa
1-bit	11-bits	52-bits

Figure 4. Double precession (64−bit)

Sign	Exponent	Significant/Mantissa
1-bit	15-bits	112-bits

Figure 5. Double extended (128−bit)

D. Spartan 6's built-in IPcore of floating- point multiplier

Along with other built in IPcores, Xilinx's Spartan 6 FPGA also facilitate the designer with more complicated core of floating-point arithmetic (figure 6 shows the interface of the built-in floating- point unit).

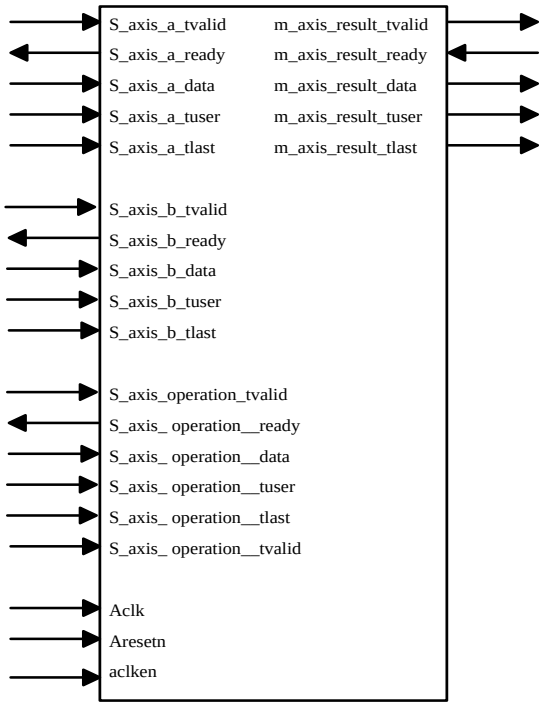

Figure 6. Floating- point unit interface

The provided core may be used for customized design- as the operand word length, required latency and even the interface may be selected as per design requirement. A range of arithmetic operations may be performed with this built-in core, where each operation variant has a common interface[5] .

Amongst the steps of floating-point multiplier, the more resource consuming part is the mantissa multiplier. The mantissa multiplier takes more hardware as compare to the exponent addition and even the sign bit calculation of floating-point conversion.

Using Spartan 6's built-in IPcore, the floating-point multiplier may be designed particularly in four configurations, categorized on the basis of DSP48 usage. Those possible configurations are summarized in the table1.

TABLE I. IMPACT OF FAMILY AND MULTIPLIER USAGE ON THE IMPLEMENTATION OF THE MULTIPLIER

Multiplier Usage	**Spartan-6 FPGA Family**	**Virtex 6 and 7 Series FPGA Families**
No Usage	Logic	Logic
Medium Usage	DSP48A1+logic in multiplier body	DSP48E1+logic in multiplier body
Full Usage	DSP48A1 used in multiplier body	DSP48E1 used in multiplier body
Maximum Usage	DSP48A1 multiplier body and rounder	DSP48E1 multiplier body and rounder

3. SYSTEM DESIGN IN FPGA

A. spartan 6's built-in IPcore based design

Eight implementations were carried out in this work using Spartan 6 FPGA. In first four implementations, Spartan 6's built-in core for floating- point multiplier was instantiated using IPcore based design option. Figure 7 shows the general arithmetic functions of floating- point multiplier carried out by built-in IPcore.

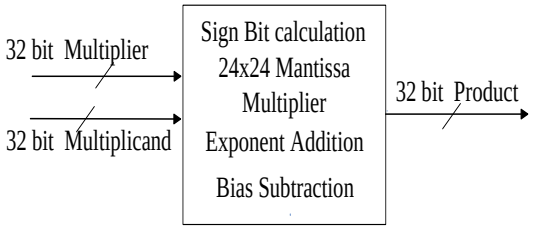

Figure 7. Arithematic functions for floating-point multiplier

Interface for core based design of floating- point unit provides various configuration options. Like, selection of floating- point arithmetic operation, setting the latency for the output and to make the design synchronous, asynchronous, clock enabled and using DSP48 or not.

In our design, we selected the option of floating- point multiplier with zero latency, hence getting the output with no delay.

After instantiating the core to main module, the clock was introduced in order to get the registered synchronous output. This configuration caused the output to observe the latency of one cycle.

Among the four implementations using built in core, in the first design, the primitive of floating-point multiplier was configured with the option of no usage of DSP48 as discussed in table 1. Hence, whole floating-point multiplier was carried out on logic units-the Look up tables (LUTS).

In the second design, using the same method the option of medium usage of DSP48 was selected. This option translated the some part of 24 bit mantissa multiplication on built-in DSP ; whereas, the other logic (i.e. sign bit calculation, exponent addition, bias subtraction and mantissa normalization) was translated on LUTS.

The third design was carried out using the full usage option of DSP48, that completely translated the mantissa multiplier on DSP48 while the other logic was translated on LUTS.

And finally, the option of maximum usage of DSP48 was utilized that not only translated the multiplier on DSP48 but exponent addition was also translated on DSP 48.

The Register Transfer Level (RTL) view of the built-in core based design may be seen in figure 8.

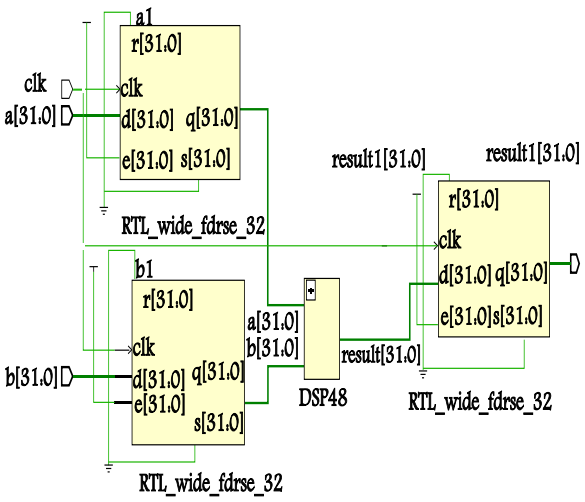

Figure 8. RTL view of IPcore based floating- point multiplier

B. Carry- lookahead based design

Besides the designs mentioned above, four designs were implemented using verilog syntax.

As per steps of floating-point multiplier, all the required modules were designed and implemented using the proper coding style and logic optimization to make the design more compact and efficient.

In order to make the designs comparable with those of built-in IPcore of floating-point multiplier, some changes were made in the project goals and strategy setting options (we selected balanced design for our project).

In the first design, the project goals and strategy option was customized to no usage of DSP48 (hence making the design comparable built in core based design). This resulted in LUT based implementation of the multiplier.

In second, the option of selecting the DSP48 be used was set to 60%. This customized strategy resulted in same number of DSP 48 multipliers to that of built in primitive based design ,while 40 % logic was translated on LUTS.

For third design, the 100 % DSP48 was utilized that resulted whole mantissa multiplier to be translated on DSP48.

Whereas in fourth design, besides the mantissa multiplier, the bias subtraction was also translated on DSP 48.

The first step of floating- point multiplier (that is sign bit calculation) was simply carried out using XOR gate as shown in figure 9.

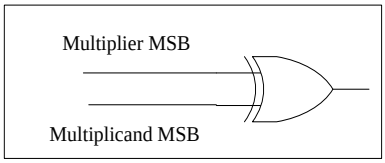

Figure 9. Sign bit calculation

Twenty-four bit mantissa multiplier was set to be translated either on the LUTs or on DSP48 as shown in figure 10.

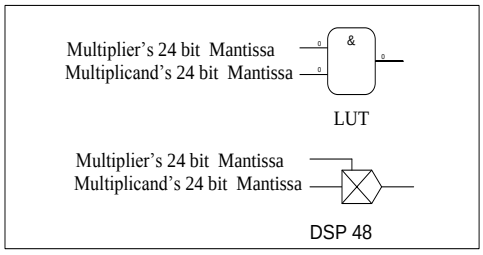

Figure 10. LUT/DSP48 based mantissa multiplier

Mantissa normalization is the method of converting the 48 bit output got after the mantissa multiplication to 23 bits as per requirement of single precision floating-point multiplication.

In this step, condition is set on the most significant bit of the output product, that is, if it is high the immediate 23 bits would be the mantissa bit otherwise right shift would be made to check for the next MSB. The procedure would be continued till the first high bit is achieved.

In our design, the multiplexer logic is implemented to perform this task.

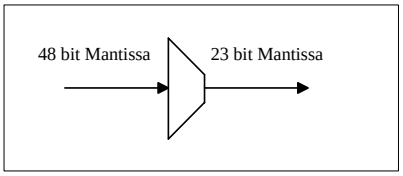

Figure 11. Mantissa Normalization

Exponent addition is another important element of floating- point multiplier. In our design we have used 8 bit carry-look ahead adder to do this work. This proposed modification brought very good effect in overall performance of the multiplier.

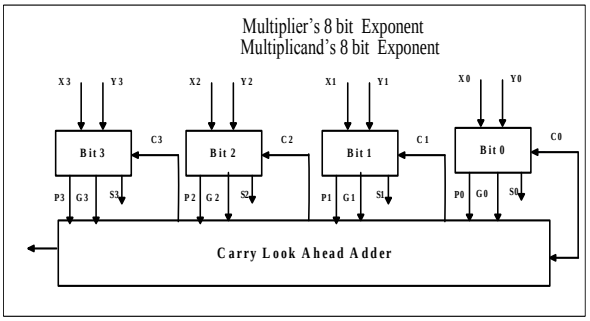

Figure 12. Exponent Addition

After the addition of exponents, to avoid the over flow the constant bias value (127 in decimal) is subtracted from the result to restrict the output of single precession floating-point multiplier to 32 bits.

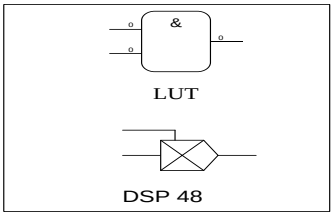

Figure 13. Bias subtraction

The bias subtraction can be translated on the look up tables or DSP48 as per required option.

4. RESULTS

The results given in tables are written after implementing the built-in IPcore of floating-point multiplier and the proposed design using carry- look ahead adder. Xilinx ISE 14.2 software and Spartan 6 xc6slx16-3csg324 FPGA device is used in implementation. All eight implementations are compared in terms of LUTS, DSP48 primitives, total delay observed, logic levels, maximum achieved frequency and estimated power.

TABLE II. DESIGN SUMMARY OF FLOATING- POINT MULTIPLIER WITH BUILT IN CORE USING LOGIC ONLY AND PROPOSED DESIGN

No Usage	Built in Primitive	Our Design
LUTS	841	1051
DSP48	0	0
Delay	17.529ns	5.027ns
Levels of Logic	60	3
Maximum Frequency	57.049 MHz	198.922MHz
Power Estimation	0.023 (W)	0.026 (W)

TABLE III. DESIGN SUMMARY OF FLOATING- POINT MULTIPLIER WITH BUILT IN CORE USING LOGIC AND MEDIUM USAGE OF DSP48 AND PROPOSED DESIGN

Medium Usage	Built in Primitive	Our Design
LUTS	397	1020
DSP48	1	1
Delay	15.935ns	5.027ns
Levels of Logic	44	3
Maximum Frequency	62.755 MHz	198.922MHz
Power Estimation	0.024(W)	0.026(W)

TABLE IV. DESIGN SUMMARY OF FLOATING- POINT MULTIPLIER WITH BUILT IN CORE USING LOGIC AND FULL USAGE OF DSP48 AND PROPOSED DESIGN

Full Usage	Built in Primitive	Our Design
LUTS	106	177
DSP48	4	4
Delay	23.053ns	5.080ns
Levels of Logic	11	3
Maximum Frequency	43.378MHz	196.866MHz
Power Estimation	0.024 (W)	0.026(W)

TABLE V. DESIGN SUMMARY OF FLOATING- POINT MULTIPLIER WITH BUILT IN CORE USING LOGIC AND MAXIMUM USAGE OF DSP48 AND PROPOSED DESIGN

Maximum Usage	Built in Primitive	Our Design
LUTS	108	170
DSP48	5	5
Delay	23.055ns	5.080ns
Levels of Logic	11	3
Maximum Frequency	43.374MHz	196.866MHz
Power Estimation	0.024 (W)	0.026(W)

The results shown in above tables very clearly indicate the difference between the resource utilization, maximum frequency achieved, delay observed, logic level and power estimation. All this comparison is carried out after post place and route simulation of the design. The power estimation is carried out using Xilinx Power Analyzer tool.

The results in table show that with a small increase in look up table utilization, a significant increase is observed in the maximum frequency achieved.

The effect of using carry-look ahead adder may be observed in the logic levels achieved in implementations. As in carry- look ahead adder, the carry is calculated prior to next stage, hence the logic level is reduces as observed in the results. As for as power is concerned, all eight implementations consume approximately same amount.

The timing performance diagram shown in figure 14 verifies our design requirement of maximum latency of one cycle. Here we can see that the multiplier's output comes immediately at the second positive edge of the clock.

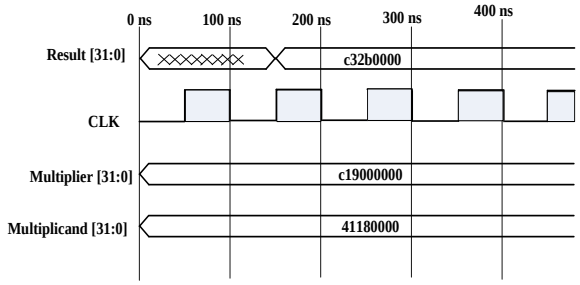

Figure 14. Timing performancen with latency of one clock cycle

5. CONCLUSION

In this work, we have shown the implementation of IEEE-754 single precision floating- point multiplier on FPGA using carry-look ahead adder for exponent addition. Also the comparative analysis of proposed design with Spartan 6 FPGA's building IPcore for floating- point multiplier is carried out . The results are compared in terms of recourse utilization, power consumption, observed delay, logic levels and maximum achieved frequency. The maximum frequency achieved with IPcore based design is 62.755 MHz while our proposed design results in 196.866MHz that is about three times larger than IPcore based design, hence making it suitable for high speed circuits.

It can be concluded here that not all IPcores always serve the selected aim. As we can see that in area point of view the floating- point multiplier's built-in core is better but when talking about the speed, with slight increase in look up tables our design is better than the first.

So, here comes the trade of for selecting the best as per user requirement. If the priority is area then the built-in primitive of floating- point multiplier is better, but if speed is of concerned, defiantly our design is much better.

REFERENCES

[1] D. Bismor, "LMS algorithm step size adjustment for fast convergence," *Archives of Acoustics*, vol. 37, pp. 31-40, 2012.

[2] J. Sohn and E. E. Swartzlander, "Improved architectures for a floating-point fused dot product unit," in *Computer Arithmetic (ARITH), 2013 21st IEEE Symposium on*, 2013, pp. 41-48.

[3] G. Govindu, L. Zhuo, S. Choi, P. Gundala, and V. K. Prasanna, "Area, and power performance analysis of a floating-point based application on FPGAs," UNIVERSITY OF SOUTHERN CALIFORNIA LOS ANGELES DEPT OF ELECTRICAL ENGINEERING2003.

[4] "Spartan-6 Libraries Guide for HDLDesigns (UG615), Xilinx Inc" 2009.

[5] I. LogiCORE, "Floating-Point Operator v6. 0," Xilinx Inc, 2012.

[6] G. Renxi, Z. Shangjun, Z. Hainan, M. Xiaobi, G. Wenying, X. Lingling, et al., "Hardware implementation of a high speed floating point multiplier based on FPGA," in Computer Science & Education, 2009. ICCS*E'09. 4th International Conference on*, 2009, pp. 1902-1906.

[7] S. Raghav and R. Mittal, "Implementation of Fast and Efficient Mac Unit on FPGA," 2016.

[8] S. Kakde, M. Mahindra, A. Khobragade, and N. Shah, "FPGA Implementation of 128-Bit Fused Multiply Add Unit for Crypto Processors," in *International Symposium on Security in Computing and Communication*, 2015, pp. 78-85.

[9] K. Manolopoulos, D. Reisis, and V. A. Chouliaras, "An efficient multiple precision floating-point Multiply-Add Fused unit," *Microelectronics Journal*, vol. 49, pp. 10-18, 2016.

[10] B. Fagin and C. Renard, "Field programmable gate arrays and floating point arithmetic," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 2, pp. 365-367, 1994.

[11] N. Shirazi, A. Walters, and P. Athanas, "Quantitative analysis of floating point arithmetic on FPGA based custom computing machines," in *FPGAs for Custom Computing Machines, 1995. Proceedings. IEEE Symposium on*, 1995, pp. 155-162.

[12] B. Lee and N. Burgess, "Parameterisable floating-point operations on FPGA," in *Signals, Systems and Computers, 2002. Conference Record of the Thirty-Sixth Asilomar Conference on*, 2002, pp. 1064-1068.

[13] J. Allan and W. Luk, "Parameterised floating-point arithmetic on FPGAs," in *Acoustics, Speech, and Signal Processing, 2001. Proceedings.(ICASSP'01). 2001 IEEE International Conference on*, 2001, pp. 897-900.

[14] M. Al-Ashrafy, A. Salem, and W. Anis, "An efficient implementation of floating point multiplier," in *Electronics, Communications and Photonics Conference (SIECPC), 2011 Saudi International*, 2011, pp. 1-5.

[15] A. Jain, B. Dash, A. K. Panda, and M. Suresh, "FPGA design of a fast 32-bit floating point multiplier unit," in *Devices, Circuits and Systems (ICDCS), 2012 International Conference on*, 2012, pp. 545-547.

[16] R. S. S. Teja and A. Madhusudhan, "FPGA implementation of low-area floating point multiplier using Vedic mathematics," *IJ of Emerging Technology and Advanced Engineering*, 2013.

[17] S. Goswami, P. Deka, B. Bardoloi, D. Dutta, and D. Sarma, "A novel approach for design of a speech enhancement system using NLMS adaptive filter and ZCR based pattern identification," in *Emerging Trends and Applications in Computer Science (ICETACS), 2013 1st International Conference on*, 2013, pp. 125-129.

[18] "Design and Comparative Analysis of Various Adders through Pipelining Techniques."

[19] P. Kunche and K. Reddy, "Adaptive Noise Cancellation to Speech Enhancement," in *Metaheuristic Applications to Speech Enhancement*, ed: Springer, 2016, pp. 7-15.

[20] R. V. G. KRISHNA and N. V. SATISH, "Implementation of Fixed-Point LMS Adaptive Filter," 2016.

[21] "Simultaneous carry adder," ed: Google Patents, 1960.

[22] P. Shrivastava, "Analysis and Design of an Area-Efficient Fastest Carry Look Ahead Adder with Enhanced Multiple Output Transmission Gate Logic," 2014.

[23] W. Kahan, "IEEE standard 754 for binary floating-point arithmetic," *Lecture Notes on the Status of IEEE*, vol. 754, p. 11, 1996.

Aneela Pathan received degrees of BE in Telecommunication from Mehran UET, Jamshoro Sindh, Pakistan in 2008 and ME in Electronic Engineering from NED university Karachi in 2010. Currently, she is pursuing her PhD degree (FPGA based DSP system Design) from Meharn UET. Mrs. Pathan worked as Assistant Manager (satellite receivers' design) in SUPARCO from March 2008 to Feboury 2013.
Since then she serves as Assistant Professor in Quaid-Awam University College of Engineering Science and Technology Larkana.

Tayab D Memon received a BE (Hons) Electronics Engineering (First Class) and a PG Diploma Telecommunication and Control Engineering (First Class) from Mehran University of Engineering & Technology, Jamshoro, Pakistan, in 2003 and 2006 respectively. He received PhD from Royal Melbourne Institute of Technology (RMIT) Melbourne Australia in 2012.
Currently he is working as Associate Professor in the Department of Electronic Engineering, MUET. His research interests include short word length DSP Systems, embedded systems, and their FPGA-based implementation.

Sheeraz Memon is working with the department of Computer SystemEngineering,MehranUET,Sindh ,Pakistan,sinceSep,2004. He completed his Bachelor of Engineering in computer system in Feb, 2004 and Master of Engineering in Communication Systems and Networks in Feb 2007, form Mehran UET Jamshoro.
The same year he was awarded Ph.D scholarship under faculty development program and Mr.Memon left for securing the doctoral degree from RMIT University Australia. He is skilled in Digital Signal Processing (DSP), Voice/Speech Feature Extraction, Machine Learning/Pattern recognition and Digital Image Processing.