International Journal of Electrical and Computer Engineering (IJECE)
Vol. 3, No. 6, December 2013, pp. 805~814
ISSN: 2088-8708

Evaluation of High Speed Hardware Multipliers - Fixed Point and Floating point

Awais Ahmed[1], Syed Haider Abbas[2], Muhammad Faheem Siddique[2], Hussnain Haider[2]
[1]School of Electrical Engineering, University of Faisalabad, Pakistan
[2]Departement of Electrical Engineering, Sarhad University of Science and IT, Pakistan

Article Info

Article history:

Received Aug 11, 2013
Revised Oct 14, 2013
Accepted Nov 1, 2013

Keyword:

Bit Serial Multiplier
Fixed Point
Floating Point
Parallel Multiplier
VHDL

ABSTRACT

There is a huge demand in high speed arithmetic blocks, due to increased performance of processing units. For higher frequency clocks of the system, the arithmetic blocks must keep pace with greater requirement of more computational power. Area and speed are usually conflicting constraints so that improving speed results mostly in larger areas. In our research we will try to determine the best solution to this problem by comparing the results of different multipliers. Different sized of two algorithms for high speed hardware multipliers were studied and implemented ie. Parallel multiplier, Bit serial multiplier. The workings of these two multipliers were compared by implementing each of them separately in VHDL. A number of high speed adder designs are developed and algorithm and design of these adders are discussed. The result of this research will help us to choose the better option between serial and parallel multipliers for both fixed point and floating point multipliers to fabricate in different systems. As multipliers form one of the most important components of many systems, analysing different multipliers will help us to frame a better system with area and better speed.

Corresponding Author:

Syed Haider Abbas
Departement of Electrical Engineering,
Sarhad University of Science and Information Technology,
Hayatabad Link, Ring Road, Peshawar, Pakistan
Email: habbas33@gmail.com

1. INTRODUCTION

Multipliers are the major components of high performance systems used extensively in digital electronics such as microprocessors, digital signal processors and FIR Filters etc. The performance of any system is determined by the performance of multipliers because they are the slowest part in the system. Moreover, they require greater area than other components. Therefore, optimizing the speed and area of the multipliers is the foremost issue. As area and speed are both conflicting constraints this means that for greater speed we need larger area. A number of algorithms are proposed and used to design multipliers and the actual implementation is mostly some little refinements and variations of the few basic algorithms presented here. In addition to choosing those algorithms for addition, subtraction, multiplication etc an architect must make other decisions like how exceptions should be handled and what precisions should be implemented. We have designed two typed of multiplier for fixed and floating point [1-2].

Our discussion on floating point will focus almost exclusively on the IEEE floating-point standard (IEEE 754) because of its rapidly increasing acceptance. Although floating point arithmetic involves manipulating exponents and shifting fractions, the bulk of the time in floating point is spent operating on fractions using integer algorithms. Thus, after our discussion of floating point we will take a more detailed look at efficient algorithms and architectures.

When two binary numbers, multiplicand "N" and multiplier "M" are multiplied the algorithm utilises distributive property of multiplication. The multiplication of M*N consists of partial products which represents a single component of the total product [3]. A binary multiplier can be decomposed in to a sum of partial products. It can be done by selection of a partial product value, and some shifting that is independent of the selection value. Opting a group length for multiplication limit us to make a trade-off between the number of partial products and the complexity of computing them.

2. BINARY NUMBER REPRESENTATION

2.1. Fixed Point Arithmetic

Most of the general Digital processing applications use fixed point multiplication due to its easier algorithm and clear understanding. A multiplication can be implemented using a network of binary shifts and adders [1] [4-6]. The hardware cost can be approximated with the required numbers of adders and subtractors. The basic concept of multiplication is similar to the traditional pencil and paper method. A similar example is shown in the Figure 1.

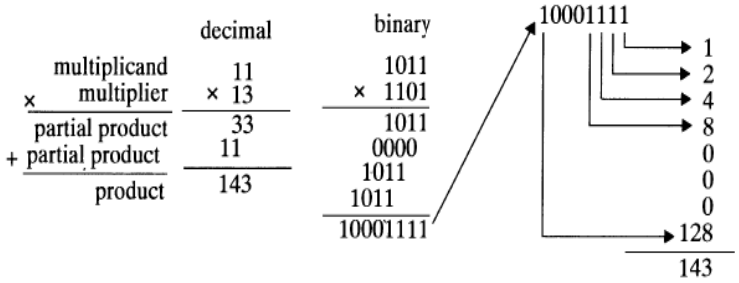

Figure 1. Block Diagram of the Hardware Setup of Project

We can see that, there is no carry unlike the decimal multiplication which makes the hardware easier. From above example we can see that the partial products were either zero or same as multiplicand except that they were shifted to left.

2.2. Floating Point Arithmetic

There are number of applications that require decimal notations. The non-integer numbers are represented in various forms. Whereas, one of those method has gain wide recognition and is called floating point representation.The real number is represented in scientific notation where the binary word is divided into mantissa and exponent. For instance 123.456 can be represented as 1.23456 x 102. This method solves many problems. However, it has a fixed window of representation, which limits it from representing very large or very small numbers [7-9].

Floating point representation is not straight forward as the schematic of the rest of representation methods. It contains variation which includes the number of bits allocated to the exponent and mantissa, the range of exponents, how rounding is carried out, the action taken in exceptional conditions like underflow and overflow. These days IEEE 754-1985 standard also known as international standard IEC 559 is used vastly. IEEE 754 has defined two types of storage layouts for floating points [10]

Single Precision 32 bits: 1 S-bit, 8 Exponent bits & 23 fraction bits.

Double Precision 64 bits: 1 S-bit, 11 Exponent bits & 52 fraction bits.

A simple floating point multiplication for two binary numbers can be seen below mathematically.

$$(F_1 \times 2^{E_1}) \times (F_2 \times 2^{E_2}) = (F_1 \times F_2) \times 2^{(E_1+E_2)}$$
$$= F \times 2^{E}$$

The multiplication includes the addition of exponents and multiplication of fractions. The product fraction is then normalised and exponent overflow or underflow is observed. Finally the product fraction is rounded. The flow chart diagram for floating point multiplication can be seen in Figure 2.

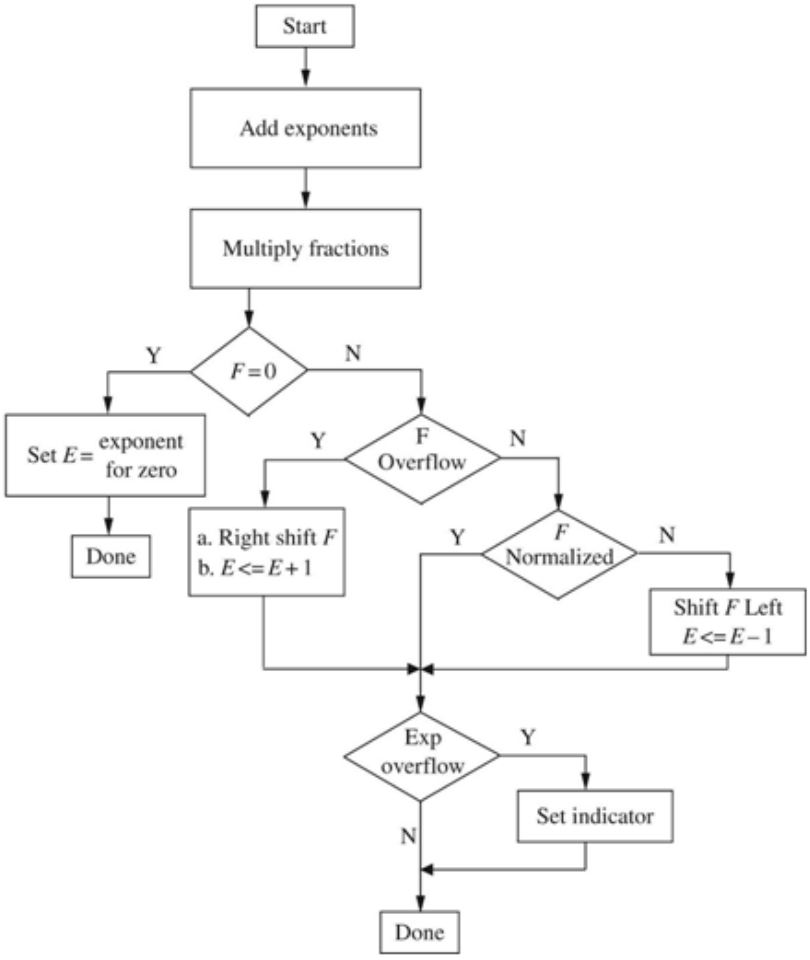

Figure 2. Flowchart for floating point multiplication

3. ALGORITHMS AND ARCHITECTURES

There are many algorithms and architectures used for multipliers some of them are suitable for FPGA while few are not. Here, we will discuss and implement two of these algorithms i.e, bit-serial multipliers and parallel multipliers.

3.1. Bit-Sequential Multiplier

In Bit sequential multiplier also known as bit serial multiplier, input bits arrive on a single wire over a number of clock cycles and output bits are produces serially (Figure 3). Digital serial architecture process more than one bits at a time.

This is done by dividing the N-bit word into X different digits of Y bits wide each, where N is X*Y. The main objective behind this approach is to maximise the speed of operator by a factor greater than its size. It works in the similar manner of manual multiplication of two decimals. This is relatively slow because adding N partial product requires N clock cycles. The easiest clocking scheme is to make use of a system clock. If the multiplier is embedded in a larger system. The clock in normally much slower than the maximum speed at which the simple iterative multiplier can be clocked, so if the delay is to be minimized and expensive and tricky clock multiplier is needed or the hardware must be self clocked [2].

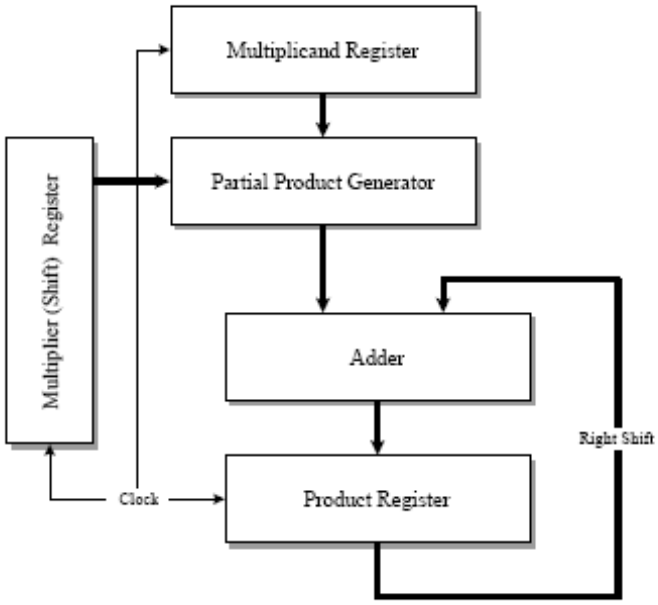

Figure 3. Simple bit-serial multiplier

3.1.1. Fixed Point 16x16 Bit-serial Design

To design the fixed point 16x16 Bit Serial multiplier 33 bit accumulator was used to store the result. Bit adders were designed using "and", "or" and "xor" gates [3]. A state machine was designed for multiplication to shift the bits. Total number of 32 states were used to perform the operation. when state is 0 and st is 1 the multiplier is loaded into accumulator and at the same time state changes to 1. The function named add16 is used to computes two 16 bit vectors whose result is 17 bit. This represents the adder output which is loaded into ACC and at the same time the state counter is incremented. The right shift on the ACC is accomplished by loading ACC with 0 concatenated with upper 32 bits of accumulator. The states will define the number of shifting and bit to bit addition of multiplier and multiplicand. Figure 4 is the simulation result of the fixed point bit serial multiplier.

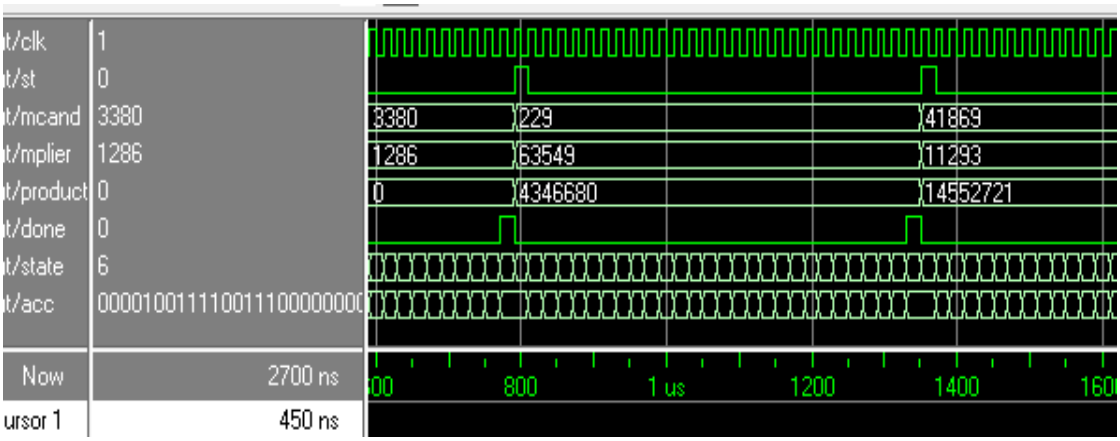

Figure 4. 16x16 Bit Serial Fixed point multiplier simulation

We can seen that at 800ns 16 bit multiplicand having value of 229 in decimal is multiplied with the multiplier of 63549 value in decimal and their product is 14552721 which is at 1350ns because all the states are finished and the done signal is up at this stage.

3.1.2. Floating Point 16x16 Bit-serial Design

The same multiplier is designed for greater number of bits by using greater accumulator size. Floating Point 16x16 Bit serial design 16 bits for multiplicand and multipliers were divided into 10 bit mantissas and 6 bits of exponents. The 10x10 bit serial multiplication for mantissas of the 16x16 bit floating point multiplier. The 10x10bit multiplication is done in the similar manner as done above for the fixed point multiplier. The exponents are added to find the exponent of the product. Afterwards resulted exponents were normalised and adjusted the decimal point for the product. The final exponent and mantissa of the product is loaded properly according to floating point representation.
The simulation results of the floated point can be seen below in Figure 5.

$X=(0101110010101111)_{2(floating)} = (44000.25)_{10}$
$Y=(0000000000000000)_{2(floating)} = (0)_{10}$
$Z=X*Y=(0001111010101111)_{2(floating)}$
$= (0.00002047)_{10} \approx (0)_{10}$

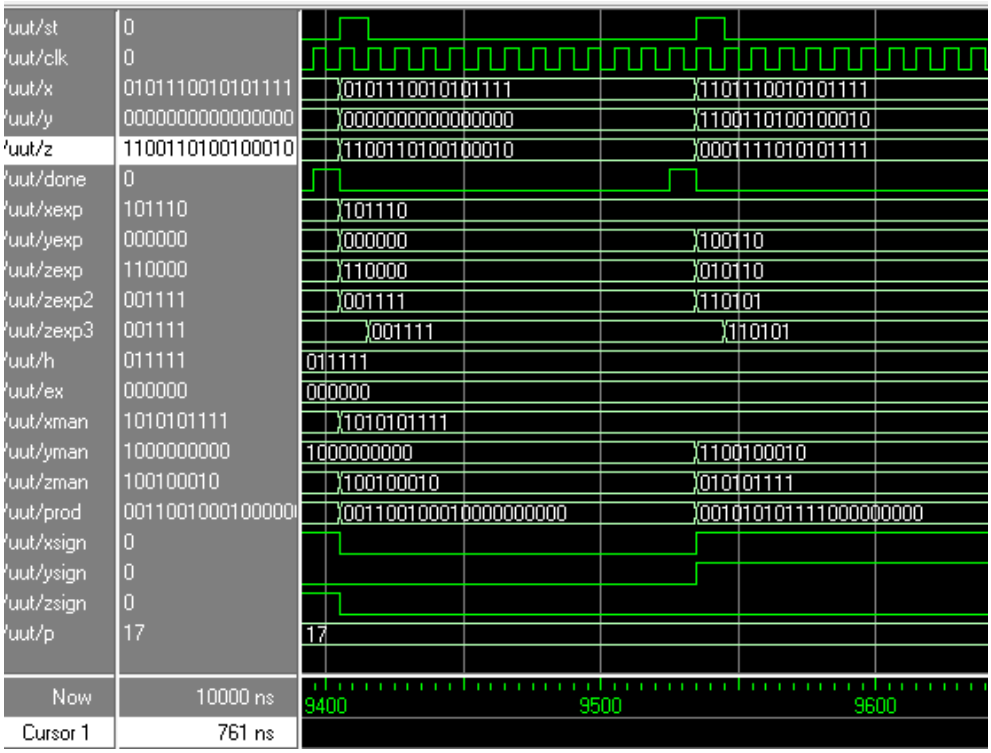

Figure 5. 16x16 Bit Serial Floating point multiplier simulation

Note that representations of 0 is an irritating anomaly, but the IEEE committee decided that it was benign compared to the complexity of any techniques they considered for avoiding it. Floating point representation has some limitations which includes the length of mantissa for example 0000000000000000 is when converted to decimal it comes to 4.65e-10. This is because of the non zero digit to the left side of the mantissa when converting it to the decimal. This is the same reason why we are getting not exact values for our examples.

3.2. Parallel Multiplier

In bit parallel design all the input bits of the sample word are read in single clock cycle and output bits are read together. In addition, when the partial products are to be added the adders are connected in parallel. This requires no more hardware than a linear array but does have more complex interconnections. The time require to add partial products is now proportional to log N, so this can be much faster than large values of N. On other hand the extra complexity in the interconnection may result in greater size and more delay [4]. The partial products can be formed by an array of AND gates. These partial products can be added

together using three adders to add the four partial products after partial products are shifted and filled with zeros. For instance, A0 is 00001011 and A1 is 00000000 A2 is 00101100 and A3 is 01011000 whereas the partial product is equal to A0+A1+A2+A3. Another approach can be tried using carry save adder arrays [5].

3.2.1. Fixed Point 16x16 bits Parallel Multiplier Design

To design the fixed point parallel multiplier in VHDL 16 bit multiplier and multiplicand vectors were used and other vectors for partial product PP, partial sum PS, and partial carry PC are declared. Multiplier and multiplicand was loaded onto first layer of partial product. Full adderswere implemented for the addition of partial products [6]. Whereas, the product output from the partial sum forms the most significant bit of the product from the left most full adder's carry output. Figure 6 is the 16x16 fixed point multiplier. We can see from the figure that there is no clock because the design has generate statements. Moreover, the product is shown at the same time in parallel multiplier.

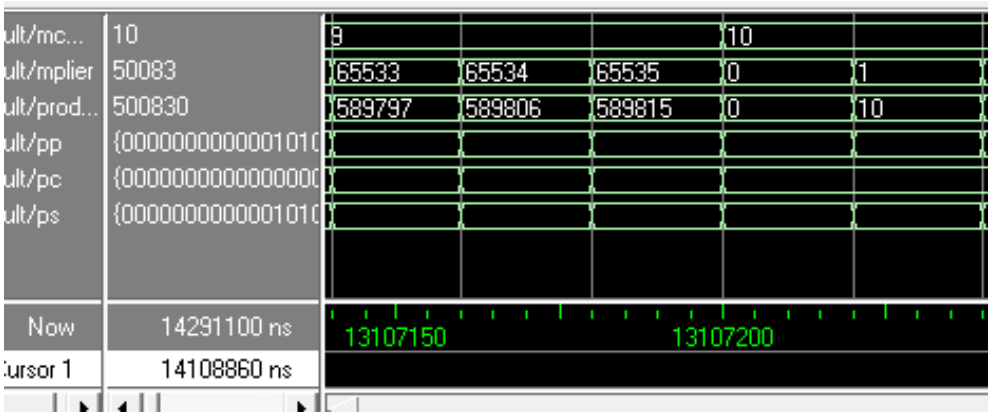

Figure 6.16x16 Bit Fixed point Parallel multiplier simulation

3.2.2. Floating Point 16x16 bits Parallel Multiplier Design

Design of the floating point parallel multiplier is similar to that of bit serial parallel multiplier. The only difference is that instead of multiplying the 10 bit mantissa using bit serial multiplication we use parallel multiplication technique. The rest of the code is similar and need not many changes. The simulation results of the floated point can be seen in Figure 7.

$X=(1101110010101111)_{2(floating)} = (44000.25)_{10}$
$Y=(1100110100100010)_{2(floating)} = (200.5)_{10}$
$Z=X*Y= (0110110000011010)_{2(floating)}$
$= (8814592)_{10} \approx (8822050.125)10$

4. IMPLEMENTATION RESULTS

The main objective of this research is to compare different multiplier algorithms. The multiplier designs presented above are designed using VHDL and synthesised using Xilinx ISE 13.1. Here, we present the implementation results of multiplier designs.

The performance measures considered here are cell usage (Area) and delay [8]. Two different techniques of multiplier designs are investigated for fixed point for the various size of the multipliers. These sizes provides fair comparison of the two fixed point multipliers described in previous sections. Similarly for floating point multipliers 16 bit coefficient of the same multiplication techniques were implemented and their comparison is made between themselves and fixed point multiplication. During synthesis similar constraints and operating conditions were made as far as possible.

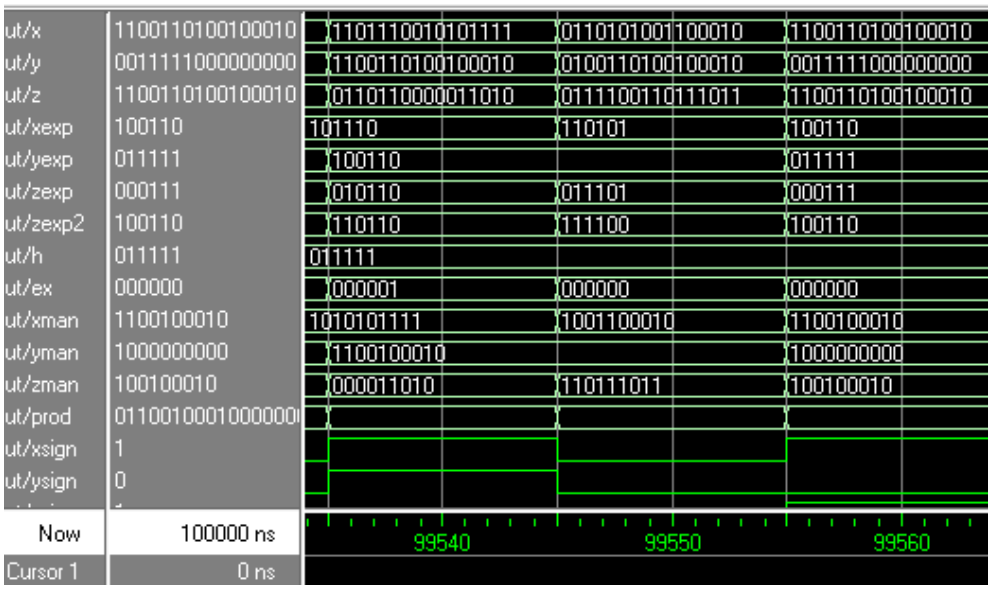

Figure 7. 16x16 Bit Floating point Parallel multiplier simulation

The table below gives the implementation results from synthesis of these multipliers using Xilinx ISE 13.1 tool. The best results based upon compiled values on maximum speed and minimum area is presented here. Speed optimization id focused on implementing the design in target technology library with shortest delay and fastest clock rate. Whereas area optimization looks for functional implementation of the design with smallest total cell area in the target library [9]. The area column is taken from the Xilinx synthesis tool. It shows the cell usage which includes the number of slices occupied by each design. Another result taken from the same tool is the estimated maximum clock rate for each design and optimization mode. it also represents the worst case propagation delay for each design in these tables.

Table 1. Comparison results for fixed point multiplier

Fixed Point Multipliers	Serial		Parallel	
	Area	Delay (ns)	Area	Delay(ns)
8x8 bit	61	7.190	71	27.569
16x16 bit	118	8.124	290	52.708
32x32 bit	257	9.816	1194	101.214
64x64 bit	516	11.754	3612	199.053
128x128 bit	772	145.632	8229	391.98

The Table 1 gives the results of fixed point multipliers of different sizes form 8 bits to 128 bits simulated for two different types of multipliers Bit serial andParallel. It can be observed that the parallel multipliers have relatively greater delay and it occupies more area while bit serial multipliers consumes less space and little delay except 128x128 bit multiplier where greater delay can be seen but still its better than the parallel multipliers.

Table 2. Comparison result for floating point multiplier

Floating Point Multiplier (16x16 bit)	Area (No of slices used)	Delay (ns)
Bit Serial	59	9.883
Parallel	108	30.637

Table two gives the synthesis results for the floating point multiplier for 16x16 bit vector. These results can be compared with the fixed point multipliers of same technique to observe the difference.

Now we will compare the fixed point multiplier results with the floating point designs the area for the both multiplier design with 16x16 bit size can be seen in the figure below. Where it can be observe clearly that the floating point design occupied less area than the fixed point multiplier. This result was anticipated earlier and the reason behind it is simple as in floating point multiplier only mantissa bits are multiplier while other bits for sign and exponent are added. Therefore, the number of slices occupied by the floating point is lower in number because few bits are multiplied using the multiplication techniques as shown.

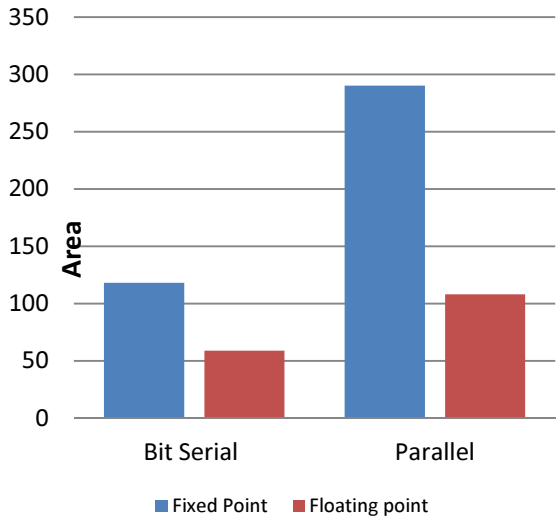

Figure 8. Fixed point and Floating point Area comparison graph

The delay comparison can be seen below between fixed point and floating point multiplier and it can be seen clearly that fixed point multiplier has greater delay than floating point multiplier design whereas in bit serial the delay of both designs are very close to each other.

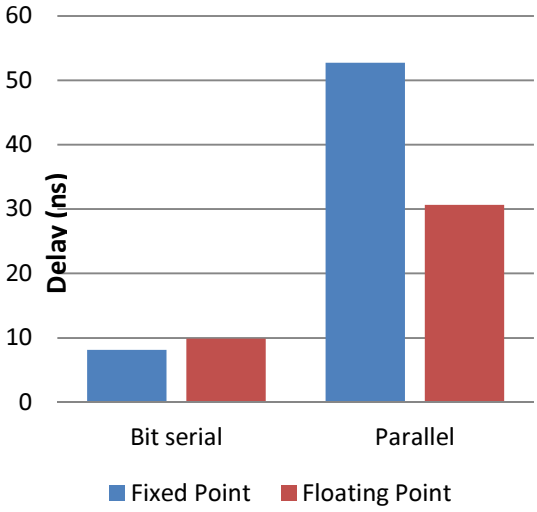

Figure 9. Fixed point and Floating point Delay comparison graph

5. CONCLUSION

Multipliers are used as benchmark designs. Two different multipliers for floating and fixed point multiplication are designed, implemented and studied. A comparative analysis of these multipliers is presented, which is the principal contribution of this research. It was found that as we increase in size the delay as well as area of the hardware increases. Moreover, Parallel multiplier posses more area as well as delay as compared to serial multiplier. Whereas, a floating point design of a parallel multiplier occupies lesser area than the fixed point design of the bit serial multiplier.

REFERENCES

[1] P Denyerand D Renshaw. ''VLSI Signal Processing.A Bit Serial Approach''.
[2] SG Smith, MS McGregor and PB Denyer. "*Techniques to increase the computational throughput of bit serial architectures*". IEEE Proc on Intl Conf on Accoustics, Speech and Signal Processing.
[3] L Kuhnel and Hartmut Schmeck. "A closer look at VLSI Multiplication*". Integration- the VLSI Journal*. Sept 1988
[4] Avizienis A. "Signed digit number representations for fast parallel arithmetic". *IRE Trans, Elec Computer*. Sept 1961.
[5] Baugh CR and BA Wooly. "*A two complement parallel array multiplier*". Dec 1973.
[6] Oscar Gustafsson, Henrikohlsson. "Minimum adder integer multipliers using carry save adders". Linkoping university SWEDON.
[7] *Carl Burch, Hendrix College, September 2011,* ''Floating point representation IEEE-754''
[8] Garry W Bewick. "*Fast multiplication: Algorithms and implementation*". Feb 1994.
[9] L Dadda. "Some schemes for parallel multipliers". *Alta Frequen*za. May 1965.
[10] "IEEE Standard for Binary floating point arithmetic". *ANSUIEEE std 754-1985*, Newyork Aug 1985.