# Complex Multipliers:Implementation using Efficient Algorithms for Signal Processing Application

**Aniket Kumar, R.P. Agarwal**

***Abstract*: The research efforts in low power electronic devices and the cellular networks has been strengthened with the continuous growth in mobile and portable systems . In the modern era there are various portable applications that needs low power(smaller & efficient battery) and higher mili ampere hour then before. Due to this, design of low power devices has now become a significant Performance criteria. While considering the elementary structure of Finite impulse Response Filter, that is the arrangement of multipliers(which is a systematic arrangements of adders) and dely. This manuscript represents the simulation , implementation & analysis report for performance evaluation to minimize delay & RAM consumption during calculation procedure. In this manuscript, we have coded , simulated & implemented selected multipliers such as Vedic, Wallace, Dadda, Booth, Array & Sequential multiplier. Comparative analysis has been done using Xylinx 14.4 with family Spartan6, device as xc6slx45, package csg324 with speed grade of -3 for bit length 2,4,8,16 & 32 using Wallace, dada, Sequential, array, Vedic & Booth Algorithm respectively.***

***Keywords* : *Array ; FPGA ; LUTs; Memory; Vedic.***

## I. INTRODUCTION

The requirement of high speed and low memory requirement has created the need of design and optimization of low power VLSI circuits. A binary multiplier is an digital circuit used in signal processing to multiply two binary numbers. Multipliers plays a vital role in many high performance systems such as filters , speech enhancement , image processing , filtration , compression etc. which requires mathematical calculations and more hardware and area for its implementation. The delay in processing the mathematical operation is more, hence the need to reduce delay in a process become significant [1]. To achieve this, designer focuses on the adders & algorithms to be used in the multiplier architecture as the propagation delay in data outputs depends majorly on the propagation delay in carry output and dependency of it on cascaded adders, thus the speed and cost of digital equipment.

In the binary numbers as the digits are limited by base two, the result of multiplying any binary number by a single binary number is either 0 or the number itself which makes the forming of intermediate- partial products simple and efficient and for the multiplier summing these partial products is time consuming task. One of the techniques to reduce this tedious task is, forming partial product at the time and then sum them as they are generated [3]. With this approach, applications using operations of multiplication may be implemented directly.

## II. MULTIPLIER ALGORITHM

The technique or the way to multiply two numbers is discussed in a algorithm, depending on the bit-length of the numbers & applications, different algorithms are in use.

### *1. Vedic Algorithm*

Urdhva –Tiryakbhyam(vertical and crosswise) sutra of ancient vedic mathematics is the basis of Vedic Algorithm[7]. The technique used in this algorithm is discussed properly in series of steps for calculating product of two decimal number.

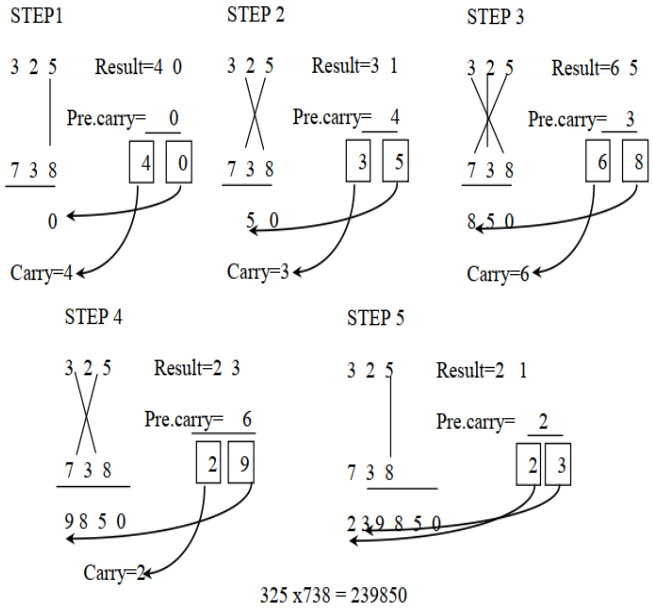



**Fig.1. Multiplication of two decimal numbers by UrdhvaTiryakbhyam**

### *2. Array Algorithm*

**Revised Manuscript Received on November 15, 2019**

***** Correspondence Author

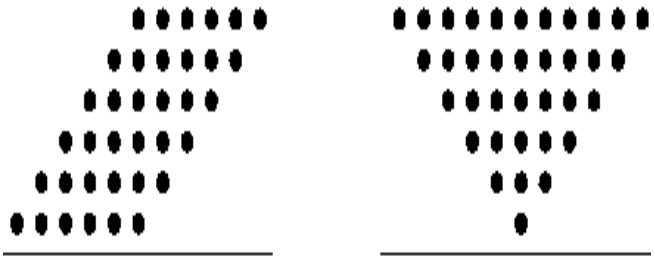
**Aniket Kumar***, Department of Electronics & Communication, Shobhit University,Modipuram, Meerut, India.

**R.P. Agarwal**, Department of Electronics & Communication, Shobhit University,Modipuram, Meerut, India.

For regularity in circuit an array multiplier is well known. The algorithm circuit is based on the loop sequenced addition and sifting procedure. Partial products are generated by multiplication of number by one multiplier digit and the partial products are shifted by one step per bit sequence superimposed. The addition is performed with ripple carry adder,N( N-1 )adders and N *N AND gates are needed for multiplying N*N bits.

### *3. Wallace Algorithm*

For large operation, Wallace algorithm offers faster response. The partial product matrix is rearranged as a tree-like format, reducing each critical path and the amount of adder cells required as illustrated in fig. 2.



Partial Product- Matrix [Array structure]

Reorganized- Matrix [Wallace Tree Structure]

**Fig. 2. Tree- structure of partial product matrix**

The Wallace tree formula belongs to a unit , known as column compression multipliers [3]. The technique used in this algorithm is to obtain partial accumulation of products that progressively reduces the number of bits in each column. The ability of full adder to accept three inputs and provide two output , known as 3:2 compressor, three bits from a partial product column is reduced to two, one bit in the same column and one in the successive column, similarly half adder known as 2:2 compressor for its ability to accept two bits from the column of partial products and provide two output, one in the same column and one in the successive column.

### *4. Dadda Algorithm*

The hardware multiplier designed similar to Wallace multiplier that performs reduction column wise. The technique used is the progression of reduction that is governed by the maximum height sequence $d_j$, defined by $d_1 = 2$ and $d_{j+1} = floor(1.5 d_j)$. This yields a sequence like so: $d_1 = 2, d_2 = 3, d_3 = 4, d_4 = 6.$ . The starting value of $j$ is chosen as the highest value such that $d_j = \min(n_1, n_2)$ where $n_1$ is the number of bits in multiplicand and $n_2$ in multiplier. The behind the algorithm is illustrated in Fig.3 and is explained as If height of column $(c_i) \le d_j$ then the column does not need any reduction therefore shift to column $c_{i+1}$ , else if height of column $(c_i) = d_{j+1}$ , first two elements are added using 2:2 compressor, placing sum at the end of column and the carry at the start of column $c_{i+1}$. Else add the first three elements using 3:3 compressor, placing the result at the end of column $c_i$ and carry in column $c_{i+1}$, the process is repeated to get minimum value of $d_j$. At last look ahead adder is used to get final result.

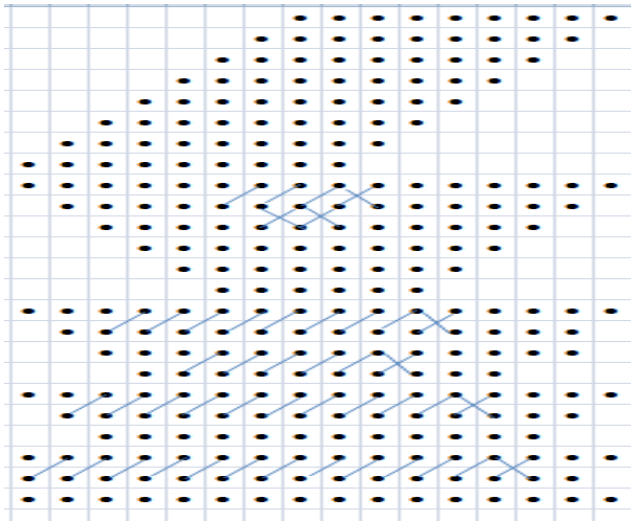


**Fig.3. Dadda structure for 8-bit multiplier**

### *5. Booth Algorithm*

In this algorithm multiplication is generally done in two steps i.e. generation of partial products and addition, the partial product generation is done by encoding. Bits of multiplicand are grouped from left to right and corresponding operations on multiplier is done to obtain the partial product terms.

### *6. Sequential Algorithm*

In this algorithm shift and add method is used to have efficient use of hardware. The technique used in the algorithm is explained by an FSM, multiplying two 4-bit numbers., two register are used one for multiplicand of size n and other for multiplier register of size 2*n, lower n bits space is for storing n-size of multiplier and 2*n size is to store partial product term and finally product.

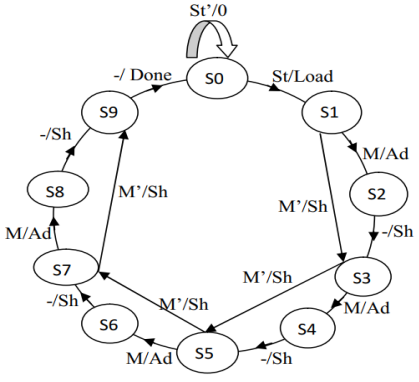
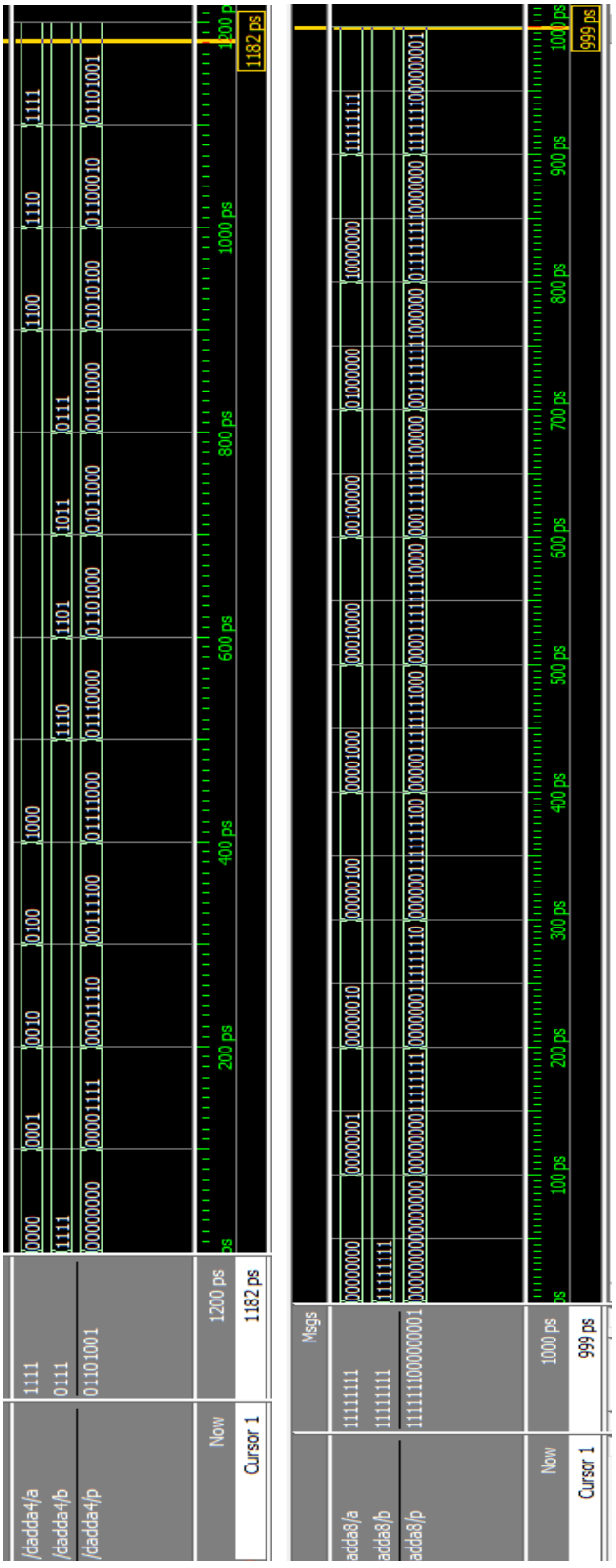



**Fig. 4. FSM**

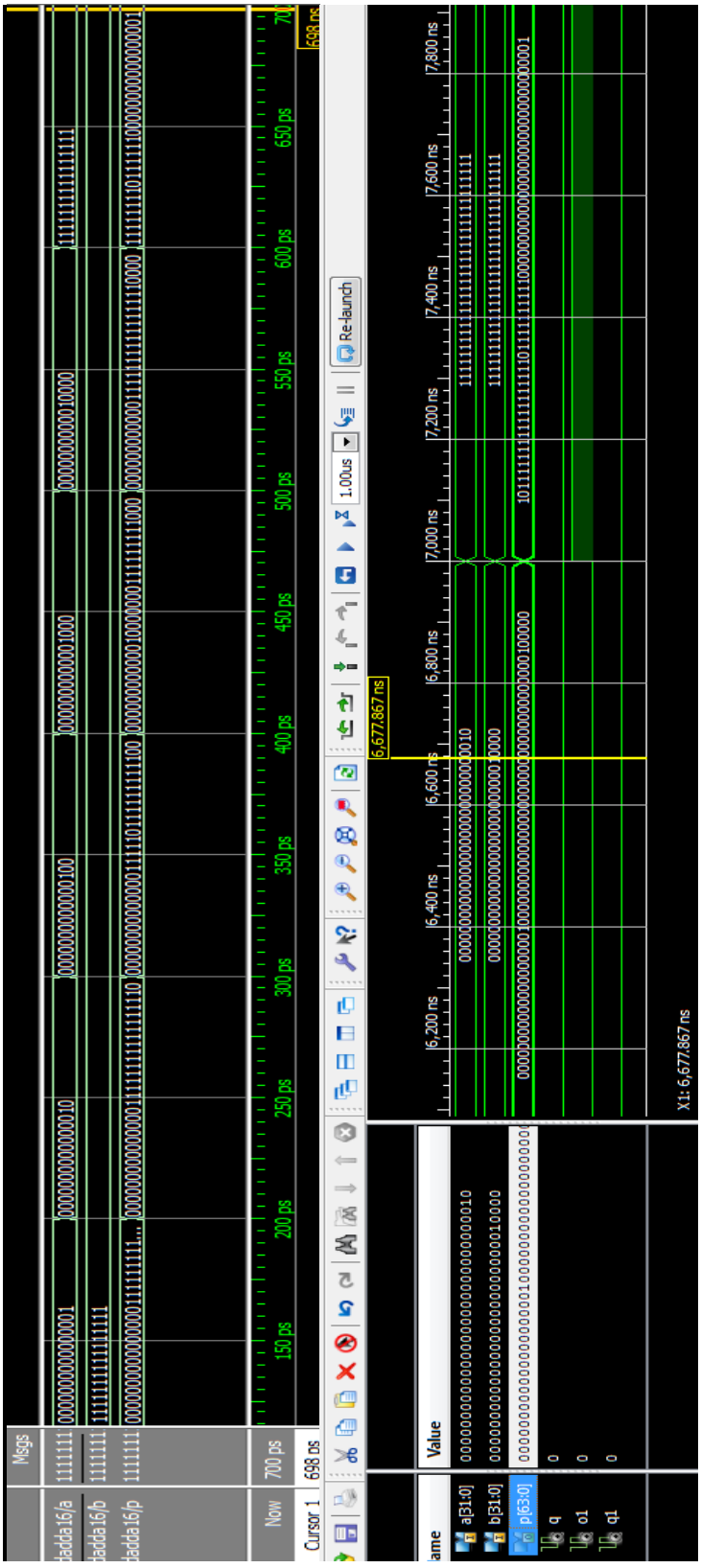
If bit at LSB position is '1', the multiplicand is added to multiplier register and then shifted left by one position (State 1 ,2 & 3 of FSM) else only shifted to left by one-bit position(State 1 & 3 of FSM), the same is repeated at different stages of FSM for 2*n times.

## III. SIMULATION & SYNTHESIS

Simulation has been done on Modelsim &Implementation has been done using Xilinx 14.4 with family Spartan6, device as xc6slx45, package csg324 with speed grade of -3for bit length 2, 4, 8, 16, & 32 using Wallace, Dadda, Vedic, Booth, Sequential & Array algorithm respectively [7].




(a) (b)




(c) (d)

**Fig.5. Simulation results of 4-bit (a), 8-bit (b), 16-bit (c) and 32-bit (d) of Dadda Multiplier**

Above simulation results has been obtained on model sim as well as on xylinx software for all the possible combinations. 4-bit simulation results includes all the possible multiplication ranging from bit size one to four ,

similary simulation results of 32-bit multiplier has been verified for all the bit size ranging from bitsize one to bit size thirty two.

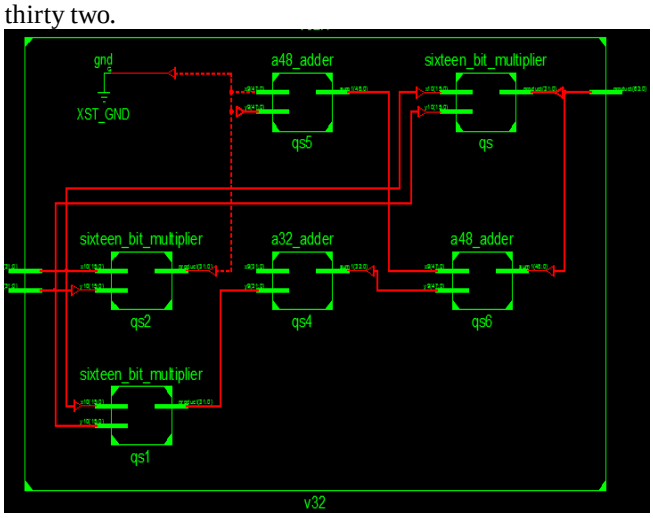



**Fig. 6. RTL view of 32x32Vedic multiplier**

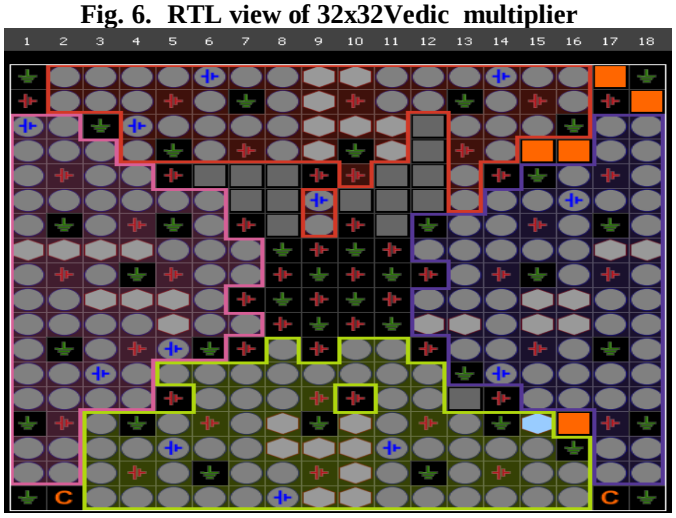



**Fig. 7. I/O planning package of 32x32 Wallace Tree multiplier**

## IV. COMPARATIVE ANALYSIS

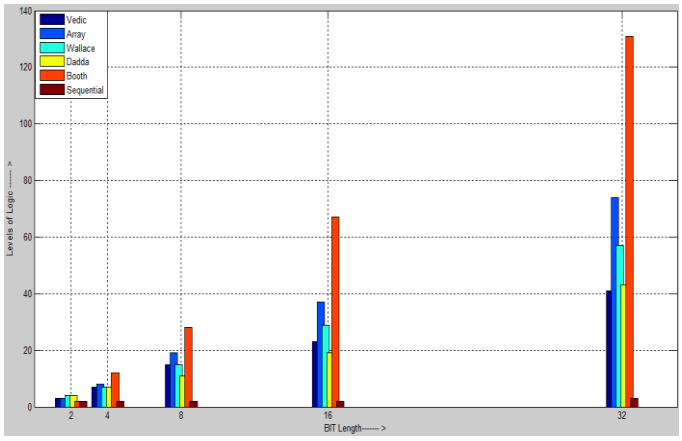



**Fig. 8. Levels of Logic for different bit length**

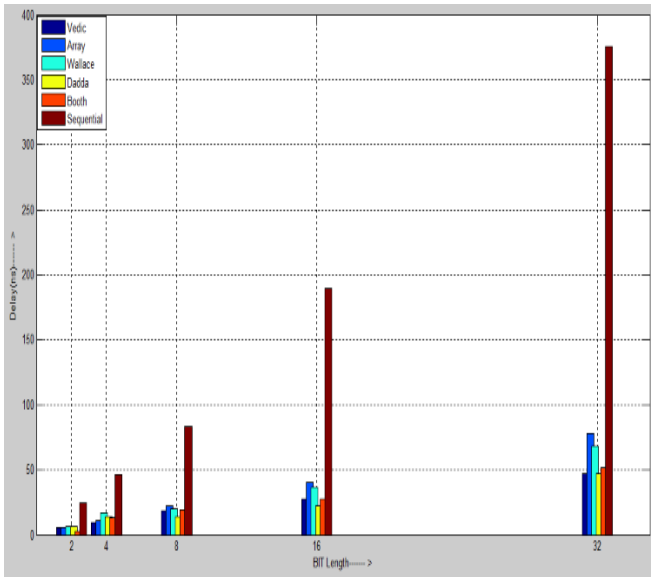



**Fig. 9. Delay in ns for different bit length**

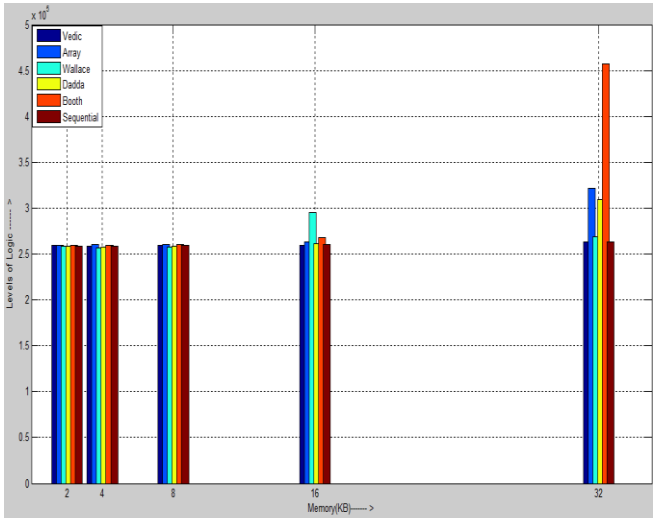



**Fig. 10. Memory usage for different bit length**

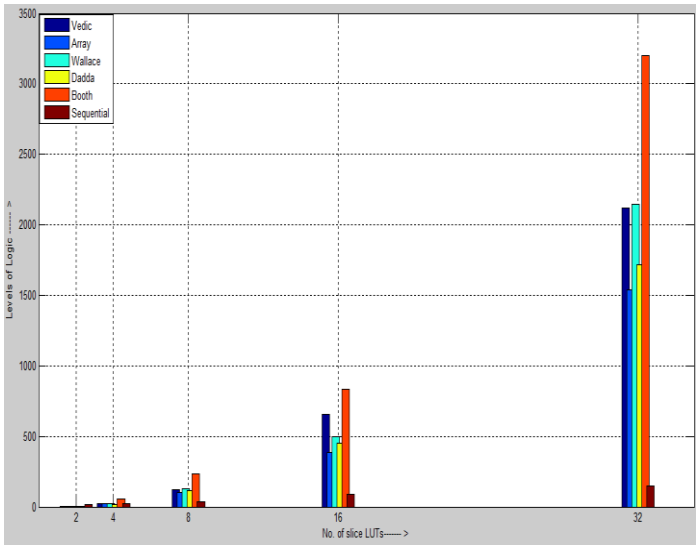



**Fig. 11. LUTs for different bit length**

The comparative results has been obtained with the help of analysis report( generated after simulation of vhdl coding of different algorithms for different bit lengths on xylinx) and MATLAB R2017.The simulation results in tabular form is as below

**TABLE I Comparison b/w Vedic, Array, Wallace, Dadda, Booth &Sequential algorithmfor different bit length on Device Spartan6, XC6SLX45-CSG324**

| Bit Length | Type | Delay (ns) | Levels of Logic | No. of slice LUTs | Memory (KB) |
|---|---|---|---|---|---|
| 2 | Vedic | 5.505 | 3 | 4 | 259592 |
| | Array | 5.505 | 3 | 4 | 259592 |
| | Wallace | 6.494 | 4 | 6 | 258052 |
| | Dadda | 6.494 | 4 | 6 | 258052 |
| | Booth | 2.606 | 2 | 5 | 259080 |
| | Sequential | 4.936 *5=24.68 | 2 | 15 | 258312 |
| 4 | Vedic | 9.557 | 7 | 23 | 258888 |
| | Array | 10.697 | 8 | 23 | 260168 |
| | Wallace | 17.00 | 7 | 23 | 257036 |
| | Vedic | 14.02 | 7 | 19 | 257292 |
| | Booth | 13.548 | 12 | 57 | 259784 |
| | Sequential | 5.140*9=46.26 | 2 | 26 | 258760 |
| 8 | Vedic | 18.270 | 15 | 121 | 259528 |
| | Array | 22.055 | 19 | 99 | 260808 |
| | Wallace | 19.829 | 15 | 128 | 257612 |
| | Vedic | 13.907 | 11 | 114 | 258508 |
| | Booth | 19.081 | 28 | 235 | 260360 |
| | Sequential | 5.194*16=83.104 | 2 | 37 | 259272 |
| 16 | Vedic | 27.278 | 23 | 656 | 259464 |
| | Array | 40.341 | 37 | 387 | 262984 |
| | Wallace | 36.731 | 29 | 499 | 295564 |
| | Vedic | 22.124 | 19 | 453 | 261004 |
| | Booth | 27.742 | 67 | 834 | 267912 |
| | Sequential | 5.745*33=189.585 | 2 | 90 | 260808 |
| 32 | Vedic | 47.146 | 41 | 2120 | 263240 |
| | Array | 78.313 | 74 | 1539 | 321224 |
| | Wallace | 68.383 | 57 | 2147 | 268680 |
| | Vedic | 46.720 | 43 | 1716 | 309640 |
| | Booth | 51.826 | 131 | 3202 | 456712 |
| | Sequential | 5.782x65=375.83 | 3 | 150 | 263240 |

## V. CONCLUSION

From implementation, simulation, comparative graphs [Fig. 5 to 11] & Table- I, it is concluded that Dadda multiplier is best in terms of speed as its delay for 32 bit is 46.720ns followed by Vedic multiplier with delay47.146ns for same bit length. If memory usage is prime concern, then Vedic leads all other multipliers as it uses 263240 KB of memory whereas Wallace uses 268680KB for 32-bit length. If packing is concerned, then Sequential algorithm uses lesser LUT's followed by vedic. From fig. 8 it can be said that with the increase in bit length the delay nature in different algorithms is not of same nature, however it can be said that vedic multipliers performs well in all the terms. Hence this comparison can help us to select a suitable multiplier for a particular task or application.

## VI. FUTURE WORK

This work may be preceded for 64- bit multiplier in order to see its applications for modern days processor. The work may be tested for filtration of signals as ADC & DAC is very common during transmission & reception of signals.

## REFERENCES

1. Aniket K. and R.P. Agarwal, "Simulation & Implementation of Efficient Multiplier Circuits", Proceedings of 2nd International Conference on Recent Multidisciplinary Research (ICRMR-2018), Asian Institute of Technology Conference Centre, Thailand, pp 1-6, 23-24 November 2018.Published in The Online International Interdisciplinary Research Journal (OIIRJ) (ISSN: 2249-9598), UGC J. No. 46964, Vol. 8, Dec. 2018 Special Issue (02), pp. 104-111.
2. Akhilesh G. Naik and Dipankar Pal, "Delay estimation, chip-power analyses and comparision of single-level and multi-level recursive vedic algorithm with conventional algorithms for digital multiplier" Proceeding IEEE International Conference on IC Design & Technology, Otanto, Italy, pp 29-32, June, 4th -6th , 2018.
3. W. Liu, et al., "Design of Approximate Radix-4 Booth Multipliers for Error-Tolerant Computing", IEEE Trans. Computers, vol. 66, no. 8, pp.1435-1441, 2017
4. Gandhi, D.R.; Shah, N.N., "Comparative analysis for hardware circuit architecture of Wallace tree multiplier," Intelligent Systems and Signal Processing (ISSP), 2013 International Conference on , vol., no., pp.1,6, 1-2 March 2013.
5. R Pratibha, P. Sandhya, and R. Varun, "Design of High Performance and Low Power Multiplier using Modified Booth Encoder", IEEE International Conference on Electrical, Electronics and Optimization Techniques (ICEEOT), pp. 794-798, 2016.
6. Khan, S.; Kakde, S.; Suryawanshi, Y., "VLSI implementation of reduced complexity wallace multiplier using energy efficient CMOS full adder," Computational Intelligence and Computing Research (ICCIC), 2013 IEEE International Conference on , vol., no., pp.1,4, 26-28 Dec. 2013.
7. Prakash, A.R.; Kirubaveni, S., "Performance evaluation of FFT processor using conventional and Vedic algorithm," Emerging Trends in Computing, Communication and Nanotechnology (ICE-CCN), 2013 International Conference on , vol., no., pp.89,94, 25-26 March 2013.