Scientific Journal Impact Factor (SJIF): 1.711
International Journal of Modern Trends in Engineering and Research
ISSN (ONLINE) : 2349 - 9745 ISSN (PRINT) : 2393 - 8161

Design, Implementation and performance analysis of 8-bit Vedic Multiplier

Sudhir Dakey[1], Avinash Nandigama[2]
[1]*Faculty ,Department of E.C.E., MVSR Engineering College*
[2]*Student, Department of E.C.E., MVSR Engineering College*

Abstract— Multiplication is the most important arithmetic operation in signaled processing applications. As speed is always a constraint in multiplication operation, increase in speed can be achieved by reducing the number of steps in the computation process. The speed of multiplier determines the efficiency of such system.Vedic mathematics is the name given to the ancient Indian system of mathematics that was rediscovered in early twentieth century. Vedic mathematics is mainly based on sixteen principles or word formulae which are termed as sutras. A simple Vedic multiplier architecture based on Urdhva Triyakbhyam (Vertical and Cross wise) sutra is presented. Performances of Vedic multiplier using various adders are analyzed in this paper. The delay of the multiplier using Carry Select Adder is significantly reduced when compared to Ripple Carry, Carry Increment, Carry Select using Binary to Excess-1 converter and Carry Save Adders.

Keywords — Vedic Multiplier, Urdhva Triyakbhyam, Ripple Carry Adder (RCA), Carry Select Adder (CSLA), Carry Save Adder (CSA), Carry Increment Adder (CIA).

I. INTRODUCTION

A Vedic Mathematics is an ancient system of math practiced during Vedic age which was reconstructed by Jagadguru Swami Sri Bharati Krishna Tirthaji Maharaja between 1911 and 1918 from certain Sanskrit manuscripts. It is perhaps the most refined and efficient mathematical system possible. One of such efficient technique has been employed to enhance the design of a multiplier. Multipliers are the key blocks of a Digital Signal processor. Multiplication is the key aspect, whereby improvement in computational speed of multiplication decreases the processing time of Digital Signal Processors. Convolution, Fast Fourier transforms and various other transforms make use of multiplier blocks.

Urdhva Triyagbhyam is a general multiplication formula applicable to all cases of multiplication which works on the principle of Vertical and Crosswise technique. The choice of which adder architecture to use is of utmost importance, since the performance of adders may determine the whole system performance. Area and power consumption are also relevant figures of merit to be considered, especially when the design targets VLSI realization. Recently, energy-efficiency has also become an important metric due to the dramatic growth of battery powered portable devices.

Therefore, when designing high performance addition-based arithmetic circuits using cell based VLSI design, designers must rely on fast adder architectures that are optimized at the logic-level, which reduces the number of fast adder architecture to a few classic ones, as the Carry-Select Adder (CSLA) and the Carry- Look-ahead Adder (CLA). The performance of Vedic Multiplier using various adders is analyzed in this paper. Among various methods of multiplications in Vedic mathematics, Urdhva Triyagbhyam is used in this paper.

II. URDHVA TRIYAGBHYAM SUTRA

The use of Vedic mathematics lies in the fact that it reduces the typical calculations in conventional mathematics to very simple ones. This is so because the Vedic formulae are claimed to be based on the natural principles on which the human mind works. Vedic Mathematics is a methodology of arithmetic rules that allow more efficient speed implementation. This is a very interesting field and presents some effective algorithms which can be applied to various branches of engineering such as computing.

Urdhva Triyakbhyam Sutra is a general multiplication formula applicable to all cases of multiplication. It means “Vertically and Crosswise”. The digits on the two ends of the line are multiplied and the result is added with the previous carry. When there are more lines in one step, all the results are added to the previous carry. The least significant digit of the number thus obtained acts as one of the result digits and the rest act as the carry for the next step. Initially the carry is taken to be as zero. The line diagram for multiplication of two 4-bit numbers is as shown in Figure 1.

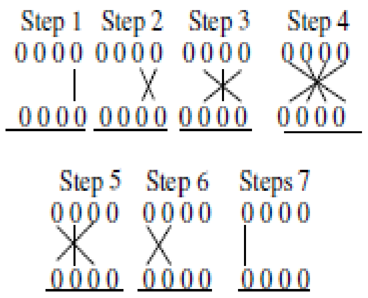


Figure 1: Line diagram for multiplication of two 4-Bit Numbers

For this multiplication scheme, let us consider the multiplication of two decimal numbers (325 × 728). Line diagram for the multiplication is shown in Figure 2. The digits on the two ends of the line are multiplied and the result is added with the previous carry. When there are more lines in one step, all the results are added to the previous carry. The least significant digit of the number thus obtained acts as one of the result digits and the rest act as the carry for the next step. Initially the carry is taken to be as zero.

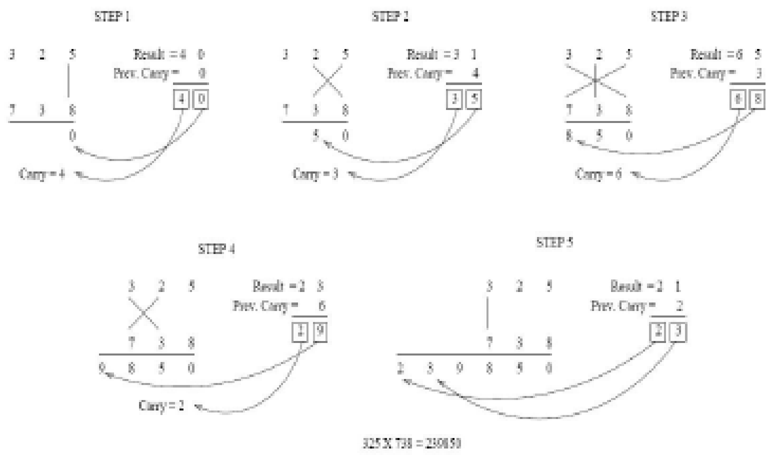


Figure 2: Multiplication of two Decimal Numbers by Urdhva Triyakbhyam

Now we will extend this Sutra to binary number system. For the multiplication algorithm, let us consider the multiplication of two 8 bit binary numbers $A_7A_6A_5A_4A_3$ $A_2A_1A_0$ and $B_7B_6B_5B_4B_3B_2$ B_1B_0. As the result of this multiplication would be more than 8 bits, we express it as $\ldots R_7R_6R_5R_4R_3R_2R_1R_0$. As in the last case, the digits on the both sides of the line are multiplied and added with the carry from the previous step. This generates one of the bits of the result and a carry. This carry is added in the next step and hence the process goes on. If more than one line are there in one step, all the results are added to the previous carry. In each step, least significant bit acts as the result bit and the other entire bits act as carry. For example, if in some intermediate step, we will get 011, then1 will act as result bit and 01 as the carry. Thus we will get the following expressions:

$R_0=A_0B_0$
$C_1R_1=A_0B_1+A_1B_0$
$C_2R_2=C_1+A_0B_2+A_2B_0+A_1B_1$
$C_3R_3=C_2+A_3B_0+A_0B_3+A_1B_2+A_2B_1$
$C_4R_4=C_3+A_4B_0+A_0B_4+A_3B_1+A_1B_3+A_2B_2$
$C_5R_5=C_4+A_5B_0+A_0B_5+A_4B_1+A_1B_4+A_3B_2+A_2B_3$
$C_6R_6=C_5+A_5B_0+A_0B_6+A_5B_1+A_1B_5+A_4B_2+A_2B_4+A_3B_3$
$C_7R_7=C_6+A_7B_0+A_0B_7+A_6B_1+A_1B_6+A_5B_2+A_2B_5+A_4B_3+A_3B_4$
$C_8R8=C_7+A_7B_1+A_1B_7+A_6B_2+A_2B_6+A_5B_3+A_3B_5+A_4B_4$
$C_9R_9=C_8+A_7B_2+A_2B_7+A_6B_3+A_3B_6+A_5B_4+A_4B_5$
$C_{10}R_{10}=C_9+A_7B_3+A_3B_7+A_6B_4+A_4B_6+A_5B_5$
$C_{11}R_{11}=C_{10}+A_7B_4+A_4B_7+A_6B_5+A_5B_6$
$C_{12}R_{12}=C_{11}+A_7B_5+A_5B_7+A_6B_6$
$C_{13}R_{13}=C_{12}+A_7B_6+A_6B_7$
$C_{14}R_{14}=C_{13}+A_7B_7$

C_{14} $R_{14}R_{13}R_{12}R_{11}R_{10}R_9R_8R_7R_6R_5R_4R_3R_2R_1R_0$ being the final product. Hence this is the general mathematical formula applicable to all cases of multiplication. All the partial products are calculated in parallel and the delay associated is mainly the time taken by the carry to propagate through the adders which form the multiplication array.

III. ADDERS

A. Ripple Carry Adder

The Ripple carry adder (RCA) provides the most compact design but takes longer computing time. If there is N-bit RCA, the delay is linearly proportional to N. Thus for large values of N the RCA gives highest delay of all adders. The ripple carry adder is constructed by cascading full adders (FA) blocks in series. One full adder is responsible for the addition of two binary digits at any stage of the ripple carry. The carryout of one stage is fed directly to the carry-in of the next stage. Even though this is a simple adder and can be used to add unrestricted bit length numbers, it is however not very efficient when large bit numbers are used. One of the most serious drawbacks of this adder is that the delay increases linearly with the bit length. The worst-case delay of the RCA is when a carry signal transition ripples through all stages of adder chain from the least significant bit to the most significant bit, which is approximated by

$$t=(n-1)tc+ts$$

Where tc is the delay through the carry stage of a full adder, and ts is the delay to compute the sum of the last stage. The delay of ripple carry adder is linearly proportional to n, the number of bits; therefore the performance of the RCA is limited when n grows bigger. The advantages of the RCA are lower power consumption as well as compact layout giving smaller chip area. The design schematic of RCA is shown in Figure 3.

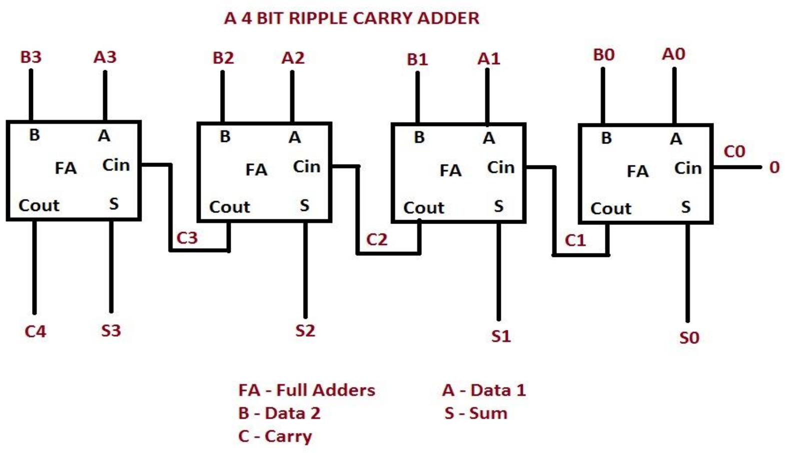


Figure 3: Block Diagram of Ripple Carry Adder

B. Carry Look-Ahead Adder

The carry Look-Ahead Adder (CLA) gives fast results but consumes large area. If there is N-bit adder, CLA is fast for N≤4, but for large values of N its delay increases more than other adders. So for higher number of bits, CLA gives higher delay than other adders due to presence of large number of fan-in and a large number of logic gates.

Carry look-ahead adder is designed to overcome the latency introduced by the rippling effect of the carry bits. The propagation delay occurred in the parallel adders can be eliminated by carry look ahead adder. This adder is based on the principle of looking at the lower order bits of the augends and addend if a higher order carry is generated. This adder reduces the carry delay by reducing the number of gates through which a carry signal must propagate.

Carry look ahead depends on two things: Calculating for each digit position, whether that position is going to propagate a carry if one comes in from the right and combining these calculated values to be able to deduce quickly whether, for each group of digits, that group is going to propagate a carry that comes in from the right. The net effect is that the carries start by propagating slowly through each 4-bit group, just as in a ripple-carry system, but then moves 4 times faster, leaping from one look ahead carry unit to the next. Finally, within each group that receives a carry, the carry propagates slowly within the digits in that group.

This adder consists of three stages: a propagate block/ generate block, a sum generator and carry generator. The generate block can be realized using the expression.

$$Gi= Ai.Bi \qquad \text{for } i=0,1,2,3$$

Similarly the propagate block can be realized using the expression

$$Pi=Ai\text{^}Bi \qquad \text{for } i=0,1,2,3$$

The carry output of the (i-1)th stage is obtained from

$$Ci(cout)=Gi+Pi.Ci-1 \qquad \text{for } i=0,1,2,3$$

The sum output can be obtained

$$Si=Ai\text{^}\ Bi\text{^}\ Ci-1 \qquad \text{for } i=0,1,2,3$$

An 8 bit look ahead adder using two four bit look ahead block is shown in Figure 4.

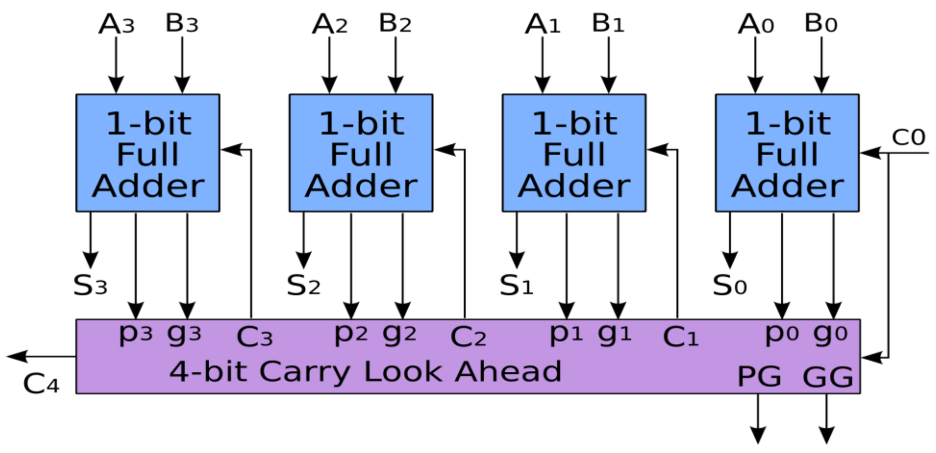


Figure 4: Block Diagram of Carry Look Ahead Adder

C. Carry Save Adder

The carry-save unit consists of n full adders, each of which computes a single sum and carries bit based solely on the corresponding bits of the three input numbers. The entire sum can then be computed by shifting the carry sequence left by one place and appending a 0 to the front (most significant bit) of the partial sum sequence and adding this sequence with RCA produces the resulting n + 1-bit value. But as the number of bits increases the area of it also increases.

The carry-save adder reduces the addition of 3 numbers to the addition of 2 numbers. The propagation delay is 3 gates regardless of the number of bits. The carry-save unit consists of n full adders, each of which computes a single sum and carries bit based solely on the corresponding bits of the three input numbers. The entire sum can then be computed by shifting the carry sequence left by one place and appending a 0 to the front (most significant bit) of the partial sum sequence and adding this sequence with RCA produces the resulting n+ 1-bit value. This process can be continued indefinitely, adding an input for each stage of full adders, without any intermediate carry propagation. These stages can be arranged in a binary tree structure, with cumulative delay logarithmic in the number of inputs to be added, and invariant of the number of bits per input. The main application of carry save algorithm is, well known for multiplier architecture is used for efficient CMOS implementation of much wider variety of algorithms for high speed digital signal processing .CSA applied in the partial product line of array multipliers will speed up the carry propagation in the array. The design schematic of Carry Save Adder is shown in Figure 5.

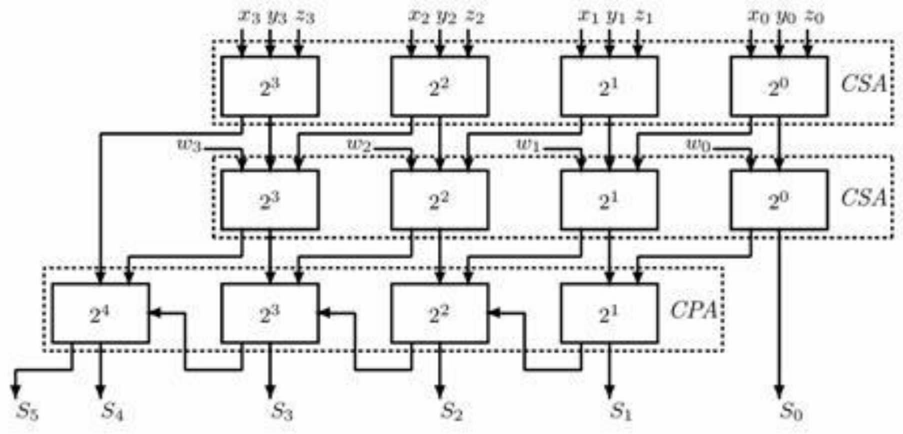


Figure 5: Block diagram of Carry Save Adder

D. Carry Increment Adder

An 8-bit increment adder includes two RCA (Ripple carry adder) of four bit each. The first ripple carry adder adds a desired number of first 4-bit inputs generating a plurality of partitioned sum and partitioned carry. Now the carry out of the first block RCA is given to CIN of the conditional increment block. Thus the first four bit sum is directly taken from the ripple carry output. The second RCA block regardless of the first RCA output will carry out the addition operation and will give out results which are fed to the conditional increment block. The input CIN to the first RCA block is given always low value. The conditional increment block consists of half adders. Based on the value of cout of the 1st RCA block, the increment operation will take place. Here the half adder in carry increment block performs the increment operation. Hence the output sum of the second RCA is taken through the carry increment block. The design schematic of Carry Increment Adder is shown in Figure 6.

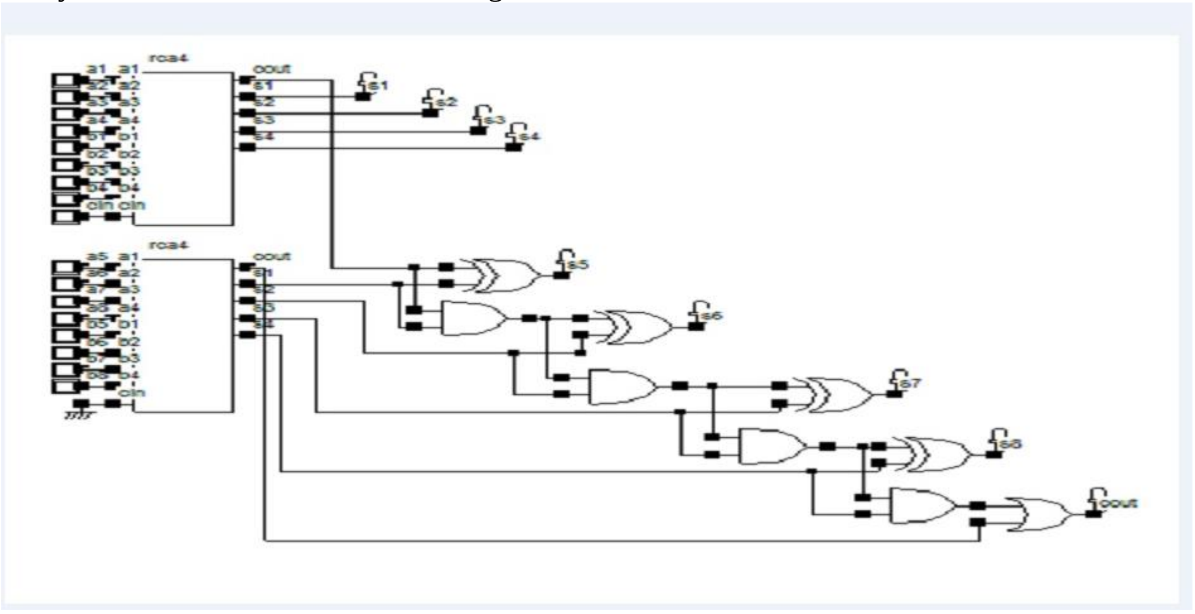


Figure 6: Block Diagram of Carry Increment Adder

E. Carry Select Adder

A carry select adder is divided into groups, each of which (except for the least-significant bit group) performs two additions in parallel, one assuming a carry-in of zero, the other a carry-in of one. The carry select adder is simple but rather fast, having a gate level depth of O√N. Adding two n-bit numbers with a carry select adder is done with two ripple carry adders in order to perform the calculation twice, one time with the assumption of the carry being zero and the other assuming one. After the two results are calculated, the correct sum, as well as the correct carry, is then selected with the multiplexer. It’s faster than RCA by performing additions in parallel and reducing the maximum carry path.

A carry-select adder is divided into sectors, each of which – except for the least-significant – performs two additions in parallel, one assuming a carry-in of zero, the other a carry-in of one. A four bit carry select adder generally consists of two ripple carry adders and a multiplexer. The carry-select adder is simple but rather fast. Adding two n-bit numbers with a carry select adder is done with two adders (two ripple carry adders) in order to perform the calculation twice, one time with the assumption of the carry being zero and the other assuming one. After the two results are calculated, the correct sum, as well as the correct carry, is then selected with the multiplexer once the correct carry is known. The design schematic of Carry Select Adder is shown in Figure 7. A carry-select adder speeds 40% to 90%faster than RCA by performing additions in parallel and reducing the maximum carry path.

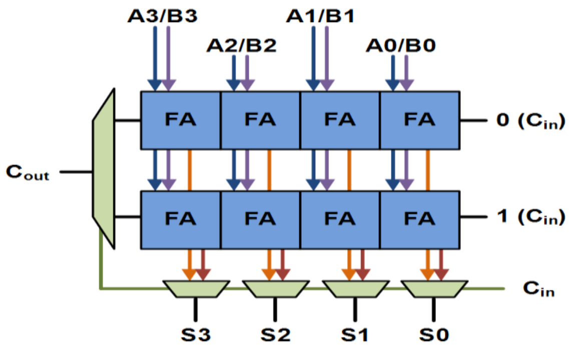
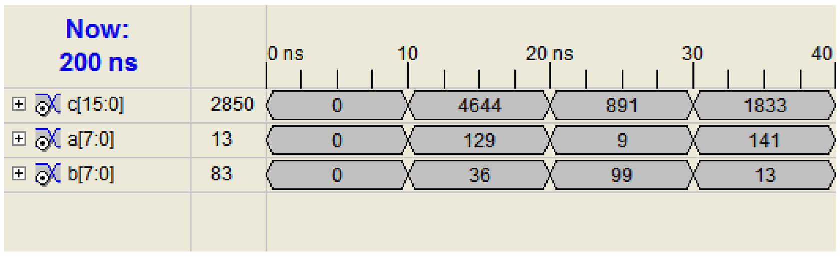


Figure 7: Block Diagram of Carry Select Adder

IV. SIMULATION RESULTS

The simulation is carried out for the Vedic multiplier using various adders. The performance evaluation of Vedic multiplier using adder designs (Ripple Carry Adder, Carry Select Adder, Carry Look Ahead Adder, Carry Save Adder and Carry Select Adder using Binary to Excess 1 Converter) is carried out using Xilinx 8.1 and implemented on Spartan 3E FPGA. The multipliers and adders are compared for word size of 8 bit.

A. Simulation result of 8-bit Vedic Multiplier using Ripple Carry Adder



Now: 200 ns		0 ns	10	20 ns	30 / 40
c[15:0]	2850	0	4644	891	1833
a[7:0]	13	0	129	9	141
b[7:0]	83	0	36	99	13

Figure 8: 8-bit Vedic Multiplier using Ripple Carry Adder

Figure 9 shows the RTL schematic of Vedic Multiplier using RCA obtained in simulation.

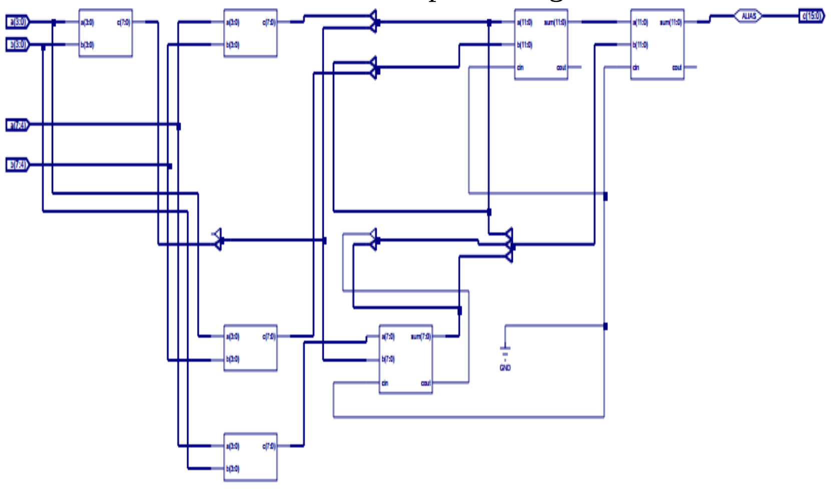
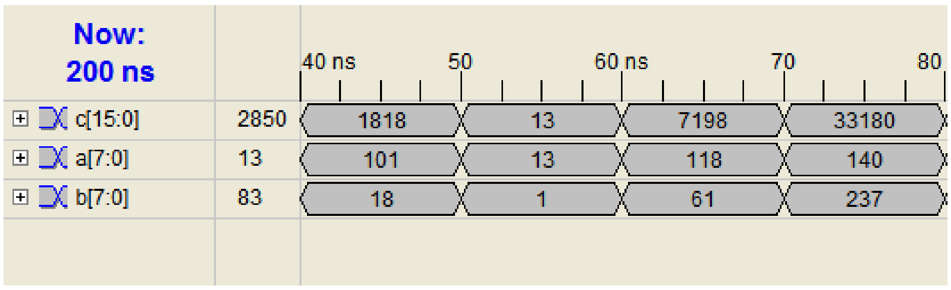


Figure 9: RTL schematic of 8-bit Vedic Multiplier using Ripple Carry Adder

B. Simulation result of 8-bit Vedic Multiplier using Carry Select Adder



```
Now:
200 ns              40 ns      50       60 ns      70        80
c[15:0]    2850     1818       13       7198       33180
a[7:0]     13       101        13       118        140
b[7:0]     83       18         1        61         237
```

Figure 10: 8-bit Vedic Multiplier using Carry Select Adder

Figure 11 shows the RTL schematic of 8-bit Vedic Multiplier using CSA obtained in simulation.

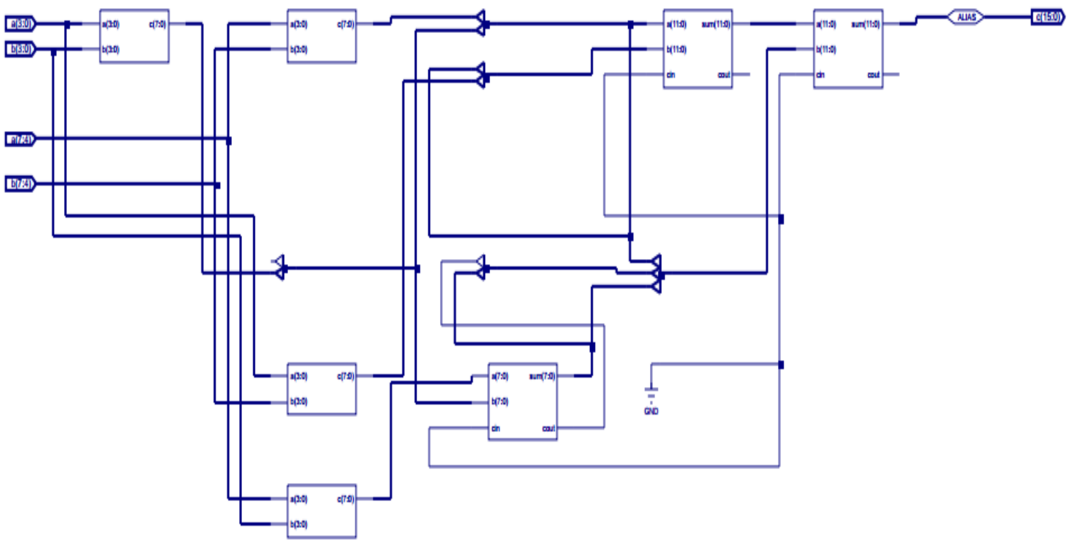
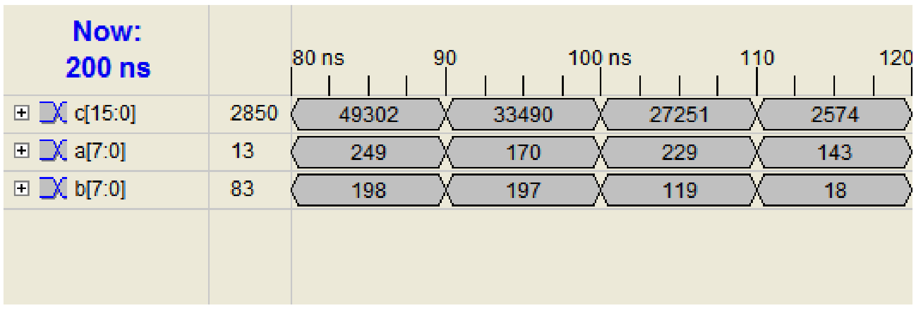


Figure 11: RTL schematic of 8-bit Vedic Multiplier using Carry Select Adder

C. Simulation result of 8-bit Vedic Multiplier using Carry Look Ahead Adder



```
Now:
200 ns              80 ns      90       100 ns     110       120
c[15:0]    2850     49302      33490    27251      2574
a[7:0]     13       249        170      229        143
b[7:0]     83       198        197      119        18
```

Figure 12: 8-bit Vedic Multiplier using Carry Look Ahead Adder

Figure 13 shows the blocks of Vedic Multiplier using CLA obtained in simulation.

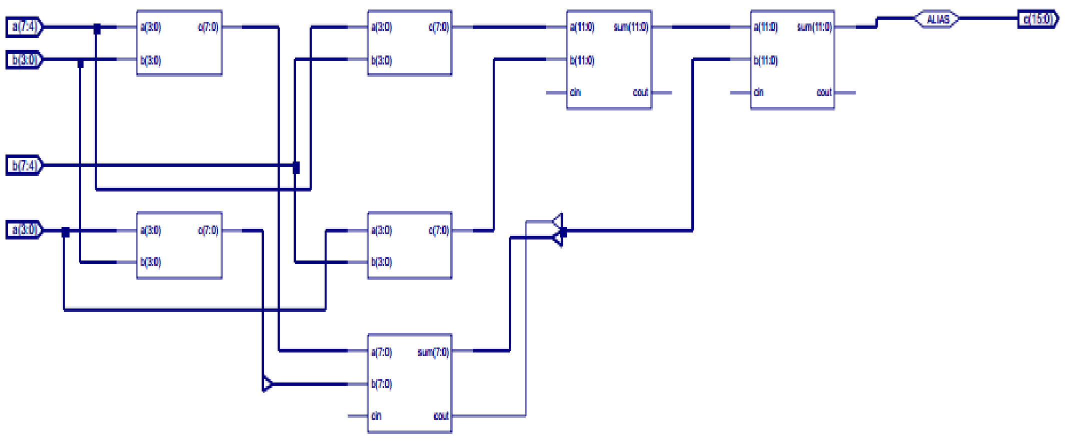
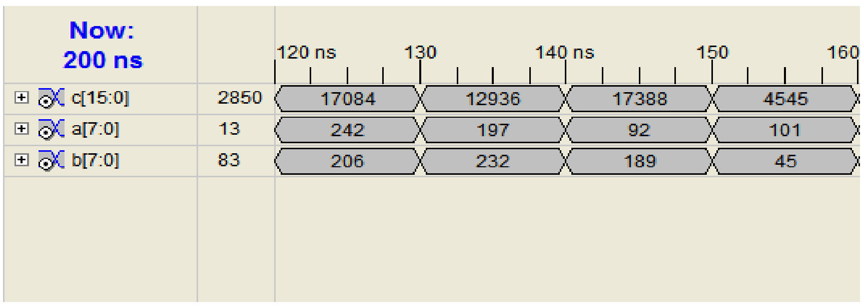


Figure 13: RTL schematic of 8-bit Vedic Multiplier using Carry Look Ahead Adder

D. Simulation result of 8-bit Vedic Multiplier using Carry Increment Adder



```
Now:
200 ns            120 ns      130        140 ns      150        160
c[15:0]   2850    17084       12936      17388       4545
a[7:0]    13      242         197        92          101
b[7:0]    83      206         232        189         45
```

Figure 14: 8-bit Vedic Multiplier using Carry Increment Adder

Figure 15 shows the RTL schematic of Vedic Multiplier using CIA obtained in simulation.

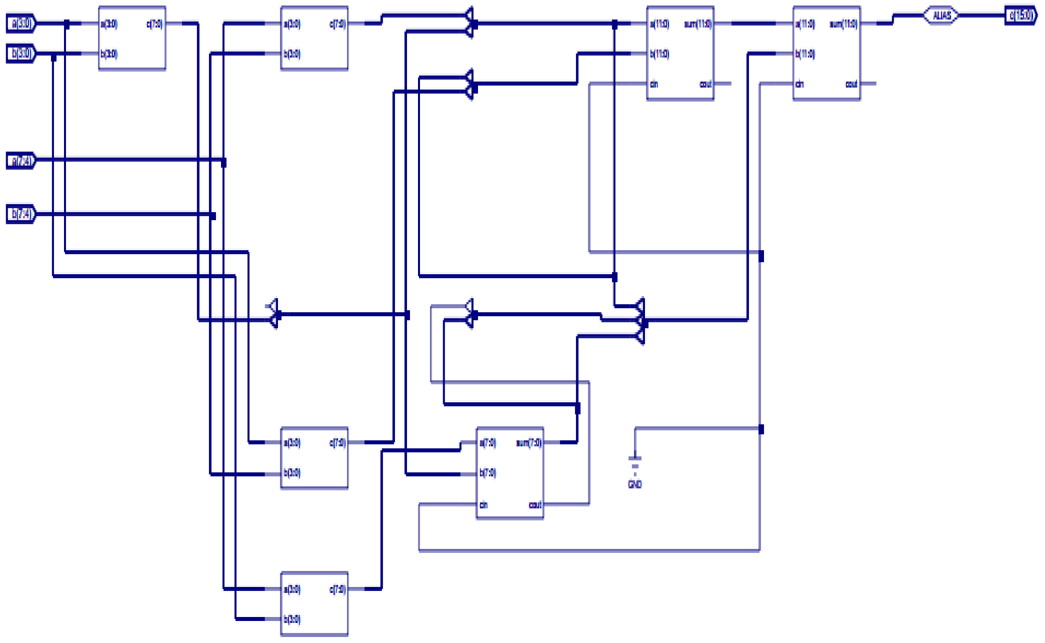


Figure 15: RTL schematic of 8-bit Vedic Multiplier using Carry Increment Adder

E. Simulation result of 8-bit Vedic Multiplier using Carry save Adder

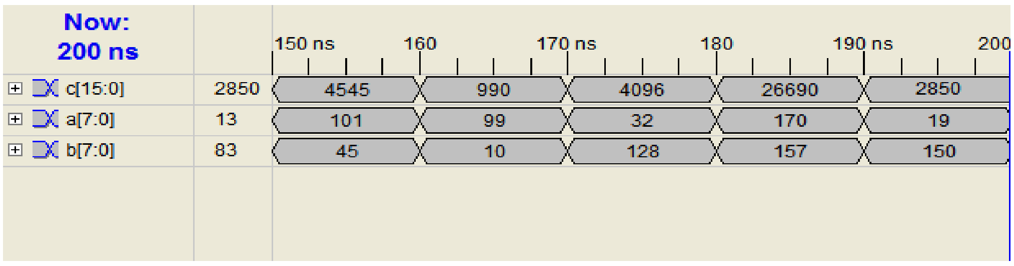

Now:
200 ns
150 ns
160
170 ns
180
190 ns
200
c[15:0] 2850 4545 990 4096 26690 2850
a[7:0] 13 101 99 32 170 19
b[7:0] 83 45 10 128 157 150

Figure 16: 8-bit Vedic Multiplier using Carry Save Adder

Figure 17 shows the RTL schematic of Vedic Multiplier using Carry Save Adder obtained in simulation

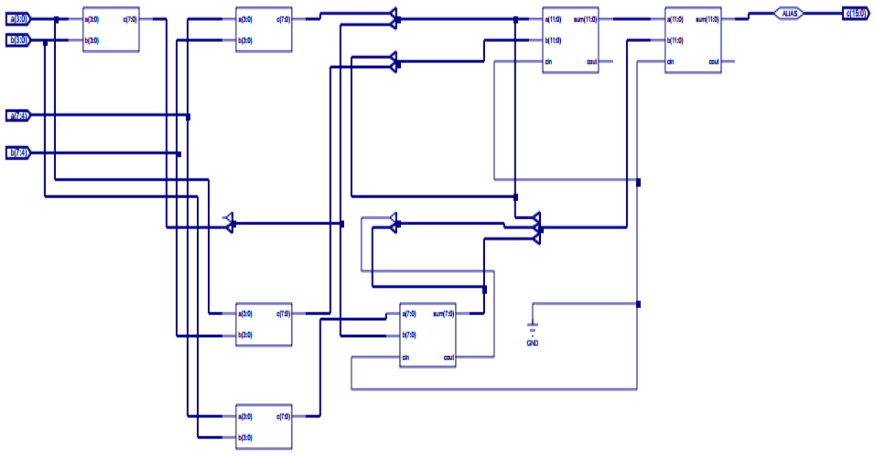


Figure 17: RTL schematic of 8-bit Vedic Multiplier using Carry Save Adder

V. SYNTHESIS RESULTS

The designed Ripple Carry Adder, Carry Select Adder, Carry Save Adder, Carry Look Ahead Adder and Carry Increment Adder are synthesized using Xilinx tool. This tool offers design capture, RTL schematic, Technology schematic and timing analysis. Comparison of all the Multiplier using five Adder designs in 8-bit for delay, area and area delay product is given below in the following tables.

Table 1: Delay Comparison of Vedic Multiplier using Different Adders

Multiplier using different Adders	Delay(ns)
RIPPLE CARRY ADDER (RCA)	36.018
CARRY SELECT ADDER (CSLA)	31.136
CARRY LOOK AHEAD ADDER (CLA)	32.629
CARRY INCREMENT ADDER (CIA)	39.975
CARRY SAVE ADDER (CSA)	36.245

In table 1 it is clear that the delay of Vedic Multiplier using CSLA is the least of all the designs. The delay of CSLA has reduced by 22.1% whereas CLA has reduced by 18.3% when compared to CIA.

Table 2: Look up Tables Comparison of Vedic Multiplier using different Adders

Multiplier using different Adders	Number of Look Up Tables
RIPPLE CARRY ADDER (RCA)	195
CARRY SELECT ADDER (CSLA)	198
CARRY LOOK AHEAD ADDER (CLA)	214
CARRY INCREMENT ADDER (CIA)	305
CARRY SAVE ADDER (CSA)	206

In table 2 it is clear that the Look up Tables is the least for Vedic Multiplier using Ripple Carry Adder. The Look up Tables of Multiplier using RCA has reduced by 36.1% and by using CSA is reduced by 35.1% when compared to CIA.

Table 3: Area-Delay product Comparison of Vedic Multiplier using various Adders

Multiplier using different Adders	(Area*Delay)
RIPPLE CARRY ADDER (RCA)	7055.1
CARRY SELECT ADDER (CSLA)	6164.9
CARRY LOOK AHEAD ADDER (CLA)	6982.6
CARRY INCREMENT ADDER (CIA)	12192.3
CARRY SAVE ADDER (CSA)	7466.4

In table 3 it is clear that the Area-Delay product is the least for the Vedic Multiplier using Carry Select Adder. The Area-Delay product of the Multiplier using Carry Select Adder has 6164.9, by the using RCA is 7055.1 and by using Carry Save Adder is 7466.47.

VI. CONCLUSION AND FUTURE SCOPE

After designing and implementing 8-bit Vedic multiplier using various adder architectures, it is concluded that Vedic multiplier with Carry Select Adder gives fast results in terms of delay. From the tables present in the earlier section, it can be concluded that Carry Select Adder is 22.1% faster and 35.1% smaller in area when compared to Carry Increment Adder.

The work presented in this paper can further be extended by considering other sutras for vedic mathematics like Nikhilam Sutra. The multiplier presented here is 8 bits in width, however it is desirable to have data widths of 32 bits, 64 bits and sometimes even 128 bits. So, this work can be extended for above mentioned widths.

REFERENCES

[1] Premananda B.S, Samarth S. Bai Shashank B, “Design and Implementation of 8-bit Vedic Multiplier”, International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering, Vol. 2, Issue 12, December 2013
[2] R.UMA, VidyaVijayan, M. Mohanapriya, Sharon Paul, “Area, Delay and Power Comparison of Adder Topologies”, International Journal of VLSI design & Communication Systems (VLSICS) Vol.3, No.1, February 2012.
[3] M. Basak, M. Sutradhar, B. Santra, M. Saha, D. Chowdhury, J. Samanta “STUDY THE PERFORMANCE ANALYSIS OF LOW POWER–HIGH SPEED CARRY SELECT ADDER “ ,International Journal of VLSI and Embedded Systems-IJVES, Vol 04, Article 06102; June 2013.
[4] Laxman Shanigarapu, Bhavana P. Shrivastava” Low-Power and High Speed Carry Select Adder” International Journal of Scientific and Research Publications, Volume 3, Issue 8, August 2013.