International Journal of Modern Engineering Research (IJMER)
www.ijmer.com Vol. 3, Issue. 5, Sep - Oct. 2013 pp-3230-3141 ISSN: 2249-6645

Design and Implementation of Multiplier Using Kcm and Vedic Mathematics by Using Reversible Adder

V S Kumar Chunduri[1], G.Sree Lakshmi[2], Dr.M.J.C.Prasad[3]
*(PG Student, Dept. of DSCE(ECE), Malla Reddy Engineering College/Autonoumous,India)
** (Assoc.Professor, Dept. of ECE, Malla Reddy Engineering College/Autonoumous,India)
***(professor & Head of the Department, Dept. of ECE, Malla Reddy Engineering College/ Autonoumous,India)

ABSTRACT : *This work is devoted for the design and FPGA implementation of a 16bit Arithmetic module, which uses Vedic Mathematics algorithms. For arithmetic multiplication various Vedic multiplication techniques like Urdhva Tiryakbhyam Nikhilam and Anurupye has been thoroughly analyzed. Also Karatsuba algorithm for multiplication has been discussed. It has been found that Urdhva Tiryakbhyam Sutra is most efficient Sutra (Algorithm), giving minimum delay for multiplication of all types of numbers. Using Urdhva Tiryakbhyam, a 16x16 bit Multiplier has been designed and using this Multiplier, a Multiply Accumulate (MAC) unit has been designed. Then, an Arithmetic module has been designed which employs these Vedic multiplier and MAC units for its operation. Logic verification of these modules has been done by using Model sim 6.5.Further, the whole design of Arithmetic module has been realized on Xilinx Spartan 3E FPGA kit and the output has been displayed on LCD of the kit. The synthesis results show that the computation time for calculating the product of 16x16 bits is 10.148 ns, while for the MAC operation is 11.151 ns. The maximum combinational delay for the Arithmetic module is 15.749 ns. The further extension of this 8 x 8 Array multiplication and Urdhava multiplication can be implemented by using reversible DKG adder replacing with adders(H.A or F.A), and by using 16 x 16 – bit, 32 X 32 – bit are more than that. It can be dumped in to Xilinx tools, and also finding the comparison between the adders like power consumption, speed etc..,*

Keywords: KCM; Urdhava; Vedic Maths; Array Multiplier; DKG Adder; FPGA.

I. INTRODUCTION

Multiplication is one of the more silicon-intensive functions, especially when implemented in Programmable Logic. Multipliers are key components of many high performance systems such asFIR filters, Microprocessors, Digital Signal Processors, etc. A system's performance is generally determined by the performance of the multiplier,because the multiplier is generally the slowest element in the system. Furthermore, it is generally the most area consuming. Hence, optimizing the speed and area of the multiplier is a major design issue.Vedic mathematics [I] is the ancient Indian system of mathematics which mainly deals with Vedic mathematicalformulae and their application to various branches of mathematics. The word 'Vedic' is derived from the word 'Veda' which means the store-house of all knowledge. Vedic mathematics was reconstructed from the ancient Indian scriptures (Vedas) by Sri Bharati Krshna Tirthaji (1884-1960), after his eight years of research on Vedas [1]. According to his research, Vedic mathematics is mainly based on sixteen principles or word-formulae which are termed as Sutras. This is a very interesting field and 978-1-4577-0697-4/12/$26.00 ©2012 IEEE presents some effective algorithms which can be applied tovarious branches of Engineering such as Computing and Digital Signal Processing.

II. VLSI DESIGN

The complexity of VLSI is being designed and used today makes the manual approach to design impractical. Design automation is the order of the day. With the rapid technological developments in the last two decades, the status of VLSI technology is characterized by the following

A steady increase in the size and hence the functionality of the ICs:

• A steady reduction in feature size and hence increase in the speed of operation as well as gate or transistor density.

• A steady improvement in the predictability of circuit behavior.

• A steady increase in the variety and size of software tools for VLSI design.

The above developments have resulted in a proliferation of approaches to VLSI design.

Final step in the development process, starting in the 1980s and continuing through the present, was in the early 1980s, and continues beyond several billion transistors as of 2009. In 1986 the first one megabit RAM chips were introduced, which contained more than one million transistors. Microprocessor chips passed the million transistor mark in 1989 and the billion transistor mark in 2005.The trend continues largely unabated, with chips introduced in 2007 containing tens of billions of memory transistors. The complexity of VLSIs being designed and used today makes the manual approach to design impractical. Design automation is the order of the day. With the rapid technological developments in the last two decades, the status of VLSItechnology is characterized by the following [Wai-kai, Gopalan]:

•A steady increase in the size and hence the functionality of the ICs.

•A steady reduction in feature size and hence increase in the speed of operation as well as gate or transistor density.

•A steady improvement in the predictability of circuit behavior.

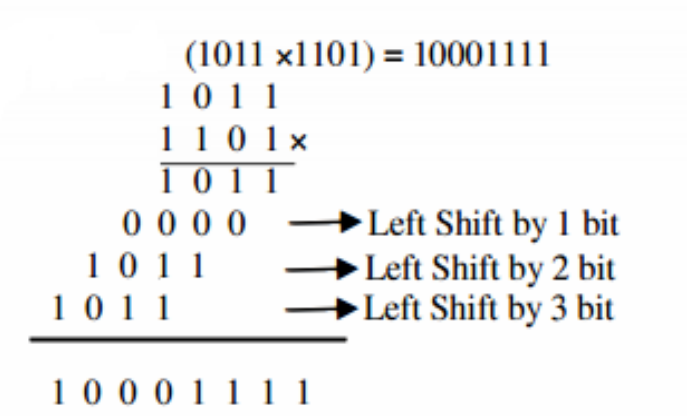
•A steady increase in the variety and size of software tools for VLSI design. The above developments have resulted in a proliferation of approaches to VLSIdesign. We briefly describe the procedure of automated design flow [Rabaey, Smith MJ]. The aim is more to bring out the role of a Hardware Description Language (HDL) in the design process. An abstraction

based model is the basis of the automated design. The model divides the whole design cycle into various domains. With such an abstraction through a division process the design is carried out indifferent layers. The designer at one layer can function without bothering about the layers above or below. The thick horizontal lines separating the layers in the figure signify the compartmentalization. As an example, let us consider design at the gate level. The circuit to be designed would be described in terms of truth tables and state tables. With these as available inputs, he has to express them as Boolean logic equations and realize them in terms of gates and flip-flops. In turn, these form the inputs to the layer immediately below.

III. ARRAY MULTIPLIER

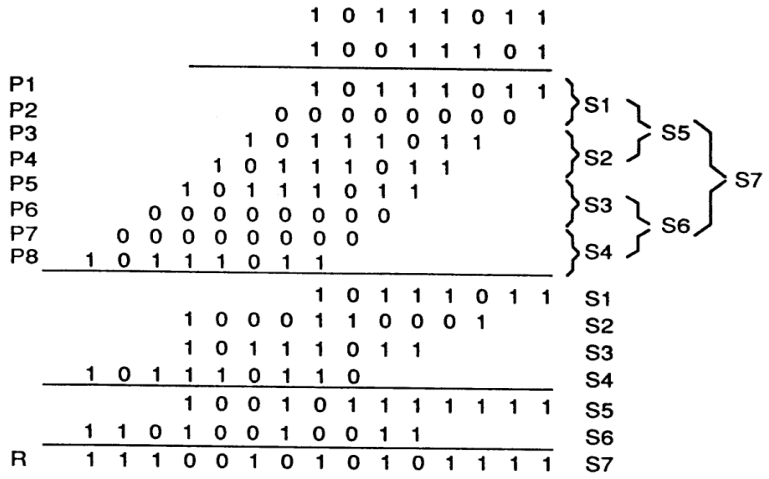
In Array multiplier , AND gates are used for generation of the bit-products and adders for accumulation of generated bit products. All bit-products are generated in parallel and collected through an array of full adders or any other type of adders. Since the array multiplier is having a regular structure, wiring and the layout are done in a much simplified manner. Therefore, among other multiplier structures, array multiplier takes up the least amount of area. But it is also the slowest with the latency proportional to O(Wct), where Wd is the word length of the operand.

Example 1:



Example1 for Array multiplier 4*4

Example 2:



Example2 for Array multiplier 8*8

Instead of Ripple Carry Adder (RCA), here Carry Save Adder (CSA) is used for adding each group of partial product terms, because RCA is the slowest adder among all other types of adders available. In case of multiplier with CSA , partial product addition is carried out in Carry save form and RCA is used only in final addition. Here from the above example it is inferred that partial products are generated sequentially, which reduces the speed of the multiplier. However the structure of the multiplier is regular.

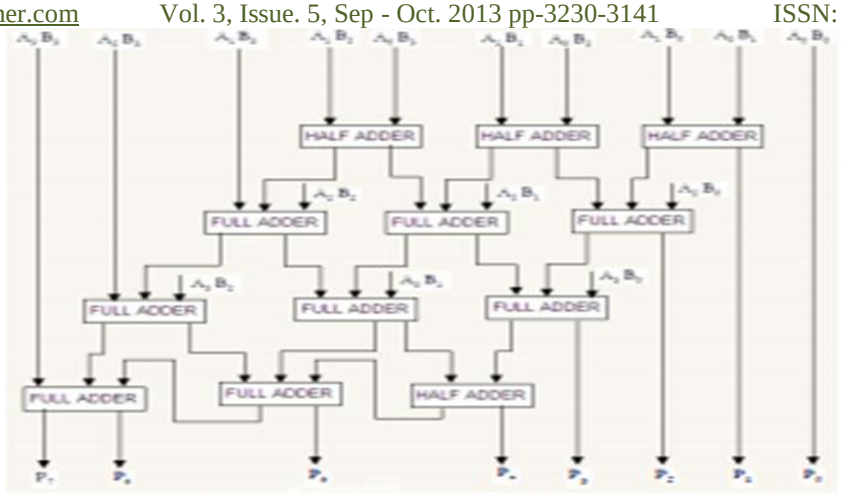


Fig : 1 Array Multiplier 4 * 4 using CSA Hardware Architecture.

In this method, for the first 3 numbers a row of full adder are used. Then a row of full adder is added for each additional number. The final results, in the form of two numbers sum and carry, are then summed up with a carry propagate adder or any other adder. An example 4 numbers addition is shown in Fig 1.There are many cases where it is desired to add more than two numbers together. The straight forward way of adding together m numbers (all n bits wide) is to add the first two, then add that sum to the next, and so on. This requires a total of m − 1 additions, for a total gate delay of (assuming look ahead carry adders). Instead, a tree of adders can be formed, taking only gate delays. Using carry save addition, the delay can be reduced further still. The idea is to take 3 numbers that we want to add together, x + y + z, and convert it into 2 numbers c + s such that x + y + z = c + s, and do this in time. The reason why addition cannot be performed in time is because the carry information must be propagated. In carry save addition, we refrain from directly passing on the carry information until the very last step. We will first illustrate the general concept with a base 10 example. To add three numbers by hand, we typically align the three operands, and then proceed column by column in the same fashion that we perform addition with two numbers. The three digits in a row are added, and any overflow goes into the next column. Observe that when there is some non-zero carry, we are really adding four digits (the digits of x ,y and z, plus the carry).In many cases we need to add several operands together, carry save adder are ideal for this type of addition. A carry save adder consists of stand-alone full adders, and carries out a number of partial additions. The principal idea is that the carry has a higher power of 2 and thus is routed to the next column. Doing addition with carry save adder saves time and logic.In this method, for the first 7 numbers a row of full adder are used. Then a row of full adder is added for each additional number. The final results, in the form of two numbers sum and carry, are then summed up with a carry propagate adder or any other adder.

IV. URDHAVA MULTIPLIER

In Urdhava Tiryakbhyam is a Sanskrit word which means vertically and crosswire in English. The method is a general multiplication formula applicable to all cases of multiplication. It is based on a novel concept through which all partial products are generated concurrently. Fig. Demonstrates a 4 x 4 binary multiplication using this method. The method can be generalized for any N x N bit multiplication. This type of multiplier is independent of the clock frequency of the processor because the partial products and their sums are calculated in parallel.The net advantage is that it reduces the need of microprocessors to operate at increasingly higher clock frequencies. As the operating frequency of a processor increases the number of switching instances also increases. This results more power consumption and also dissipation in the form of heat which results in higher device operating temperatures. Another advantage of Urdhava Tiryakbhyam multiplier is its scalability T.

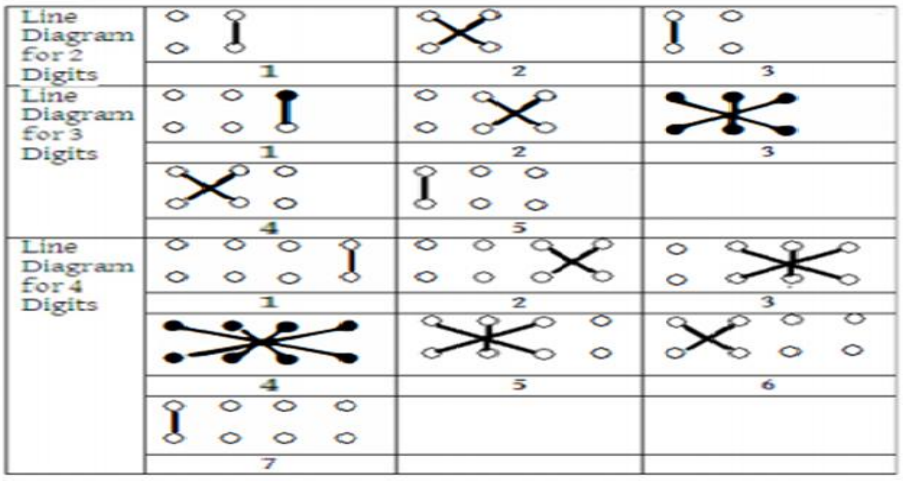


Fig: 2 Line Diagram for Urdhava Multiplication.

The processing power can easily be increased by increasing the input and output data bus widths since it has a regular structure. Due to its regular structure, it can be easily layout in a silicon chip and also consumes optimum area. As the number of input bits increase, gate delay and area increase very slowly as compared to other multipliers. Therefore Urdhava Tiryakbhyam multiplier is time, space and power efficient.

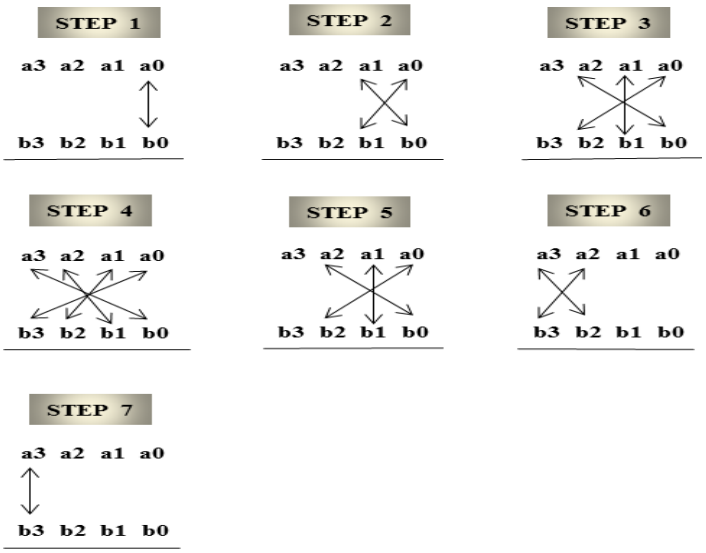
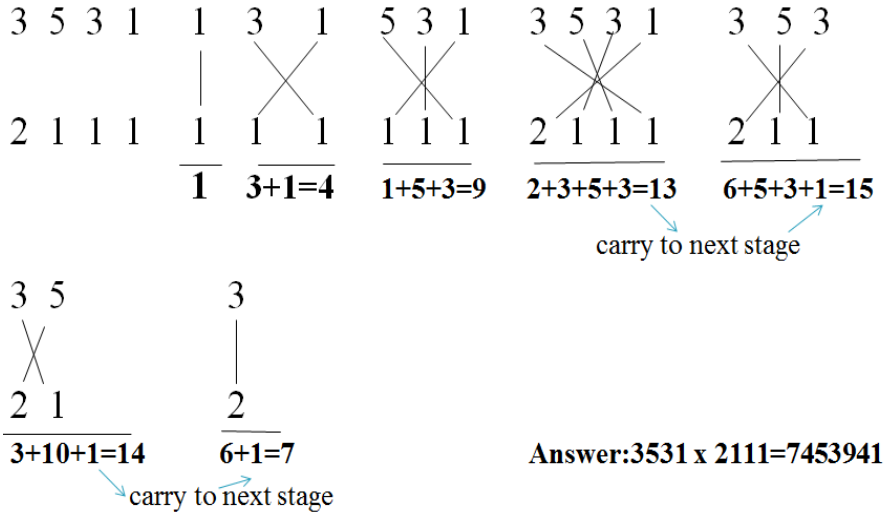


Fig : 3 Multiplication of two 4 bit numbers using Urdhava Tiryakbhyam method

Example 3:



Example3 for the Multiplication of two 4 bit numbers using Urdhava Tiryakbhyam method

The line diagram in fig. 3 illustrates the algorithm for multiplying two 4-bit binary numbers a3, a2, a1, a0 and b3, b2, b1, b0. The procedure is divided into 7 steps and each step generates partial products. Initially as shown in step 1 of fig. 2, the least significant bit (LSB) of the multiplier is multiplied with least significant bit of the multiplicand (vertical multiplication). This result forms the LSB of the product. In step 2 next higher bit of the multiplier is multiplied with the LSB of the multiplicand and the LSB of the multiplier is multiplied with the next higher bit of the multiplicand (crosswire multiplication). These two partial products are added and the LSB of the sum is the next higher bit of the final product and the remaining bits are carried to the next step. For example, if in some intermediate step, we get the result as 1101, then 1 will act as the result bit(referred as rn) and 110 as the carry (referred as cn). Therefore cn may be a multi-bit number. Similarly other steps are carried out as indicated by the line diagram. The important feature is that all the partial products and their sums for every step can be calculated in parallel. Thus every step in fig. 3.1 has a corresponding expression as follows:

r0=a0b0. (1)
c1r1=a1b0+a0b1. (2)
c2r2=c1+a2b0+a1b1 + a0b2. (3)
c3r3=c2+a3b0+a2b1 + a1b2 + a0b3. (4)
c4r4=c3+a3b1+a2b2 + a1b3. (5)
c5r5=c4+a3b2+a2b3. (6)
c6r6=c5+a3b3 (7)

With c6r6r5r4r3r2r1r0 being the final product. Hence this is the general mathematical formula applicable to all cases of multiplication and its hardware architecture is shown in fig. 3. In order to multiply two 8-bit numbers using 4-bit multiplier we proceed as follows.

Consider two 8 bit numbers denoted as AHAL and BHBL where AH and BH corresponds to the most significant 4 bits, AL and BL are the least significant 4 bits of an 8-bit number. When the numbers are multiplied multiplied according to Urdhava Tiryakbhyam (vertically and crosswire) method, we get,

AH AL
BH BL

(AH x BH) + (AH x BL + BH x AL) + (AL x BL).

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.

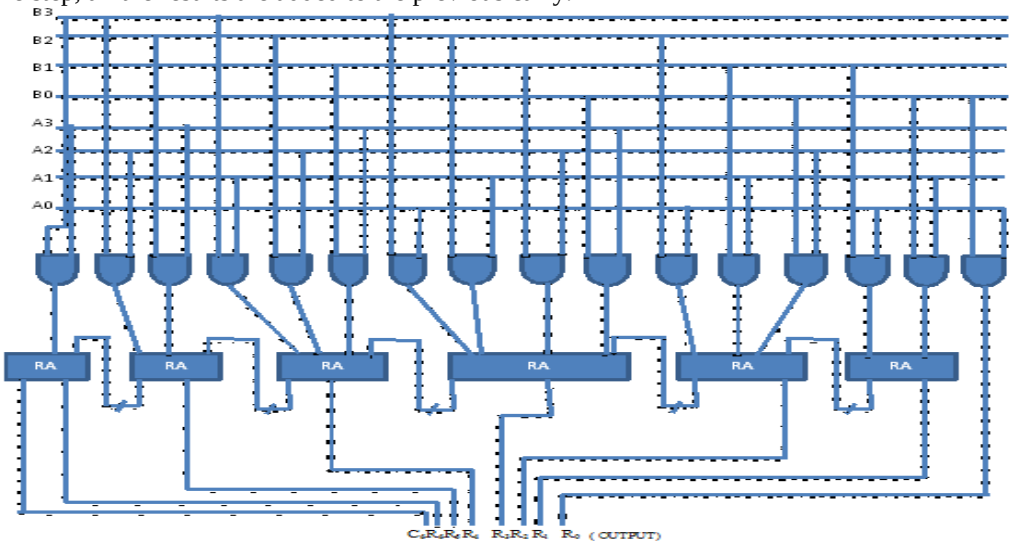


Fig: 4 Hardware architecture of 4 X 4 Urdhava Tiryakbhyam multiplier using reversible DKG added.

Thus we need four 4-bit multipliers and two adders to add the partial products and 4-bit intermediate carry generated. Since product of a 4 x 4 multiplier is 8 bits long, in every step the least significant 4 bits correspond to the product and the remaining 4 bits are carried to the next step. This process continues for 3 steps in this case. Similarly, 16 bit multiplier has four 8 x 8 multiplier and two 16 bit adders with 8 bit carry. Therefore we see that the multiplier is highly modular in nature. Hence it leads to regularity and scalability of the multiplier layout. The multiplier architecture is based on this Urdhava tiryakbhyam sutra. The advantage of this algorithm is that partial products and their sums are calculated in parallel. This parallelism makes the multiplier clock independent. The other main advantage of this multiplier as compared to other multipliers is its regularity. Due to this modular nature the lay out design will be easy. The architecture can be explained with two eight bit numbers i.e. the multiplier and multiplicand are eight bit numbers. The multiplicand and the multiplier are split into four bit blocks. The four bit blocks are again divided into two bit multiplier blocks. According to the algorithm the 8 x 8 (AH x BH) bit multiplication will be as follows.

AH = AHH - AHL, BH = BHH - BHL
AH=AH7AH6AH5AH4AH3AH2AH1AH0,
BH = BH7BH6BH5BH4BH3BH2BH1BH0,
AHH = AH7AH6AH5AH4,
AHL = AH3AH2AH1AH0
BHH = BH7BH6BH5BH4, BHL = BH3BH2BH1BH0

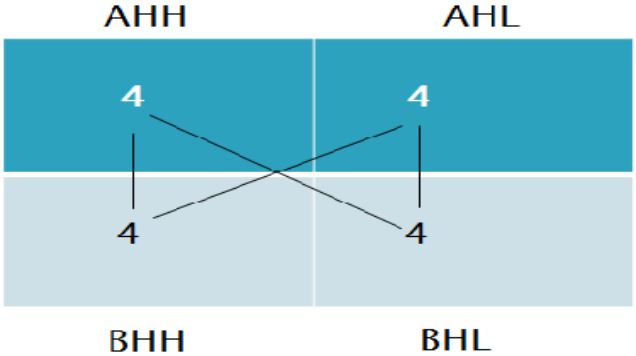


Fig : 5 Multiplication of two 8 bit numbers using Urdhava Tiryakbhyam method

By the algorithm, the product can be obtained as follows.
Product of AH x BH = AHL x BHL + (AHH x BHL + AHL x BHH) + AHH x BHH
Thus 8 x 8 multiplications can be decomposed into 2 x 2 multiplication units. By using this algorithm any complex N x N multiplication can be implemented using the basic 2 x 2 multiplier units.

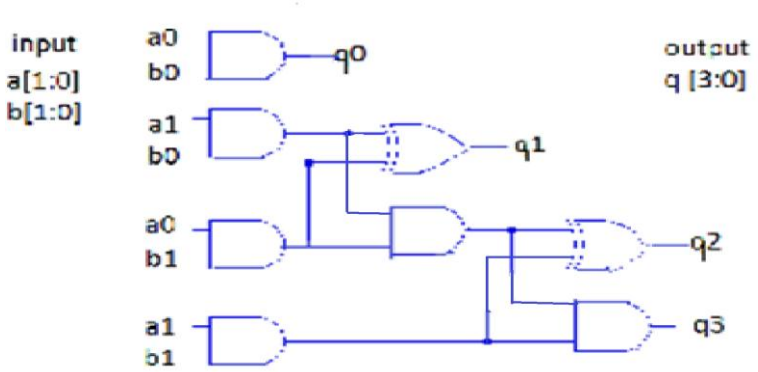


Fig: 6 Hardware Realization of 2x2 block

Hear a0=AL, a1=AH;
b0=BL, b1=BH;

For Multiplier, first the basic blocks, that are the 2x2 bit multipliers have been made and then, using these blocks, 4x4 block has been made and then using this 4x4 block, 8x8 bit block, 16x16 bit block. Urdhava Tiryakbhyam 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 Fig.

8 X 8 Bit Multiplication Using Urdhava Triyakbhyam (Vertically and crosswise) for two Binary numbers

Consider two binary numbers A and B of 8 bits as respectively

$$A = \underbrace{A_7A_6A_5A_4}_{(X_1)}\ \underbrace{A_3A_2A_1A_0}_{(X_0)}$$

$$B = \underbrace{B_7B_6B_5B_4}_{(Y_1)}\ \underbrace{B_3B_2B_1B_0}_{(Y_0)}$$

Which can be viewed as two four bit numbers each, i.e. A can be viewed as X_1 X_0 and B can be viewed as Y_1 Y_0 respectively, as shown above, thus the multiplication can be written as

$$\begin{array}{r} X_1\ X_0 \\ *\ Y_1\ Y_0 \\ \hline EDC \end{array}$$

Where, CP= C = X_0Y_0
CP= A = X_1Y_0
CP = B = X_0Y_1
CP= D = A+B
CP= E = X_1Y_1 here CP= Cross Product

Thus, A*B= EDC, is achieved using Urdhava Triyakbhyam (Vertically and crosswise) sutra.

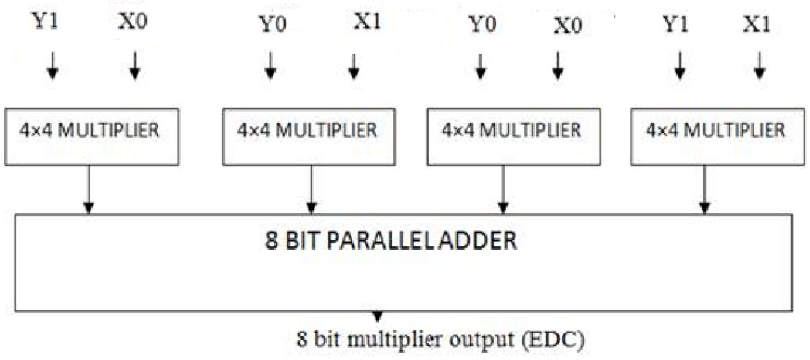


Fig :7 Hardware architecture of 8 X 8 Urdhava Tiryakbhyam multiplier.

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 A7A6A5A4A3A2A1A0 and B7B6B5B4B3B2B1B0. As the result of this multiplication would be more than 8 bits, we express it as ...R7R6R5R4R3R2R1R0. 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 lines 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 all the other bits act as carry. For example, if in some intermediate step we will get 011, then I will act as result bit and 01 as the carry.

V. REVERSIBLE LOGIC GATES

There exist many reversible gates in the literature. Among them 2*2 Feynman gate , 3*3 Fredkin gate, 3*3 Toffoli and 3*3 Peres is the most referred. The detailed cost of a reversible gate depends on any particular realization of quantum logic. Generally, the cost is calculated as a total sum of 2*2 quantum primitives used. The cost of Toffoli gate is exactly the same as the cost of Fredkin gate and is 5. The only cheapest quantum realization of a complete (universal) 3*3 reversible gate is Peres gate and its cost is 4.

Controlled NOT (CNOT) gate is an example for a 2*2 gate. The Reversible 2*2 gate with Quantum Cost of one having mapping input (A, B) to output (P = A, Q = A B)

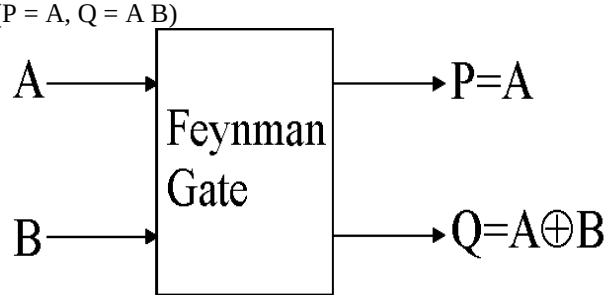


Figure 8: 2*2 Feynman gate

Reversible 3*3 gate maps inputs (A, B, C) to outputs (P=A, Q=A'B+AC, R=AB+A'C) having Quantum cost of 5 and it requires two dotted rectangles, is equivalent to a 2*2 Feynman gate with Quantum cost of each dotted rectangle is 1, 1 V and 2 CNOT gates.

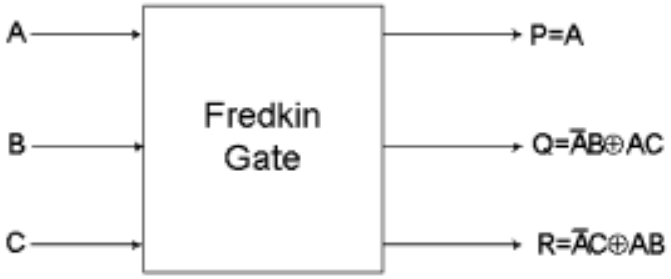


Figure 9: 3*3 Fredkin gate

The 3*3 Reversible gate with three inputs and three outputs. The inputs (A, B, C) mapped to the outputs (P=A, Q=B, R=A.BC)

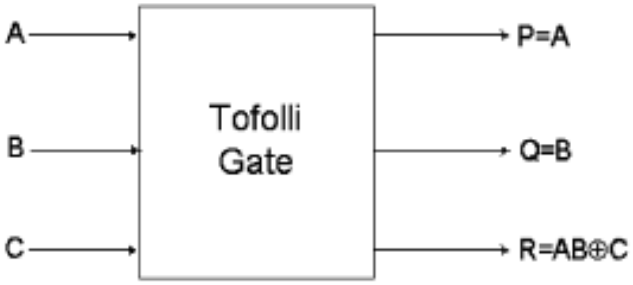


Figure 10: 3*3 Toffoli gate

The three inputs and three outputs i.e., 3*3 reversible gate having inputs (A, B, C) mapping to outputs (P = A, Q = A B, R = (A.B) C). Since it requires 2 V+, 1 V and 1 CNOT gate, it has the Quantum cost of 4.

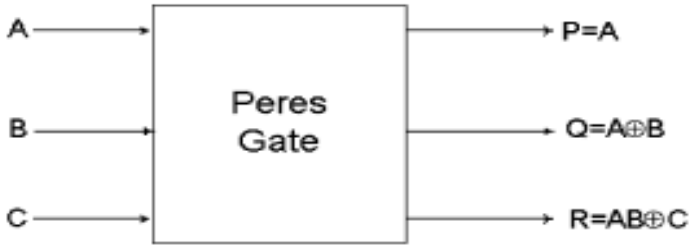


Figure 11: 3*3 Peres gate

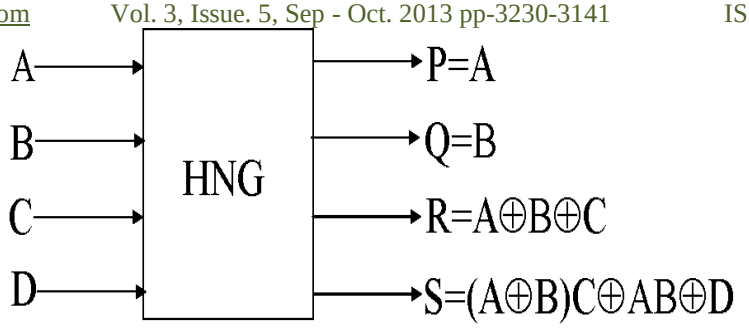


Figure 12: 4*4 HNG gate

Reversible DKG Gate:

Reversible DKG gate has 4 inputs and 4 outputs, so it is called Reversible 4*4 DKG gate, A 4* 4 reversible DKG gate that can work singly as a reversible Full adder and a reversible Full subtractor is shown in Fig . It can be verified that input pattern corresponding to a particular output pattern can be uniquely determined. If input A=0, the proposed gate works as a reversible Full adder, and if input A=1, then it works as a reversible Full subtractor. It has been proved that a reversible full-adder circuit requires at least two garbage outputs to make the output combinations unique figures.

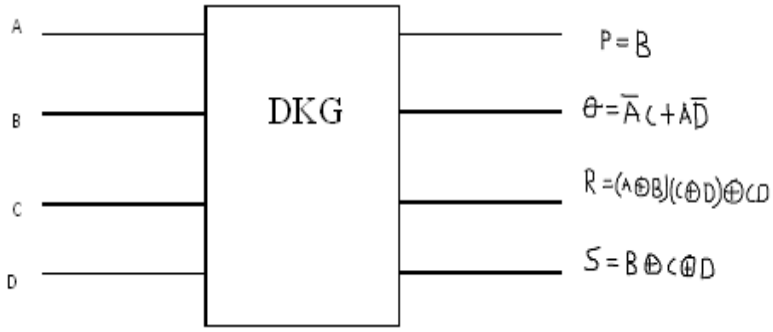


Figure 13: Reversible DKG gate

DKG gate with inputs A, B, C, D and outputs are P, Q, R, S. This gate is known as DKG gate. Figure 8 shows the DKG gate with 4*4 inputs and outputs. The binary Full adder/subtractor is capable of handling one bit of each input along with a carry in/borrow in generated as a carry out/ borrow from addition of previous lower order bit position. If two binary numbers each consisting of n bits are to be added or subtracted, then n binary full adders/subtractors are to be cascaded.

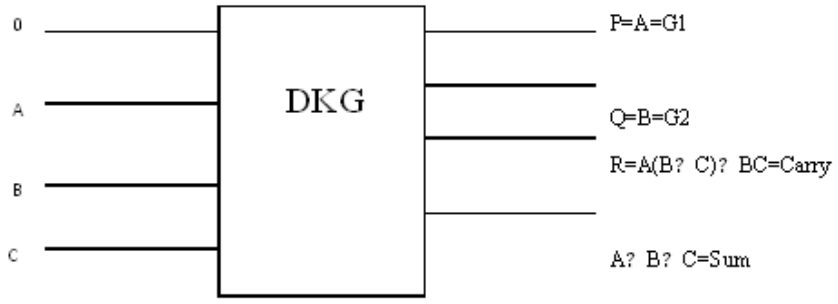


Figure 14: DKG gate implemented as Full adder

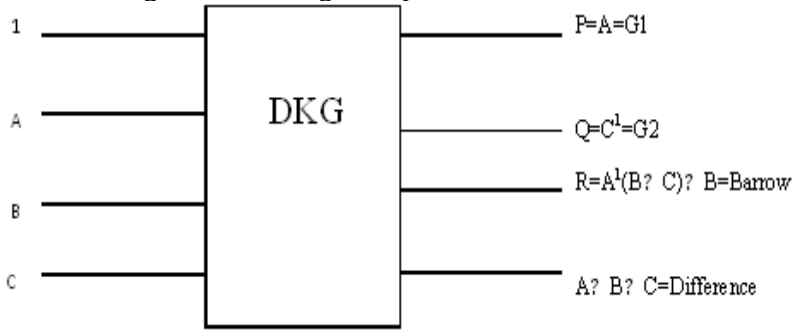


Figure 15: DKG gate implemented as Full subtractor

The binary Full adder/subtractor is capable of handling one bit of each input along with a carry in/borrow in generated as a carry out/ borrow from addition of previous lower order bit position. If two binary numbers each consisting of n bits are to be added or subtracted, then n binary full adders/subtractors are to be cascaded. A Parallel adder/subtractor is an interconnection of full adders/subtractors and inputs are simultaneously applied. The carry/borrow generated at a stage is

propagated to the next stage. Thus, delay is more in such type of adders/subtractors. A 4 bit reversible parallel adder/subtractor is implemented using the reversible DKG gate and shown in Fig 10a. When the control input A=0, the circuit acts as a parallel adder, produces a 4 bit sum and a carry out, as shown in Fig 10b. If the control input A=1, the circuit acts as a parallel subtractor, produces a 4 bit difference and borrow out, as shown in Fig. The same design can be extended to n bits.

VI. PROPOSED MULTIPLIER

The proposed method is based on ROM approach however both the inputs for the multiplier can be variables. In this proposed method a ROM is used for storing the squares of numbers as compared to KCM where the multiples are stored.

operation:

To find (a x b), first we have to find whether the difference between 'a' and 'b' is odd or even. Based on the difference, the product is calculated.

In case of Even Difference

Result of Multiplication= [Average]^2- [Deviation]^ 2

In case of Odd Difference

Result of Multiplication = [Average x (Average + 1)] -[Deviation x (Deviation+ I)]

Where

Average = [(a+b)/2]

Deviation = [Average -smallest(a, b)]

Example 4 (Even difference) and Example 5 (Odd difference) depict the multiplication process. Thus the two variable multiplication is performed by averaging, squaring and subtraction. To find the average[(a+b)/2], which involves division by 2 is performed by right shifting the sum by one bit.

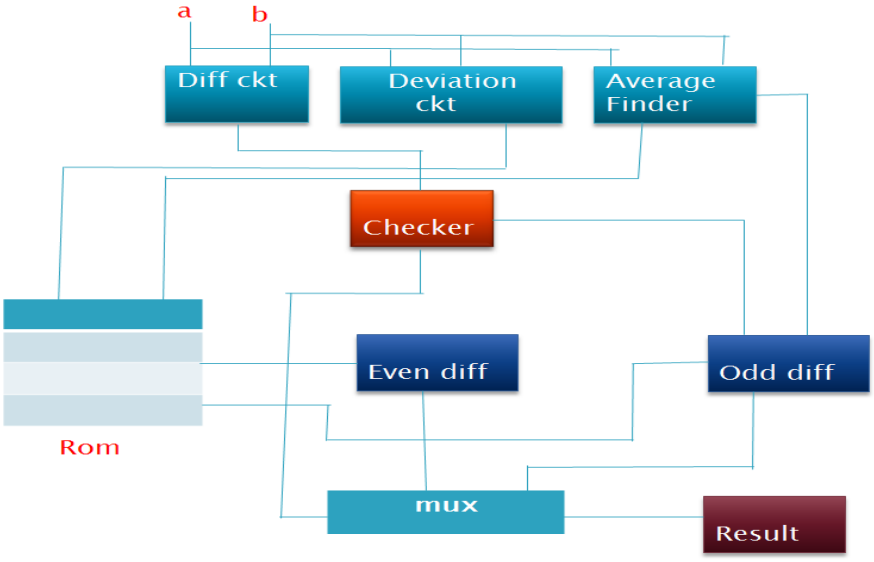


Fig : 16 Block diagram for proposed multiplier.

If the squares of the numbers are stored in a ROM, the result can be instantaneously calculated. However, in case of Odd difference, the process is different as the average is a floating point number. In order to handle floating point arithmetic, Ekadikena Purvena - the Vedic Sutra which is used to find the square of numbers end with 5 is applied. Example 4 illustrates this. In this case, instead of squaring the average and deviation, [Average x (Average + 1)] - [Deviation x (Deviation+ I)] is used. However, instead of performing the multiplications, the same ROM is used and using equation the result of multiplication is obtained.

n(n+l) = (n2+n) ... (10)

Here n2 is obtained from the ROM and is added with the address which is equal to n(n+l)

Example 4:

16 x 12 = 192

1) Find the difference between (16-12) = 4 ----Even Number

2) For Even Difference, Product = [Average]^2- [Deviation]^2

i. Average = [(a+b)/2] = [(16+12)12] = [28/2] = 14

ii. Smallest(a,b) = smallest(l6,12) =12

iii. Deviation = Average - Smallest (a,b) = 14 -12 =2

3) Product = $14^2 - 2^2 = 196 - 4 = 192$.

Example 5:
15 x 12 = 180
I) Find the difference between (15-12)=3 -7 Odd Number
2) For Odd Number Difference find the Average and
Deviation.
i.Average = [(a+b)/2] = [(12+15)/2] = 13.5
ii.Deviation = [Average - smallest(a, b)] =
[12.5 - smallest(l3,12)] = [13.5 - 12] = 1.5
3)Product = (l3xI4) - (lx2) = 182 - 2 =180.

VII. SIMULATION RESULTS AND TABLES

Fig : 17 Speed Comparison for (8x8)

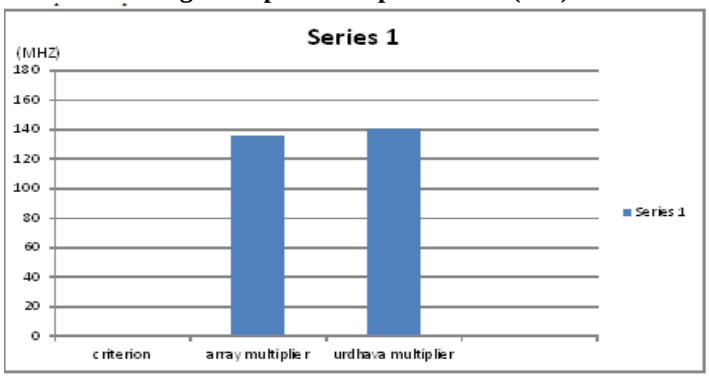


Fig : 18 speed comparison for DKG (8*8)

Simulation result

The comparison is carried out in between the reversible and conventional logic gates by using XILINX 9.1and program is written in VERILOG language. . In reversible logic we use DKG and TSG gates for both adder/subtractor as it has low power consumption and less garbage output as already discussed in the section3. The comparison is carried out for the four operand four bit adder/subtractor in reversible and conventional gates.

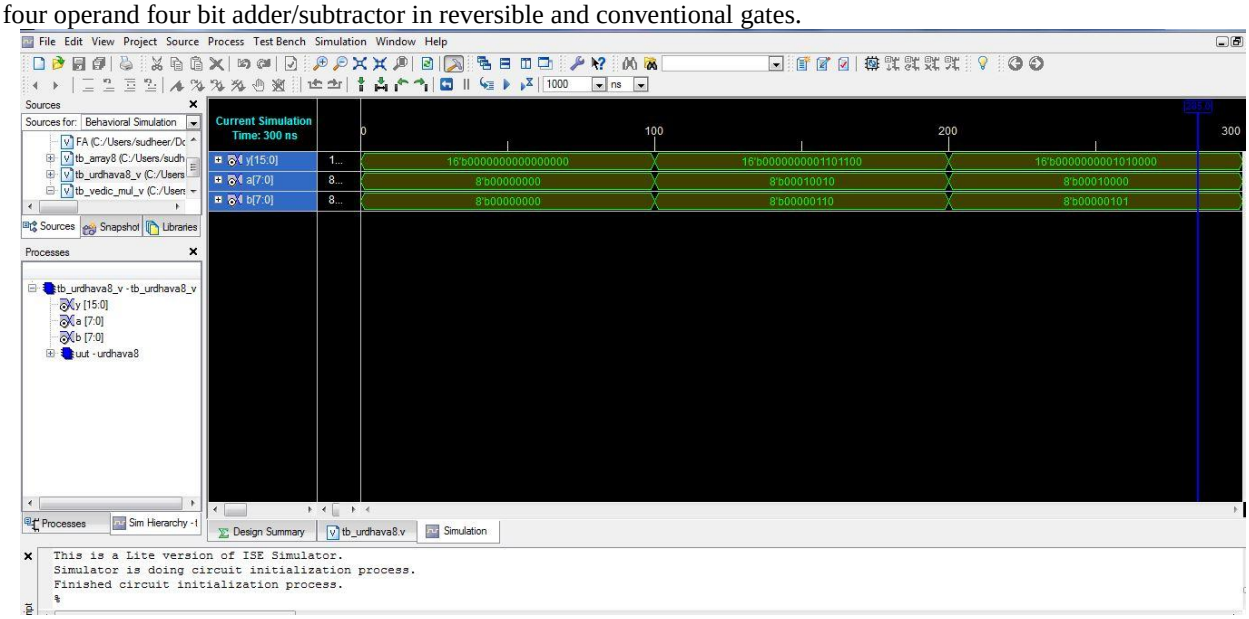


Fig 19 : 8 bit Array multiplier using DKG

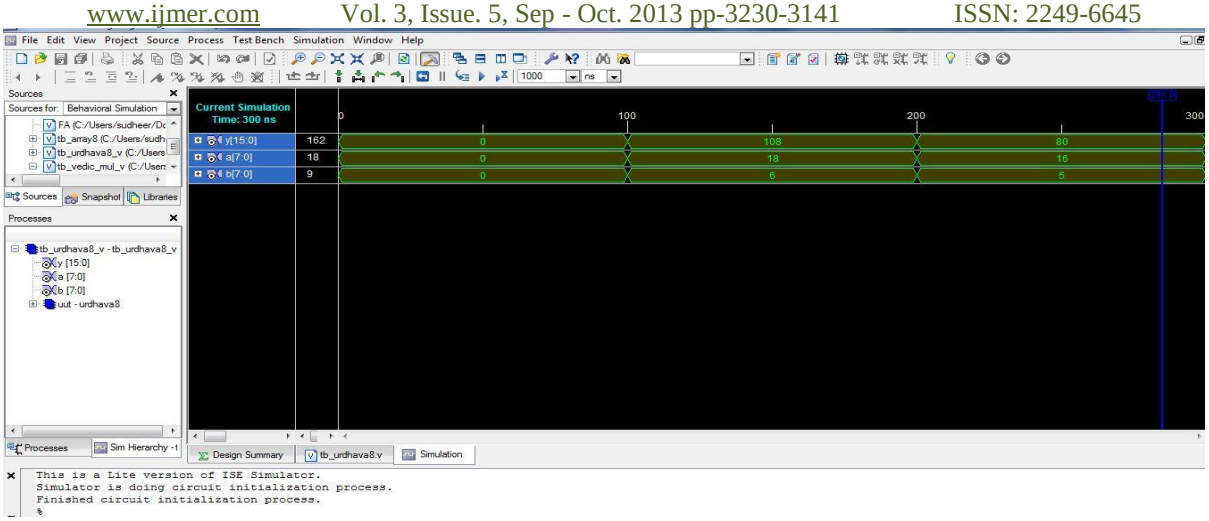


Fig 20: 8 bit Urdhava multiplier using DKG

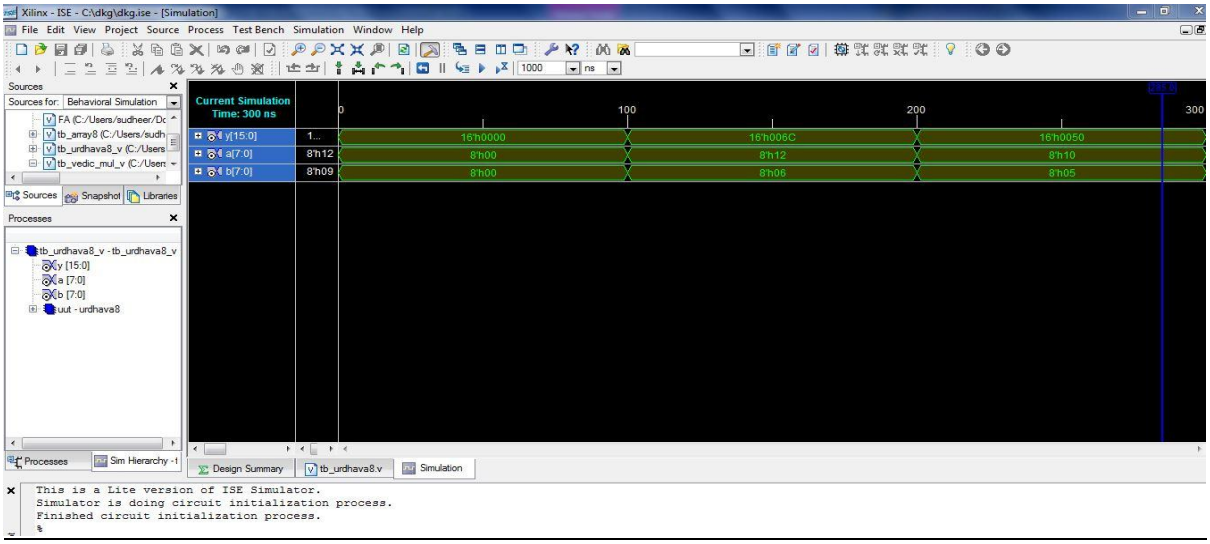


Fig 21: 8 bit proposed multiplier

Criterion	Array Multiplier	Urdhava Multiplier
Area	126	180
Total Combinational Functions	163	149
Dedicated Logic Registers	48	48
Total Memory Bits(Kb)	0	0
Transitions	1557	1501
Speed(After Pipelining)(MHz)	137.46	142.67
Power	100	90
temperature	27c	27c

Table1: results for 8x8 multiplier

Criterion	Array Multiplier	Urdhava Multiplier
Area	126	180
Total Combinational Functions	158	146
Dedicated Logic Registers	46	46
Total Memory Bits(Kb)	0	0
Transitions	1551	1547
Speed(After Pipelining)(MHz)	139.35	145.03
Total Power	90	82
temperature	27c	27c

Table 2: Results For Dkg 8*8 multiplier

VIII. CONCLUSION

Thus the proposed multiplier provides higher performance for higher order bit multiplication. In the proposed multiplier for higher order bit multiplication i.e. for 16x16 and more, the multiplier is realized by instantiating the lower order bit multipliers like 8x8. This is mainly due to memory constraints. Effective memory implementation and deployment of memory compression algorithms can yield even better results.

ACKNOWLEDGMENTS

I am very grateful to my project guide **Mrs. G.SREE LAKSHMI, Associate prof** of Electronic & Communication Engineering, Malla Reddy Engineering College, for her extensive patience and guidance throughout my project work. Also having provided the freedom to use all the facilities available in the department, especially the laboratories and the library.

I am happy to express my since thanks to Mrs. P.S. INDRANI, Associate prof, PG coordinator of Electronic & Communication Engineering department.

I would like to thank Dr. M. Jagdissh Chandra Prasad, Professor & HOD, Department of Electronics and communication engineering, Malla Reddy Engineering College, for their timely suggestions, healthy criticism and motivation during the course of this work.

I am happy to express my deep sense of gratitude to the principal of the college **Dr. S. Sudhakara Reddy,** professor for having provided us with facilities to pursue our project.

I sincerely thank to all teaching and non- teaching staff of the department of Electronics & Communication Engineering for their timely suggestions, healthy criticism and motivation during the course of this work.

Finally, I express my immense gratitude with pleasure to other individual who having directly or indirectly contributed to my need at right time for the development and success if this project work.

REFERENCE

[1.] Bharati Krshna Tirthaji, Vedic Mathematics. Delhi: Motilal Banarsidass Publishers, 1965.

[2.] Harpreet Singh Dhillon and Abhijit Mitra "A Digital MultiplierArchitecture using Urdhava Tiryakbhyam Sutra oj Vedic Mathematics" IEEE Conference Proceedings,2008.

[3.] Asmita Haveliya "A Novel Design ./i)r High Speed Multiplier .fi)r Digital Signal Processing Applications (Ancient Indian Vedic mathematics approach)" International Journal of Technology AndEngineering System(IJTES):Jan - March 2011- Vo12 .Nol

[4.] P. D. Chidgupkar and M. T. Karad, "The Implementation oj Vedic Algorithms in Digital Signal Processing", Global J. oj /c'ngg. /c'du., vol. 8, no.2, pp. 153-158, 2004.

[5.] J.Bhasker, "Verilog HDL Primer" BS P Publishers, 2003.

[6.] M. Ramalatha, K. Deena Dayalan,P. Dharani, S. Deborah Priya,"High Speed Jc'nergy ElJzcient AL U Design using Vedic Multiplication Techniques", ACTEA 2009, Zouk Mosbeh, Lebanon.

[7.] Landauer, R., 1961. Irreversibility and heat generation in the computing process, IBM J. Research and Development, 5 (3): 183-191.

[8.] Thaplyal, H. and M.B. Srinivas, 2006. Novel Reversible Multiplier Architecture Using Reversible TSG gate. IEEE international Conference on Computer Systems and Applications, pp: 100-103.

[9.] Shams, M., M. Haghparast and K. Navi, 2008. Novel Reversible Multiplier Circuit in Nanotechnology. World Appl. Sci. J., 3 (5): 806- 810.

[10.] D. MASLOV, G. W. DUECK, AND D. M. MILLER, Synthesis of Fredkin-Toffoli Reversible Networks, *IEEE Trans. VLSI Systems*, **13(6)**, pp. 765-769, 2005.

[11.] C.H. Bennett , "Logical Reversibility of Computation", IBM J.Research and Development, pp. 525-532, November 1973.

[12.] Himanshu Thapliyal, M.B Srinivas and Hamid R. Arabnia, " A Reversible Version of 4 x 4 Bit Array Multiplier With Minimum Gates and Garbage Outputs ", The 2005 International Conference on Embedded System and Applications(ESA'05), Las Vegas, U.S.A, June 2005,pp-106-114.

[13.] Saiful Islam and Rafiqul Islam., 2005. Minimization of Reversible Adder Circuits. Asian Journal of Information Technology 4 (12): 1146-1151.

[14.] Fredkin, E. and Toffoli, T. (1982). Conservative logic. Int. Journal of Theoretical Physics, 21, 219–253.

[15.] Md. Saiful Islam, Md. Rafiqul Islam, Muhammad Rezaul Karim, Abdullah Al Mahmud and Hafiz Md. Hasan Babu, "Minimization of Adder Circuits and Variable Block Carry Skip Logic using Reversible Gates", In Proc. of 7th International Conference on Computer and Information Technology, ICCIT 2004, Brac University, Dhaka, Bangladesh, 26-28 December, 2004, pp. 378- 383.