# Vedic Multiplier Using Nikhilam Navatascaramam Dasatah Sutra

## ModernVLSI High Speed Multiplier

K.BHARATHA BABU[1], REEBA KORAH[2] , A. SWAMALATHA[3]

[1]Department of ECE, Research Scholar, Anna University, Chennai, Tamil Nadu, India,
[2]Alliance College of Engineering and Design, Alliance University, Bangalore-562106, Karnataka, India
[3]St. Joseph's College of Engineering, Chennai-600119, Tamil Nadu, India
[1]kbharathababu@gmail.com

***Abstract*—In recent years, digital devices require arithmetic units, particularly multipliers at high speed and efficiency due to the growing complexity in performing arithmetic operations. Conventional multipliers are unstable when operating on large amount of data leading to errors and time delay. Vedic mathematics can be used to satisfy these demands. The proposed algorithm aims at improving the speed and reducing the power consumption of the Vedic multiplier by modifying the Nikhilam NavatascaramamDasatah Sutra based multiplier. The Vedic multiplier is suitable for multiplying real and floating point numbers. The algorithm is simulated using Xilinx ISE and implemented in Virtex-V. The results when compared to the conventional multipliers show a 15% improvement in the speed and 10% improvement in power reduction and 5 % improvement in area for 4, 8 and 16 bit inputs.**

***Keywords—Vedic mathematics, Vedic Multiplier, Nikhilam Navatascaramam Dasatah.***

## I. INTRODUCTION

The growth in digital communication devices and application has increased the demand to process large volume of data at high speed. Multiplier plays an important role in making the digital devices efficient. Multipliers are expensive and they tend to slow down while processing vast amount of data. Speed is an important constraint during multiplication. Higher speed can be achieved by reducing the steps in the computational process. To improve the efficiency, Vedic Mathematics is employed in the design process. Vedic mathematics is an ancient system for mathematical operations that was initially obtained from Ancient Indian Sculptures and later presented as a book. It comprises of the research work carried out by Swamiji Sri BharathiKrisnaThirthaji Maharaja and Jagadguru Sri Sankaracharya during AD 1885-1960. They formulated 16 principles for Vedic mathematics and were termed as Sutras. These Sutras comprised of many algorithms that are interesting and applicable for many fields of engineering such as digital signal processing. Thus, designing a multiplier using Vedic Mathematics will improve the speed and efficiency of the multiplication operation.

The paper [1] provides a compressor based Vedic multiplier architecture to generate the partial product. A 2bit, 4bit and 8bit multiplier was proposed in [2] a multiplier constructed using sub multipliers to reduce the propagation delay. Area is reduced in [3] utilizing a ripple carry adder in designing the Vedic multiplier. The complexity in [3] is reduced in [4] by employing carry save adder and carry select adder that is better than ripple carry based Vedic multiplier. Kogestone adder being the fastest parallel prefix adder as suggested in [6] to reduce delay.

A modified Vedic multiplication algorithm proposed in [7] is comparatively better than Wallace tree multiplier in terms of delay. In [8], two main multiplication sutras UrdhvaTiryagbhyam and NikhilamNavatascaramamDasatah Sutra are proposed and compared to Array multiplier and Booth multiplier in terms of combinational path delay.

In this paper, an N-bit Vedic multiplier is proposed which is based on Nikhilam Sutra. The proposed 4 bit, 8 bit and 16 bit Vedic multiplier is compared with that of Urdhva and Nikhilam Vedic multipliers. The proposed Vedic multiplier has lesser delay than other multipliers.

## II. VEDIC MULTILIER

### *A. Nikhilam Navatascaramam Dasatah*

NikhilamNavatascaramamDasatah Sutra utilizes its nearest base (a power of 10) to perform multiplication operation. Hence, the complexity of the operation is reduced irrespective of the size of the numbers to be multiplied. The multiplier and multiplicand have their respective common nearest base as a power of 10 [1].

The two numbers are written in two rows along with their differences termed as Deviation as shown in Fig.1. Column-1 represents the numbers that are to be multiplied and the column-2 represents the deviation from their bases. By cross-adding the values in the columns, we obtain the respective left hand side value as 106. To obtain the right hand side value as 08, we multiply the deviations together as d1*d2.

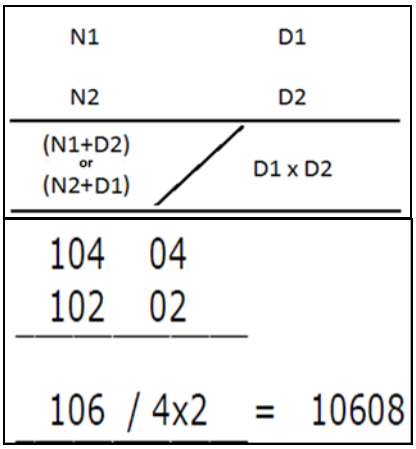



Fig.1 Existing NikhilamNavatascaramamDasatah multiplication

On concatenation of the resulting L.H.S and R.H.S values, the product of the numbers 104 and 102 is obtained to be 10608. The main advantage of this method is that, there is a noticeable reduction in their complement obtained when compared to their original numbers and steps are reduced. This is the reason for Nikhilam Sutra to be efficient but the serious drawback is that, the numbers to be multiplied must have their base nearest to 10 or the power of ten. This motivates for a modification in the existing architecture that not only allows us to have base values other than 10 (or power of ten) but also allows us to multiply any type of number including floating point numbers.

### *B. Urdhva Tiryagbhyam*

Urdhva tiryagbhyam sutra refers to vertical and crosswise [6]. The algorithm of urdhva sutra involves producing summation and partial products. The multiplication steps vary based on the digits. For multiplying two 4 bit numbers, step-1 is initiated by multiplying the right most bit of the two binary numbers. In step 2, last and previous bit is cross multiplied. In step-3, 2nd bit of A and 4th bit of B are multiplied, 4th bit of B and 2nd bit of A are multiplied and 3rd bit of A&B are multiplied. In Step-4, 1st bit of A is multiplied with 4th bit of B, 2nd bit of A multiplied to 3rd bit of B, 3rd bit of A multiplied with 2nd bit of B and finally the 4th bit of B is multiplied to 1st bit of B. Step 5 is similar to step-3, the 1st bit of A is multiplied to 3rd bit of B, 1st bit of B is multiplied to 3rd bit of A and 2nd bit of A&B are multiplied. In step-6, 1st and 2nd bit of A&B are cross multiplied. In step-7, the 1st bit of A&B are multiplied. The steps are as shown below:

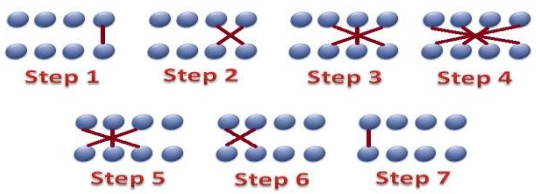



Fig.2. 4-bit multiplication Steps of Urdhva Tiryagbhyagam Sutra

The figure.2 explains the line diagram steps 1-7. The combined result will generate the final product of the two 4-bit numbers.

### *C. Array Multiplier:*

Figure.3 shows that array multiplier has a regular structure and it is based on shift and add algorithm. The multiplier and multiplicand together produce the partial product. Each row of the partial product is shifted left and added. The addition operation is performed through n-1 adders, where "n" is the length of the multiplier [2].

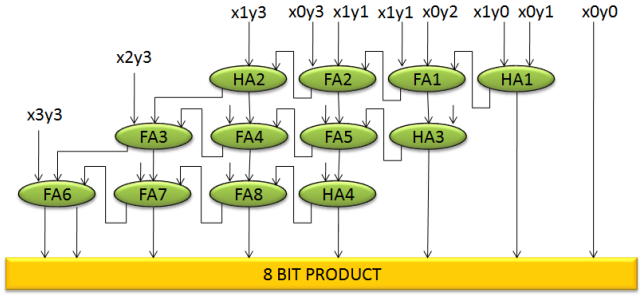



Fig.3 4-bit Array Multiplication

Let m be the number of bits in the multiplicand then n*x*m partial products are produced through an array of AND gates. The bits are multiplied as $x_0y_0$ till $x_{n-1}y_{n-1}$ sequentially.

### *D. Booth Multiplier:*

Booth Multiplier reduces number of iterations required to perform the multiplication when compared to conventional multipliers. The architecture is built using four parts: 2's Complement Generator, Carry Look-Ahead Adder, Booth Encoder and Partial product generator as shown in figure.4.

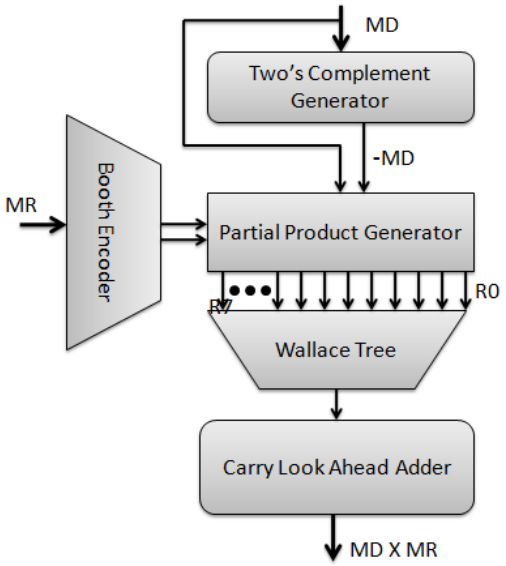



Fig.4. Architecture of Booth Multiplier

Booth Multiplier can reduce the number of additions that is required when compared to conventional multipliers, where each bit of the multiplier and multiplicand is multiplied and their respective partial products are arranged

and summed together. The number of additions depends upon the data.

*E. Wallace Tree :*

The Wallace tree has a powerful and an efficient design to multiply 2 integer values. In Wallace tree, the partial product bits are added in parallel as shown in Fig.5. The number of adders and critical paths are reduced in this technique [7].

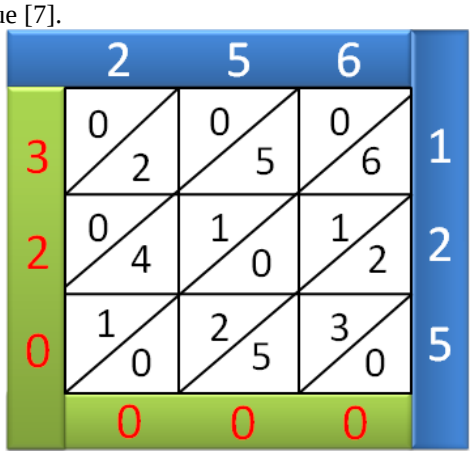



Fig.5 Wallace tree multiplication

- Multiplying the bits of an argument in a row with the bits in the column yields an $n^2$ result.
- The partial products produced are reduced by using half and full adders.
- The results are added diagonally and the carry is added to the neighboring bits.
- By concatenating the resultant values, the product of A and B is generated.

## III. Proposed Work

Vedic sutra has been used to design the Vedic multiplier. The approach to design the proposed multiplier is based on NikhilamNatascaramamDasatah (NND) Sutra. The algorithm of the NND sutra acts as a building block for the proposed Vedic multiplier. The preliminary stages of the work are based on NND algorithm. The modification to the design is performed in the secondary stages which optimizes the functionality. A standard architecture is presented for the proposed technique. Each block aims at reducing the complexity of the design flow when operating on a considerable input. Optimization is achieved by raising the speed and reducing the power consumption. At the initial stages, a 4x4 bit Vedic multiplier is designed using the proposed algorithm and its operation is evaluated using MATLAB. Further, an 8x8 bit Vedic multiplier is designed and tested. The results are perfectly content to design a digital Vedic multiplier. A floating point multiplier is designed using the following design.

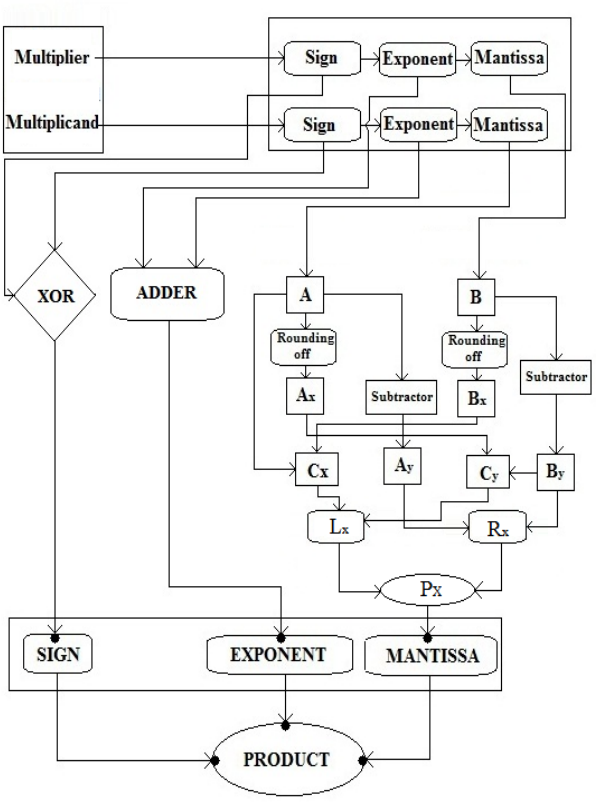



Fig.6 Floating Point Multiplier Design

The figure.6 shows the multiplication operation between multiplier and multiplicand consisting of n-input bits. The input bits are processed and the 2n-outputs are obtained as result. The algorithm of figure.6 is as follows:

*A. Algorithm:*

Notations
$A_x$ = Base of A
$B_x$ = Base of B
$A_y$ = Deviation of A
$B_y$ = Deviation of B
$C_x$ = Partial product 1 (PP1)
$C_y$ = Partial product 2 (PP2)
$L_x$ = Summation of $C_x$ & $C_y$.
$L_y$ = Partial product 3 (PP3)
$P_x$= Partial product 4 (PP4)

The Floating point number multiplication is carried out using the proposed technique. The case 1 is to be satisfied for the numbers to be accepted by the multiplier. Floating point numbers are transformed when case 2 is satisfied.

**Case 1:**

For a real number "A", n > A > 0 value of the multiplier ranges from 0 to infinity. For a real number "B", n > B >0 value of the multiplicand ranges from 0 to infinity. The product of the real numbers is obtained through the proposed Vedic multiplier without any transformation of the given bits.

**Case 2:** For a floating point number "n.n", n.n> A > 0.0 value of the multiplicand ranges from 0 to infinity with bits

on either sides of the decimal point. For an n.n> B > 0.0value of the multiplicand ranges from 0 to infinity with bits on either sides of the decimal point. The floating point input value is converted into sign, exponent and mantissa bits. Based on the given input the conversion takes place in three different IEEE-standards: Single Precision Floating Point (32 bit), Double Precision Floating Point (64 bit), Quadruple Precision Floating Point (128 bit).

*B. Proposed Algorithm:*

```
// Consider A as multiplier and B as multiplicand
//A0 & B0 are the last digits of A & B and are obtained from sub algorithm
if (A0>=5)
        p1 = 10 – A0;
        Base1 = A + p1;
        //Ax is obtained from sub algorithm with Base1 &10
        Ay = A - Base1;
else
        Base1 = A-A0;
        //Ax is obtained from sub algorithm with Base1 &10
        Ay = A - Base1;
end
if (B0>=5)
        p2 = 10 – B0;
        Base2 = Num2 + p2;
        //Bxis obtained from sub algorithm with Base2 &10
        By = B - Base2;
else
        Base2 = B – B0;
        //Bxis obtained from sub algorithm with Base2 &10
        By = B - Base2;
end
```

*C. Sub algorithm:*

```
//Division algorithm
//N is the Numerator and a is the denominator
if a==0
        then error
end
// initialize remainder and quotient to zero
Q == 0
R == 0
//n is the number of bits in N
for  i = n-1..0
do
//left shifting R by 1
        R == R << 1
//Setting the least significant bit of R equal to bit i of the numerator
        R (0) == N (i)
        if   R >= a
                R == R – a
                Q (i) == 1
        end
end
```

*D. Algorithm explanation:*

1. Corresponding base values and their deviations are stored in individual registers.
2. The left and right side values are obtained using the values stored in the previous registers.
3. $C_x$ is given as the multiple of multiplier A and deviation of multiplicand B.($C_x$= A.$B_x$)
4. $C_y$ is given as the multiple of Base of A and deviation of B. ($C_y$= $B_y$.$A_x$)
5. Left side value is given as the summation of $C_x$ and $C_y$. (LHS = $C_x$+ $C_y$)
6. Right side value is given as the multiple of the values stored in the deviation registers of multiplier A and multiplicand B. (RHS= $A_y$.$B_y$)
7. On concatenation of the left and right side values, the product of the multiplier and multiplicand is obtained. (PRODUCT = (LHS) || (RHS))

*E. Floating point multiplier:*

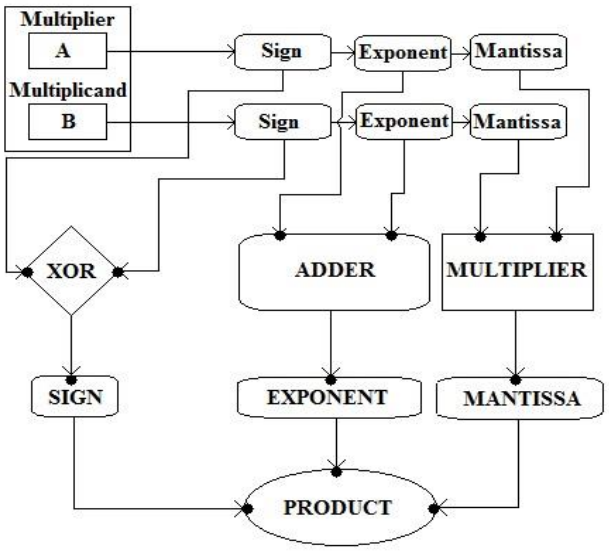



Fig.7Modern Multiplier Design

The above figure.7 shows the working of the proposed modern multiplier. The Algorithm is as follows:

1. The multiplier A and multiplicand B are given as the input to the multiplier.
2. The input bits are converted into sign, exponent and mantissa.
3. The Sign bit of multiplier A and multiplicand B is given to an XOR block.
4. The Exponent bits of A and B are given to an adder.
5. The Mantissa bits are given as an input to the Vedic Multiplier.

The resultant sign, exponent and mantissa values are normalized and the product of A and B is obtained.

## IV. ILLUSTRATION

### A. *Algorithm Analysis using MATLAB*

The proposed design shown in Fig.7 is evaluated using MATLAB.

To study the potential of the architecture, the exploration is advanced by employing 2 different digits. The architecture is now evaluated for a multiplier with a 3 digit multiplier and a 5 digit multiplicand, result as shown in figure-11.

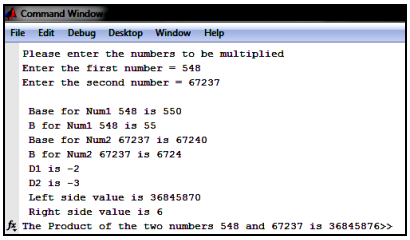



Fig.8Multiplying a 3 & 5 digit number

Finally, the architecture is tested for multiplying floating point numbers. Consider the multiplication of a multiplier 324.777777 and a multiplicand 2938.933333, result as shown in fig.

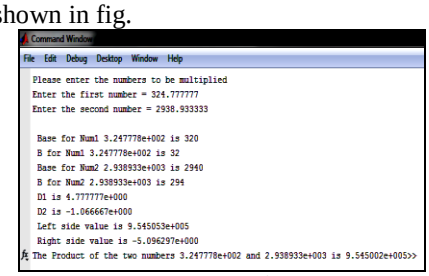



Fig.9Multiplying a 3 & 5 digit floating point number

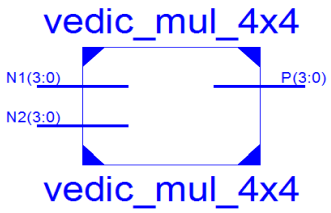
The study exhibit that the method proposed is suitable for multiplying n x n bit real and floating point numbers. This assures the speed and efficiency of the multiplier to be designed.

### B. *Algorithm analysis using XILINX ISE:*

The proposed technique is used to design a Vedic multiplier using Xilinx. A 4x4, 8x8 and 16x16 bit multiplier is constructed in Verilog. Analysis and synthesis is performed in Xilinx ISE 14.1.

- *4x4 bit Proposed Vedic Multiplier:*

Let "N" be the multiplier and "M" be the multiplicand representing 4-bit number, N= $n_3\ n_2\ n_1\ n_0$ and M= $m_3\ m_2\ m_1\ m_0$. The output bits are represented by "Y", Y= $y_7\ y_6\ y_5\ y_4\ y_3\ y_2\ y_1\ y_0$.




(a) RTL Schematic

| dr Project Status (04/10/2016 - 10:17:25) | | | |
|---|---|---|---|
| Project File: | oupt.xise | Parser Errors: | No Errors |
| Module Name: | vedic_mul_4x4 | Implementation State: | Synthesized |
| Target Device: | xc5vlx50t-2ff1136 | • Errors: | No Errors |
| Product Version: | ISE 14.1 | • Warnings: | 30 Warnings (30 new) |
| Design Goal: | Balanced | • Routing Results: | |
| Design Strategy: | Xilinx Default (unlocked) | • Timing Constraints: | |
| Environment: | System Settings | • Final Timing Score: | |

| Device Utilization Summary (estimated values) | | | |
|---|---|---|---|
| Logic Utilization | Used | Available | Utilization |
| Number of Slice LUTs | 37 | 28800 | 0% |
| Number of fully used LUT-FF pairs | 0 | 37 | 0% |
| Number of bonded IOBs | 12 | 480 | 2% |
| Number of DSP48Es | 2 | 48 | 4% |

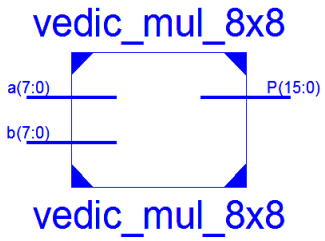
(b)Design Summary

Fig.10 4x4 bit Proposed Vedic Multiplier

A 4x4 bit multiplier device utilization summary is shown in figure.10 and the input 4-bit numbers are multiplied to yield an 8-bit product. The 4x4 bit multiplier consumes small amount of hardware resources.

- *8x8 bit Proposed Vedic Multiplier:*

Let "N" be the multiplier and "M" be the multiplicand representing 4-bit number, N= $n_7\ n_6\ n_5\ n_4\ n_3\ n_2\ n_1\ n_0$ and M= $m_7\ m_6\ m_5\ m_4\ m_3\ m_2\ m_1\ m_0$. The output bits are represented by "Y", Y= $y_{15}\ y_{14}\ldots y_2\ y_1\ y_0$.




(a) RTL Schematic

| sat Project Status (04/09/2016 - 15:52:35) | | | |
|---|---|---|---|
| Project File: | multiplier.xise | Parser Errors: | No Errors |
| Module Name: | sat | Implementation State: | Synthesized |
| Target Device: | xc5vlx50t-2ff1136 | • Errors: | No Errors |
| Product Version: | ISE 14.1 | • Warnings: | 54 Warnings (10 new) |
| Design Goal: | Balanced | • Routing Results: | |
| Design Strategy: | Xilinx Default (unlocked) | • Timing Constraints: | |
| Environment: | System Settings | • Final Timing Score: | |

| Device Utilization Summary (estimated values) | | | |
|---|---|---|---|
| Logic Utilization | Used | Available | Utilization |
| Number of Slice LUTs | 158 | 28800 | 0% |
| Number of fully used LUT-FF pairs | 0 | 158 | 0% |
| Number of bonded IOBs | 27 | 480 | 5% |
| Number of DSP48Es | 4 | 48 | 8% |

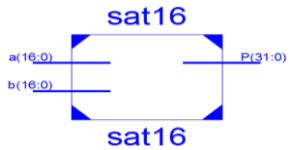
(b)Design Summary

Fig.11 8x8 bit Proposed Vedic Multiplier

An 8x8 bit multiplier device utilization summary is shown in figure.11 and the input 8-bit numbers are multiplied to yield a 16-bit product. Comparing with 4 bit multiplier hardware resources consumed by 8 bit multiplier is slightly larger.

- *16x16 bit Proposed Vedic Multiplier:*

Let "N" be the multiplier and "M" be the multiplicand representing 4-bit number, N= $n_{15}\ n_{14}\ldots n_2\ n_1\ n_0$ and M= $m_{15}\ m_{14}\ldots m_2\ m_1\ m0$. The output bits are represented by "Y", Y= $y_{31}\ y_{30}\ldots y_2\ y_1\ y_0$.




(a)RTL Schematic

| sat16 Project Status (04/11/2016 - 09:50:34) | | | |
|---|---|---|---|
| **Project File:** | multiplier 16.xise | **Parser Errors:** | No Errors |
| **Module Name:** | sat16 | **Implementation State:** | Synthesized |
| **Target Device:** | xc5vlx50t-2ff1136 | **•Errors:** | No Errors |
| **Product Version:** | ISE 14.1 | **•Warnings:** | 123 Warnings (3 new) |
| **Design Goal:** | Balanced | **•Routing Results:** | |
| **Design Strategy:** | Xilinx Default (unlocked) | **•Timing Constraints:** | |
| **Environment:** | System Settings | **•Final Timing Score:** | |

| Device Utilization Summary (estimated values) | | | |
|---|---|---|---|
| **Logic Utilization** | **Used** | **Available** | **Utilization** |
| Number of Slice Registers | 128 | 28800 | 0% |
| Number of Slice LUTs | 2372 | 28800 | 8% |
| Number of fully used LUT-FF pairs | 124 | 2376 | 5% |
| Number of bonded IOBs | 2511 | 480 | 523% |
| Number of BUFG/BUFGCTRLs | 14 | 32 | 43% |
| Number of DSP48Es | 12 | 48 | 25% |

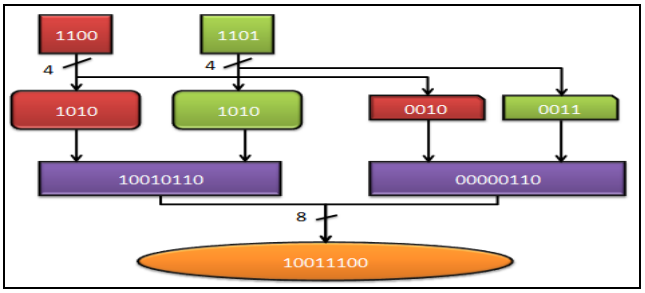
(b) Design Summary

Fig.12 16x16 bit Proposed Vedic Multiplier

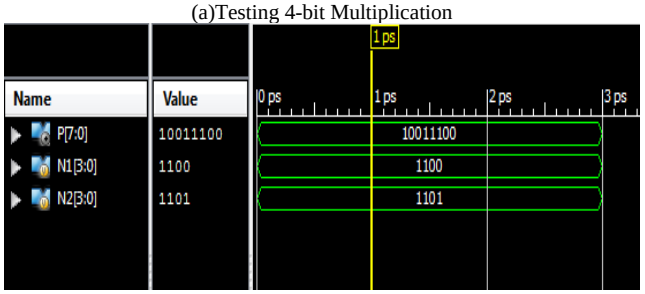
A 16x16 bit multiplier example is shown in figure.12 and the input 16-bit numbers are multiplied to yield a 32-bit product. As the number of bits increases hardware resources consumed also increases.

## V. Simulation and Implementation Results

### A. 4x4 Vedic multiplication:



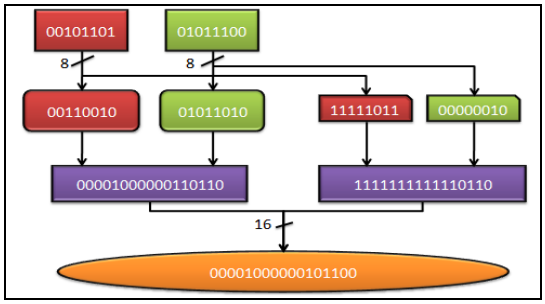

(a)Testing 4-bit Multiplication



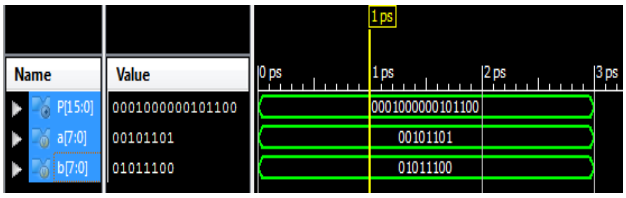

(b) Output

Fig.13Testing 4x4 bit multiplication

### B. 8x8 Vedic Multiplication:



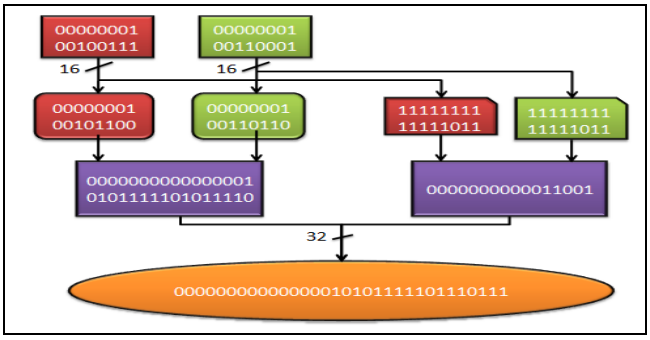

(a) Testing 8-bit Multiplication



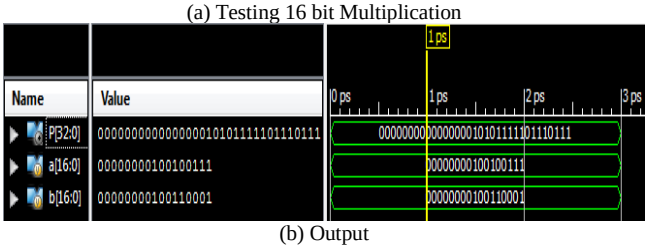

(b) Output

Fig.14Testing 8x8 bit multiplication

### C. 16x16 Vedic Multiplication:




(a) Testing 16 bit Multiplication




(b) Output

Fig.15Testing 16x16 bit multiplication

The table below shows that the propagation delay of the proposed multiplier is better when compared to existing multipliers [7, 8]. The algorithm reduces the complexity of the process.

TABLE I. Comparison With Existing Multipliers

| Multiplier | Bits | Propagation Delay | Power (mW) | Area |
|---|---|---|---|---|
| Wallace Tree | 4 | 16.104ns | 221.93 | 4% |
| | 8 | 36.904ns | 315.02 | 7% |
| | 16 | 64.408ns | 402.38 | 12% |
| Urdhva Tiryagbhyagam | 4 | 15.025ns | 192.37 | 3% |
| | 8 | 25.236ns | 287.90 | 5% |
| | 16 | 60.327ns | 330.63 | 11% |
| Booth | 4 | 20.322ns | 225.15 | 4% |
| | 8 | 46.740ns | 318.31 | 8% |
| | 16 | 63.435ns | 337.92 | 14% |
| Array | 4 | 22.741ns | 230.26 | 5% |
| | 8 | 45.917ns | 329.11 | 8% |
| | 16 | 68.868ns | 412.23 | 15% |
| **Proposed** | 4 | **13.028ns** | **150.45** | **3%** |
| | 8 | **25.281ns** | **225.36** | **5%** |
| | 16 | **35.354ns** | **303.21** | **8%** |

**Graphical Representation:**

The proposed multiplier is compared with the conventional multipliers such as Array, Booth, Wallace tree and UrdhvaTiryagbhyam based multiplier on the parameters of speed, area utilization and power consumption.

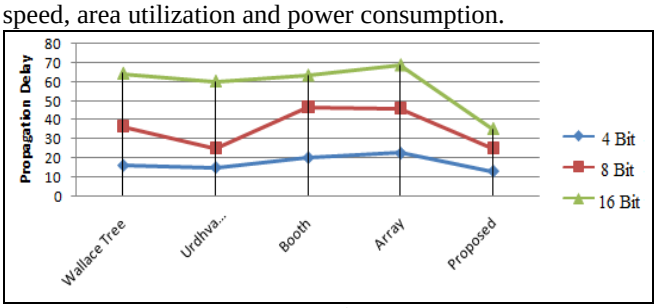



Fig.16 Comparison chart for Propagation Delay

From the comparison of the proposed multiplier with the existing multipliers, the minimum propagation delayis achieved by the proposed multiplier at 13.028ns for 4 bit, 25.281ns for 8 bit and 35.354nsfor 16 bit. The proposed multiplier is faster as there is less number of steps involved in the algorithm.

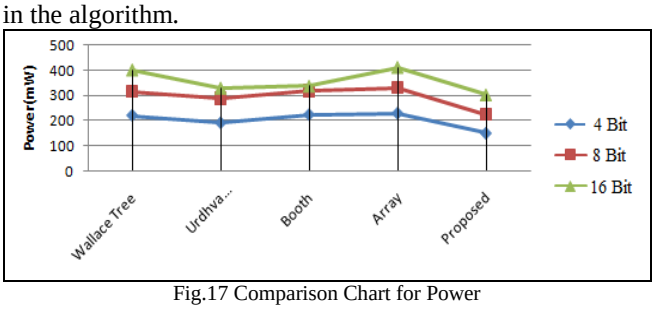



Fig.17 Comparison Chart for Power

On comparing the proposed multiplier with the conventional multipliers, minimum power is achieved on the proposed multiplier design of 150.45mW for 4 bit, 225.36mW for 8 bit and 303.21mW for 16 bit multiplier. Low power consumption is achieved using the compact design.

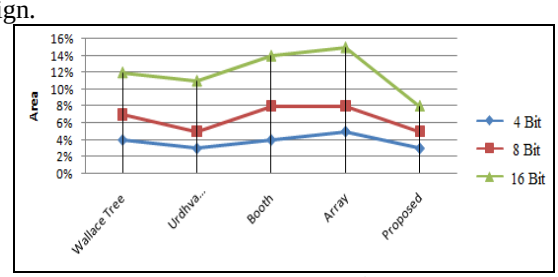



Fig.18 Comparison Chart for Area

On comparing the area utilization of the proposed multiplier with the conventional multiplier, proposed vedic multiplier occupies a negligibly low area when compared to Urdhva Tiryagbhyam.

## VI. Conclusion

A 4x4, 8x8 and 16x16 high speed multiplier is design using Vedic mathematics which is efficient. The architecture of the multiplier is based on NikhilamNavatascaramamDasatah sutra of Vedic mathematics. The proposed design proves to be efficient in terms of power as the number of bits increases when compared to the existing conventional multipliers such as Wallace Tree, UrdhvaTiryagbhyam, Booth and Array multiplier.

As a future enhancement, better adders can be used. As a modification, the design can be made for choosing different base for each value. This will result further in the reduction of time and space.

## *References*


[1] G.Murugesan and S.Lavanya, "Design and Implementation of High Speed multiplier using vedic mathematics", ARPN, VOL.10, NO.16, 2015, 6758-6764.

[2] Annam Aravind Kumar and SK.Mastan Basha, "Design and Implementation of high speed 8bit vedic multiplier on FPGA",IJCERT, VOL.2, Issue12, 2015, pp.1062-1069.

[3] Pranita Soni, Swapnil Kandam, Harish Dhurape and Nikhil Gulavani, "Implementation of 16x16 Bit multiplication algorithm by using vedic mathematics over booth algorithm", IJRET, VOL.4, Issue:5, 2015, 371-376.

[4] M.Vengadapathiraj, V.Rajendhiran, M.Gururaj, A.Vinoth Kannan and R.Gomathi, "Design of high speed 128x128 bit vedic multiplier using high speed adder", IJSERT, VOL.4, Issue:3, 2015,615-619.

[5] B.Keerthi Priya and R.Manoj Kumar, "A novel low power vedic multiplier using modified GDI technique in 45nm technology", IJIRCCE, VOL.3, Issue:11, 2015, 11721-11728.

[6] Parul Agrawal and Rahul Sinha, "Comparative analysis and FPGA implementation of Vedic multiplier for various bit lengths using different adders", IJIRCCE, VOL.3, Issue:10, 2015, 9844-9850.

[7] Shazeeda and D.Monika Sharma, "Design and Implementation of an N-bit vedic multiplier using DCT", IJEAT, VOL.5, Issue:2, 2015, 34-41.

[8] Harish Kumar and A.R.Hemanth Kumar, "Design and Implementation of vedic multiplier using compressor", IJERT, VOL.4, Issue:06, 2015.

[9] Sudhir Dakey and Avinash Nandigama, "Design, Implementation and performance analysis of 8bit vedic multiplier", IJMTER, VOL.2, Issue:06, 2015.

[10] Abilasha and K.M.Sudharshan, A review of an efficient 8bit vedic multiplier using reversible logic, IJRD, VOL.2, Issue:05, 2015.

[11] Amruta Ingle and Shruti Oza, "FPGA implementation of novel high speed vedic multiplier", IJAREEIE, VOL.4, Issue:06, 2015.

[12] Design approach of high performance 32bit multiplier based on vedic mathematics using pipelining", IJARCCE, Vol.04, Issue:09, 2015.

[13] V.B.Baru and Deepak Kurmi, "High Speed 16 bit digital multiplier architect using urdhva tiryagbhyam and compressors", IJAREEIE, VOL04,Issue:3, 2015.

[14] G.Vasudeva and P.Cyril Prasanna Raj , "Study of 8 bit fast multipliers for low power applications", IJSCE, VOL.5, Issue:1, 2015.

[15] Harsh Yadav and Ankit Jain, "Verilog Implementation of an efficient multiplier using vedic mathematics", IJERA, VOL.5, Issue:7, 2015,pp.113-116.