Design of Compact Multiplier Using Vedic Mathematics Technique

V. Poojitha[1], E. Ramakrishna[2]
[2]Assistant Professor, HOD, Dept of ECE
[1, 2] SKD Engineering College, Gooty

Abstract- *This paper describes the design of high speed Vedic multiplier that uses the techniques of Vedic mathematics based on 16 sutras (algorithms) to improve the performance. In this paper the efficiency of Urdhva Tiryagbhyam (vertical and crosswise) Vedic method for multiplication which is different from the process of normal multiplication is presented. Urdhva -Tiryagbhyam is the most efficient algorithm that gives minimum delay for multiplication for all types of numbers irrespective of their size. Vedic multiplier is coded in Verilog HDL and stimulated and synthesized by using XILINX software 12.2 on Spartan 3E kit. Further the design of array multiplier is compared with the proposed multiplier in terms of delay, memory and power consumption.*

I. INTRODUCTION

STRUCTURED VLSI DESIGN:

VLSI aim is flexible approach initiated by CM(Carver Mead) and LC (Lynn Conway) intended for equivalent microchip region by diminishing inter relate fabrics region. This approach is acquired through monotonous procedure regarding quadrangular universal blocks that could be inter related. An instance is apportioning the explain of adder addicted to row about identical tad wedges cell. During compound strategies structure might accomplished through graded nest.

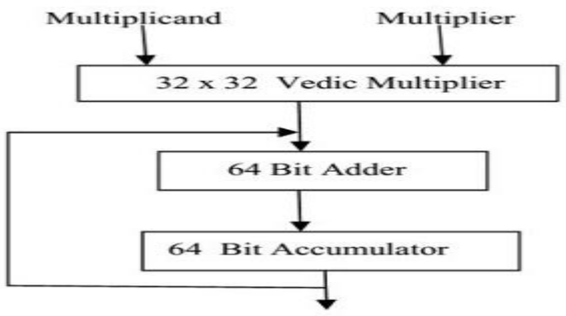
II. LITERATURE SURVEY

EXISTING METHOD:

DESIGN OF 32 BIT MAC ARCHITECTURE:

The design of MAC architecture consists of 3 sub designs is shown in fig

- Design of 32×32 bit Vedic multiplier.
- Design of adder using DKG gate reversible logic.
- Design of accumulator which integrates both multiplier and adder stages.



Modified MAC Architecture

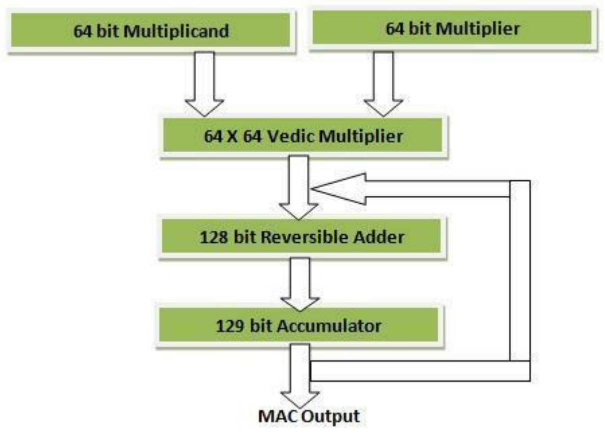
32 X 32 BIT VEDIC MULTIPLIER:

Vedic mathematics is an ancient system of mathematics, which was formulated by Sri Jagadguru Swami Bharati Krishna Tirthaji(1884 - 1960). After a research of eight years,he developed sixteen mathematical formulae from Atharvana Veda. The sutras (aphorisms) covered each and every topic of Mathematics such as Arithmetic, Algebra, Geometry,Trigonometry, differential, integral, etc., The word “Vedic” is derived from the word “Veda” which means the power house of all knowledge and divine. The proposed Vedic multiplier is based on the “UrdhavaTriyagbhayam” sutra (algorithm). These Sutras have been traditionally used for the multiplication of two numbers in the decimal number system..

PROPOSED METHOD:

A multiplying block function can be conceded in three different ways: conventional addition, partial product addition (PPA) and finally partial product Generation (PPG). The two bud vase materials that must be considered are raising the speed of MAC which is accumulator block partial and product reduction. The 64 bit MAC design which will make use of Vedic multiplier and reversible logic gate can be accomplished in two stages. Firstly, multiplier stage, where a usual multiplier is replaced by Vedic multiplier using Urdhava Triyagbhayam sutra from Vedic Mathematics. Multiplication is the primary operation of MAC unit. Speed, area, Power dissipation, consumption and latency are the major concerns in the multiplier stage. So, to evade them, we will go for fast multipliers in different applications of DSP, networking, etc.

DESIGN OF MAC ARCHITECTURE



MAC Architecture

The design of MAC architecture consists of 3 sub designs is shown in fig.

- Design of 64×64 bit Vedic Multiplier.
- Design of adder using DKG gate reversible logic.
- Design of Accumulator which integrates both multiplier and adder stages.

III. MULTIPLIERS

A compressor is a digital circuit which is used to increases the calculation speed and decreases time of the addition of 4 or more than 4 bits at the same time. Compressor adders can capable to replace the combination of several half adders and full adders, thereby enabling high speed performance of the processor which incorporates the same. The compressor adder used in the paper is a 4:2 and 7:2 compressor adders.

Multipliers are playing a crucial role in today‟s DSP and various other applications. With improvements in the field of science and technology, many researchers have tried and are trying to design multipliers which can provide good features like low power consumption, high speed, regularity in layout structure and due to this less area or even combining of them in one multiplier thus making multipliers suitable for various high speed, low power and compact VLSI designs.

MULTIPLICATION ALGORITHM:

The general multiplication requires multiplier and multiplicand. The multiplication algorithm used to multiply an N bit multiplicand by N bit multiplier is given below:

A= An-1 An-2 ....A2 A1 A0 Multiplier

B= Bn-1 Bn-2 .... B2 B1 B0
Multiplicand
Generally,

$$B = B_{n-1}\,B_{n-2}\ \dots\dots\dots\dots\dots\ B_2\,B_1B_0 \qquad \text{equation (3.1)}$$

$$A = A_{\,n-1}\,A_{n-2}\ \dots\dots\dots\dots\dots\ A_2\,A_1A_0 \qquad \text{equation (3.2)}$$

$$B_{n-1}A_0B_{n-2}A_0B_{n-3}A_0 \dots\dots\dots\dots B_1A_0B_0A_0$$

$$B_{n-1}A_1B_{n-2}A_1B_{n-3}A_1 \dots\dots\dots\dots B_1A_1B_0A_1$$

$$\dots\ \dots\ \dots\ \dots\ \dots\ \dots\ \dots$$

$$\dots\ \dots\ \dots\ \dots\ \dots\ \dots\ \dots$$

$$\dots\ \dots\ \dots\ \dots\ \dots\ \dots\ \dots$$

$$B_{n-1}A_{n-2}B_{n-2}A_{n-2}B_{n-3}A_{n-2} \dots\dots\dots\dots B_1A_{n-2}B_0A_{n-2}$$

$$B_{n-1}A_{n-1}B_{n-2}A_{n-1}B_{n-3}A_{n-1} \dots\dots\dots\dots B_1A_{n-1}B_0A_{n-1}$$

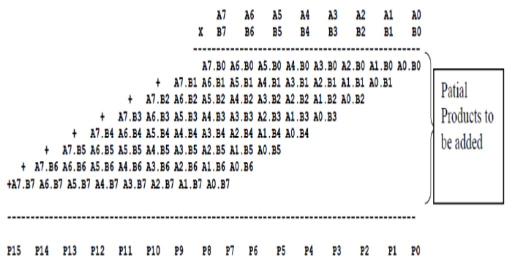
$$P_{2n-1}\quad P_{2n-2}P_{2n-3}\quad \dots\ \dots\ \dots\ \dots\quad P_2\qquad P_1P_0 \qquad \text{equation (3.3)}$$

AND gates are used as the multiplier of two bits these also called partial products short name is PP. If the multiplicand is A-bits and the Multiplier is B-bits then there is A* B partial product terms. The approach that the partial products are obtained or added up is the difference between the different architectures of several multipliers.

```
Example        1101    4-bits
               1101    4-bits
         ----------------------
               1101
              0000
             1101
            1101
         ______________________
           10101001
```

Example of binary normal multiplication.

The Multiplication of binary numbers can be decomposed into some additions. Suppose if we consider the multiplication of two 8-bit numbers A and B and it produce the 16 bit product P.


A7 A6 A5 A4 A3 A2 A1 A0
X B7 B6 B5 B4 B3 B2 B1 B0
A7.B0 A6.B0 A5.B0 A4.B0 A3.B0 A2.B0 A1.B0 A0.B0
+ A7.B1 A6.B1 A5.B1 A4.B1 A3.B1 A2.B1 A1.B1 A0.B1
+ A7.B2 A6.B2 A5.B2 A4.B2 A3.B2 A2.B2 A1.B2 A0.B2
+ A7.B3 A6.B3 A5.B3 A4.B3 A3.B3 A2.B3 A1.B3 A0.B3
+ A7.B4 A6.B4 A5.B4 A4.B4 A3.B4 A2.B4 A1.B4 A0.B4
+ A7.B5 A6.B5 A5.B5 A4.B5 A3.B5 A2.B5 A1.B5 A0.B5
+ A7.B6 A6.B6 A5.B6 A4.B6 A3.B6 A2.B6 A1.B6 A0.B6
+A7.B7 A6.B7 A5.B7 A4.B7 A3.B7 A2.B7 A1.B7 A0.B7
P15 P14 P13 P12 P11 P10 P9 P8 P7 P6 P5 P4 P3 P2 P1 P0
Patial Products to be added

multiplication of two variables

The addition equation is shown below.

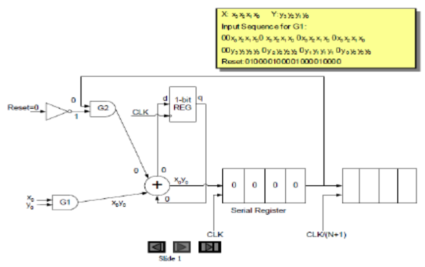
$$P(m+n) = A(m)\,B(n) = \sum_{i=0}^{m-1}\sum_{j=0}^{n-1} a_i b_j 2^{i+j} \qquad \text{equation (3.4)}$$

- If the Least Significant Bit of Multiplier is „1", then add the multiplicand into an accumulator.
- Shift the multiplier one bit to right side and multiplicand one bit to the left.
- Stop when all bits of the multiplier are zero.

From above it is clear that the multiplication has been changed to addition of numbers. If the Partial Products are added serially then a serial adder is used with less number of gates. It is possible to add all the partial products with one combinational circuit using parallel multipliers. On the other hand it is possible also, to use compression technique then the number of partial products can be reduced before addition operation is performed

SERIAL MULTIPLIER:

Where power and area is of highest importance and delay can be tolerating the serial multiplier is used. This circuit uses one adder to add the × partial products. The circuit is shown in the fig 3.3 below for m=n=4. Multiplicand and Multiplier inputs have to be arranged in a special manner synchronized with circuit behavior as shown in the figure.



Serial multiplier

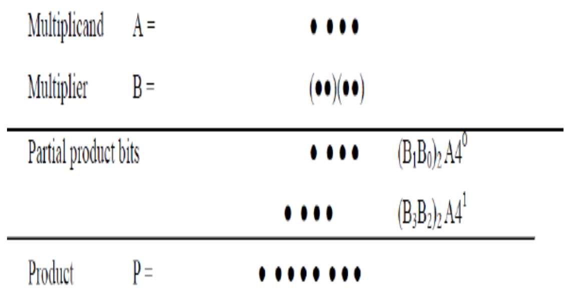
The inputs could be presented at different rates depending on the length of the multiplicand and the multiplier. Two clocks are used, one to clock the data and one for the reset. A first order approximation of the delay is O (m,n). With this circuit arrangement the delay is given as

$$D = [\,(m+1)n + 1\,]\, t_{fa}. \qquad \text{equation (3.5)}$$

As shown the individual partial products are formed individually. The addition of the partial products are performed as the intermediate values of partial products addition are stored in the DFF, circulated and added together with the newly formed PP. This approach is not suitable for large values of M or N.

BOOTH MULTIPLIERS:

Booth multiplier is a powerful algorithm for signed number multiplication, which treats both negative and positive numbers uniformly. For the standard add and shift operations, each multiplier bit generate one multiple of the multiplicand to be added to the partial product. If the multiplier is large, then a large number of multiplicands have to be added. In this case the delay of multiplier is determined by the number of additions to be performed. If there is a way present to reduce the number of the additions, the performance will be high



Radix-4 multiplication in dot notation

Booth algorithm is a method that will reduce the number of multiplicand multiples. For a given range of numbers to be represented, a higher representation radix leads to lesser digits. Since a k-bit binary number can be interpreting as K/2-digit radix-4 number, a K/3-digit radix-8 number, and so on, it can deal with more than single bit of the multiplier in each cycle by using radix multiplication. This is shown for Radix-4 in the above example.

As shown in the figure, if multiplication is done in radix 4, in each step, the partial product term (Bi+1Bi)2 A needs to be formed and added to the cumulative partial product. Whereas in radix-2 multiplication, each row of dots in the partial products matrix represents 0 or a shifted version of A must be included and added. Table 1below is used to convert a binary number to radix-4 number.

Initially, a "0" is placed to the right most bit of the multiplier. Then 3 bits of the multiplicand is recoded according to table below or according to the following equation:

$$Zi = -2x_{i+1} + x_i + x_{i-1} \qquad \text{equation (3.6)}$$

Example:

Multiplier is equal to 0 1 0 1 1 10

gives 0 1 0 1 1 10 0 the 3 digits are selected at a time with overlapping left most bit as follows:

Radix-4 Booth recoding

X_i+1	Y	X_i-1	Z_i/2
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	2
1	0	0	-2
1	0	1	-1
1	1	0	-1
1	1	1	0

For example, an unsigned number can be converted into a signed-digit number radix 4:
(10 1 11 01 10 10 11 10)2 = (–2 2 –1 2 –1 –1 0 –2)4

IV. VEDIC MATHEMATICS

Vedic mathematics is part of four Vedas (books of wisdom). It is part of Sthapatya- Veda (book on civil engineering and architecture), which is an upa-veda (supplement) of Atharva Veda.

It covers explanation of several modern mathematical terms including arithmetic, geometry (plane, co-ordinate), trigonometry, quadratic equations, factorization and even calculus. His Holiness Jagadguru Shankaracharya Bharati Krishna Teerthaji Maharaja (1884-1960) comprised all this work together and gave its mathematical explanation while discussing it for various applications. Swamiji constructed 16 sutras (formulae) and 16 Upa sutras (sub formulae) after extensive research in Atharva Veda. The very word "Veda" has the derivational meaning i.e. the fountainhead and illimitable storehouse of all knowledge. Vedic mathematics is the name given to the ancient system of mathematics or, to be precise a unique technique of calculations based on simple rules and principles with which many mathematical problems can be solved, be it arithmetic, algebra, geometry or trigonometry.

Vedic Mathematics existed in ancient India and was rediscovered by a popular mathematician, Sri Bharati Krishna Tirthaji. He divided Vedic mathematics into 16 simple sutras (formulae). These Sutras deal with Arithmetic, Algebra, Geometry, Trigonometry, Analytical Geometry etc. The simplicity in the Vedic mathematics sutras paves way for its application in several prominent domains of engineering like Signal Processing, Control Engineering and VLSI.

- (Anurupye) Shunyamanyat -If one is in ratio, the other is zero.
- Chalana Kalanabyham -Differences and similarities.
- Ekadhikina Purvena- By one more than the previous One.
- Ekanyunena Purvena -By one less than the previous one.
- Gunakasamuchyah-Factors of the sum is equal to the sum of factors.
- Gunitasamuchyah-The product of sum is equal to sum of the product.
- Nikhilam Navatashcaramam Dashatah -All from 9 and last from 10.
- Paraavartya Yojayet-Transpose and adjust.
- Puranapuranabyham -By the completion or noncompletion.
- Sankalana- vyavakalanabhyam -By addition and by subtraction.
- Shesanyankena Charamena- The remainders by the last digit.
- Shunyam Saamyasamuccaye -When the sum is same then sum is zero.
- Sopaantyadvayamantyam -The ultimate and twice the penultimate.
- Urdhva- Triyagbhayam -Vertically and crosswise.
- Vyashtisamanstih -Part and Whole.
- Yaavadunam- Whatever the extent of its deficiency.

Vedic Mathematics can be bifurcated into 16 different sutras to perform mathematical operations. Among these surtras the Urdhwa Tiryakbhyam Sutra is one of the most highly preferred algorithms for performing multiplication. The algorithm is competent enough to be employed for the multiplication of integers as well as binary numbers. The term "Urdhva Tiryagbhyam ” originated from 2 Sanskrit words Urdhwa and Tiryakbhyam which mean “vertically” and “crosswise” respectively. The main advantage of utilizing this algorithm in comparison with the existing multiplication techniques, is the fact that it utilizes only logical “AND” operations, half adders and full adders to complete the multiplication operation. Also, the partial products required for multiplication are generated in parallel and apriority to the actual addition thus saving a lot of processing time.

URDHVA TIRYAGBHYAM ALGORITHM:

Let us consider two 8 bit numbers X7-X0 and Y7-Y0, where 0 to 7 represent bits from the Least Significant Bit (LSB) to the Most Significant Bit (MSB). P0 to P15 represent each bit of the final computed product. It can be seen from equation (1) to (15), that P0 to P15 are calculated by adding partial products, which are calculated previously using the logical AND operation. The individual bits obtained from equations (1) to (15), in turn when concatenated produce the final product of multiplication which is depicted in (16).The carry bits generated during the calculation of the individual bits of the final product are represented from C1 to C30. The carry bits generated in (14) and (15) are ignored since they are superfluous.

$$P0 = A0 * B0$$

$$C1P1 = (A1 * B0) + (A0 * B1)$$

$$C3C2P2 = (A2 * B0) + (A0 * B2) + (A1 * B1) + C1$$

$$C5C4P3 = (A3 * B0) + (A2 * B1) + (A1 * B2) + (A0 * B3) + C2$$

$$C7C6P4 = (A4 * B0) + (A3 * B1) + (A2 * B2)$$

$$+ (A1 * B3) + (A0 * B4) + C3 + C4$$

$$C10C9C8P5 = (A5 * B0) + (A4 * B1) + (A3 * B2) +$$

$$(A2 * B3) + (A1 * B4) + (A0 * B5) + C5 + C6$$

$$C13C12C11P6 = (A6 * B0) + (A5 * B1) + (A4 * B2) +$$

$$(A3 * B3) + (A2 * B4) + (A1 * B5)$$

$$+ (A0 * B6) + C7 + C8$$

$$C16C15C14P7 = (A7 * B0) + (A6 * B1) + (A5 * B2) +$$

$$(A4 * B3) + (A2 * B5) + (A1 * B6) +$$

$$(A0 * B7) + C9 + C11$$

$$C19C18C17P8 = (A7 * B1) + (A6 * B2) + (A5 * B3) +$$

$$(A4 * B4) + (A3 * B5) + (A2 * B6) +$$

$$(A1 * B7) + C10 + C12 + C14$$

$$C22C21C20P9 = (A7 * B2) + (A6 * B3) + (A5 * B4) +$$

$$(A4 * B5) + (A3 * B6) + (A2 * B7)$$

$$+ C13 + C15 + C17$$

$$C25C24C23P10 = (A7 * B3) + (A6 * B4) + (A5 * B5) +$$

$$(A4 * B6) + (A3 * B7) + C16 + C18 + C20$$

$$C27C26P11 = (A7 * B4) + (A6 * B5) + (A5 * B6) +$$

$$(A4 * B7) + C19 + C21 + C23$$

$$C29C28P12 = (A7 * B5) + (A5 * B6) +$$

$$(A4 * B7) + C19 + C21 + C23$$

$$C29C28P12 = (A7 * B5) + (A5 * B6) +$$

$$(A5 * B7) + C22 + C24 + C26$$

$$C30P13 = (A7 * B6) + (A6 * B7) + C25 + C27 + C28$$

$$P14 = (A7 * B7) + C29 + C30$$

$$P15 = (A7 * B7)$$

Graphically illustrates the step by step method of multiplying two 8 bit numbers using the Urdhwa Tiryagkbhyam Sutra. The black circles indicate the bits of the multiplier and multiplicand, and the two-way arrows indicate the bits to be multiplied in order to arrive at the individual bits of the final product. The hardware architecture of the 8x8 Urdhwa multiplier has been designed and shown in Fig.

EXAMPLE FOR URDHVATIRYAGBHYAM ALGORITHM:

Now let an example to explain the multiplication procedure of Urdhwa Tiryakbhyam Sutra. The 3 by 3 decimal number is taken and the procedure is shown.

654 – Multiplier
321 – Multiplicand

- The first step is like as shown in the figure. The multiplication should done between the first digits in each numbers from right hand side.

4×1= 4

- The second step is multiplying cross wise first digit in first number and second digit in second number and vice versa.

4×2=8
5×1=5

And now add 8 and 5 gives 13. In number 13 tens place number is the Carry.

- The third step is multiplying first digit in first number with third digit in second number , second digit in both numbers and first digit in second number with third digit in first number

4×3=12
5×2=10
6×1=6

Here we got 2 digit number, so we have another carry which will propagate to next stage.

- Next step is multiplying second digit in first number with third digit in second number and vice

5×3=15
6×2=12

Here we got 2 digit number, so we have another carry which will propagate to next stage.

- Final step is multiplying last digits in both numbers with each other. 6×3=18
- And now add this partial product and previous carry that results final partial product
  18+2=20.

Now arrange partial products without carry from bottom to top in a sequence from left to right, we will get final product

Result = 209934

The procedure which we use here is used for binary numbers also. Now, let applying same procedure for two 8 bit numbers as shown below. Partial products can be denoted as P0 to P15.

P=10110110 – multiplier
Q=11011001– multiplicand

Now the partial products are from P14 to P0. The partial product has two parts carry and product. Right most digit in the partial product is the product and remaining part is the carry. To get final product write the each product from P14 to P0,this is the required final product.

Final Result = 1 0 0 1 1 0 1 0 0 1 0 0 0 1 1 0

The number of multiplications required is same for the Urdhwa Tiryakbhyam algorithm and normal multiplication but the advantage of the Urdhwa Tiryakbhyam algorithm is while implementing the circuit the partial product come directly, and also there is no need of complex time consuming operations like shifting and rotating. This algorithm eliminates sequential operations so speed of operation increases. So UT algorithm is best suited for the hard ware implementation with less complexity.

HALF ADDER:

Half adder is the basic unit of this multiplier circuit. The half adder circuit adds two bits and gives two outputs, sum and the carry. The sum is the result of the XOR gate and carry is the result of the AND gate. The circuit of the half adder is shown in below fig.

Half Adder circuit diagram

The truth table of the half adder is shown in below table

Half adder Truth table

X	Y	SUM
0	0	0
0	1	1
	0	1
1	1	0

FULL ADDER:

The full adder is the circuit which will add two bits besides carry, totally three bits, same as half adder it gives two outputs sum an carry. The full adder is combination of two half adders and one or gate. The circuit diagram is shown in below figure.

Full Adder circuit diagram

The truth table of the half adder is shown in below table

Full adder Truth table

X	Y	CIN	SUM	CARRY
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

As mentioned earlier, the partial products obtained are added with the help of full adders and half adders. It can be seen, from equation (1) to (16), that in few equations there is a necessity of adding more than 3 bits at a time. This leads to additional hardware and additional stages, since the full adder is capable of adding only 3 bits at a time. In the next section two different types of compressor architectures are explored which assist in adding more that 3 bits at a time, with reduced architecture and increased efficiency in terms of speed.

IMPLEMENTATION

In the accumulate adder the previous MAC output and the present output will added and it consists of Multiplier unit, one adder unit and both will get be combined by an accumulate unit. The major applications of Multiply-Accumulate (MAC) unit are microprocessors, logic units and digital signal processors, since it determines the speed of the overall system. The efficient designs by MAC unit are Nonlinear Computation like Discrete Cosine or wavelet Transform (DCT), FFT/IFFT. Since, they are basically executed by insistent application of multiplication and addition, the entire speed and performance can be compute by the speed of the addition and multiplication taking place in the system. Generally the delay, mainly critical delay, happens due to the long multiplication process and the propagation delay is observed because of parallel adders in the addition stage. The main idea of this paper is comparison of area, speed and other parameters of Conventional MAC unit with the Vedic MAC design.

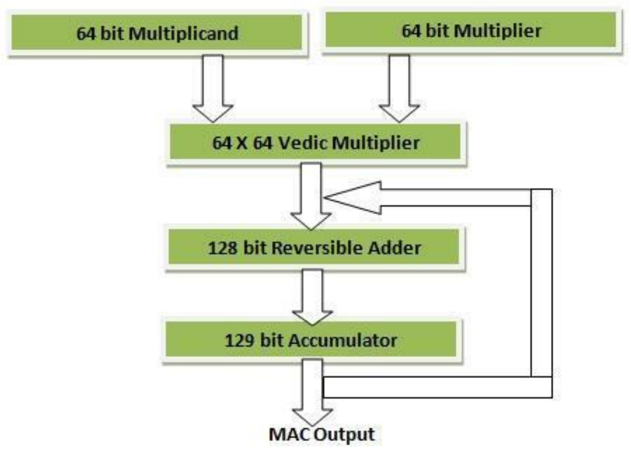
A multiplying function can be carried out in three ways: partial product Generation (PPG), partial product addition (PPA), and final conventional addition. The two bottle necks that should be considered are increasing the speed of MAC are partial product reduction and accumulator block. The 32 bit Mac design by using Vedic multiplier and reversible logic gate can be done in two parts. First, multiplier unit, where a conventional multiplier is replaced by Vedic multiplier using Urdhva Triyagbhayam sutra. Multiplication is the fundamental operation of MAC unit. Power consumption, dissipation, area, speed and latency are the major issues in the multiplier unit. So, to avoid them, we go for fast multipliers in various applications of DSP, networking, etc. There are two major criterions that improve the speed of the MAC units are reducing the partial products and because of that accumulator burden is getting reduced. The basic operational blocks in digital system in which the multiplier determines the critical path and the delay. The (log2N + 1) partial products are produced by 2N-1 cross products of different widths for N*N. The partial products are generated by Urdhava sutra is by Criss Cross Method. The maximum number of bits in partial products will lead to Critical path. The second part of MAC is Reversible logic gate. In modern VLSI, fast switching of signals leads to more power 2015 International Conference on Circuit, Power and Computing Technologies [ICCPCT] dissipation. Loss of every bit of information in the computations that are not reversible is kT*log2 joules of heat energy is generated, where k is Boltzmann‟s constant and T the absolute temperature at which computation is performed. In recent years, reversible logic functions has emerged and played a vital role in several fields such as Optical, Nano, Cryptography, etc.

DESIGN OF MAC ARCHITECTURE:

The design of MAC architecture consists of 3 sub designs.

- Design of 64 × 64 bit Vedic multiplier.
- Design of adder using DKG gate reversible logic.

- Design of accumulator which integrates both multiplier and adder stages.



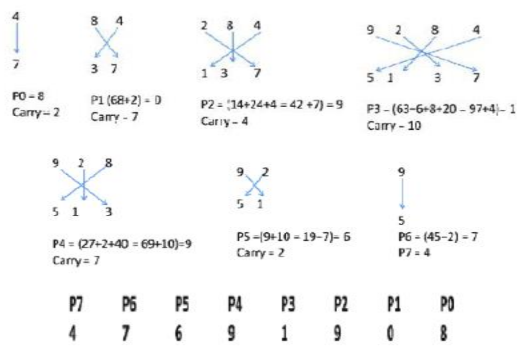
Modified MAC Architecture

64 X 64 BIT VEDIC MULTIPLIER:

Vedic mathematics is an ancient system of mathematics, which was formulated by Sri Jagadguru Swami Bharati Krishna Tirthaji (1884 - 1960). After a research of eight years, he developed sixteen mathematical formulae from Atharvana Veda. The sutras (aphorisms) covered each and every topic of Mathematics such as Arithmetic, Algebra, Geometry, Trigonometry, differential, integral, etc., The word “Vedic” is derived from the word “Veda” which means the power house of all knowledge and divine. The proposed Vedic multiplier is based on the “Urdhva Triyagbhyam” sutra (algorithm). These Sutras have been traditionally used for the multiplication of two numbers in the decimal number system. In this work, we will utilize similar techniques to solve the binary number system to make the new aphorism, which will be more compatible for the digital systems. It is a general multiplication formula applicable to all cases of multiplication.

URDHVA TRIYAGBHYAM SUTRA:

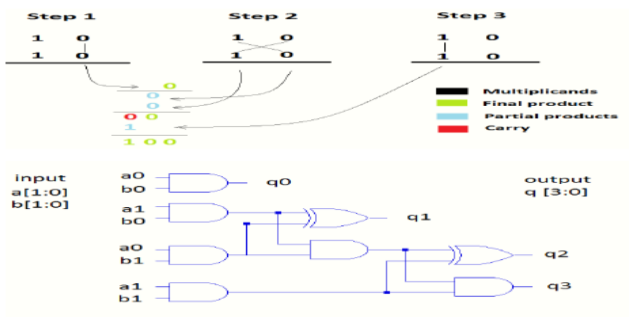
It literally means “Vertically and Cross wise”. Shift operation is not necessary because the partial product calculation will perform it in a single step, which in turn saves time and power. This is the main advantage of the Vedic multiplier. An example for the Urdhva Triyagbhayam sutra is as follows:
9284 * 5137



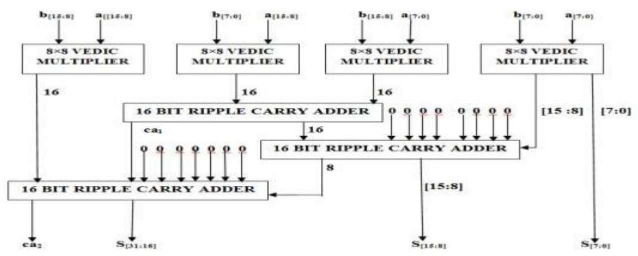
The word Vedic is derived from the word Veda which means the store-house of all knowledge. Vedic mathematics is mainly based o n 16 Sutras (or aphorisms) dealing with various branches of mathematics like arithmetic, algebra, geometry etc. The proposed Vedic multiplier is based on the Vedic multiplication for mul (Sutras) .These Sutras have been traditionally used for the multiplication of two numbers in the decimal number system. In this work, we apply the same ideas to the binary number system to make the proposed algorithm compatible with the digital hardware.

2X2 BIT MULTIPLIER:

In 2x2 bit multiplier, the multiplicand has 2 bits each and the result of multiplication is o f 4 bits. So in input the range of inputs goes from (00) to (11) and output lies in the set of (0000, 0001, 0010, 0011, 0100, 0110, 1001). Focusing on these facts, a simple design by using Urdhva Triyagbhyam. Here multiplicands a and b are taken to be (10) both. The first step in the multiplication is vertical multiplication of LSB of both multiplicand s, then is the second step, that is crosswise multiplication and addition of the partial pro ducts. Then Step 3 involves vertical multiplication of MSB o f the multiplicands and addition with the carry propagated from Step 2.

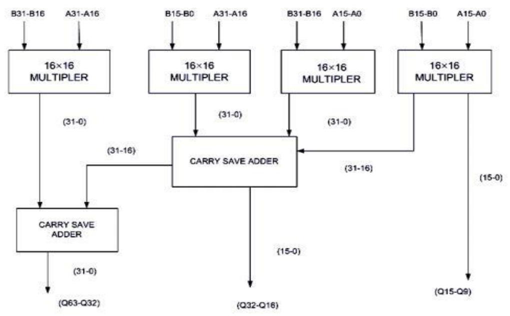


bit multiplier using logic Gates

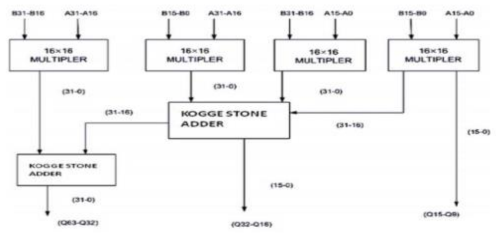


16×16 Vedic multiplier using 8×8 Vedic multiplier

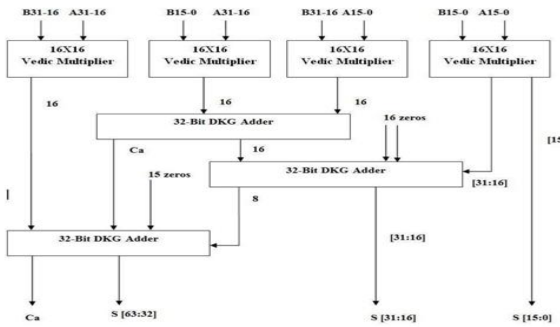
32 × 32 Vedic Multiplier with



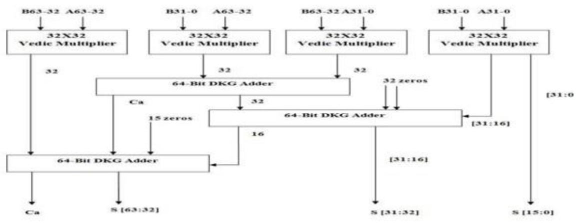
Carry save Adder



32 × 32 Vedic Multiplier with Kogge Stone Adder



32 ×32 Vedic multiplier using 16 × 16 Vedic multiplier



64 × 64 Vedic multiplier using 32 × 32 Vedic multiplier

DESIGN OF ADDER USING REVERSIBLE LOGIC DKG GATE:

REVERSIBLE LOGIC:

Reversible logic is a unique technique (different from other logic). Loss of information is not possible in here. In this, the numbers of outputs are equal to the number of inputs.

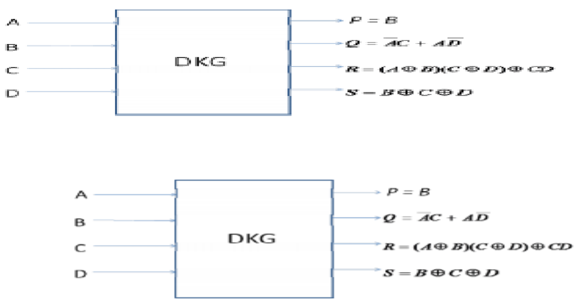
GENERAL CONSIDERATION FOR REVERSIBLE LOGIC GATE:

A Boolean function is reversible if each value in the input set can be mapped with a unique value in the output set. Landauer proved that the usage of traditional irreversible circuits leads to power dissipation and Bennet showed that a circuit consisting of only reversible gates does not dissipate power. In the design of reversible logic circuits, the following points must be kept in mind to achieve an optimized circuit:

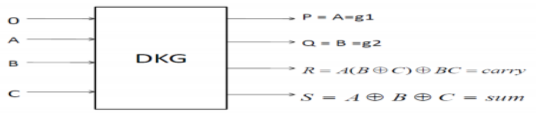
- Fan-out is not permitted
- Loops or feedbacks are not permitted
- Garbage outputs must be Minimum
- Minimum delay
- Minimum quantum cost
- Zero energy dissipation

DKG GATE:

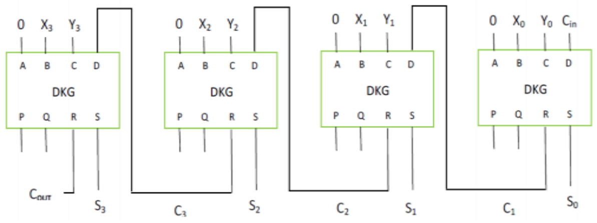
A 4* 4 reversible DKG gate that can work singly as a reversible full adder and a reversible full subtractor is shown below. If input A=0, the DKG gate works as a reversible Full adder, and if input A=1 then it works as a reversible Full subtractor. It has been verified that a reversible full-adder circuit requires at least two or three garbage outputs to make the output combinations unique.



DKG gate


O
A
B
C
DKG
P = A=g1
Q = B =g2
$R = A(B \oplus C) \oplus BC = carry$
$S = A \oplus B \oplus C = sum$

DKG gate as a Full adder



Parallel adder using DKG gate

ACCUMULATOR STAGE:

Accumulator has an important role in the DSP applications in various ranges and is a very basic and common method. The register designed in the accumulator is used to add the multiplied numbers. Multiplier, adder and an accumulator are forming the essential foundation for the MAC unit. The conventional MAC unit has a multiplier and multiplicand to do the basic multiplication and some parallel adders to add the partial products generated in the previous step. To get the final multiplication output we add the partial product to these results. Vedic Multiplier has put forward to intensify the action of the MAC Unit. The suggested MAC is compared with the conventional MAC and the results are analyzed. The results obtained using our design had better performance when compared to the pervious MAC designs.

IV. XILINX

XILINX ISE OVERVIEW:

The Integrated Software Environment (ISE™) is the Xilinx® design software suite that allows you to take our design from design entry through Xilinx device programming. The ISE Project Navigator process and manages our design through the following steps in the ISE design flow.

DESIGN ENTRY:

Design entry is the first step in the Xilinx ISE design flow. During design entry, we create your source files based on our design. We can construct our top-level design file using a Hardware Description Language (HDL), such as Verilog, VHDL or ABEL, or using a schematic. We can use multiple formats for the lower-level source files in our design.

SYNTHESIS:

After completion of design entry and simulation of the code, we run synthesis. During synthesis, Verilog, VHDL, or mixed level designs become net list files that are accepted as input to the implementation step.

IMPLEMENTATION:

After synthesis, we run design implementation, which converts the logical design into a physical file format that can be downloaded to the selected target device. From Project Navigator, we proceed to the implementation process, or we can run each of the implementation processes separately. Implementation processes vary depending on whether we are targeting a Complex Programmable Logic Device (CPLD) or a Field Programmable Gate Array (FPGA).

VERIFICATION:

We can verify the functionality of our design at steps in the design flow. We can use simulator software to verify the functionality and timing of our design or a portion of our design. Simulation allows you to create and verify complex functions in a relatively small amount of time. The simulator interprets Verilog or VHDL code into circuit functionality and displays logical results of the described Hardware Description Language to determine correct operation of circuit.

Simulation allows you to create and verify complex functions in a relatively small amount of time. we can also run in-circuit verification after programming your device.

DEVICE CONFIGURATION:

After generating a programming file, you configure your device. During configuration, you generate configuration files and download the programming files from a host computer to a Xilinx device

PROJECT NAVIGATOR OVERVIEW:

Project Navigator organizes your design files and runs processes to move the design from design entry through implementation to programming the targeted Xilinx® device. Project Navigator is the high-level manager for your Xilinx FPGA and CPLD designs, which allows you to do the following:

- Add and create design source files, which appear in the Sources window
- Modify your source files in the Workspace
- Run processes on your source files in the Processes window

- View output from the processes in the Transcript window

PROJECT NAVIGATOR MAIN WINDOW:

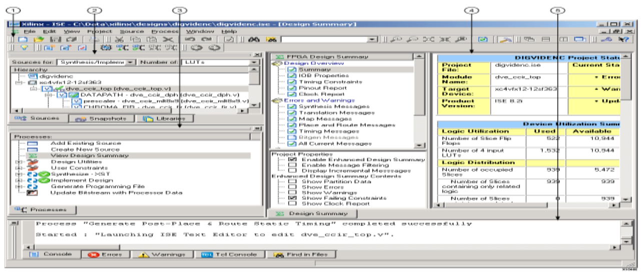
After generating a programming file, you configure your device. During configuration, you generate configuration files and download the programming files from a host computer to a Xilinx device.

The following figure shows the Project Navigator main window, which allows you to manage your design starting with design entry through device configuration.

1. Toolbar
2. Sources window
3. Processes window
4. Workspace
5. Transcript window

USING THE SOURCES WINDOW:

The first step in implementing your design for a Xilinx® FPGA or CPLD is to assemble the design source files into a project. The Sources tab in the Sources window shows the source files you create and add to your project, as shown in the following figure. For information on creating projects and source files, see Creating a Project and Creating a Source File.

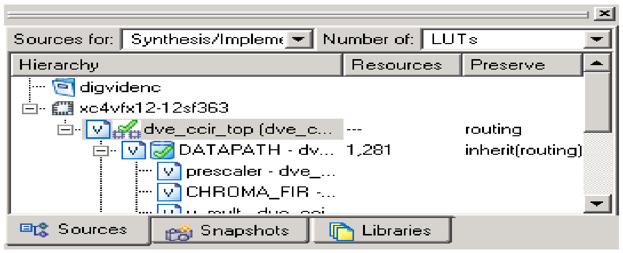


DIGVIDENC Project Status		
Project File:	digvidenc.ise	Current Sta
Module Name:	dve_ccir_top	• Error
Target Device:	xc4vfx12-12sf363	• War
Product Version:	ISE 8.2i	• Upd

Device Utilization Summary		
Logic Utilization	Used	Available
Number of Slice Flip Flops	522	10,944
Number of 4 input LUTs	1,532	10,944
Logic Distribution		
Number of occupied Slices	939	5,472
Number of Slices containing only related logic	939	939
Number of Slices	0	939

```
Process "Generate Post-Place & Route Static Timing" completed successfully
Started : "Launching ISE Text Editor to edit dve_ccir_top.v".
```

Project navigator



Hierarchy	Resources	Preserve
digvidenc		
xc4vfx12-12sf363		
dve_ccir_top (dve_c...	---	routing
DATAPATH - dv...	1,281	inherit(routing)
prescaler - dve_...		
CHROMA_FIR -...		

Sources Window

USING THE PROCESSES WINDOW:

The Process tab shows the available processes in a hierarchical view. You can collapse and expand the levels by clicking the plus (+) or minus (-) icons. Processes are arranged in the order of a typical design flow: project creation, design entry, constraints management, synthesis, implementation, and programming file creation.

PROCESS TYPES:

The following types of processes are available as you work on your design:

- Tasks

When you run a task process, the ISE software runs in "batch mode," that is, the software processes your source file but does not open any additional software tools in the Workspace. Output from the processes appears in the Transcript window.

- Reports

Most tasks include report sub-processes, which generate a summary or status report, for example, the Synthesis Report or Map Report. When you run a report process, the report appears in the Workspace.

- Tools

When you run a tools process, the related tool launches in standalone mode or appears in the Workspace where you can view or modify your design source files.

PROCESS STATUS:

As you work on your design, you may make changes that require some or all of the processes to be rerun. For example, if you edit a source file, it may require that the. Synthesis process and all subsequent process be rerun. Project Navigator keeps track of the changes you make and shows the status of each process with the following status icons:

- Running

This icon shows that the process is running.

- Up-to-date

This icon shows that the process ran successfully with no errors or warnings and does not need to be rerun. If the icon is next to a report process, the report is up-to-date; however, associated tasks may have warnings or errors. If this occurs, you can read the report to determine the cause of the warnings or errors.

- Warnings reported ⚠

This icon shows that the process ran successfully but that warnings were encountered.

- Errors reported ⊗

This icon shows that the process ran but encountered an error.

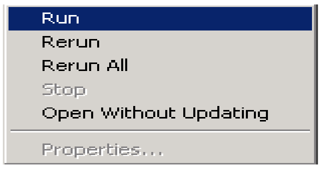
- Out-of-Date ?

This icon shows that you made design changes, which require that the process be rerun. If this icon is next to a report process, you can rerun the associated task process to create an up-to-date version of the report.

No icon-If there is no icon, this show that the process was never run.

RUNNING PROCESSES:

Right-click while positioned over the process, and select **Run** from the popup menu, as shown in the following fig .

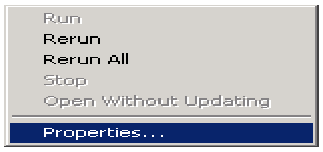


Running processes

- Select the process, and then click the Run toolbar button .
- To run the Implement Design process and all preceding processes on the top module for the design, select **Process>Implement Top Module**, or click the Implement Top Module toolbar button .

SETTING PROCESS PROPERTIES:

Most processes have a set of properties associated with them. Properties control specific options, which correspond to command line options. When properties are available for a process, you can right-click while positioned over the process and select **Properties** from the popup menu, as shown in the following fig.



Setting process properties

When you select Properties, a Process Properties dialog box appears, with standard properties that you can set. The Process Properties dialog box differs depending on the process you select.

USING THE WORKSPACE:

To open a file or view in a standalone window outside of the Project Navigator Workspace, use the Float toolbar button. To dock a floating window, use the Dock toolbar button.

Float

Dock

USING THE TRANSCRIPT WINDOW:

The Console tab of the Transcript window shows output messages from the processes you run. When the following icons appear next to a message, you can right-click the message and select **Goto Answer Record** to open the Xilinx website and show any related Answer Records. If a line number appears as part of the message, you can right-click the message and select **Goto Source** to open the source file with the appropriate line number highlighted.

Warning ⚠

Error ⊗

USING THE TOOLBARS:

Toolbars provide convenient access to frequently used commands. Click once on a toolbar button to execute a command. To see a short popup description of a toolbar button, hold the mouse pointer over the button for about two seconds. A longer description appears in the status bar at the bottom of the main window.

CREATING A PROJECT:

Project Navigator allows you to manage your FPGA and CPLD designs using an ISE™ project, which contains all the files related to your design. First, you must create a project and then add source files. With your project open in Project Navigator, you can view and run processes on all the files in your design. Project Navigator provides a wizard to help you create a new project, as follows.

TO CREATE A PROJECT:

1. Select File > New Project.
2. In the New Project Wizard Create New Project page, do the following:In the Project Name field, enter a name for the project. Follow the naming conventions described in File Naming Conventions.

HDL: Select this option if your top-level design file is a VHDL, Verilog, or ABEL (for CPLDs) file. An HDL Project can include lower-level modules of different file types, such as other HDL files, schematics, and "black boxes," such as IP cores and EDIF files.

- **Schematic:** Select this option if your top-level design file is a schematic file. A schematic project can include lower-level modules of different file types, such as HDL files, other schematics, and "black boxes," such as IP cores and EDIF files
- **EDIF:** Select this option if you converted your design to this file type, for example, using a synthesis tool. Using this file type allows you to skip the Project Navigator synthesis process and to start with the implementation processes.

NGC/NGO: Select this option if you converted your design to this file type, for example, using a synthesis tool. Using this file type allows you to skip the Project Navigator synthesis process and start with the implementation processes.

3. Click Next.
4. If you are creating an HDL or schematic project, skip to the next step. If you are creating an EDIF or NGC/NGO project, do the following in the Import EDIF/NGC Project page:

a) In the Input Design field, enter the name of the input design file, or browse to the file and select it.
b) Select Copy the input design to the project directory to copy your file to the project directory. If you do not select this option, your file is accessed from the remote location.
c) In the Constraint File field, enter the name of the constraints file, or browse to the file and select it.
d) Select Copy the constraints file to the project directo**ry** to copy your file to the project directory. If you do not select this option, your file is accessed from the remote location.
e) Click Next.

In the Device Properties page, set the following options. These settings affect other project options, such as the types of processes that are available for your design.

- Product Category
- Family
- Device
- Package
- Speed
- Top-Level Source Type
- Synthesis Tool

Select one of the following synthesis tools and the HDL language for your project. VHDL/Verilog is a mixed language flow. If you plan to run behavioral simulation, your simulator must support multiple language simulation.

 XST (Xilinx® Synthesis Technology)

XST is available with ISE Foundation™ software installations. It supports projects that include schematic design files and projects that include mixed language source files, such as VHDL and Verilog sources files in the same project. Synplify and Synplify Pro (Synplicity®, Inc.)The Synplify and Synplify Pro software do *not* support projects that include schematic design files.

The Precision® software supports projects that include schematic design files and projects that include mixed language source files, such as VHDL and Verilog sources files in the same project.

 Simulator

Select one of the following simulators and the HDL language for simulation.

 ISE Simulator (Xilinx®, Inc.)

This simulator allows you to run integrated simulation processes as part of your ISE design flow. For more information, see the ISE Simulator Help.

 ModelSim (Mentor Graphics®, Inc.)

You can run integrated simulation processes as part of your ISE design flow using any of the following Modelsim® editions: ModelSim Xilinx Edition (MXE), ModelSim MXE Starter, ModelSim PE, or ModelSim SE™. For more information on ModelSim, including the differences between each edition, see Using the ModelSim Simulator.

- NC-Sim (Cadence®, Inc.)

The NC-Sim simulator is not integrated with ISE and must be run standalone. For more information, see the documentation provided with the simulator.

- VCS (Synopsys®, Inc.)

The VCS® simulator is not integrated with ISE and must be run standalone. For more information, see the documentation provided with the simulator.

- Other

Select this option if you do not have ISE Simulator or ModelSim installed or if you want to run simulation outside of Project Navigator. This instructs Project Navigator to disable the integrated simulation processes for your project.

- Preferred Language

Select one of the following to set your preferred language. The Preferred Language project property controls the default setting for process properties that generate HDL output. If the Synthesis Tool and/or Simulator options are set to a single-language tool, the default language for generated HDL output files will be automatically chosen appropriately Verilog

Simulation are set to mixed-language and you want the default language to be Verilog.

- VHDL

Select this option if both Synthesis Tool and Simulation are set to mixed-language and you want the default language to be VHDL.

- N/A

This option will appear if both Synthesis Tool and Simulation are set to a single language.

- Enable Enhanced Design Summary

Select this option to show the number of errorsand warnings for each of the Detailed Reports in the Design Summary.

Enable Message Filtering

Select this option to turn on Message Filtering.

Display Incremental Messages

Select this option to show the number of new messages for the most recent software run in the Design Summary. You must enable this option and then run the software to show the number of new messages.

If you are creating an EDIF or NGC/NGO project, skip to step 8. If you are creating an HDL or schematic project,

Click next, and optionally, add existing source files to your project in the Add Existing Sources page.

Click Next to display the Project Summary page.

Click Finish to create the project.

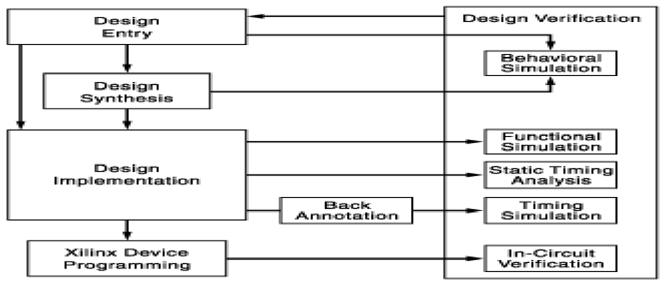
WHAT TO EXPECT:

Project Navigator creates the project file, *project_name*.ise, in the directory you specified. All source files related to the project appear in the Project Navigator Sources tab. Project Navigator manages your project based on the project properties (top-level module type, device type, synthesis tool, and language) you selected when you created the project. It organizes all the parts of your design and keeps track of the processes necessary to move the design from design entry through implementation to programming the targeted Xilinx device.

WHAT TO DO NEXT:

FPGA DESIGN FLOW OVERVIEW:

The ISE™ design flow comprises the following steps: design entry, design synthesis, design implementation, and Xilinx® device programming. Design verification, which includes both functional verification and timing verification, takes places at different points during the design flow.



FPGA design flow overview

FUNCTIONAL VERIFICATION:

You can verify the functionality of your design at different points in the design flow as follows:

- Before synthesis, run behavioral simulation (also known as RTL simulation).
- After Translate, run functional simulation (also known as gate-level simulation), using the SIMPRIM library.
- After device programming, run in-circuit verification.

DESIGN IMPLEMENTATION:

Implement your design as follows:

1. Implement your design, which includes the following steps:
   - Translate
   - Map
   - Place and Route

2. Review reports generated by the Implement Design process, such as the Map Report or Place & Route Report, and change any of the following to improve your design:
   - Process properties
   - Constraints
   - Source files

TIMING VERIFICATION:

You can verify the timing of your design at different points in the design flow as follows:

- Run static timing analysis at the following points in the design flow:
- After Map
- After Place & Route
- Run timing simulation at the following points in the design flow:
- After Map (for a partial timing analysis of CLB and IOB delays)
- After Place and Route (for full timing analysis of block and net delays)

XILINX DEVICE PROGRAMMING:

Program your Xilinx device as follows:

1. Create a programming file (BIT) to program your FPGA.
2. Generate a PROM, ACE, or JTAG file for debugging or to download to your device.
3. Use impact to program the device with a programming cable.

V.HARDWARE DESCRIPTION LANGUAGEDESIGN USING HDL

As a result of the efficiency gains realized using Hardware Description Language; a majority of recent digital circuit design revolves around it. Most designs begin as a set of requirements or a high-level architectural diagram. Control and decision structures are often prototyped in flowchart applications, or entered in a state diagram editor. The process of writing the Hardware Description Language is highly dependent on the nature of the circuit and the designer's preference for coding style. The Hardware Description Language is merely the 'capture language', often beginning with a high-level algorithmic description such as a C++ mathematical model. Designers often use scripting languages such as Perl to automatically generate repetitive circuit structures in the Hardware Description Language. Special text editors offer features for automatic indentation, syntax-dependent coloration, and macro-based expansion of entity/architecture/signal declaration.

The HDL code then undergoes a code review, or auditing. In preparation for synthesis, the HDL description is subject to an array of automated checkers. The checkers report deviations from standardized code guidelines, identify potential ambiguous code constructs before they can cause misinterpretation, and check for common logical coding errors, such as dangling[jargon] ports or shorted outputs. This process aids in resolving errors before the code is synthesized.

WHAT IS VHDL?

VHDL is a programming language that has been designed and optimized for describing the behavior of digital systems.

VHDL has many features appropriate for describing the behavior of electronic components ranging from simple logic gates to complete microprocessors and custom chips. Features of VHDL allow electrical aspects of circuit behavior (such as rise and fall times of signals, delays through gates, and functional operation) to be precisely described. The resulting VHDL simulation models can then be used as building block sin larger circuits (using schematics, block diagrams or system-level VHDL descriptions) for the idea of simulation.

VHDL is also a general-purpose programming language: just as high-level programming languages allow complex design concepts to be expressed as computer programs, VHDL allows the behavior of complex electronic circuits to be captured into a design system for automatic circuit synthesis or for system simulation. Like Pascal, C++, C and VHDL includes features useful for structured design techniques, and offers a rich set of control and data representation features. Unlike these other programming languages, VHDL provides features allowing concurrent events to be described. This is significant because the hardware described using VHDL is inherently concurrent in its operation.

VHDL Design Units:

VHDL provides five different types of primary constructs called design units

1. Entity declaration: Describes external view of entity, that is nothing but it shows input and output signal names.
2. Architecture body: It contains the internal description of the entity.
3. Configuration declaration: Used to create configuration for an entity.
4. Package declaration: It encapsulates a set of related declarations such as type declarations and sub program declarations.
5. Package body: Package body contains the definitions of subprograms declared in package declaration

.
VHDL Modeling styles:

VHDL support basically three main design styles, those are

DATA FLOW MODEL:

The data flow modeling has several additional concurrent statements allow VHDL to describe a circuit in terms of the flow of data and operations on it within the circuit. This style is called a data flow description or dataflow design. The simple data flow model example is given below library IEEE; use IEEE.STD_LOGIC_1164.ALL; entity HALF_ADDER is port(p: in std_logic; in std_logic; sum: out std_logic; carry : out std_logic);

```
end HALF_ADDER;
architecture Behavioral of HALF_ADDER is begin
sum<= p xor q;
carry<= p and q;
end Behavioral;
```

BEHAVIORAL MODEL:

It is sometimes possible to directly describe a desired logic circuit behavior using a concurrent statement. This is a good thing, as ability to create a behavioral design or behavioral description is one of the key benefits of HDL in general and VHDL in particular. VHDL key behavioral element is the „process‟. A process is a collection of sequential statements that executes in parallel with other concurrent statements and other processes.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity HALF_ADDER is
port(p: in std_logic;
q: in std_logic;
sum: out std_logic;
carry : out std_logic);
end HALF_ADDER;
architecture Behavioral of HALF_ADDER is
begin
process(p,q,sum,carry)
begin
sum<= p xor q;
carry<= p and q;
end process;
end Behavioral;
```

STRUCTURAL MODEL:

A VHDL architecture that uses components often called structural description or structural design. The structural model has two parts, component declaration part and statement declaration part. The structural model used most basic VHDL concurrent statements is component statement.

HOW TO INSTANTIATE COMPONENT:

The example is given below there is a small difference in instantiation of component in Xilinx 9.2 and Xilinx 14.2 versions. Below example shows how to instantiates in Xilinx 9.2.The simple half adder code is given below

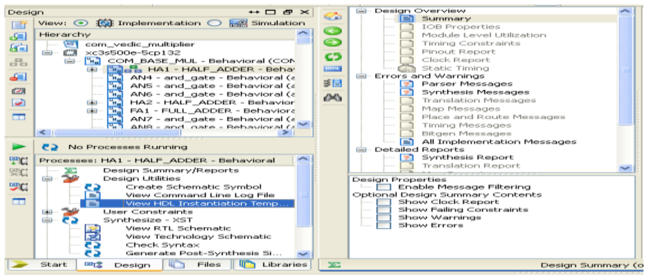
HALF ADDER CODE:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity HALF_ADDER is
Port ( p : in STD_LOGIC;
q : in STD_LOGIC;
r : out STD_LOGIC;
cou : out  STD_LOGIC);
end HALF_ADDER;
architecture Behavioral of HALF_ADDER is
```

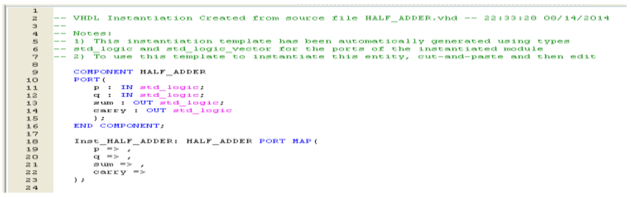
```
component xor2
port( I0,I1:in STD_LOGIC;O: out STD_LOGIC);
end component;
component and2
port (I0,I1:in STD_LOGIC; O:out STD_LOGIC);
end component;
begin
x1: xor2 port map(p,q,r);
c1:and2 port map(p,q,cou);
end Behavioral;
```

The component declaration is must, and after declaration of component by using label we use declared component and port map to assign actual parameters to formal parameters.

Now, how to instantiate a component in Xilinx 14.2 there is instantiation template under design utilities in process tab in Xilinx ISE 14.2 as shown in below fig.



VHDL Instantiation template in Xilinx 14.2



```

-- VHDL Instantiation Created from source file HALF_ADDER.vhd -- 22:33:28 08/14/2014
--
-- Notes:
-- 1) This instantiation template has been automatically generated using types
-- std_logic and std_logic_vector for the ports of the instantiated module
-- 2) To use this template to instantiate this entity, cut-and-paste and then edit

   COMPONENT HALF_ADDER
   PORT(
      p : IN std_logic;
      q : IN std_logic;
      sum : OUT std_logic;
      carry : OUT std_logic
      );
   END COMPONENT;

   Inst_HALF_ADDER: HALF_ADDER PORT MAP(
      p => ,
      q => ,
      sum => ,
      carry =>
   );

```

Instantiation template example.

The Inst_HALF_ADDER is the label name, it may be any name. we have to assign actual parameter right at right hand side of arrow symbol in our program use work.HALF_ADDER port map(parameters).

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity FULL_ADDER is
Port ( a : in STD_LOGIC;
b : in STD_LOGIC;
c : in STD_LOGIC;
sum : out STD_LOGIC;
carry : out  STD_LOGIC);
end FULL_ADDER;
architecture Behavioral of FULL_ADDER is
signal s1,s2,s3:std_logic;
begin
HALF_ADDER1:entitywork.HALF_ADDER
PORT MAP(
p => a,
q => b,
sum => s1,
carry => s2
);
HALF_ADDER2:entitywork.HALF_ADDER
PORT MAP(
p => s1,
q => c,
sum => sum,
carry => s3
);
or_gate1:entitywork.or_gate
PORT MAP(
a => s3,

b => s2,
c => carry
);
end Behavioral;
```

we have to write code previously for the component. That code is present at work, we use that by giving work.name, but in Xilinx 9.2 we have to declare component in declaration part. This is structural model. We can also represent the above example in Verilog code

```
module half_adder (in_x, in_y, out_sum, out_carry);
input  in_x;
input  in_y;
output out_sum;
output out_carry;
assign out_sum = in_x^in_y;
assign out_carry = in_x&in_y;
endmodule
```

VI. SPARTAN 3-FPG

Spartan-3 FPGA

The Spartan-3 family builds on the success of the earlier Spartan-IIE family by increasing the amount of logic resources, the capacity of internal RAM, the total number of I/Os, and the overall level of performance as well as by improving clock management functions. Numerous enhancements derive from the Vertex®-II platform technology. These Spartan-3 FPGA enhancements, combined with advanced process technology, deliver more functionality and bandwidth per dollar than was previously possible, setting new standards in the programmable logic industry. Because of their exceptionally low cost, Spartan-3 FPGAs are ideally suited to a wide range of consumer electronics applications, including broadband access, home networking, display/projection and digital television equipment. The Spartan-3 family is a superior alternative to mask programmed ASICs. FPGAs avoid the high initial cost, the lengthy development cycles, and the inherent inflexibility of conventional ASICs. Also, FPGA programmability permits design upgrades in the field with no hardware replacement necessary, an impossibility with ASICs.

ARCHITECTURAL OVERVIEW:

The Spartan-3 family architecture consists of five fundamental programmable functional elements:

- Configurable Logic Blocks (CLBs) contain RAM-based Look-Up Tables (LUTs) to implement logic and storage elements that can be used as flip-flops or latches. CLBs can be programmed to perform a wide variety of logical functions as well as to store data.
- Input/Output Blocks (IOBs) control the flow of data between the I/O pins and the internal logic of the device. Each IOB supports bidirectional data flow plus 3-state operation. Twenty-six different signal standards, including eight high-performance differential standards. Double Data-Rate(DDR) registers are included. The Digitally Controlled Impedance (DCI) feature provides automatic on-chip terminations, simplifying board designs.
- Block RAM provides data storage in the form of 18-Kbit dual-port blocks.
- Multiplier blocks accept two 18-bit binary numbers as inputs and calculate the product.
- Digital Clock Manager (DCM) blocks provide self-calibrating, fully digital solutions for distributing, delaying, multiplying, dividing, and phase shifting clock signals.

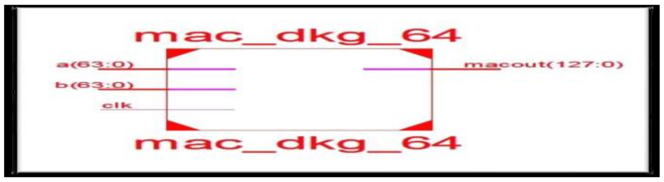
CONFIGURATION:

Spartan-3 FPGAs are programmed by loading configuration data into robust, reprogrammable, static CMOS configuration latches (CCLs) that collectively control all functional elements and routing resources. Before powering on the FPGA, configuration data is stored externally in a PROM or some other nonvolatile medium either on or off the board. After applying power, the configuration data is written to the FPGA using any of five different modes: Master Parallel, Slave Parallel, Master Serial, Slave Serial, and Boundary Scan (JTAG). The Master and Slave Parallel modes use an 8-bit-wide Select MAP port. The recommended memory for storing the configuration data is the low-cost Xilinx Platform Flash PROM family, which includes the XCF00S PROMs for serial configuration and the higher density XCF00P PROMs for parallel or serial configuration.

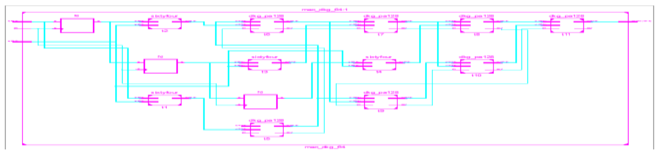
VII. RESULTS

The modified multiplier using Vedic multiplier and kogge stone adder is fast and design of MAC done using Xilinx. This design is implemented in VHDL code using Xilinx. The below figure shows the simulation result of the proposed design. The below fig shows the RTL Schematic of MAC unit.

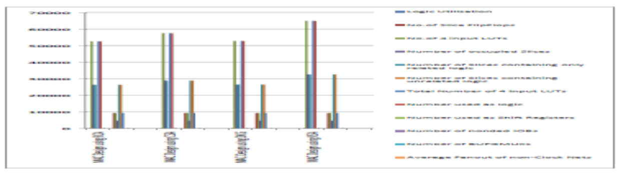
The modified 64 bit multiplier using Vedic multiplier and DKG adder is fast and design of MAC done using Xilinx. This design is implemented in Verilog code using Xilinx. The below figure shows the simulation result of the proposed design. The below figure shows the RTL Schematic of MAC unit.



(a): RTL Schematic of MAC Unit



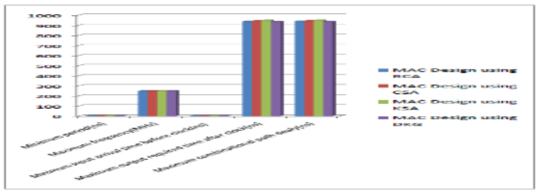
(b): RTL Schematic of MAC Unit



Synthesis report of 64-bit MAC using Vedic Multiplier using RCA,DKG and KSA Reversible logic gates

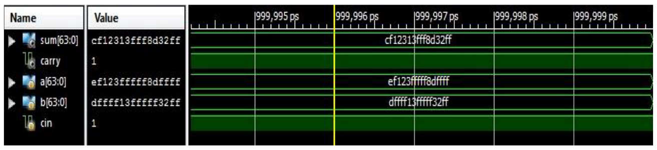
The above fig 9.2 shows comparison between MAC design unit using different Multipliers. The power consumption in MAC design using booth multiplier will be greater than 4 nW and speed is also high. But it will take more area.

The above fig shows the results of Vedic multiplier with different gates and adders. In which DKG Adders has moderate delay. But it consumes very less power and it can be designed in small area.



Delay Analysis report of 64-bit MAC using Vedic Multiplier using RCA,DKG and KSA Reversible logic gates

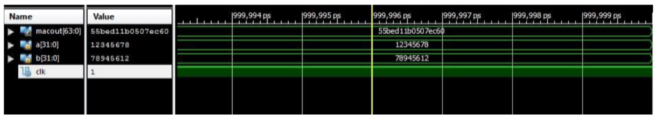
The above fig shows the results of Vedic multiplier with different gates and adders. In which DKG Stone Adders has moderate delay. But it consumes very less power and it can be designed in small area. The above figure 8 shows that simulation result of DKG adder. It is a 64 bit adder. In this design we used two 64 bit adders.



```
Name        Value               999,995 ps  999,996 ps  999,997 ps  999,998 ps  999,999 ps
sum[63:0]   cf12313fff8d32ff    cf12313fff8d32ff
carry       1
a[63:0]     ef123fffff8dffff    ef123fffff8dffff
b[63:0]     dffff13fffff32ff    dffff13fffff32ff
cin         1
```

Simulation result of Adder

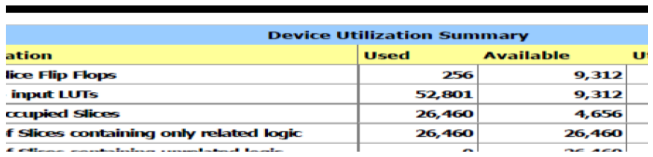
The above fig shows that simulation result of DKG adder. It is a 64 bit adder. In this design we used two 64 bit adders. The above figure 9 shows simulation result of 64 bit MAC design unit. For this we applied two inputs. Which values are a=12345678 and b=78945612 and it will give result of 55bed11b057ec60.



```
Name            Value              999,994 ps  999,995 ps  999,996 ps  999,997 ps  999,998 ps  999,999 ps
macout[63:0]    55bed11b0507ec60   55bed11b0507ec60
a[31:0]         12345678           12345678
b[31:0]         78945612           78945612
clk             1
```

Simulation result of 64 bit MAC unit

The above fig shows simulation result of 32 bit MAC design unit. For this we applied two inputs. Which values are a=305419896 and b=305419896 and it will give result of 932813128726508160.



Device Utilization Summary			
ation	Used	Available	U
lice Flip Flops	256	9,312	
input LUTs	52,801	9,312	
ccupied Slices	26,460	4,656	
f Slices containing only related logic	26,460	26,460	

Design utilization summary

VIII. CONCLUSION AND FUTURE SCOPE

CONCLUSION:

The results obtained by the proposed design of Vedic multiplier with 32 bits and reversible logic are quite good. The work presented is based on 32 bit MAC unit with Vedic Multipliers. MAC unit basic building blocks were designed and its performance has been analyzed for all the blocks. Therefore, we can say that the Urdhva Triyagbhyam sutra with 32-bit Multiplier and reversible logic is the best in all aspects like delay, speed, area and complexity as compared to all other architectures which are shown in table 3.1 Many researchers are reconfiguring the structure of MAC unit, which is the basic block in different designs and aspects especially using reversible logic which develops in recent days. Spectrum Analysis and Correlation linear filtering which are the applications of transform algorithm further add to the field of communication, signal and image processing and instrumentation.

FUTURE SCOPE:

By Combining the Vedic and reversible logic will lead to new and efficient achievements in developing various fields of mathematics, science as well engineering. Future work is to implement the designs using Synopsys IC Compiler to analyze the post layout results for area and delay. Synopsys Prime Time can be used to analyze the multipliers for their power consumption.

REFERENCES

[1] R.NareshNaik , P.Siva Nagendra Reddy and K. Madan Mohan "Design of Vedic Multiplier for Digital Signal Processing Applications" in International Journal ofEngineering Trends and Technology, volume 4 issue 7- 2013 ISSN: 2231-5381(IJETT).

[2] VaijyanathKunchigi ,LinganagoudaKulkarni, SubhashKulkarni 32-bit MAC unit design using Vedic multiplier International Journal of Scientific and Research Publications, Volume3, Issue 2, February 2013

[3] Ramalatha, M.Dayalan, K D Dharani, P Priya, and S Deoborah, High Speed Energy Efficient ALU design using Vedic multiplication techniques, International Conference on Advances in Computational Tools for Engineering Applications, 2009. ACTEA "09.pp. 600 -3, Jul 15-17, 2009.

[4] SreeNivas A and Kayalvizhi N. Article: Implementation of Power Efficient Vedic multiplier. International Journal of Computer Applications 43(16):21-24, April 2012. Published by Foundation of Computer Science, New York, USA

[5] Abdelgawad, MagdyBayoumi ," High Speed and Area-Efficient Multiply Accumulate (MAC) Unit for Digital Signal Processing Applications", IEEE Int.Symp. Circuits Syst. (2007) 3199–3202.

[6] R.Bhaskar, GanapathiHegde, P.R.Vaya," An efficient hardware model for RSA Encryption system using Vedic mathematics",International Conference on Communication Technology and System Design 2011 Procedia Engineering 30 (2012) 124 – 128.

[7] FatemehKashfi, S. Mehdi Fakhraie, Saeed Safari," Designing an ultra-high-speed multiply-accumulate structure", Microelectronics Journal 39 (2008) 1476–1484.