

# **Design and Hardware realization of a 16 Bit Vedic Arithmetic Unit**

**Thesis Report**

*Submitted in partial fulfillment of the requirements for the award of the degree of*

**Master of Technology  
in  
VLSI Design and CAD**

**Submitted By**

Amandeep Singh  
Roll No. 600861019

**Under the supervision of**

**Mr. Arun K Chatterjee  
Assistant Professor, ECED**



**Department of Electronics and Communication Engineering**

**THAPAR UNIVERSITY  
PATIALA(PUNJAB) – 147004**

June 2010

## DECLARATION

---

I hereby declare that the work which is being presented in the thesis entitled, "**Design and Hardware Realization of a 16bit Vedic Arithmetic Unit**" in partial fulfillment of the requirement for the award of degree of M.Tech (VLSI Design & CAD) at Electronics and Communication Department of Thapar University, Patiala, is an authentic record of my own work carried out under the supervision of **Mr. Arun Kumar Chatterjee**, Assistant Professor, ECED.

The matter presented in this thesis has not been submitted in any other University/Institute for the award of my degree.

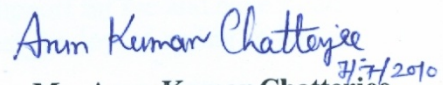
Date: *July 7, 2010*



**Amandeep Singh**

**Roll.No. 600861019**


It is certified that the above statement made by the student is correct to the best of my knowledge and belief.



**Mr. Arun Kumar Chatterjee**

Assistant Professor  
ECED, Thapar University  
Patiala -147004

Countersigned by:



**Dr. A. K. Chatterjee**  
Professor & Head  
Electronics and Communication Engg. Department  
Thapar University, Patiala – 147004



**Dr. R. K. Sharma**  
Dean Academics Affairs  
Thapar University  
Patiala - 147004

## ACKNOWLEDGEMENT

---

Without a direction to lead to and a light to guide your path, we are left in mist of uncertainty. Its only with the guidance, support of a torch bearer and the will to reach there, that we reach our destination.

Above all I thank, Almighty God for giving me opportunity and strength for this work and showing me a ray of light whenever I felt gloomy and ran out of thoughts.

I take this opportunity to express deep appreciation and gratitude to my revered guide, **Mr.Arun K.Chatterjee**, Assistant Professor, ECED, Thapar University, Patiala who has the attitude and the substance of a genius. He continually and convincingly conveyed a spirit of adventure and freeness in regard to research and an excitement in regard to teaching. Without his guidance and persistent help this work would not have been possible.

I would like to express gratitude to **Dr. A.K Chatterjee**, Head of Department, ECED, Thapar University, for providing me with this opportunity and for his great help and cooperation.

I express my heartfelt gratitude towards **Ms. Alpana Agarwal, Assistant Professor & PG coordinator, ECED** for her valuable support.

I would like to thank all faculty and staff members who were there, when I needed their help and cooperation.

My greatest thanks and regards to all who wished me success, especially my family , who have been a constant source of motivation and moral support for me and have stood by me, whenever I was in hour of need, they made me feel the joy of spring in harsh times and made me believe in myself.

And finally, a heartfelt thanks to all my friends who have been providing me moral support, bright ideas and moments of joy throughout the work. I wish that future workers in this area, find this work helpful for them.

**Amandeep Singh**

## ABSTRACT

---

This work is devoted for the design and FPGA implementation of a 16bit Arithmetic module, which uses Vedic Mathematics algorithms.

For arithmetic multiplication various Vedic multiplication techniques like Urdhva Tiryakbhyam, Nikhilam and Anurupye has been thoroughly analysed. Also Karatsuba algorithm for multiplication has been discussed. It has been found that Urdhva Tiryakbhyam Sutra is most efficient Sutra (Algorithm), giving minimum delay for multiplication of all types of numbers.

Using Urdhva Tiryakbhyam, a 16x16 bit Multiplier has been designed and using this Multiplier, a Multiply Accumulate (MAC) unit has been designed. Then, an Arithmetic module has been designed which employs these Vedic multiplier and MAC units for its operation. Logic verification of these modules has been done by using Modelsim 6.5.

Further, the whole design of Arithmetic module has been realised on Xilinx Spartan 3E FPGA kit and the output has been displayed on LCD of the kit. The synthesis results show that the computation time for calculating the product of 16x16 bits is 10.148 ns, while for the MAC operation is 11.151 ns. The maximum combinational delay for the Arithmetic module is 15.749 ns.

# CONTENTS

---

DECLARATION	ii
ACKNOWLEDGEMENT	iii
ABSTRACT	iv
CONTENTS	v
LIST OF FIGURES	viii
TERMINOLOGY	x

## CHAPTER 1 INTRODUCTION.....1-4

- 1.1 Objective.....3
- 1.2 Thesis Organization.....3
- 1.3 Tools Used.....4

## CHAPTER 2 BASIC CONCEPTS.....5-19

- 2.1 Early Indian Mathematics.....5
- 2.2 History of Vedic Mathematics.....5
- 2.3 Vedic Multiplication.....7
  - 2.3.1 *Urdhva Tiryakbhyam Sutra*.....7
  - 2.3.2 *Nikhilam Sutra*.....10
  - 2.3.3 *Square Algorithm*.....11
  - 2.3.4 *Cube Algorithm*.....13
- 2.4 Karatsuba Multiplication.....14
- 2.5 Performance .....16
  - 2.5.1 *Speed*.....16
  - 2.5.2 *Area*.....17
- 2.6 Multiply Accumulate Unit.....18
- 2.7 Arithmetic Module.....18

## CHAPTER 3 DESIGN IMPLEMENTATION.....20-27

- 3.1 Vedic Multiplier.....20
  - 3.1.1 *2x2 bit Multiplier*.....20
  - 3.1.2 *4x4 bit Multiplier*.....22
  - 3.1.3 *8x8 bit Multiplier*.....23

- 3.1.4 16x16 bit Multiplier.....24
- 3.2 16bit MAC Unit using 16x16 Vedic Multiplier.....25
- 3.3 Adder.....26
- 3.4 Arithmetic Module.....27

**CHAPTER 4      SYNTHESIS & FPGA IMPLEMENTATION.....28-39**

- 4.1 2x2 Multiply Block.....28
  - 4.1.1 Description.....28
  - 4.1.2 Device utilization summary.....29
- 4.2 4x4 Multiply Block.....30
  - 4.2.1 Description.....30
  - 4.2.2 Device utilization summary.....31
- 4.3 8x8 Multiply Block.....31
  - 4.3.1 Description.....31
  - 4.3.2 Device utilization summary.....32
- 4.4 16x16 Multiply Block.....33
  - 4.4.1 Description.....33
  - 4.4.2 Device utilization summary.....34
- 4.5 16bit MAC unit.....34
  - 4.5.1 Description.....34
  - 4.5.2 Device utilization summary.....35
- 4.6 16bit Arithmetic Unit.....36
  - 4.6.1 Description.....36
  - 4.6.2 Device utilization summary.....37
- 4.7 LCD Interfacing.....38

**CHAPTER 5      RESULTS & CONCLUSION.....40-48**

- 5.1 Results.....40
  - 5.1.1 Simulation of 16x16 bit Multiplier.....40.
  - 5.1.2 Simulation of 16bit MAC Unit.....42
  - 5.1.3 Simulation of 16bit Arithmetic Unit.....43
- 5.2 Conclusion.....48
- 5.3 Future work.....48

<b>Appendix A.....</b>	<b>49</b>
<b>Appendix B.....</b>	<b>51</b>
<b>Appendix C.....</b>	<b>54</b>
<b>References.....</b>	<b>55</b>

## LIST OF FIGURES

---

Fig 2.1	Example of Early Multiplication Techniques	5
Fig 2.2	Multiplication of 2 digit decimal numbers using Urdhva Tiryakbhyam Sutra	8
Fig 2.3	Using Urdhva Tiryakbhyam for binary Numbers	9
Fig 2.4	Better Implementation of Urdhva Tiryakbhyam For binary numbers	10
Fig 2.5	Multiplication using Nikhilam Sutra	11
Fig 2.6	Karatsuba Algorithm for 2bit Binary numbers	15
Fig 2.7	Comparison of Vedic and Karatsuba Multiplicatoin	16
Fig 2.8	Basic MAC unit	18
Fig 2.9	Basic Block diagram of Arithmetic Unit	19
Fig 3.1	2x2 Multiply block	21
Fig 3.2	Hardware realization of 2x2 block	21
Fig 3.3	4x4 Multiply block	22
Fig 3.4	Addition of partial products in 4x4 block	22
Fig 3.5	Block diagram of 8x8 Multiply block	23
Fig 3.6	Addition of partial products in 8x8 block	23
Fig 3.7	Block diagram of 16x16 Multiply block	24
Fig 3.8	Addition of partial products in 16x16 block	24
Fig 3.9	Block diagram of 16 bit MAC unit	25
Fig 3.10	Block diagram of Arithmetic module	27
Fig 4.1	Black box view of 2x2 block	28
Fig 4.2	RTL view of 2x2 block	29
Fig 4.3	Black box view of 4x4 block	30
Fig 4.4	RTL view of 4x4 block	30
Fig 4.5	Black box view of 8x8 block	31
Fig 4.6	RTL view of 8x8 block	32
Fig 4.7	Black box view of 16x16 block	33
Fig 4.8	RTL view of 16x16 block	33
Fig 4.9	Black box view of 16 bit MAC unit	34

Fig 4.10	RTL view of 16 bit MAC unit	35
Fig 4.11	Black box view of 16 bit Arithmetic Unit	36
Fig 4.12	RTL view of 16 bit Arithmetic Unit	37
Fig 4.13	Black box view of LCD interfacing	38
Fig 5.1	Simulation waveform of 16x16 multiplier	40
Fig 5.2	Simulation waveform of 16 bit MAC unit	42
Fig 5.3	Simulation waveform of MAC operation from 16 bit Vedic Arithmetic module	44
Fig 5.4	Simulation waveform of Multiply operation from 16 bit Vedic Arithmetic module	44
Fig 5.5	Simulation waveform of Subtraction operation from 16 bit Vedic Arithmetic module	45
Fig 5.6	Simulation waveform of Addition operation from 16 bit Vedic Arithmetic module	45
Fig 5.7	LCD output for Addition operation Of Arithmetic module	46
Fig 5.8	LCD output for Subtraction operation Of Arithmetic module	46
Fig 5.9	LCD output for Multiplication operation Of Arithmetic module	46
Fig 5.10	LCD output for MAC operation Of Arithmetic module during 1 <sup>st</sup> ,2 <sup>nd</sup> ,3 <sup>rd</sup> and 4 <sup>th</sup> clock Cycles	47
Fig A1	Another Hardware realization of 2x2 multiply block	50
Fig A2	Xilinx FPGA design flow	51
Fig A3	LCD Character Set	54

# TERMINOLOGY

---

<b>DSP</b>	Digital signal Processing
<b>XST</b>	Xilinx synthesis technology
<b>FPGA</b>	Field programming gate array
<b>DFT</b>	Design for test
<b>DFT</b>	Discrete Fourier transforms
<b>MAC</b>	Multiply and Accumulate
<b>FFT</b>	Fast Fourier transforms
<b>IFFT</b>	Inverse Fast Fourier transforms
<b>CIAF</b>	Computation Intensive arithmetic functions
<b>IC</b>	Integrated Circuits
<b>ROM</b>	Read only memory
<b>PLA</b>	Programmable logic Arrays
<b>NGC</b>	Native generic circuit
<b>NGD</b>	Native generic database
<b>NCD</b>	Native circuit description
<b>UCF</b>	User constraints file
<b>CLB</b>	Combinational logic blocks
<b>IOB</b>	Input output blocks
<b>PAR</b>	Place and Route
<b>ISE</b>	Integrated software environment
<b>IOP</b>	Input output pins
<b>CPLD</b>	Complex programmable logic device
<b>RTL</b>	Register transfer level
<b>JTAG</b>	Joint test action group
<b>RAM</b>	Random access memory
<b>FDR</b>	Fix data register
<b>ASIC</b>	Application-specific integrated circuit
<b>EDIF</b>	Electronic Design Interchange Format

## INTRODUCTION

---

Arithmetic is the oldest and most elementary branch of Mathematics. The name Arithmetic comes from the Greek word **ἀριθμός** (arithmos). Arithmetic is used by almost everyone, for tasks ranging from simple day to day work like counting to advanced science and business calculations. As a result, the need for a faster and efficient Arithmetic Unit in computers has been a topic of interest over decades. The work presented in this thesis, makes use of Vedic Mathematics and goes step by step, by first designing a Vedic Multiplier, then a Multiply Accumulate Unit and then finally an Arithmetic module which uses this multiplier and MAC unit. The four basic operations in elementary arithmetic are addition, subtraction, multiplication and division. Multiplication, basically is the mathematical operation of scaling one number by another. Talking about today's engineering world, multiplication based operations are some of the frequently used Functions, currently implemented in many Digital Signal Processing (DSP) applications such as Convolution, Fast Fourier Transform, filtering and in Arithmetic Logic Unit (ALU) of Microprocessors. Since multiplication is such a frequently used operation, it's necessary for a multiplier to be fast and power efficient and so, development of a fast and low power multiplier has been a subject of interest over decades.

Multiply Accumulate or MAC operation is also a commonly used operation in various Digital Signal Processing Applications. Now, not only Digital Signal Processors, but also general-purpose Microprocessors come with a dedicated Multiply Accumulate Unit or MAC unit. When talking about the MAC unit, the role of Multiplier is very significant because it lies in the data path of the MAC unit and its operation must be fast and efficient. A MAC unit consists of a multiplier implemented in combinational logic, along with a fast adder and accumulator register, which stores the result on clock.

Minimizing power consumption and delay for digital systems involves optimization at all levels of the design. This optimization means choosing the optimum Algorithm for the situation, this being the highest level of design, then the circuit style,

the topology and finally the technology used to implement the digital circuits. Depending upon the arrangement of the components, there are different types of multipliers available. A particular multiplier architecture is chosen based on the application.

Methods of multiplication have been documented in the Egyptian, Greek, Babylonian, Indus Valley and Chinese civilizations.[1] In early days of Computers, multiplication was implemented generally with a sequence of addition, subtraction and shift operations. There exist many algorithms proposed in literature to perform multiplication, each offering different advantages and having trade off in terms of delay, circuit complexity, area occupied on chip and power consumption.

For multiplication algorithms performing in DSP applications, latency and throughput are two major concerns from delay perspective. Latency is the real delay of computing a function. Simply it's a measure of how long the inputs to a device are stable is the final result available on outputs. Throughput is the measure of how many multiplications can be performed in a given period of time. Multiplier is not only a high delay block but also a major source of power dissipation. So, if one aims to minimize power consumption, it is of great interest to reduce the delay by using various optimization methods.

Two most common multiplication algorithms followed in the digital hardware are array multiplication algorithm and Booth multiplication algorithm. The computation time taken by the array multiplier is comparatively less because the partial products are calculated independently in parallel. The delay associated with the array multiplier is the time taken by the signals to propagate through the gates that form the multiplication array. Booth multiplication is another important multiplication algorithm. Large booth arrays are required for high speed multiplication and exponential operations which in turn require large partial sum and partial carry registers. Multiplication of two  $n$ -bit operands using a radix-4 booth recording multiplier requires approximately  $n / (2m)$  clock cycles to generate the least significant half of the final product, where  $m$  is the number of Booth recorder adder stages

First of all, some ancient and basic multiplication algorithms have been discussed to explore Computer Arithmetic from a different point of view. Then some Indian Vedic Mathematics algorithms have been discussed. In general, for a multiplication of a  $n$  bit word with another  $n$  bit word,  $n^2$  multiplications are needed. To challenge this, Karatsuba Algorithm has been discussed which brings the multiplications required, down to  $n^{1.58}$ , for  $n$  bit word. Then “Urdhva tiryakbhyam Sutra” or “Vertically and Crosswise

Algorithm” for multiplication is discussed and then used to develop digital multiplier architecture. This looks quite similar to the popular array multiplier architecture. This Sutra shows how to handle multiplication of a larger number ( $N \times N$ , of  $N$  bits each) by breaking it into smaller numbers of size ( $N/2 = n$ , say) and these smaller numbers can again be broken into smaller numbers ( $n/2$  each) till we reach multiplicand size of  $(2 \times 2)$ . Thus, simplifying the whole multiplication process. The multiplication algorithm is then illustrated to show its computational efficiency by taking an example of reducing a  $N \times N$ -bit multiplication to a  $2 \times 2$ -bit multiplication operation. This work presents a systematic design methodology for fast and area efficient digit multiplier based on Vedic Mathematics and then a MAC unit has been made which uses this multiplier. Finally the Multiplier and MAC unit thus made, have been used in making an Arithmetic module.

## 1.1 OBJECTIVE

The main objective of this work is to implement an Arithmetic unit which makes use of Vedic Mathematics algorithm for multiplication. The Arithmetic unit that has been made, performs multiplication, addition, subtraction and Multiply Accumulate operations. The MAC unit, used in the Arithmetic module uses a fast multiplier, built with Vedic Mathematics Algorithm. Also, square and cube algorithms of Vedic Mathematics, along with Karatsuba Algorithm have been discussed to reduce the multiplications required. Hardware implementation of the Arithmetic unit has been done on Spartan 3E Board.

## 1.2 THESIS ORGANIZATION

The basic concept of multiplication, a historical and simple algorithm for multiplication, to motivate creativity and innovation has been discussed first of all, then the focus has been brought to Vedic Mathematics Algorithms and their functionality. Then, Karatsuba-Ofman Algorithm and finally MAC unit architecture along with Arithmetic module architecture has been discussed in **Chapter 2**.

**Chapter 3** presents a methodology for implementation of different blocks of the Arithmetic module, step by step, from  $2 \times 2$  multiply block to 16 bit Arithmetic module itself.

In **Chapter 4**, realization of Vedic multiplier, MAC unit and Arithmetic module on FPGA kit, in terms of speed and hardware utilization have been discussed.

In **Chapter 5**, the results have been shown and a conclusion has been made by using these results and future scope of the thesis work has been discussed.

### **1.3 TOOLS USED**

**Software used:** Xilinx ISE 9.2 has been used for synthesis and implementation. Modelsim 6.5 has been used for simulation.

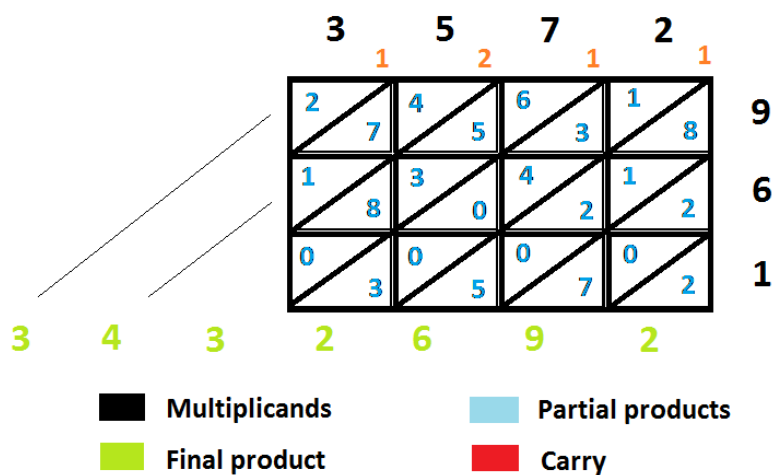
**Hardware used:** Xilinx Spartan3E (Family), XC3S500 (Device), FG320 (Package), -4 (Speed Grade) FPGA devices.

## BASIC CONCEPTS

---

### 2.1 EARLY INDIAN MATHEMATICS

The early Indian mathematicians of the Indus Valley Civilization used a variety of intuitive tricks to perform multiplication. Most calculations were performed on small slate hand tablets, using chalk tables. One technique was of lattice multiplication. Here a table was drawn up with the rows and columns labeled by the multiplicands. Each box of the table is divided diagonally into two, as a triangular lattice. The entries of the table held the partial products, written as decimal numbers. The product could then be formed by summing down the diagonals of the lattice. This is shown in Fig 2.1 below



**Fig 2.1 Example of Early Multiplication Technique**

### 2.2 HISTORY OF 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 gives explanation of several 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. Obviously these formulae are not to be found in present text of Atharva Veda because these formulae were constructed by Swamiji himself. Vedic mathematics is not only a mathematical wonder but also it is logical. That's why it has such a degree of eminence which cannot be disapproved. Due these phenomenal characteristics, Vedic maths has already crossed the boundaries of India and has become an interesting topic of research abroad. Vedic maths deals with several basic as well as complex mathematical operations. Especially, methods of basic arithmetic are extremely simple and powerful [2, 3].

The word "Vedic" is derived from the word "Veda" which means the store-house of all knowledge. Vedic mathematics is mainly based on 16 Sutras (or aphorisms) dealing with various branches of mathematics like arithmetic, algebra, geometry etc. These Sutras along with their brief meanings are enlisted below alphabetically.

- 1) (Anurupyeh) Shunyamanyat – If one is in ratio, the other is zero.
- 2) Chalana-Kalanabyham – Differences and Similarities.
- 3) Ekadhikina Purvena – By one more than the previous One.
- 4) Ekanyunena Purvena – By one less than the previous one.
- 5) Gunakasamuchyah – The factors of the sum is equal to the sum of the factors.
- 6) Gunitasamuchyah – The product of the sum is equal to the sum of the product.
- 7) Nikhilam Navatashcaramam Dashatah – All from 9 and last from 10.
- 8) Paraavartya Yojayet – Transpose and adjust.
- 9) Puranapuranyam – By the completion or noncompletion.
- 10) Sankalana- vyavakalanabhyam – By addition and by subtraction.
- 11) Shesanyankena Charamena – The remainders by the last digit.
- 12) Shunyam Saamyasamuccaye – When the sum is the same that sum is zero.
- 13) Sopaantyadvayamantyam – The ultimate and twice the penultimate.
- 14) Urdhva-tiryagbhyam – Vertically and crosswise.
- 15) Vyashtisamanstih – Part and Whole.
- 16) Yaavadunam – Whatever the extent of its deficiency.

These methods and ideas can be directly applied to trigonometry, plain and spherical geometry, conics, calculus (both differential and integral), and applied mathematics of various kinds. As mentioned earlier, all these Sutras were reconstructed from ancient Vedic texts early in the last century. Many Sub-sutras were also discovered at the same time, which are not discussed here. The beauty of Vedic mathematics lies in the fact that it reduces the otherwise cumbersome-looking calculations in conventional mathematics to a very simple one. This is so because the Vedic formulae are claimed to be based on the natural principles on which the human mind works. This is a very interesting field and presents some effective algorithms which can be applied to various branches of engineering such as computing and digital signal processing [ 1,4].

The multiplier architecture can be generally classified into three categories. First is the serial multiplier which emphasizes on hardware and minimum amount of chip area. Second is parallel multiplier (array and tree) which carries out high speed mathematical operations. But the drawback is the relatively larger chip area consumption. Third is serial- parallel multiplier which serves as a good trade-off between the times consuming serial multiplier and the area consuming parallel multipliers.

## **2.3 VEDIC MULTIPLICATION**

The proposed Vedic multiplier is based on the Vedic multiplication formulae (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. Vedic multiplication based on some algorithms, is discussed below:

### **2.3.1 Urdhva Tiryakbhyam sutra**

The multiplier is based on an algorithm Urdhva Tiryakbhyam (Vertical & Crosswise) of ancient Indian Vedic Mathematics. Urdhva Tiryakbhyam Sutra is a general multiplication formula applicable to all cases of multiplication. It literally means “Vertically and crosswise”. It is based on a novel concept through which the generation of all partial products can be done and then, concurrent addition of these partial products can be done. Thus parallelism in generation of partial products and their summation is obtained using Urdhava Tiryakbhyam. The algorithm can be generalized for  $n \times n$  bit number. Since the partial products and their sums are calculated in parallel, the multiplier is independent of the clock frequency of the processor. Thus the multiplier will require

the same amount of time to calculate the product and hence is independent of the clock frequency. The net advantage is that it reduces the need of microprocessors to operate at increasingly high clock frequencies. While a higher clock frequency generally results in increased processing power, its disadvantage is that it also increases power dissipation which results in higher device operating temperatures. By adopting the Vedic multiplier, microprocessors designers can easily circumvent these problems to avoid catastrophic device failures. The processing power of multiplier can easily be increased by increasing the input and output data bus widths since it has a quite a regular structure. Due to its regular structure, it can be easily layout in a silicon chip. The Multiplier has the advantage that as the number of bits increases, gate delay and area increases very slowly as compared to other multipliers. Therefore it is time, space and power efficient. It is demonstrated that this architecture is quite efficient in terms of silicon area/speed [3,5].

### Multiplication of two decimal numbers- 43\*68

To illustrate this multiplication scheme, let us consider the multiplication of two decimal numbers (43\*68). The digits on the both sides of the line are multiplied and added with the carry from the previous step. This generates one digit of result and a carry digit. This carry is added in the next step and hence the process goes on. If more than one line are there in one step, all the results are added to the previous carry. In each step, unit's place digit acts as the result bit while the higher digits act as carry for the next step. Initially the carry is taken to be zero. The working of this algorithm has been illustrated in Fig 2.2.

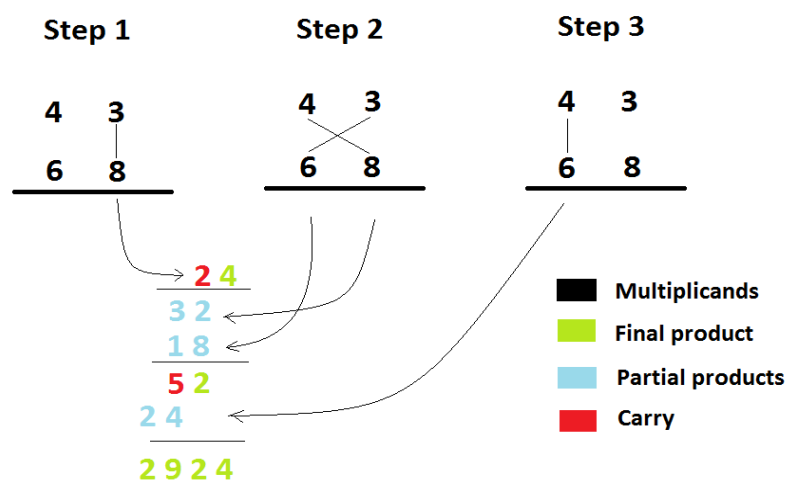
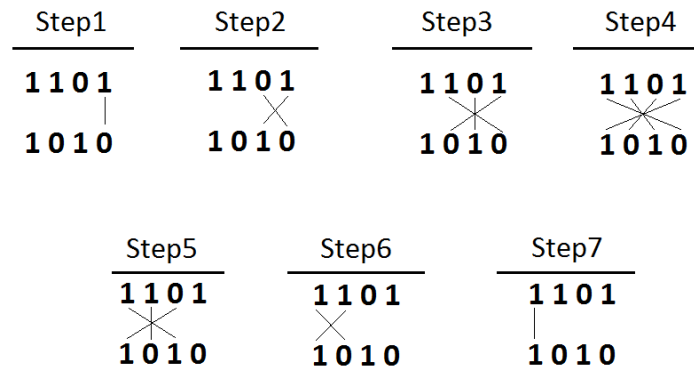


Fig 2.2 Multiplication of 2 digit decimal numbers using Urdhva Tiryakbhyam Sutra

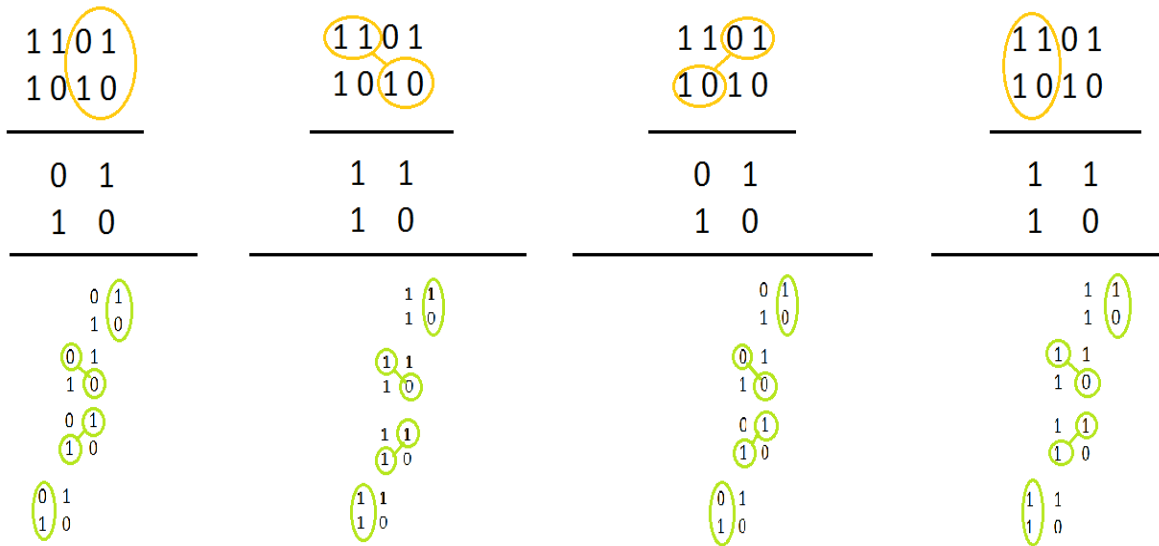
Now we will see how this algorithm can be used for binary numbers. For example (1101 \* 1010) as shown in Fig 2.3.



**Fig 2.3 Using Urdhva Tiryakbham for Binary numbers**

Firstly, least significant bits are multiplied which gives the least significant bit of the product (vertical). Then, the LSB of the multiplicand is multiplied with the next higher bit of the multiplier and added with the product of LSB of multiplier and next higher bit of the multiplicand (crosswise). The sum gives second bit of the product and the carry is added in the output of next stage sum obtained by the crosswise and vertical multiplication and addition of three bits of the two numbers from least significant position. Next, all the four bits are processed with crosswise multiplication and addition to give the sum and carry. The sum is the corresponding bit of the product and the carry is again added to the next stage multiplication and addition of three bits except the LSB. The same operation continues until the multiplication of the two MSBs to give the MSB of the product. For example, if in some intermediate step, we get 110, then 0 will act as result and 11 as the carry. It should be clearly noted that carry may be a multi-bit number.

From here we observe one thing, as the number of bits goes on increasing, the required stages of carry and propagate also increase and get arranged as in ripple carry adder. A more efficient use of Urdhva Tiryakbham is shown in Fig 2.4.

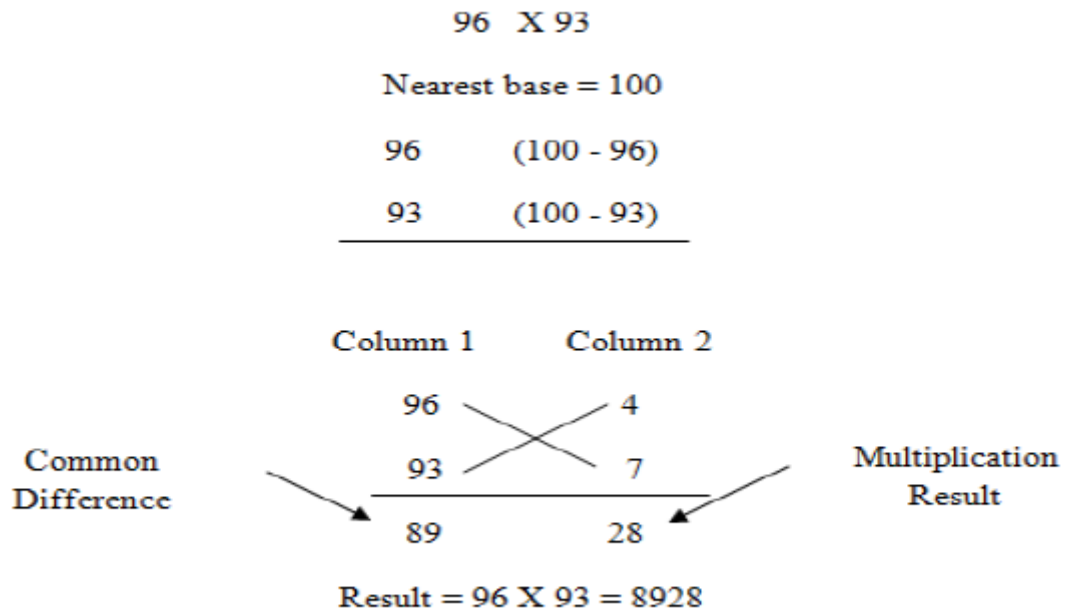


**Fig 2.4 Better Implementation of Urdhva Tiryakbhyam for Binary numbers**

Above, a 4x4 bit multiplication is simplified into 4 , 2x2 bit multiplications that can be performed in parallel. This reduces the number of stages of logic and thus reduces the delay of the multiplier. This example illustrates a better and parallel implementation style of Urdhva Tiryakbhyam Sutra. The beauty of this approach is that larger bit streams ( of say N bits) can be divided into  $(N/2 = n)$  bit length, which can be further divided into  $n/2$  bit streams and this can be continued till we reach bit streams of width 2, and they can be multiplied in parallel, thus providing an increase in speed of operation. [6]

### 2.3.2 Nikhilam Sutra

Nikhilam Sutra literally means “all from 9 and last from 10”. Although it is applicable to all cases of multiplication, it is more efficient when the numbers involved are large. Since it finds out the compliment of the large number from its nearest base to perform the multiplication operation on it, larger is the original number, lesser the complexity of the multiplication. We first illustrate this Sutra by considering the multiplication of two decimal numbers  $(96 * 93)$  in Fig 2.5. where the chosen base is 100 which is nearest to and greater than both these two numbers.



**Fig 2.5 Multiplication Using Nikhilam Sutra [3]**

The right hand side (RHS) of the product can be obtained by simply multiplying the numbers of the Column 2 ( $7 \times 4 = 28$ ). The left hand side (LHS) of the product can be found by cross subtracting the second number of Column 2 from the first number of Column 1 or vice versa, i.e.,  $96 - 7 = 89$  or  $93 - 4 = 89$ . The final result is obtained by concatenating RHS and LHS (Answer = 8928) [3].

### 2.3.3 SQUARE ALGORITHM

In order to calculate the square of a number, we have utilized “Duplex” D property of Urdhva Triyakbhyam. In the Duplex, we take twice the product of the outermost pair and then add twice the product of the next outermost pair and so on till no pairs are left. When there are odd numbers of bits in the original sequence, there is one bit left by itself in the middle and this enters as its square. Thus for 987654321,

$$D = 2 * (9 * 1) + 2 * (8 * 2) + 2 * (7 * 3) + 2 * (6 * 4) + 5 * 5 = 165.$$

Further, the Duplex can be explained as follows

For a 1 bit number D is its square.

For a 2 bit number D is twice their product

For a 3 bit number D is twice the product of the outer pair + square of the middle bit.

For a 4 bit number D is twice the product of the outer pair + twice the product of the inner pair.

The algorithm is explained for 4 x 4 bit number. The Vedic square has all the advantages as it is quite faster and smaller than the array, Booth and Vedic multiplier [5, 7].

**Algorithm for 4 x 4 bit Square Using Urdhva Tiryakbhyam D - Duplex [5, 7]**

X3 X2 X1 X0 Multiplicand

X3 X2 X1 X0 Multiplier

-----

H G F E D C B A

-----

P7 P6 P5 P4 P3 P2 P1 P0 Product

-----

**PARALLEL COMPUTATION**

1.  $D = X_0 * X_0 = A$
2.  $D = 2 * X_1 * X_0 = B$
3.  $D = 2 * X_2 * X_0 + X_1 * X_1 = C$
4.  $D = 2 * X_3 * X_0 + 2 * X_2 * X_1 = D$
5.  $D = 2 * X_3 * X_1 + X_2 * X_2 = E$
6.  $D = 2 * X_3 * X_2 = F$
7.  $D = X_3 * X_3 = G$

### 2.3.4 CUBE ALGORITHM

The cube of the number is based on the Anurupye Sutra of Vedic Mathematics which states “If you start with the cube of the first digit and take the next three numbers (in the top row) in a Geometrical Proportion (in the ratio of the original digits themselves) then you will find that the 4th figure (on the right end) is just the cube of the second digit”.

If a and b are two digits then according to Anurupye Sutra,

$$a^3 \quad a^2b \quad ab^2 \quad b^3$$

$$2a^2b \quad 2ab^2$$

-----

$$a^3 + 3a^2b + 3ab^2 + b^3 = (a + b)^3$$

-----

This sutra has been utilized in this work to find the cube of a number. The number M of N bits having its cube to be calculated is divided in two partitions of N/2 bits, say a and b, and then the Anurupye Sutra is applied to find the cube of the number. In the above algebraic explanation of the Anurupye Sutra, we have seen that  $a^3$  and  $b^3$  are to be calculated in the final computation of  $(a+b)^3$ .

The intermediate  $a^3$  and  $b^3$  can be calculated by recursively applying Anurupye sutra. A few illustrations of working of Anurupye sutra are given below [5].

$$\begin{array}{r}
 (15)^3 = 1 \quad 5 \quad 25 \quad 125 \\
 \phantom{(15)^3 = } 10 \quad 50 \\
 \hline
 \phantom{(15)^3 = } 3 \quad 3 \quad 7 \quad 5
 \end{array}$$

## 2.4 KARATSUBA MULTIPLICATION

Karatsuba Multiplication method offers a way to perform multiplication of large number or bits in fewer operations than the usual brute force technique of long multiplication. As discovered by Karatsuba and Ofman in 1962, multiplication of two  $n$  digit numbers can be done with a bit complexity of less than  $n^2$  using the identities of the form.

$$(a + b.10^n)(c + d.10^n) = a.c + [(a + b)(c + d) - a.c - b.d] 10^n + b.d.10^{2n}$$

By proceeding recursively the big complexity  $O(n^{\lg 3})$ , where  $\lg^3 = 1.58 < 2$ . Now let us take an example and consider multiplication of two numbers with just two digits  $(a_1 a_0)$  and  $(b_1 b_0)$  in base  $w$

$$N_1 = a_0 + a_1.w$$

$$N_2 = b_0 + b_1.w$$

Their product is:

$$\begin{aligned} P &= N_1.N_2 \\ &= a_0.b_0 + (a_0.b_1 + a_1.b_0)w + a_1.b_1.w^2 \\ &= p_0 + p_1.w + p_2.w^2 \end{aligned}$$

Instead of evaluating the individual digits, we can write

$$\begin{aligned} .q_0 &= a_0.b_0 \\ .q_1 &= (a_0 + a_1)(b_0 + b_1) \\ .q_2 &= a_1.b_1 \end{aligned}$$

Here  $q_1$  can be expanded, regrouped and can be written in the form of  $p$  as

$$.q_1 = p_1 + p_0 + p_2$$

Since,  $q_0 = p_0,$

$$p_1 = q_1 - q_0 - q_2$$

$$p_2 = q_2$$

So the three digits of  $p$  have been evaluated using three multiplications rather than four. This technique can be used for binary numbers as well, with a tradeoff that more additions and subtractions are required.[8,9]

Now let's use this technique for binary numbers, say 10(2) and 11(3) as shown in Fig 2.6

$$\begin{array}{r}
 a = 1 \ 0 \\
 b = 1 \ 1 \\
 \hline
 \end{array}$$

$$\begin{array}{l}
 \begin{array}{l} 1 \ 0 \\ 1 \ 1 \end{array} \Rightarrow q_0 = 0 \ (00) \\
 \begin{array}{l} 1+0 \\ 1+1 \end{array} \Rightarrow \begin{array}{l} 1+0 \\ 1+1 \end{array} \times \begin{array}{l} 1+1 \\ 1+1 \end{array} \\
 \qquad \qquad \qquad q_1 = 2 \ (1 \ 0) \\
 \\
 \begin{array}{l} 1 \ 0 \\ 1 \ 1 \end{array} \Rightarrow q_2 = 1 \ (0 \ 1)
 \end{array}$$

**Fig 2.6 Karatsuba Algorithm for 2 bit Binary numbers**

Here base,  $w = 2$ , and  $w^2 = 4$ ,  $a_1 = 1$ ,  $a_0 = 0$ ,  $b_1 = 1$ ,  $b_0 = 0$ .

So the product P will be

$$\begin{aligned}
 & q_0 + [q_1 - q_0 - q_2]w + q_2w^2 \\
 = & 0 + ((2 - 0 - 1) \cdot 2) + 1 \cdot 4 \\
 = & 6, \text{ which is } 0110 \text{ in binary}
 \end{aligned}$$

## 2.5 PERFORMANCE:

### 2.5.1 SPEED:

Vedic multiplier is faster than array multiplier and Booth multiplier. As the number of bits increases from 8x8 bits to 16x16 bits, the timing delay is greatly reduced for Vedic multiplier as compared to other multipliers. Vedic multiplier has the greatest advantage as compared to other multipliers over gate delays and regularity of structures. Delay in Vedic multiplier for 16 x 16 bit number is 32 ns while the delay in Booth and Array multiplier are 37 ns and 43 ns respectively [10]. Thus this multiplier shows the highest speed among conventional multipliers. It has this advantage than others to prefer a best multiplier.

When Karatsuba Algorithm is also studied for reducing the number of multiplications required, still Vedic Algorithm proves itself to be faster as shown in Fig 2.7.

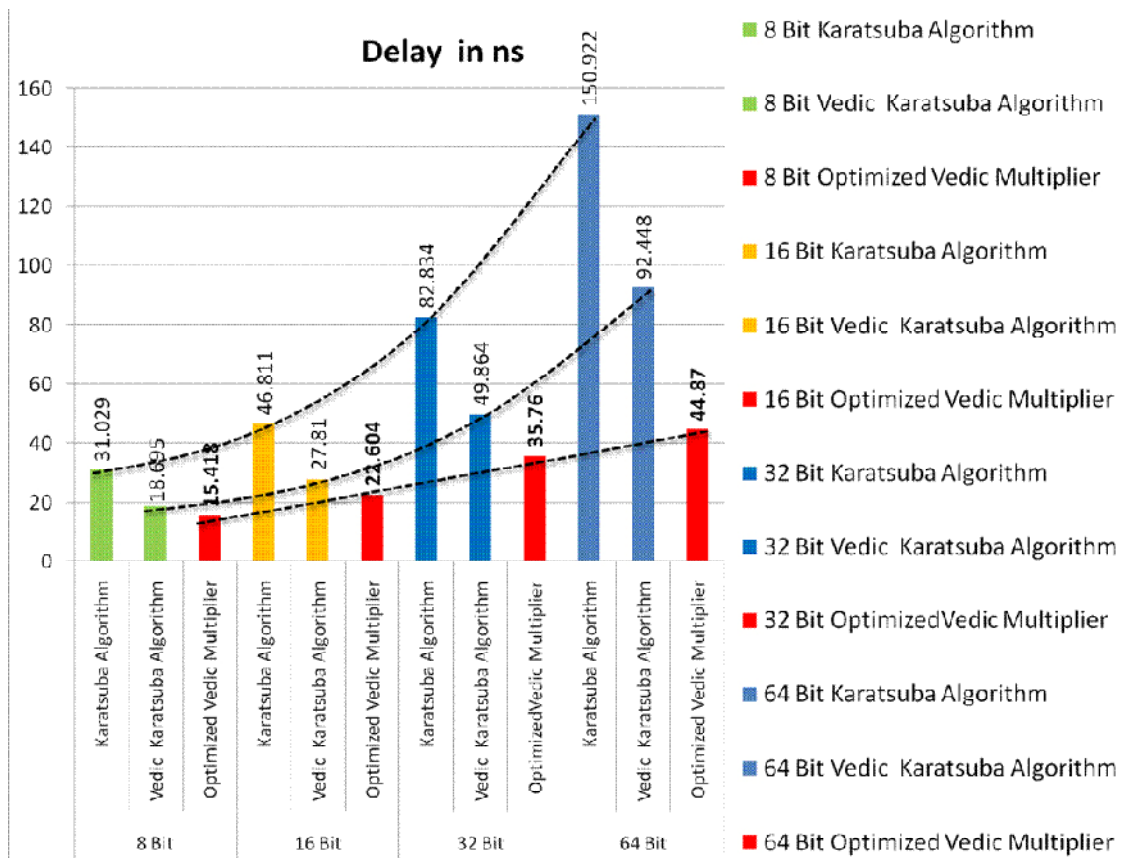


Fig 2.7 Comparison of Vedic and Karatsuba Multiplication [11]

### **2.5.2 AREA:**

The area needed for Vedic square multiplier is very small as compared to other multiplier architectures i.e. the number of devices used in Vedic square multiplier are 259 while Booth and Array Multiplier is 592 and 495 respectively for 16 x 16 bit number when implemented on Spartan FPGA [10].

Thus the result shows that the Vedic square multiplier is smallest and the fastest of the reviewed architectures. The Vedic square and cube architecture proved to exhibit improved efficiency in terms of speed and area compared to Booth and Array Multiplier. Due to its parallel and regular structure, this architecture can be easily realized on silicon and can work at high speed without increasing the clock frequency. It has the advantage that as the number of bits increases, the gate delay and area increase very slowly as compared to the square and cube architectures of other multiplier architecture. Speed improvements are gained by parallelizing the generation of partial products with their concurrent summations.

## 2.6 MULTIPLY ACCUMULATE UNIT (MAC)

Multiply-accumulate operation is one of the basic arithmetic operations extensively used in modern digital signal processing (DSP). Most arithmetic, such as digital filtering, convolution and fast Fourier transform (FFT), requires high-performance multiply accumulate operations. The multiply-accumulator (MAC) unit always lies in the critical path that determines the speed of the overall hardware systems. Therefore, a high-speed MAC that is capable of supporting multiple precisions and parallel operations is highly desirable.

### Basic MAC architecture.

Basically a MAC unit employs a fast multiplier fitted in the data path and the multiplied output of multiplier is fed into a fast adder which is set to zero initially. The result of addition is stored in an accumulator register. The MAC unit should be able to produce output in one clock cycle and the new result of addition is added to the previous one and stored in the accumulator register. Fig 2.8. below shows basic MAC architecture. Here the multiplier that has been used is a Vedic Multiplier built using Urdhva Tiryakbhyam Sutra and has been fitted into the MAC design.

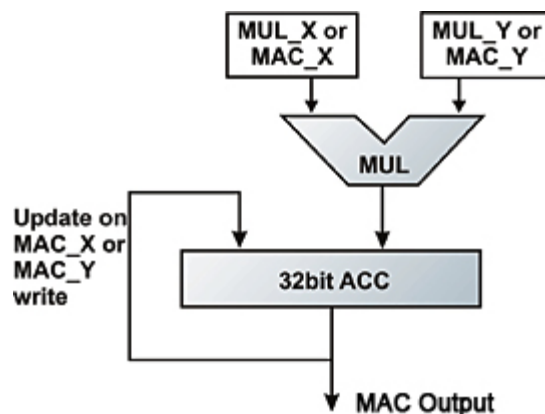
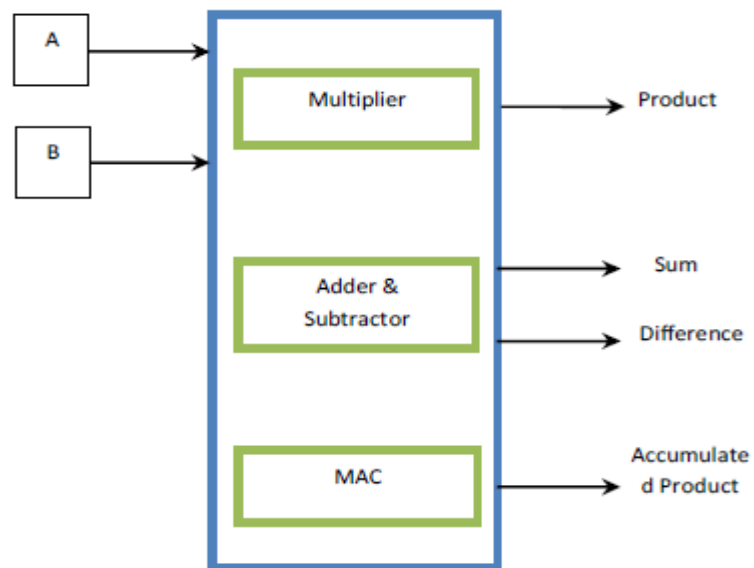


Fig 2.8 Basic MAC unit [11]

## 2.7 ARITHMETIC MODULE

Arithmetic Logic Unit can be considered to be the heart of a CPU, as it handles all the mathematical and logical calculations that are needed to be carried out. Again there may be different modules for handling Arithmetic and Logic functions. In this work, an arithmetic unit has been made using Vedic Mathematics algorithms and performs Multiplication, MAC operation as well as addition and subtraction. For performing addition and subtraction, conventional adder and subtractor have been used. The control signals which tell the arithmetic module, when to perform which operations are provided by the control unit, which is out of the scope of this thesis.



**Fig 2.9 Basic block diagram of Arithmetic unit [11]**

## DESIGN IMPLEMENTATION

---

The proposed Arithmetic Module has first been split into three smaller modules, that is 1. Multiplier 2. MAC unit 3. Arithmetic module, as a whole. These modules have been made using Verilog HDL.

### 3.1 VEDIC MULTIPLIER

The design starts first with Multiplier design, that is 2x2 bit multiplier. Here, “Urdhva Tiryakbhyam Sutra” or “Vertically and Crosswise Algorithm” for multiplication has been effectively used to develop digital multiplier architecture. This algorithm is quite different from the traditional method of multiplication, that is to add and shift the partial products. This Sutra shows how to handle multiplication of a larger number ( $N \times N$ , of  $N$  bits each) by breaking it into smaller numbers of size ( $N/2 = n$ , say) and these smaller numbers can again be broken into smaller numbers ( $n/2$  each) till we reach multiplicand size of  $(2 \times 2)$ . Thus, simplifying the whole multiplication process.

For Multiplier, first the basic blocks, that are the 2x2 bit multipliers have been made and then, using these blocks, 4x4 block has been made and then using this 4x4 block, 8x8 bit block and then finally 16x16 bit Multiplier has been made. The device selected for synthesis is Device family Spartan 3E, device is xc3s500, package fg320 with speed grade -4.

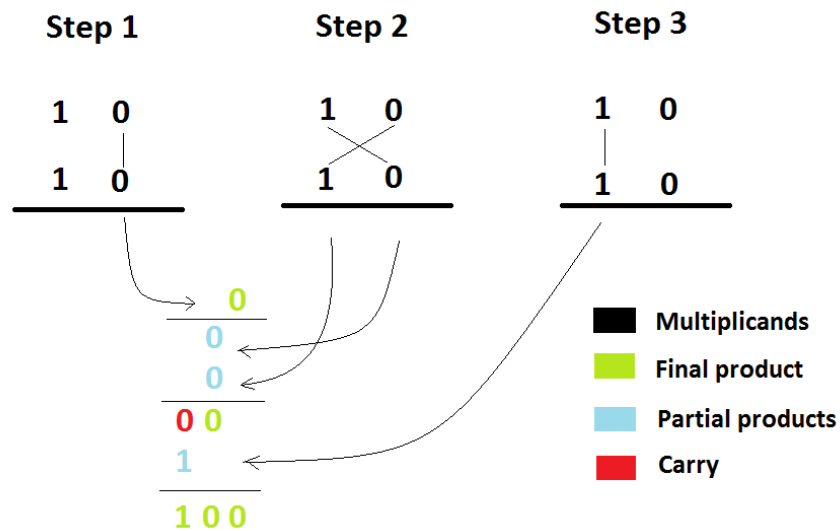
So let's start from the synthesis of a 2x2 bit multiplier.

#### 3.1.1 2x2 bit Multiplier

In 2x2 bit multiplier, the multiplicand has 2 bits each and the result of multiplication is of 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 is given in Appendix A.

By using Urdhva Tiryakbhyam, the multiplication takes place as illustrated in Fig 3.1. Here multiplicands  $a$  and  $b$  are taken to be (10) both. The first step in the multiplication

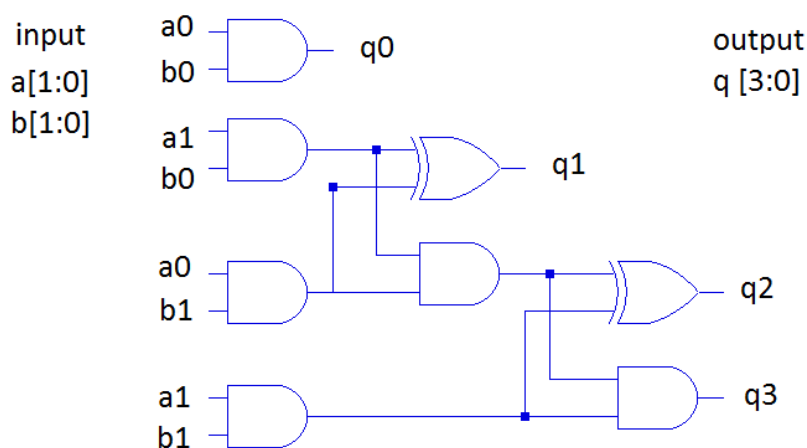
is vertical multiplication of LSB of both multiplicands, then is the second step, that is crosswise multiplication and addition of the partial products . Then Step 3 involves vertical multiplication of MSB of the multiplicands and addition with the carry propagated from Step 2.



**Fig 3.1 2x2 Multiply Block**

### Hardware realization of 2x2 multiplier block

The hardware realization of 2x2 multiplier blocks is illustrated in Fig 3.2. For the sake of simplicity, the usage of clock and registers is not shown, but emphasis has been laid on understanding of the algorithm.

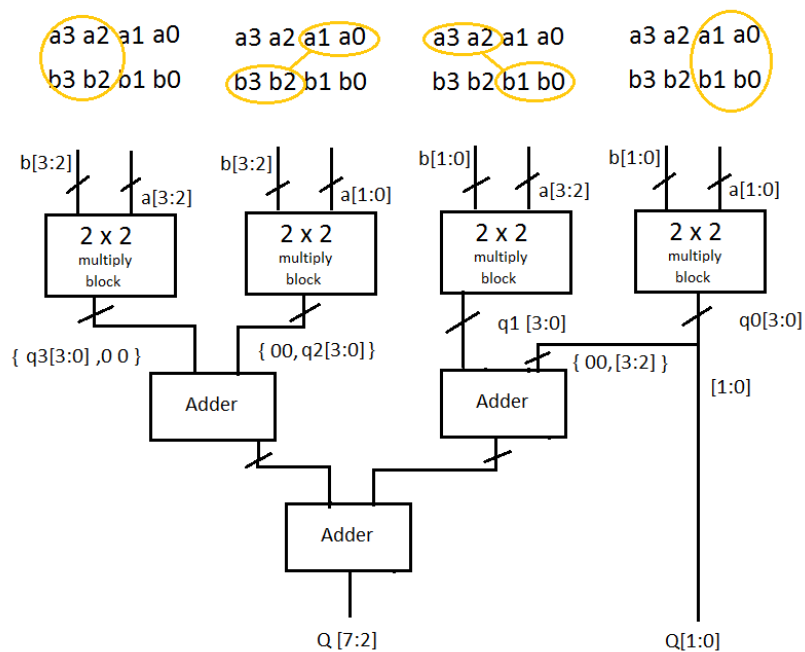


**Fig 3.2 Hardware Realization of 2x2 block**

Now let's move on to the next block in the design, that is 4x4 multiply block.

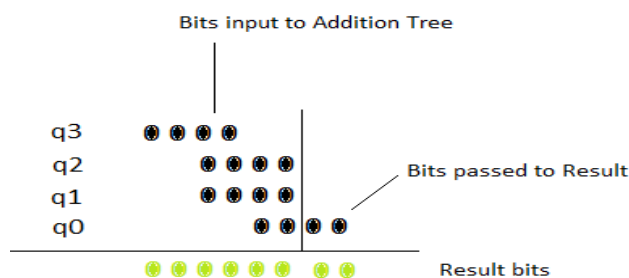
### 3.1.2 4x4 Bit Multiplier

The 4x4 Multiplier is made by using 4, 2x2 multiplier blocks. Here, the multiplicands are of bit size ( $n=4$ ) where as the result is of 8 bit size. The input is broken into smaller chunks of size of  $n/2 = 2$ , for both inputs, that is  $a$  and  $b$ . These newly formed chunks of 2 bits are given as input to 2x2 multiplier block and the result produced 4 bits, which are the output produced from 2x2 multiplier block are sent for addition to an addition tree, as shown in the Fig 3.3.



**Fig 3.3 4x4 Multiply Block**

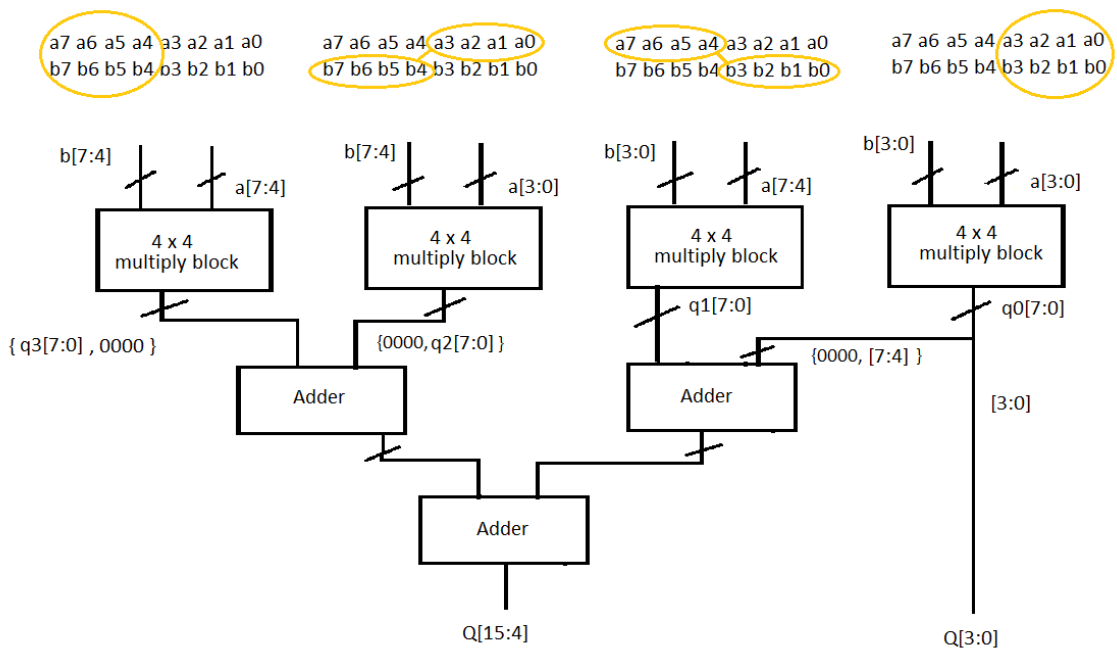
Here, instead of following serial addition, the addition tree has been modified to Wallace tree look alike, thus reducing the levels of addition to 2, instead of 3. Here, two lower bits of  $q_0$  pass directly to output, while the upper bits of  $q_0$  are fed into addition tree. The bits being fed to addition tree can be further illustrated by the diagram in Fig 3.4.



**Fig 3.4 Addition of partial products in 4x4 block**

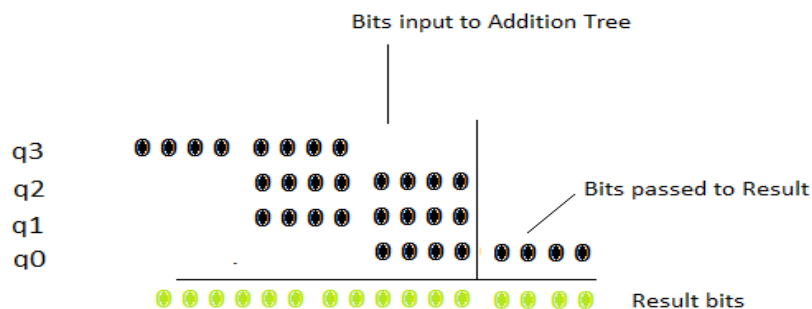
### 3.1.3 8x8 bit Multiplier

The 8x8 Multiplier is made by using 4 , 4x4 multiplier blocks. Here , the multiplicands are of bit size (n=8) where as the result is of 16 bit size. The input is broken into smaller chunks size of  $n/2 = 4$ , for both inputs, that is a and b, just like as in case of 4x4 multiply block. These newly formed chunks of 4 bits are given as input to 4x4 multiplier block, where again these new chunks are broken into even smaller chunks of size  $n/4 = 2$  and fed to 2x2 multiply block. The result produced, from output of 4x4 bit multiply block which is of 8 bits, are sent for addition to an addition tree , as shown in the Fig 3.5 below.



**Fig 3.5 Block Diagram of 8x8 Multiply block**

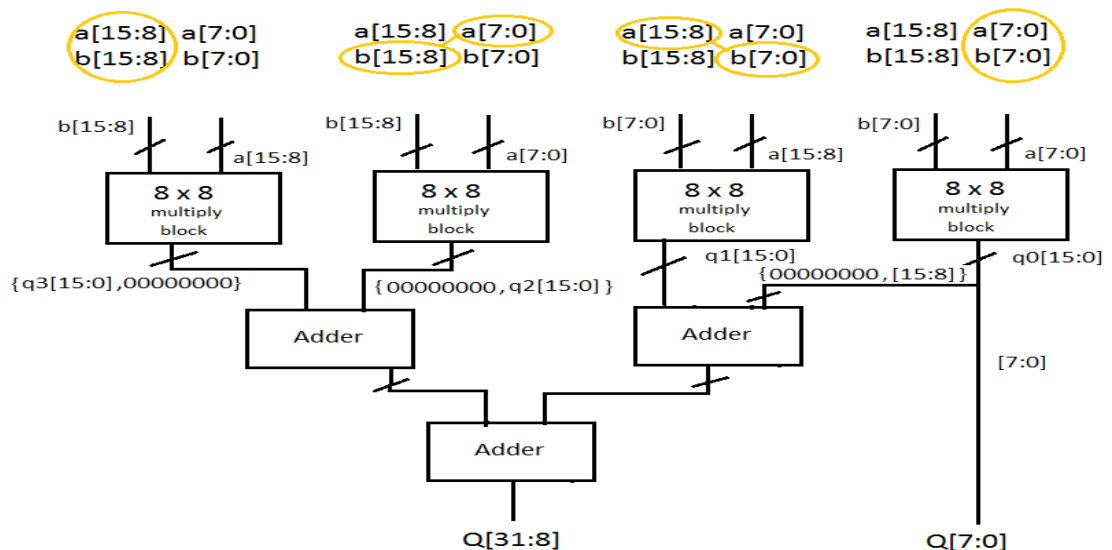
Here, one fact must be kept in mind that, each 4x4 multiply block works as illustrated in Fig 3.3. In 8x8 Multiply block, lower 4 bits of  $q0$  are passed directly to output and the remaining bits are fed for addition into addition tree, as shown in Fig 3.5. The addition of partial products is shown in Fig 3.6.



**Fig 3.6 Addition of Partial products in 8x8 block**

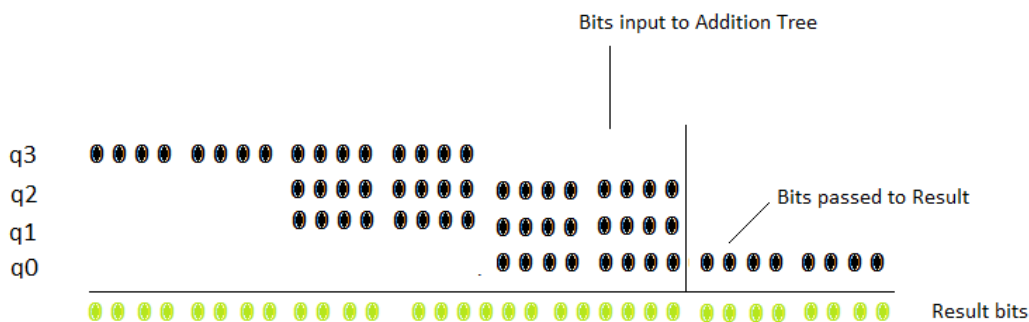
### 3.1.4 16x16 bit Multiplier

The 16x16 Multiplier is made by using 4, 8x8 multiplier blocks. Here, the multiplicands are of bit size ( $n=16$ ) where as the result is of 32 bit size. The input is broken into smaller chunks size of  $n/2 = 8$ , for both inputs, that is  $a$  and  $b$ . These newly formed chunks of 8 bits are given as input to 8x8 multiplier block, where again these new chunks are broken into even smaller chunks of size  $n/4 = 4$  and fed to 4x4 multiply block, just as in case of 8x8 Multiply block. Again, the new chunks are divided in half, to get chunks of size 2, which are fed to 2x2 multiply block. The result produced, from output of 8x8 bit multiply block which is of 16 bits, are sent for addition to an addition tree, as shown in the Fig 3.7.



**Fig 3.7 Block diagram of 16x16 Multiply block**

Here, as shown in Fig 16, the lower 8 bits of  $q_0$  directly pass on to the result, while the higher bits are fed for addition into the addition tree. The addition of partial products is shown in Fig 3.8.



**Fig 3.8 Addition of Partial products in 16x16 block**

### 3.2 16 BIT MAC UNIT USING 16X16 VEDIC MULTIPLIER

The 16 bit MAC unit here, uses a 16x16 vedic multiplier in its data path. As the 16x16 vedic multiplier is faster than other multipliers for example booth multiplier, even the MAC unit made using vedic multiplier is faster than the conventional one.

In the MAC unit, the data inputs, A and B are 16 bits wide and they are stored in two data registers, that is Data a\_reg and Data b\_reg, both being 16 bits wide, Then the inputs are fed into a vedic multiplier, which stores the result “Mult” in a 32 bit wide register named, Multiply\_reg. The contents of Multiply\_reg are continuously fed in to a conventional adder and the result is stored in a 64 bit wide register “Dataout\_reg”.

Here we notice that the MAC unit makes use of two clocks, one for the operation of MAC unit and other one, namely clk2 for the Multiplier. The frequency of clk2 should be 4 times the frequency of MAC unit for proper operation. A clock divider by 4 circuit may be used, in future here, which takes clk2 as the parent clock and produces ‘clk’ as the daughter clock, which is 4 times slower than the parent clock. But with 50% duty cycle. The faster clock ‘clk2’ is used for the multiplier while slower clock ‘clk’ is used for the MAC unit. The data coming as input to MAC unit may vary with clock ‘clk’. The signal “clr” when applied, makes the contents of all the data registers that is Data a\_reg, Data b\_reg, Multipl\_reg and Dataout\_reg to be forced to 0. The “clken” signal is used to enable the MAC operation.

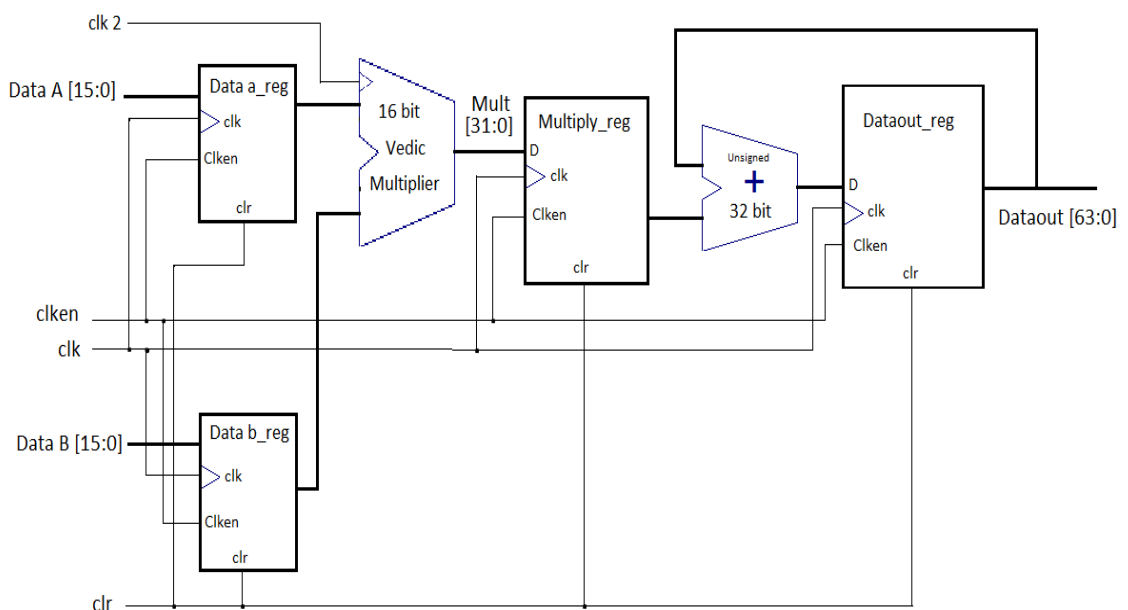


Fig 3.9\_ Block Diagram of 16 bit MAC unit

### 3.3 ADDER

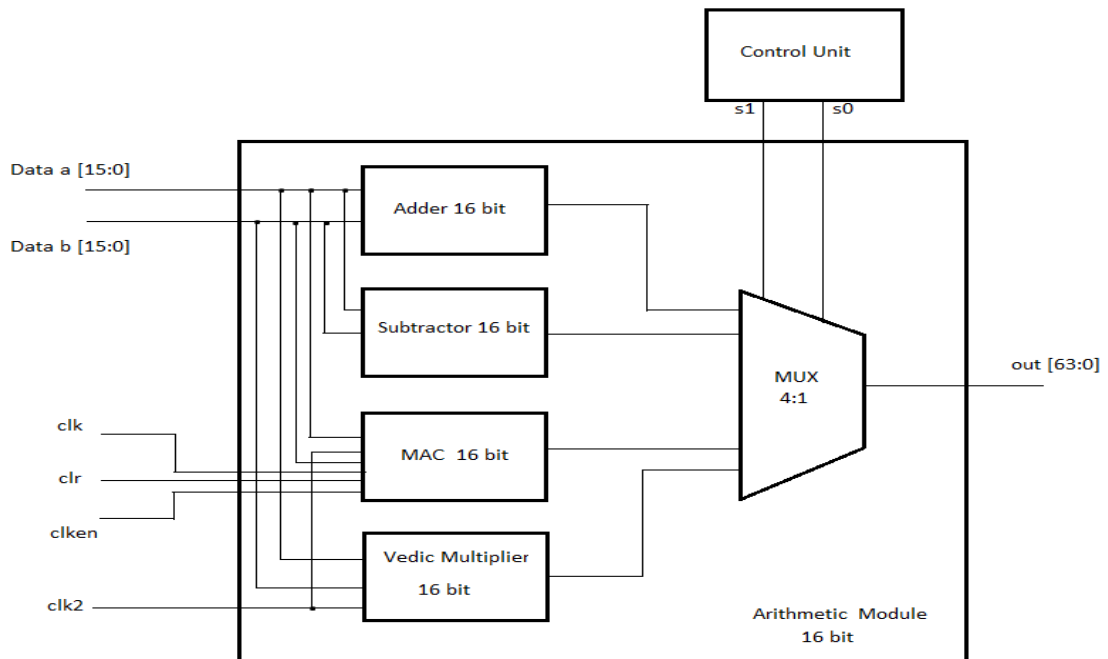
In general, carry ripple adders can be used when they meet timing constraints because they are compact and easy to build. When faster adders are required, carry increment and carry skip architectures can be used, particularly for 8 to 16 bit lengths. Hybrids combining these techniques are also popular. At word lengths of 32 and especially 64 bits, tree adders are distinctly faster.

Good logic synthesis tools automatically map the “+” operator onto an appropriate adder to meet timing constraints while minimizing the area. For example, the Synopsys Designware libraries contain carry-ripple adders, carry select adders, carry lookahead adders and a variety of prefix adders.[12]

### 3.4 ARITHMETIC MODULE

The Arithmetic module designed in this work, makes use of 4 components, that are, Adder, Subtractor, Multiplier and MAC unit. As a result, the Arithmetic unit can perform fixed point addition, subtraction, multiplication and multiply accumulate operations on 16 bit data. Here the inputs are Data a and Data b, which are 16 bits wide. The arithmetic unit uses conventional adder and subtractor, while the multiplier and MAC unit are made using Vedic Mathematics Algorithm. The control signals which guide the Arithmetic unit to perform a particular operation, ie Addition, subtraction, multiplication or MAC operation are s0 and s1, which are provided by the control circuit. Control circuit is beyond the scope of this thesis. The clock used by multiplier, ie “clk2” runs 4 times as fast as the global clock “clk” which is used for controlling MAC operation. The details of MAC operation and Multiplier have already been discussed in previous sections. Now let's have a look at the status of control lines s0 and s1 and the corresponding arithmetic operation being performed.

S1	S0	Operation performed
0	0	Addition
0	1	Subtraction
1	0	Multiplication
1	1	Multiply Accumulate



**Fig 3.10 Block Diagram of Arithmetic module**

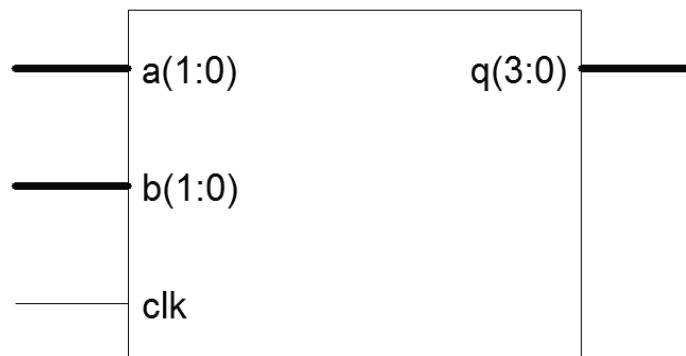
## SYNTHESIS & FPGA IMPLEMENTATION

---

This Chapter deals with the Synthesis and FPGA implementation of the Arithmetic module. The FPGA used is Xilinx **Spartan3E** (Family), **XC3S500** (Device), **FG320** (Package), **-4** (Speed Grade)

Here, the RTL view, its description, the device used and its Hardware utilization summary is given for each module, starting from the most basic component i.e. 2x2 multiply block and then going up to the Arithmetic module level. More information about Xilinx FPGA Design flow is given in Appendix B

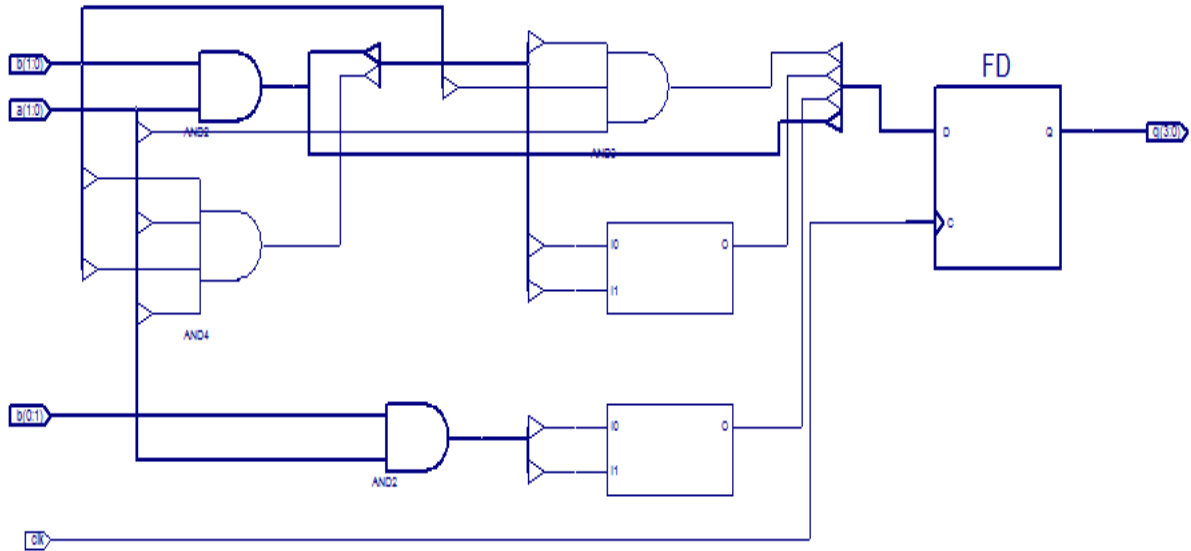
### 4.1 2X2 MULTIPLY BLOCK



**Fig 4.1 Black Box view of 2x2 block**

#### 4.1.1 Description

A	input data 2bit
B	input data 2 bit
Clk	clock
Q	output 4 bit



**Fig 4.2 RTL view of 2x2 block**

**4.1.2 Device utilization summary:**

-----

**Selected Device : 3s500efg320-4**

Number of Slices:	2 out of 4656	0%
Number of Slice Flip Flops:	4 out of 9312	0%
Number of 4 input LUTs:	4 out of 9312	0%
Number of bonded IOBs:	9 out of 232	3%
Number of GCLKs:	1 out of 24	4%

## 4.2 4X4 MULTIPLY BLOCK

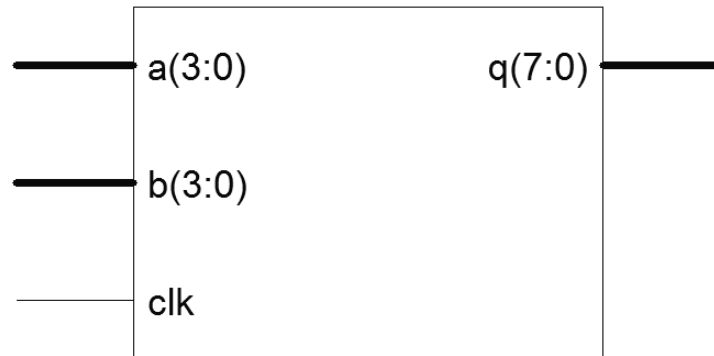


Fig 4.3 Black Box view of 4x4 block

### 4.2.1 Description

A	input data 4bit
B	input data 4 bit
Clk	clock
Q	output 8 bit

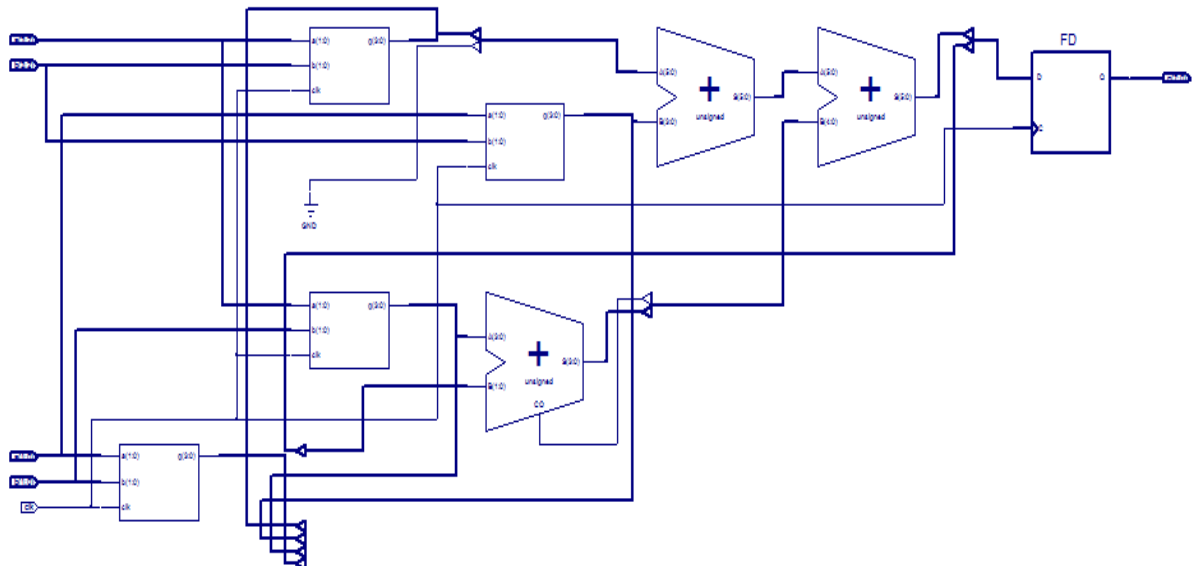


Fig 4.4 RTL view of 4x4 block

## 4.2.2 Device utilization summary:

-----

### Selected Device : 3s500efg320-4

Number of Slices:	21 out of 4656	0%
Number of Slice Flip Flops:	24 out of 9312	0%
Number of 4 input LUTs:	28 out of 9312	0%
Number of bonded IOBs:	17 out of 232	7%
Number of GCLKs:	1 out of 24	4%

Minimum period: 8.196ns (Maximum Frequency: 122.018MHz)

## 4.3 8X8 MULTIPLY BLOCK

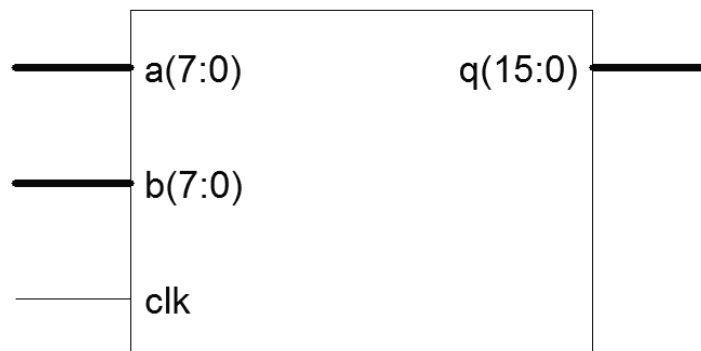
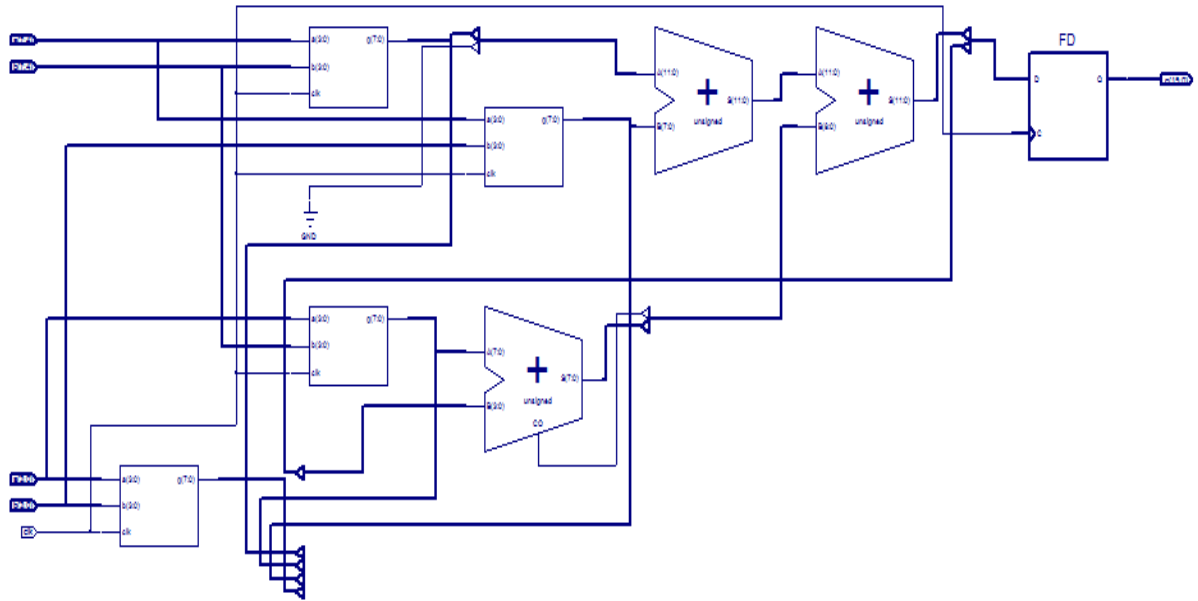


Fig 4.5 Black Box view of 8x8 block

### 4.3.1 Description

A	input data 8bit
B	input data 8 bit
Clk	clock
Q	output 16 bit



**Fig 4.6 RTL view of 8x8 block**

**4.3.2 Device utilization summary:**

-----

**Selected Device : 3s500efg320-4**

Number of Slices:	103 out of 4656	2%
Number of Slice Flip Flops:	112 out of 9312	1%
Number of 4 input LUTs:	138 out of 9312	1%
Number of bonded IOBs:	33 out of 232	14%
Number of GCLKs:	1 out of 24	4%

Minimum period: 8.847ns (Maximum Frequency: 113.039MHz)

#### 4.4 16X16 MULTIPLY BLOCK



Fig 4.7 Black Box view of 16x16 block

##### 4.4.1 Description

- A input data 16bit
- B input data 16 bit
- Clk clock
- Q output 32 bit

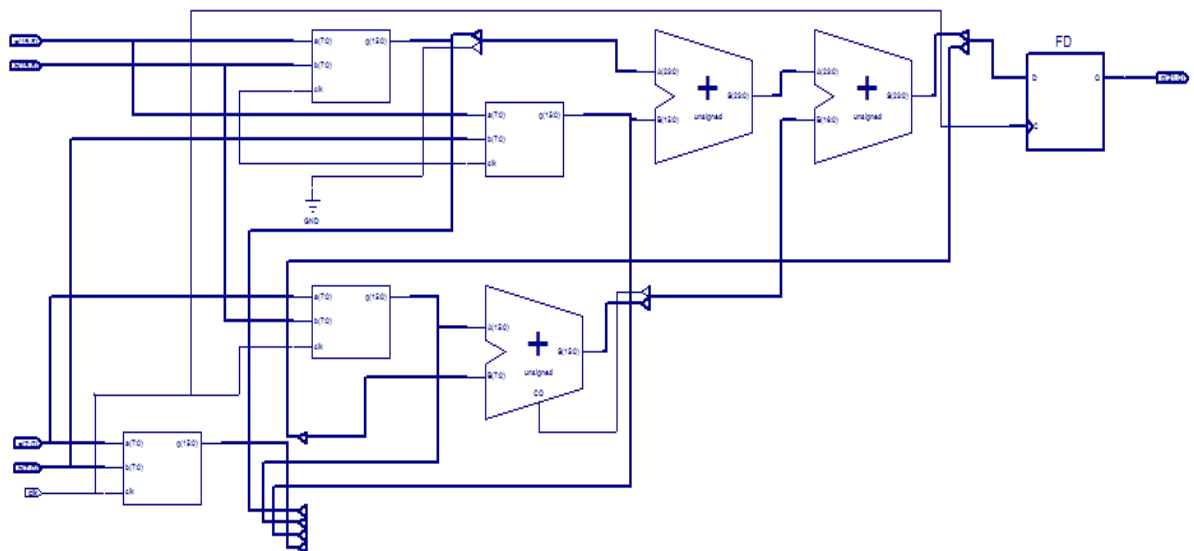


Fig 4.8 RTL view of 16x16 block

#### 4.4.2 Device utilization summary:

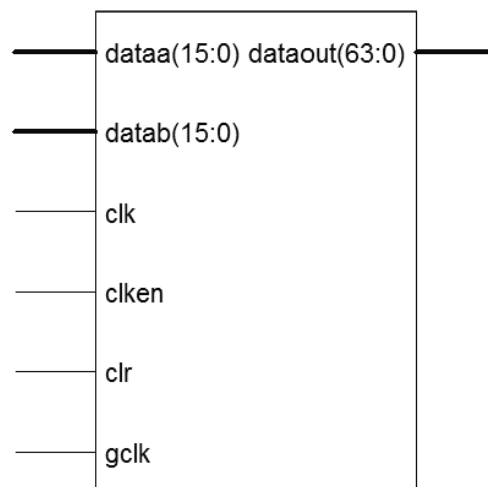
-----

**Selected Device : 3s500efg320-4**

Number of Slices:	396	out of	4656	8%
Number of Slice Flip Flops:	480	out of	9312	5%
Number of 4 input LUTs:	606	out of	9312	6%
Number of bonded IOBs:	65	out of	232	28%
Number of GCLKs:	1	out of	24	4%

Minimum period: 10.148ns (Maximum Frequency: 98.537MHz)

#### 4.5 16 BIT MAC UNIT

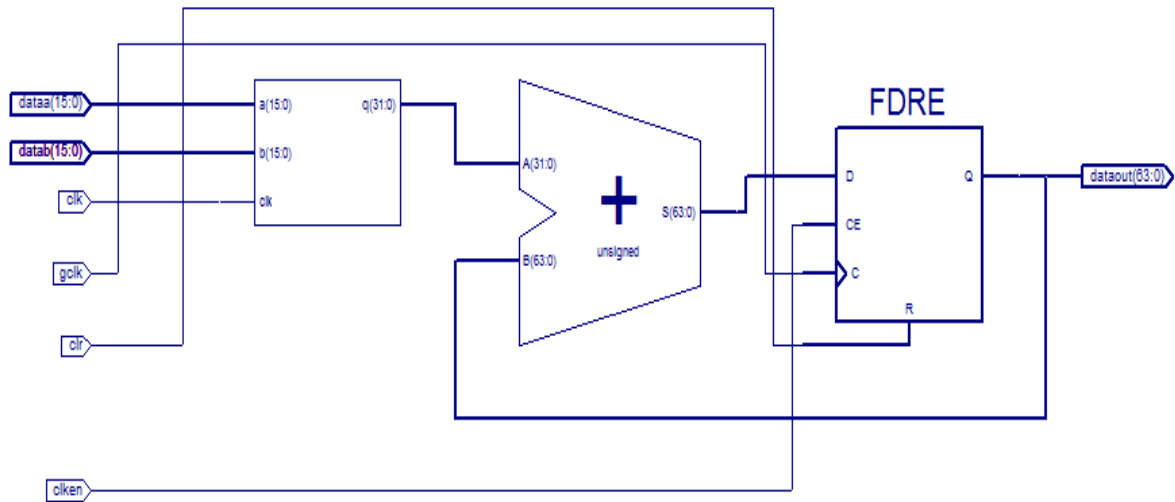


**Fig 4.9 Black Box view of 16 bit MAC unit**

##### 4.5.1 Description

A	input data 16bit
B	input data 16 bit
gclk	global clock on which MAC unit operates
clk	clock frequency of multiplier
clr	reset signal which forces output = 0

clken            enable signal which enables MAC operation, must be 1 to produce  
                   output  
 Dataout        output 64 bit



**Fig 4.10 RTL view of 16bit MAC unit**

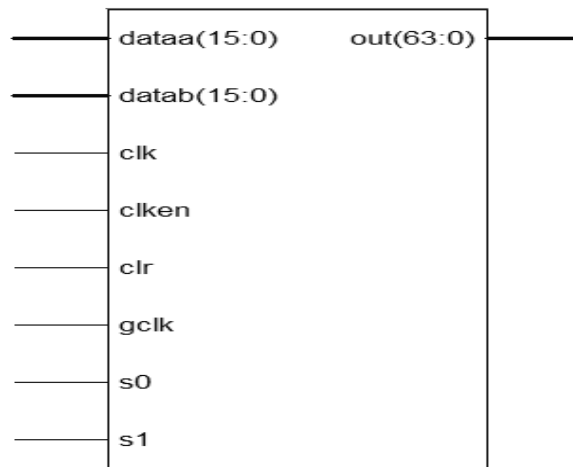
#### 4.5.2 Device utilization summary:

##### Selected Device : 3s500efg320-4

Number of Slices:	428 out of 4656	9%
Number of Slice Flip Flops:	544 out of 9312	5%
Number of 4 input LUTs:	670 out of 9312	7%
Number of bonded IOBs:	100 out of 232	43%
Number of GCLKs:	2 out of 24	8%

Minimum period: **11.151ns** (Maximum Frequency: 89.678MHz)

## 4.6 16 BIT ARITHMETIC UNIT

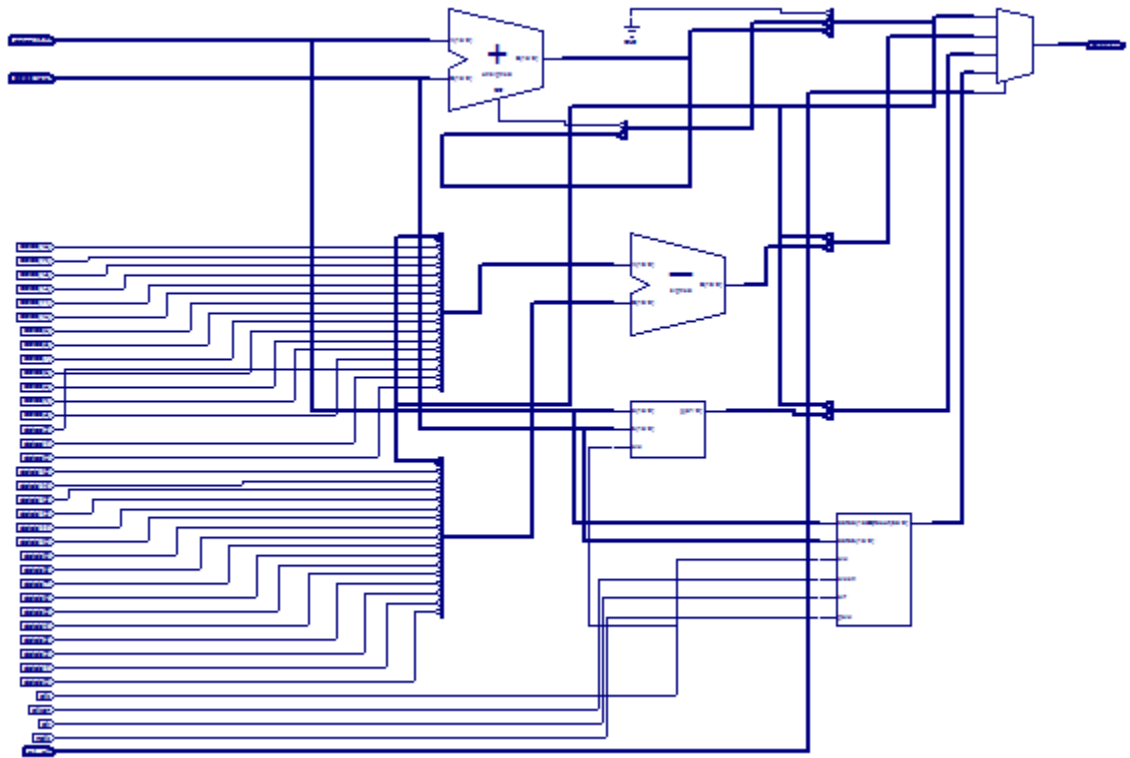


**Fig 4.11 BlackBox view of 16bit Arithmetic Unit**

### 4.6.1 Description

A	input data 16bit
B	input data 16 bit
gclk	global clock on which MAC unit operates
clk	clock frequency of multiplier
clr	reset signal which forces output = 0
clken	enable signal which enables MAC operation, must be 1 to produce output
Dataout	output 64 bit
S0	select line input from control circuit
S1	select line input from control circuit.

S1	S0	Operation performed
0	0	Addition
0	1	Subtraction
1	0	Multiplication
1	1	Multiply Accumulate



**Fig 4.12 RTL view of 16 bit Arithmetic Unit**

**4.6.2 Device utilization summary:**

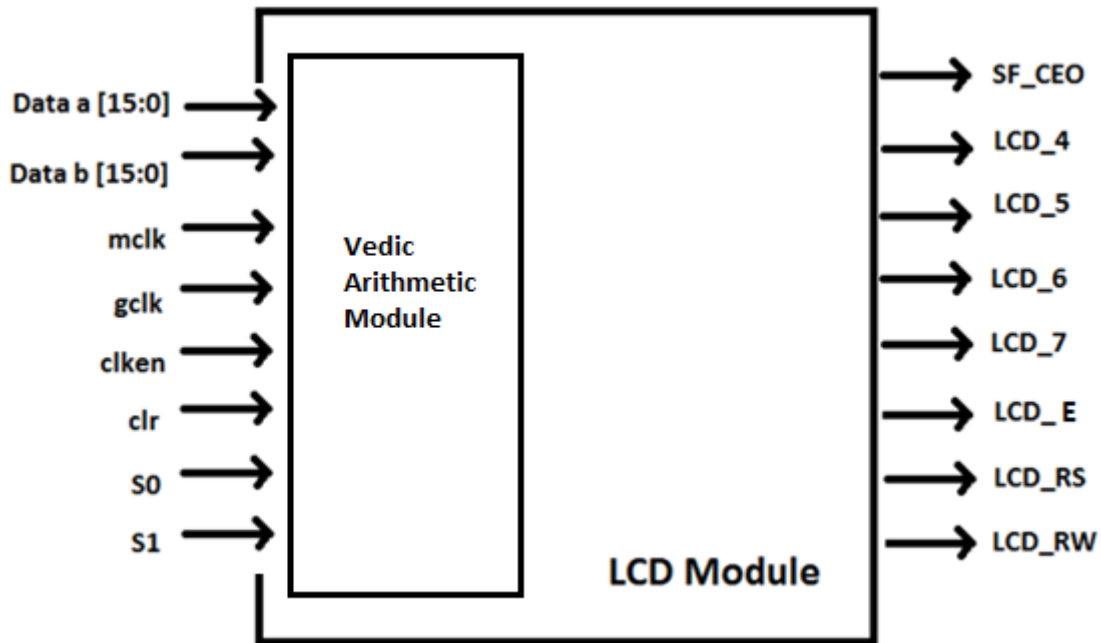
-----

**Selected Device : 3s500efg320-4**

Number of Slices:	690 out of 4656	14%
Number of Slice Flip Flops:	726 out of 9312	7%
Number of 4 input LUTs:	1166 out of 9312	12%
Number of bonded IOBs:	102 out of 232	43%
Number of GCLKs:	2 out of 24	8%

Maximum combinational path delay: **15.749ns**

## 4.7 LCD INTERFACING



**Fig 4.13 Black Box view of LCD Interfacing**

The output of Arithmetic module has been displayed on LCD. For this, a module called LCD module has been made. The Arithmetic module has been made a component inside the LCD module, the inputs of the module being, that of the Arithmetic module, that is, Data\_a, Data\_b, both being 16 bits wide, mclk is the input clock for vedic multiplier, gclk is global clock on which the MAC unit operates, clr is the reset signal which forces output = 0, clken is the enable signal which enables MAC operation and must be 1 to produce output for MAC operation. S0 and S1 are select lines input from control circuit.

The output displayed on the LCD depends on the values of SF\_CEO, LCD\_4, LCD\_5, LCD\_6, LCD\_7, LCD\_E, LCD\_RS, LCD\_RW.

The function and FPGA pin of the pins mentioned above, is given in Table 1.

<b>Signal Name</b>	<b>FPGA pin</b>	<b>Function</b>
SF_CEO	D16	If 1, StrataFlash disabled. Full read/write access to LCD.
LCD_4	R15	Data pins
LCD_5	R16	
LCD_6	P17	
LCD_7	M15	
LCD_E	M18	<b>Read/Write Enable Pulse</b> 0: Disabled 1: Read/Write operation enabled
LCD_RS	L18	<b>Register Select</b> 0: Instruction register during write operations. Busy Flash during read operations 1: Data for read or write operations
LCD_RW	L17	<b>Read/Write Control</b> 0: WRITE, LCD accepts data 1: READ, LCD presents data

**Table 1**

For more information on displaying characters on LCD , please refer to Appendix C

## RESULTS & CONCLUSION

### 5.1 RESULTS

#### 5.1.1 SIMULATION OF 16X16 MULTIPLIER

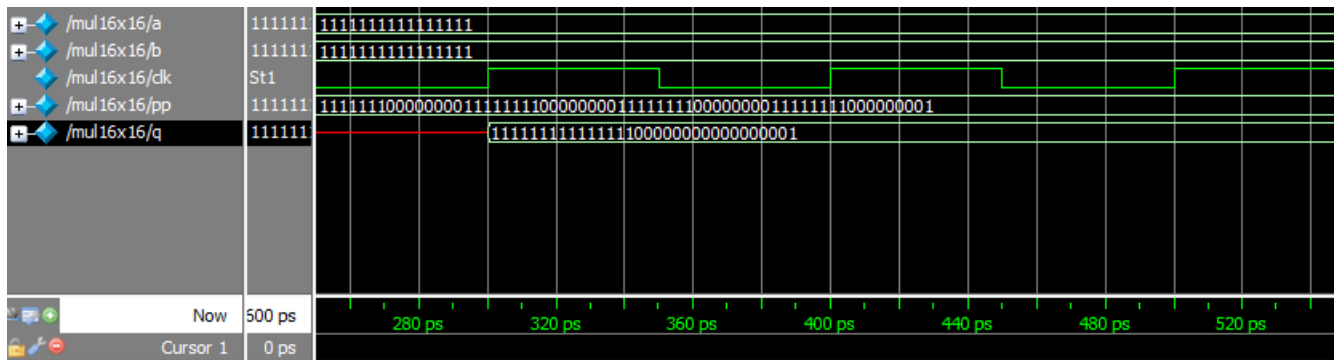


Fig 5.1 Simulation Waveform of 16x16 multiplier

#### Description

A	input data 16bit
B	input data 16 bit
Clk	clock
Q	output 32 bit
PP	partial product

## 5.1.2 SIMULATION OF 16 BIT MAC UNIT

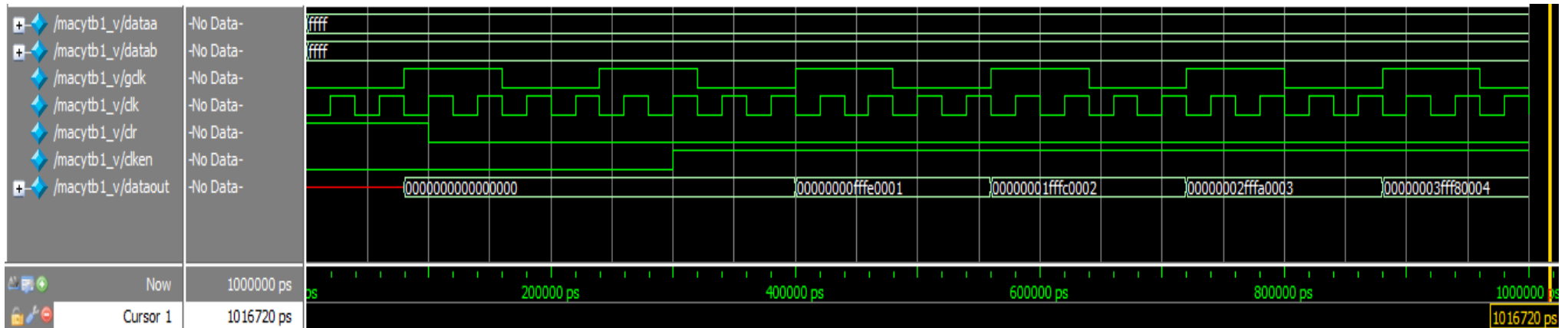


Fig 5.2 Simulation waveform of 16 bit MAC unit

### Description

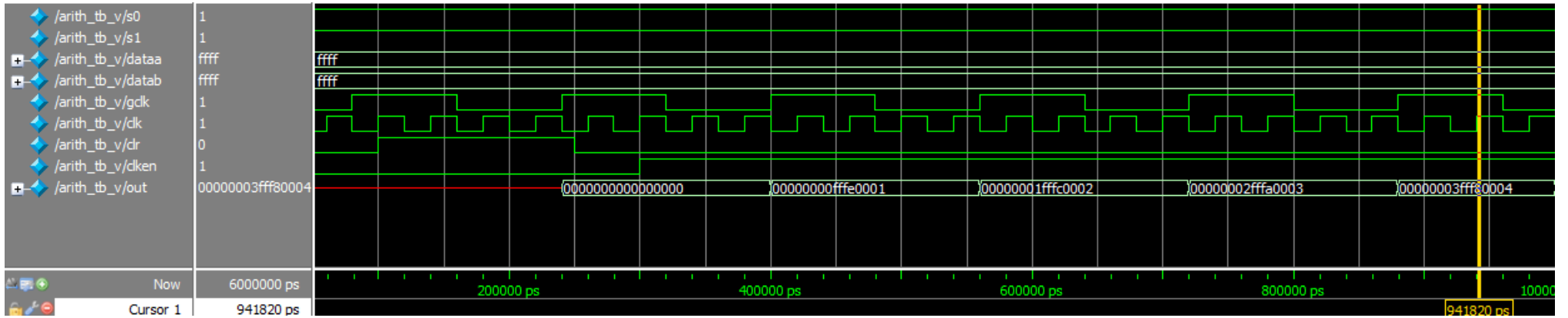
A	input data 16bit
B	input data 16 bit
gclk	global clock on which MAC unit operates
clk	clock frequency of multiplier
clr	reset signal which forces output = 0
clken	enable signal which enables MAC operation, must be 1 to produce output
Dataout	output 64 bit

### 5.1.3 SIMULATION OF 16 BIT ARITHMETIC UNIT

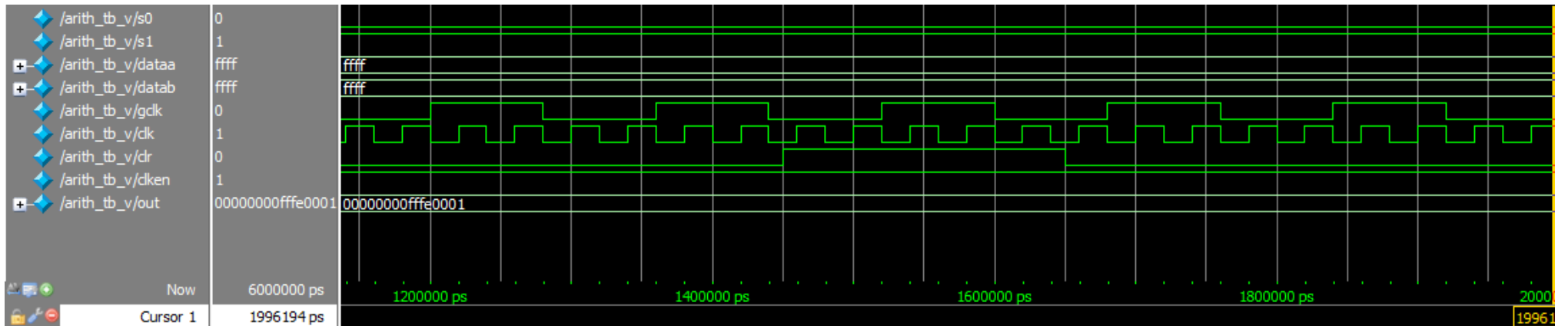
#### Description

A	input data 16bit
B	input data 16 bit
gclk	global clock on which MAC unit operates
clk	clock frequency of multiplier
clr	reset signal which forces output = 0
clken	enable signal which enables MAC operation, must be 1 to produce output
Dataout	output 64 bit
S0	select line input from control circuit
S1	select line input from control circuit.

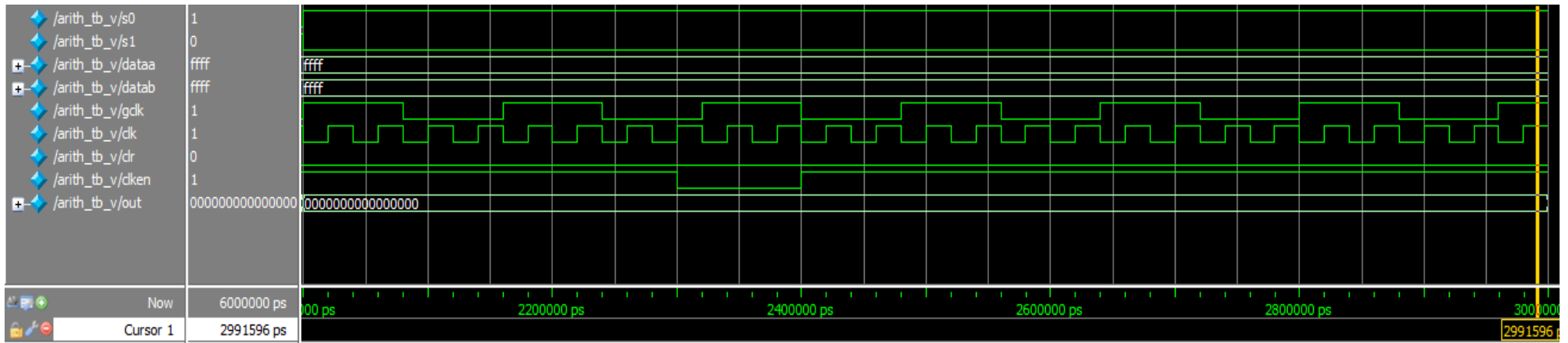
S1	S0	Operation performed
0	0	Addition
0	1	Subtraction
1	0	Multiplication
1	1	Multiply Accumulate



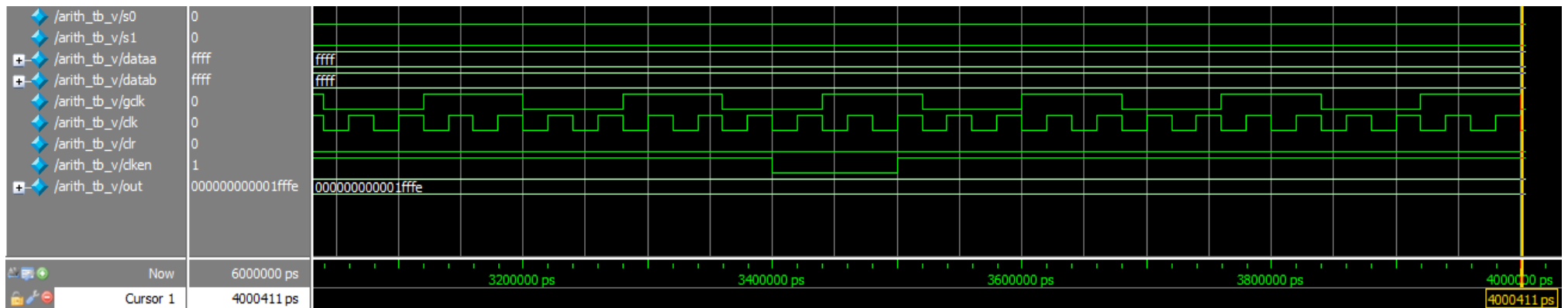
**Fig 5.3 Simulation waveform of MAC operation from 16 bit Vedic Arithmetic module**



**Fig 5.4 Simulation waveform of Multiplication operation from 16 bit Vedic Arithmetic module**



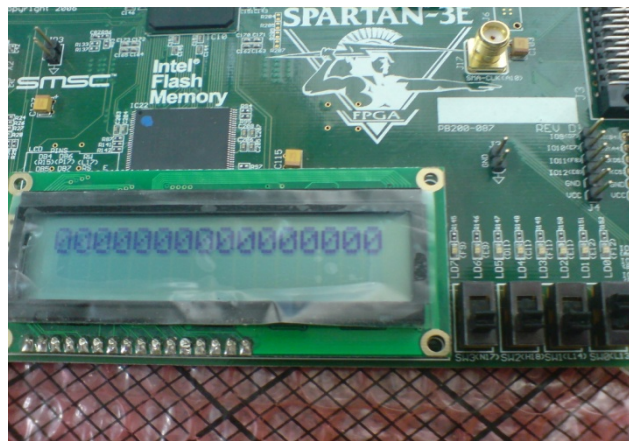
**Fig 5.5 Simulation waveform of Subtraction operation from 16 bit Vedic Arithmetic module**



**Fig 5.6 Simulation waveform of Addition operation from 16 bit Vedic Arithmetic module**



**Fig 5.7** LCD output for Addition operation of Arithmetic module



**Fig 5.8** LCD output for Subtraction operation of Arithmetic module



**Fig 5.9** LCD output for Multiplication operation of Arithmetic module



**Fig 5.10 (a)**



**Fig 5.10(b)**



**Fig 5.10(c)**



**Fig 5.10(d)**

Fig 5.10 (a , b, c, d) show the LCD output for MAC operation of Arithmetic module during first second, third and fourth clock cycle respectively.

Multiplier type	Booth	Array	Karatsuba	Vedic Karatsuba	Vedic Urdhva Tiryakbhyam
Delay	37ns	43ns	46.11ns	27.81ns	10.148ns

**Table 2 Delay comparison of different multipliers [10,11]**

## 5.2 CONCLUSION

- The design of 16 bit Vedic multiplier, 16 bit Multiply Accumulate unit and 16 bit Arithmetic module has been realized on Spartan XC3S500-4-FG320 device. The computation delay for the MAC unit and Arithmetic module are 11.151 ns and 15.749 ns respectively which clearly shows improvement in performance.
- FPGA implementation proves the hardware realization of Vedic Mathematics Algorithms.
- Urdhva Tiryakbhyam Sutra is highly efficient algorithm for multiplication.

## 5.3 FUTURE WORK

Even though Urdhva Tiryakbhyam Sutra is fast and efficient but one fact is worth noticing, that is 2x2 multiplier being the basic building block of 4x4 multiplier and so on. This leads to generation of a large number of partial products and of course, large fan-out for input signals a and b. To tackle this problem, a 4x4 multiplier can be formed using other fast multiplication algorithms possible, and keeping Urdhva Tiryakbhyam for higher order multiplier blocks. Also multiplication methods like Toom Cook algorithm can be studied for generation of fewer partial products.

In this work, some steps have been taken towards implementation of fast and efficient ALU or a Math Co processor, using Vedic Mathematics and maybe in near future, the idea of a very fast and efficient ALU using Vedic Mathematics algorithms is made real.

## APPENDIX A

In 2x2 multiply block, one fact clicks the mind, that why do we have to generate and propagate a carry, as shown in fig11, a carry from the data path of q1 is being generated with goes on to produce the output at q2 and q3. So, lets modify the algorithm at this 2x2 block level and solve this situation of carry generation and propagation.

Firstly we observe that for two bit input data for a and b, the range of input in terms of binary lies in the range (00 – 11). Lets elaborate this

It lies in the set of

{00, 01, 10, 11} for a input

Similarly, {00, 01, 10, 11} for b input

And , the output q which is 4 bits wide, lies in the set of

{0000, 0001, 0010, 0011, 0100, 0110, 1001} that is {0,1,2,3,4,6,9} in decimal

But, {5,7} never appear because they are not multiples of 1,2,3 from input.

So lets draw a truth table and minimize outputs q3,q2,q1,q0 independently

A1	A0	B1	B0	Q3	Q2	Q1	Q0
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	0
0	0	1	0	0	0	0	0
0	0	1	1	0	0	0	0
0	1	0	0	0	0	0	0
0	1	0	1	0	0	0	1
0	1	1	0	0	0	1	0
0	1	1	1	0	0	1	1
1	0	0	0	0	0	0	0
1	0	0	1	0	0	1	0
1	0	1	0	0	1	0	0
1	0	1	1	0	1	1	0
1	1	0	0	0	0	0	0
1	1	0	1	0	0	1	1
1	1	1	0	0	1	1	0
1	1	1	1	1	0	0	1

The minimized Boolean expressions for  $q_3, q_2, q_1, q_0$  independently is:

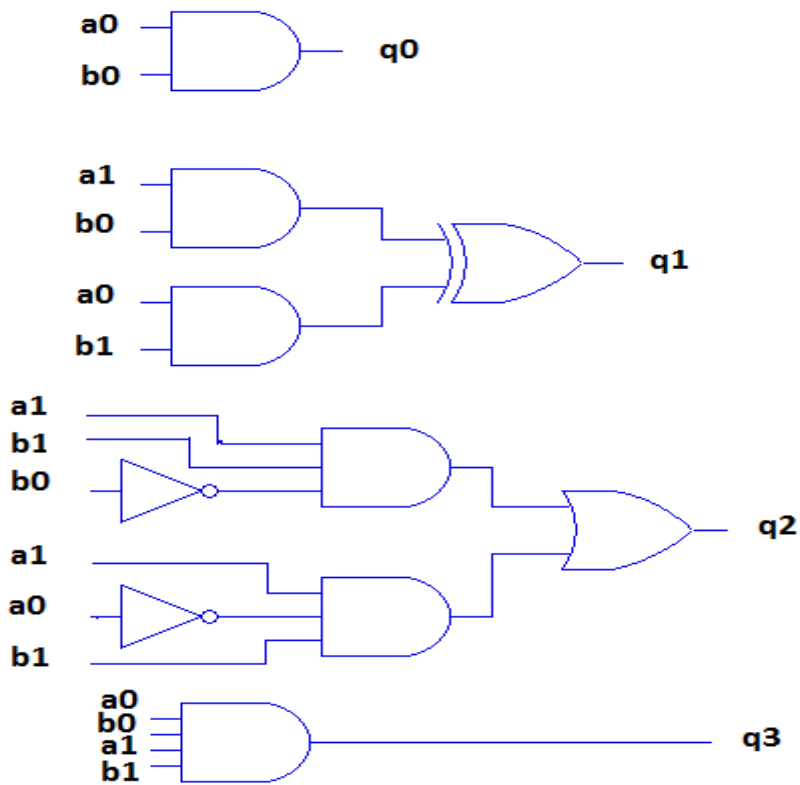
$$q_3 = a_1 a_0 b_1 b_0;$$

$$q_2 = a_1 a_0' b_1 + a_1 b_1 b_0';$$

$$q_1 = (a_1 b_0) \oplus (a_0 b_1)$$

$$q_0 = a_0 b_0;$$

and so, our hardware realization of 2x2 multiplier can be seen as :



**Fig A1 Another Hardware realization of 2x2 multiply block**

## APPENDIX B

### XILINX FPGA DESIGN FLOW

This section describes FPGA synthesis and implementation stages typical for Xilinx design flow.

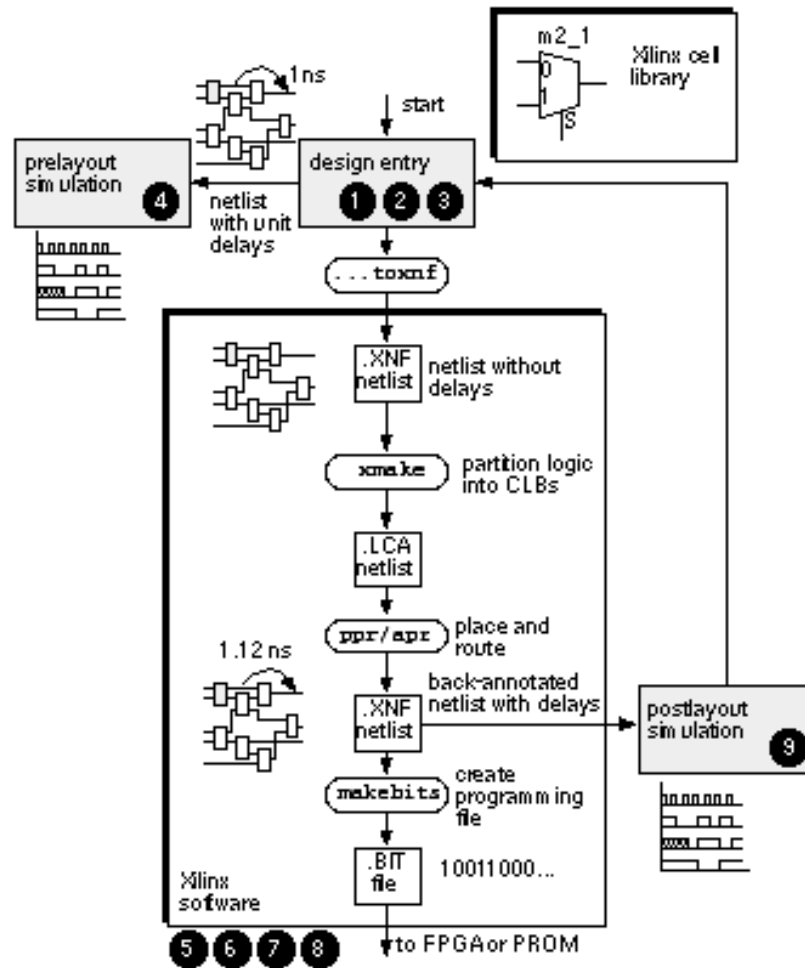


Fig A2 Xilinx FPGA Design flow [15]

#### Synthesis

The synthesizer converts HDL (VHDL/Verilog) code into a gate-level netlist (represented in the terms of the UNISIM component library, a Xilinx library containing basic primitives). By default Xilinx ISE uses built-in synthesizer XST (Xilinx Synthesis Technology). Other synthesizers can also be used.

Synthesis report contains many useful information. There is a maximum frequency estimate in the "timing summary" chapter. One should also pay attention to warnings since they can indicate hidden problems.

After a successful synthesis one can run "View RTL Schematic" task (RTL stands for register transfer level) to view a gate-level schematic produced by a synthesizer.

XST output is stored in NGC format. Many third-party synthesizers (like Synplicity Synplify) use an industry-standard EDIF format to store netlist.

## **Implementation**

Implementation stage is intended to translate netlist into the placed and routed FPGA design.

Xilinx design flow has three implementation stages: *translate*, *map* and *place and route*. (These steps are specific for Xilinx: for example, Altera combines translate and map into one step executed by *quartus\_map*.)

### Translate

Translate is performed by the NGDBUILD program.

During the translate phase an NGC netlist (or EDIF netlist, depending on what synthesizer was used) is converted to an NGD netlist. The difference between them is in that NGC netlist is based on the UNISIM component library, designed for behavioral simulation, and NGD netlist is based on the SIMPRIM library. The netlist produced by the NGDBUILD program contains some approximate information about switching delays.

### Map

Mapping is performed by the MAP program.

During the map phase the SIMPRIM primitives from an NGD netlist are mapped on specific device resources: LUTs, flip-flops, BRAMs and other. The output of the MAP program is stored in the NCD format. It contains precise information about switching delays, but no information about propagation delays (since the layout hasn't been processed yet).

Place and route

Placement and routing is performed by the PAR program.

Place and route is the most important and time consuming step of the implementation. It defines how device resources are located and interconnected inside an FPGA.

Placement is even more important than routing, because bad placement would make good routing impossible. In order to provide possibility for FPGA designers to tweak placement, PAR has a "starting cost table" option.

PAR accounts for timing constraints set up by the FPGA designer. If at least one constraint can't be met, PAR returns an error.

The output of the PAR program is also stored in the NCD format.

### **Timing Constrains**

In order to ensure that no timing violation (like period, setup or hold violation) will occur in the working design, *timing constraints* must be specified.

Basic timing constraints that should be defined include frequency (period) specification and setup/hold times for input and output pads. The first is done with the PERIOD constraint, the second - with the OFFSET constraint.

Timing constraints for the FPGA project are defined in the UCF file. Instead of editing the UCF file directly, an FPGA designer may prefer to use an appropriate GUI tool. However, the first approach is more powerful.[15]



## REFERENCES

---

- [1] [www.en.wikipedia.com](http://www.en.wikipedia.com)
- [2] Jagadguru Swami Sri Bharati Krishna Tirthji Maharaja, "Vedic Mathematics", Motilal Banarsidas, Varanasi, India, 1986.
- [3] Harpreet Singh Dhillon and Abhijit Mitra, "A Reduced- Bit Multiplication Algorithm for Digital Arithmetics", International Journal of Computational and Mathematical Sciences 2;2 © www.waset.org Spring 2008.
- [4] Shripad Kulkarni, "Discrete Fourier Transform (DFT) by using Vedic Mathematics", report, [vedicmathsindia.blogspot.com](http://vedicmathsindia.blogspot.com), 2007.
- [5] Himanshu Thapliyal, Saurabh Kotiyal and M. B Srinivas, "Design and Analysis of A Novel Parallel Square and Cube Architecture Based On Ancient Indian Vedic Mathematics", Centre for VLSI and Embedded System Technologies, International Institute of Information Technology, Hyderabad, 500019, India, 2005 IEEE
- [6] Shamim Akhter, "VHDL Implementation of Fast NXN Multiplier Based on Vedic Mathematics", Jaypee Institute of Information Technology University, Noida, 201307 UP, INDIA, 2007 IEEE
- [7] Himanshu Thapliyal and M.B Srinivas, "An Efficient Method of Elliptic Curve Encryption Using Ancient Indian Vedic Mathematics", IEEE, 2005.
- [8] El hadj yousef wajih, Zeghid Medien, Machhout Mohsen, Bouallegue Belgacem, Tourki Rached, "Efficient Hardware Architecture of Recursive Karatsuba-Ofman Multiplier", 2008 International Conference on Design & Technology of Integrated Systems in Nanoscale Era
- [9] [www.mathworld.wolfram.com/KaratsubaMultiplication.html](http://www.mathworld.wolfram.com/KaratsubaMultiplication.html)
- [10] Abhijeet Kumar, Dilip Kumar, Siddhi, "Hardware Implementation of 16\*16 bit Multiplier and Square using Vedic Mathematics", Design Engineer, CDAC, Mohali.

[11] M. Ramalatha, K. Deena Dayalan, P. Dharani, S. Deborah Priya, High Speed Energy Efficient ALU Design using Vedic Multiplication Techniques, July 15-17, 2009 Zouk Mosbeh, Lebanon

[12] [Neil H.E Weste, David Harris, Ayan Banerjee, "CMOS VLSI Design, A Circuits and Systems Perspective", Third Edition, Published by Person Education, PP-327-328]

[13] Himanshu Thapliyal and Hamid R. Arabnia, "A Time-Area- Power Efficient Multiplier and Square Architecture Based On Ancient Indian Vedic Mathematics", Department of Computer Science, The University of Georgia, 415 Graduate Studies Research Center Athens, Georgia 30602-7404, U.S.A.

[14] E. Abu-Shama, M. B. Maaz, M. A. Bayoumi, "A Fast and Low Power Multiplier Architecture", The Center for Advanced Computer Studies, The University of Southwestern Louisiana Lafayette, LA 70504.

[15] [www.fpga-central.com](http://www.fpga-central.com)

[16] "Spartan-3E FPGA Starter Kit Board User Guide", UG230 (v1.1) June 20, 2008.