# Design of an Optimized Low Power Vedic Multiplier Unit

**for Digital Signal Processing Application**

Digital multipliers play a crucial role in various Digital Signal Processing units. They carry the major responsibility of power expenditure in the system and ultimately determine its speed. As a result, it is always beneficial to develop high performance, low power multipliers.

Vedic Mathematics is a set of mathematical rules, derived from ancient Indian scripts that makes arithmetic calculations extremely fast and simple. There are 16 rules or *Sutras* expounded in Vedic Mathematics. This report presents novel designs of a multiplier based on the Vedic Sutras on multiplication - *Urdhva Tiryakbhyam* and *Nikhilam*.

# Chapter 1

**Introduction**

***Digital signal processing*** (DSP) is firmly being established as an extremely vibrant and vital field in the Electronics industry. The past few decades have seen an exponential growth in the number of products and applications that involve DSP, with a wide reach into diverse domains such as audio signal processing, digital image processing, video compression, speech processing, speech recognition, digital communications, RADAR, SONAR, financial signal processing, seismology and even biomedicine. Especially since computers have evolved into powerful machines capable of high computational complexity, almost all the signal processing takes place in the Digital Domain.

Frequently used algorithms include the Convolution operation, Finite Impulse Response (FIR) Filter, Infinite Impulse Response (IIR) Filter, and Fast Fourier Transform (FFT), all of which require intensive computation. DSP algorithms generally require a large number of mathematical operations to be performed quickly and repeatedly on a series of incoming data. The signals are constantly converted from analog to digital, digitally manipulated, and then converted back to analog.



Figure 1.1: A typical Digital Processing System

Most general–purpose microprocessors and operating systems can execute DSP algo- rithms successfully, but consume more power and occupy a larger area which is not suitable for most portable applications like those on mobile phones, biomedical devices, etc. A specialized digital signal processor, the **Digital Signal Processor (DSP processor)**, having different architectures and features optimized specifically for digital signal processing, is hence preferred. This will tend to provide a lower-cost solution, with better performance, lower latency and lesser power consumption. Thus, the efficiency in the design of the under- lying hardware in the DSP processors will reflect in the performance of the applications.

One of the most important hardware structures in a DSP processor is the **Multiply- Accumulate (MAC)** unit. A conventional MAC unit consists of an *n bit* multiplier, the output of which is added to/subtracted from the contents of an Accumulator that stores the result. Thus, the MAC unit implements functions of the type *A* + *BC*. The ability to compute with a fast MAC unit is essential to achieve high performance in many DSP algorithms, and which is why there is at least one dedicated MAC unit in all of the modern commercial DSP processors.

*−*

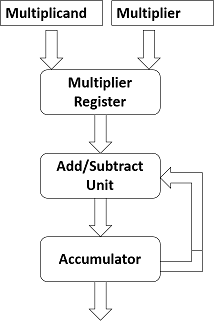


Figure 1.2: A MAC unit

Hence as it can be observed, **Digital Multipliers** are the core components of all MAC units and hence all DSP processors. The multiplier lies in the ***Critical Delay Path*** and ultimately determines the performance of any algorithm in the processor. Currently, multiplication time is still the major factor in determining the instruction cycle time of a DSP chip apart from contributing to the bulk of its power expenditure. Since multiplication drains power quickly and dominates the ***execution time*** of most DSP algorithms, there is a need for **Low–Power, High–Speed Multipliers**. In this concern, design of efficient multipliers has long been a topic of interest to digital design engineers.

The other function that a MAC unit inherently performs is the addition operation. It is one of the most essential operations in the instruction set of any processor. Other instructions such as subtraction and multiplication employ addition in their operations, and their underlying hardware is primarily dependent on the addition hardware. Hence the performance of a design will be often be limited by the performance of its adders. It is therefore as important to choose the correct adder to implement in a design as it is to choose a multiplier because of the many factors it affects in the overall chip.

The main expected features of any DSP block, be it an adder or a multiplier, are *speed*, *accuracy* and *easy integrability*. A number of interesting algorithms have been reported in literature, each offering different advantages and having trade-offs in terms of speed, circuit complexity, area and power consumption, forming an active area of research.

## Vedic Mathematics - An Overview

Vedic Mathematics is the name given to a set of rules derived from Ancient Indian Scriptures, elucidating different mathematical results and procedures in simple and un- derstandable forms. The word ***Vedic*** is derived from the word ***Veda*** which means the store–house of all knowledge.

It is claimed to be a part of the ***Sthapatya Veda*** , a book on civil engineering and architecture, which is an ***Upaveda*** (supplement) of the ***Atharva Veda*** . It covers explanations of several modern mathematical terms including arithmetic, geometry (plane, co-ordinate), trigonometry, quadratic equations, factorization and even calculus.

The beauty of Vedic mathematics lies in the fact that it reduces the otherwise cumbersome- looking calculations in conventional mathematics to very simple ones. This is because the Vedic formulae are claimed to be based on the natural principles on which the human mind works.

Vedic mathematics is mainly based on 16 Sutras (or aphorisms) dealing with various branches of mathematics like arithmetic, algebra, geometry, etc. Of these, there are two Vedic Sutras meant for quicker multiplication. They have been traditionally used for the multiplication of two numbers in the decimal number system. They are –

* + 1. ***Nikhilam Navatashcaramam Dashatah*** : All from 9 and last from 10
    2. ***Urdhva Tiryakbhyam*** : Vertically and crosswise

## Motivation

Multiplication involves two basic operations – the generation of partial products and their accumulation. Clearly, a smaller number of partial products reduces the complexity, and, as a result, reduces the partial products accumulation time.

When two *n bit* numbers are multiplied, a 2*n bit* product is produced. Previous research on multiplication, i.e. shift and add techniques, focused on multiplying two *n bit* numbers to produce *n* partial products and then adding the *n* partial products to generate a 2*n bit* product. In which case, the process is sequential and requires ***n*** processor cycles for an *n n* multiplication. Advances in VLSI have rendered ***Parallel Multipliers*** – fully combinational multipliers, which minimize the number of clock cycles/steps required, feasible.

*−*

*×*

*−*

*− −*

Two most common multiplication algorithms followed in the digital hardware are the ***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. In the case of Booth multiplication algorithm, it multiplies two signed binary numbers in two’s complement notation. Andrew Donald Booth used desk calculators that were faster at shifting than adding and created the algorithm to increase their speed. It is possible to reduce the number of partial products by half, by using the technique of Radix-4 Booth recoding. But both do have their own limitations. The search for a new design of a multiplier which will radically improve the performance is always on.

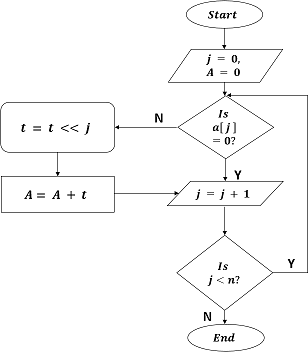


Figure 1.3: Array Multiplier – Algorithm

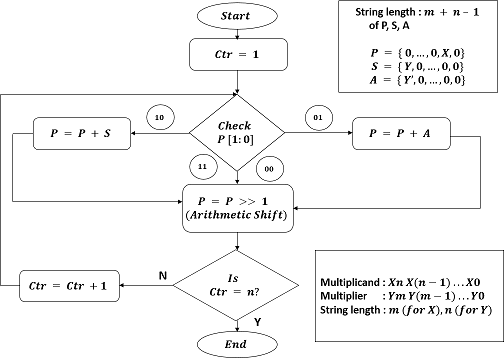


Figure 1.4: Booth Multiplier – Algorithm

The ***main motivation*** of this project is to make use of the simplicity of the Vedic sutras and adapt it for binary arithmetic to get an efficient and optimum digital multiplier fulfilling the demands of the growing technology. This, if implemented correctly, has the capability to reduce the computational time of DSP applications to a fraction of what it is today and revolutionize the standard power consumption of DSP chips. Fortunately or unfortunately, the potential of the Vedic Algorithms has remained untapped and unplun- dered for long. It would be a source of pride to prove that the Indian–originated methods can surpass the existing algorithms and to use them widely in various applications for the benefit of the industry.

## Objective

The aim of this project is to design and implement an optimized digital multiplier which will multiply two real integers for DSP applications incorporating Vedic Multiplication principles.

The goals are:

* To reduce the **computational time**
* To optimize the **area** occupied
* To minimize the **power** consumed
* To realize it for **128 bits**
* To develop novel algorithms for obtaining a highly optimum multiplier

To design and implement an integrated multiplier which will decide the algorithm to be used depending on the given inputs

*•*

* To finalize on an optimum adder and use it to implement all the addition operations

# Chapter 2

**Literature Review**

Here, a brief summary of the work that has already been done in this field with Vedic Multipliers is presented. A few results are noted down for comparison later.

The implementation of an 8 *bit* Vedic multiplier enhanced in terms of propagation delay when compared with conventional multipliers like Array multiplier, Braun multiplier, Modified Booth multiplier and Wallace tree multiplier has been given by *Pavan Kumar U.C.S, et al, 2013*. Here, they have utilized an 8 *bit* barrel shifter which requires only one clock cycle for *n* number of shifts. The design could achieve propagation delay of **6.781 ns** using barrel shifter in base selection module and multiplier.

*−*

*−*

*S. Deepak, et al, 2012*, have proposed a new multiplier design which reduces the number of partial products by 25 %. This multiplier has reported to have been used with different adders available in literature to implement multiplier accumulator (MAC) unit and parame- ters such as propagation delay, power consumed and area occupied have been compared in each case. From the results, Kogge Stone adder was been chosen as it was claimed to have provided optimum values of delay and power dissipation. The results obtained have been compared with that of other multipliers and it has been reported that the proposed multiplier has the lower propagation delay when compared with Array and Booth multipliers.

A high speed complex multiplier design (ASIC) using Vedic Mathematics has also been reported by *Prabir Soha, et al, 2011*. A complex number multiplier design based on the formulas of the ancient Indian Vedic Mathematics, was said to have been implemented in Spice spectre and compared with the mostly used architecture like distributed arithmetic, parallel adder based implementation, and algebraic transformation based implementation. It claims to have combined the advantages of the Vedic mathematics for multiplication which encounters the stages and partial product reduction. The proposed complex number multiplier has been reported to offer 20% and 19% improvement in terms of propagation delay and power consumption respectively, in comparison with parallel adder based im- plementation. The corresponding improvement in terms of delay and power was reported to be 33% and 46% respectively, with reference to the algebraic transformation based implementation.

*Mohammed Hasmat Ali, et al, 2013*, have presented a detailed study of different multipliers based on Array Multiplier, Constant coefficient multiplication (KCM) and multiplication based on Vedic Mathematics. The reported multipliers have been coded in Verilog HDL (Hardware Description Language) and simulated in ModelSimXEIII6.4b and synthesized in EDA tool Xilinx ISE12. All multipliers are compared based on LUTs (Look up table) and path delays. Results report that Vedic Urdhva Tiryakbhyam sutra is the fastest Multiplier with least path delay. The computational path delay for proposed 8 8 bit Vedic Urdhava Tiryakbhyam multiplier was reported to be **17.995 ns**.

*×*

Karatsuba-Ofman algorithm has been reported to have been used by *M.Ramalatha, et al, 2009*, in the implementation of an efficient Vedic multiplier which is meant to have high speed,less complexity and consuming less area. Also after using this multiplier module a Vedic MAC unit was constructed and both these modules were integrated into an arithmetic unit along with the basic adder subtractor.

A generalized algorithm for multiplication has been reported by Ajinkya Kala, 2012, through recursive application of the Nikhilam Sutra from Vedic Mathematics, operating in radix - 2 number system environment suitable for digital platforms. Statistical analysis has been carried out based on the number of recursions profile as a function of the smaller multiplicand. The proposed algorithm was claimed to be efficient for smaller multiplicands as well, unlike most of the asymptotically fast algorithms. It was implemented for same sized inputs but an algorithm was presented which could be used to compute multiplication of two variable bit numbers. The algorithm was reported to solely depend on the ratio of the number of 1’s and 0’s used to represent a number in binary, rather than on the magnitude of the number. It was mentioned that as the ratio approaches 1, the number of operations required for the multiplication increases and decreases as the ratio tends to move close to 0.

*Ramachandran.S, et al, 2012*, have thought of an Integrated Vedic multiplier architecture, which by itself selects the appropriate multiplication sutra (UT or Nikhilam) based on the inputs. So depending on inputs, whichever sutra is faster, that sutra is to be selected by the proposed integrated Vedic multiplier architecture. It was implemented for 16 bits but there has not been a clear report on the results or the design.

*Kabiraj Sethi, et al, 2012*, have proposed a high speed squaring circuit for binary numbers is proposed. High speed Vedic multiplier is used for design of the proposed squaring circuit. Only one Vedic multiplier is used instead of four multipliers as reported previously. In addition, one squaring circuit is used twice.

In paper presented by *G.Ganesh Kumar, et al, 2012*, the Verilog HDL coding of Urdhva tiryakbhyam Sutra for 32 32 bits multiplication and their FPGA implementation by Xilinx Synthesis Tool on Spartan 3E kit have been done and the output has been displayed on LCD of Spartan 3E kit. The synthesis results show that the computation time for calculating the product of 32 *×* 32 bits is **31.526 ns**.

*×*

The designs of 16 16 bits, 32 32 bits and 64 64 bits Vedic multiplier have been implemented as reported by *Vinay Kumar, 2009* on Spartan XC3S500-5-FG320 and XC3S1600-5-FG484 device according to this thesis. The computation delay for 16 16 bits Booth multiplier was **20.09 ns** and for 16 16 bits Vedic multiplier was **6.960 ns**. Also computation delays for 32 32 bits and 64 64 bits Vedic multiplier was obtained **7.784 ns** and **10.241 ns** respectively.

*× × ×*

*× ×*

*×*

*×*

A new reduced-bit multiplication algorithm based on Vedic mathematics has been proposed by *Honey Durga Tiwari, et al, 2008*. The framework of the proposed algorithm is taken from Nikhilam Sutra and is further optimized by use of some general arithmetic operations such as expansion and bit shifting to take full advantage of bit-reduction in mul- tiplication. The computational efficiency of the algorithm has been illustrated by reducing a general 44 multiplication to a single 22 multiplication operation.

*Manoranjan Pradhan, et al, 2011*, have presented the concepts behind the ”Urdhva Tiryagbhyam Sutra” and ”Nikhilam Sutra” multiplication techniques in their paper. It then shows the architecture for a 1616 Vedic multiplier module using Urdhva Tiryagbhyam Sutra. The paper then extends multiplication to 1616 Vedic multiplier using ”Nikhilam Sutra” technique. The 1616 Vedic multiplier module using Urdhva Tiryagbhyam Sutra uses four 88 Vedic multiplier modules; one 16 *bit* carry save adders, and two 17 *bit* full adder stages. The carry save adder in the multiplier architecture increases the speed of addition of partial products. The 1616 Vedic multiplier is reported to have been coded in VHDL, synthesized and simulated using Xilinx ISE 10.1 software. This multiplier is implemented on Spartan 2 FPGA device XC2S30-5pq208.

*− −*

An integer multiplication algorithm was proposed by Shri Prakesh Dwivedi, 2013, using Nikhilam method of Vedic mathematics which can be used to multiply two binary numbers efficiently taking advantage of the fact that this sutra can convert large-digit multiplication to corresponding small digit multiplication.

*Himanshu Thapliyal, et al, 2009*, have proposed parallel architectures for computing square and cube of a given number based on Vedic mathematics. For the Xilinx FPGA family, it is observed that for8 *bit*, the gate delay of the proposed square architecture is **28 ns** with area of 90(device utilized) while it is **70 ns** for previously reported squares with area of **77**. For the same operand size, the gate delay in the proposed cube architecture is **28 ns** with area of **90** while for the cube previously reported is **79 ns** with area of **768**. As the operand width is increased to 16, the gate delay of the proposed square architecture increases slightly to **38 ns** with area of **348**(device utilized) while for the square proposed earlier, it significantly increases to **70 ns** with area of **441**. For the operand size of **16**, the cube statistics are found to be **54 ns** with area of **1336** for the proposed Vedic cube while it is **186 ns** with area of 6550 for the cube proposed before.

*−*

## CHAPTER 3

## MULTIPLIERS

**INTRODUCTION**

Multiplication is a very important arithmetic operation for many signal processing applications such as convolution, correlation, frequency analysis, image processing etc. With advances in technology, many researchers have tried and are trying to design multipliers which offer either of the following design targets – high speed, low power consumption, regularity of layout and hence less area or even combination of them in one multiplier thus making them suitable for various high speed, low power and compact VLSI implementation.

The speed of multiplication operation is an important aspect in the processors of these applications. The processor requires considerable amount of time in performing the multiplication operation and performance of the multiplier determines system performance. Hence, the speed and power efficient multipliers are of greater importance in signal and image processing applications.

This chapter discusses the performance of different types of multipliers particularly on Vedic multipliers and constant multipliers and compares the efficiency of the multipliers in order to utilize in the processors to enhance the system performance.

* 1. **ARRAY MULTIPLIER**

Array multiplier is an efficient layout of a combinational multiplier. Multiplication of two binary number can be obtained with one micro-operation by using a combinational circuit that forms the product bit all at once thus making it a fast way of multiplying two numbers since only delay is the time for the signals to propagate through the gates that forms the multiplication array.

In array multiplier, consider two binary numbers A and B, of m and n bits. There are summands that are produced in parallel by a set of an AND gates. n x n multiplier requires n (n-2) full adders, n half-adders and n2 AND gates. Also, in array multiplier worst case delay would be (2n+1) Designing multipliers that are of high-speed, low power, and regular in layout are of substantial research interest. Speed of the multiplier can be increased by reducing the generated partial products. Many attempts have been made to reduce the number of partial products generated in a multiplication process one of them is array multiplier.

In Array multiplier, half adder has been used to sum the carry products in minimum time. Achieving high speed integrated circuits with low power consumption is a major concern for the VLSI circuit designers.

* + 1. **CONVENTIONAL ARRAY MULTIPLIER**

An existing array multiplier is very much regular in its structure when compared to the conventional array multiplier and uses an only short wire that goes from one full adder cell to adjacent full adder cell. It has very simple and efficient layout in VLSI and can be easily and efficiently pipelined. The conventional array multiplier is synthesized. This low power full adder cell has a drawback of giving large delay when compared to other adders. This multiplier consumes significant less amount of power and have more delay compared to other adders. The product of these two bits can be written as P7P6P5P4P3P2P1P0.Where P0 is the LSB AND P7 is the MSB as shown in figure 6.1.

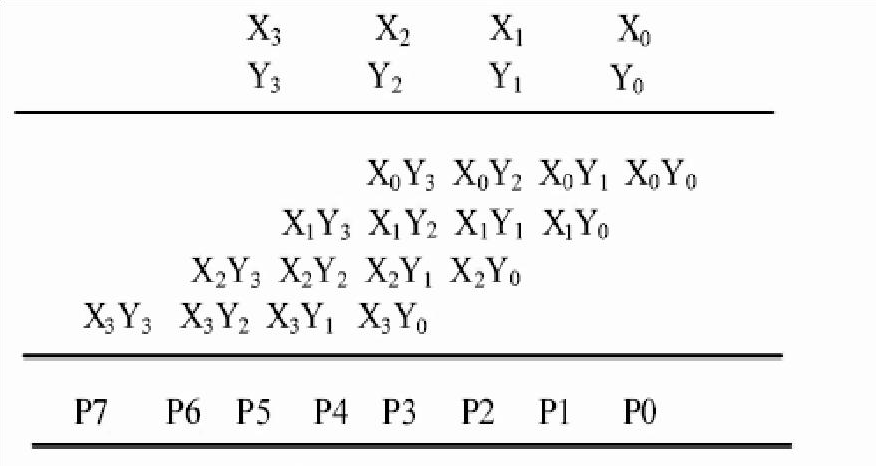


Figure 3.1: 4x4 Array Multiplication

The conventional array multiplier uses carry save addition to add the products. In the carry save addition method, the first row will be either half adders or full adders. If the first row of the partial products is implemented with full adders, Cin will be considered as 0. Then the carries of each full adder can be diagonally [15]forwarded to the next row of the adder.The resulting multiplier is said to be carry save array multiplier as the carry bits are not immediately added but rather saved for the next stage of addition. Hence the name carry save multiplier. In the design if the full adders have two input data the third input is considered as zero.

The final adder which is used to add carries and sums of the multiplier is removed. Then the carries of the multiplier at the final stage is carefully added to the inputs of the multiplier as shown in Figure 6.2. The carry of the fourth column of the multiplier is given to the input of the fifth column instead of zero. Then the carry of the fifth column is forwarded to the input of the

sixth column so on. And in this carry of the seventh column of the adder is considered as the most significant bit.

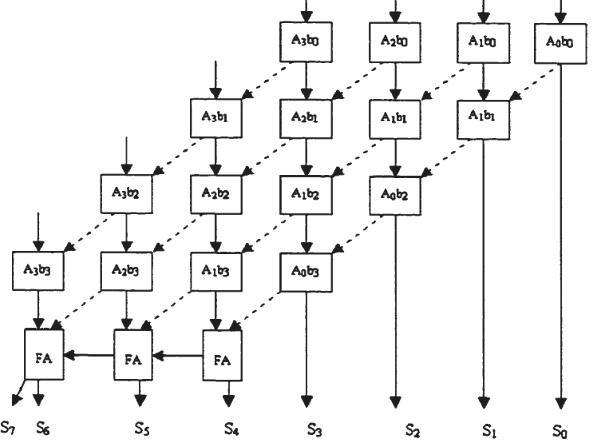


Figure 3.2: Array Multiplier

* 1. **MODIFIED BOOTH ALGORITHM**

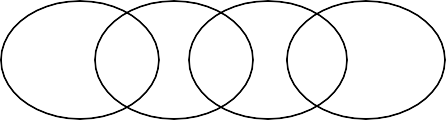
Mac Soorley [12] proposed a modification to the Booth Algorithm. The advantage of this algorithm is minimizing the partial products by half regardless of the input bits. The partial products generated in the second stage of multiplication are reduced, hence it[13] is known as one of the fastest multiplication algorithm.

Table 6.1: Booth bits

|  |  |  |
| --- | --- | --- |
| Qi | Qi-1 | No. of partial products |
| 0 | 0 | Zero partial products |
| 0 | 1 | One partial product |
| 1 | 0 | One partial product |
| 1 | 1 | Two partial products |

In Table 6.1 we can infer that in the last operation (1 1), two partial products are generated. In order to reduce the partial products, the last operation of (1 1) should be reduced or replaced according to the booth recoded table. Three bits are taken at a time and are encoded into one of

{-2, -1,0,1,2}. Grouping is started from the LSB, the first block uses only two bits of the multiplier and assumes a zero for the third bit as shown in figure 6.3.



1 1 0 1 0 1 0 1

Figure 3.3: Grouping of numbers by overlapping technique

Table 3.2: Booth Recoded Table

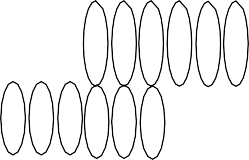
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Multiplier Bits | | | Encoded Multiplier value ∑ | Partial products | Operation |
| Qi+1 | Qi | Qi- 1 |
| 0 | 0 | 0 | 0 | 0M | No action |
| 0 | 0 | 1 | 1 | 1M | Add |
| 0 | 1 | 0 | 1 | 1M | Add |
| 0 | 1 | 1 | 2 | 2M | Shift left and add |
| 1 | 0 | 0 | -2 | -2M | Shift left and subtract |
| 1 | 0 | 1 | -1 | -1M | Subtract |
| 1 | 1 | 0 | -1 | -1M | Subtract |
| 1 | 1 | 1 | 0 | 0M | No action |

Table 3.2 describes the Booth Recoding table. In the above table, 3 bits of the Multiplier Q are taken for inspection to find the encoded value. ∑ is the encoded value which is to be multiplied with the Multiplicand ‘M’ and is given by ∑ = Qi-1 + Qi -2 \* Qi+1.

* 1. **WALLACE-TREE MULTIPLIER**

The Wallace tree multiplier [16] is a parallel multiplier and offers faster performance for larger operands because its height is logarithmic in word size, not linear. It uses the carry save addition algorithm to reduce the propagation delay.

The multiplication operation performed in the Wallace tree, every possible bit in every column is covered by the 3:2 (*full adder*) or 2:2 (half adder) compressors repetitively until the final partial product is left with a depth of only 2. Thus, to compress the partial products a Wallace tree multiplier uses more hardware is utilized to get final product as quickly as possible. Figure 3.4(a) shows the logic used for 8x8 bits Wallace tree multiplication and the tree structure organised according to the addition performed for the partial products shown in figure 3.4(b).



# (b)

~~(a)~~

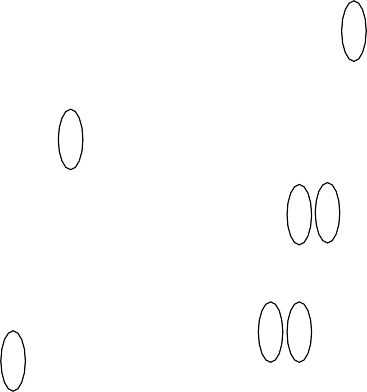
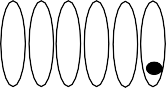
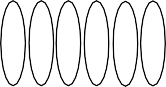
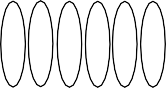
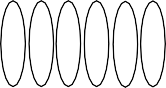


Figure 3.4 (a)Wallace tree structure for 8x8 multiplication. (b) Reorganized Matrix of Wallace tree matrix.

Wallace tree has three steps: -

1. Multiply each bit of multiplier with same bit position of multiplicand. Depending on the position of the multiplier bits generated partial products have different weights.
2. Reduce the number of partial products to two by using layers of full and half adders.
3. After second step we get two rows of sum and carry, add these rows with conventional adders.

Explanation of second step: -

As long as there are three or more rows with the same weight add a following layer:

* 1. Take any three rows with the same weights and input them into a full adder. The result will be an output row of the same weight i.e. sum and an output row with a higher weight for each three input wires i.e. carry.
  2. If there are two rows of the same weight left, input them into a half adder.
  3. If there is just one row left, connect it to the next layer.
  4. The advantage of the Wallace tree is that there are only O (log n) reduction layers (levels), and each layer has O (1) propagation delay. As making the partial products is O (1) and the final addition is O (log n), the multiplication is only O (log n), not much slower than addition (however, much more expensive in the gate count). For adding partial products with regular adders would require O (log n2) time.
  5. **VEDIC MULTIPLIER**

Vedic multiplier or Urdhva Tiryakbhyam Sutra is a multiplication algorithm [13-14] which is one of the easiest and fastest approach to perform mathematical operation. Due to coherence and symmetry, these algorithms when deployed in Digital processors consume less area with lower power consumption. If the number of bits are increased, then the gate delay and area increase very slowly as compared to other multipliers.

Figure 6.5 shows multiplication of two binary numbers, the vertical and crosswise combination of the binary bits generates the partial products concurrently. The partial products of 4x4 multiplier (P7, P6, P5, P4, P3, P2, P1, P0) are given by the following equations:

Steps for performing multiplication using sutra

The above figure illustrates the steps involved in Vedic multiplication using Urdhva Tiryagbhyam Sutra which are explained below:

* + Step1: Vertical multiplication of first digits of the numbers is done i.e. (a0b0)
  + Step2: Crosswise Multiplication Addition is done of 1st 2 digits i.e. (a1b0+a0b1)

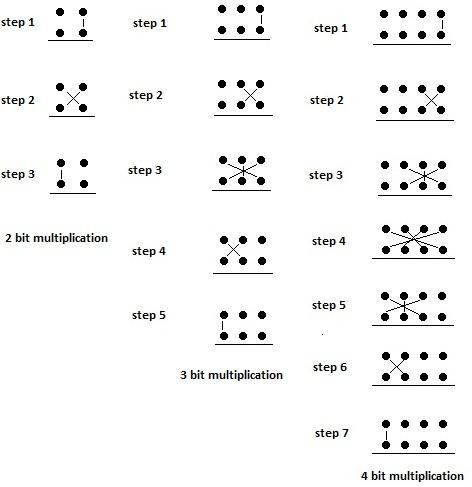


Figure 3.5: Illustration of Urdhva Tiryakbhyam Sutra for 4 bits.

* + Step3: Crosswise Multiplication addition of 1st 3 digits of the inputs is being done. (a2b0+a0b2+a1b1)
  + Step4: Crosswise multiplication addition of all the digits of both the inputs is performed (a0b3+a3b0+a2b1+a1b2)
  + Step5:Crosswise multiplication addition of last 3digits isperformed i.e.(a1b3+a3b1+a2b2)
  + Step6: Crosswise multiplication addition of last 2 digits is done i.e. (a2b3+a3b2)
  + Step7: Finally, Vertical multiplication of last two digits is performed i.e. (a3b3)

For all above steps except the first step, the carry is initialized and used for further multiplication process.

In 4\*4 Vedic multiplier, multiplication is done in a single line using Urdhva Tiryakbhyam Sutra, where in 4x4 multiplier partial products are generated to obtain the resultant product. Initially carry is taken as zero, if we encounter more than one line in any step, then all the results are added to the previous carry. In each step, LSB acts as the result bit and all other bits act as a carry for further step. Hence the number of steps required for the multiplication process gets reduced and speed of the multiplier increases.

* 1. **CONSTANT MULTIPLIER**

Constant coefficient multiplier has one fixed input and one variable input. By shifting, adding and subtracting the two variables are multiplied. The proposed multiplier design uses two techniques CSD and CSE to reduce the hardware and fixed point representation to improve the speed.

* + 1. **CANONICAL SIGNED DIGIT (CSD)**

In computing, CSD is a method that can be used to encode a value in a signed digit representation and using it one number can be represented in many ways. Probability of digit being zero is 60% and 50% in two‟s complement encoding. CSD leads to efficient implementation of adders and subtractors (one such example is multiplication by a constant) in a hardwired DSP[14].

* Important Characteristics of CSD

1. CSD representation of a number consists of numbers 0, 1, -1
2. It is a unique method of representation.
3. Number of nonzero digits is minimal.
4. There cannot be two consecutive nonzero digits.

* Representation

The representation is done using a sequence of one or more of the symbols -1, 0, 1 or (-, 0,+) i.e., each position with symbol -, + or -1 or +1 represents subtraction or addition of a power of 2.

In CSD, the representation of numbers is considered as a succession of bits, where each bit is in the set {0, 1, and -1}.

* Implementation

CSD is obtained by transforming every sequence of zero followed by ones (011…1) into +1 followed by zeros and the least significant bit by -1.Starting from the right (LSB), if there are more than one non-zero elements (1 or - 1) in a row, take all of them, plus the next zero. (if there is not zero at the left side of the MSB, create one there). CSD is used to realize the multiplication when the one factor is constant. The CSD unit is used to construct number that contains the minimum possible number of non-zero bits.

Table 3.3 Represent of binary and CSD

|  |  |  |
| --- | --- | --- |
| **Coefficient Values** | **Binary** | **CSD** |
| 0.008956 | 00111100\_00010010\_10111100\_00110000 | 01000100\_00010101\_01000100\_01010000 |
| 0.005974 | 00111011\_11000011\_11000001\_10001011 | 01000100\_01000100\_01000010\_10010101 |
| -0.00800 | 10111100\_00000011\_00010010\_01101111 | 101000100\_00000101\_00010010\_10010001 |
| -0.00132762 | 10111100\_10010101\_10000111\_01010001 | 101000101\_01010010\_00010001\_01000101 |

**3.7 SUMMARY**

This chapter summarizes a comparative study on Vedic multipliers, conventional array multiplier, Wallace tree multiplier, modified booth multiplier and constant multiplier. Multipliers are mainly classified into two types viz. serial multiplier and parallel multiplier. Serial multiplier which requires more processing time and the Parallel multiplier such as Modified Booth multiplier, Wallace tree multiplier, Array multiplier, floating point arithmetic, fixed point arithmetic etc. Array multiplier reduces the delay, but it requires a large number of gates, which increases area and power consumption. Booth multipliers are used for multiplication of signed binary numbers but, it does not work when there are alternate zeros and ones. Wallace tree multiplier works at high speed, but it exhibits structural irregularity which is more complex for hardware implementation.

# Chapter 4

**Theoretical Background – Algorithms**

## Urdhva Tiryakbhyam

This multiplication scheme is best understood by using an example. To illustrate, consider the multiplication of two decimal numbers 325 738. As shown in figure 3.1, the digits on either side of a line are multiplied and the products from each line are added along with the carry from the previous step. This generates one bit of the result as well as a carry. This carry is added in the next step and the process goes on. In each step, the least significant bit (LSB) acts as the result bit and all other bits act as carry for the next step. Initially, the carry is taken to be zero. (*Vinay Kumar, 2009* )

*×*

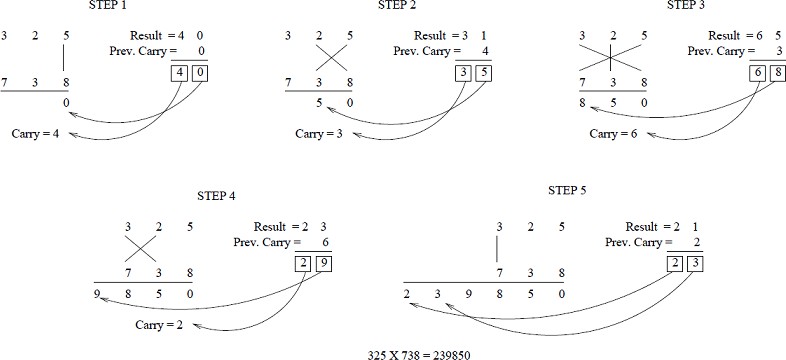


Figure 4.1: UT for a 3 *×* 3 Decimal Multiplication

The UT algorithm is based on a novel concept through which the generation of all partial products can be done with the ***concurrent*** addition of these partial products. The algorithm can be easily generalized for *n n* bit multiplication due to its highly modular structure. Since the partial products and their sums are calculated in parallel, the multiplier is ***independent of the clock frequency*** of the processor in case of a synchronous design. Thus the multiplier will require the less amount of time to calculate the product. The net advantage is that it reduces the need of microprocessors to operate at increasingly high clock frequencies. (*Vinay Kumar, 2009* )

*×*

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 oper- ating temperatures. By adopting the Vedic multiplier, microprocessor 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 this, layout can be made on a silicon chip easily. 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. (*Vinay Kumar, 2009* )

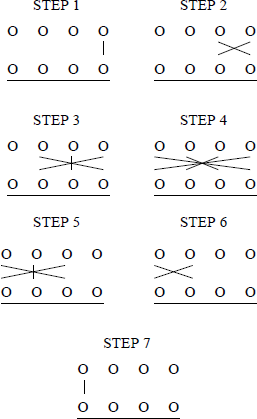


Figure 4.2: UT for a 4 *− bit* multiplication

## Carry Look Ahead Adder

The Carry Look Ahead (CLA) Adder generates carries before the sum is produced using the propagate and generate logic to make addition much faster. Thus the carry chain (the logic that propagates the carry through the full adders of the RCA) is separated from the sum logic (the part of the full adders that produce the sum).

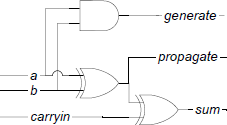


Figure 4.6: Partial Full Adder (PFA)

Consider a 4 *bit* CLA. There are two additional variables called the Generate (*G*) and Propagate (*P* ) which are fundamental for the CLA logic.

*−*

Let *A* and *B* be the two 4 *− bit* input variables.

*Gi* = *Ai.Bi* (3.11)

*Pi* = *Ai ⊗ Bi* (3.12)

*C*1 = *G*0 + *P*0*.C*0 (3.13)

*C*2 = *G*1 + *P*1*.C*1 = *G*1 + *P*1*.G*0 + *P*1*.P*0*.C*0 (3.14)

*C*3 = *G*2 + *P*2*.C*2 = *G*2 + *P*2*.G*1 + *P*2*.P*1*.G*0 + *P*2*.P*1*.P*0*.C*0 (3.15)

*C*4 = *G*3 + *P*3*.C*3 = *G*3 + *P*3*.G*2 + *P*3*.P*2*.G*1 + *P*3*.P*2*.P*1*.G*0 + *P*3*.P*2*.P*1*.P*0*.C*0 (3.16)

*Si* = *Pi ⊗ Ci* (3.17)

*GG* = *G*3 + *P*3*.G*2 + *P*3*.P*2*.G*1 + *P*3*.P*2*.P*1*.G*0 (3.18)

*PG* = *P*3*.P*2*.P*1*.P*0 (3.19)

These equations show that every carryout in the adder can be determined with just the input operands and initial carryin (*C*0). The size of a CLA adder block is chosen as 4 bits. An 8 *bit* CLA can be built from two 4 *bit* CLA blocks, a 16 *bit* CLA from four while a 32 *bit* CLA can be built from two 16 *bit* CLA blocks and so on with the help of the Group Generate (*GG*) and Group Propagate (*PG*) pins.

*− −*

*− − −*

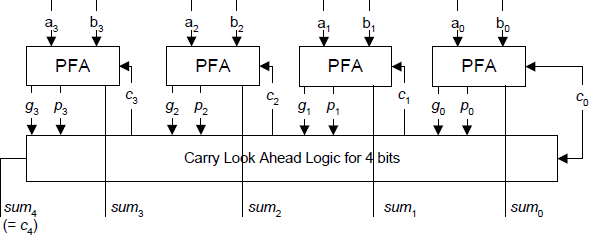


Figure 4.7: 4 *− bit* CLA

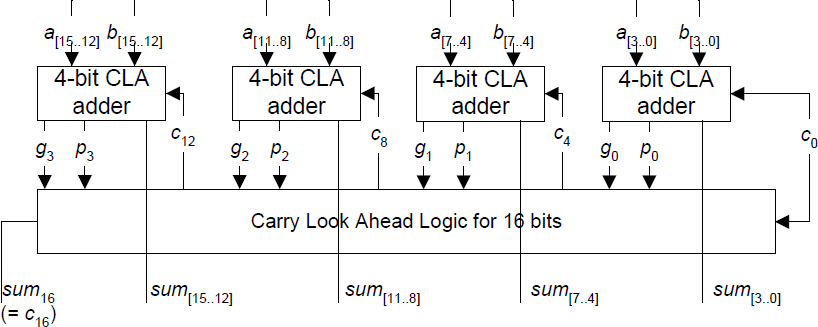


Figure 4.8: 16 *− bit* CLA adder from 4 *− bit* CLA adders

#### Critical Path Determination

Assuming that all gate delays are the same, the delay for a 4 *bit* CLA adder includes one gate delay to calculate the propagate and generate signals, two gate delays to calculate carry signals, and one gate delay to calculate the sum signals; i.e four gate delays. (*Michael Andrew Lai, 2002* )

*−*

For a 16 *bit* CLA adder there is one gate delay to calculate the propagate and generate signal (from the PFA), two gate delays to calculate the group propagate and generate in the first level of carry logic, two gate delays for the carryout signals in the second level of carry logic, and one gate delay for the sum signals. The second level of carry logic for the 16 *bit* CLA adder contributes an additional two gate delays over the 4 *bit* CLA adder, thus increasing the total to six gate delays. Hence,

*−*

*− −*

*CLA levels* (*groupsize* = 4) = *log*4*N* (3.20)

*CLA levels* (*groupsize* = *k*) = *logkN* (3.21)

*CLA gate delay* = 2 + 2*.log*4*N* (3.22)

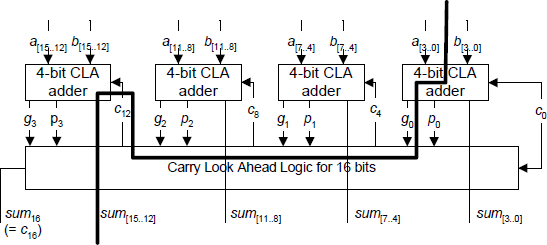


Figure 3.9: Critical Path of a 16 *− bit* CLA

In table 3.1, the delay of a CLA adder is **logarithmically dependent** on the size of the adder which theoretically results in one of the fastest adder architectures. And it has the regularity that will allow size adjustment of the adder without much additional design time. It is for these reasons that the CLA architecture is chosen as the adder after comparison with the Ripple Carry Adder (RCA), the Carry Skip Adder (CSKA) and the Linear Carry Select Adder (CSLA). Power consumption of the CLA might be slightly higher as compared to Kogge-Stone Adders, etc but the ease of scaling higher bit adders, the lower area and the lesser delay make this trade-off very slight. (*Michael Andrew Lai, 2002* )

Table 4.1: Adder Comparison – *N* : input size, *k*: group size

|  |  |  |  |
| --- | --- | --- | --- |
| Adder | Delay | Normalized Area | Normalized Design Time |
| RCA | *N* | 1 | 1 |
| CSKA | *N/k* | 1.14 | 2 |
| CLA | *logkN* | 1.88 | 8 |
| CSLA | *logkN*  (*N/k*) | 2.56 | 10 |

The Delay column expresses how the delay of the adder is proportional to the length (input size). The next column, Area, normalizes the area for the RCA (based on the subcells) and compares the relative sizes of the other adders to this normalized value. And finally, the Design Time column is an estimate of the normalized time required to design the particular adder based on the RCA design time. (*Michael Andrew Lai, 2002* )

# Chapter 5

**Design & Implementation**

## Synthesizable Code for Hardware Efficiency

The process of automatically converting the description in RTL to gates from the target technology is called ***Synthesis***. It converts the design from a higher level description to a lower one. Normal programming and HDL (Hardware Description Language) programming are usually built over the same platform C. But both are fundamentally different. HDL’s (eg. Verilog) are aimed at both simulation and synthesis of digital circuits. All descriptions can be simulated, but only some can be synthesized. And by changing the method of coding, the synthesized design can be made optimum.

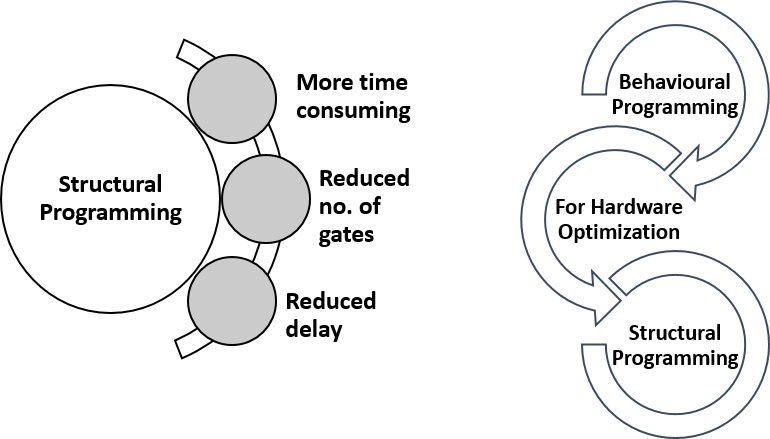


Figure 4.1: Structural Programming

Verilog has two major programming models - **Behavioural** and **Structural**, the former more useful for simulations and the latter for hardware level. However, structural mode is more time consuming and requires more effort. An overall plan is required for implementing structural codes. Developing code as the project progresses will not yield optimal results. But for making the codes truly synthesizable as well as for true hardware optimization, structural programming is a must.

Here, the design of each of the modules along with its implementation using various optimization methods and logical techniques is presented. Also, several new designs and original combinations of existing designs have also been implemented in order to increase the efficiency.

## Vedic UT

The basic unit of the 128 *bit* multiplier is the 4 *bit* Vedic multiplier built using the Urdhva Tiryakbhyam logic which was explained for decimal numbers in the previous chapter.

*− −*

Consider the inputs to be the 4 *− bit* numbers *A*[3 : 0] and *B*[3 : 0].

*A*[3 : 0] = *a*3*a*2*a*1*a*0 (4.1)

*B*[3 : 0] = *b*3*b*2*b*1*b*0 (4.2)

Then we have,

*c*0*p*0 = *a*0*b*0 (4.3)

*c*1*p*1 = *a*1*b*0 + *a*0*b*1 + *c*0 (4.4)

*c*2*p*2 = *a*2*b*0 + *a*1*b*1 + *a*0*b*2 + *c*1 (4.5)

*c*3*p*3 = *a*3*b*0 + *a*2*b*1 + *a*1*b*2 + *a*0*b*3 + *c*2 (4.6)

*c*4*p*4 = *a*3*b*1 + *a*2*b*2 + *a*1*b*3 + *c*3 (4.7)

*c*5*p*5 = *a*3*b*2 + *a*2*b*3 + *c*4 (4.8)

*c*6*p*6 = *a*3*b*3 + *c*5 (4.9)

Here, the *ci*’s can be multi-bit numbers while the *pi*’s are single-bit numbers. The additions are performed using CLA adders as mentioned before. The code is optimized for synthesis with power and delay in consideration. Higher order UT blocks are built according to the proposed scaling plan mentioned below.

## Scaling – The overall plan

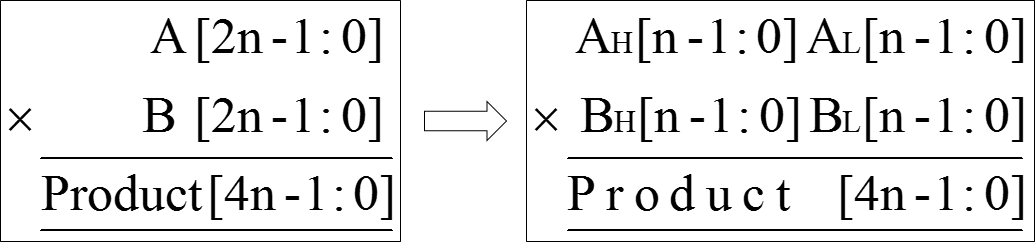


Figure 4.2: Multiplication of two 2*n − bit* numbers

An *n bit* multiplier is used to build a 2*n bit* multiplier. This uses the Karatsuba- Ofman Algorithm mentioned before, as well as innovative logic and coding techniques for maximum efficiency. After this is implemented, 8 *bit,* 16 *bit,* 32 *bit,* 64 *bit and* 128 *bit* multipliers can easily be built by substituting *n* = 4*,* 8*,* 16*,* 32 *and* 64 respectively starting with the basic 4 *− bit* UT multiplier as the building block. The *n − bit* additions are done using *n − bit* CLA’s.

*− −*

*− − − − −*

The scaling up procedure is as follows:

Given:

*A*[(*n −* 1) : 0] *× B*[(*n −* 1) : 0] *→ Prod*[(2*n −* 1) : 0]

To build:

n-bit multiplier

2n-bit multiplier

The input operands are split into higher order and lower order terms as shown.

*A*[(2*n −* 1) : 0] *× B*[(2*n −* 1) : 0] *→ Prod*[(4*n −* 1) : 0]

*AH* [(*n −* 1) : 0] = *A* [(2*n −* 1) : *n*] (4.10)

*AL* [(*n −* 1) : 0] = *A* [(*n −* 1) : 0] (4.11)

*BH* [(*n −* 1) : 0] = *B* [(2*n −* 1) : *n*] (4.12)

*BL* [(*n −* 1) : 0] = *B* [(*n −* 1) : 0] (4.13)

*X* = *AH × BH* (4.14)

*Y* = *AL × BL* (4.15)

*P* = *X* + *Y* (4.16)

*Z*1 = *AH* + *AL* (4.17)

*Z*2 = *BH* + *BL* (4.18)

*T* 1 = *Z*1 [(*n −* 1) : 0] *× Z*2 [(*n −* 1) : 0] (4.19)

*T* 2 = *Z*1 [(*n −* 1) : 0] *<< n.Z*2 [*n*] (4.20)

*T* 3 = *Z*2 [(*n −* 1) : 0] *<< n.Z*1 [*n*] (4.21)

*TX* = *T* 1 + *T* 2 (4.22)

*TY* = *TX* + *T* 3 (4.23)

*x* = *Z*1[*n*] *. Z*2[*n*] (4.24)

*y* = *x* + *TX* [2*n*] + *TY* [2*n*] (4.25)

*Z* = *{y, TY }* (4.26)

*R* = *Z* + *P* + 1 (4.27)

*Prod* = *{X, Y }* + (*R << n*) (4.28)

### Where have the optimizations taken place?

* + - 1. A more efficient 4 *− bit* Vedic Multiplier is developed in structural modelling
      2. The faster and efficient CLA adder is used throughout
      3. The traditional behavioural left shift operator (*<<*) is replaced with AND logic
      4. (2*n* + 1) *− bit* addition (which will have to be done by a 4*n − bit* adder) is converted to a 2*n − bit* addition and a single bit addition (Full adder)
      5. An efficient combiner logic is developed which requires only one addition and a concatenation

## UT Squaring

It can be seen easily that a special case of multiplication, i.e. Squaring, is much simpler in terms of hardware and software design as compared to the normal situations. This can be exploited to build the integrated multiplier.

For squaring a 4 *− bit* number, i.e *A*[3 : 0] = *B*[3 : 0]

*c*0*p*0 = *a*0 (4.29)

*c*1*p*1 = 2*.a*1*a*0 + *c*0 (4.30)

*c*2*p*2 = 2*.a*2*a*0 + *a*1*a*1 + *c*1 (4.31)

*c*3*p*3 = 2*.a*3*a*0 + 2*.a*2*a*1 + *c*2 (4.32)

*c*4*p*4 = 2*.a*3*a*1 + *a*2*a*2 + *c*3 (4.33)

*c*5*p*5 = 2*.a*3*a*2 + *c*4 (4.34)

*c*6*p*6 = *a*3*a*3 + *c*5 (4.35)

Thus, since multiplication by 2 is just a **left shift by 1 bit**, the square is a very special case that is easy to implement and which consumes less area, power and delay as compared to the normal Vedic multiplier.

### Scaling – Adapted for Squares

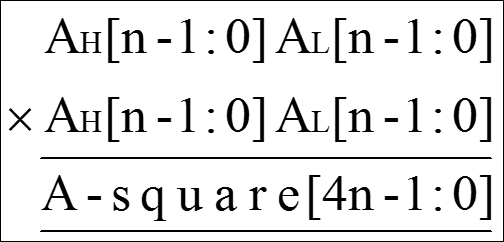


Figure 4.3: 2*n − bit* Squaring Unit

Building a 2*n − bit* squaring unit from an *n − bit* squaring unit is much easier than building a 2*n − bit* multiplier from an *n − bit* multiplier.

*AH* [(*n −* 1) : 0] = *A* [(2*n −* 1) : *n*] (4.36)

*AL* [(*n −* 1) : 0] = *A* [(*n −* 1) : 0] (4.37)

*X* = *AH*2 (4.38)

*Y* = *AL*2 (4.39)

*Z* = 2 (*AH × AL*) (4.40)

*Prod* = *{X, Y }* + (*Z << n*) (4.41)

As can be seen, the 15 steps in scaling the normal UT multiplier have been reduced to just 4 for scaling the UT square. Again, the multiplication by 2 is just a left shift by 1 bit as mentioned before.

## Chapter 6: Future Scope

The multiplier can be made more optimum by using different adders apart from the one proposed using CLA. This is possible because of its modularity. Each multiplier module can be made more optimum by designing a more efficient adder since almost every block depends on an adder. Better designed comparators can also highly optimize the integrated multiplier since the Logic unit makes use of comparators.

The multiplier can be extended for signed numbers (which just involves an XOR op- eration of the MSBs) as well as for floating point numbers. Globally synchronous, locally asynchronous or globally asynchronous, locally synchronous MAC units can be developed using this multiplier.

This multiplier can be fabricated after making the layout. Also, the results obtained from the RTL analysis can be made better by using more updated technology like 90 nm processes. Like in ASIC design, the multiplier can be customized for each constraint criterion with respect to specific applications.

There is always a huge demand for an optimized multiplier in the industry and it is truly amazing to see that the ancient centuries-old Vedic principles can be used to satisfy the present day demands. More research should be done to uncover these hidden facts which can reshape our future completely.

# Bibliography

* + 1. Ajinkya Kale, Shaunak Vaidya, Ashish Joglekar, A Generalized Recursive Algorithm for Binary Multiplication based on Vedic Mathematics
    2. Devika Jaina, Kabiraj Sethi, Rutuparna Panda, Vedic Mathematics based Multiply Accumulate Unit, 2011 International Conference on Computational Intelligence and Communication Systems
    3. G.Ganesh Kumar, V.Charishma, Design of High Speed Vedic Multiplier using Vedic Mathematics Techniques, International Journal of Scientific and Research Publications,

Volume 2, Issue 3, March 2012

* + 1. Harish Kumar, Implementation and Analysis of power, area and delay of Array, Urdhva and Nikhilam Vedic Multipliers, International Journal of Scientific and Research Publications, Volume 3, Issue 1, January 2013
    2. Harpreet Singh Dhillon and Abhijit Mitra, A Reduced-Bit Multiplication Algorithm for Digital Arithmetic, World Academy of Science, Engineering and Technology 19, 2008
    3. 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
    4. Honey Durga Tiwari, Ganzorig Gankhuyag, Chan Mo Kim, Yong Beom Cho, Multiplier design based on ancient Indian Vedic Mathematics, 2008 IEEE International SOC Design Conference
    5. Jagadguru Swami Sri Bharath, Krishna Tirathji, Vedic Mathematics or Sixteen Simple Sutras from The Vedas, Motilal Banarsidas, Varanasi (India), 1992.
    6. Jan M. Rabaey, Low power design essentials
    7. Kabiraj Sethi, Rutuparna Panda, An Improved Squaring Circuit for Binary Numbers, International Journal of Advanced Computer Science and Applications, Vol. 3, No.2, 2012
    8. Manoranjan Pradhan, Rutuparna Panda, Sushanta Kumar Sahu, Speed Comparison of 16 16 Vedic Multipliers, International Journal of Computer Applications (0975 8887), Volume 21 No.6, May 2011

*×*

* + 1. Michael Andrew Lai, 2002, Arithmetic units for a high performance digital signal processor, B.S. (University of California, Davis), Thesis report
    2. Mohammed Hasmat Ali, Anil Kumar Sahani, June 2013 Study, Implementation and Comparison of Different Multipliers based on Array, KCM and Vedic Mathematics Using EDA Tools, International Journal of Scientific and Research Publications,

Volume 3, Issue 6

* + 1. M. Ramalatha, K. Deena Dayalan, P. Dharani, S. Deborah Priya, High Speed Energy Efficient ALU Design using Vedic Multiplication Techniques, ACTEA 2009, July 15-17, 2009 Zouk Mosbeh, Lebanon
    2. Pavan Kumar U.C.S, Saiprasad Goud A, A.Radhika, FPGA Implementation of High Speed 8-bit Vedic Multiplier Using Barrel Shifter, International Journal of Emerging Technology and Advanced Engineering, Volume 3, Issue 3, March 2013
    3. Prabir Saha, Arindam Banerjee , Partha Bhattacharyya , Anup Dandapat, 2011. High Speed ASIC Design of Complex Multiplier Using Vedic Mathematics, IEEE Students’

Technology Symposium, IIT Kharagpur

* + 1. Ramachandran.S, Kirti.S.Pande, Design, Implementation and Performance Analysis of an Integrated Vedic Multiplier Architecture, International Journal Of Computational Engineering Research, May-June 2012, Vol. 2. Issue No.3, 697-703
    2. Shri Prakash Dwivedi, 2013, An Efficient Multiplication Algorithm Using Nikhilam Method, (arxiv:1307.2731v5)
    3. S. Deepak and Binsu J Kailath, 2012, Optimized MAC unit design, (2012) IEEE EDSSC 2012, IEEE International Conference on Electron Devices and Solid-State Circuits held from 3-5 Dec. 2012
    4. Vinay Kumar, 2009 Analysis, Verification and FPGA Implementation Of Vedic Multiplier With Bist Capability, Thapar University