A Mini Project Report

On

# 16 BIT HARDWARE MULTIPLIER

**Department of Electronics & Telecommunication Engineering**

Submitted By

**01 Divya Mokal**

**05 Adarsh Nambiar**

**11 Prasoon Pandey**

Supervisor

**Rajendrakumar Khade**



**Department of Electronics & Telecommunication Engineering**

**PILLAI COLLEGE OF ENGINEERING**

**New Panvel – 410 206**

**UNIVERSITY OF MUMBAI**

**Academic Year 2021 – 22**

**Department of Electronics & Telecommunication Engineering**

**PILLAI COLLEGE OF ENGINEERING**

**New Panvel – 410 206**

CERTIFICATE

This is to certify that the following students:

**Name Roll No.**

Divya Mokal B01

Adarsh Nambiar B05

Prasoon Pandey B11

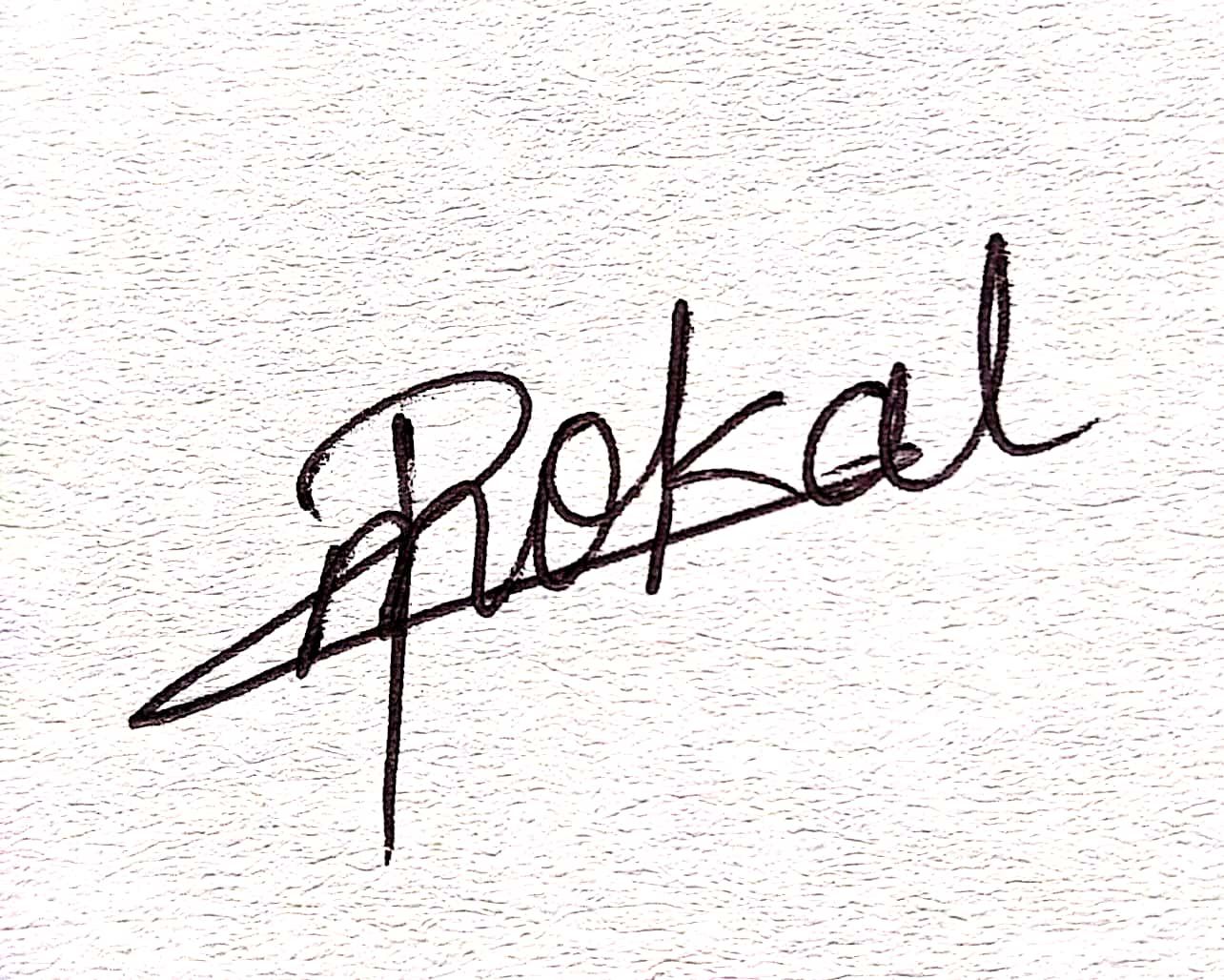
have satisfactorily carried out the Mini Project 2B work entitled “16 BIT HARDWARE MULTIPLIER“ for the semester VI of Bachelor of Engineering in Electronics & Telecommunication Engineering of Mumbai University during the Academic Year 2021 – 2022.

|  |  |  |
| --- | --- | --- |
| **Guide** |  | **Head of Department** |
| **(Dr. R. H. Khade)** |  | **Dr. Avinash Vaidya** |
|  |  |  |
|  |  |  |
| **Internal Examiner** |  | **External Examiner** |

**Declaration**

We declare that this written submission for Mini Project 2B entitled “16 BIT HARDWARE MULTIPLIER” represents our ideas in our own words and where others' ideas or words have been included, we have adequately cited and referenced the original sources. We also declare that we have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any ideas / data / fact / source in our submission. We understand that any violation of the above will cause disciplinary action by the institute and also evoke penal action from the sources which have thus not been properly cited or from whom paper permission has not been taken when needed.

**Project Group Members:**

Divya Mokal 

Adarsh Nambiar:

Prasoon Pandey:

Date:

Place:

iii

**Table of Contents**

| Abstract................................................................................................................................ | | | | i |
| --- | --- | --- | --- | --- |
| List of Figures...................................................................................................................... | | | | ii |
| List of Tables....................................................................................................................... | | | | iii |
| **1.** | Introduction................................................................................................................. | | | 1 |
|  | **1.1** | Fundamentals................................................................................................... | | 1 |
|  | **1.2** | Objectives........................................................................................................ | | 1 |
|  | **1.3** | Scope............................................................................................................... | | 2 |
|  | **1.4** | Outline……………………………….............................................................. | | 2 |
| **2.** | Literature Survey......................................................................................................... | | | 3 |
|  | **2.1** | Introduction……. ……………………............................................................ | | 3 |
|  | **2.2** | Literature Review ………………………………………................................ | | 4 |
|  | **2.3** | Summary of Literature Survey.…………………………................................ | | 4 |
| **3.** | Theory *(rename it as per your project domain)*.......................................................... | | | 5 |
|  | **3.1** | Overview…………………….......................................................................... | | 6 |
|  |  | 3.1.1 | Existing System Architecture………………………………………. | 8 |
|  |  | 3.1.2 | Proposed System Architecture……………………………………… | 9 |
|  | **3.2** | Requirements for Implementation ………...................................................... | | 10 |
|  |  | 3.2.1 | Algorithms / Techniques….................................................................. | 10 |
|  |  | 3.2.2 | Use Case Diagram / Activity Diagram…........................................... | 10 |
|  |  | 3.2.3 | Sample Dataset…............................................................................... | 11 |
|  |  | 3.2.4 | Hardware and Software Specifications…........................................... | 11 |
|  | **3.3** | Evaluation Metrics........................................................................................... | | 12 |
| **4.** | Applications……………………................................................................................. | | | 13 |
|  | **4.1** | Social …………….......................................................................................... | | 13 |
|  | **4.2** | Technical …..……........................................................................................... | | 14 |
| **5.** | Summary….................................................................................................................. | | | 15 |
| References............................................................................................................................ | | | | 16 |
| Acknowledgement……………………………………………………………………......... | | | | 17 |

## **Abstract**

Multiplication and square are elementary mathematical operations extremely important for the core computing process. Also, exponentiation, the process of raising a base number to a power is an important operation in many numerical computations. To keep pace with the technology, high-speed applications require faster methods of multiplication and square architecture. This paper reports a new faster algorithm for multiplication and square based on Verilog code. The design for the architecture of 16-bit multiplier and square are proposed and described using VHDL hardware description language. The code description is simulated using ISE Xilinx. The synthesis showed reduced time delay for the multiplier and square.

vi

**List of Figures**

| Figure 1 | 2X2 multiplier | 1 |
| --- | --- | --- |
| Figure 2 | Line diagrams for four bits. | 13 |
| Figure 3 | 8 X 8 Bits multiplier | 14 |
| Figure 4 | 16 X 16 Bits multiplier | 15 |
| Figure 5 | 8 X 8 Bits square | 16 |

| Figure 6 | 16 X 16 Bits square | 1 |
| --- | --- | --- |
| Figure 7 | Algorithms | 13 |
| Figure 8 | Activity Diagram | 14 |
| Figure 9 | 16 X 16 Bits multiplier | 15 |
| Figure 10 | 8 X 8 Bits square | 16 |

vii

**Chapter 1**

**Introduction**

**1.1 Fundamentals**

There are several algorithms for multiplication such as: Booth, carry-save, array, modified Booth and Wallace tree. A large number of possible architectures have been developed in accordance with these algorithms indicating good performance efficiency. In an array multiplier, a combinational circuit is utilized to multiply two binary numbers. The architecture resembles an array which is also an efficient layout of combinational architecture. All of the product bits are obtained simultaneously resulting in a faster method. However, it requires a large number of gates and for this reason it is less economical. Another method to have improved efficiency of multipliers is the arrangement of adders, that is, tree method. There are two algorithms based on this method: carry-save array (CSA) and Wallace tree. In the CSA method, bits are added one by one and give the carry input signal to another adder at one bit higher position, thus forming the shape of a tree.

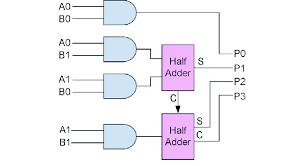


Fig 1 2X2 Multiplier

**1.2 Objectives**

Our project aims to make an improvement in the multiplier architecture and is therefore in need for high speed applications.

The main objectives of the system are as follows :

* To improve the efficiency in both speed and area
* To reduce the time delay in the proposed multiplier design
* To generate computer systems whose power consumption is less.
* To develop a design, which requires enhanced computation capabilities at low cost

**1.3 Scope**

These projects can be updated and modified making it more useful to implement. Various systems can be created to increase the bit width and process large quantities of data. It can also make improvements in overall power consumption and area taken by adopting the latest design of 90nm,45nm and 18nm technologies. Analyze the parasitic elements in our design and eliminate them to minimize leakage power and delays. Apply in real time working scenarios and prove its accuracy and precision.Improve the design and promote the fabrication of IC and PC.

**1.4 Outline**

This paper presents a novel architecture for16-bit hardware multiplier. This paper is organized in the following way: Section 2 presents related work done on the implementation of a 16-bit hardware multiplier. Section 3 provides a brief description of the methods used to implement the project and the proposed architecture. Section 4 provides an overview of the applications of hardware multipliers. Section 5 shows a brief summary of the overall project.

**Chapter 2**

**Literature Survey**

**[1]** In this paper, a high speed and low power 16x16 Vedic Multiplier is designed by using low power and high speed modified carry select adder. Modified Carry Select Adder employs a newly incremented circuit in the intermediate stages of the Carry Select Adder (CSA)

**[2]** This paper reports a new faster algorithm for multiplication and square based on ancient Indian mathematics, called Vedic Mathematics is proposed and described using VHDL

**[3]** This paper suggests that a . A VHDL designed architecture based on booth multiplication algorithm is proposed which not only optimize speed but also efficient on energy use

[4] This paper describes the design and development of a Hybrid Final Adder using Multiplexer Binary Excess Converters (MBEC) which further accelerates the speed and consumes less area

**[5]** This paper proposes the performance of Multiplier is the major component for processing of large amount of data in DSP applications, using different recoding schemes in Fused Add-Multiply (FAM)

**[6]** This paper depicts that Recently Low Power and High Speed of real-time computing is necessary for applications like image processing, neural network, IOT and DSP.

**[7]** This paper gives an idea about the device combines a powerful 16-bit RISC microcontroller and integrated peripherals including multiple high-speed interfaces and flexible port I/O.

**[8]** In this paper, multiplication of the floating point numbers described in IEEE 754 single precision floating point multiplier is done using VHDL

**[9]** This paper analyses two approximate leading one detector (LOD) designs and an approximate adder that can be used to improve the hardware efficiency of the Mitchell logarithmic multiplier.

**[10]** This paper discusses the reversible logic gate algorithm will reduce the garbage bits and logical components during arithmetic manipulations

**2.3 Literature Summary**

Since most of these devices are required to run efficiently, their performance might take a hit as these are used in devices with limited power supply, mainly battery powered devices. also these are limited to 16 bit inputs, any higher inputs require separate multipliers affecting the overall speed and performance of the system. but they are very much suited to low power devices where reliable outputs are preferred over speedy performance.

**Chapter 3**

**Proposed System**

# 3.1.1 Overview

**Multiplication Method**

One of the aphorisms of Vedic Mathematics implied for multiplication is Urdhva Tiryakbhyam (Vertical and Crosswise) which is also the foundation of the proposed design. It 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 parallelism in the generation of partial products and their summation is obtained by vertical and crosswise multiplication and addition. According to this algorithm, 4\*4 bit multiplication can be carried out in the following way: Firstly, the least significant bits are multiplied which gives us 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 the LSB of the multiplier and the next higher bit of the multiplicand (crosswise). The sum gives a second bit of the product and the carry is added in the output of the next stage sum obtained by the crosswise and vertical multiplication and addition of three bits of the two numbers from a 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. To make the methodology clear, an alternate illustration is given with the help of line diagrams in figure 1 where the dots represent bit ‘0’ or ‘1’.

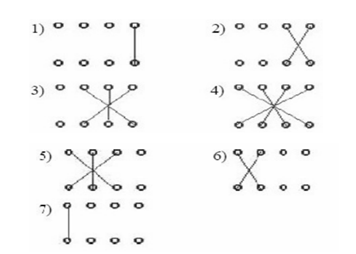


Fig 2 Line diagrams for four bits.

According to this example, the digits on the two sides of the line are multiplied and the result is added to the previous carry. When more than one line is in the step, all the results are added with the previous carry and the process is thus continued. Initially, the previous carry is equal to zero. A unit place digit of the addition result is one of the digits in the answer; this is derived from full multiplication, while the remaining digits act as a carrier. If the numbers of the digits are not the same in the multiplier and multiplicand, then the bigger number has to be determined. The number of digits then needs to be counted. The smaller number should be prepended with 0s so that both numbers will be of the same digits.

**Square Method**

In most of the computations, the multiplier unit is used to compute the square of an operand. Since the square is a special case of multiplication, dedicated square hardware will significantly improve the computation time. The squaring algorithm makes use of the Duplex or Dwandwa (D) operator. 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 an odd number of bits in the original sequence there is one bit left by itself in the middle, and this enters as its square.

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 + the 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.

# 3.1.2 Proposed System Architecture

The basic building blocks of this multiplier are one-bit multipliers and adders. One bit multiplication can be performed through a two-input AND gate and for addition, a full adder can be utilized. The 8x8-bit multiplier can be structured using 4X4 bit blocks. 8-bit multiplicand A can be decomposed into pairs of 4 bits AH-AL. Similarly, multiplicand B can be decomposed into BH-BL. The 16-bit product can be written as:

P= A x B= (AH-AL) x (BH-BL) = AH x BH+AH x BL + AL x BH+ AL x BL

The outputs of 4X4 bit multipliers are added accordingly to obtain the final product. Thus, in the final stage, two adders are also required

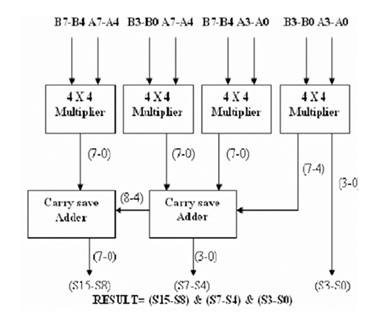


Fig 3 8 X 8 Bits multiplier

Architecture for 16x16 bit multiplier. The 16X16-bit multiplier can be structured using 8X8-bit blocks as shown in Figure 3. In this Figure 3.2, the 16-bit multiplicand A can be decomposed into pairs of 8 bits AH-AL. Similarly, multiplicand B can be decomposed into BH-BL.

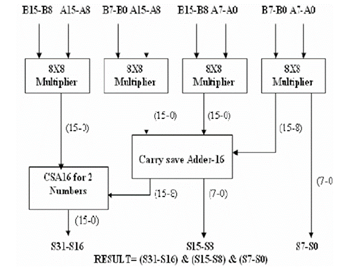


Fig 4 16 X 16 Bits multiplier

The outputs of 8X8 bit multipliers are added accordingly to obtain the 32 bits final product. Thus, in the final stage, two adders are also required.

**Architecture for 8x8-bit square**

The 8X8 bit square can be structured using 4X4 bit blocks as shown in Figure 4. In this Figure 3.2.3, the 8-bit multiplicand A can be decomposed into pairs of 4 bits AHAL.

The 16 bit product can be written as:

P = A x A= (AH-AL) x (AH-AL) = AH x AH + 2\*(AL x AH) + AL x AL

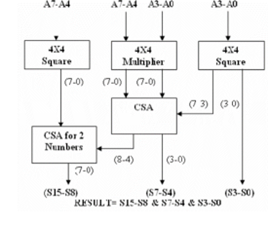


Fig 5 8 X 8 Bits square

The outputs of the 4X4 bit multiplier and squares are added accordingly to obtain the final product of 16 bits. Thus, in the final stage, two carry-save adders are also required.

**Architecture for 16x16-bit square**

The 16X16 bit square can be structured using 8X8 bit blocks as shown in Figure 3.5. In this the 16-bit multiplicand A can be decomposed into a pair of 4 bits AH-AL.

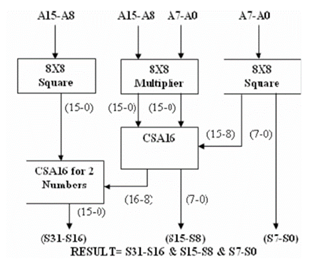


Fig 6 16 X 16 Bit square

3.2 **Requirements for implementation**

**3.2.1 Algorithms**

For the following project 16 bit hardware multiplier we have implemented using the following code:

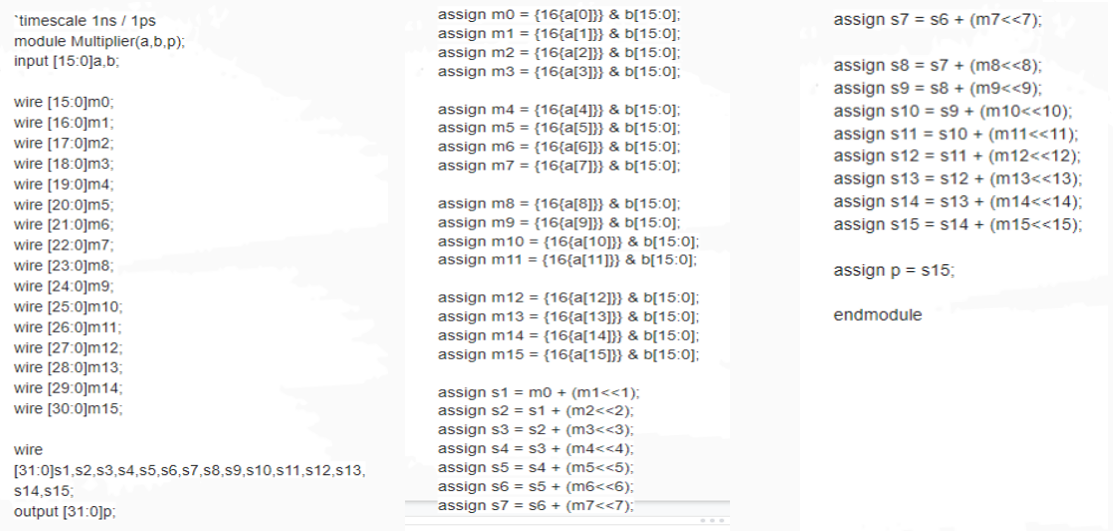


Fig 7

**3.2.2 Use Case Diagram / Activity Diagram**

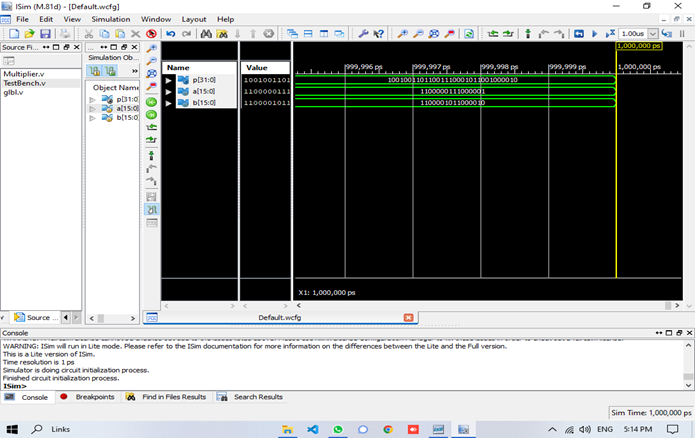


Fig 8

**3.3.4 Software Specifications**

Xilinx ISE (Integrated Synthesis Environment) is a discontinued software tool from Xilinx for synthesis and analysis of HDL designs, which primarily targets development of embedded firmware for Xilinx FPGA and CPLD integrated circuit (IC) product families. It was succeeded by Xilinx Vivado. Use of the last released edition from October 2013 continues for in-system programming of legacy hardware designs containing older FPGAs and CPLDs otherwise orphaned by the replacement design tool, Vivado Design Suite. ISE enables the developer to synthesize ("compile") their designs, perform timing analysis, examine RTL diagrams, simulate a design's reaction to different stimuli, and configure the target device with the programmer. Other components shipped with the Xilinx ISE include the Embedded Development Kit (EDK), a Software Development Kit (SDK) and ChipScope Pro. The Xilinx ISE is primarily used for circuit synthesis and design, while ISIM or the ModelSim logic simulator is used for system-level testing.

**Chapter 4**

**Applications**

A hardware multiplier is an electronic circuit used in digital electronics, such as a computer, to multiply two binary numbers.

A variety of computer arithmetic techniques can be used to implement a digital multiplier. Most techniques involve computing the set of partial products, which are then summed together using binary adders. This process is similar to long multiplication, except that it uses a base-2 (binary) numeral system.

Older multiplier architectures employed a shifter and accumulator to sum each partial product, often one partial product per cycle, trading off speed for die area. Modern multiplier architectures use the (Modified) Baugh–Wooley algorithm, Wallace trees, or Dadda multipliers to add the partial products together in a single cycle. The performance of the Wallace tree implementation is sometimes improved by modified Booth encoding one of the two multiplicands, which reduces the number of partial products that must be summed. For speed, shift-and-add multipliers require a fast adder (something faster than ripple-carry). A "single cycle" multiplier (or "fast multiplier") is pure combinational logic.

In a fast multiplier, the partial-product reduction process usually contributes the most to the delay, power, and area of the multiplier. For speed, the "reduce partial product" stages are typically implemented as a carry-save adder composed of compressors and the "compute final product" step is implemented as a fast adder (something faster than ripple-carry).

Many fast multipliers use full adders as compressors ("3:2 compressors") implemented in static CMOS. To achieve better performance in the same area or the same performance in a smaller area, multiplier designs may use higher-order compressors such as 7:3 compressors; implement the compressors in faster logic (such transmission gate logic, pass transistor logic, domino logic); connect the compressors in a different pattern, or some combination.

**4.1 Booth's multiplication algorithm**

Booth's multiplication algorithm is a multiplication algorithm that multiplies two signed binary numbers in two's complement notation. The algorithm was invented by Andrew Donald Booth in 1950 while doing research on crystallography at Birkbeck College in Bloomsbury, London. Booth's algorithm is of interest in the study of computer architecture.

Booth's algorithm examines adjacent pairs of bits of the 'N'-bit multiplier Y in signed two's complement representation, including an implicit bit below the least significant bit, y−1 = 0. For each bit yi, for i running from 0 to N − 1, the bits yi and yi−1 are considered. Where these two bits are equal, the product accumulator P is left unchanged. Where yi = 0 and yi−1 = 1, the multiplicand times 2i is added to P; and where yi = 1 and yi−1 = 0, the multiplicand times 2i is subtracted from P. The final value of P is the signed product.

The representations of the multiplicand and product are not specified; typically, these are both also in two's complement representation, like the multiplier, but any number system that supports addition and subtraction will work as well. As stated here, the order of the steps is not determined. Typically, it proceeds from LSB to MSB, starting at i = 0; the multiplication by 2i is then typically replaced by the incremental shifting of the P accumulator to the right between steps; low bits can be shifted out, and subsequent additions and subtractions can then be done just on the highest N bits of P.[2] There are many variations and optimizations on these details.

The algorithm is often described as converting strings of 1s in the multiplier to a high-order +1 and a low-order −1 at the ends of the string. When a string runs through the MSB, there is no high-order +1, and the net effect is interpreted as a negative of the appropriate value.

**4.2 Wallace tree**

A Wallace multiplier is a hardware implementation of a binary multiplier, a digital circuit that multiplies two integers. It uses a selection of full and half adders (the Wallace tree or Wallace reduction) to sum partial products in stages until two numbers are left. Wallace multipliers reduce as much as possible on each layer, whereas Dadda multipliers try to minimize the required number of gates by postponing the reduction to the upper layers.

Wallace multipliers were devised by the Australian computer scientist Chris Wallace in 1964.The Wallace tree has three steps. Multiply each bit of one of the arguments, by each bit of the other.Reduce the number of partial products to two by layers of full and half adders.Group the wires in two numbers, and add them with a conventional adder.

Compared to naively adding partial products with regular adders, the benefit of the Wallace tree is its faster speed. It has {\displaystyle O(\log n)}O(\log n) reduction layers, but each layer has only {\displaystyle O(1)}O(1) propagation delay. A naive addition of partial products would require {\displaystyle O(\log ^{2}n)}O(\log^2n) time. As making the partial products is {\displaystyle O(1)}O(1) and the final addition is {\displaystyle O(\log n)}O(\log n), the total multiplication is {\displaystyle O(\log n)}O(\log n), not much slower than addition. From a complexity-theoretic perspective, the Wallace tree algorithm puts multiplication in the class NC1. The downside of the Wallace tree, compared to the naive addition of partial products, is its much higher gate count.

These computations only consider gate delays and don't deal with wire delays, which can also be very substantial. The Wallace tree can be also represented by a tree of 3/2 or 4/2 adders. It is sometimes combined with Booth encoding.

**Chapter 5**

**Summary**

In this report, the study and implementation of a 16 bit hardware multiplier is presented. The different techniques such as the Multiplication method and Square architecture is explained with this project.The comparative study of various techniques like Booth's multiplication algorithm and Wallace tree method is presented in this report.Since in performing multiplication a computer spends a considerable amount of its processing time, an improvement in the speed for performing multiplication is highly required. The speed improvements are gained by parallelizing the generation of partial products with their concurrent summations

**References**

1. Santoro, M.R., and Horowitz., M.: "Spim : A Pipelined 64 x 64 -Bit Iterative Multiplier", IEEE J. Solid-State Circuits,1989,24,pp.487-493.
2. Ciminiera, L, and Valenzano, A.:" Low cost Serial Multiplier of High Speed Specialized Processors", IEE Proc. E. 1988, 135,(5), pp-259-265.
3. Chidgupkar, P. D. and Karad, M.T.,”The Implementation of VedicAlgorithms in Digital Signal Processing”, Global Congress on Engineering Education, Vol. 8, No.2, 2004.
4. Thapliyal, H. and Arabnia H.R., “A time-Area-Power Efficient Multiplier and Square Architecture Based on Ancient Indian Vedic Mathematics”.

5. Singh, B., Kumar, L. and Rana, D.R., “Design and Hardware Implementation of 8 bit \* 8 bit Multiplication Algorithm”.

6. Sjoholm, S. and Lindh, L., “VHDL for Designers, Prentice- Hall PTR (1997).

7. Jagadguru Swami Sri Bharati Krisna Tirthaji Maharaja, Vedic Mathematics: Sixteen Simple Mathematical Formulae from the Veda. Delhi (1965).

8. VedicMaths.org (2004) http://www.vedicmaths.org

9. Deschamps, Jean-Pierrie and Sutter, D. Gustavo, Synthesis of Arithmetic Circuits, FPGA, ASIC and Embedded Systems, John Wiley & sons Inc. Publication (2006).

10. Himanshu Thapliyal and Vishal Verma ,“High Speed Efficient Signed/Unsigned N X N Bit Multiplier Based On Ancient Indian Vedic Mathematics” proceedings of the 7th IEEE VLSI Design & Test Workshop, Bangalore, August 2003. **Acknowledgement**

I would like to convey my heartfelt gratitude to our Guide Rajendrakumar Khade for her tremendous direction and assistance in the completion of my project.

I would also like to thank our HOD Dr Avinash Vidya, for providing me with this wonderful opportunity to work on a project. This project would not have been accomplished without their help and insights.

I would also like to thank my friends who helped me in finishing this project. It helped me increase my knowledge and skill

Divya Mokal

Adarsh Nambiar

Prasoon Pandey