# DESIGN AND IMPLEMENTATION OF A MULTIPLIER USING REGULAR PARTIAL PRODUCT

A dissertation submitted in partial fulfillment of the requirements

for the award of degree of

**MASTER OF TECHNOLOGY**

**In**

**VLSI Design**

Submitted By

**BIPIN**

Roll No. 601161004

Under guidance of

**Ms. Sakshi**

**Assistant Professor, ECED**

**T.U, Patiala**

**Department of Electronics and Communication Engineering**

**THAPAR UNIVERSITY, PATIALA**

**July 2013**

# DECLARATION

I hereby declare that the work which is being presented in the dissertation entitled, **"Design and Implementation of Multiplier using regular partial product"** in partial fulfillment of the requirement for the award of degree of Master of Technology in VLSI Design submitted in Electronics and Communication Engineering Department of Thapar University, Patiala, is an authentic record of my own work carried out under the supervision of Ms. Sakshi, Assistant Professor, ECED and refers other researcher's work which are duly listed in the reference section.

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

**Date:** 11/07/13

**(BIPIN)**

**Roll No: 601161004**

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

**(Ms. Sakshi)**

**Assistant Professor**

**ECED,Thapar University**

**Countersigned by:**

**Head**

**ECED, Thapar University**

**Patiala-147004**

**Dean of Academic Affairs**

**Thapar University**

**Patiala- 147004**

# ACKNOWLEDGEMENT

To discover, analyze and to present something new is to venture on an untrodden path towards and unexplored destination is an arduous adventure unless one gets a true torchbearer to show the way. I would have never succeeded in completing my task without the cooperation, encouragement and help provided to me by various people. Words are often too less to reveals one's deep regards. I take this opportunity to express my profound sense of gratitude and respect to all those who helped me through the duration of this dissertation. I acknowledge with gratitude and humility my indebtedness to **Ms. Sakshi, Assistant Professor,** Electronics and Communication Engineering Department, Thapar University, Patiala, under whose guidance I had the privilege to complete this dissertation. I wish to express my deep gratitude towards her for providing individual guidance and support throughout the dissertation work.

I convey my sincere thanks to **Head of the Department, Dr. Rajesh Khanna** as well as **PG Coordinator, Dr. Kulbir Singh, Assistant Professor,** Electronics and Communication Engineering Department, entire faculty and staff of Electronics and Communication Engineering Department for their encouragement and cooperation.

My greatest thanks are to all who wished me success especially my parents. Above all I render my gratitude to the Almighty who bestowed self-confidence, ability and strength in me to complete this work for not letting me down at the time of crisis and showing me the silver lining in the dark clouds. I do not find enough words with which I can express my feelings of thanks to my dear friends for their help, inspiration and moral support which went a long way in successful competition of the present study.

**(BIPIN)**

# ABSTRACT

The conventional Modified Booth Encoding (MBE) generates n/2+1 rows instead of n/2 rows and also irregular partial product array because of the extra partial product bit at the LSB position of each partial product row. In this brief, a simple approach is proposed to generate n/2 partial product rows along with a regular partial product arrays and negligible overhead, thereby lowering the complexity of partial product and reducing the area and power of MBE multipliers. The proposed approach can also be utilized to regularize the partial product array of MBE multipliers along with the issue of disposal of the negative partial products efficiently by computing the 2's complement thereby avoiding the additional adder for adding 1 and generation of long carry chain. The proposed mechanism also continues to support the concept of reducing the partial product from n/2 +1 partial products achieved via modified booths algorithm to n/2.In this direct two's complement method has been used to reduce partial product rows from n/2+1 to n/2.Implementation results demonstrate that the proposed MBE multipliers with a regular partial product array really achieve significant improvement in area and power consumption when compared with conventional MBE multipliers. Along with this these regularized multipliers are designed with different adders such carry select adder , carry lookahed adder and ripple carry adder and then compared on the basis of area, power and delay.

These multipliers have been designed with the help of Verilog, simulated on Modelsim SE 6.3f and synthesized on Xilinx 13.1, that helps in comparing there area , power and delay.

# TABLE OF CONTENTS

# LIST OF FIGURES

# LIST OF TABLES

# ABBREVIATIONS

| | |
|---|---|
| CLBs | Configurable logic blocks |
| CSA | Carry save adder |
| DSP | Digital signal processing |
| FA | Full adder |
| HDL | Hardware Description Language |
| IEEE | Institute of Electrical and Electronics Engineers |
| ISE | Integrated Software Environment |
| LSB | least significant bit |
| MAC | Multiply and accumulate |
| MBA | Modified Booth's Algorithm |
| MBE | Modified Booth Encoding |
| PP | Partial Products |
| RCA | Ripple-carry Adder |
| VLSI | Very Large Scale Integration |
| W.R.T | With Respect To |

# EQUATIONS

# CHAPTER 1
# INTRODUCTION

Multiplication is a mathematical operation that at its simplest is an abbreviated process of adding an integer a specified number of times. Multiplication is the fundamental arithmetic operation important in several processors and digital signal processing systems. Multiplication of two k bit number needed multi operand addition process that can be realized in k cycles of shifting and addition with hardware, firmware or software. Multiplication based operations such as multiply and accumulate (MAC) and inner product are among some of the frequently used intensive arithmetic functions currently implemented in many digital signal processing (DSP) applications such as convolution, fast fourier transform(FFT),filtering and in microprocessors in its arithmetic and logic unit.

Portable multimedia and digital signal processing (DSP) systems, which typically require low power consumption, short design cycle, and flexible processing ability, have become increasingly popular over the past few years. As many multimedia and DSP applications are highly multiplication intensive so that the performance and power consumption of these systems are dominated by multipliers. Unfortunately, portable devices mostly operate with stand-alone batteries, but multipliers consumes large amount of power. Digital signal processing systems need multiplication algorithms to implement DSP algorithms such as filtering where the multiplication algorithm is directly within the critical path. Along with signal processing applications, multimedia, and 3D graphics, performance, in most cases, strongly depends on the effectiveness of the hardware used for computing multiplications, since multiplication is, besides addition, massively used in these environments. Consequently, it's greatly imperative to develop power-efficient multipliers to compose a high-performance and low-power portable multi-media and DSP system.

As the scale of integration keeps growing, more and more sophisticated signal processing systems are being implemented on a VLSI chip. These signal processing applications not only demand great computation capacity but also consume considerable amount of energy. While speed and area remain to be the two major design tools. The higher speed results to enlarged power consumption, thus, low power

architectures will be the choice of the future. The need for low-power VLSI system arises from two main forces. First, with the steady growth of operating frequency and processing capacity per chip, large currents have to be delivered and the heat due to large power consumption must be removed by proper cooling techniques. Second, battery life in portable electronic devices is limited. Low power design directly leads to prolonged operation time in these portable devices. This has given way to the growth of new circuit algorithms, with the plan of reducing the power consumption of multiplication algorithms with having high-speed structures and appropriate performance.The multiplier is fairly large block of a computing system. The size of multiplier is directly proportional to the square of its resolution i.e. size of multiplier.

### 1.1 Classification of multiplier

There are two kinds of multiplier as shown in fig. 1.1

a) Serial multiplication algorithms

b) Parallel multiplication algorithms

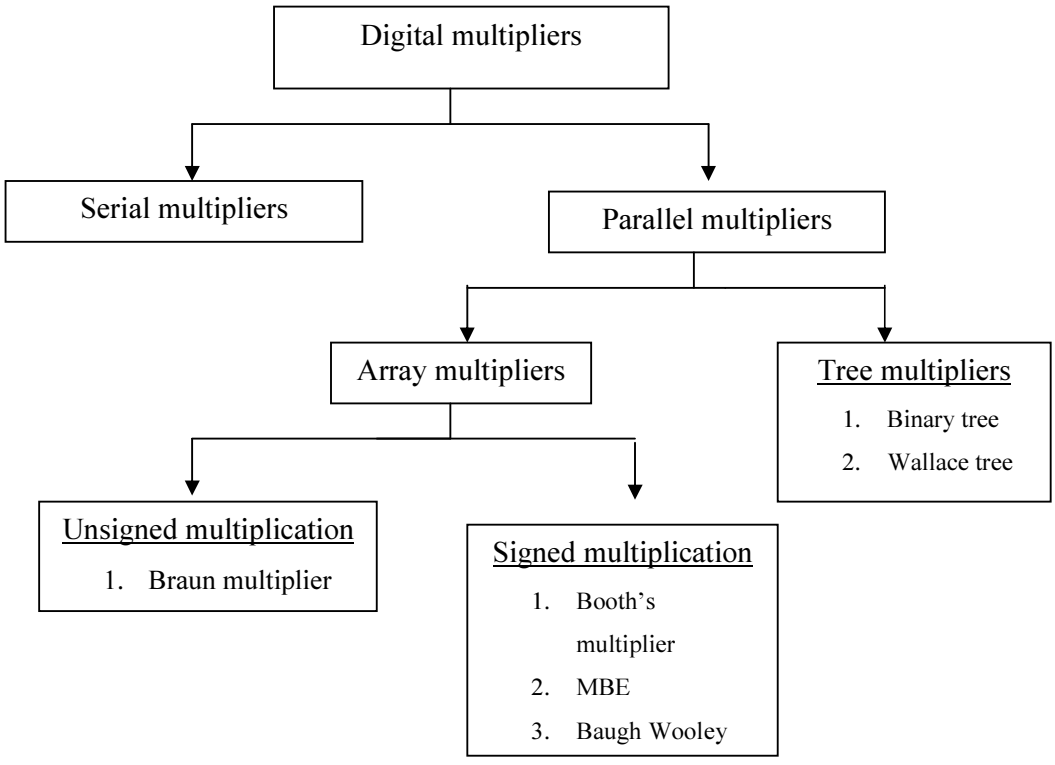



Fig. 1.1: Classification of multipliers [11]

Serial multiplication algorithms use sequential circuits with feedbacks. In serial multiplication algorithms, inner products are sequentially produced and computed.

But speed of serial multipliers is very less than parallel multipliers. The only reason one prefer parallel multiplier. Parallel multiplication algorithms often use combinational circuits, and do not contain feedback structures.

In parallel multipliers, there are two main classifications. They are

1. Array multipliers
2. Tree multipliers.

C.S. Wallace proposed a tree multiplier architecture which performs high speed multiplication. But this has a high structural irregularity and is unsuitable for VLSI implementation as it demands regularity. Array multiplier classified as signed and unsigned multipliers.Braun multiplier is used for unsigned multiplications whereas Baugh-Wooley multiplier, booth's multiplier and modified booth's multiplier can be used for signed multiplication.

### 1.2 Circuit size and speed

Performance of a circuit can be compared on two bases:-

1) Size
2) Speed

Size of circuit can be estimated on total number of gates used. The actual size of chip depends upon routing of these gates i.e. how these gates have been placed on the chip (layout of the circuit). In other words non regular circuits are usually consumes larger area than that of regular circuits, because regularity allows more compact layout.

Total delay of the whole circuit is total sum of delay associated with each single gate, and interconnection between them. Irregular or non regular structures have larger interconnections hence results into larger delay on the other hand regular structures have smaller interconnections hence smaller delay. The gate delay can be estimated by the time input signal took to propagate to output.

### 1.3 Multiplier

In various computing and signal processing applications, parallel multiplier has been a basic building block for many algorithms. Many high performance algorithms and architectures have been proposed to accelerate multiplication. Multiplication can be divided into three steps-:

a) generating partial products
b) summing up all partial products until only two rows remain

c) Adding the remaining two rows of partial products by using a carry propagation adder.

In the first step, two methods are commonly used to generate partial products. The first method generates partial product directly by using a 2-input AND gate. The second one uses Baugh Wooley, radix-2, radix-4 modified Booth's encoding (MBE), and radix-8 to generate partial products. Radix-4 MBE has been widely used in parallel multipliers to reduce the number of partial products by a factor of two. Baugh Wooley generally not used because they are not suitable for large size operands.Techniques like Wallace tree, Compressor tree are used in the second step to reduce the number of rows of the partial product. During third step, advanced adding concepts like carry-look-ahead, carry select adder are used.

### 1.4 Objective

This dissertation deals with the study, design and comparison of different multipliers designed with different technique, along with the comparison of adder circuitry that has been used in 3$^{rd}$ step of multiplication i.e. carry look ahead adder, carry save adder, carry select adder and ripple carry adder. Considering their advantages and disadvantages these are compared on the basis of area, speed and delay.

### 1.5 Language and tools used

Here different multipliers have been designed and compared. These multipliers have been designed with the help of Verilog, simulated on Model Sim SE 6.3f and synthesized on Xilinx design tool 13.1 that helps in comparing their delay, also area and power were estimated by using Synopsys design compiler.

# CHAPTER 2
# LITERATURE REVIEW

**Manoj Sharma et al. [1]** presented a new technique to compute the 2's complement by avoiding the additional adder for adding 1 and generation of long carry chain. The proposed mechanism also continues to support the concept of reducing the partial product and in persuasion of the same it is able to reduce the number of partial product and also improved further from n/2 +1 partial products achieved via modified booths algorithm to n/2.

To calculate the 2's complement first is to inverse all the bits of the data A denoting them as Abar. Now perform "Exclusive OR" (XOR) operation on Abar(0) with 1'b1, Abar(1) xor Abar(0), Abar(2) xor Abar(1) and so on till a 1'b0 is found while traversing the data bits A(i). Once 1'b0 is arrived keep the remaining bits as it is without any change. Let's consider an example where A=10101000, then 2's complement of A be denoted as A2_c_bar, then

Step 1: Abar=01010111.

Step 2:

A2_c_bar (0) = 1 xor 1 = 0

A2_c_bar (1) = 1 xor 1 = 0

A2_c_bar (2) = 1 xor 1 = 0

A2_c_bar (3) = 1 xor 0 = 1

A2_c_bar (4) = A'4 = 1

A2_c_bar (5) = A'5 = 0

A2_c_bar (6) = A'6 = 1

A2_c_bar (7) = A'7 =0

Therefore, the calculated 2's complement of A is 01011000. The 'xor' operation on all the bits is performed in parallel.

**Shiann-Rong Kuang et al. [2]** proposed a simple approach to generate a regular partial product array with fewer partial product rows, thereby reducing the area, delay, and power of MBE multipliers. The proposed MBE multiplier combines the advantages of the following two approaches. First one is to add the least significant bit $p_{i0}$ with $neg_i$ in

advance and obtained a new least significant bit $\tau_{i0}$. And second is to reduce the partial product rows from n/2+1 to n/2 by incorporating the last neg bit into the sign extension bits of the first partial product row .The proposed MBE multiplier combines the advantages of both of these two approaches to produce a very regular partial product array, as shown in Figure 2.1

| b_p | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $PP_0$ | | | | | | $\alpha_2$ | $\alpha_1$ | $\alpha_0$ | $p_{07}$ | $p_{06}$ | $p_{05}$ | $p_{04}$ | $p_{03}$ | $p_{02}$ | $p_{01}$ | $\tau_{00}$ |
| $PP_1$ | | | | | 1 | $\overline{s_1}$ | $p_{17}$ | $p_{16}$ | $p_{15}$ | $p_{14}$ | $p_{13}$ | $p_{12}$ | $p_{11}$ | $\tau_{10}$ | $c_0$ | |
| $PP_2$ | | | 1 | $\overline{s_2}$ | $p_{27}$ | $p_{26}$ | $p_{25}$ | $p_{24}$ | $p_{23}$ | $p_{22}$ | $p_{21}$ | $\tau_{20}$ | $c_1$ | | | |
| $PP_3$ | 1 | $\overline{s_3}$ | $p_{37}$ | $p_{36}$ | $p_{35}$ | $p_{34}$ | $p_{33}$ | $p_{32}$ | $\tau_{31}$ | $\tau_{30}$ | $c_2$ | | | | | |

Figure 2.1. Regular partial product array for 8 × 8 multiplication.

This regular array is generated by only slightly modifying the original partial product generation circuits and introducing almost no area and delay overhead.

**Haimin Chen,et al.[3]** proposed new technique for the disposal of negative PP based on Radix-4 Booth algorithm. Through recombining PP, it advances a skilled method avoiding the additive arithmetic of "plus 1" when computing the complement for negative PP, without increasing new PP. The experiment result shows it could obviously improve the performance of multiplier. This method is of great significance for shortening the key path of multiplier and then improving its execution speed, which could be applied in all multipliers based on Radix-4 Booth encoding. The design proposed herer effectively avoid the additive arithmetic "plus 1", and without increasing new PP. It not only doesn't increase chip resources, but also shorten key path, which could obviously improve the performance of multiplier. This design could be applied in all the multipliers based on Radix-4 Booth encoding, with great significance of application value.

**Wen-Chang Yeh et al. [4]** presented a design methodology for high-speed Booth encoded parallel multiplier. For partial product generation they proposed a new modified Booth encoding (MBE) scheme.

The conventional MBE partial product array has two drawbacks: 1) an additional partial product term at the (n-2) $^{th}$ bit position; 2) poor performance at the LSB-part compared with the non-Booth design when using the TDM algorithm. To remedy the two

drawbacks, the LSB part of the partial product array is modified. Referring to equation 2.1 the Row_LSB (gray circle) and the Neg_cin terms are combined and further simplified using Boolean minimization. The new equations for the Row_LSB and Neg_cin are given below

$$Row\_LSB_i = Y_{lsb..}(X_{2i-1} \oplus X_{2i})$$

$$Neg\_Cin_i = \overline{(X_{2i+1} + X_{2i})}.\overline{(X_{2i-1} + Y_{lsb})}\ \overline{(X_{2i} + Y_{lsb})}$$

Equation 2.1 Shows the new partial product array.

The additional partial product term is now moved to (n-1) $^{th}$ bit position, and the LSB-part array becomes more regular. Resultant partial product array is shown in fig 2.2. Note that this optimization does not incur any overheads.

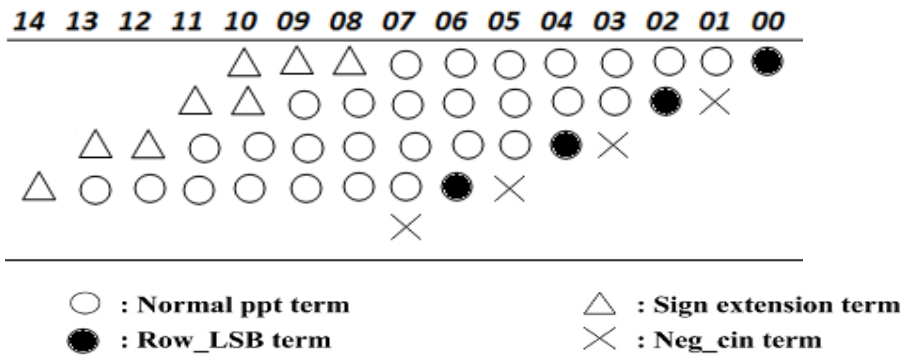



Fig. 2.2: New MBE partial product array.

Compared with a binary tree-based conditional-sum adder, the speed performance improvement is up to 25 percent. On average, the design developed here in reduces the total delay by 8 percent for parallel multiplier.

**Jung-Yup Kang et al. [5]** presented a simple, high-speed, and well-structured multiplication algorithm. Proposed multiplication algorithm focuses on the first step of a multiplication algorithm, which is the partial product generation step to reduce from n/2 +1 to n/2 the number of partial product row generated. They did this by removing the last neg signal. This would prevent the extra partial product row and, thus, save the time of one additional carry save adding stage and the hardware required for the additional carry save adding.

In this paper they presented a simple, high-speed, and well-structured multiplication algorithm by taking 2's complement of $PP_{n/2-1}$(partial product) row so that no need for the last neg because this neg signal would have already been applied when generating the two's complement of the multiplicand. A faster method to calculate the two's complement of a binary number has been presented. The conventional method (complement a binary number and add 1 to the complemented number) did not work because the propagation delay of the carry linearly increases with the word size and it would be much greater than the delay to generate the partial products. Their method is an extension of the well-known algorithm where all the bits after the rightmost "1" in the word are complemented but all the other bits are unchanged. Booth 2 modified to produce at most n/2+1 partial products.MBE is widely been adopted in parallel multipliers since it can reduce the number of partial product rows.By taking the 2's compliment of the last partial product row, the structure of the partial product array becomes more regular and easier to implement as shown in Fig 2.3. Even more importantly, the product is found faster because of the smaller number of partial product rows to add.

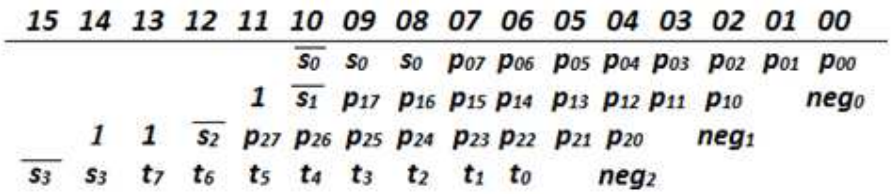



Fig.2.3 Partial product rows after removing last neg bit

**O.Saloman et al. [6]** presented two algorithms for both a simplified carry save and carry ripple addition of 2's complement numbers. The algorithms form the partial products so that they exclusively have positive coefficients which eliminate the need for the common sign bit extension. This results in a reduction of circuit area by up to six full adders per row of adders when partial products are added in an n/2 or Wallace tree. Furthermore, the capacitive load of the intermediate sum and carry sign bit signals decreases by up to a factor of seven which leads to an appropriate reduction of delay. Although the algorithms are derived for multipliers they can always be applied to appropriate adder circuits.

**Edwin de Angel et al. [7]** proposed an efficient sign extension scheme which results in significant reduction in power dissipation. In order to generate the correct sign extension they add the one's complement of the sign and a 1 on each row. Since the ones feed

into full adders with empty positions, this means that the implementation of this technique does not require any additional hardware. An extra 1 (i.e. $2^8$) is added in the eighth position. In this paper they complemented sign of the multiplicand and not the sum and carry of any row. As a result the fan out of the multiplier is the same for every adder cell improving the speed and decreasing the power dissipation. Figure 2.4 shows a partial product diagram which shows the implementation of the sign extension. T is the one's complement of the sign and C is the correction constant required to generate the negative partial products. In addition to the reduction in fan-out from 3 to 1 from the conventional sign extension, a reduction in the number of full adders per row is achieved from n + 1 to n reducing area complexity and power dissipation and also implementation of this sign extension scheme does not require additional hardware or any penalty.

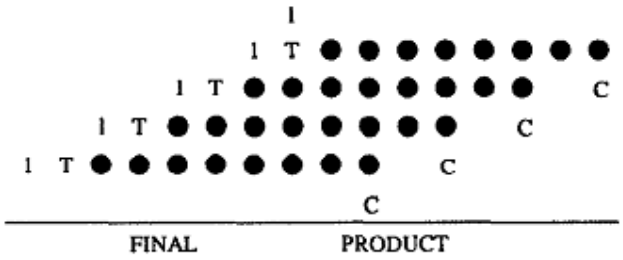



Fig. 2.4: Partial Product Diagram with Sign Generate Sign Extension

**Aamir A. Farooqui et al. [8]** described the data-path and VLSI implementation of a 32x32 bit signed unsigned multiply accumulate (MAC) unit. In this design they had solved the problem of dealing with signed and unsigned numbers simultaneously, using modified Booth algorithm. This MAC unit can perform 32x32, 32x16, and two 16x16 multiplications, on signed unsigned operands with a throughput of 2, 1, and 1 cycle, respectively. In this, MAC unit inputs are applied using 32-bit registers A (multiplier) and B (multiplicand), while the result is stored in 64-bit register C. To balance the pipeline stages of the MAC, the 16x16 multiplier result is produced in carry save form and finally added in the second stage of the MAC. This reduces the clock cycle and power consumption of the MAC (fewer glitches due to a carry propagate adder) unit. They used special circuitry to accommodate signed unsigned operands and to deal with sign extension. In Booth multiplier the number of Booth encoders and selectors required for m-bits is m/2 and it requires one extra bit to handle unsigned operands.

Therefore in Booth multipliers (for number of bits in power of 2), one extra encoder and selector is required, to deal with unsigned numbers.
In this paper they solved this problem in a very simple manner, with minimum hardware and delay. In this design signed and unsigned multiplication is controlled by two signals. When the multiplier is unsigned and the most significant bit of the multiplier is one then multiplicand is added with the 8-partial products (means multiplication by l), else a zero is added with the partial products. This can be accomplished by using and gates. To deal with unsigned multiplicand, the multiplicand is sign extended to 17 bits. They used Modified Booth algorithm coupled with (three dimension reduction method) TDM and sign correction circuitry results in a multiplier, with a delay (partial product addition) equivalent to 6 XOR gates.

**Fayez Elguibaly [9]** presented a dependence graph (DG) to visualize and describe a merged multiply-accumulate (MAC) hardware that is based on the modified Booth algorithm (MBA). He used carry-save technique in the Booth encoder, the Booth multiplier, and the accumulator sections to ensure the fastest possible implementation. He apply DG MAC data word size and allows designing multiplier structures that are regular and have minimal delay, sign-bit extensions, and data path width. He proposed a fast pipelined implementation by using dependence graph, in which he used an accurate delay model for deep submicron CMOS technology. The delay model describes multi-level gate delays, taking into account input ramp and output loading. Based on the delay model, pipelined parallel MAC design proposed by him is three times faster than other parallel MAC schemes that are based on the MBA. The speedup resulted from merging the accumulate and the multiply operations and the wide use of carry-save techniques.

**Osman Hasan et al. [10]** presented a formal synthesis methodology that can be automated and thus it not only ensures formally verified synthesis results but also is very easy to use for end users who do not have any background in formal semantics and reasoning. Their synthesis methodology achieves correctness by construction and thus eliminates the post synthesis verification requirements, which in turn reduces design time. They had demonstrated the practical effectiveness of their methodology by successfully constructing an automated tool that is capable of correctly synthesizing WT multipliers of arbitrary length operands. The proposed formal synthesis methodology is quite general and can be applied to correctly synthesize any digital circuit. This methodology helps to

enhance the library of formally verified correctness- preserving synthesis transformations and thus formally synthesize a bigger set of combinational digital circuits.

**S.Sri Sakthi et al. [11]** proposed architecture for efficient reconfiguration is the one Level Recursive Architecture. He implemented an n-bit multiplier using four n/2 bit multipliers, where n is the number of bits. These four n/2 bit multipliers executes in parallel and their results are added up. Power consumption of the multipliers is reduced with the introduction of power efficient scheme Dynamic Operand Interchange to the reconfigurable booth architecture. Dynamic operand interchange technique interchanges operands dynamically during the execution. The operand with the smallest dynamic range is chosen as the multiplier operand. This requires a Dynamic Range Detector (DRD) circuit to detect dynamic range of the operand and a switcher to exchange the operand dynamically. When the operand with smaller dynamic range is encoded with modified booth, there is increased probability of partial products becoming zero. This reduces the switching activities and thereby reduces power consumption.

**S.Saravanan et al. [12]** proposed a design approach of a low power Hybrid Encoded Booth Multiplier (HEBM) with Reduced Switching Activity Technique (RSAT). This RSAT approach has been applied on the hybrid encoder of the multiplier to reduce the power consumption. The hybrid encoder in the low power multiplier uses both the Booth and proposed technique. If the number of 1's less than or equal to three, the proposed encoding technique used otherwise go for Booth technique.

# Chapter 3

# MULTIPLIER

Multiplication can be done serially or parallel. Theoretically multiplication can be done by repeated addition. Consider multiplication A x B where A is multiplier and B is multiplicand, if we add B to itself A times the sum will be the product of A x B. Practically this process is very slow so never been used. This method involves generating n intermediate products and then adding them properly taking into account the weight of each bit while moving from LSB to MSB. The following steps are used in manual calculation:

1) Start with LSB of the multiplier, if it is 1 the multiplicand will be the first partial product. If LSB is 0, put 0s in the current partial product.
2) Analyze next bit of the multiplier and generate the partial products as per the first rule. The new partial product will be placed one bit left to that of previous partial product. Repeat this step till last bit of the multiplier covered.
3) Add all the partial products to get final product.

The previous method is slightly modified while implementing in a computer by the hardware using parallel binary adder. The partial product is immediately added after each cycle instead of adding them at the end. The partial product is shifted right by one bit after each cycle so that multiplicand can be added straight to next cycle.

Booth's multiplication algorithm is a multiplication algorithm that multiplies two signed or unsigned binary numbers. Booth's multiplication algorithm make use of booth's encoding algorithm in order to reduce number of partial products by considering number of bits at a time (depend upon type of algorithm i.e. radix2, radix4 or higher radix), thereby achieving significant improvement in area power and delay compared to other methods.

### 3.1 Multiplication of signed and unsigned number

Multiplication of unsigned numbers can be done by simple multiplication algorithm. If the sign of the multiplier and multiplicand are different, sign of the product is negative. The magnitude is multiplied in the same way as the unsigned numbers. If both are

negative, the sign of the result is positive. The sign requirements are met by the general rule: the sign of the product id exclusive or of the sign of the number.

### 3.1.1 Two's Complement multiplication

If one of the negative operand is in 2's complement form, the previous method cannot be used. The number has to be converted into sign magnitude before multiplication and the product has to be reconverted into 2's complement form. Though it works well, it increases the total time taken for multiplication. There are several methods for 2's complement multiplication. Booth's algorithm is a popular method for 2's complement multiplication. It also speeds up multiplication process by analysis multiple bits of multiplier at a time.

## 3.2 Booth's Multiplier

The booth's algorithm handles both signed and unsigned number. When the multiplier has a stream of 1's, the number of additions required is minimized. This speeds up the multiplication operation as compared to method followed for unsigned number. The exact amount of time reduction depends on the bit pattern in the multiplier. The basic principle followed in Booth's algorithm i.e. multiplication is nothing but addition of properly shifted multiplicand pattern.

### 3.2.1 Radix 2 Booth's Algorithm

Radix 2 multiplication algorithm is multiplication algorithm that multiplies two signed binary numbers in two's compliment notations.

The Booth's algorithm serves two purposes:

1) Fast multiplication
2) Signed multiplication

Booth's algorithm requires examination of the multiplier bits, and shifting of the partial product. Prior to the shifting, the multiplicand may be added to partial product or subtracted from the partial product, or left unchanged according to the following rules:

I. First step is to add '0' to the LSB of the multiplier i.e.$B_{-1}$ is 0.

II. Second step is to look at the bits $B_x$ and $B_{x-1}$ and perform operation according to the table given below.

0 0 Shift only

1 1 Shift only

0 1 Add Y to U, and shift

1 0 Subtract Y from U, and shift or add (-Y) to U and shift

III. The next step is to set two registers, which we name u and v, to be zero. These are going to be the registers where we store our product throughout the working of the problem.

IV. Once added, we then do an arithmetic right shift on u and v, with the last bit of v dropping off, and a circular right shift on x, also copying the LSB of x into x-1.

V. Now repeat steps for n number of times where n is number of bits in multiplier and multiplicand. And finally we have result of A x B.

### 3.2.2 Drawbacks of Radix-2 Booth's Algorithm

a) Variable number of add/subtract operations and of shift operations between two consecutive add/subtract operations .Inconvenient when designing a synchronous multiplier.

b) Algorithm inefficient with isolated 1's.

### 3.2.3 Radix-4 Modified Booth's Algorithm

Booth's 2 modified to produce at most n/2+1 partial products.MBE is widely been adopted in parallel multipliers since it can reduce the number of partial product rows to be added by half, thus reducing the size and enhancing the speed of the reduction tree.

The booth's encoding algorithm is a bit-pair encoding algorithm that generates partial products which are multiples of the multiplicand. The booth's algorithm shifts and/or complements the multiplicand (X operand) based on the bit patterns of the multiplier (Y operand). Essentially, three multiplier bits [Y (i+1), Y (i) and Y (i-1) ] are encoded into nine bits that are used to select multiples of the multiplicand {-2X, -X, 0, +X, +2X}. The three multiplier bits consist of a new bit pair [Y (i+1), Y (i)] and the leftmost bit from the previously encoded bit pair [Y (i-1)] as shown in figure 3.1.Partial products can be generated with the help of table 3.1.

- Separately: $x_{i-2}$ and $x_{i-3}$ recoded into $y_{i-2}$ and $y_{i-3}$ - $x_{i-4}$ serves as reference bit.
- Groups of 3 bits each overlap - rightmost being $x_1$ $x_0$ ($x_{-1}$), next $x_3$ $x_2$ ($x_1$), and so

on.

- Bits $x_i$ and $x_{i-1}$ recoded into $y_i$ and $y_{i-1}$ - $x_{i-2}$ serves as reference bit.

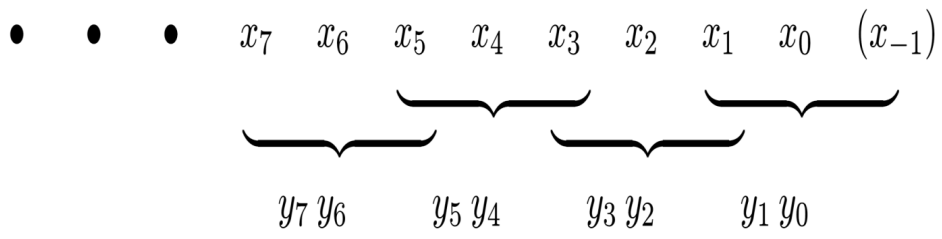



Fig. 3.1: Recoding in Radix 4

Table 3.1 : Booth's Radix-4 Algorithm Table

| $x_i$ | $x_{i-1}$ | $x_{i-2}$ | **Operation** | **Comments** |
|---|---|---|---|---|
| 0 | 0 | 0 | +0 | String of zeroes |
| 0 | 1 | 0 | +A | A single 1 |
| 1 | 0 | 0 | -2A | Beginning of 1's |
| 1 | 1 | 0 | -A | Beginning of 1's |
| 0 | 0 | 1 | +A | End of 1's |
| 0 | 1 | 1 | +2A | End of 1's |
| 1 | 0 | 1 | -A | A single 0 |
| 1 | 1 | 1 | +0 | String of 0's |

The modified Booth's algorithm (radix-4 recoding) starts by appending a zero to the right of $X_2$(multiplier LSB). Triplets are taken beginning at position $X_{-1}$ and continuing to the MSB with one bit overlapping between adjacent triplets. If the number of bits in X (excluding $x_{-1}$) is odd, the sign (MSB) is extended one position to ensure that the last triplet contains 3 bits. In every step we will get a signed digit that will multiply the multiplicand to generate a partial product entering the reduction tree. The meaning of each triplet can be seen in figure 3.1. Fig 3.2 shows array of partial products for signed number.

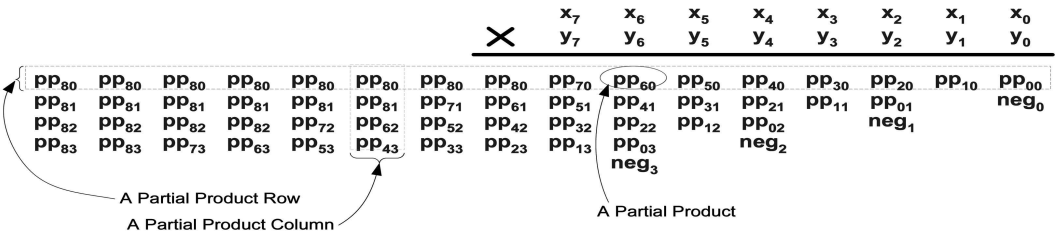



Fig.3.2 The array of partial products for signed multiplication with MBE [5]

### 3.2.4 Booth's Encoder (Radix-4)

In this topic booth's encoding table has been converted into combinational circuit so that it results in less power and delay. Table 3.2 describes recoding scheme.

Table 3.2 : Recoding in Booth's Radix-4 Algorithm

| $x_i$ | $x_{i-1}$ | $x_{i-2}$ | Sgn | 2a | A |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 0 | 1 |
| 0 | 1 | 0 | 0 | 0 | 1 |
| 0 | 1 | 1 | 0 | 1 | 0 |
| 1 | 0 | 0 | 1 | 1 | 0 |
| 1 | 0 | 1 | 1 | 0 | 1 |
| 1 | 1 | 0 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 | 0 | 0 |

$$a = x_{i-2}\,(x_{i-1} \text{ xor } x_i) \quad ; \quad 2a = (\sim x_i \,.\, x_{i-1} \,.\, x_{i-2}) + (x_i \,.\, \sim x_{i-1} \,.\, \sim x_{i-2}) \quad ; \quad sgn = x_i$$

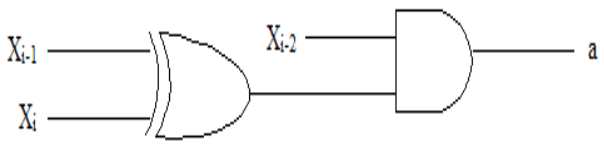



Fig.3.3 Combinational Circuit to generate **a**

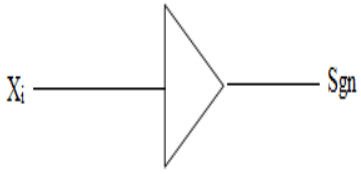



Fig.3.4 Combinational Circuit to generate sgn bit

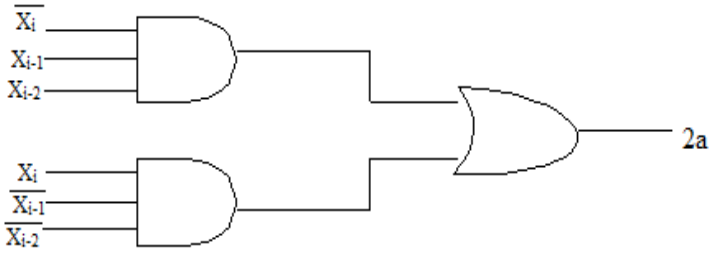



Fig.3.5 Combinational Circuit to generate **2a**

### 3.2.5 Sign Extension Problem and its Solution

Partial products can be negative require sign extension, which is cumbersome. In signed multiplication, the sign bit of a partial product row would have to be extended all the way to the MSB position which would require the sign bit to drive that many output loads. This makes the partial product rows unequal in length The first row spans 16 bits the second row 14 bit, the third row 12 bit and the fourth row 10 bits as shown in fig 3.6.

Sign extension prevention method arrives at a newly formed partial product rows as in figure 3.7 where the sign extension has been removed. Sign bits are either all 0's or all 1's.

- Note that all 0's is all 1's + 1 in proper column.
- Use this to reduce loading on MSB.
- No need to add all the 1's.

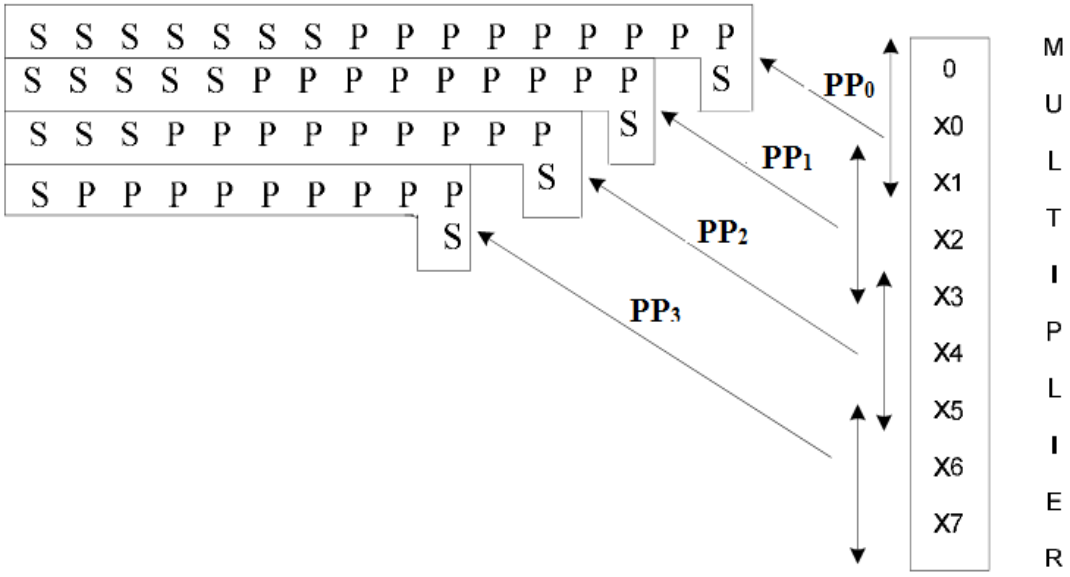



Fig. 3.6: Multiplication with Sign Extension Problem

New architecture in Fig (3.8) shows partial products array after removing the sign extension problem in Booth's algorithm.

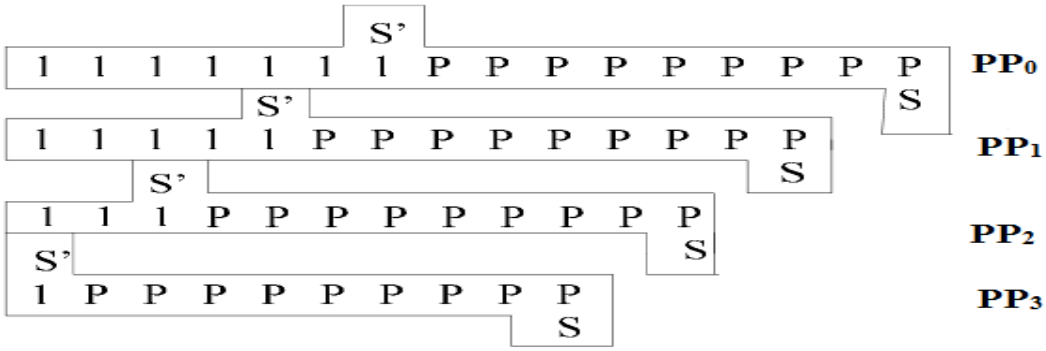



Fig. 3.7: Solution of Sign Extension Problem

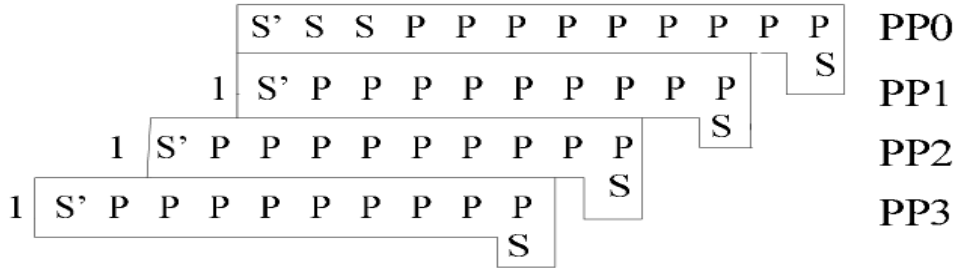



Fig.3.8 Algorithm after removing sign extension

### 3.2.6 Radix-8 Booth's Algorithm

Recoding extended to 3 bits at a time - overlapping groups of 4 bits each. Only n/3 partial products generated - multiple 3A needed - more complex basic step. Example: recoding 010(1) yields $y_i\ y_{i-1}\ y_{i-2}$=011. Technique for simplifying generation and accumulation of ±3A exists.

Radix-8 recoding applies the same algorithm as radix-4, but now we take quartets of bits instead of triplets. Consequently, a multiplier based on this radix-8 scheme generates fewer partial products than a radix-4 multiplier, but the computation of each partial product is more complex. In particular, a partial product corresponding to an encoding x=±3 requires the computation of 3x, and therefore a full addition.

Here we have an odd multiple of the multiplicand 3Y, which is not immediately available. To generate it we need to perform this previous add: 2Y+Y=3Y.The multiplication of two binary numbers, 24-bit length, 2s-complement and using the algorithm with radix-8 recoding of the multiplier presents the following features:

- Radix-8 recoding of the multiplier implies a reduction in the number of digits to 8.
- The partial products multiplexer must choose one out of nine possibilities depending on the value of the corresponding signed-digit.

Fig 3.9 shows example of radix 8 algorithm:

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| A | | | 00 | 01 | 00 | 01 | | | +17 |
| X | | | 00 | 00 | 10 | 10 | | | +10 |
| Add A | | | 00 | 10 | 00 | 10 | | | |
| 3 bit shift | | | 00 | 00 | 10 | 00 | 10 | | |
| Add A | | | 00 | 01 | 00 | 01 | | | |
| | 00 | 01 | 01 | 01 | 01 | 0 | | +170 | |

Fig. 3.9: Example of Radix 8 Algorithm

### 3.2.7 Comparison of radix 2, radix 4 and radix 8 algorithm

- The shortcoming of Radix 2 Booth's algorithm is that it becomes inefficient when there are isolated 1's. For example, 001010101(decimal 85) gets reduced to 01-11-11-11-1(decimal 85), requiring eight instead of four operations. 001010101(0) recoded as 011111111, requiring 8 instead of 4 operations. This problem can be overcome by using high radix Booth's algorithms.
- As we move towards Radix 8 less number of partial products are generated but more number of operations are required to generate {+1,+2, +3, +4, -1, -2, -3, -4}. In Radix 4 we need to save {+2, -2, +1, -1}.
- Speed of Radix 8 is highest among Radix 2, 4 and 8 but the complexity increases.

# CHAPTER 4

# ADDERS

Addition is the most common and often used arithmetic operation on microprocessor, digital signal processor, especially digital computers. Also, it serves as a building block for synthesis all other arithmetic operations. Therefore, regarding the efficient implementation of an arithmetic unit, the binary adder structures become a very critical hardware unit. In any book on computer arithmetic, someone looks that there exists a large number of different circuit architectures with different performance characteristics and widely used in the practice. Although many researches dealing with the binary adder structures have been done, the studies based on their comparative performance analysis are only a few. In this dissertation, qualitative evaluations of the classified binary adder architectures are given. Among the huge member of the adders here verilog (Hardware Description Language) code for Ripple-carry, Carry-save adder to emphasize the common performance properties belong to their classes. With respect to asymptotic delay time and area complexity, the binary adder architectures can be categorized into three primary classes as given in Table 4.1. The given results in the table are the highest exponent term of the exact formulas, very complex for the high bit lengths of the operands.

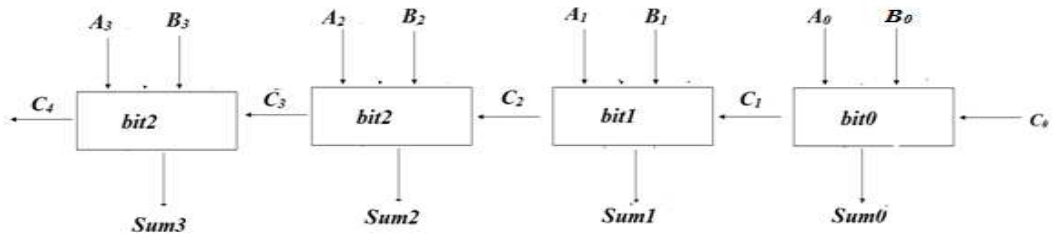
The first class consists of the very slow ripple-carry adder with the smallest area. In the second class, the carry-skip, carry-select adders with multiple levels have small area requirements and shortened computation times. From the third class, the carry-look ahead adder and from the fourth class, the parallel prefix adder represents the fastest addition schemes with the largest area complexities.

Table 4.1 Categorization of adder's w.r.t delay time and capacity

| Complex(A) | Delay(T) | Product (A x T) | Adder |
|---|---|---|---|
| O(n) | O(n) | $O(n^2)$ | Ripple Carry |
| O(n) | O(n) | $O(n^2)$ | Carry Save Adder |
| O(n) | O(logn) | O(n logn) | Carry lookahead adder |

### 4.1 Ripple Carry Adders (RCA)

The well-known adder architecture, ripple carry adder is composed of cascaded full adders for n-bit adder, as shown in figure 4.1.It is constructed by cascading full adder blocks in series. The carry out of one stage is fed directly to the carry-in of the next stage. For an n-bit parallel adder it requires n full adders.




.

Fig. 4.1 A 4-bit Ripple Carry Adder

- Not very efficient when large number bit numbers are used.
- Delay increases linearly with bit length.

### 4.2 Carry Save Adder (CSA)

One of the major speed enhancement techniques used in modern digital circuits is the ability to add numbers with minimal carry propagation. The basic idea is that three numbers can be reduced to 2, in 3:2 compressor, by doing the addition while keeping the carries and the sum separate. This means that all of the columns can be added in parallel without relying on the result of the previous column, creating a two output "adder" with a time delay that is independent of the size of its inputs.

```
          10111001
          00101010
          00111001
Sum:      10101010
Carry:  00111001
Result:  100011100
```

The sum and carry can then be recombined in a normal addition to form the correct result. This process may seem more complicated and pointless in the above trivial example, but the power of this technique is that any amount of numbers can be added together in this manner. It is only the final recombination of the final carry and sum that requires a carry propagating addition. The use of a wallace tree, arranges the adder tree so that all of the output bits could be obtained while minimizing the size of the circuit. However, in the

case of multipliers, we know what the expected output size will be, and so we can set all of the input and output sizes to that value. We do not care about any overflowing sign bits, so they can be discarded and the carries can simply be shifted left to the correct alignment. All of the results can then be grouped together as one and continually reduced until we are left with two values. This is demonstrated by fig.4.2

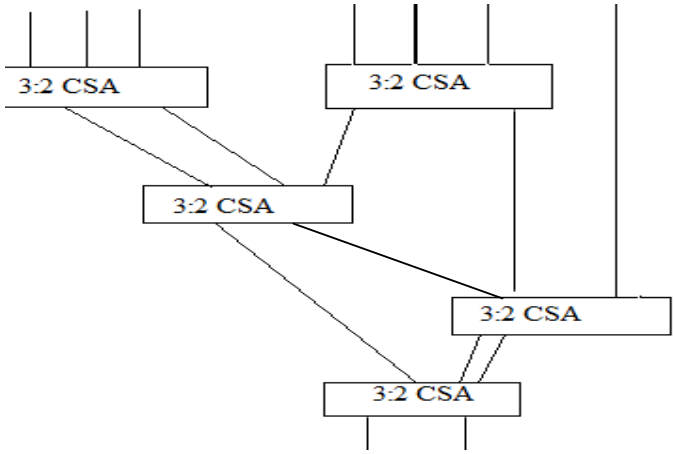



Fig. 4.2: Carry-save adder tree

This method may appear wasteful because a lot of bits in the first stages of the adder tree will be frozen to zero. However, these will be optimized during synthesis, and this technique seems to produce more favorable synthesis results than trying to code the design efficiently.

### 4.2.1 3:2 Compressor

The design of the 3:2 compressor is simple, with the following truth table showing that it is nothing more than a full adder. Adding three k-bit numbers together simply involves an array of k 3:2 compressors, each being independent of each other and operating on a single bit position. This has been shown in fig. 4.3.

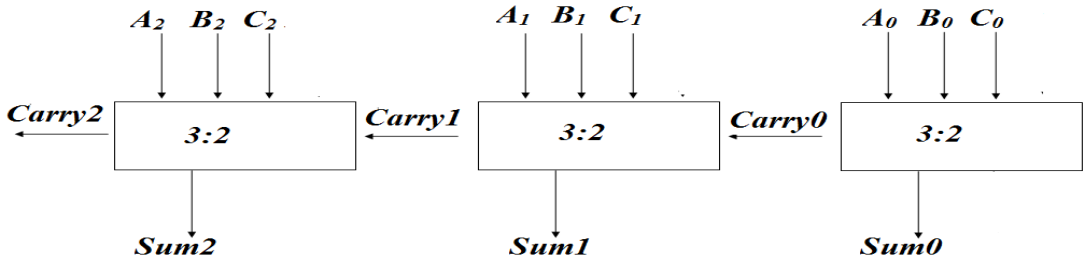



Fig. 4.3 : Architecture of the full word 3:2 compressor, using individual bit 3:2 compressors.

Behavior of 3:2 compressor is shown in table 4.2.

Table 4.2 : Truth table for the 3:2 compressor.

| Inputs | | | Sum | Carry |
|---|---|---|---|---|
| A | B | C | | |
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | 1 |
| 1 | 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 1 | 1 |

**4.2.2.4:2Compressors**

The discussion so far has referred only to 3:2 carry-save adders, but it is also possible to add four bits in this format. In reality, there are actually five inputs (one being a carry in), and three outputs (two carries and the sum) as shown in fig 4.4:

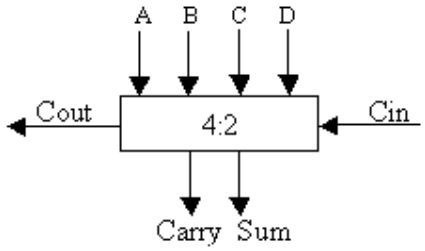



Fig. 4.4 : High level view of the 4:2 compressor

The characteristics of the 4:2 compressor are:

- The outputs represent the sum of the five inputs, so it is really a 5 bit adder.
- Both carries are of equal weighting (i.e. add "1" to the next column).
- To avoid carry propagation, the value of Cout depends only on A, B, C and D. It is independent of Cin.
- The Cout signal forms the input to the Cin of a 4:2 of the next column.

The behavior of the 4:2 compressor is described by table 4.3

Table 4.3: Truth table for the 4:2 compressor cell

| Inputs | | | | Cin = 0 | | Cin = 1 | | Cout |
|---|---|---|---|---|---|---|---|---|
| A | B | C | D | Carry | Sum | Carry | Sum | |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| 0 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 0 |
| 0 | 0 | 1 | 0 | | | | | |
| 0 | 1 | 0 | 0 | | | | | |
| 1 | 0 | 0 | 0 | | | | | |
| 0 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 1 |
| 0 | 1 | 1 | 0 | | | | | |
| 1 | 1 | 0 | 0 | | | | | |
| 0 | 1 | 0 | 1 | | | | | |
| 1 | 0 | 1 | 0 | | | | | |
| 1 | 0 | 0 | 1 | | | | | |
| 0 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 0 | | | | | |
| 1 | 1 | 0 | 1 | | | | | |
| 1 | 1 | 1 | 0 | | | | | |
| 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 |

**A** k-bit 4:2 word adder is then formed as shown below, in figure 4.5.

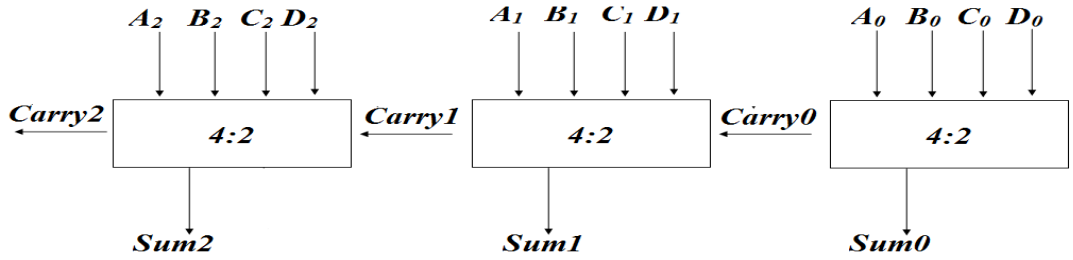



Fig. 4.5: Architecture of the full word 4:2 compressor, using individual bit 4:2 compressors

### 4.3 Carry Select Adders (CSLA)

An 8-bit carry-select adder, built as a cascade from a 1-bit full-adder, a 3-bit carry-select block, and a 4-bit carry-select adder. Click the input switches or type the 'a', 'b', 'c' bind

keys to control the first-stage adder. The problem of the ripple-carry adder is that each adder has to wait for the arrival of its carry-input signal before the actual addition can start. The basic idea of the carry-select adder is to use blocks of two ripple-carry adders, one of which is fed with a constant 0 carry-in while the other is fed with a constant 1 carry-in. Therefore, both blocks can calculate in parallel. When the actual carry-in signal for the block arrives, multiplexers are used to select the correct one of both precalculated partial sums. Also, the resulting carry-out is selected and propagated to the next carry-select block. In total, the carry propagation time through an n-bit adder block is reduced from O(n) to the number of stages times the delay of the multiplexers. Naturally, using n blocks of 1-bit carry-select adders would incur a complexity of n multiplexers, again resulting in O(n) delay. Therefore, a partition with (slowly) increasing block-size is chosen. In the example, the first (least-significant) block consists of a simple full adder, followed by a 3-bit carry-select block, and finally a 4-bit carry-select block. A common choice for a 16-bit carry-select adder is to use a 6-4-3-2-1 bit partitioning. While the delay of the standard ripple-carry adder with n-bits is O(n), the delay through the carry-select adder behaves as O(sqrt(n)) at a hardware cost of O(3*n).Block diagram of carry select adder is shown in figure 4.6.

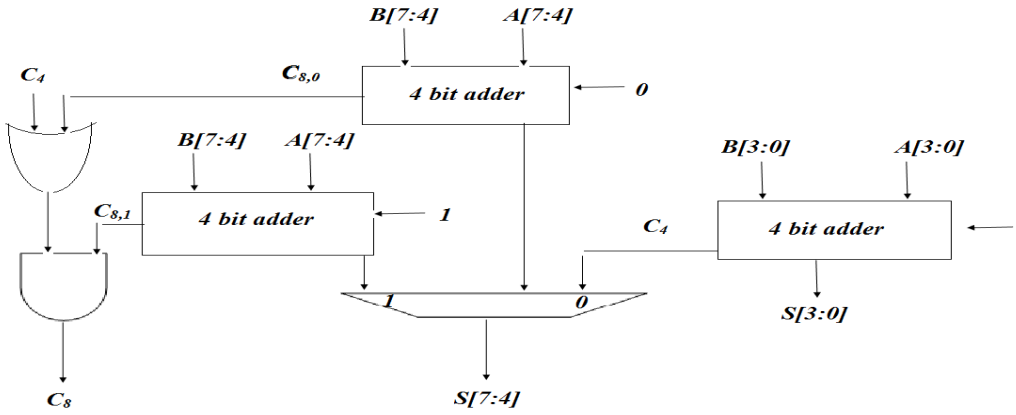



Fig. 4.6 A Carry Select Adder

### 4.4 Carry Look Ahead Adders (CLA)

As the name implies, carry look ahead adders [2] predicts the block carry output instead of calculating the carry bit by bit. The CLA blocks define two different terms, propagate and generate. The generate G term determines if a carry out would be set

from inside the block independently from the inputs. Propagate term P determines whether the input carry to the block would propagate to the output or not. Figure 4.7 shows the architecture of one level CLA adder.[15]

The delay of a carry look ahead adder is proportional to n/k where k is the number of bits per block. The expressions for the CLA adder is given by equation 4.1

$$G_i=A_i.B_i$$

$$P_i=A_i+B_i$$

$$G^*=G_3+G_2.P_3+G_1.P_3.P_2+G_0.P_3.P_2.P_1$$

$$P^*=P_3.P_2.P_1.P_0$$

$$C^*=C_i.P^*+G^*$$

Equation 4.1 : Equation to generate $G_i, P_i, G^*, P^*, C^*$

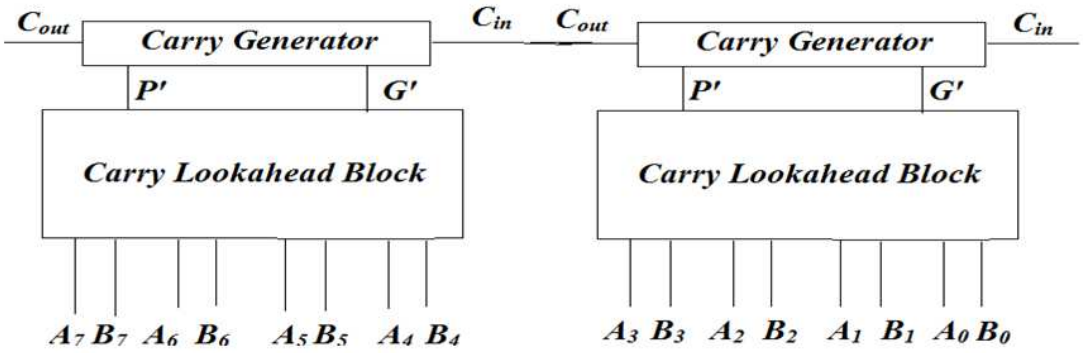



Fig. 4.7 Architecture of CLA adder[15]

Where Pi and Gi are the bit propagate and generate and P*, G*, Ci and C* are the block propagate, generate, block input carry and carry output respectively. From the previous equations, it is clear that as the number of bits per block k increases, the size of the generate circuit increases exponentially. Therefore, larger block size would require more area and power. However, as the block size k increases, the speed enhancement would increase too because the carry of the whole will be calculated in one step. Thus, the power and area have to be compromised with the delay.

# CHAPTER 5

# Proposed Technique

### 5.1 Regularized Partial Products

As discussed earlier multiplication can be completed in 3 steps:

1) Generation of partial products.
2) Accumulation of these partial products until two rows left.
3) Addition of remaining two rows to get final result.

First step can be completed with the help of Modified Booth's Algorithm (radix-4). Radix- 4 booth's algorithm generates n/2 partial product rows. One of the drawback of Booth's algorithm is it generates non regular or irregular partial products. As we discussed earlier irregular structures have routing problem or require longer interconnections which results into more area and delay, or in other words irregular structures are difficult to implement. So our first priority is to convert this irregular partial product array into regular structure, so that it consumes less area and time i.e. easier to implement.

After applying radix -4 booth's algorithm on 8x8 bit multiplier we have 4 partial products ($PP_0$, $PP_1$, $PP_2$, $PP_3$) as shown in fig. 5.1.

| b_p | 13 | 12 | 11 | 10 | 09 | 08 | 07 | 06 | 05 | 04 | 03 | 02 | 01 | 00 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $PP_0$ | $p_{07}$ | $p_{07}$ | $p_{07}$ | $p_{07}$ | $p_{07}$ | $p_{07}$ | $p_{07}$ | $p_{06}$ | $p_{05}$ | $p_{04}$ | $p_{03}$ | $p_{02}$ | $p_{01}$ | $p_{00}$ |
| $PP_1$ | $p_{17}$ | $p_{17}$ | $p_{17}$ | $p_{17}$ | $p_{17}$ | $p_{16}$ | $p_{15}$ | $p_{14}$ | $p_{13}$ | $p_{12}$ | $p_{11}$ | $p_{10}$ | | |
| $PP_2$ | $p_{27}$ | $p_{27}$ | $p_{27}$ | $p_{26}$ | $p_{25}$ | $p_{24}$ | $p_{23}$ | $p_{22}$ | $p_{21}$ | $p_{20}$ | | | | |
| $PP_3$ | $p_{37}$ | $p_{36}$ | $p_{35}$ | $p_{34}$ | $p_{33}$ | $p_{32}$ | $p_{31}$ | $p_{30}$ | | | | | | |

Fig 5.1 Partial Products generated with the help of booth's algorithm

Drawback of this approach is irregular structure and also sign extension problem, which results in more adder circuitry. In order to make it regular structure one more bit known as neg bit is added to it. Neg bit is used for negative encoding i.e. if sign of partial product is negative then neg bit will be 1 else it will be zero. This neg bit along with sign prevention technique discussed in chapter 2 will give more regularized partial products. Partial products after using neg bit and sign prevention method is shown in fig. 5.2(a). Here compared to fig 5.1 we have more regularized partial products.

In order to make it more regularized $P_{i0}$ bit of each partial product row is added to $neg_i$ bit, sum of these two bit will replace $P_{i0}$ bit and carry will be added to the next partial product row as shown in fig. 5.2(b). Here sum is denoted by $t_{i0}$ and carry is denoted by $c_{i..}$ This will give more regularized partial product array. [2]

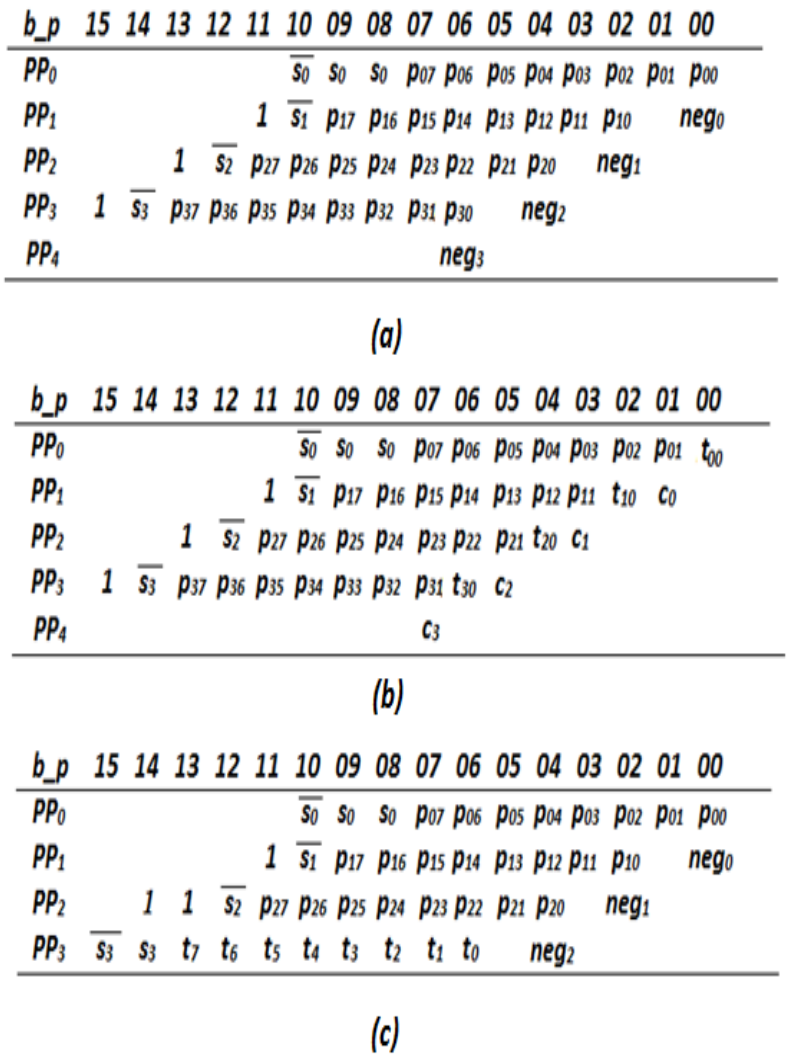



Fig 5.2 Conventional MBE partial product array for 8*8 bit multiplication [2]

This approach gives more regularize partial products but problem is that it will give regularized partial product but generate n/2+1 partial products, which require extra adder circuitry.In order to remove this extra partial product row one can do direct 2's complement of $PP_3$ row,results in n/2 rows , fig 5.2(c) shows partial products with n/2 rows. This structure will give lesser delay and area compared to others discussed above.

### 5.2 Two's Complement Methods

As discussed earlier in order to make structure more regular and to reduce partial product rows from n/2+ 1 to n/2, direct 2's complement can be applied to $PP_3$ row. Basic mechanism to find 2's complement is to firstly take the 1's complement and then adding 1, which is an inefficient operation because of generation of long carry chain. There are number of techniques to find 2's complement, some of them are discuss here:-

So in order to enhance the performance of multiplier efficient disposal system of adding 1 is needed. For example 2's complement of 01011000 will be:-

1) First step is to take 1's complement of a number i.e. 10100111.

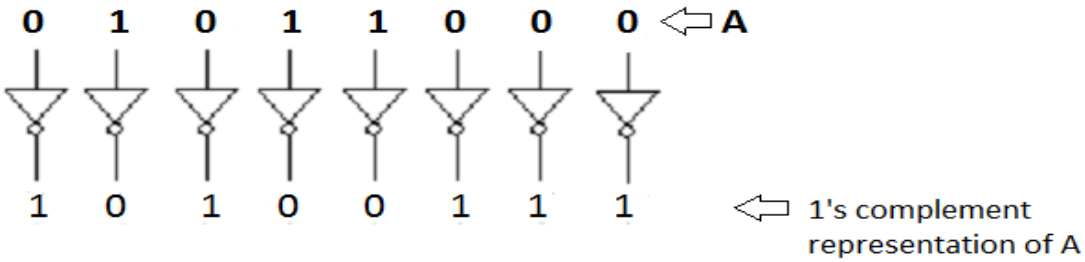



Fig.5.3 Shows 1's complement representation

2) Now add 1 to the resultant 1's complement i.e.

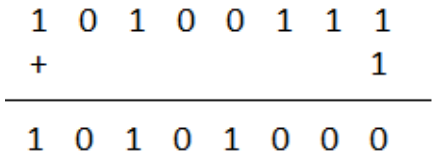



Fig.5.4 Shows 2's complement of a number

The first and obvious method is to enhance the capability and performance of the adder used for the addition of 1 while calculating the 2's complement whenever needed. But this method is not efficient in terms of additional hardware requirement and extra time required.

Second method to find 2's complement is where all the bits after the rightmost "1" in the word are complemented but all the other bits are unchanged. The two's complement of a binary number $001010_2$ is $110110_2$ (Fig. 5.4). First thing to do in this algorithm is to find the right most 1. Here in this example right most 1 lies at a position 1. Therefore, values in bit positions 0 to 1 are kept unchanged, while values in bit positions 0 and 1 can simply be complemented.

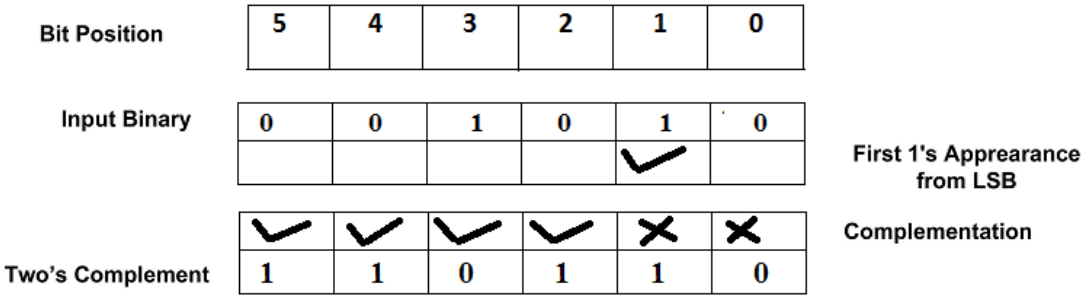



Fig. 5.5 Shows 2's complement conversion example [5]

Now here one of the challenging tasks is how to find right most 1. This rightmost 1 is called conversion signals. Therefore, 2's complement now comes down to finding the conversion signals that are used for selectively complementing some of the input bits. If the conversion signal at any position is "0" (the crosses in Fig. 5.5), then the value is kept unchanged and, if the conversion signal is "1" (the checks in Fig. 5.5), then the value is complemented. The conversion signals after the rightmost "1" are always 1. They are 0 otherwise. Once a lower order bit has been found to be a "1," the conversion signals for the higher order bits to the left of that bit position should all be "1"[5].

However, this searching for the rightmost "1" could be as time consuming as rippling a carry through to the MSB since the previous bits information must be transferred to the MSB. Therefore, we must find a method to expedite this detection of the rightmost "1."

As we will see, this search for the rightmost "1" can be achieved in logarithmic time using a binary search tree-like structure. We first find the conversion signals for a 2-bit group by grouping two consecutive bits (the grouping always starts from the LSB) from the input and finding the conversion signals in each group, as shown in Fig. 5.6a. Then, we find the conversion signals for a 4-bit group (formed by two consecutive 2-bit groups). Then, we find the conversion signals for a 8-bit group (formed by two consecutive 4-bit groups). This divide -and- conquer approach is pursued until the whole input has been covered. When grouping two 2n-bits groups, the leftmost conversion signals from the right group contain the accumulative information of its group about whether a "1" ever appeared in any bit position of its group so that a conversion signal should force all the conversion signals from the left group all the way to the "1" if it is itself a "1." For instance, as shown in Fig. 5.6b, if $CS_1$ (the leftmost conversion signal from the right group) = "1," the conversion signals from the left group ($CS_2$ and $CS_3$) should be forced to a "1," regardless of their previous values. If $CS_1$ ="0", nothing happens to the conversion signals from the left group. This variable control is shown with

a dashed arrow. Likewise, $CS_5$ may affect conversion signals $CS_6$ and $CS_7$. The same goes for $CS'_3$, which may affect the conversion signals ($CS'_7$, $CS'_6$, CS'5, and $CS'_4$).

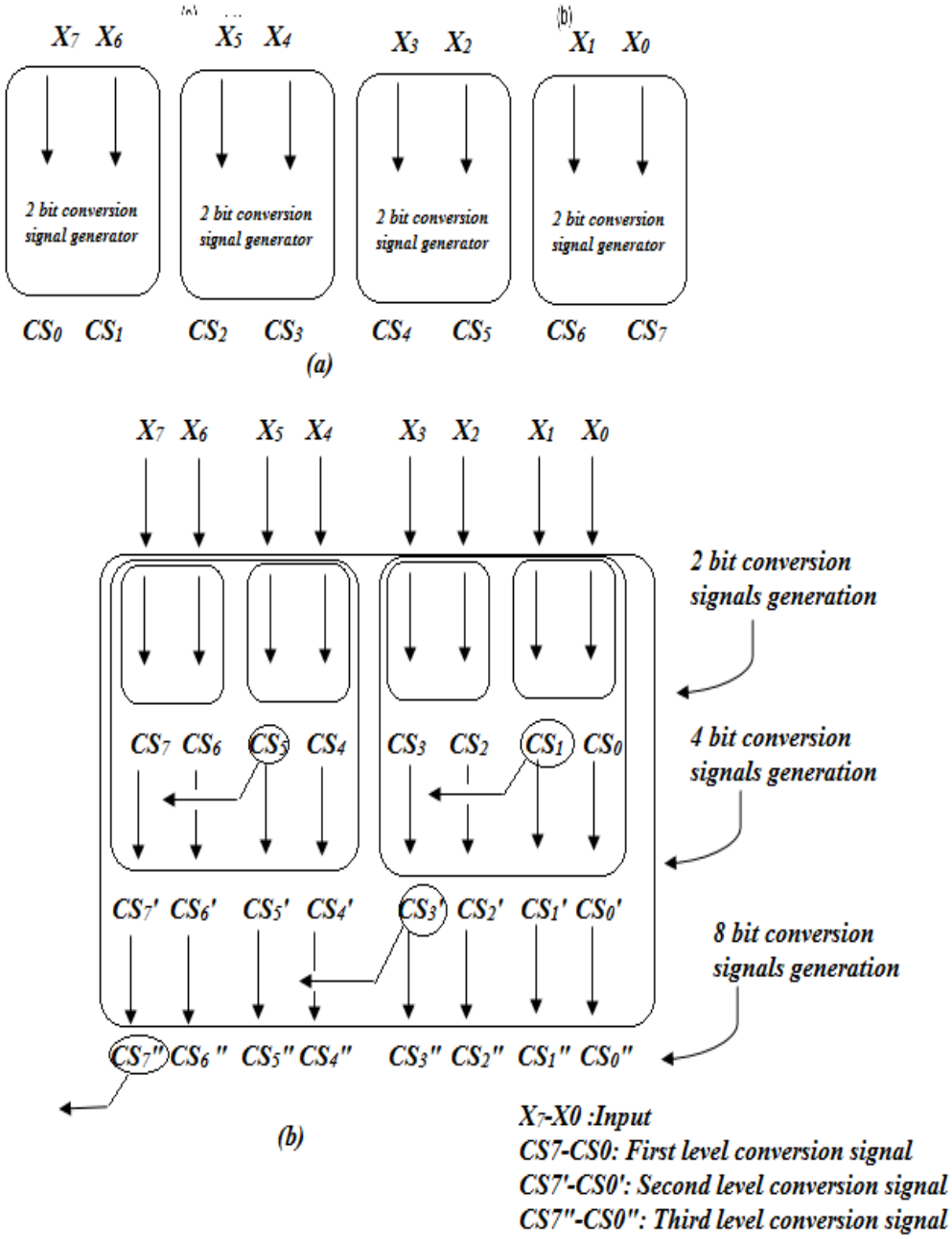



Fig. 5.6 Shows 2's complement conversion signal (a) finding conversion signal for 2 bit (b) finding conversion signal for 8 bit [5]

Third method to find two's complement is to divide PP into two parts--pp and cin, meeting PP=pp+cin, as Table 5.1 shows. This separation is for easy placement and routing seeing cin=$A_{2n+1}$, also for the skillful disposal of negative PP.

Table 5.1 Encoding of radix 4 booth [3]

| $A_{2n+1}$ | $A_{2n}$ | $A_{2n-1}$ | $E_k$ | PP | Pp | Cin |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | +B | +B | 0 |
| 0 | 1 | 0 | 1 | +B | +B | 0 |
| 0 | 1 | 1 | 2 | +2B | +2B | 0 |
| 1 | 0 | 0 | -2 | -2B | ~2B | 1 |
| 1 | 0 | 1 | -1 | -B | ~B | 1 |
| 1 | 1 | 0 | -1 | -B | ~B | 1 |
| 1 | 1 | 1 | 0 | 0 | 10`h3FF | 1 |

For the positive PP, pp=PP, cin=0. For scanning "111", we see the PP in two way, which are 10 'h0+0and 10'hFF+1. Generally, when scanning "111", we set the appropriate pp=10'hFF, cin=1.

We take 8 x 8 bit signed/unsigned multiplier for example. Multiplier A is extended its sign bit twice to 10 bits, marked as A'. Scanning A' with Radix-4 Booth encoding, we gain five PPs, marking them as PP0, PP1, PP2, PP3 and PP4_MSB. Of course, PP4_MSB comes from scanning the most significant three bit of A', meeting PP4_MSB =pp4_msb + cin_msb. We set pseudo product PP4, meeting PP4 = pp4 + cin4.

We could see, if the most significant three bit of A' is "111", set pp4=10'h000`cin4=1'b0. And then we find that cin3 and cin4 couldn't be "1" at the same time. Because of the twice signed extension, A[9]=A[8]. If cin3=1, A[7]=1; if cin4=1, A[9]=1,and then A[9]=A[8]=A[7]=1. But this case couldn't appear because the two line codes above shows avoiding it. With the disposal above, we can easily deal with the negative PP. Through recombining the pp and cin of PP, we entirely avoid the additive arithmetic of "pp + cin", without change for the sum of the five PPs. As Figure 5.7 shows, "X" means

signed extension. From Figure 5.7, the location of cin0, cin1 and cin2 doesn't change its weight, which couldn't affect the sum value of the five PPs. For cin3 and cin4, if cin4=0, the location of cin3 also doesn't change its weight; if cin3=0, summing up the four cin4 according to its weight could also get the value of its original weight. So this disposal doesn't change the sum of the five PPs, and with no additive arithmetic. [3]

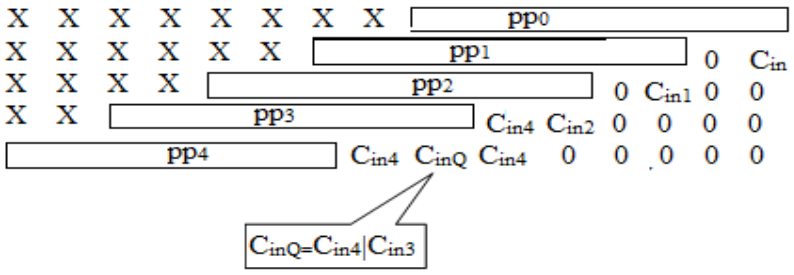
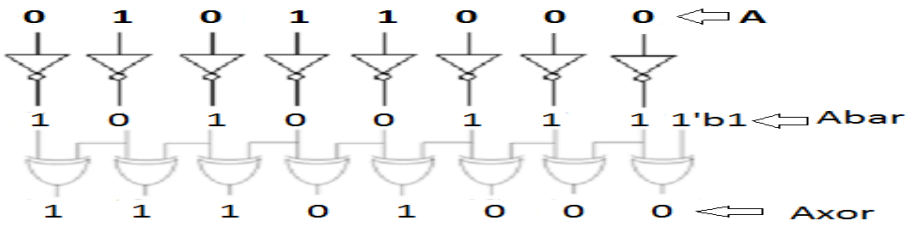
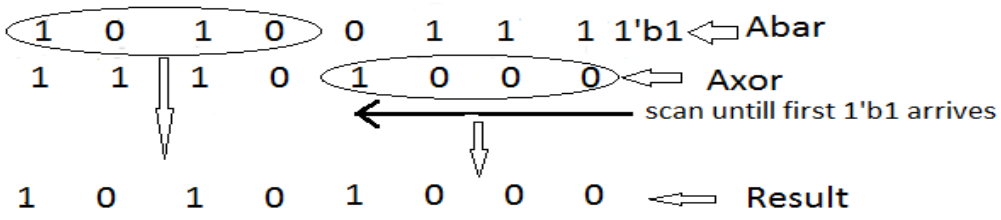



Fig. 5.7 Recombining of PP [3]

Finally one of the efficient methods to find 2's complement is discuss here, consider a 8 bit number A, to calculate 2's complement first is to invert all the bits of data, and denote them Abar. Next step is to perform "XOR" operation on Abar(0) with 1'b1, Abar(1) xor Abar(0), Abar(2) xor Abar(1) and so on, and denote them Axor.




(a)




(b)

Fig. 5.8 Shows method to find 2's complement of 8 bit data [1]

Now scan each bit of Axor until 1'b1 arrives, copy those bits into results and remaining bits will be copied from Abar. This has been shown in fig 5.8. And final array of partial products is shown in fig. 5.9. [1]

| b_p | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $PP_0$ | | | | | | $\overline{s_0}$ | $s_0$ | $s_0$ | $p_{08}$ | $p_{07}$ | $p_{06}$ | $p_{05}$ | $p_{04}$ | $p_{03}$ | $p_{02}$ | $p_{01}$ | $t_{00}$ |
| $PP_1$ | | | | | 1 | $\overline{s_1}$ | $p_{18}$ | $p_{17}$ | $p_{16}$ | $p_{15}$ | $p_{14}$ | $p_{13}$ | $p_{12}$ | $p_{11}$ | $t_{10}$ | $c_0$ | |
| $PP_2$ | | | 1 | $\overline{s_2}$ | $p_{28}$ | $p_{27}$ | $p_{26}$ | $p_{25}$ | $p_{24}$ | $p_{23}$ | $p_{22}$ | $p_{21}$ | $t_{20}$ | $c_1$ | | | |
| $PP_3$ | 1 | $\overline{s_3}$ | $t_8$ | $t_7$ | $t_6$ | $t_5$ | $t_4$ | $t_3$ | $t_2$ | $t_1$ | $t_0$ | $c_2$ | | | | | |

Fig. 5.9 Final partial products after applying 2's complement

# CHAPTER 6

# RESULTS AND CONCLUSION

This chapter introduces about the implementation and the simulation and synthesis results of different multiplier. Multiplier synthesized by Xilinx ISE 13.1and Synopsys Design Compiler Tool.

The Xilinx ISE13.1 is used for implementation of all the circuits. The Working Environment for the design is:

- Target Device : xc3s500e-5fg320
- Tool Version : Xilinx design tool 13.1
- Optimization Goal : Speed
- Total Slices : 4656
- Total LUTs : 9312
- Modelsim: 6.3f

### 6.1 Array Multiplier

#### 6.1.1 Booth's multiplier

Here simple 8 bit multiplier and 16 bit multiplier have been designed with the help of radix 4 booth's algorithm, first step in basic multiplication is partial product generation those were generated with the help of booths recoding table as shown in table 6.1. Here sign bit of the partial product are copied and extended to the $15^{th}$ bit position, no sign prevention method is used. Numbers of partial product rows are four for 8 bit multiplier and eight for 16 bit multiplier. Now second step in multiplication is accumulation of these partial product rows until 2 rows left, is done with the help of carry save adder, and last step in multiplication is accumulation of remaining two rows is done with the help of ripple carry adder. Multiplication of two 8 bit numbers is shown in Fig. 6.1.

| | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | $a_7$ | $a_6$ | $a_5$ | $a_4$ | $a_3$ | $a_2$ | $a_1$ $a_0$ |
| | | | | | | | | $b_7$ | $b_6$ | $b_5$ | $b_4$ | $b_3$ | $b_2$ | $b_1$ $b_0$ |
| $PP_0$ | $p_{07}$ | $p_{07}$ | $p_{07}$ | $p_{07}$ | $p_{07}$ | $p_{07}$ | $p_{07}$ | $p_{06}$ | $p_{05}$ | $p_{04}$ | $p_{03}$ | $p_{02}$ | $p_{01}$ | $p_{00}$ |
| $PP_1$ | $p_{17}$ | $p_{17}$ | $p_{17}$ | $p_{17}$ | $p_{17}$ | $p_{16}$ | $p_{15}$ | $p_{14}$ | $p_{13}$ | $p_{12}$ | $p_{11}$ | $p_{10}$ | | |
| $PP_2$ | $p_{27}$ | $p_{27}$ | $p_{27}$ | $p_{26}$ | $p_{25}$ | $p_{24}$ | $p_{23}$ | $p_{22}$ | $p_{21}$ | $p_{20}$ | | | | |
| $PP_3$ | $p_{37}$ | $p_{36}$ | $p_{35}$ | $p_{34}$ | $p_{33}$ | $p_{32}$ | $p_{31}$ | $p_{30}$ | | | | | | |

Fig 6.1 Multiplication of two 8 bit numbers

Table 6.1 Radix 4 Booth's recoding table

| $x_i$ | $x_{i-1}$ | $x_{i-2}$ | Partial product |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | A |
| 0 | 1 | 0 | A |
| 0 | 1 | 1 | 2A |
| 1 | 0 | 0 | -2A |
| 1 | 0 | 1 | -A |
| 1 | 1 | 0 | -A |
| 1 | 1 | 1 | 0 |

### 6.1.2 Multiplier with n/2+1 rows

Here radix 4 booth multiplier recoding table is modified and new recoding table is developed as shown in table 6.2.

Table 6.2 New radix 4 recoding table

| $x_i$ | $x_{i-1}$ | $x_{i-2}$ | Sgn | 2a | A |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 0 | 1 |
| 0 | 1 | 0 | 0 | 0 | 1 |
| 0 | 1 | 1 | 0 | 1 | 0 |
| 1 | 0 | 0 | 1 | 1 | 0 |
| 1 | 0 | 1 | 1 | 0 | 1 |
| 1 | 1 | 0 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 | 0 | 0 |

Generation of this table has been discussed in chapter 2 under section 2.2.4. By using this new recoding scheme partial product are generated, now partial products are increased from n/2 to n/2+1 i.e. 5 for 8 bit multiplier and 9 for 16 bit multiplier. Along with this sign prevention methods are also used. This makes partial products more regularized in structure. Again these partial products are accumulated with the help of carry save adder until two rows left and finally ripple carry adder is used to generate final result.

Multiplication of two 8 bit numbers using new recoding scheme is shown in fig 6.2. Comparison results of 8 bit and 16 bit multiplier with different technique is shown in table 6.3.

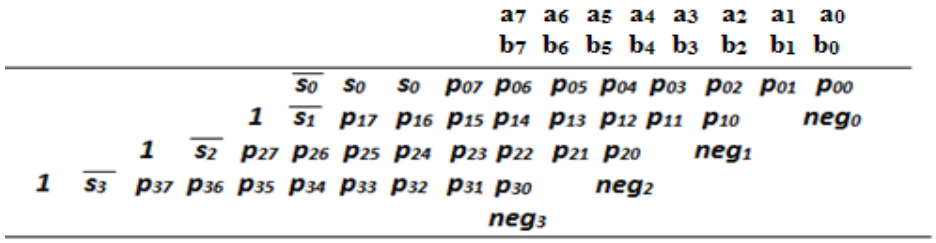



Fig 6.2 Multiplication of two 8 bit numbers using new recoding scheme

Table 6.3 Comparison results of 8 bit and 16 bit multiplier with different technique

| | Basic Booth Multiplier( 8*8 bit with n/2 partial product rows | Proposed Booth Multiplier( 8*8 bit with n/2 regular partial product rows) | Basic Booth Multiplier( 16*16 bit with n/2 partial product rows | Proposed Booth Multiplier( 16*16 bit with n/2 regular partial product rows) |
|---|---|---|---|---|
| **No. of Slices** | 91/4656 | 108/4656 | 295/4656 | 334/4656 |
| **Minimum Period** | 12.563 nS | 12.22 ns | 18.565ns | 15.48ns |
| **Maximum Frequency** | 79.59 MHz | 81.381 MHz | 53.86 MHz | 64.57 MHz |
| **Maximum combinational delay** | 21.76 ns | 21.5 ns | 39.15 ns | 38.49 ns |
| **Total dynamic power** | 4.015 mW | 2.82 mW | 20.22 mW | 17.06mW |
| **Area (1 unit = 1 nand gate area)** | 5446.55 | 4065.35 | 18,845.71 | 16,164.59 |

In next approach in order to make partial products more regularized and to reduce partial products from n/2+1 to n/2 different techniques were used. In order to make partial product regularized approach discussed in chapter 4 are used. And in order to reduce partial products from n/2+1 to n/2 different two's complement method (discuss in chapter 4) are used and compared. Final multiplier with regular partial product and n/2 partial product rows is shown in figure 6.3. Simulation results of 8 bit and 16 bit are shown in figure 6.4 and 6.5.

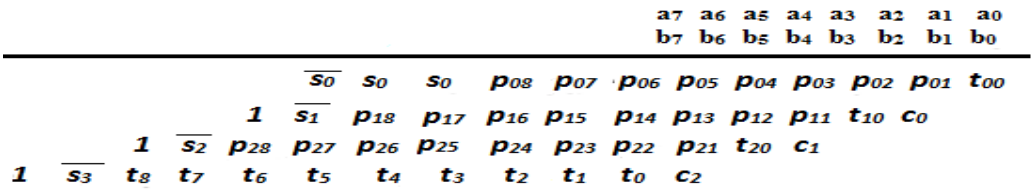



Fig 6.3 Multiplication of two 8 bit numbers having n/2 regular partial products

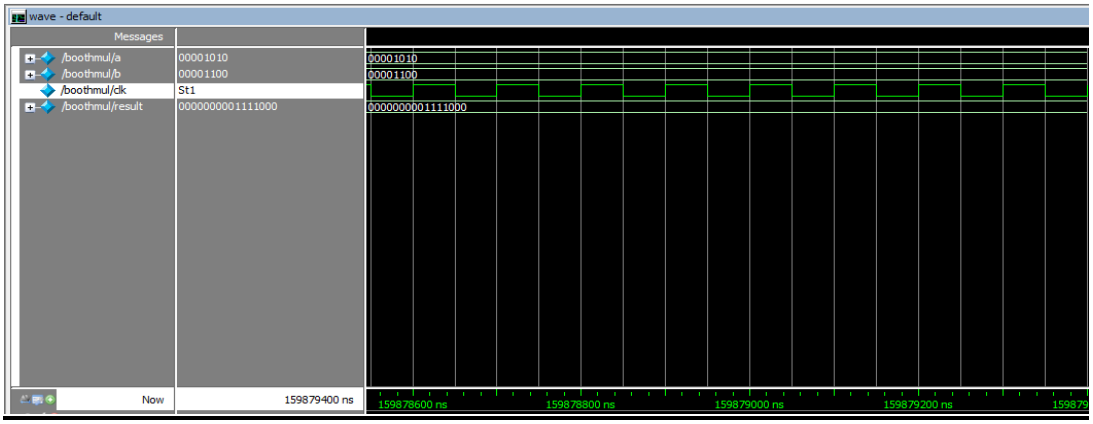



Fig 6.4 Simulation results of 8*8 bit radix 4 booth multiplier

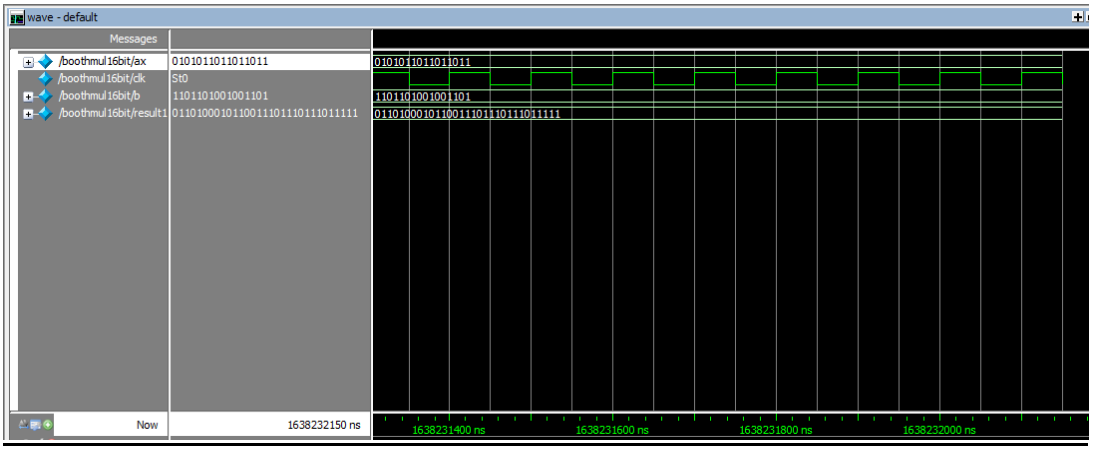



Fig. 6.5 Simulation results of 16*16 bit radix 4 booth multiplier

Comparison of 8 bit and 16 bit multipliers with different adders in terms of area and delay is shown in table 6.4 and 6.5 respectively.

Table 6.4 Comparison results of 8 bit multiplier with different adders

| | Proposed Multiplier (8 bit multiplier) | | |
|---|---|---|---|
| | Multiplier with n/2 regular partial product rows (RCA adder) | Multiplier with n/2 regular partial product rows (CLA adder) | Multiplier with n/2 regular partial product rows (CSA adder) |
| **No. of Slices** | 107/4656 | 201/4656 | 108/4656 |
| **Minimum Period** | 12.287ns | 2.009 ns | 2ns |
| **Maximum Frequency** | 81.38MHz | 98.246MHz | 102.763MHz |
| **Maximum combinational delay** | 21.6 ns | 20.054 ns | 16.8 ns |
| **Total dynamic power** | 2.82mW | 2.07mW | 1.341mW |
| **Area (1 unit = 1 nand gate area)** | 4065.23 | 4786.43 | 2765.76 |

Table 6.5 Comparison results of 16 bit multiplier with different adders

| | Proposed Multiplier (16 bit multiplier) | | |
|---|---|---|---|
| | Multiplier with n/2 regular partial product rows (RCA adder) | Multiplier with n/2 regular partial product rows (CLA adder) | Multiplier with n/2 regular partial product rows (CSA adder) |
| **No. of Slices** | 334/4656 | 412/4656 | 359/4656 |
| **Minimum Period** | 15.48ns | 15.8 ns | 14.2ns |
| **Maximum Frequency** | 64.57 MHz | 63.04 MHz | 62.645 MHz |
| **Maximum combinational delay** | 38.49 ns | 35.59 ns | 28.41 ns |
| **Total dynamic power** | 17.06mW | 14.07mW | 12.57mW |
| **Area (1 unit = 1 nand gate area)** | 16164 | 17134 | 13483 |

### 6.2 Conclusion

A new hardware implementation of a low power modified booth multiplier is developed based on 2's complement technique. Verilog is used to implement the code for radix 4

modified booth multiplier, followed by Xilinx design tool 13.1.

First partial products are generated directly with the help of booth encoding table and these partial products are accumulated with the help of carry save adder and finally last two partial products are added with the help of ripple carry adder.

Further booth encoding table is converted into combination circuits which results into n/2+1 PP rows ,further direct 2's complement technique are implement to reduce n/2+1 rows to n/2 ,that shows very little improvement in delay but significant improvement in area and power.

Finally different adder were used at the third stage of multiplication such as carry select adder , carry lookahed adder and ripple carry adder and then compared. Carry select adder shows significant improvement in terms of delay.

### 6.3 Future Scope

The present work on the multiplier architecture can be extended in various directions. Some suggestions are given below:

1. Higher order radix algorithm can be implemented for further reducing partial products in order to enhance the speed of the multiplier.
2. Some other method such as wallace tree and dada tree in accumulation stages can be used to improve the delay of the circuit.

# List of Publication

- Bipin and Sakshi, "Design of multiplier using regular partial products," *International Journal of Advanced Research in Computer and Communication Engineering (IJARCCE)* , Vol. 2, Issue 7, pp 2559-2562, July 2013.

# REFERENCES


[1] M. Kumar and R. Verma, "Disposition (reduction) of (negative) partial product for radix 4 booth's algorithm," in *IEEE world congress on information and communication technologies*, Dec 2011, pp.1169-1174.

[2] S.R. Kuang, J.P. Wang, and C.Y. Guo, "Modified booth multipliers with a regular partial product array," in *IEEE Transaction on circuits and systems-II*, vol. 56, no. 5, May 2009, pp. 404-408.

[3] Z. Li, H. Chen, X. Yang "Research on disposal of negative partial products for booth algorithm," in Proc. *IEEE Conference on information theory and information security (ICITIS)*, Dec 2010, pp 1115-1117.

[4] W.C. Yeh and C.W. Jen, "High-speed booth encoded parallel multiplier design," in *IEEE Transaction on computer society,* vol. 49, no. 7, Jul. 2000, pp. 692–701.

[5] J.Y.Kang and J.L. Gaudiot "A simple high-speed multiplier design ,"*IEEE Transaction on computer society*, vol. 55, no. 10, Oct. 2006 ,pp. 1253–1258.

[6] O. Salomon, J.M. Green, and H. Klar, "General algorithms for a simplified addition of 2's complement numbers," in *IEEE Solid-state circuits*, vol. 30, no. 7, Jul. 1995,pp. 839–844.

[7] E.de Angel and E. E. Swartzlander, "Low power parallel multipliers," *IEEE workshop on VLSI signal process. IX*, Oct 1996, pp. 199–208.

[8] A. A. Farooqui, V. G. Oklobdzija "General data-path organization of a MAC unit for VLSI implementation of DSP processors". *IEEE interactive symposium on ISCAS*, vol.2, June 1998.

[9] F. Elguibaly, "A Fast parallel multiplier-accumulator using the modified booth's algorithm", *IEEE Transactions on analog and digital signal processing, vol*. 47 Sept. 2000, pp 902-908

[10] O.Hasan, S.Kort "Automated formal synthesis of wallace tree multipliers," *IEEE Transactions on MWSCAS*, Aug 2007 pp 293-296.

[11] S.SriSakthi, N.Kayalvizhi, "Power aware and high speed reconfigurable modified booth's multiplier", *IEEE International conference on RAICS*, Feb. 2011, pp 352-356.

[12] S.Saravanan, M.Madheswaran "Design of hybrid encoded booth's multiplier with reduced switching activity technique and low power 0.13µm adder for dsp block in wireless sensor node", *IEEE International conference on wireless communication and sensor computing,* Jan. 2010, pp 1-6

[13] F.Lamberti, N. Andrikos, E.Antelo, and P.Montuschi, "Reducing the computation time in (short bit-width) two's complement multipliers", *IEEE Transactions on computers*, vol. 60, Feb. 2011.

[14] K.H.Chen and Y.S.Chu, "A low power multiplier with spurious power suppression technique*," IEEE Transaction on very large scale integration (VLSI) system*.vol. 15, no.7, pp. 846–850, July 2007.

[15] M.W.Allam and M.I.Elmasry "Low power implementation of fast addition algorithms", *International conference on electrical and computer engineering,* vol.2*,* May 1998.

[16] C.S.Wallace, "A suggestion for a fast multiplier," *IEEE Transaction computers*, vol. 13, 1964, pp. 14-17.

[17] H. Kaul, M.A. Anders, S.K. Mathew, S.K. Hsu, A. Agarwal, R.K. Krishnamurthy, and S. Borkar, "A 300 mv 494gops/w reconfigurable dual-supply 4-way simd vector processing accelerator in 45 nm cmos," *IEEE Journal on. solid state circuits*, vol. 45, no. 1, pp. 95- 101, Jan. 2010.

[18] P.E.Madrid, B.Millar, and E.E.Swartzlander "Modified booth algorithm for high radix fixed-point multiplication" *IEEE Transactions on very large scale integration (VLSI) systems*, vol. 1, no. 2, June 1993, pp 164-167

[19] V. Garofalo, N. Petra, D. De Caro, A. G. M. Strollo, and E. Napoli, "Low error truncated multipliers for DSP applications," in Proc. *15th IEEE International conference on electronics circuits system*, 2008, pp. 29–32.

[20] S. Quan, Q. Qiang, and C. L. Wey, "A novel reconfigurable architecture of low-power unsigned multiplier for digital signal processing," *IEEE International symposium on circuits system*, May 2005, pp. 3327–3330.

[21] O. Gustafsson, "Lower bounds for constant multiplication problems," *IEEE Transaction on circuits syst. ii, analog digital signal processing* , vol. 54,no. 11, pp. 974–978, Nov. 2007

[22] P. Mokrian, M. Ahmadi, G. Jullien, and W. C. Miller, A reconfigurable digit multiplier architecture," *IEEE. Conference on electronics comput. eng.*, May

2003, pp. 125–128.

[23] C. L. Wey and J. F. Li, "Design of reconfigurable array multipliers and multiplier-accumulators," *IEEE International conference asia-pacific circuits syst.*, Dec. 2004, pp. 37–40.

[24] Behrooz Parhami, Computer Arithmetic, Algorithms and Hardware Design, 2nd edition.

[25] M.Morris Mano, Computer System Architecture, 3rd edition.