FPGA IMPLEMENTATION OF EFFICIENT MODIFIED BOOTH WALLACE MULTIPLIER

Thesis submitted in partial fulfilment of the requirements
for the award of degree of

Master of Technology
in
VLSI Design and CAD

Submitted by

Simran Kaur
Roll No. 600961020

Under the Guidance of

Ms. Manu Bansal
Assistant Professor

Department of Electronics & Communication Engineering
Thapar University, Patiala-147004, India
June, 2011

CERTIFICATE

I hereby certify that the work which is being presented in this thesis entitled **"FPGA Implementation of Efficient Modified Booth Wallace Multiplier"**, is an authentic record of my own work carried out as the requirements for the award of degree of Master of Technology in VLSI Design & CAD at Thapar University, Patiala under the guidance of **Mrs. Manu Bansal**, Assistant Professor, Electronics and Communication Engineering Department, Thapar University.

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

Dated: 27 June 2011

Simran Kaur
(Simran Kaur)

Roll No. 600961020

This is to certify that the above statement made by the candidate is correct and to the best of my knowledge.

Manu Bansal
(Mrs. Manu Bansal)
Assistant Professor, ECED
Thapar University, Patiala

Counter Signed by:

(Dr. A. K. Chatterjee)
Head, ECED
Thapar University, Patiala

(Dr. S. K. Mohapatra)
Dean of Academics Affairs
Thapar University, Patiala

ABSTRACT

The multiplication operation is present in many parts of a digital system or digital computer, most notably in signal processing, graphics and scientific computation. With advances in technology, various techniques have been proposed to design multipliers, which offer high speed, low power consumption and lesser area. Thus making them suitable for various high speed, low power compact VLSI implementations. These three parameters i.e. power, area and speed are always traded off.

This thesis work is devoted for the design of a low power high-speed Booth Wallace multiplier and its implementation on reconfigurable hardware. For arithmetic multiplication, various multiplication architectures like array multiplier, Booth multiplier, Wallace tree multiplier and Booth Wallace multiplier have been thoroughly discussed. It has been found that Booth Wallace multiplier is most efficient among all, giving optimum delay, power and area for multiplication. Low power modified Booth recoder and pipelining techniques have been used to reduce power and delay.

Further, the VHDL coding of Booth Wallace multiplier for 8×8 bit, 16×16 bit and 32× 32 bit multiplication and their FPGA implementation by Xilinx Synthesis Tool on Spartan 3E kit have been done. The output has been displayed on LCD of Spartan 3E kit. The synthesis results show that the computation time for calculating the product of 8×8 bit is 8.78 ns, while for the product of 16×16 bit is 10.13 ns and 32× 32 bit is 13.38ns.

ACKNOWLEDMENT

I wish to express my sincere gratitude to my supervisor **Mrs. Manu Bansal** for her invaluable guidance and advice during every stage of this endeavour. I am greatly indebted to her for encouragement and support without which, it would not have been possible for me to complete this undertaking successfully. Her insightful comments and suggestions have continually helped me to improve my understanding.

My sincere thanks are due to **Dr. A. K. Chatterjee**, Head of the Department, Department of Electronics and Communication Engineering for providing the constant encouragement and providing the facilities in the department for the completion of my thesis work. I express my gratitude towards **Mrs. Alpana Agarwal**, PG Coordinator, Department of Electronics and Communication Engineering**,** for her valuable guidance and encouragement.

I also want to thanks my friends Aman, Puneet, Shiva and Amit for accompanying me during the most outstanding year of my life and by me in every situation.

I take pride of myself being daughter of ideal parents for their over lasting desire, sacrifice, affection blessings and help without which it would not have been possible for me to complete my studies. I would like to thank my sister Japneet for moral encouragement and support.

Last but not least, I would like to thank God for all good deeds.

(Simran Kaur)

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABBREVIATIONS

DSP	Digital Processing System
MAC	Multiply and Accumulate
FFT	Fast Fourier Transform
LSD	Least Significant Digit
CSA	Carry Save Adder
CLA	Carry Look-ahead Adder
PP	Partial Product
MBE	Modified Booth Encoding
MBA	Modified Booth Algorithm
LSB	Least Significant Bit
BE	Booth Encoder
BS	Booth Selector
WT	Wallace Tree
LCD	Liquid Crystal Display
CLC	Carry look-Ahead Circuit
OC	Output Carry
FPGA	Field Programmable Gate Array
NGD	Native Generic Database
CLBs	Configurable Logic Blocks
LUT	Look Up Table
PAR	Place and Route

CHAPTER 1

INTRODUCTION

Multiplication is one of the basic functions used in digital signal processing (DSP). It requires more hardware resources and processing time than addition and subtraction. In fact, 8.72% of all instructions in a typical processing unit is multiplier [1]. In computers, a typical central processing unit devotes a considerable amount of processing time in implementing arithmetic operations, particularly multiplication operations. Most high performance digital signal processing systems rely on hardware multiplication to achieve high data throughput. Multiplication is an important fundamental arithmetic operation. Multiplication-based operations such as Multiply and Accumulate (MAC) are 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 [2]. Since multiplication dominates the execution time of most DSP algorithms, so there is a need of high speed multiplier. Currently, multiplication time is still that dominant factor in determining the instruction cycle time of a DSP chip. The multiplier is a fairly large block of a computing system. The amount of circuitry involved is directly proportional to square of its resolution i.e., a multiplier of size of n bits has O (n^2) gates.

In the past, many novel ideas for multipliers have been proposed to achieve high performance. The demand for high speed processing has been increasing as a result of expanding computer and signal processing applications. Higher throughput arithmetic operations are important to achieve the desired performance in many real-time signal and image processing applications. One of the key arithmetic operations in such applications is multiplication and the development of fast multiplier circuit has been a subject of interest over decades. Reducing the time delay and power consumption are very essential requirements for many applications.

This thesis presents various multiplier architectures. Multiplier architectures fall generally into two categories i.e., “tree” multipliers and “array” multipliers. Tree multipliers add as many partial products in parallel as possible and therefore, are very high performance architectures. Unfortunately, tree multipliers are very irregular, hard to layout and hence large. Array multipliers, on the other hand, are very regular, small in size, but suffer in latency and propagation delay [3]. Due to array organization, determining the propagation delay of array multiplier is not straightforward. Multiplier based on Modified Booth algorithm andWallace addition is one of the fast and low power multiplier. The speed of modified Booth Wallace multiplier can further be increased by pipelining.

The speed of the multipliers is greatly improved by properly deciding the number of pipeline stages and the positions for the pipeline registers to be inserted [4]. Pipelines are widely used to improve the performance of digital circuits, since they provide a simple way of implementing parallelism from streams of sequential operations. In a pipelining system, the maximum operating frequency is limited by slowest stage which has the longest delay. As more stages are inserted in the pipeline, each stage becomes shorter, and ideally presents a smaller delay. Theoretically, the pipeline depth can be pushed to a level of using a single gate between two registers [5]. But usually, there is a compromise between performance improvements obtained with increased pipeline depth and the penalties imposed by the additional memory elements inserted in between the stages. So trade-off between the reduction in delay due to pipelining and increase in area must always be considered.

1.1Thesis Objective

The main objective of this thesis is design and implementation of a Booth Wallace multiplier. The programming objectives of Booth Wallace multiplier fall into following categories:

- Accuracy: The multiplier produces the correct result.
- Speed: The multiplier produces high speed.
- Area: The multiplier occupies less number of slices and LUTs.
- Power: The multiplier consumes less power.

1.2 Thesis Organization

This thesis is organized into six subsequent chapters as follows:

Chapter 2 Encloses general concept of binary multiplication and describe functionality and architecture of various array multipliers.

Chapter 3 Describe functionality and architecture of Wallace tree multipliers and different compressors. This chapter provides comparison of various multipliers.

Chapter 4 Describe the design of modified Booth Wallace multiplier. This chapter describe various modules of multiplier. Implementation of various multipliers is given in form of block diagram.

Chapter 5 Encloses the basic description of design methodology and design flow of FPGA, how the design is implemented on FPGA(Spartan 3E).

Chapter 6 Encloses the result and conclusion of thesis work in terms of various results obtained during implementation.

1.3 Tools Used

The tools used in the thesis are as follows:

Simulation Software:

- ISE 9.2i is used for design and implementation.
- ModelSim 6.1e is used for modelling and simulation.

Hardware Used:

Xilinx Spartan 3E (Family), XC3S500E (Device), FG320 (Package) FPGA device.

CHAPTER 2

ARRAY MULTIPLIERS

2.1 Introduction

Multiplication is a mathematical operation that at its simplest is an abbreviated process of adding an integer to itself a specified number of times. A number (multiplicand) is added to itself a number of times as specified by another number (multiplier) to form a result (product). The multiplicand is multiplied by each digit of the multiplier beginning with the rightmost, LSD. Intermediate results (partial-products) are placed one atop the other, offset by one digit to align digits of the same weight. The final product is determined by summation of all the partial-products. Multiplication involves three main steps [6]:

- Partial product generation
- Partial product reduction
- Final addition

For the multiplication of an n-bit multiplicand with an m-bit multiplier, m partial products are generated and product formed is $n + m$ bits long.

The multiplier architectures can be generally classified into following categories:

- Serial multiplier
- Parallel multiplier
- Serial-parallel

2.2 Serial Multipliers

The simplest method to perform multiplication is to add series of partial products. The serial multipliers use a successive addition algorithm. They are simple in structure because both the operands are entered in a serial manner. Therefore, the physical circuit requires less hardware and a minimum amount of chip area. However, the speed performance of the serial multiplier is due to the operands entered sequentially.

2.3 Parallel Multipliers

Most advanced digital systems incorporate a parallel multiplication unit to carry out high speed mathematical operations. A microprocessor requires multipliers in its arithmetic logic unit and a digital signal processing system requires multipliers to implement algorithms such as convolution and filtering. Some examples of parallel multipliers are array multipliers such as Braun multipliers, Booth multipliers and Baugh-Wooley multipliers, as well as the tree multipliers like Wallace multipliers. Array multipliers have a regular layout, although tree multipliers are generally faster. Parallel multipliers [7] present high-speed performance, but are expensive in terms of silicon area and power consumption because in parallel multipliers both the operands are input to the multiplier in parallel manner. Therefore, the circuitry occupies a much larger area and is more complex as compared to serial multipliers. These multipliers are extensively used in RISC, DSP and graphic acceleration.

2.3.1 Array Multipliers

Array multiplier can be classified into following categories:

- Braun Multiplier
- Booth Multiplier
- Modified Booth Multiplier
- Baugh -Wooley Multiplier

2.3.1.1 Braun Array Multiplier

Braun Array multiplier is well known due to its regular structure. It is a simple parallel multiplier that is commonly known as carry save array multiplier. This multiplier is restricted to performing multiplication of two unsigned numbers. It consist of array of AND gates and adders arranged in iterative structure that does not require logic registers. This is also known as the non-additive multiplier since it does not add an additional operand to result of multiplication [7]. A four-bit Braun multiplier is shown in Figure 2.1. To perform N-bit by N-bit multiplication, the N-bit multiplicand A is multiplied by N-bit multiplier B to produce product. The unsigned binary numbers A and B can be expressed as:

$$A = \sum_{i=0}^{n-1} A_i \,.\, 2^i \tag{2.1}$$

$$B = \sum_{j=0}^{n-1} B_j \,.\, 2^j \tag{2.2}$$

The product of A and B is P and it can be written in the following form:

$$P = \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} A_i \,.\, B_j 2^{i+j} \tag{2.3}$$

A $n \times n$ bit multiplier requires $n(n-1)$ adders and n^2AND gates. In this array multiplier, counters are connected in a serial fashion for all bit slices of partial product.

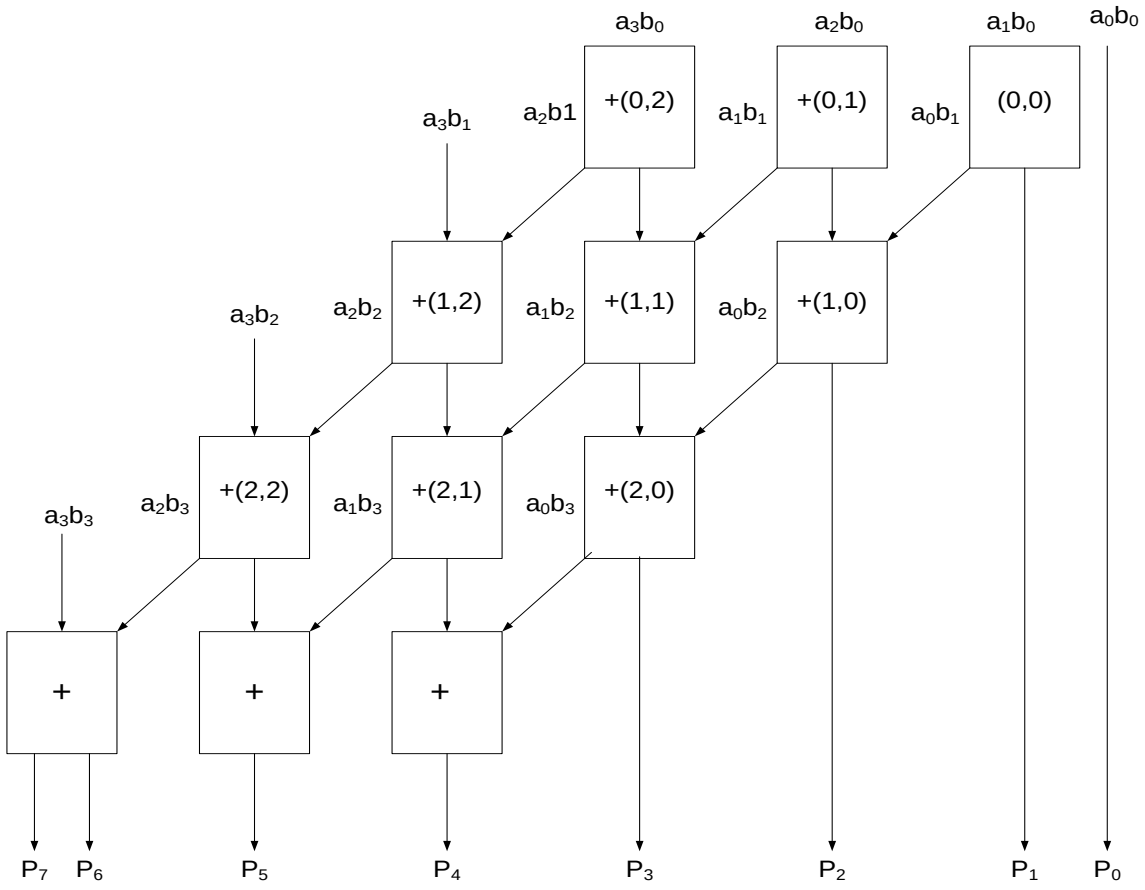

Figure 2.1: Braun multiplier [8]

In the simple array, each row of [3:2] compressors adds a partial product to the partial sum, generating a new partial sum and a sequence of carries. The delay of the array depends on the depth of the array. Therefore, the summing time for the simple array is

$N-2[3:2]$ compressor delays, where N is the number of partial products. The array multiplier originates from the multiplication parallelogram.

Multiplier circuit is based on add and shift algorithm. Each partial product is generated by the multiplication of the multiplicand with one multiplier bit. The partial product are shifted according to their bit orders and then added. Addition is mainly done by carry save algorithm in which every carry and sum signal is passed to the adders of the next stage. Final product is obtained in a final adder by any fast adder.

2.3.1.2 Advantages of Braun Array Multiplier

First advantage of the array multiplier is that it has a regular structure. Since it is regular, it is easy to layout and has a small size. The design time of an array multiplier is much less than that of a tree multiplier. A second advantage of the array multiplier is its ease of design for a pipelined architecture.

2.3.1.3 Limitations of Braun Array Multiplier

Major limitation of Braun array multiplier is its size. As operand sizes increase, arrays grow in size at a rate equal to the square of the operand size. The large size of full arrays typically prohibits their use, except for small operand sizes, or on special purpose math chips where a major portion of the silicon area can be assigned to the multiplier array. Another problem with array multipliers is that the hardware is underutilized [6].

2.4 Booth Multiplier

Conventional array multipliers, like the Braun multiplier and Baugh Woolley multiplier achieve comparatively good performance but they require large area of silicon, unlike the add-shift algorithms, which require less hardware and exhibit poorer performance. The Booth multiplier makes use of Booth encoding algorithm in order to reduce the number of partial products by considering two bits of the multiplier at a time, thereby achieving a speed advantage over other multiplier architectures. This algorithm is valid for both signed and unsigned numbers. It accepts the number in 2's complement form, based on radix-2 computation [7].

2.4.1 Booth Recoding

Andrew D. Booth proposed the Booth recoding, or Booth algorithm in 1951 [10]. This method can be used to multiply two 2's complement number without the sign bit extension. Booth observed that when strings of '1' bits occur in the multiplicand the number of partial products can be reduced by subtraction. Table 2.1 shows the booth algorithm operation. Booth classified group of bits into beginning, middle or end of run.

Table 2.1

Booth Algorithm [7]

X_i	X_{i-1}	**Operations**	**Comments**	Y_i
0	0	Shift only	String of zeros	0
1	0	Sub and shift	Beg of string of ones	1
1	1	Shift only	String of zeros	0
0	1	Add and shift	End of string of ones	1

String of zero's avoid arithmetic, so these can be left alone. Booth algorithm changed the original algorithm by looking at two bits of multiplier in contrast to the old algorithm that looks at only one bit at a time. New algorithm has four cases, depending on the values of two bits. Let us assume that the pair of bits examined consist of current bit and bit to right. Second step is to shift the product right.

2.4.2 Booth Example

Assume that two numbers to be multiplied are B = -34 = - $(0100010)_2$ and A = 22 = $(0010110)_2$. Representing both operand and their negation in signed 2's compliment:

22: 0010110, -22: 1101010

34: 0100010, -34: 1011110

Table 2.2 shows example of Booth algorithm. $[A]$ and $[Q]$ are two registers in which

result is to be stored. $[M]$ is multiplicand. Two bits of multiplier are recoded at time to perform the required action according to Table 2.1. Multiplier is first appended with a '0' at LSB position and thereafter bits are recoded. The upper half of the result 1111010 0010100 is in register $[A]$ while lower half is in register$[Q]$. The product is given in signed 2's complement and its actual value is negative of the 2's complement:

$$A \times B = -00001011101100 = -(748)_{10}$$

Table 2.2

Booth Example

q_iq_{i-1}	**Action**	$[M]$	**0010110**			
		$[A]$	**0000000**	$[Q]$	**1011110**	**0**
00	Right shift		0000000			
			0101111	0		
01	$-A$	+	1101010			
			1101010		0101111	0
	Right shift		1110101			
			0010111	1		
11	Right shift		1111010			
			1001011	1		
11	Right shift		1111101		0100101	1
11	Right shift		1111110			
			1010010	1		
01	$+A$	+	0010110			
			0010100			
			1010010	1		
	Right shift		0001010			
			0101001	0		
10	$-A$	+	1101010			

		1110100 0101001	 0
	Right shift	1111010 0010100	 1

The serial recoding scheme is usually applied in serial multipliers. The advantage of this method is that the partial product circuit is simpler and easy to implement. Booth's algorithm results in reduction in number of add/subtract operations if multiplier contains sequence of 1's and 0's. Worst case scenario that occurs in booth algorithm is if a sequence such as 01010101...01 is encountered, where there are $\frac{n}{2}$ (where n is multiplier length) isolated 1's which forces $\frac{n}{2}$ subtractions and $\frac{n}{2}$ additions. This is worse case standard multiplier.

2.5 Modified Booth Algorithm

The modified Booth encoding (MBE), or modified Booth's algorithm (MBA), was proposed by O. L. Macsorley in 1961 [11]. The recoding method is widely used to generate the partial products for implementation of large parallel multipliers, which adopts the parallel encoding scheme. One of the solutions of realizing high-speed multipliers is to enhance parallelism, which helps to decrease the number of subsequent stages. The original version of Booth algorithm (Radix-2) had two drawbacks:

- The number of add subtract operations and the number of shift operations becomes variable and becomes inconvenient in designing parallel multipliers.
- The algorithm becomes inefficient when there are isolated 1's.

These problems can be overcome by modified Booth algorithm. MBA process three bits at a time during recoding. Recoding the multiplier in higher radix is a powerful way to speed up standard Booth multiplication algorithm. In each cycle a greater number of bits can be inspected and eliminated therefore, total number of cycles required to obtain products get reduced. Number of bits inspected in radix r is given by $n = 1 + \log_2 r$. Algorithm for modified booth is given below [12]:

Consider two n-bit numbers X and Y to be multiplied where Y can be expressed as:

$$Y = -Y_{n-1}2^{n-1} + Y_{n-2}2^{n-2} + \cdots + Y_0 2^0 \quad (2.4)$$

$$\begin{aligned} Y = {} & (-2Y_{n-1} + Y_{n-2} + Y_{n-3})2^{n-2} \\ & + (-2Y_{i-3} + Y_{i-4} + Y_{i-5})2^{i-4} \\ & + \cdots + (-2Y_1 + Y_0 + Y_{-1})2^0 \end{aligned} \quad (2.5)$$

Where $Y_{-1} = 0$ and $Y_{i-3}2^{i-2} - Y_{i-2}2^{i-4} = Y_{i-3}$ have been used in the expression. Eq. (2.5) can be represented by

$$Y = \sum_{i=0}^{i=\frac{n}{2}-1} (-2Y_{2i+1} + Y_{2i} + Y_{2i-1}) . 2^{2i} = \sum_{i=0}^{i=\frac{n}{2}-1} Y_i . 2^i \quad (2.6)$$

$$X.Y = \left(-X_{n-1}2^{n-1} + \sum_{i=0}^{n-2} X_i . 2^i\right)\left(-Y_{i-1}2^{n-1} + \sum_{j=0}^{n-2} Y_j . 2^j\right) \quad (2.7)$$

$$X.Y = \left(-X_{n-1}2^{n-1} + \sum_{i=0}^{n-2} X_i . 2^i\right)\left(\sum_{j=0}^{\frac{n}{2}-1} Y_j . 2^{2j}\right) \quad (2.8)$$

In each cycle of radix-4 algorithm, 3 bits are inspected and two are eliminated. Procedure for implementing radix-4 algorithm is as follows

- Append a 0 to the right of LSB.
- Extend the sign bit 1 position if necessary to ensure that n is even.
- According to the value of each vector, find each partial product.

Table 2.3

Modified Booth Algorithm [9]

Y_{2i+1}	Y_{2i}	Y_{2i-1}	**Recoded Digit**	**Operand Multiplication**
0	0	0	0	0*Multiplicand
0	0	1	+1	+1*Multiplicand
0	1	0	+1	+1*Multiplicand
0	1	1	+2	+2*Multiplicand
1	0	0	-2	-2*Multiplicand
1	0	1	-1	-1*Multiplicand

1	1	0	-1	-1*Multiplicand
1	1	1	0	0*Multiplicand

Radix-4 encoding reduces the total number of multiplier digits by a factor of two, which means in this case the number of multiplier digits will reduce from 16 to 8. Booth's recoding method does not propagate the carry into subsequent stages. This algorithm groups the original multiplier into groups of three consecutive digits where the outermost digit in each group is shared with the outermost digit of the adjacent group. Each of these group of three binary digits then corresponds to one of the numbers from the set {2, 1 0,-1,-1}. Each recoder produces a 3-bit output where the first bit represents the number 1 and the second bit represent number 2. The third and final bit indicates whether the number in the first or second bit is negative.

2.5.1 Modified Booth Example

Assume two numbers to be multiplied are A = 34 and B = -42.

Multiplicand A = 34 = 00100010

Multiplicand B = -42 = 11010110(2's Complement)

$A \times B =$ -1428

Table 2.4

Modified Booth Example

									0	0	1	0	0	0	1	0		34
									1	1	0	1	0	1	1	0		-42
						1	1	1	1	0	1	1	1	1	0	0		PP1
				1	1	0	0	1	0	0	0	1	0	0				PP2
		1	1	0	0	0	1	0	0	0	1	0						PP3
1	0	1	1	1	0	1	1	1	1	0								PP4
1	1	1	1	1	1	0	1	0	0	1	1	0	1	1	0	0		-1428

Table 2.4 shows Modified Booth example. PP1, PP2, PP3, PP4 are the partial products formed. The first partial product is determined by three digits LSB of multiplier with a

appended zero. This 3 digit number is 100 which mean the multiplicand A has to multiply by -2. To multiply by -2, the process takes two's complement of the multiplicand value and then shift left one bit of that product. Hence, the first partial product is 110111100. All of the partial products will have nine bits length.

Next, the second partial product is determined by next three bits i.e. multiply by 2. Multiply by 2 means the multiplicand value has to shift left one bit. So, the second partial product is 001000100. Similarly the third partial product have to multiply by 1. So, the third partial product is the multiplicand value namely 000100010. The fourth partial product is determined by next three bits i.e. to multiply by -1. Multiply by -1 means the multiplicand has to convert to two's complement value. So, the forth partial product is 111011110.

LSB of each block gives information about sign bit of the pervious block, and there are never any negative products before the least sgnificant block, so LSB of first block is always taken to be zero. Block diagram of an $n \times n$ – bit modified Booth multiplier is shown in Figure 2.2. It consists of the Booth encoder and the sign extension bits, the multiplier array, which comprises the partial product's generator and 1-bit adders, and the final stage adder, which executes the $2n$-bit addition.

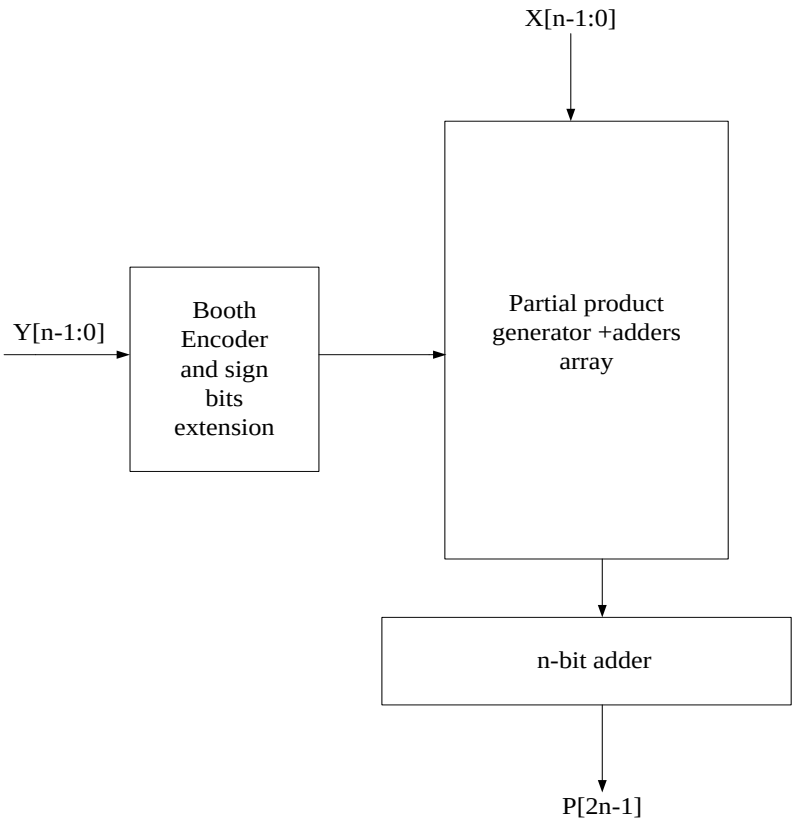

Figure 2.2: Block diagram of a $n \times n$ modified Booth multiplier

There are two operations in implementing a modified Booth multiplier design, which are generating the partial products and accumulating the entire partial products. In producing partial products the Booth Encoder (BE) and Booth Selector (BS) i.e. partial product generator are used. Accumulation of partial product is done by the adder or compressor circuit. BE decode the multiplier signal and output is used by BS to generate partial product. Ideally, the performance and size of multiplier circuit are dependent on these two operations.

The BE can be designed in many ways and the design will be trade-off between area and speed. Several Modified Booth Encoder (MBE) techniques have been evaluated and based on evaluation, Wen-Chang's MBE with slight modification is most efficient [13]. Wen-Chang MBE with modification is shown in Figure 2.3. Booth Selector is shown in Figure 2.4.

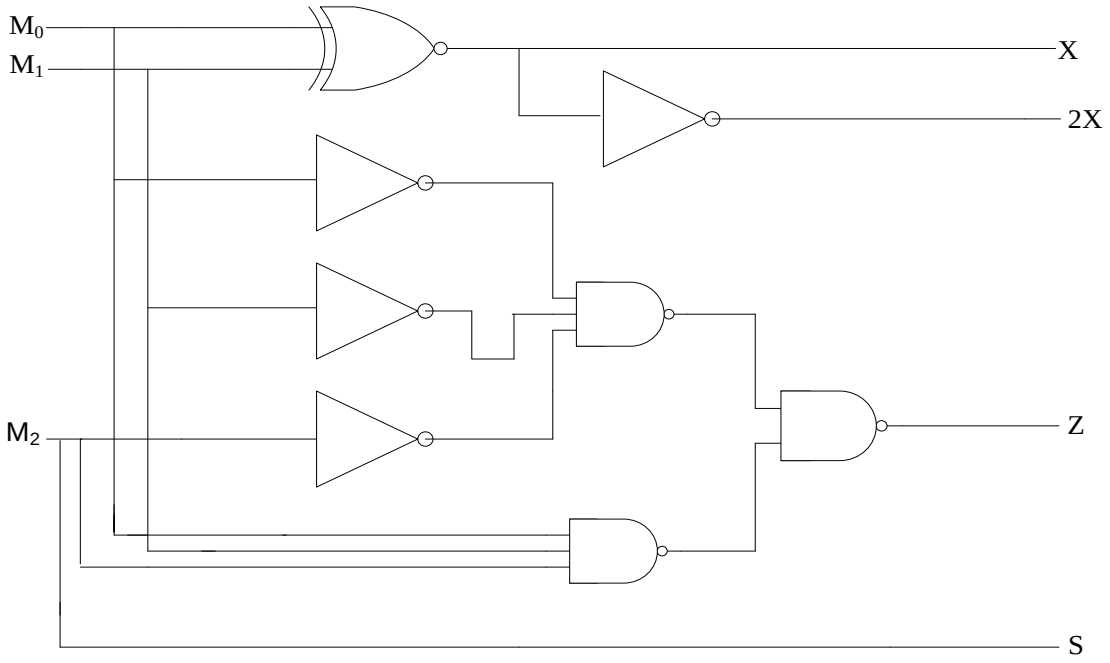

Figure 2.3: New MBE based on Wen-Chang architecture [13]

In BS, the S signal is needed first from BE to generate the partial product. The BE generated the S signal depending only on M_2 signal. As a result the combination of BE and BS produce the partial product.

The combination of S and Z will drive the sign extension to produce the correct result of multiplication.

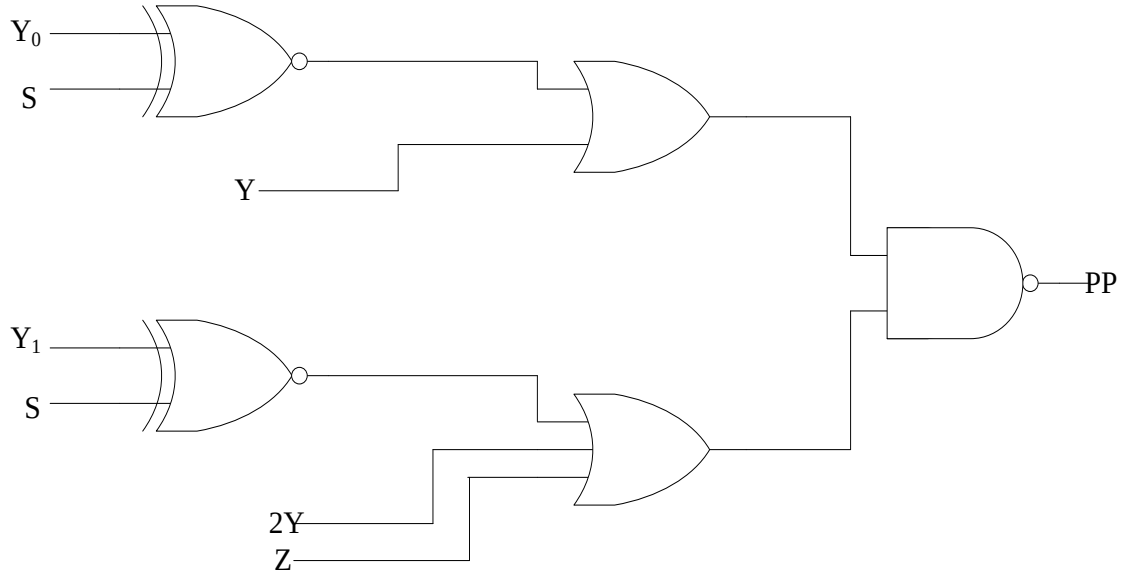

Figure 2.4: Booth Selector based on Wen-Chang architecture [13]

Table 2.5 shows the truth table for BE. M_0, M_1 andM_2 are inputs of BE and X, 2X, Z and S are the outputs.

Table 2.5

Truth Table for BE [13]

Inputs			**Outputs**			
M_2	M_1	M_0	X	2X	Z	S
0	0	0	0	0	1	0
0	0	1	1	0	0	0
0	1	0	1	0	0	0
0	1	1	0	1	0	0
1	0	0	0	1	0	1
1	0	1	1	0	0	1
1	1	0	1	0	0	1
1	1	1	0	0	1	1

CHAPTER 3

TREE MULTIPLIERS

3.1 Introduction

Trees are an extremely fast structure for summing partial-products. Tree structures require only order log N stages to reduce N partial products by performing parallel additions. The tree multiplication algorithm can reduce the number of partial products by employing multiple input compressors capable of accumulating several partial products concurrently [6]. Tree multiplier can handle the multiplication process for large operands. This is achieved by minimizing the number of partial product bits in a fast and efficient way by means of a CSA tree constructed from 1-bit full adders.

Consider a full adder with A and B as the adder inputs and C is the carry input. The full adder produces a bit of summand and a bit of carry out. It can be observed that a full adder is actually a “one's counter". A, B and C can all be seen as the inputs of 3:2 compressor. An adder as 1’s counter is shown in Table 3.1.

Table 3.1

An Adder as a 1’s Counter [9]

A	B	C	SUM	CARRY	NO OF ONES
0	0	0	0	0	0
0	0	1	1	0	1
0	1	0	1	0	1
0	1	1	0	1	2
1	0	0	1	0	1

1	0	1	0	1	2
1	1	0	0	1	2
1	1	1	1	1	3

The outputs carry and sum are the encoded output of the three inputs in binary notation. The tree multiplier is based on this property of the full adder. The addition of summands can be accelerated by adopting a 3:2 compressor. The advantage of the 3:2 compressors is that it can operate without carry propagation along its digital stages and hence is much faster than the conventional adder. In any scheme employing 3:2 compressors, the number of adder passes occurring in a multiplication before the product is reduced to the sum of two numbers, will be two less than the number of summands, since each pass through an adder converts three numbers to two, reducing the count of numbers by one. Figure 3.1 shows a 4-bit by 4-bit tree multiplier.

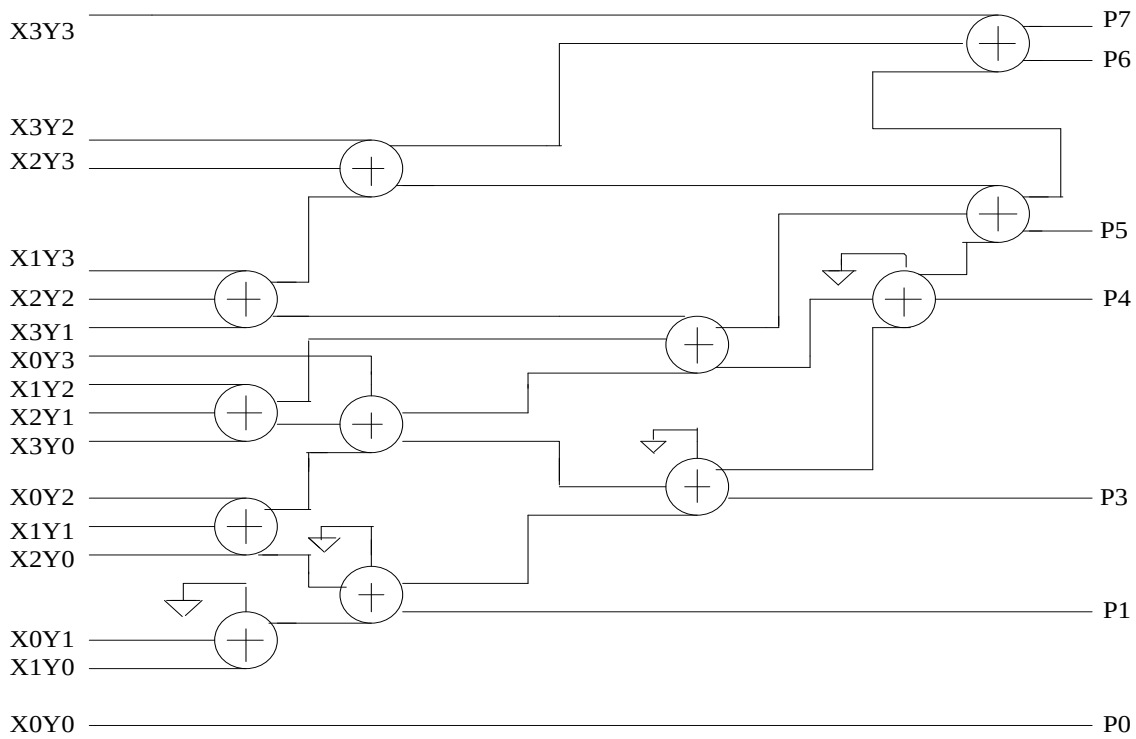

Figure 3.1: Tree multiplication 4 × 4 bit

The first step for a tree multiplier is to group the summands into threes, and introduce each group into its own 3:2 compressors, thus reducing the count of numbers by a factor of 1.5. The second step is to group the numbers resulting from the first step into threes

and again add each group in its own 3:2 compressors. By continuing such steps until only two numbers remain, the addition is completed in a time proportional to the logarithm of the number of summands. Other type of compressors can also be used in place of 3:2 compressors. The first tree structure was introduced by Wallace. Wallace showed that PPs can be reduced by connecting [3:2] compressors in parallel in a tree topology.

3.2 Wallace Tree Multiplier

A fast process for multiplication of two numbers was developed by Wallace. In 1964, C.S. Wallace [14] observed that it is possible to find a structure, which performs the addition operations in parallel, resulting in less delay. Wallace introduced a different way of parallel addition of the partial product bits using a tree of carry save adders, which is known as "Wallace Tree" [9]. A Wallace tree is an efficient hardware implementation of a digital circuit that multiplies two integers In order to perform the multiplication of two numbers with the Wallace method, partial product matrix is reduced to a two-row matrix by using a carry save adder and the remaining two rows are summed using a fast carry-propagate adder to form the product. This advantage becomes more pronounced for multipliers of bigger than 16 bits. WT increases speed because the addition of partial products is now. In WT architecture, all the bits of all of the partial products in each column are added together by a set of counters in parallel without propagating any carries. Another set of counters then reduces this new matrix and so on, until a two-row matrix is generated. Wallace method uses three-steps to process the multiplication operation.

- Formation of bit products.
- The bit product matrix is reduced to a 2-row matrix by using a carry-save adder.
- The remaining two rows are summed using a fast carry-propagate adder to produce the product.

3.2.1 Carry Save Adder

CSA performs the addition of m numbers in lesser duration compared to the simple addition. A carry save adder consist of full adders like ripple adders, but the carry output from each bit is taken out to form second result vector rather than being wired to the next most significant bit [15]. It takes three numbers (a, b, c) to add together and output two numbers, sum (s) and carry (c). It is carried out in one time unit duration. In

carry-save adder, the carry (c) is bought until the last step and the ordinary addition is carried out in the very last step. The most important application of a carry-save adder is to calculate the partial products in integer multiplication. This allows for architectures, where a tree of carry- save adders (also called *Wallace tree*) is used to calculate the partial products very fast. One 'normal' adder is then used to add the last set of carry bits to the last partial products to give the final multiplication result. Usually, a very fast carry-look ahead or carry-select adder is used for this last stage, in order to obtain the optimal performance. By using carry-save adder the need of carry propagation in the adder is avoided and latency of one addition is equal to gate delay of adder.

Let us illustrate this adder using an example. For simple addition of three numbers, we do as following:

Carry:	**1**	**1**	**0**	**1**	
I:	1	2	3	7	9
J:	3	3	9	0	1
K:	0	4	1	1	1
Sum:	**5**	**0**	**3**	**9**	**1**

The carry-save addition is carried out in three steps. In the first step, computes sum without considering carries as shown below:

I:	1	2	3	7	9
J:	3	3	9	0	1
K:	0	4	1	1	1
S:	**5**	**0**	**3**	**9**	**1**

In the second step, compute the carry on each column ignoring sum as shown below. The important thing to notice is that the carry calculated for one column is the carry obtained from the digits in previous column.

I:	1	2	3	7	9
J:	3	3	9	0	1
K:	0	4	1	1	1

C: **0** **1** **0** **1**

In the final step, add both sum and carry to obtain the final result. In the last step, the ordinary addition is carried out as shown below:

S: **4** **9** **3** **8** **1**

C: **0** **1** **0** **1**

SUM: **5** **0** **3** **9** **1**

3.2.2 Multiplication Operation in Wallace Tree Multiplier

A Wallace tree is an efficient hardware implementation of a digital circuit that multiplies two integers. The Wallace tree has three steps:

- Multiply (that is - AND) each bit of one of the arguments, by each bit of the other, yielding *n*2 results. Depending on position of the multiplied bits, the wires carry different weights.
- 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.

The second phase works as follows. As long as there are three or more wires with the same weight add a following layer:

- Take any three wires with the same weights and input them into a full adder. The result will be an output wire of the same weight and an output wire with a higher weight for each three input wires.
- If there are two wires of the same weight left, input them into a half adder.
- If there is just one wire left, connect it to the next layer.
- Wallace introduced a different way of parallel addition of the partial product bits using a tree of carry save adders, which is known as “Wallace Tree”. In order to perform the multiplication of two numbers with the Wallace method, partial product matrix is reduced to a two-row matrix by using a carry save adder and the remaining two rows are summed using a fast carry-propagate adder to form the product.

The conventional Wallace tree algorithm reduces the propagation by incorporating 3:2 compressors, however Wallace tree algorithm can also reduce the propagation using higher order compressor. The Figure 3.2 explains the various steps of Wallace tree multiplier.

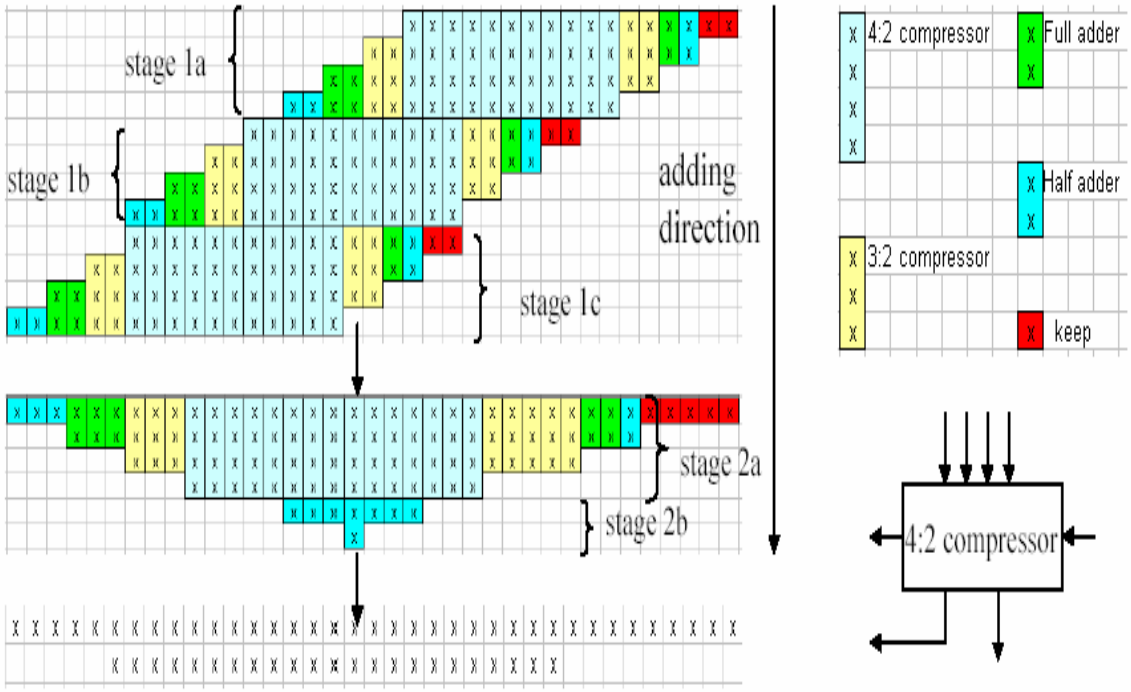

Figure 3.2: Wallace Tree Example [16]

In stage 1 the partial products are reduced using compressors. The partial terms marked as red are kept as such, the one marked with dark green indicates that they are compressed with full adder, the one marked with light yellow indicate 3:2 compressor and the one with blue box indicates 4:2 compressor.

3.2.3 Advantage of Wallace Multiplier

Propagation delay in this multiplier is($log_{\frac{3}{2}}(N)$), reduced in comparison to array multiplier.

3.2.4 Limitations of Wallace Multiplier

Wallace multiplier has limitation of being very irregular, so efficient layout is not possible. Routing between the levels become complicated, longer wires have greater capacitance

3.3 Compressors

Compressor is the building block for partial product reduction. A compressor is a combinatorial device that compresses N input lines to 2 output lines i.e. sum and carry.

3.3.1 [3:2] Compressor

A [3:2] compressor [17] is basically a Full adder. Gate level diagram of [3:2] compressor is shown in Figure 3.3.

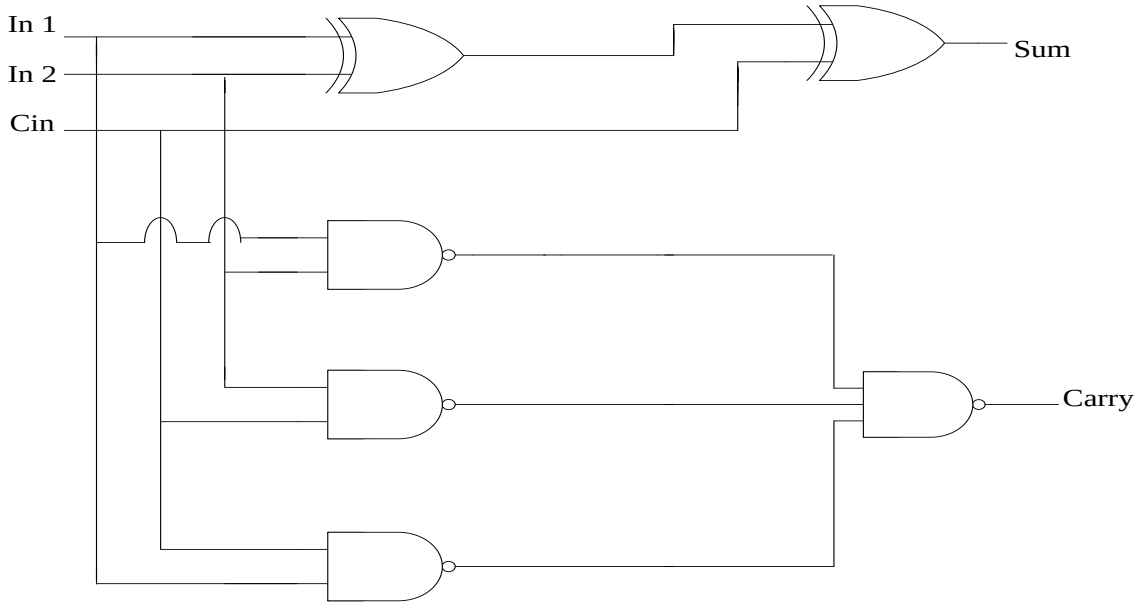

Figure 3.3: Gate level design of [3:2] compressor [17]

3.3.2 [4:2] Compressor

A [4:2] compressor [7] [17] takes four inputs of equal weight and produce two output. Truth table of [4:2] compressor is shown in Table 3.2.

Table 3.2

Truth table of [4:2] compressor [7]

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					
1	0	0	1					
1	0	1	0					
1	1	0	0					
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

It can be constructed from two 3:2 compressors. A [4:2] compressor has 4 input lines In1, In2, In3 and In4 that must be summed and has two output lines sum and carry, which are so called results of compression. The additional lines are input and output carries. In actual the structure compresses five partial product bits (four input bits and one carry-in bit) into three. However it acts as a compressor reducing the four bits into two, since carry-in and carry-out bits connect the adjacent 4:2 compressor. Thus, the number of partial product bits is reduced by half in one stage, making the efficiency higher. Block

diagram of [4:2] compressor is shown in Figure 3.4.The gate level design of a [4:2] compressor is shown in Figure 3.5.

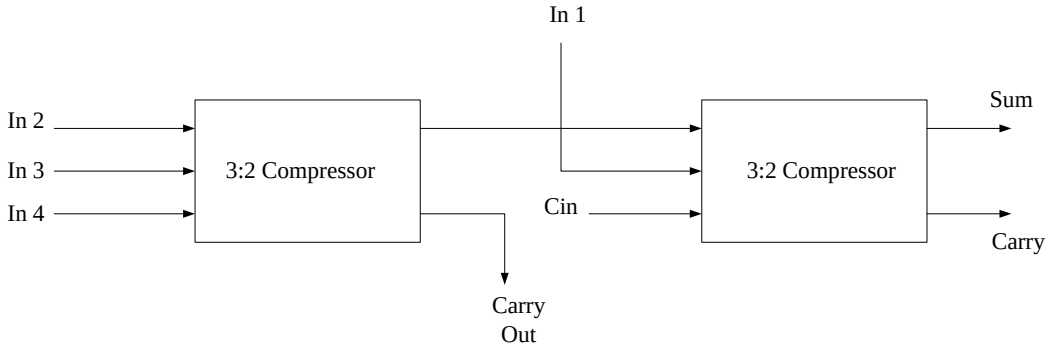

Figure 3.4: [4:2] compressor using [3:2] compressor [17]

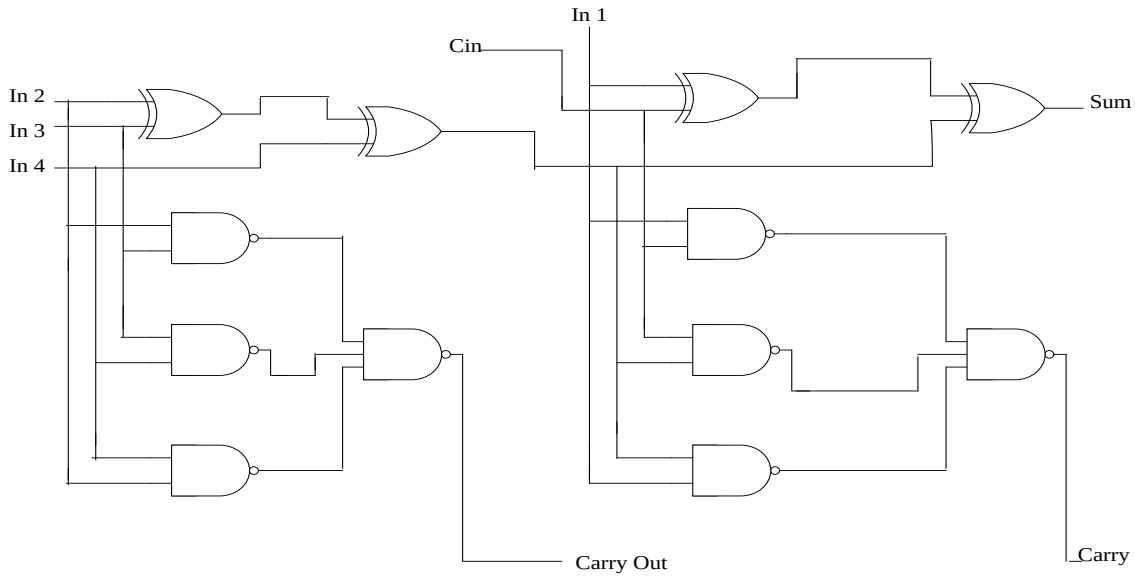

Figure 3.5: Design of [4:2] compressor [17]

This design of [4:2] compressor has three XOR gate delays regardless of the path. The use of [4:2] compressor permits the reduction of vertical path while horizontal path i.e. path involving the carry propagation is not changed.

3.3.3 Higher order Compressor

Different compressors logic based upon the concept of the counter of full adder. It can be defined as single bit adder circuit that has four/five/six/seven inputs and three outputs. As long as there are two outputs the circuit is able to count up to three. It means that only three inputs can be accommodated in the circuit. But now if one more output is

incorporated in the circuit then it is possible to count up to seven (111 i.e. decimal 7). A [9:2] compressor can be designed using three [3:2] and one [6:2] compressors and is shown in Figure 3.6.

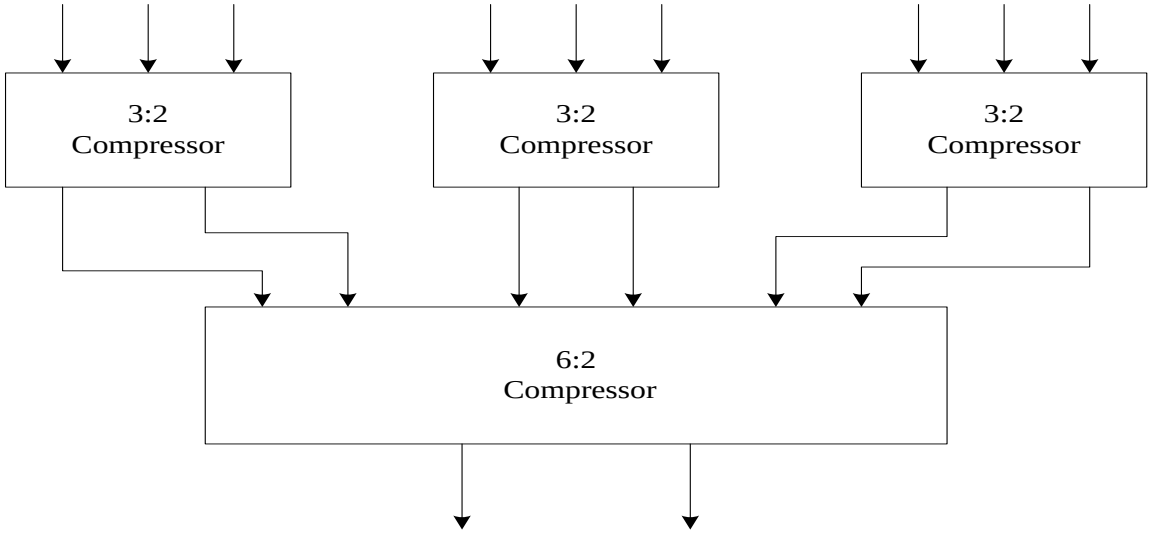

Figure 3.6: 9:2 compressor [18]

3.4 Hitachi's Multiplier

This multiplier is basically a Wallace Booth multiplier. The architecture consists of block performing Wallace tree and Booth's algorithm as well as carry look-ahead adder. The Modified Booth algorithm reduces the number of partial products by half. Modified Booth algorithm is used in order to save chip area rather than reducing delay time and by using Wallace tree reduction operation time is reduced. Wallace multiplier increases speed because delay dependence now becomes logarithmic. By using Wallace power consumption also reduces because power consumption of Wallace tree is small in comparison to other multipliers. This multiplier is purposed by Ohkubo and Suzuki. Block diagram of Hitachi's multiplier is shown in Figure 3.7.

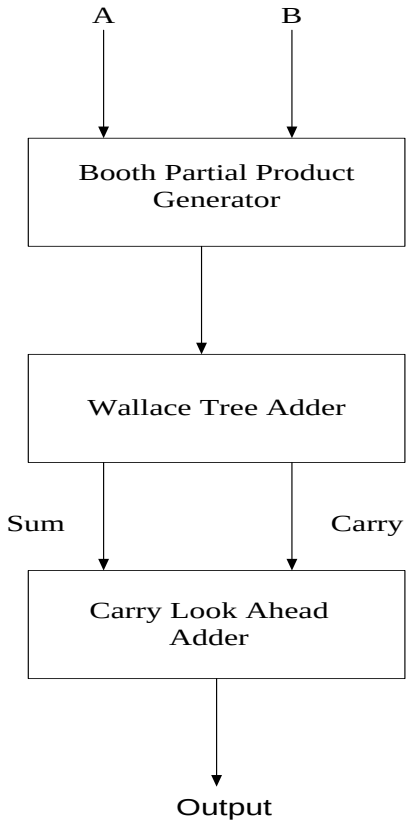

Figure 3.7: Block diagram of Hitachi multiplier

3.5 Comparison of Multipliers

Generally, it is not possible to say that an exact multiplier yields to greater cost-effectiveness, since trade-off is design and technology dependent. These basic array multipliers like Braun multiplier consume low power and exhibit good performance, however there use is limited to sixteen bits. For operands of sixteen bits and higher, the modified Booth algorithm reduces the partial product number by half. Thus speed of multiplier is increased. Due to circuitry overhead in Booth multiplier its power dissipation is comparable to Braun multiplier. Wallace's strategy for carry save adder trees is to combine the partial product bits as early as possible. This method yields to simpler CSA tree and a wider carry propagate adder and the designs using the Wallace tree method are fast. However a logarithmic depth reduction tree based CSA's has an irregular structure that makes the design and layout difficult. Moreover connections and signal path of varying length may lead to signal skew that have implications for both power and performance. Comparison of various multipliers is shown in Table 3.2.

Table 3.3

Comparison between Multipliers [19]

Multiplier Type	**Speed**	**Circuit Complexity**	**Layout**	**Area**
Array	Low/Medium	Simple	Regular	Smallest
Booth	High	Complex	Irregular	Medium
Wallace	Higher	Medium	More irregular	Large
Booth Wallace	Highest	More complex	Medium regularity	Largest

CHAPTER 4

DESIGN OF MULTIPLIER

Multiplier architecture comprise of two architectures, i.e., Modified Booth and Wallace tree. Based on the study of various multiplier architectures, we find that Modified Booth increases the speed because it reduces partial products to half. Further, the delay in multiplier can be reduced by using Wallace tree. Power consumption of Wallace tree multiplier is also less as compared to booth and array. Features of both multipliers can be combined to produce high speed and low power multiplier.

Modified Booth multiplier consists of Modified Booth Recoder (MBR). MBR have two parts, i.e., Booth Encoder (BE) and Booth Selector (BS). The basic operation of BE is to decode the multiplier signal and output will be used by BS to generate the partial product. The partial products are then, added with the Wallace tree adders, similar to the carry save adder approach. The last row of carry and sum output is added together by carry look-ahead adder with the carry skewed to the left by position.

4.1 Multiplier Architecture

Multiplier architecture consists of MBR, Wallace tree (4:2 compressor) and a carry look-ahead adder. The first block is the new modified booth algorithm and Booth is one such algorithm which is based on the fact that fewer partial products have to be generated for groups of consecutive zero's and one's. For consecutive groups of zero's and one's, there is no need to generate any new partial product. We only need to shift the previously accumulated partial product one bit position to the right for every zero in the multiplier. In this, three consecutive bits of the multiplier are examined to find the PP's. Partial products so formed are added using 4:2 compressors. An adder that uses 4:2 compressors has a more regular structure.

Wallace tree include half adders, full adders, 4:2 carry save adders in the same stage, reducing partial product at the same time. Block diagram of Booth Encoded Wallace Tree multiplier is shown in Figure 4.1.

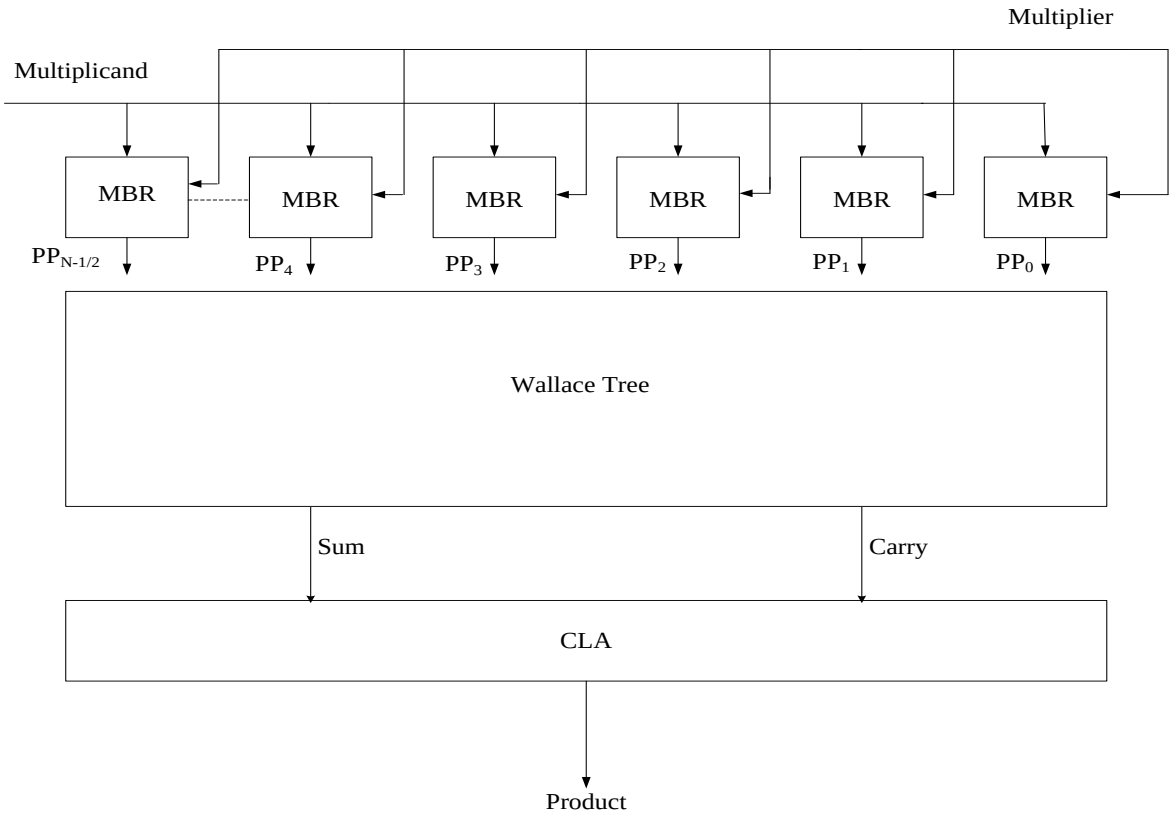

Figure 4.1: Block diagram of Booth Encoded Wallace Tree multiplier

The outputs of Wallace tree i.e. final sum and carry are added using carry look-ahead adder and adder give the final product.

4.1.1 Low Power Modified Booth Recoder

A low power MBR is proposed which calculates redundant computation and saves those computations thereby significantly reducing the power consumption. Low power MBR is controlled by a detection unit. The detection unit using an and gate has one of the two operands as its input to decide whether the Booth encoder calculates redundant computations. For example if Booth multiplier is multiplying two numbers “3A2C” and “004D” then the last four PPs are all zero and this part of computation can be neglected. Saving these computations can significantly reduce the power consumption caused by transient signals. Detection unit generates close signal which is when zero can freeze the inputs to reduce power consumption. In this technique, from the multiplicand, the partial product generator generate five candidates of partial products i.e. {-2X,-X, 0, X, 2X} which are then selected according to the Booth encoding result of multiplier. Meanwhile, detection unit has multiplier at its input to decide whether BE include redundant

computations. Low power MBR is shown in Figure 4.2.

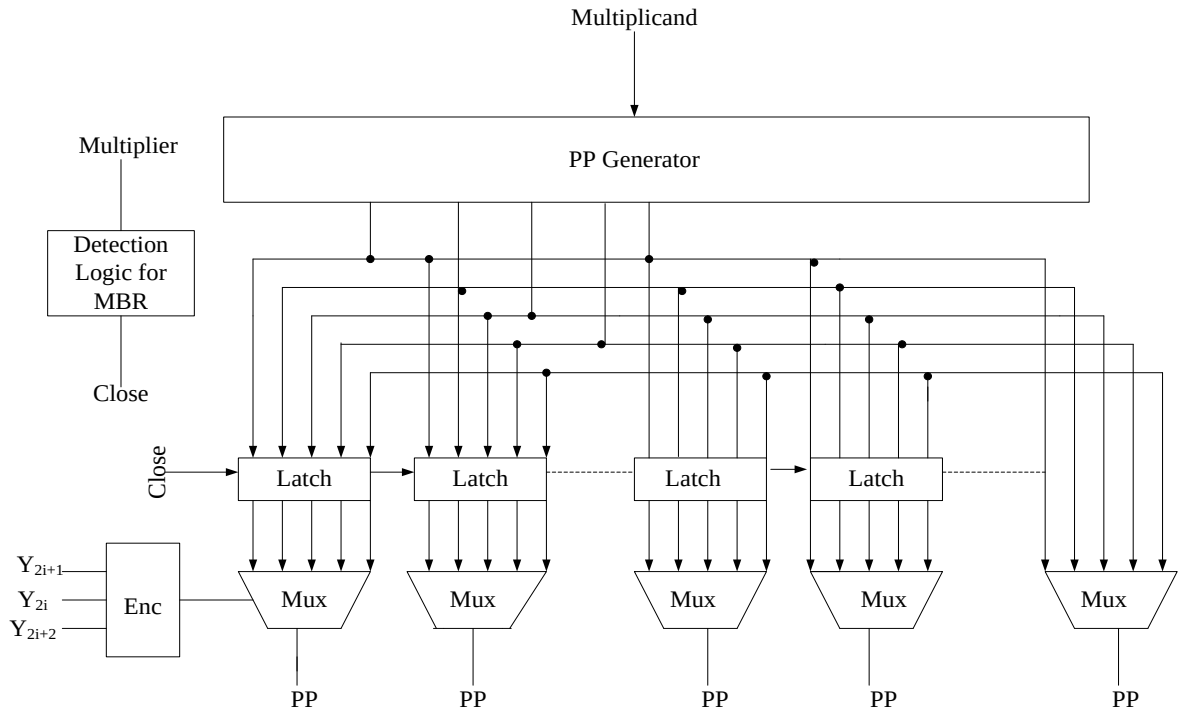

Figure 4.2: Low power Modified Booth Encoder

As shown in Figure 4.2 the latches can freeze the inputs of mux thereby decreasing power.

4.2 Implementation of 8 × 8 Bit Booth Wallace Multiplier

Basic building blocks of this multiplier are MBR which generates 4 partial products. Block diagram of 8 × 8 bit Booth Wallace multiplier is shown in Figure 4.3. Each MBR block is connected to X, 2X, 0, -2X, -X signals where X is multiplicand. Hence, each 8 MBR behaves at the same time. The multiplicand comes from the left to go into four MBR. Each MBR takes 3 bits from the multiplier with '0' appended at the right end. Each recoder output is shifted to its correct position and sign extended. PP are added using Wallace tree which includes 4 half adders (HA), 3 full adders (FA) and 5 4:2 compressor. Output of Wallace tree i.e. one row of sum bits and other row of carry bits are added using 16-bit carry CLA. Thus final product of 16 bit is obtained.

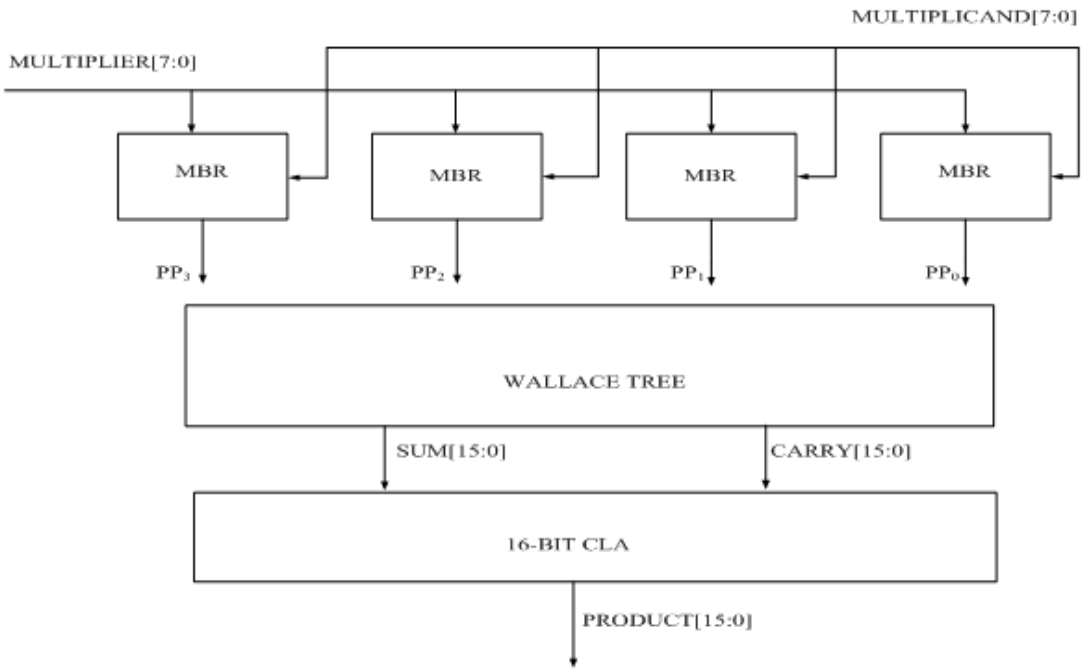

Figure 4.3: Block diagram of 8 × 8 bit Booth Wallace multiplier

16-bit CLA is shown in Figure 4.4. This adder requires two carry look-ahead circuit (CLC) levels and five carry look-ahead circuits (CLCs). Second level CLC uses the group P and outputs from the first level CLCs as inputs and provides the carry outputs.

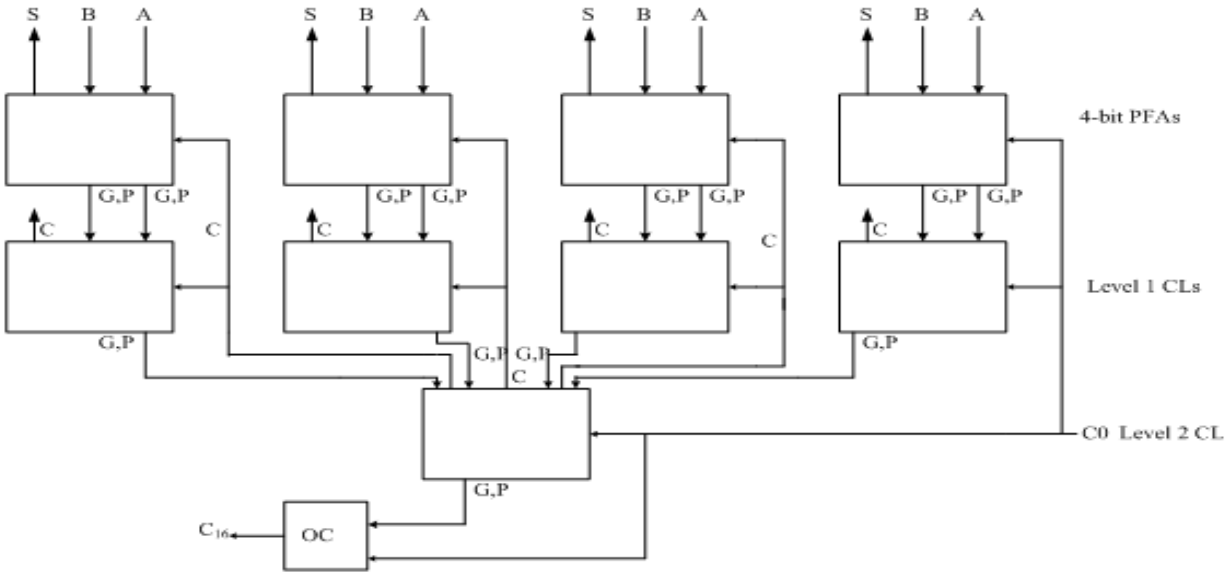

Figure 4.4: 16-bit CLA

Also, the P and G group outputs from the second level CLC cover carry generation and propagation for all 16 bits and by using an output carry (OC) circuit, can combine these two outputs with C_0 to produce output carry, i.e., C_{16}.

4.3 Implementation of 16×16 Bit Booth Wallace Multiplier

In 16×16 bit multiplier, eight MBR generate eight partial products of 17-bits. The multiplicand comes from the left to go into eight Modified Booth recoders. Each recoder takes 3 bits from the multiplier with a '0' appended at the right end. MBR has a 17-bit output. Each MBR output is shifted to its correct position and sign extended. Block diagram of this multiplier is shown in Figure 4.5.

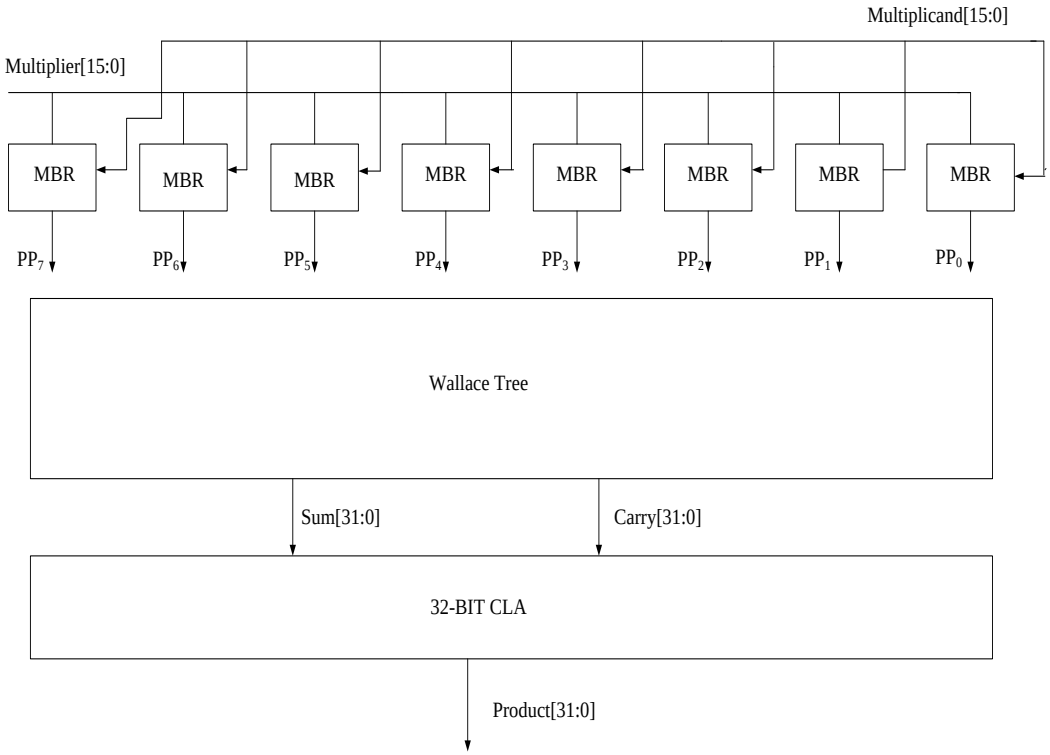

Figure 4.5: Block diagram of 8×8 bit Booth Wallace multiplier

Wallace tree includes three rows of 4:2 compressors. The first two rows of 4:2 compressors add partial products. The first row 4:2 compressors add partial products PP_0, PP_1, $PP_2$and PP_3. The second row of 4:2 compressors adds partial products PP_4, PP_5, PP_6 and PP_7. The third row of 4:2 compressors add the sum outputs from the first rows and the carry bits from 4:2 compressors in the right column. Figure 4.6 shows the 4:2 compressor organizations for adding eight partial products.

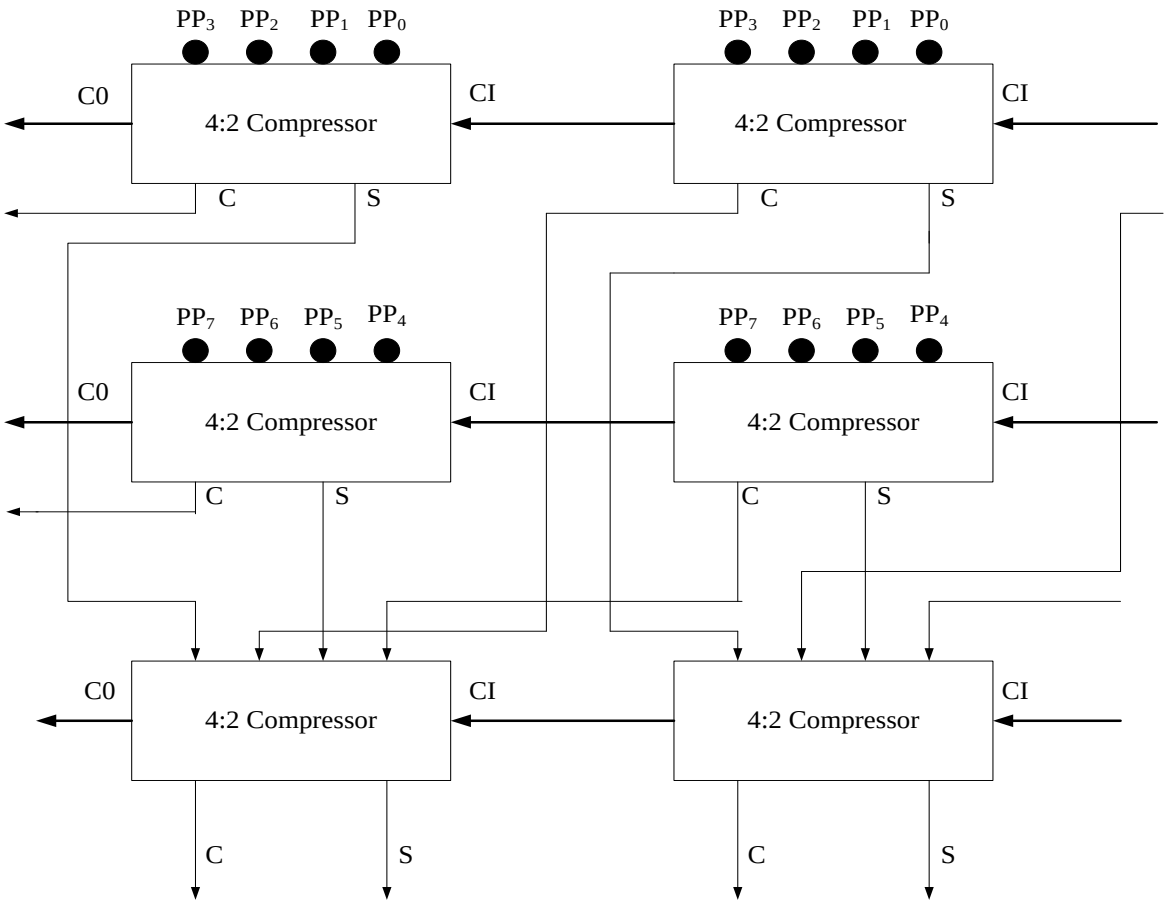

Figure 4.6: 4:2 Compressor organization for eight partial products

The carry and sum outputs of last row of 4:2 compressors are added with 32-bit carry look-ahead adder with the carry output bits shifted left one bit position to add with the sum bits.

Figure 4.7 shows 32-bit CLA. This adder requires three CLC levels and eleven carry look-ahead circuits. This adder was designed by adding a single third-level CLC and one OC circuit to two 16-bit CLAs minus their respective OC circuits. Second level CLC uses the group P and G outputs from the first level CLCs as inputs and provides the carry outputs C_4, C_8, C_{12}, C_{20}, C_{24} and $C_{28.}$ Third level CLC uses the group P and G outputs from the second level CLCs as inputs and provides the carry output C_{16}. Also, the *P* and *G* group outputs from the third-level CLC circuit cover carry generation and propagation for all 32 bits and, by using an OC circuit, can combine these two outputs with C_0 to produce carry output C_{32}. Thus final 32-bit product is obtained.

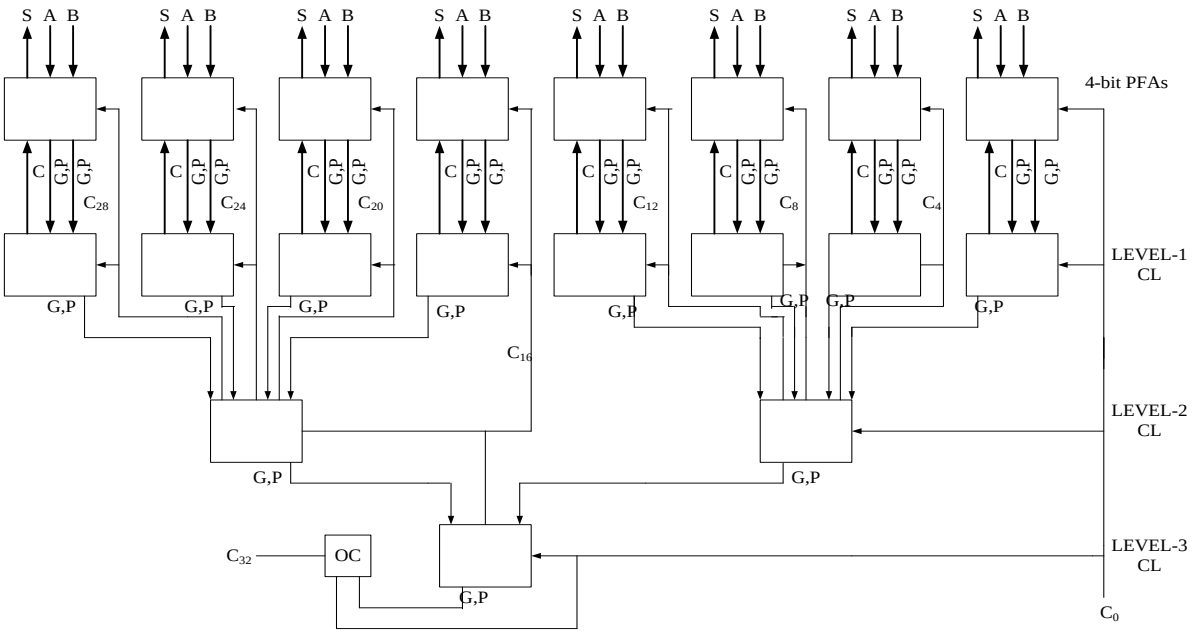

Figure 4.7: 32-bit CLA

4.4 Implementation of 32 × 32 Bit Booth Wallace Multiplier

Block diagram of 32 × 32 bit Booth Wallace multiplier is shown in Figure 4.8.

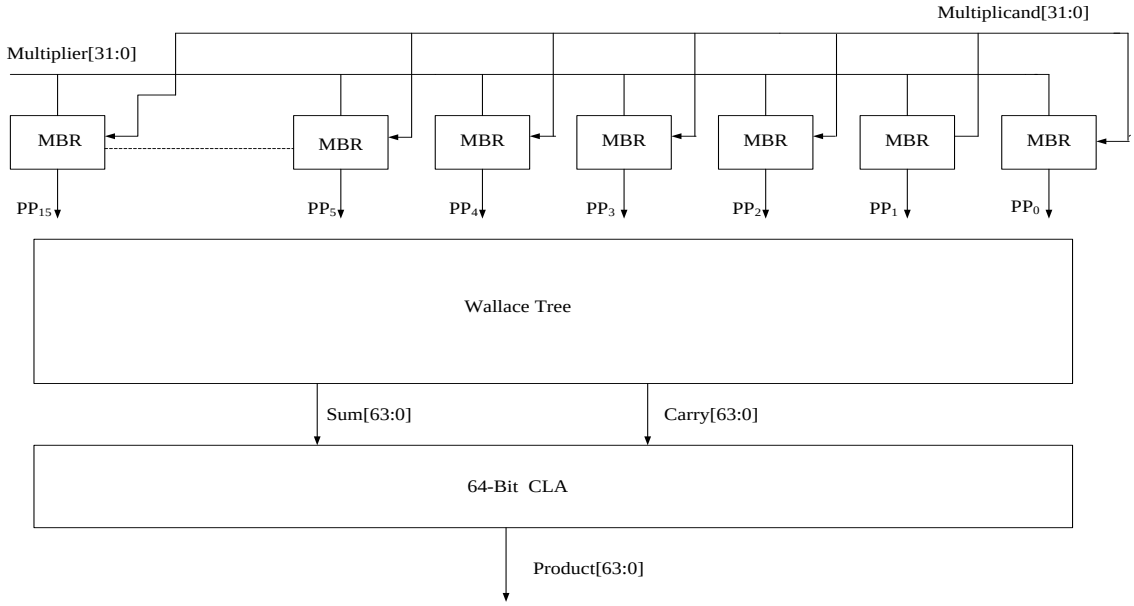

Figure 4.8: Block diagram of 32 × 32 bit Booth Wallace multiplier

In 32 × 32 bit multiplier, sixteen MBR generate sixteen partial product of 33-bit. Sixteen PP are reduced using seven rows of 4:2 compressors.

Figure 4.9 shows the 4:2 compressor organizations for adding sixteen partial products.

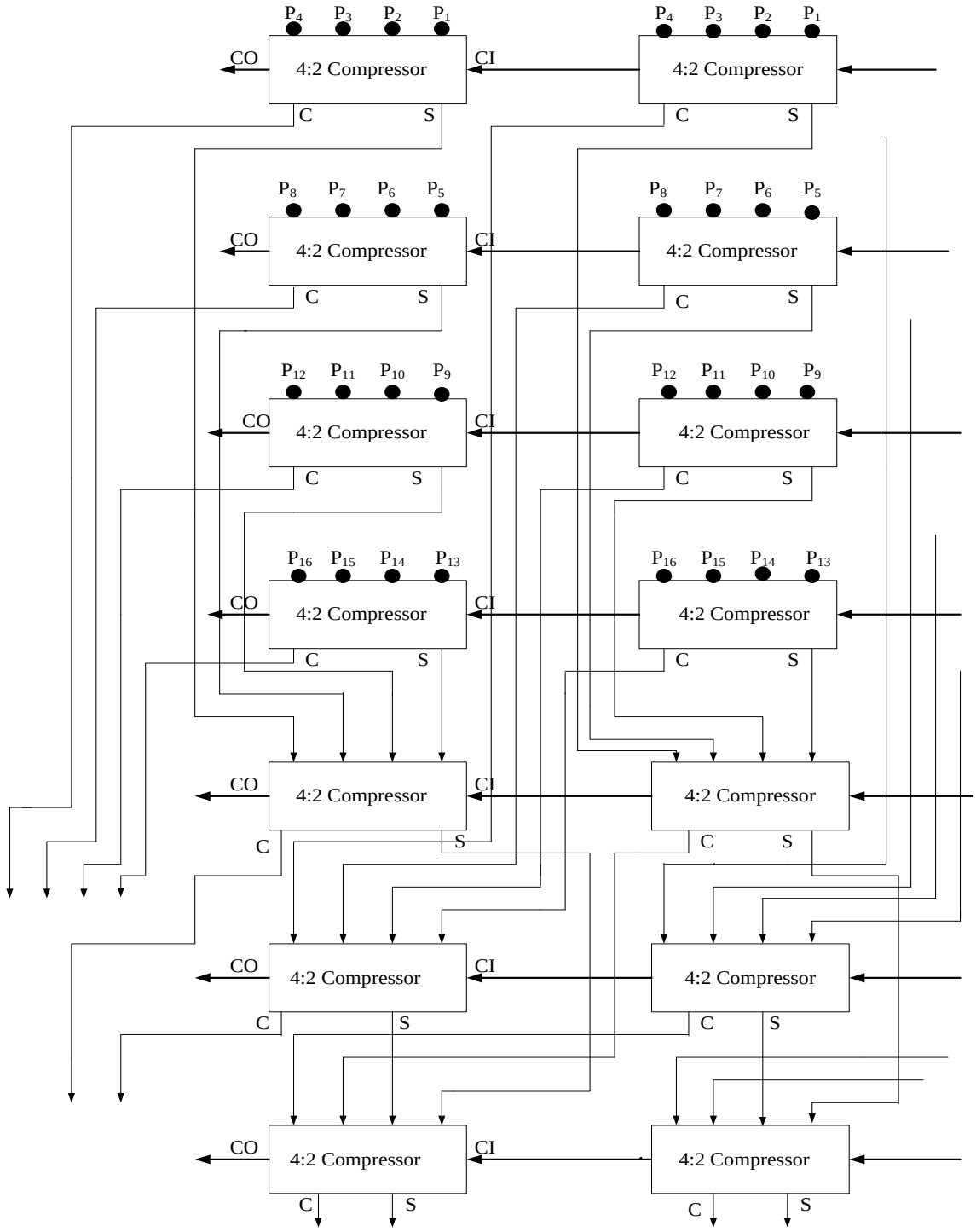

Figure 4.9: 4:2 Compressor organization for eight partial products

The carry and sum outputs of last row of 4:2 compressors are added with 64-bit carry look-ahead adder with the carry output bits shifted left one bit position to add with the sum bits. Figure 4.10 shows organization of 64-bit CLA.

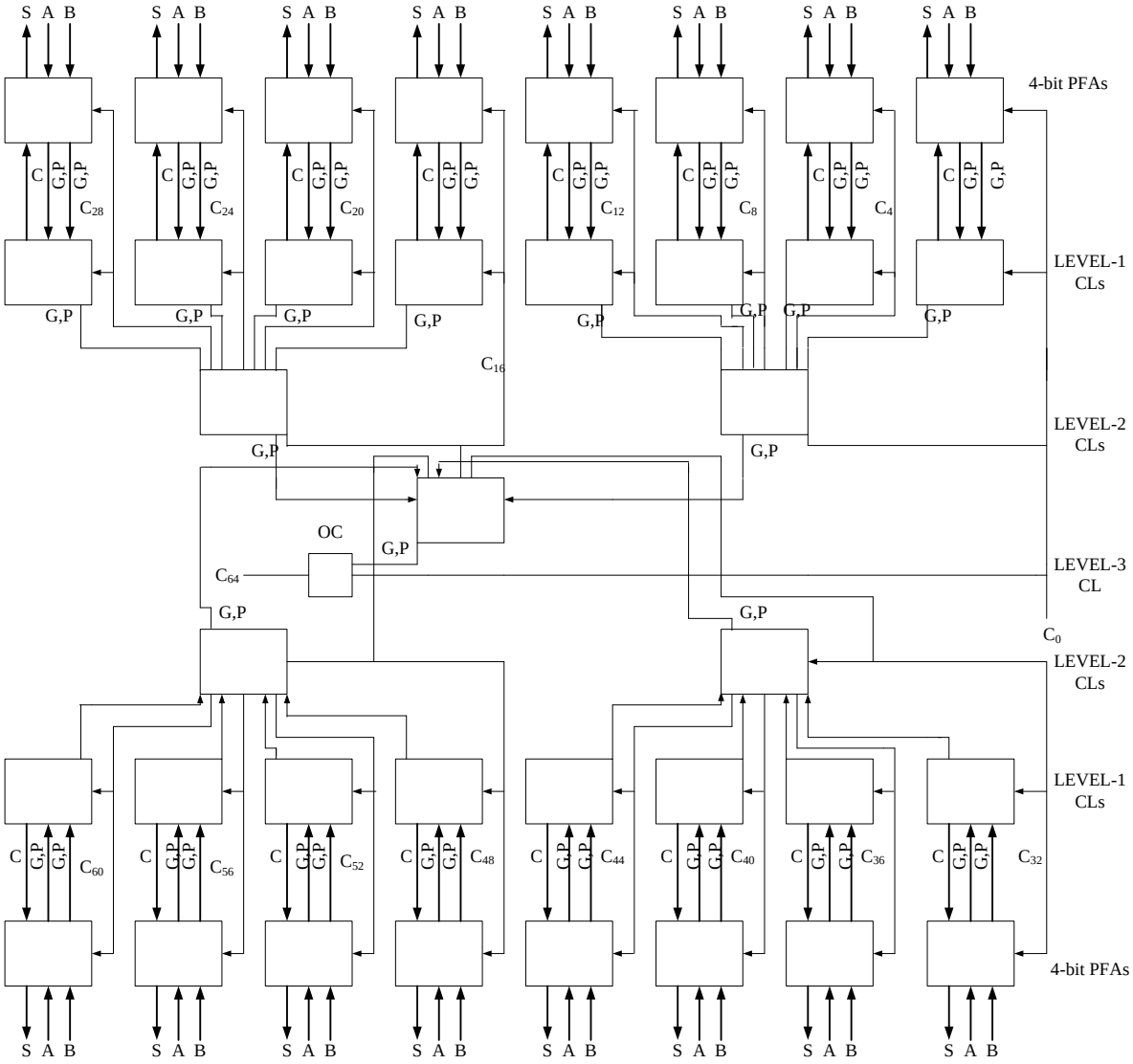

Figure 4.10: 64-bit CLA

This adder was designed by adding a single third-level CLC and one OC circuit to four 16-bit CLAs minus their respective OC circuits. Third-level CLC uses the group *P* and *G* outputs from the four second-level CLCs as inputs and provides the carry outputs C_{16}, C_{32}, and C_{48}. Also, the *P* and *G* group outputs from the third-level CLC circuit cover carry generation and propagation for all 64 bits and, by using an OC circuit, can combine these two outputs with C_0 to produce carry output C_{64}.

4.5 Pipelining of 32 × 32 Bit Multiplier

The pipeline technique is widely used to improve the performance of digital circuits. A simple calculation reveals that close to 33 gate-delays are needed to complete a multiplication of 32-bit operands without pipelining. From this number nine gate delays

belong to the Wallace structure (4:2 compressor), ten gate delays to the MBR and fourteen gate delays belong to the 64-bit carry look-ahead adder. Now, with addition of three registers, each carrying one-gate delays we come up with overall 36 gate delays. Hence, if we split this delay into three, then each stage of the pipeline takes about twelve gate delays to complete.

The next step is to locate the positions of the registers within the data path. The best location for the first register, R1, is after MBR. This is because, with ten gate delays for the MBR, and one gate delay for the register R1 we get the limit of eleven gate delays. Similarly, the best location for the second register, R2, is before first level of carry look-ahead adder. This stage takes nine gate delays of the Wallace structure, two gate delay of carry look-ahead adder and one gate delay for the register R2 we get the limit of twelve gate delays. Third register is placed at the end of carry look-ahead adder.

CHAPTER 5

FPGA IMPLEMENTATION

5.1 Introduction

FPGA stands for field programmable gate arrays that can be configured by the customer or designer after manufacturing. Field programmable gate arrays are called this because rather than having a structure similar to a PAL or other programmable device, they are structured very much like a gate array ASIC. An FPGA is a device that consists of thousands or even millions of transistors connected to perform logic functions. They perform functions from simple addition and subtraction to complex digital filtering and error detection and correction. Aircraft, automobiles, radar, missiles and computers are just some of the system that uses FPGA. FPGA are truly revolutionary devices that blend the benefits of both hardware and software. They can implement circuits just like hardware, providing huge power, area, and performance benefits over software, yet can be reprogrammed easily to implement wide range of tasks.

Xilinx, Altera and Quick logic are just few companies that manufacture FPGAs. FPGA architecture consists of three basic capabilities: input/output (I/O) interfaces, basic building blocks, and interconnections. Figure 5.1 shows FPGA architecture. I/O interfaces are the mediums in which data are sent from internal logic to external sources and from which data are received from external sources. The interface signals can be unidirectional or bidirectional, single-ended or differential. Some I/O standards are:

- GTL (gunning transceiver logic).
- HSTL (high-speed transceiver logic).
- LVCMOS (low-voltage CMOS).
- LVTTL (low-voltage transistor-transistor logic).
- LVDS (low-voltage differential signalling).

The basic logic building blocks are preconfigured logic or resources used to build design. Xilinx's basic building blocks are called configurable logic blocks (CLBs). Each CLB contains slices, and each slice has lookup tables (LUTs).

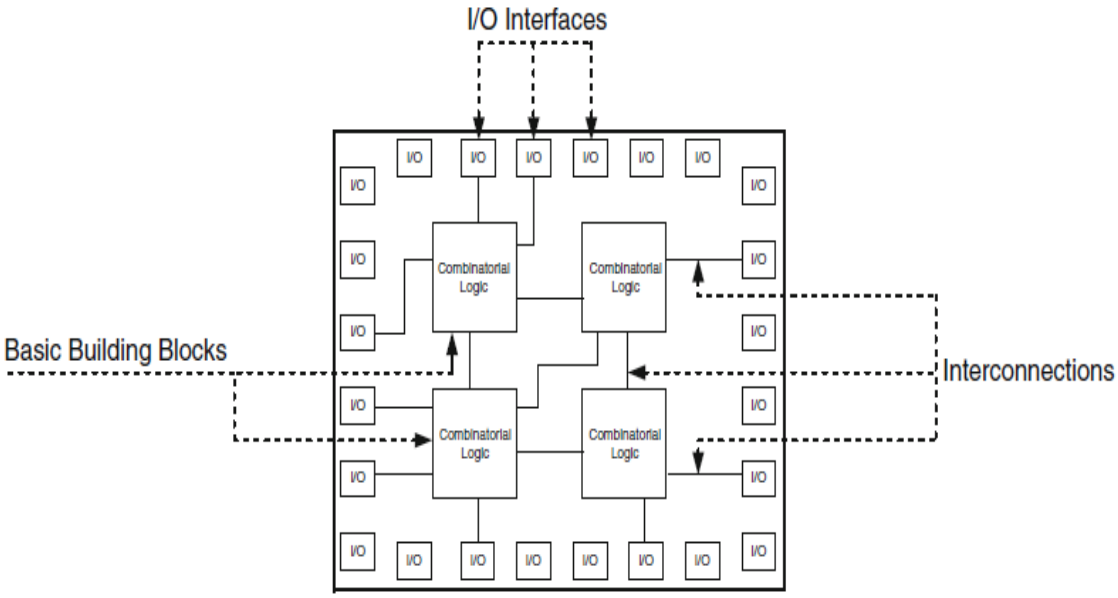

Figure 5.1: FPGA Architecture

Interconnection involves connecting the basic building blocks to perform design-specific functions as well as connecting the internal logic to I/O interface..

5.2 FPGA Implementation

The FPGA that is used for the implementation of the circuit is the Xilinx Spartan 3E (Family), XC3S500 / XC3S1600 (Device), FG320 / FG484 (Package), -5 (Speed Grade). The working environment/tool for the design is Xilinx ISE 9.2i. Spartan 3E family member's data summary is shown in Table 5.1.

Table 5.1: Spartan-3E Data Summary

Device	System Gates	Total CLBs	Total Slices	Max User I/O	Max Diff I/O pairs
XC3S100E	100K	240	960	108	40
XC3S250E	250K	612	2448	172	68
XC3S500E	500K	1164	4656	232	92
XC3S1200E	1200K	2168	8672	304	124
XC3S1600E	1600K	3688	14,752	376	156

5.3 FPGA Design Flow

As the FPGA architecture evolves and its complexity increases, CAD software has become more mature as well. Today, most FPGA vendors provide a fairly complete set of design tools that allows automatic synthesis and compilation from design specifications in hardware specification languages, such as Verilog or VHDL, all the way down to a bit stream to program FPGA chips.

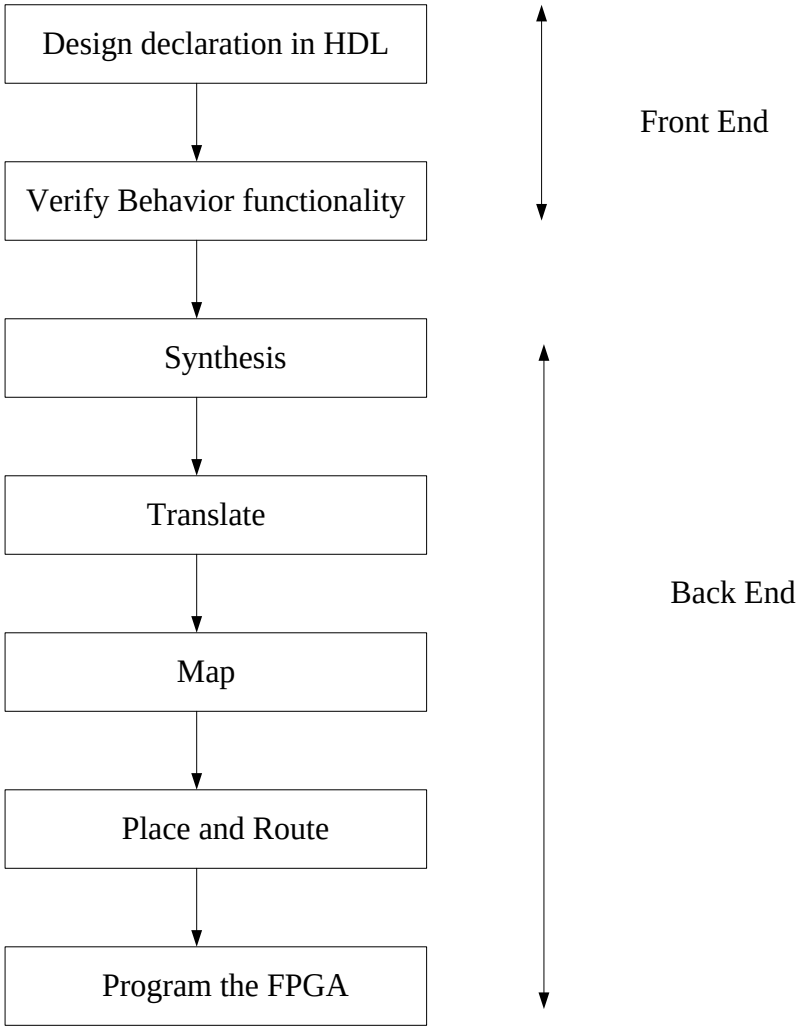

Figure 5.2: FPGA Design Flow

A design flow includes the components shown in Figure 5.2. Design constraints typically include the expected operating frequencies of different clocks, the delay bounds of the signal paths, delays from input pads to output pads (I/O) delay.

Other constraints are from input pads to registers (setup time), and from registers to output pads (clock-to-output delay). In some cases, delays between some specific pairs of

registers may be constrained. The tool then generates a netlist file (NGC file) and then optimizes it.

5.3.1 Design Implementation:

The Design implementation netlist consist of following sub-processes:

- **Translation:** The Translate process merges all the input netlists and design constraints information's and outputs a Xilinx NGD (Native Generic Database) file.
- **Mapping:** The Map process is run after the Translate process is complete. Mapping maps the logical design described in the NGD file to the components/primitives (slices/CLBs) present on the target device. The Map process creates an NCD file.
- **Place and Route:** The place and route (PAR) process is run after the design has been mapped. PAR uses the NCD file created by the Map process to place and route the design on the target FPGA design.
- **Bitstream Generation:** The collection of binary data used to program reconfigurable logic device is most commonly referred to as a "bit stream", although this is somewhat misleading because the data are no more bit oriented than that of an instruction set processor and there is generally no "streaming".
- **Functional Simulation:** It is performed on the netlist or code generated by synthesis tool. It is performed before mapping of the design. This simulation process allows the user to verify that design has been synthesized correctly.
- **Static timing analysis:** Three types of static timing analysis that can be performed are:
  - Post-fit Static timing analysis: The Analyze Post-Fit Static timing process opens the timing Analyzer window, which interactively select timing paths in design for tracing the timing results.
  - Post-Map Static Timing Analysis: It analyze the timing results of the Map process. Post-Map timing reports can be very useful in evaluating timing performance(logic delay + route delay).
  - Post Place and Route Static Timing Analysis: Post-PAR timing reports incorporate all delays to provide a comprehensive timing summary. If a placed and routed design has met all timing constraints, then proceed by creating configuration data and downloading a device.

5.4 FPGA Programming

A programming file is generated by running the Generate Programming File process. This process can be run after the FPGA design has been completely routed. The Generate Programming File process runs BitGen, the Xilinx bitstream generation program, to produce a bitstream (.BIT) or (.ISC) file for Xilinx device configuration. The FPGA device is then configured with the .bit file using the JTAG boundary scan method. After the Spartan device is configured for the intended design, then its working is verified by applying different inputs.

5.4.1 LCD Interfacing in FPGA

The Spartan-3E FPGA starter kit board features a 2-line by 16-character Liquid Crystal Display (LCD). The Figure 5.3 represents the implementation of LCD Interfacing of 32×32 bit multiplier

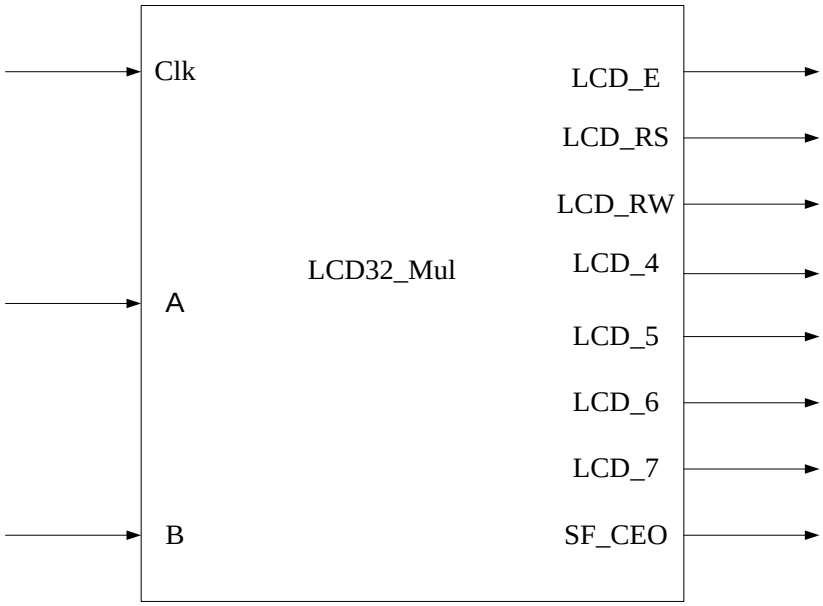

Figure 5.3: LCD interfacing of 32X32 multiplier

There are eleven ports of 32x32 bit multiplier namely:

- Clk: Input clk
- A: Multiplicand
- B: Multiplier
- LCD_E: Read/Write Enable Pulse, “0” for disabled, “1” for Read/Write operation enabled. FPGA pin number is M18.

- LCD_RS: Register Select, “0” for instruction register during write operations, busy flash during read operations, “1” for data for read or write operations. FPGA pin number is L18.
- LCD_RW: Read/Write Control, “0” for write- LCD accepts data, “1” for read- LCD presents data. FPGA pin number is L17.
- LCD_4: Data bit DB4, FPGA pin number is R15.
- LCD_5: Data bit DB5, FPGA pin number is R16.
- LCD_6: Data bit DB6, FPGA pin number is P17.
- LCD_7: Data bit DB7, FPGA pin number is M15
- SF_CE0: SF_CE0 is “1” then, FPGA application has full read/write access to the LCD.

CHAPTER 6

RESULTS AND CONCLUSION

6.1 Results

6.1.1 Simulation Results of 8×8 Bit Booth Wallace Multiplier

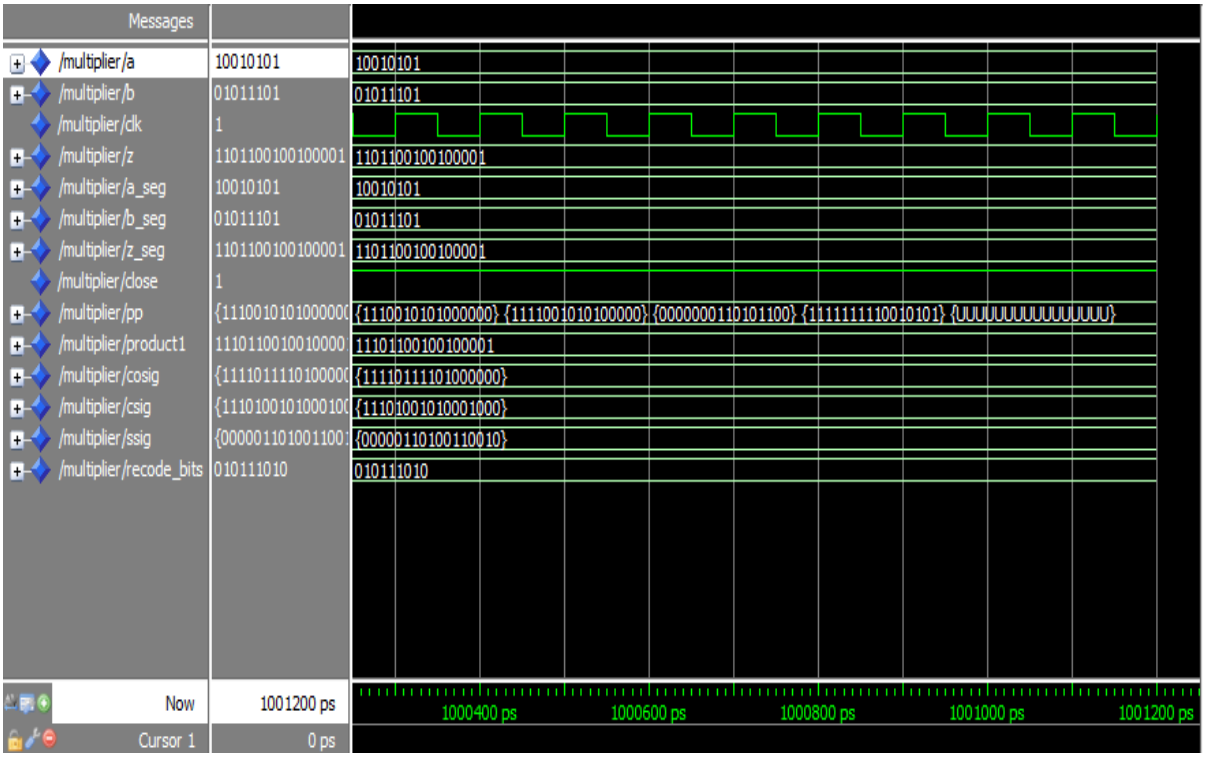

Figure 6.1: **Simulation result of 8×8 bit Booth Wallace multiplier by Model Sim**

Description:

a:		Input data 8-bit
b:		Input data 8-bit
c:		Input Clock
z:		Output data 16-bit
a	=	10010101
b	=	01011101
z	=	1101100100100001

Table 6.1 shows the synthesis results of 8×8 bit modified Booth Wallace multiplier

Table 6.1

Synthesis Results of 8×8 Bit Modified Booth Wallace Multiplier

FPGA Device Package	xc3s500efg320
Number of slices	71 out of 4656
Number of slice Flip Flops	43 out of 9312
Number of 4 input LUTs	151 out of 9312
Number of IOs	33
Number of bonded IOBs	33 out of 232
Number of GCLKs	1 out of 24
Minimum period	8.78ns
Maximum frequency	113.8MHz
Power(X-power)	84mW
Peak memory usage	126MB

6.1.2 Simulation Results of 16×16 Bit Booth Wallace Multiplier

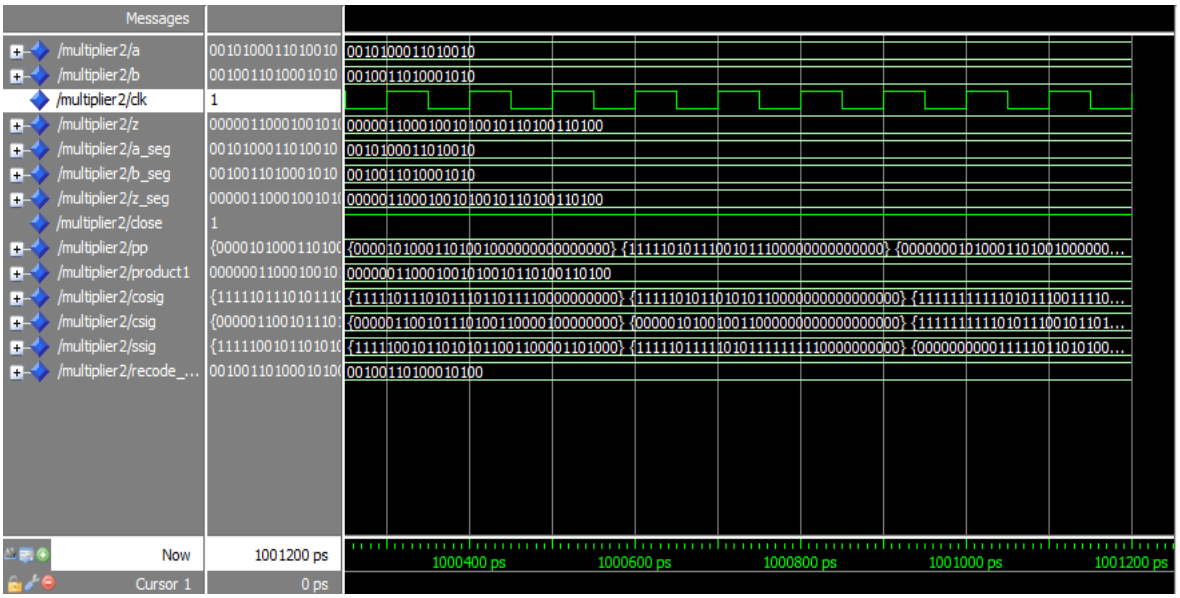

Figure 6.2: **Simulation result of 16×16 bit Booth Wallace multiplier by Model Sim**

Description:

a:		Input data 16-bit
b:		Input data 16-bit
clk:		Input clock
z:		Output data 32-bit
a	=	0010100011010010
b	=	0010011010001010
z	=	00000110001001010010110100110100

Table 6.2 shows the synthesis results of 16×16 bit modified Booth Wallace multiplier.

Table 6.2

Synthesis Results of 16×16 Bit Modified Booth Wallace Multiplier

FPGA Device Package	xc3s500efg320
Number of slices	449 out of 4656
Number of slice Flip Flops	81 out of 9312
Number of 4 input LUTs	756 out of 9312
Number of IOs	62
Number of bonded IOBs	62 out of 232
Number of GCLKs	1 out of 24
Minimum period	10.13ns
Maximum frequency	98.71MHz
Power(X-power)	95mW
Peak memory usage	133MB

6.1.3 Simulation Results of 32×32 Bit Booth Wallace Multiplier

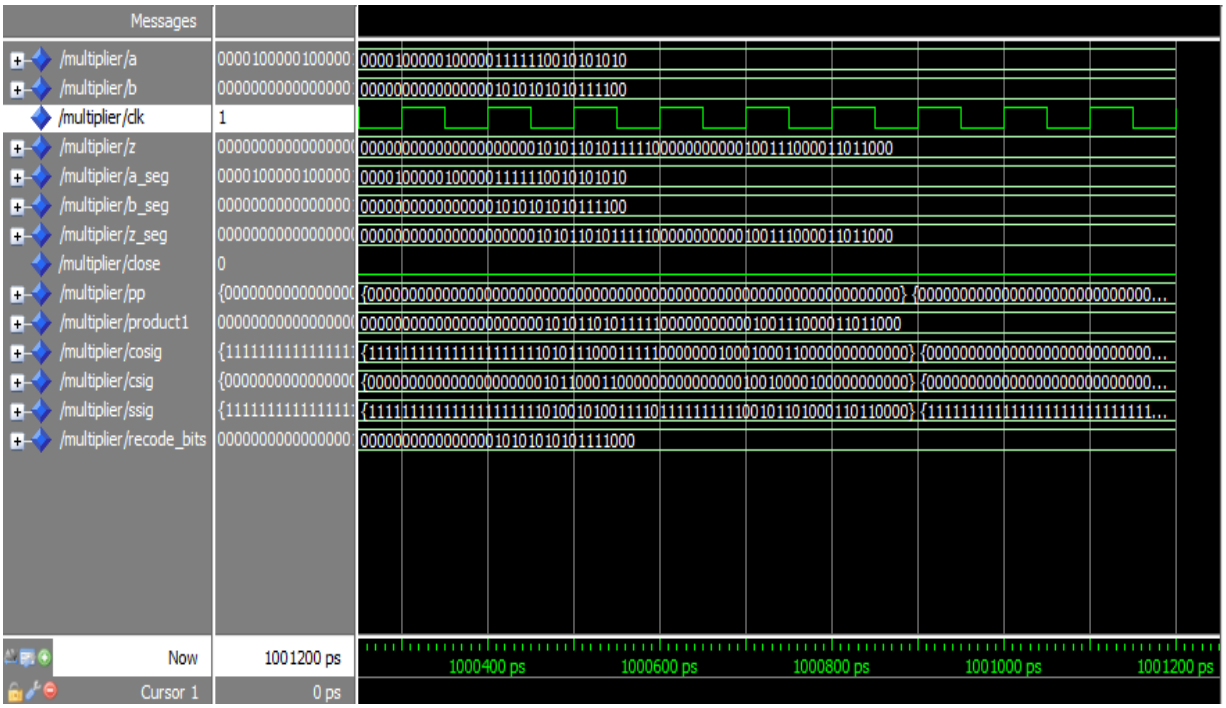

Figure 6.3: **Simulation result of 32 × 32bit Booth Wallace multiplier by Model Sim**

Description:

a:		Input data 32-bit
b:		Input data 32-bit
clk:		Input clock
z:		Output data 64-bit
a	=	01000001100010010010000100010010
b	=	00000000000000001010110011010101
z	=	00000000000000000010110000111110
		10101001010100001001101111111010

Table 6.3 shows the synthesis results of 32×32 bit Booth Wallace multiplier.

Table 6.3

Synthesis Results of 32×32 Bit Modified Booth Wallace Multiplier

FPGA Device Package	xc3s500efg320
Number of slices	2142 out of 4656
Number of slice Flip Flops	274 out of 9312
Number of 4 input LUTs	4167 out of 9312
Number of IOs	129
Number of bonded IOBs	129 out of 232
Number of GCLKs	1 out of 24
Minimum period	13.38ns
Maximum frequency	74.7MHz
Power(X-power)	102mW
Peak memory usage	163MB

Comparison of 32×32 bit 3 stage-pipelined and non-pipelined multiplier is shown in Table 6.4.

Table 6.4

Comparison of 32×32 bit pipelined and non-pipelined multiplier

	Non pipelined Multiplier	**Pipelined Multiplier**
Number of Slices	2120	2142
Number of 4 Input LUTs	3977	4176
Delay	23.38ns	13.38ns
Frequency	42.7 MHz	74.7MHz

6.1.4 Outputs of 32×32 Bit Display on LCD Screen of FPGA

As shown in Figure 6.4, when programming is complete, the program succeeded message is displayed. Table 6.5 shows synthesis results of LCD interfacing. Figure 6.5 shows output of 32×32-bit multiplier on LCD of Spartan kit.

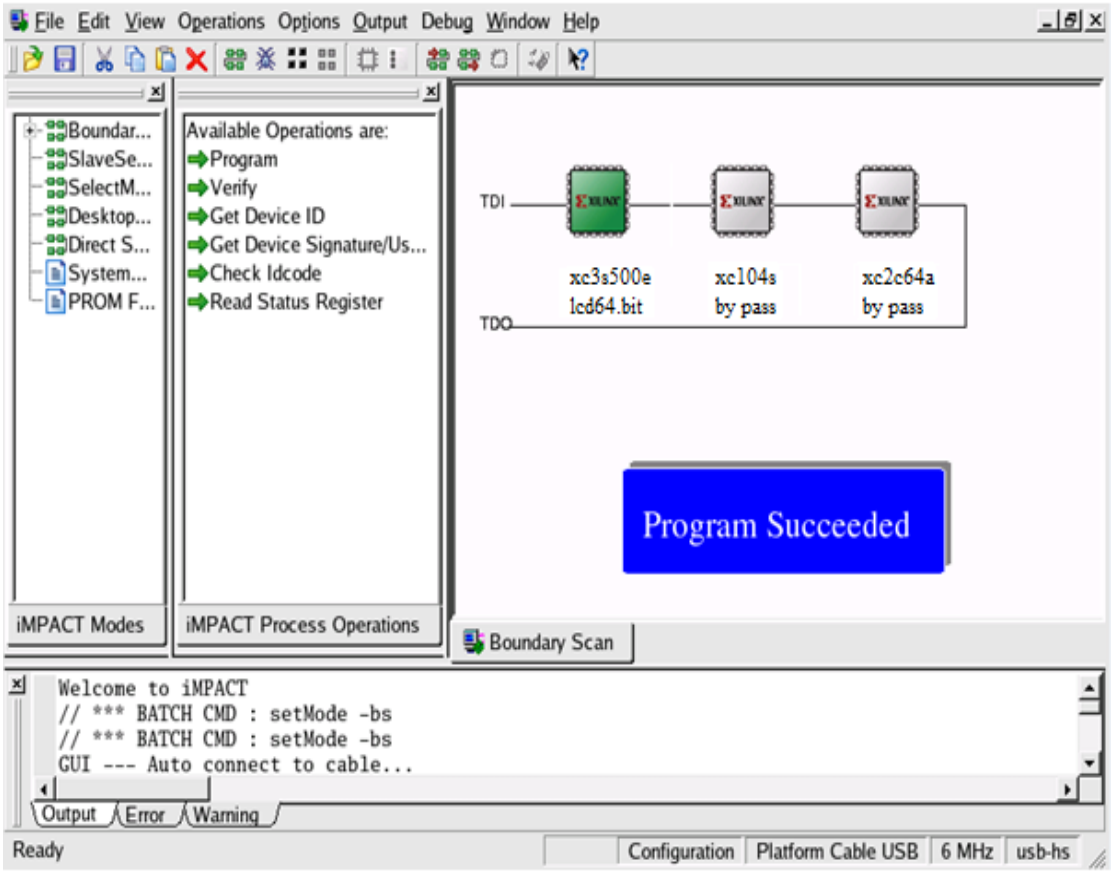

Figure 6.4: LCD display (output of 32×32) on Spartan 3E FPGA kit

Table 6.5

Synthesis Results of LCD Interfacing

FPGA Device Package	xc3s500efg320
Number of slices	120 out of 4656
Number of slice Flip Flops	86 out of 9312
Number of 4 input LUTs	226 out of 9312
Number of IOs	18
Number of bonded IOBs	18 out of 232
Number of GCLKs	1 out of 24
Minimum period	7.5ns
Maximum frequency	131.6MHz
Power(X-power)	117mW

Output on LCD Screen = 00002C3EA9509BFA (In hex)

Figure 6.5: Output of 32X32 bit multiplier shown on LCD of Spartan kit

6.1.5 Simulation Results of 32×32 Bit Array Multiplier

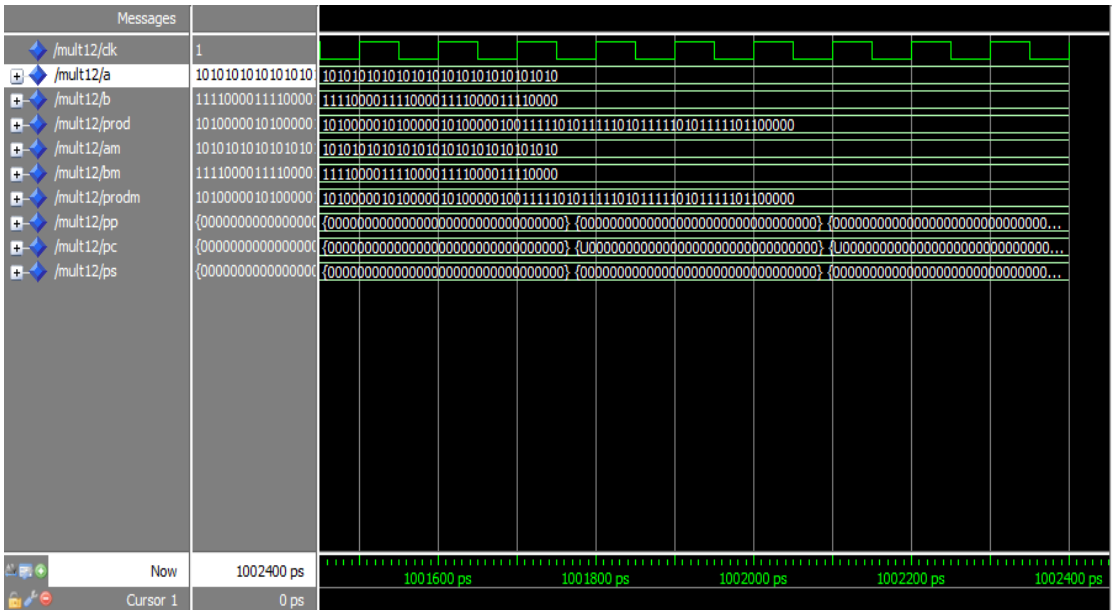

Figure 6.6: **Simulation result of 32×32 bit Array multiplier**

Description:

a:		Input data 32-bit
b:		Input data 32-bit
clk:		Input clock
prod:		Output data 64-bit
a	=	10101010101010101010101010101010
b	=	11110000111100001111000011110000
prod	=	01000001010000010100000010011111
		01011111010111110101111101100000

6.1.6 Simulation Results of 32×32 Bit Wallace Multiplier

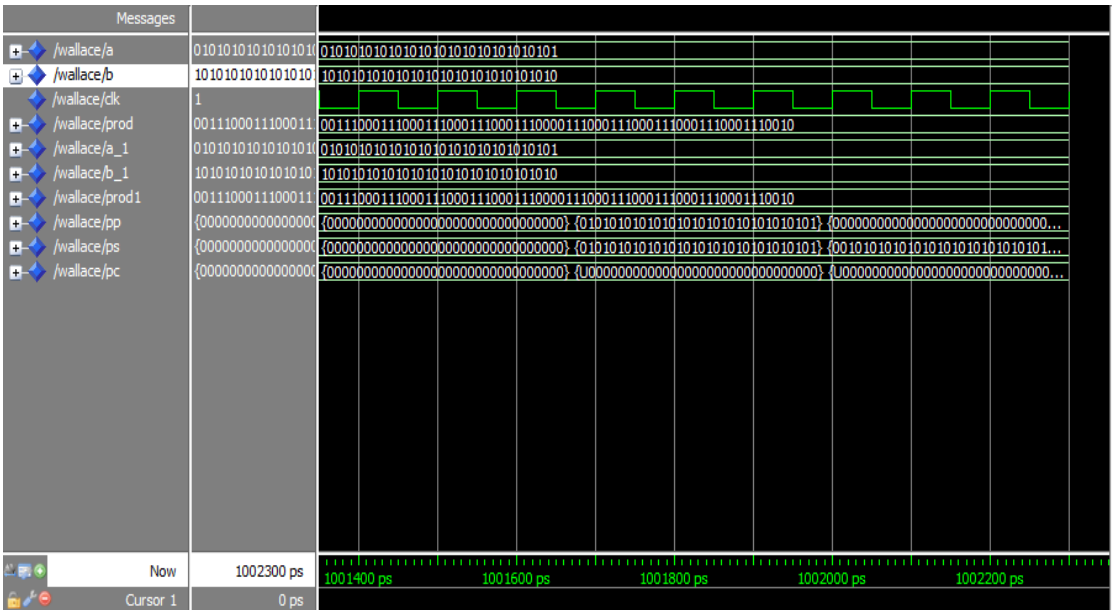

Figure 6.7: **Simulation result of 32×32 bit Wallace multiplier**

Description:

a:		Input data 32-bit
b:		Input data 32-bit
clk:		Input clock
z:		Output data 64-bit
a	=	01010101010101010101010101010101
b	=	10101010101010101010101010101010
z	=	00111000111000111000111000111000
		01110001110001110001110001110010

6.1.7 Simulation Results of 32×32 Bit Booth Multiplier

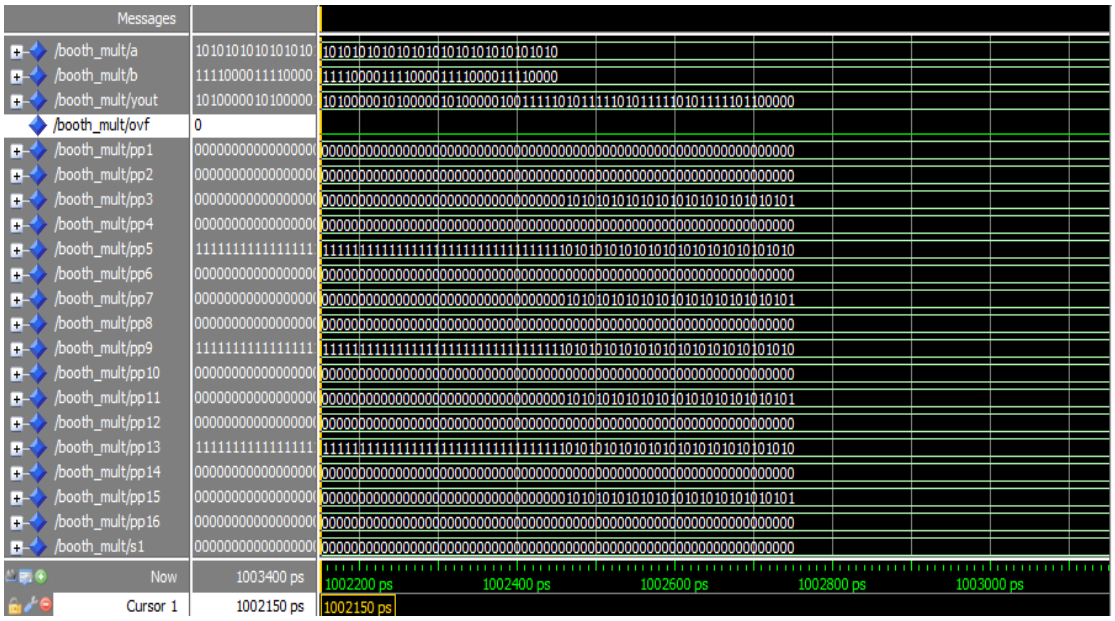

Figure 6.8: **Simulation result of 32×32 bit Booth multiplier**

Description:

a:		Input data 32-bit
b:		Input data 32-bit
Youth:		Output data 64-bit
a	=	10101010101010101010101010101010
b	=	11110000111100001111000011110000
yout	=	10100000101000001010000010011111
		01011111010111110101111101100000

Table 6.6

Comparison of 32×32 bit multipliers for Various Performance Measures

	Array multiplier	**Booth multiplier**	**Wallace multiplier**	**Booth Wallace multiplier**
Area(LUTs)	3456	4067	4378	4167
Delay(ns)	36.34	22.45	15.76	13.38
Power(mW)	89	98	108	102

6.2 Conclusion

The designs of 16×16-bit, 32×32-bit and 64×64-bit Booth Wallace multiplier have been implemented on Spartan XC3S500-5-FG320. The computation delay for 8×8 bit Booth Wallace multiplier is 8.78 ns. In addition, computation delays obtained for 16×16-bit and 32×32-bit Booth Wallace multiplier are 10.13 ns and 13.38 ns respectively. The performance of 32×32-bit multiplier increases by employing 4-stage pipelining technique. By employing pipelining, speed increases by the factor of 1.7 with penalty on area increasing by factor of 1.11. However, speed improvement ratio is larger than the area increase ratio. Hence, it is concluded that pipelining improves performance of multiplier. Power of multiplier is reduced by using low power Modified Booth Recoder. Further, comparison of four different 32-bit integer multipliers is done for various performance measures. All comparison are based on the synthesis reports keeping one common base for comparison. That means we target same FPGA device (part number and speed grade) with same design constraints implied for the synthesis of each multiplier. Delay of Wallace tree and Booth Wallace multiplier is almost same and is the least but power consumption of Wallace tree multiplier is high compared to Booth Wallace multiplier. Power consumption of Booth Wallace is more than array and Booth Multiplier. Hence, Booth Wallace multiplier is used in high-speed applications.

6.3 Future Scope

As an attempt to develop, arithmetic algorithm and architecture level optimization techniques for low power high-speed multiplier design, techniques presented in this thesis has achieved good results. However, there are limitations in this work and several future research directions are possible as follows:

- One possible direction is radix higher-than-4 recoding. Only radix-4 recoding is considered in this thesis work as it is a simple and popular choice. Higher-radix recoding further reduces the number of PPs and thus has the potential of power saving.
- In order to enhance the performance, higher order compressors like 7:2, 9:2 can be used to accumulate the partial products.
- Deep level pipeline architecture can be used for speed improvements.

The ability to construct very small high performance multipliers provides many other interesting possibilities. Multiplication intensive applications, such as DSP or graphics, could benefit significantly from several high performance multipliers on the same chip. A single very high throughput multiplier, or several multipliers working in parallel on the same chip, could open up new possibilities such as single chip video signal processors.

REFERENCES

[1] Pouya Asadi, and Keivan Navi, "A new low power 32×32- bit multiplier", World Applied Sciences Journal 2 (4): 341-347, 2007.

[2] Purushottam D. Chidgupkar, and Mangesh T. Karad, "The implementation of algorithms in digital signal processing", Global J. of Engineering Education, vol.8, no.2 © 2004 UICEE Published in Australia.

[3] Michael Andrew Lia, "Arithmetic units for high performance processors", Thesis for degree of Master of Science, University of California, 2002.

[4] Soojin Kim, and Kyeongsoon Cho, "Design of high-speed modified Booth multipliers operating at GHz ranges", World Academy of Science, Engineering and Technology, 2010.

[5] Alex Panato, Sandro Silva, Flavio Wagner, Marcelo Johann, Ricardo Reis, and Sergio Bampi, " Design of very deep pipelined multipliers for FPGAs", Proceedings of the Design, Automation and Test in Europe Conference and Exhibition Designers Forum (DATE'04)1530-1591/04, 2004.

[6] Rabey, Nikolic, and Chandrasekhran, "Digital Integrated Circuits: A Design Perspective", 2nd Edition, Prentice Hall, pp. 586-594, 2003.

[7] Roy, Kaushik, Yeo, and Kiat-Seng, "Low voltage Low-power VLSI Subsystems", McGraw-Hill, pp.124-141.

[8] M. C. Wen, S. J. Wang, Y.N. Lin, "Low-Power parallel multiplier with column bypassing", ELECTRONICS LETTERS, vol. 41, no. 10, 12th May 2005.

[9] Weste, Neil H.E. Eshraghian, and Kamran, “CMOS VLSI Design: A Circuits and Systems Perspective”, 3rd Edition, Pearson Education, pp. 345-356, 2005.

[10] A.D. Booth, “A signed binary multiplication technique”, Quart. J. Mech. Appl. Marh., vol. 4, pp. 236-240,1951.

[11] O. L. MacSorley, “High speed arithmetic in binary computers”, Proc.IRE, vol.49,pp. 67-91, 1961.

[12] L.P. Rubinfield, “A proof of the modified Booth’s algorithm for multiplication”, IEEE Trans. Comput., vol. C-24, pp. 1014-1015, Oct.1975.

[13] Razaidi Hussin, Ali Yeon Md. Shakaff, Norina Idris, Zaliman Sauli, Rizalafande Che IIsmail, and Afzan Kamaraudin, “An efficient modified Booth multiplier architecture”, International Conference on Electronic Design, 978-1-4244-2315-6/08,2008 IEEE.

[14] C.S. Wallace, “A suggestion for fast multipliers”, IEEE Trans. Electronics Comput., vol. EC-13, pp.14-17, Feb.1964.

[15] Parhami, Behrooz, “Computer Arithmetic: Algorithms and Hardware Designs”, Oxford University Press 2000.

[16] Niichi Itoh, Yuka Naemura, Hiroshi Makino, Yasunobu Nakase, Tsutomu Yoshihara, and Yasutaka Horiba, “A 600-MHz 54 x 54-bit multiplier with rectangular-styled Wallace tree”, JSSC, vol.36, no. 2, February 2001.

[17] Vojin G. Oklobdzjia, David Villeger, and Simon S. Liu, “A method for speed opitimized partial product reduction and generation of fast parallel multipliers

using an algorithmic approach", IEEE Transaction on Computer, vol. 45, no. 3, March 1996.

[18] A. Dandapat, S. Ghosal, P.Sarkar, D.Mukhopadhyay, "A 1.2ns 16X16-bit binary multiplier using high speed compressors", International Journal of Electrical and Electronics Engineering 4:3,2010.

[19] S.Shah, A.J.Kbalili,D.Al-Khabili, "Comparison of 32-bit multipliers for various performance measures", The 12th International Conference on Microelectronics Tehran, Oct. 31- Nov. 2, 2000.

[20] Gina R. Smith, "FPGAs 101: Everything you need to know to get started", Elsevier, 2010.

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

[22] Deming Chen, Jason Cong, and Peichan Pan, "FPGA Design Automation: A Survey", Foundations and Trends in Electronic Design Automation, vol. 1, Issue 3, November 2006.

[23] Ken Chapman, "Initial Design for Spartan-3E Starter Kit (LCD Display Control)", Xilinx Ltd 16th February 2006.