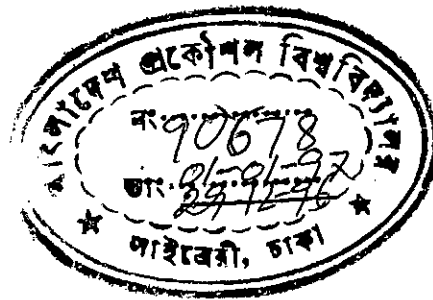


DESIGN OF AN EASILY TESTABLE ARCHITECTURE FOR SIGNED AND UNSIGNED MULTIPLICATION

A thesis submitted to the
Department of Electrical and Electronic Engineering
BUET, Dhaka
in partial fulfillment of the requirements for the degree of
Master of Science in Engineering (Electrical and Electronic)



ZAHIDUR ROUF

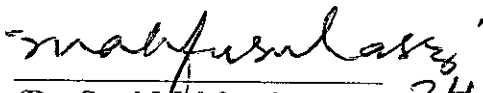
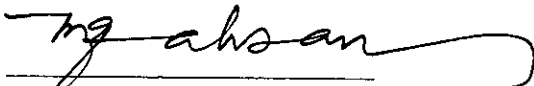
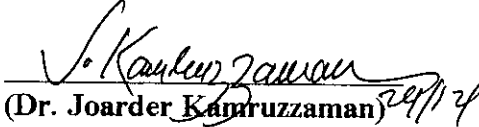
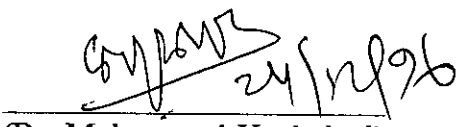
ROLL NO: 930659P
SESSION: 1992-93-94



DECEMBER 1996

The thesis titled "*Design of an Easily Testable Architecture for Signed and Unsigned Multiplication*" submitted by Zahidur Rouf, Roll No. 930659P to the Department of Electrical and Electronic Engineering, BUET has been accepted as satisfactory for partial fulfillment of the requirements for the degree of Master of Science in Engineering (Electrical and Electronic).

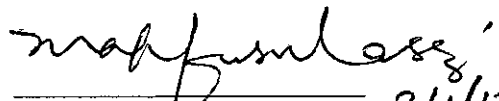
BOARD OF EXAMINERS

1. 
(Dr. Syed Mahfuzul Aziz) 24/12/96
Associate Professor
Department of Electrical and
Electronic Engineering
BUET, Dhaka-1000. Chairman
(Supervisor)
2. 
(Dr. Md. Quamrul Ahsan)
Professor and Head
Department of Electrical and
Electronic Engineering
BUET, Dhaka-1000. Member
(Ex-officio)
3. 
(Dr. Joarder Kamruzzaman) 24/12/96
Assistant professor
Department of Electrical and
Electronic Engineering
BUET, Dhaka-1000. Member
(Internal)
4. 
(Dr. Mohammad Kaykobad)
Associate Professor and Head
Department of Computer Science
and Engineering
BUET, Dhaka-1000. Member
(External)

DECLARATION

I hereby declare that this work has been done by me and it has not been submitted elsewhere for the award of any other degree or diploma.

Countersigned


(Dr. Syed Mahfuzul Aziz) 24/12/96
Supervisor


(Zahidur Rouf) 24/12/96

ACKNOWLEDGMENT

It is a matter of great pleasure on the part of the author to acknowledge his heartiest gratitude and profound obligation to his Supervisor Dr. Syed Mahfuzul Aziz, Associate professor of the Department of Electrical and Electronic Engineering, Bangladesh University of Engineering and Technology for his excellent supervision, continuous guidance and valuable suggestion throughout the progress of the work.

The author also wishes to express his special thanks and gratitude to Dr. Md. Quamrul Ahsan, Professor and Head of the Department of Electrical and Electronic Engineering, BUET for his all-out support. The author is also grateful to Dr. A. B. M. Siddique Hossain, Professor and former Head of the Department of Electrical and Electronic Engineering, BUET for his help and cooperation..

Finally the author would like to express thanks to all his friends and colleagues and staff of the Department of Electrical and Electronic Engineering, BUET for their constant support and assistance.

ABSTRACT

High speed array processors are now an integral part of most VLSI systems, for example, signal processors, satellite imaging systems etc. With the increasing complexity of VLSI circuits, it is in the manufacturers' interest to give due consideration to testability at the very early stages of chip design.

Array multipliers are part of many signal processing and other systems. This thesis examines the available array multipliers which can perform multiplication of either signed or unsigned binary numbers. A generalized multiplier capable of performing both types of multiplication is designed. The hardware overhead of the proposed generalized multiplier is very low. A C-testable version of this generalized multiplier is designed for multiplicands (X) having even number of bits and multipliers (Y) having odd number of bits. This design is shown to be testable for any single stuck at fault with only 19 test vectors irrespective of the operand wordlengths. Compared to an exhaustive functional testing approach, this would reduce the chip testing time and there by reduce the cost by a great margin, The testable design requires eight extra inputs which would not impose heavy penalty on the number of pins for large multipliers.

CONTENTS

Acknowledgment	iv
Abstract	v
List of Figures	ix
List of Tables	xi
List of Abbreviations	xii
CHAPTER 1 Introduction	1
1.1 Aims	1
1.2 Literature Review	2
1.3 Organization of the Thesis	4
CHAPTER 2 Multiplication Algorithms	5
2.1 Introduction	5
2.2 Straightforward Carry-Save Array Multiplier	5
2.3 Booth Algorithm	7
2.4 Modified Booth Algorithm	9
2.5 Removal of Sign-Bit Extension Circuitry	12
2.6 Architecture Based on Modified Booth Algorithm	14

CHAPTER 3	Generalized Architecture for Signed and Unsigned Multiplication	17
3.1	Introduction	17
3.2	Multiplication of Unsigned Numbers Using Modified Booth Algorithm	17
3.3	Hardware Implementation of Unsigned Multiplication	21
3.3.1	Sign-Extension Circuitry for the Multiplicand (X)	21
3.3.2	Sign-Extension of the Multiplier (Y) with Odd Number of Bits	22
3.3.4	Sign-Extension of the Multiplier (Y) with Even Number of Bits	23
3.4	Signed and Unsigned Multiplication Using a Single Array	24
3.4.1	Selection of Multiplicand Sign Extension Bit	24
3.4.2	Selection of Multiplier Sign-Extension Bit	25
3.5	The Generalized Architecture	28
3.6	Calculation of Overhead	
3.6.1	Hardware Overhead	28
3.6.2	Delay Overhead	34
CHAPTER 4	Testability of the Generalized Multiplier	35
4.1	Introduction	35
4.2	Testing Approach	35
4.3	Modification of the Architecture for Testability	36
4.4	Testing of Individual Cells	38

4.4.1	Testing of the MBEs for Single Stuck-at Fault	38
4.4.2	Testing of the SCs for Single Stuck-at Fault	40
4.4.3	Testing of the Mode-Selector for Single Stuck-at Fault	42
4.5	Testing the Multiplier	43
4.5.1	Test Vectors	43
4.5.2	Exhaustive Testing of the FAs	45
4.5.3	Exhaustive Testing of the MCAs	47
4.5.4	Testing of The MBEs	47
	4.5.4.1 Exhaustive Testing	48
	4.5.4.2 Testing for Single Stuck-at Fault	48
4.5.5	Test Vectors for SCs	49
4.5.6	Test Vectors for MSs	50
4.6	Calculation of Overhead	51
	4.6.1 Hardware Overhead	51
	4.6.2 Delay Overhead	51
4.7	Difficulties with Even Number of Multiplier Bits	52
4.8	Difficulties with Odd Number of Multiplicand Bits	53
4.9	Summary	56
CHAPTER 5 Conclusions and Recommendations		57
5.1	Conclusions	57
5.2	Future Works	58
References		59

Fig. 3.11	Gate level design of Full-adder and Manchester carry adder circuits	31
Fig. 3.12	Gate level design of Modified Booth encoder (MBE)	31
Fig. 3.13	Gate level design of Selector-Complementer circuit	32
Fig. 3.14	Logic diagram of Mode-Selector circuit	32
Fig. 4.1	An 8×7 bit generalized testable modified Booth multiplier	37
Fig. 4.2	Gate level design of modified Booth encoder (MBE)	38
Fig. 4.3	Gate level design of Selector-Complementer circuit	40
Fig. 4.4	Logic diagram of Mode-Selector block	42
Fig. 4.5	Arrangement of MS and extra row MBE when Y got even number of bits (8 bits)	51
Fig. 4.6	Effect of vector pair t_9 and t_{10} applied to a 9×9 bit multiplier	52
Fig. 4.7	Effect of vector pair t_{20} and t_{21} applied to a 9×9 bit multiplier	53
Fig. 4.8	Effect of vector pair t_{22} and t_{23} applied to a 9×9 bit multiplier	54

List of Figures

Fig. 2.1	A parallel multiplier array using carry save adders	6
Fig. 2.2	Multiplication example using bit-pair recoding	11
Fig. 2.3	Sign extended partial product array	12
Fig. 2.4	Recoded sign extended partial product array	14
Fig. 2.5	An 8 by 8 modified Booth multiplier array	16
Fig. 3.1	Signed and unsigned multiplication using modified Booth algorithm for odd number of bits in the operands	19
Fig. 3.2	Signed and unsigned multiplication using modified Booth algorithm for even number of bits in the operands	20
Fig. 3.3	Modification to sign extension circuitry for unsigned multiplication	21
Fig. 3.4	Sign extension of the multiplier(Y) having odd number of bits	22
Fig. 3.5	Sign extension of the multiplier(Y) having even number of bits	23
Fig. 3.6	Selection of proper version of MSB of X	25
Fig. 3.7	Arrangement of the Mode Selector and MBE when Y has odd number of bits	26
Fig. 3.8	Arrangement of the Mode Selector and extra row MBE when Y has even number of bits	27
Fig. 3.9	A 7 by 7 bit generalized modified Booth multiplier	29
Fig. 3.10	A 6 by 6 bit generalized modified Booth multiplier	30

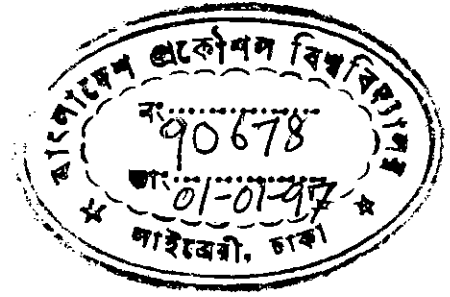
List of Tables

Table 2.1.	Multiplier bit-pair recoding scheme	11
Table 2.2	Modified Booth recoding table	15
Table 3.1	Operation of the Mode-Selector for X	24
Table 3.2	Operation of the Mode-Selector for Y	26
Table 3.3	Hardware requirements	33
Table 4.1	Fault matrix for the MBE logic circuit	39
Table 4.2	Fault matrix for the selector block	41
Table 4.3	Fault matrix for the MS	43
Table 4.4	A set of test vectors for an 8×7 bit generalized multiplier	44
Table 4.5	Exhaustive testing of the FAs	46
Table 4.6	Exhaustive testing of the MCAs	47
Table 4.7	Exhaustive testing of the MBEs	48
Table 4.8	Extra test vectors for a 9×9 bit multiplier	54

List of Abbreviations

DCVS	Differential Cascode Voltage Switch
FA	Full-Adder
LSB	Least Significant Bit
MBE	Modified Booth Encoder
MCA	Manchester Carry Adder
MS	Mode-Selector
MSB	Most Significant Bit
MUX	Multiplexers
SC	Selector-Complementer

Chapter 1



Introduction

1.1 Aims

Multiplication of binary numbers is an essential function in many applications, for example, digital signal processing, digital filtering, convolution and frequency analysis, etc. [1]-[4]. Some systems require both signed and unsigned multiplication. The hardware implementation of two separate architectures for signed and unsigned multiplication in a particular application is costly. The conventional array multipliers, that are available, perform multiplication of either signed or unsigned binary numbers. Hence a single architecture capable of performing both signed and unsigned multiplication will make effective use of the silicon area and at the same time increase the speed of multiplication. The objective of this research is to develop a regular parallel multiplier architecture for performing both two's complement (signed) and sign-magnitude (unsigned) multiplication of two binary numbers. This research also aims at augmenting the above architecture into an easily testable one using minimum extra hardware. The proposed architecture will have useful applications in arithmetic processors, digital filters, digital signal processors, etc. It will be suitable for use in logic synthesis tools for automatic generation of parallel multiplier layouts.

1.2 Literature Review

Various multiplication algorithms are available these days. Some of these algorithms perform multiplication of unsigned binary numbers while other accomplish multiplication of signed numbers. The sequential add-shift multiplier [5] performs multiplication of two unsigned numbers. It generates the partial products sequentially bit by bit and uses register latches to shift and store intermediate partial products. It uses minimum hardware, but the speed of multiplication is quite low. The most common form of parallel multiplier is based on the straightforward carry-save array multiplication algorithm [5]-[7]. This multiplier is suited only to positive operands. For multiplication of two n -bit numbers this architecture requires $n(n-2)$ full adders, n half adders, n^2 AND gates. The worst case delay associated with such multiplier is $(2n+1)\tau_g$, where τ_g is worst-case adder delay. The multiplier has a regular, compact structure suitable for VLSI implementation. A fast multiplication scheme was proposed by C. S. Wallace [8]. This multiplier is based on the use of trees of pseudo-adders, effectively composed of a set of counters, which converts three numbers to two in such a way that the value of the output word is equal to the number of '1's in the input word. Thus, rather adding together many summands, one pass through such pseudo-adder reduces the number of summands left to be summed by one. One important property of Wallace tree is that the number of adder cells needed grows as the logarithm $\log_2(n)$ of the number of input bits n . These adders are much faster than the conventional carry-save adders. The multiplier array requires irregular interconnections among various cells. It also requires a great deal of hardware although greater speed of multiplication may be achieved. The time optimal Dadda multiplication scheme [9], [10] operates faster than carry-save array multiplier by minimizing the number of adder cells as well as the critical path between the partial product generation and the final addition. However, this architecture uses irregular interconnections among various

cells which makes the layout design complicated. A high speed two's complement parallel multiplication algorithm in which the signs of all the partial product bits are positive allowing the product to be formed using array addition techniques was proposed by Baugh and Wooley [11]. One disadvantage of the scheme is the need for the complements of each multiplier and multiplicand bit in forming the partial product bits. The modified Booth algorithm [5], [12] for two's complement multiplication essentially reduces the number of partial products by a factor of two compared to the straightforward carry-save array multiplier. Multiplication speed is almost doubled. Besides, there is no need for precomplementing the multiplier or postcomplementing the product. Also, the multiplier structure is regular, therefore suitable for VLSI implementation. Overlapped multi-bit scanning algorithm [13] which scans more than three bits at a time has a potential to improve the multiplication speed over the modified Booth multiplier. However, to produce the various multiples of the multiplicand, extra hardware is needed thereby increasing the multiplier size and spoiling the regularity of design.

With the advance of integrated circuit technology, the implementation of large array multipliers on a single chip has become possible. However, due to the increasing complexity of VLSI circuits it is becoming more and more difficult and costly to test them [14], [15]. As a result, it is a common practice among circuit designers these days to give due consideration to testability at the early stages of design. Extra hardware and/or inputs are added to the original circuits to make them easily testable thereby reducing testing time and cost. The testability of parallel array multipliers have been investigated by several researchers. A number of testable multiplier architectures have been proposed by them. In [16] C-testable designs of carry-save array multiplier and Baugh-Wooley's two's complement array multiplier are presented. The term "C-testable" means that the multipliers require a constant number of test vectors irrespective of the size of the operands [17]. Two designs of easily testable gate-level and DCVS logic multipliers have been proposed in [18]. These designs are based on the straightforward carry-save array

multiplication scheme and have been shown to be C-testable. Gate-level C-testable multipliers based on the modified Booth algorithm have been presented in [19] and [20]. A C-testable DCVS design using this algorithm has also been presented in [21]. This research aims at developing a C-testable architecture capable of performing the multiplication of both signed as well as unsigned operands.

The proposed multiplier is based on the modified Booth algorithm. There is some specific reasons for this particular choice. First of all it reduces the number of partial products to almost half compared to straightforward carry-save array multiplier. This algorithm scans three multiplier bits at a time to produce multiples of the multiplicand which can be achieved just by shifting and/or complementing the multiplicand. Attempts to reduce the number of partial products further appear to require multiples not to be obtainable only by shifting [13]. These partial products can be added with an array of conventional carry-save adders. Besides, the multiplier has a regular structure which is an extremely important criterion in the selection of schemes for VLSI design. Also, due to its regular iterative structure the multiplier can be modified to an easily testable one at the expense of a very little extra logic and some extra inputs.

1.3 Organization of the Thesis

Chapter 2 presents the parallel multiplication scheme using a straightforward array of carry-save adders. It also introduces the Booth algorithm for multiplication of signed binary numbers. Bit-pair recoding technique and modified Booth multiplier is also presented in this chapter. Chapter 3 presents the design of a generalized architecture based on the modified Booth algorithm which is capable of performing multiplication of both signed and unsigned binary numbers. Chapter 4 analyzes the testability of the proposed generalized architecture and presents the design of a C-testable multiplier. Finally the conclusions and some recommendations for future research are made in chapter 5.

Chapter 2

Multiplication Algorithms

2.1 Introduction

Multiplication of two fixed point binary operands will be discussed in this chapter. Some most common parallel multiplication schemes such as the straightforward carry-save array multiplication, Booth algorithm and method of bit-pair recoding or modified Booth algorithm will be considered. Also, an architecture based on the modified Booth algorithm for multiplication of two signed numbers will be presented in this chapter.

2.2 Straightforward Carry-Save Array Multiplier

Multiplication can be defined as repeated addition. The number to be added is the multiplicand, the number of times it is added is the multiplier, and the result is the product. Each step of addition generates a partial product and when the operands are integer the product is twice the length of the operands in order to preserve the information content. It should be noted that binary multiplication is equivalent to a logical AND operation. Thus the evaluation of partial products consists of the logical ANDing of the multiplicand and the relevant multiplier bit. Each column of partial products must then be added and, if

necessary, any carry values passed to the next column. A parallel multiplier [6] is based on the observation that all partial products in the multiplication process may be independently computed in parallel. The partial product terms are called summands. If the multiplicand and the multiplier has m and n bits respectively then there will be $m \times n$ summands, which are produced by a set of mn AND gates. In a straightforward carry-save array multiplier the summands are collected through a cascaded array of carry-save adders. At the bottom of the array, an adder is used to convert the "carry save form" to the required form of output. Multiplication time is fixed by the depth of the array and the carry propagation characteristics of the adder.

Fig. 2.1 shows a 4×4 bit straightforward carry save array multiplier with the partial products enumerated [6]. The basic cell that may be used to construct this parallel multiplier is also shown in this figure. The multiplicand term x_i is propagated vertically, while the multiplier term y_j is propagated horizontally. Incoming partial product bits enter at the top and the incoming CARRY IN bits enter at the top right of the cell. The bit-wise AND operation is performed in the cell, and the SUM is passed to the next cell at the lower right. The CARRY OUT is passed to the bottom of the cell.

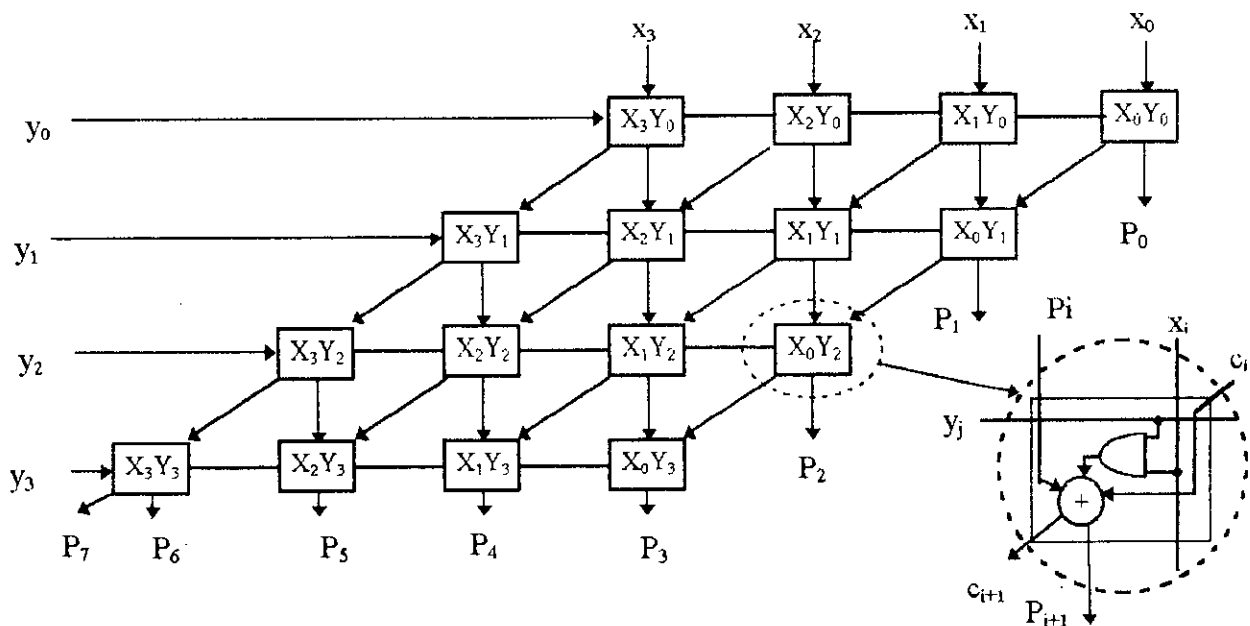
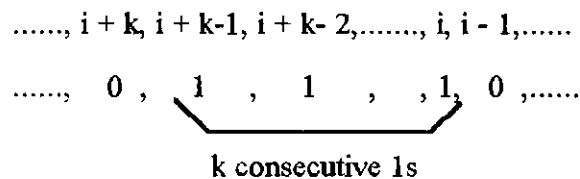


Fig. 2.1 A parallel multiplier array using carry save adders

2.3 Booth Algorithm

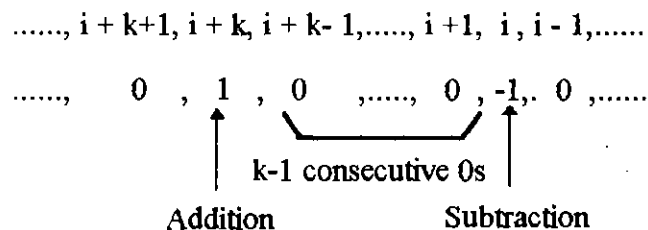
Booth algorithm is a powerful direct tool for signed-number multiplication [5]. In the standard add--shift method, each non zero bit of the multiplier causes one addition of the multiple of multiplicand to the partial product. The well known fact is that the execution time of multiplication instruction is determined mainly by the number of additions to be performed. So the execution time can be reduced if we can reduce the number of additions. This is achieved by a method of bit-scanning which reduces the number of multiplicand multiples. This technique uses recoding of the multiplier based on the string property. The process is often referred to as "skipping over 0s" and can be generalized to shift of variable lengths if string of 0s can be detected. The greater the number of 0s in the multiplier the faster the operation. Consider a string of k consecutive 1s in the multiplier as shown below.



by using the following property of binary strings

$$2^{i+k} - 2^i = 2^{i+k-1} + 2^{i+k-2} + \dots + 2^{i+1} + 2^i \quad (2.1)$$

The consecutive 1s can be replaced by the following string



Now consider a multiplication example in which a positive multiplier has a single block of 1s with at least one 0 at each end, for example 0 0 1 1 1 0 (14). The number of addition can be reduced by observing that a multiplier in this form can be regarded as the difference of two numbers as follows:

$$\begin{array}{r}
 010000 \quad (16) \\
 -) 000010 \quad (2) \\
 \hline
 001110 \quad (14)
 \end{array}$$

This was shown in Eq. 2.1 and indicates that the product can be generated by one addition (addition of 2^4) and one subtraction (subtraction of 2^1). In the standard notation, the multiplier can be written as

$$0 \ 0 \ +1 \ +1 \ +1 \ 0$$

and the recoded multiplier can be written as

$$0 \ +1 \ 0 \ 0 \ 0 \ -1 \ 0$$

Note that the -1 times the left-shifted multiplicand occurs at 0 to 1 boundaries and +1 times the left-shifted multiplicand occurs at 1 to 0 boundaries as the multiplier is scanned from right to left. The transformation that takes

$$01111 \dots 1110 \quad \text{into} \quad +10000 \dots 0-10$$

is often referred as the technique of skipping over 1s. The reasoning is that in cases in which the multiplier has its 1s grouped into a few blocks, only a few versions of the multiplicand need to be added to generate the product hence, the multiplication process becomes much faster. It can also be shown that the Booth recoded multiplier algorithm works equally well for negative multiplier.

2.4 Modified Booth Algorithm

Modified Booth algorithm is a multiplication speedup technique that guarantees that an n-bit multiplier will generate at most $n/2$ partial products [5],[12]. It can multiply two two's complement numbers directly and gives the product also in the same form. This represents a multiplication speed increase of almost a factor of 2 over the standard add-shift method.

This new technique is derived from the Booth technique. Recall from the previous discussion of a positive multiplier of 0 0 1 1 1 0 (+14). The number of addition can be reduced by observing that the multiplier in this form can be regarded as the difference of two numbers as shown below.

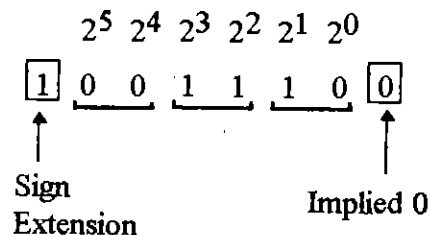
$$\begin{array}{r}
 2^5 \ 2^4 \ 2^3 \ 2^2 \ 2^1 \ 2^0 \\
 0 \ 1 \ 0 \ 0 \ 0 \ 0 \quad (16) \\
 -) \ 0 \ 0 \ 0 \ 0 \ 1 \ 0 \quad (2) \\
 \hline
 \text{Multiplier} \rightarrow 0 \ 0 \ 1 \ 1 \ 1 \ 0 \quad (14)
 \end{array}$$

This indicate that the number 0 0 1 1 1 0 (14) has the same value as

$$2^4 - 2^1 = 16 - 2 = 14$$

This is true for any number of contiguous 1s, including the case in which there is a single 1 with 0s on either side. The entire concept of bit-pair recoding revolves around this method of regarding a string of 1s as the difference of two numbers.

Now returning to the multiplier being discussed and scanning it from right to left, bit by bit. In going from 0 (2^0) to 1 (2^1), we saw previously that this resulted in subtracting the value of the 1 in that position, in this case -2^1 . Scanning from 1 (2^1) to 1 (2^2) resulted in no change, that is, neither addition nor subtraction. The same is true in scanning from 1 (2^2) to 1 (2^3). However, in going from 1 (2^3) to 0 (2^4), we saw that this resulted in an addition of 2^4 . There is no change in scanning from 0 (2^4) to 0 (2^5). The results of scanning this multiplier are as follows: 2^1 was subtracted and 2^4 was added. The same results can be obtained by looking at pairs of bits in the multiplier in conjunction with the bit that is to the right of the bit pair being considered, as shown below.



That is, bit pair $2^1, 2^0$ is examined with an implied 0 to the right of the low-order bit; bit pair $2^3, 2^2$ is examined with bit 2^1 , bit pair $2^5, 2^4$ is examined with bit 2^3 . Scanning the bit pairs from right to left and using the rightmost bit of each pair as the column reference for the partial product placement (it is the center bit of the three bits being examined), we obtain the following multiplier bit-pair recoding scheme shown in table 2.1. It should be noted that there are a total of eight possible versions of the multiplicand.

Table 2.1. Multiplier bit-pair recoding scheme

Multiplier bit-pair		Multiplier bit on the right	Multiplicand multiples to be added	Explanation
i+1	i	i-1		
0	0	0	0 × multiplicand	No string
0	0	1	+ 1 × multiplicand	End of string
0	1	0	+ 1 × multiplicand	Single 1 (+2 -1)
0	1	1	+ 2 × multiplicand	End of string
1	0	0	- 2 × multiplicand	Beginning of string
1	0	1	- 1 × multiplicand	End/beginning of string
1	1	0	-1 × multiplicand	Beginning of string
1	1	1	0 × multiplicand	Strings of 1s

Fig. 2.2 gives an example of the bit-pair recoding multiplication technique using two 5 bit operands represented in two's complement form.

Multiplicand X = 0 0 1 1 0 (+ 6)

Multiplier Y = [1] 1 0 0 1 0 [0] (- 14)
-1x +1x -2x

 1 1 1 1 1 0 1 0 0

 0 0 0 0 0 1 1 0

 1 1 1 0 1 0

Product P = 1 ← 1 1 1 0 1 0 1 1 0 0 (- 84)

Fig. 2.2 Multiplication example using bit-pair recoding

2.5 Removal of Sign-bit Extension Circuitry

The modified Booth algorithm for multiplying two binary numbers basically consists of two steps: first obtain the partial product from the proper version of the multiplicand and second add these partial products in an appropriate array of full adders considering that summation in an array has to be done with sign bit extension, because it is a signed multiplication. However, if explicit sign extension scheme is observed large amount of circuitry is required merely to accommodate the sign-extension of the partial products. The redundancy of the sign-bit extension can be eliminated by a simple method, i.e., reducing the number of variable inputs to the array, thus reducing the number of full adders involved. Several approaches of removal of sign-extension circuitry from Booth multiplier have been proposed by previous researchers [22], [23].

Let us consider the multiplication of two 8-bit binary numbers using modified Booth algorithm. Since this algorithm scans three bits of the multiplier at a time and retires two of them to generate a partial product, the total number of partial products generated for the 8-bit multiplier is four. If a, b, c, d represent these partial products, then the addition of these partial product is illustrated in Fig. 2.3. Each partial product is shifted two bit positions to the left with respect to the preceding one in accordance with the modified booth algorithm.

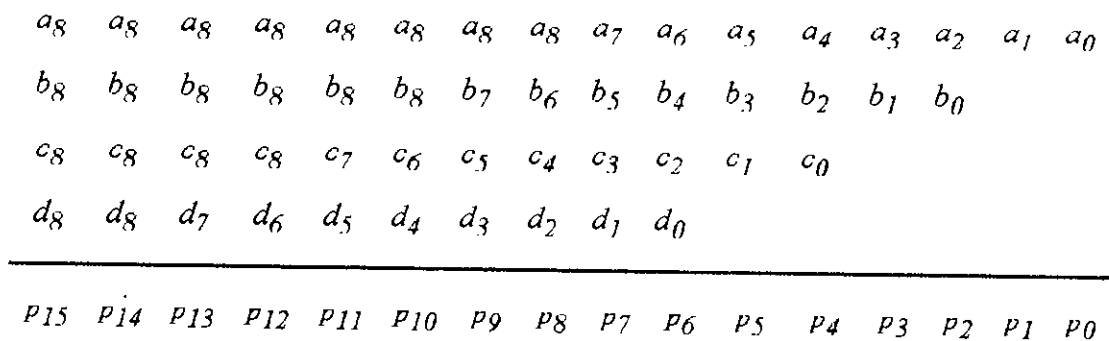


Fig 2.3 Sign extended partial product array

Here a_8, b_8, c_8, d_8 are the sign bits. It is seen that the direct implementation of explicit sign extended array will be an uneconomical choice.

Let us assume for simplicity that the arithmetic weight of the p_8 column is 2^0 , i.e., 1. Thus the p_{15} column represents a weight of 2^7 . Then the sum of the sign and sign extended bits can be written as

$$\begin{aligned} \text{Sum} &= a_8(2^7 + 2^6 + 2^5 + 2^4 + 2^3 + 2^2 + 2^1 + 2^0) + b_8(2^7 + 2^6 + 2^5 + 2^4 + 2^3 \\ &\quad + 2^2) + c_8(2^7 + 2^6 + 2^5 + 2^4) + d_8(2^7 + 2^6) \\ &= a_8(2^8 - 2^0) + b_8(2^8 - 2^2) + c_8(2^8 - 2^4) + d_8(2^8 - 2^6) \end{aligned}$$

Since p_{15} is the most significant bit of the product output, module 2^8 addition can be used to sum the sign bits. Thus, the sum of the sign bits can be written as

$$\text{Sum} = -a_8(2^0) - b_8(2^2) - c_8(2^4) - d_8(2^6)$$

which expressed as a binary number is

$$\text{Sum} = -(0 d_8 0 c_8 0 b_8 0 a_8) \quad (2.2)$$

The two's complement of the word $(0 d_8 0 c_8 0 b_8 0 a_8)$ is

$$-(0 d_8 0 c_8 0 b_8 0 a_8) = (1 \bar{d}_8 1 \bar{c}_8 1 \bar{b}_8 1 \bar{a}_8) + 1 \quad (2.3)$$

When the recoding scheme of Eq. 2.3 is used, the sign extended Booth partial product array appears like the one shown in Fig. 2.4.

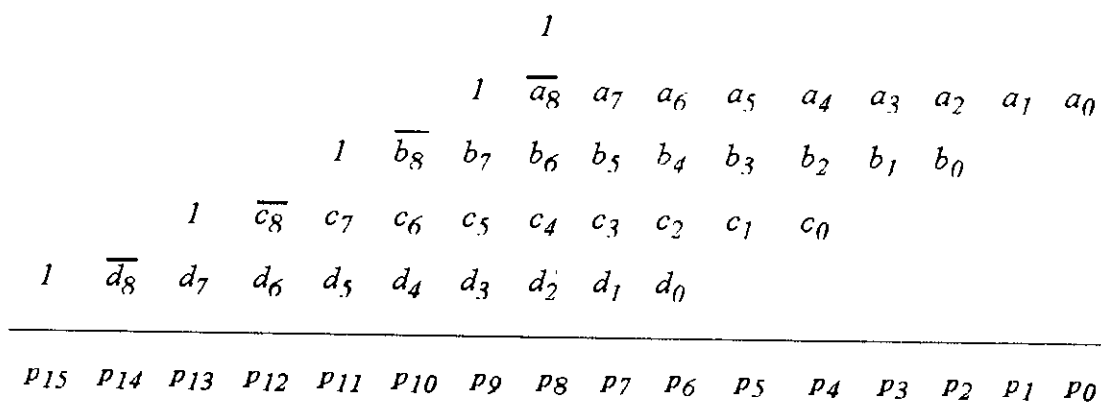


Fig 2.4 Recoded sign extended partial product array

Hence it is seen that elimination of the sign-extension circuitry in a modified Booth algorithm multipliers can be achieved by inverting the MSB of each partial product and adding a logic 1 at every higher significance (including the MSBs). This procedure is equivalent to recoding the MSBs of the partial products as a two's complement number and adding a logic 1 to the most significant full adder in each row of the main array.

2.6 An Architecture Based on Modified Booth Algorithm

Fig. 2.5 represents an 8 by 8 bit multiplier architecture based on the modified Booth algorithm for multiplication of two binary numbers that are in the two's complement form [21]. Elimination of the sign extension circuitry is achieved by the procedure described above. The modified Booth encoder (MBE) block in each row operates on three multiplier bits to generate the control signal CM, K_1 , and K_2 according to the modified Booth recoding scheme as shown in Table 2.2. In this recoding scheme five possible partial products can be formed: 0, +X, -X, +2X, -2X where X denotes the multiplicand. The selector complementers (SC) in Fig 2.5 consist of multiplexers which operate on the multiplicand bits to generate 0, X or 2X as partial products depending on the control

signals K_1 , K_2 and complementers (2-input EX-OR gates) which generate one's complements of these partial products only when CM signal is high. Moreover, these one's complemented partial products are converted to their two's complement form by addition of a logic '1' to their LSBs. The addition of the partial products are accomplished by an array of carry save full adders (FA). The Manchester carry adders (MCA) on the right-hand side and the bottom of the Fig. 2.5 operates on the results coming out of the main array (the array containing SCs and FAs) to generate the final product output.

Table 2.2: Modified Booth recoding table

MBE inputs			MBE outputs			Partial Product	SC output
Y_{i+1}	Y_i	Y_{i-1}	K_1	K_2	CM	Generated	Z
0	0	0	0	0	0	0	0
0	0	1	1	0	0	+X	X_i
0	1	0	1	0	0	+X	X_i
0	1	1	0	1	0	+2X	X_{i-1}
1	0	0	0	1	1	-2X	\bar{X}_{i-1}
1	0	1	1	0	1	-X	\bar{X}_i
1	1	0	1	0	1	-X	\bar{X}_i
1	1	1	0	0	0	0	0

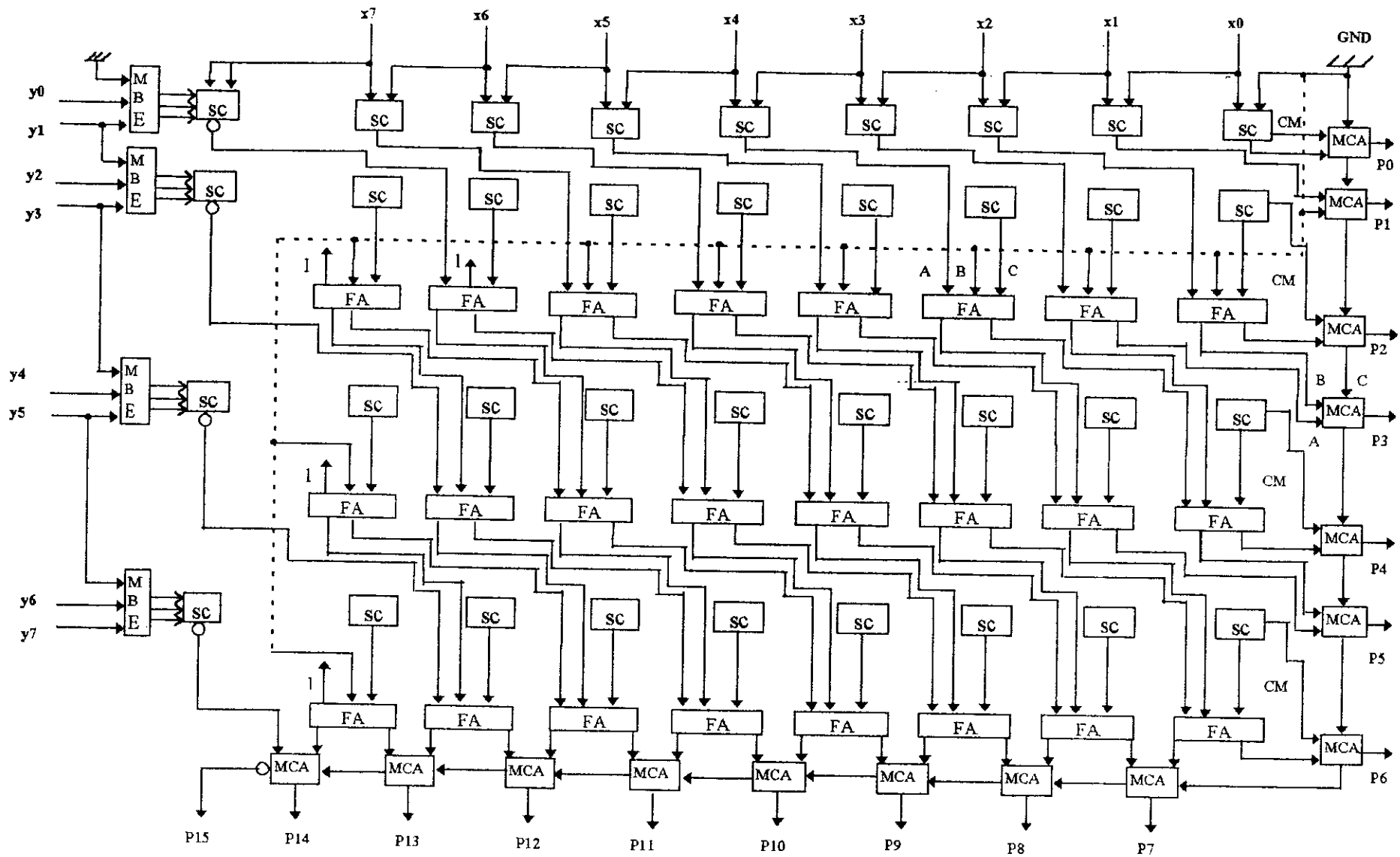


Fig 2.5 : An 8 by 8 bit modified Booth multiplier array
 (Horizontal Controls and vertical multiplicand routings are omitted for clarity)

Chapter 3

Generalized Architecture for Signed and Unsigned Multiplication

3.1 Introduction

This chapter investigates how the modified Booth algorithm can be applied to multiply binary numbers represented in the sign-magnitude (unsigned) form. The modifications needed to the modified Booth multiplier for performing multiplication of unsigned numbers will be derived. Finally a single generalized architecture capable of performing both signed and unsigned multiplication will be developed.

3.2 Multiplication of Unsigned Numbers Using Modified Booth Algorithm

Traditionally the modified Booth algorithm is regarded as a powerful tool for multiplication of two's complement numbers and gives the product output in two's complement form. This algorithm considers that any number with a 1 in its leftmost bit position (i.e., MSB) is a negative number and is given in the two's complement form. This leftmost bit is regarded as the sign bit. Positive numbers are represented in simple positional

binary notation with the sign bit set to 0. However, in order to use the modified Booth algorithm for unsigned multiplication the operands with a leading 1 (i.e., MSB=1) must not be considered as a negative number, rather all the bits in the operand (including the MSB) should indicate its true magnitude. This can be achieved by placing an extra 0 as the MSB of the operands. The operands will then be treated as positive numbers by the modified Booth multiplier. Therefore, the results of the multiplication will be the same as it would be if the original unaugmented operands were considered to be unsigned binary numbers.

Fig. 3.1 illustrates two examples of how two binary numbers can be multiplied according to the modified Booth algorithm in signed and unsigned mode of multiplication. Let us consider two 5 bit binary numbers X and Y as shown in Fig. 3.1(a). In this figure it is assumed that the numbers are represented in the two's complement notation and multiplication is performed using the original modified Booth algorithm. In Fig. 3.1(b), an extra zero is forced at the MSB of both the operands to make them positive and to perform unsigned multiplication using the same algorithm.

Some important points are worth noting in the multiplication examples of Fig. 3.1. First, during the normal signed multiplication (Fig. 3.1(a)) the MSBs of the operands are extended as sign bits. But when unsigned multiplication is desired (Fig. 3.1 (b)), the extended sign bit must be a zero instead of the MSB of the operands. Second, in the signed multiplication example of Fig. 3.1(a), since the multiplier (Y) has odd number of bits, the MSB is extended to make it even thereby allowing the completion of bit-pair recoding at the extended sign-bit. Three partial products are generated in this case. In the example of Fig. 3.1(b) forcing an extra '0' at the most significant bit position for unsigned multiplication also renders the number of bits in the multiplier (Y) even. In this case the number of partial products generated is three as well. Therefore, regardless of the number of bits in the multiplicand (odd or even), if the multiplier has odd number of bits then the number of partial products generated during either signed or unsigned multiplication process are equal.

$$\begin{array}{r}
 \text{Multiplicand X} = \quad \quad \quad 1\ 1\ 0\ 1\ 0 \quad (-6) \\
 \text{Multiplier Y} = \quad \quad \quad [1] 1\ 0\ 1\ 0\ 1 [0] \quad (-11) \\
 \begin{array}{l}
 \text{Sign} \rightarrow \quad \quad \quad \underbrace{\quad \quad \quad}_{-1x} \quad \underbrace{\quad \quad \quad}_{+1x} \quad \underbrace{\quad \quad \quad}_{+1x} \\
 \text{extension} \rightarrow \quad \quad \quad \hline
 1\ 1\ 1\ 1\ 1\ 1\ 0\ 1\ 0 \\
 1\ 1\ 1\ 1\ 1\ 0\ 1\ 0 \\
 0\ 0\ 0\ 1\ 1\ 0 \\
 \hline
 0\ 0\ 0\ 1\ 0\ 0\ 0\ 1\ 0 \quad (+66)
 \end{array}
 \end{array}$$

3 rows of
Partial Product

(a) Signed Multiplication

$$\begin{array}{r}
 \text{Multiplicand X} = \quad \quad \quad [0] 1\ 1\ 0\ 1\ 0 \quad (26) \\
 \text{Multiplier Y} = \quad \quad \quad [0] 1\ 0\ 1\ 0\ 1 [0] \quad (21) \\
 \begin{array}{l}
 \text{Forced} \quad \quad \quad \underbrace{\quad \quad \quad}_{+1x} \quad \underbrace{\quad \quad \quad}_{+1x} \quad \underbrace{\quad \quad \quad}_{+1x} \\
 \text{Zeros} \quad \quad \quad \hline
 0\ 0\ 0\ 0\ 0\ 1\ 1\ 0\ 1\ 0 \\
 0\ 0\ 0\ 1\ 1\ 0\ 1\ 0 \\
 0\ 1\ 1\ 0\ 1\ 0 \\
 \hline
 1\ 0\ 0\ 0\ 1\ 0\ 0\ 0\ 1\ 0 \quad (546)
 \end{array}
 \end{array}$$

3 rows of
Partial Product

(b) Unsigned Multiplication

Fig. 3.1 (a) Signed and (b) Unsigned multiplication using modified Booth algorithm for odd number of bits in the operands

Now consider that there are even number of bits in the multiplier. The application of normal modified Booth algorithm for signed multiplication requires no sign extension of the multiplier (Y) as bit-pair recoding completes at its MSB. However, during unsigned multiplication forcing an extra zero at the MSB of the multiplier transforms it into a positive number and makes the number of bits in the multiplier odd. Hence it requires a

second extra zero at the multiplier MSB for the completion of bit-pair recoding resulting in an extra row of partial product compared to signed multiplication. This is illustrated in the Fig. 3.2 where two 4-bit binary numbers are multiplied in both signed as well as unsigned mode of multiplication.

Multiplicand X =	1 1 1 0	(- 2)	
Multiplier Y =	1 0 0 1	[0] (- 7)	
		-2x +1x	
	1 1 1 1 1 1 1 0		} 2 rows of Partial Product
	0 0 0 1 0 0		
Product P =	0 0 0 0 1 1 1 0	(+ 14)	

(a) Signed Multiplication

Multiplicand X =	[0] 1 1 1 0	(+14)	
Multiplier Y =	[0] [0] 1 0 0 1 [0]	(+9)	
		+1x -1x -2x	
	0 0 0 0 1 1 1 0		} Extra row of Partial Product
	1 0 0 1 0 0		
	1 1 1 0		
Product P =	0 1 1 1 1 1 1 0	(+126)	

(b) Unsigned Multiplication

Fig. 3.2 (a) Signed and (b) Unsigned multiplication using modified Booth algorithm for even number of bits in the operands

3.3 Hardware Implementation of Unsigned Multiplication

The changes required in the architecture of the modified Booth multiplier of Fig. 2.5 in order to achieve unsigned multiplication will be discussed here. Detailed designs of the new hardware blocks needed will be presented.

3.3.1 Sign Extension Circuitry for the Multiplcand (X)

Recalling the modified Booth multiplier architecture of Fig. 2.5, the sign extension of the multiplicand (X) is accomplished by the left most selector-complementer (SC) blocks of each row. Both the inputs to these SCs are same which is the MSB of the multiplicand. It was shown that the output of these SCs are inverted so that complemented version of the sign bit of X is available as required for removal of explicit sign-bit extension circuitry. But if unsigned multiplication is desired then instead of passing the MSB of X to the left most SCs of each row logic zero has to be forced to these blocks. Hence for unsigned multiplication both the inputs to the left most SCs in each row should be grounded as shown by the Fig. 3.3.

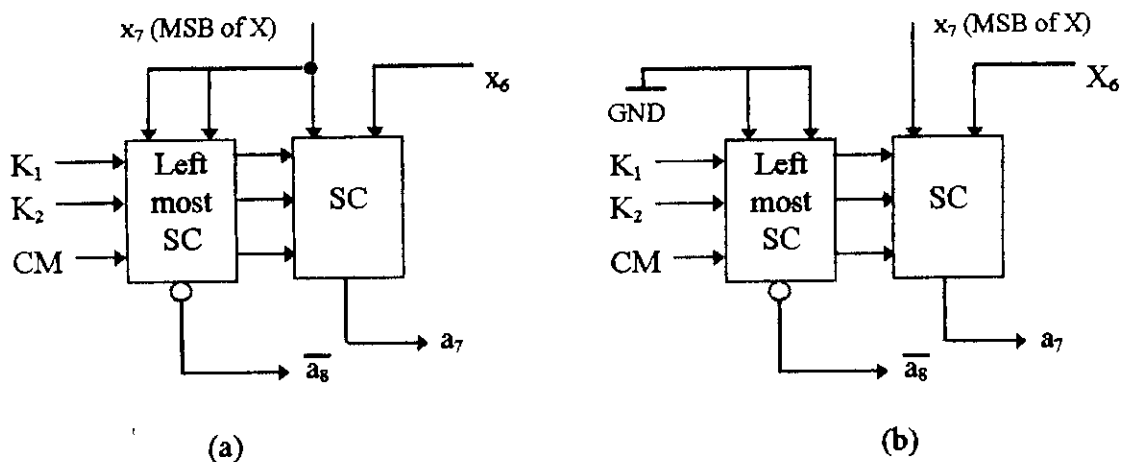
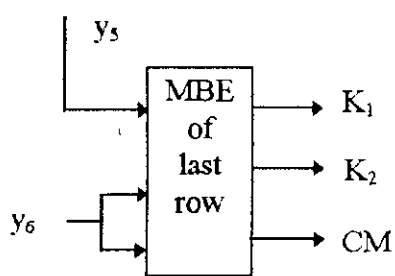


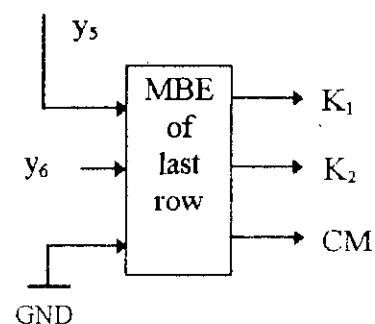
Fig 3.3 (a) Sign extension for signed multiplication as in the original Booth multiplier
(b) Modification to sign extension circuitry for unsigned multiplication

3.3.2 Sign Extension of the Multiplier (Y) with Odd Number of Bits

As mentioned earlier, if the number of bits in the multiplier (Y) is odd then the application of modified Booth algorithm for either signed or unsigned multiplication will result in equal number of partial products. So a modified Booth multiplier architecture, whose multiplier Y has n bits and which is designed for signed multiplication will have the same number of rows of partial products as that of a multiplier architecture for unsigned multiplication provided n is odd. However, for signed multiplication the MSB of Y is extended to turn its number of bits into an even one in order to complete bit-pair recoding. Thus, two of the inputs to the modified Booth encoder (MBE) in the final row of the multiplier array are wired together as shown in Fig. 3.4 (a). In contrast, for unsigned multiplication, a logic zero has to be forced at the most significant bit position of Y. Accordingly the change in hardware required in the MBE of the final row is depicted in Fig. 3.4 (b) for a multiplier (Y) having 7 bits ($y_0 - y_6$). Here y_6 is the MSB of the original multiplier.



(a) Signed Multiplication



(b) Unsigned Multiplication

Fig. 3.4 Sign extension of the Multiplier(Y) having odd number of bits

3.3.3 Sign Extension of the Multiplier (Y) with Even Number of Bits

As discussed in Section 3.2, if the multiplier has even number of bits then extension of the MSB is not required since bit-pair recoding completes at the MSB. Fig. 3.5(a) illustrates how the inputs to the MBE of the final row of the multiplier array are connected when Y has 8 bits with y_7 as the MSB. However, for unsigned multiplication if the multiplier (Y) has even number of bits then the application of modified Booth algorithm will generate an extra partial product compared to that generated for signed multiplication. This is due to the introduction of two extra logic 0s at the most significant bit position of the multiplier as was explained in Section 3.2. An extra modified Booth encoder is required to accomplish this as shown in Fig 3.5(b). Note that the two of the higher input bits of this extra MBE are grounded as required. In order to add the additional partial product generated, an extra row of full-adders is required compared to signed multiplication.

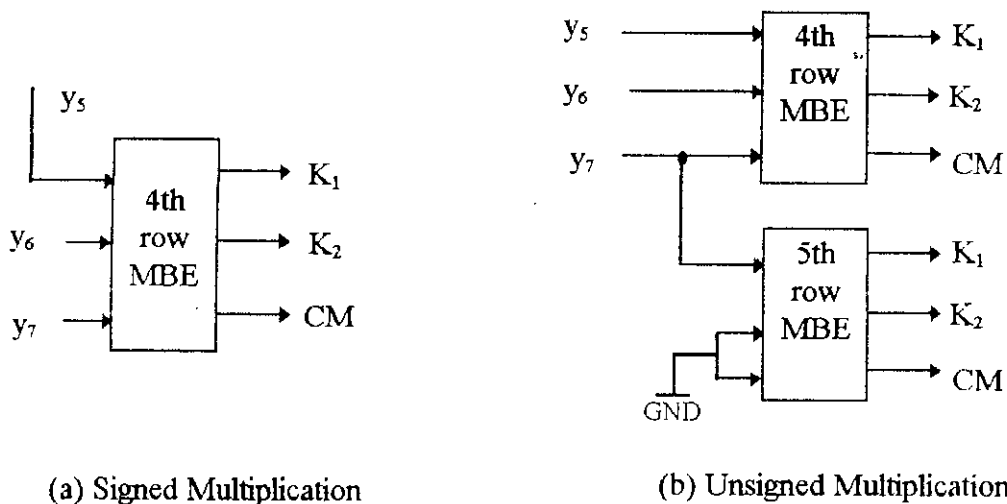


Fig. 3.5 Sign extension of the multiplier(Y) having even number of bits

3.4 Signed and Unsigned Multiplication Using a Single Array

It was shown in the previous section that the modified Booth multiplier can be used for unsigned multiplication with little change in the hardware. In this section, a generalized architecture will be developed which will be capable of both signed and unsigned multiplication. It is clear from the discussion in the previous sections that the type of multiplication performed by the multiplier depends upon the selection of the sign-extension bit for the operands. Hence, a generalized architecture would allow the selection of appropriate sign-extension bit for either of the two modes of multiplication. This can be achieved by the use of a "Mode Select" input to the multiplier.

3.4.1 Selection of Multiplicand Sign Extension Bit

The number of columns in the general multiplier array will not be affected by whether the multiplicand (X) have odd or even numbers of bits. A 2/1 multiplexer is used to select the proper sign-extension bit for signed or unsigned multiplication. This multiplexer is called "Mode Selector" (MS) and is shown in Fig. 3.6. One of the inputs of this MUX is connected to the MSB of X while the other input is grounded. The mode of multiplication will be decided according to the Table 3.1

Table 3.1 Operation of the Mode Selector for X

S	Mode	MS Output (Z_1)
0	Signed	MSB of X
1	Unsigned	0

As shown in Fig. 3.6 the output Z_1 of the Mode Selector is connected to both the inputs of the left most selector-complementer (SC). When signed multiplication is desired S should be made zero and MSB of X will be selected by the MUX as the sign-extension bit. On the other hand when S is made logic 1, a zero will be selected instead of the multiplicand MSB and unsigned multiplication will be performed.

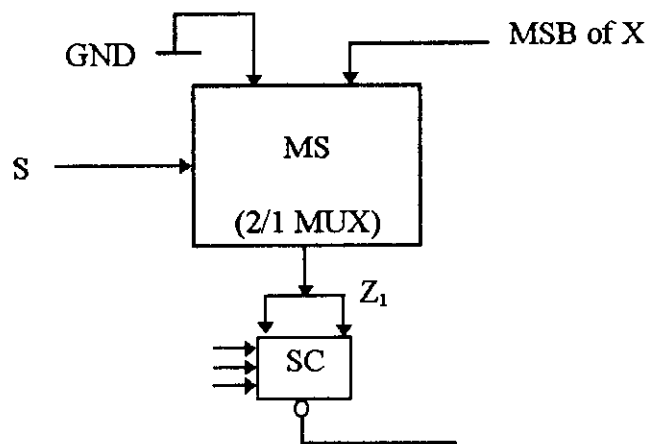


Fig. 3.6 Selection of proper version of MSB of X

3.4.2 Selection of Multiplier Sign Extension Bit

Two different architecture will be developed depending on the number of bits (odd or even) in the multiplier (Y). When multiplier has odd number of bits then the number of partial products generated will be the same for either mode of multiplication. So, the array structure will be the same in either case except that there will now be another Mode Selector for selecting the proper sign-extension bit for the multiplier. The select signal of this MUX is the input S used for the Mode-Selector of the multiplicand MSB. The connectivity of this MUX to the most significant modified Booth encoder (MBE) is shown in Fig. 3.7. One of the inputs to this MUX is the MSB of the multiplier while the other input is grounded. Its output Z_2 together with the MSB of the multiplier is input to the

MBE of the last row. The mode of multiplication is decided according to the Table 3.2. Fig. 3.7 illustrates the arrangement of the Mode Selector and the most-significant MBE for the generalized architecture when the multiplier (Y) has odd number of bits (7 in this case).

Table 3.2 : Operation of the Mode Selector for Y

S	Mode	MS Output (Z_2)
0	Signed	MSB of Y
1	Unsigned	0

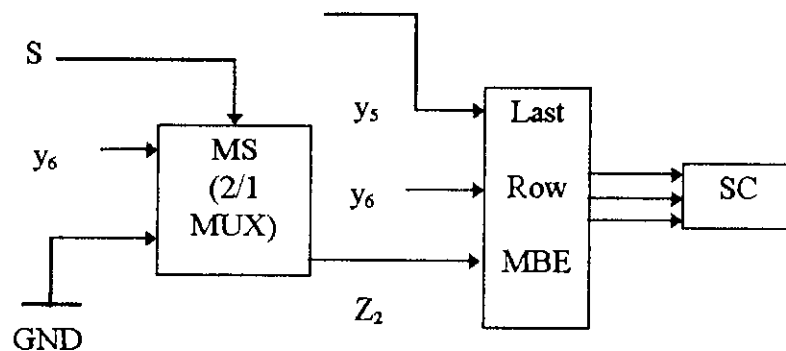


Fig. 3.7 Arrangement of the Mode Selector and MBE when Y has odd number of bits (7 bits in this illustration)

When the multiplier (Y) has even number of bits and unsigned multiplication have to be performed the generalized array must have provision for the generation of one extra row of partial product compared to the original signed multiplier in order to account for sign-extension. This is achieved by connecting a Mode Selector to the most-significant MBE according to the scheme illustrated in Fig. 3.8. Here the multiplier (Y) is assumed to have 8 bits with y_7 as the MSB. When unsigned multiplication is desired, S is made 1 and according to Table 3.2 the output of the MS (Z_2) is 0. So, the MBE of the last row of the array gets the input (Y_7 0 0). However, when signed multiplication is desired, S is made 0 and the output Z_2 of MS is y_7 as per Table 3.2. Thus, the inputs of the most-significant MBE has ($y_7 y_7 y_7$), i.e., either 000 or 111 resulting in a partial product whose bits are all 0s. Therefore, this partial product does not affect the final product output of the multiplier. This is because the final MBE along with the corresponding row of full-adders are redundant for signed multiplication as was explained in Section 3.3.3.

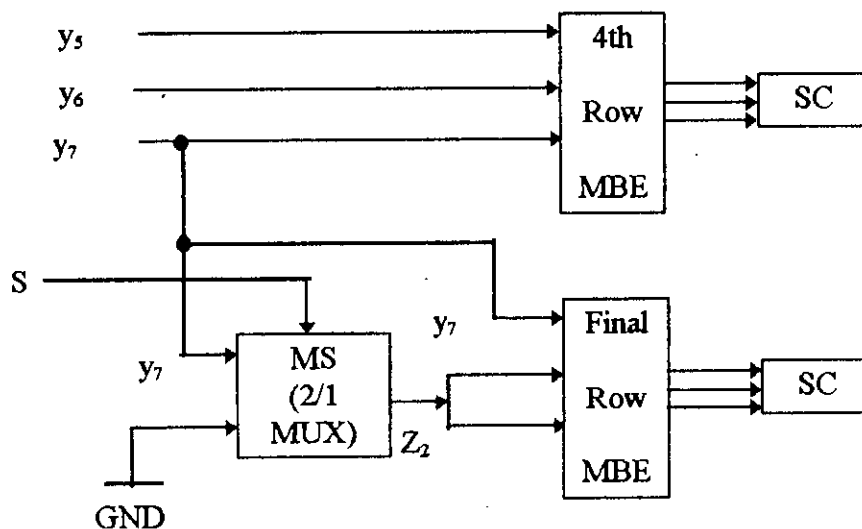


Fig. 3.8 Arrangement of Mode Selector and extra row MBE when Y has even number of bits (8 bits in this illustration)

3.5 The Generalized Architecture

Two generalized multiplier architectures are given in Figures 3.9 and 3.10. The first one is for a 7 by 7 bit multiplier array while the second one is a 6 by 6 bit multiplier.

3.6 Calculation of Overhead

In this article a comparative analysis on the hardware and delay overheads of the generalized multiplier with those of the original modified Booth multiplier will be presented.

3.6.1 Hardware Overhead

Table 3.3 shows the hardware requirement for both the generalized multiplier and the original modified Booth multiplier. Two set of calculations have been carried out based on whether Y has odd or even number of bits. These calculations are based on particular implementations of the full-adder (FA), manchester carry adder (MCA), modified Booth Encoder (MBE), Selector-Complementer (SC) and Mode-Selector (MS) in terms of basic logic gates as shown in Figures 3.11-3.14. It is clear from these figures that each of the FA and MCA has 9 gates, MBE has 8 gates, SC has 6 gates and MS has 3 gates. Inverters are excluded from these calculations. Thus, from Table 3.3, when n is odd there are a total of $[15n^2 + 47n + 14]/2$ gates in the original design. On the other hand the proposed generalized architecture requires only 2 mode-selectors (MS) resulting in an overhead of only 6 gates.

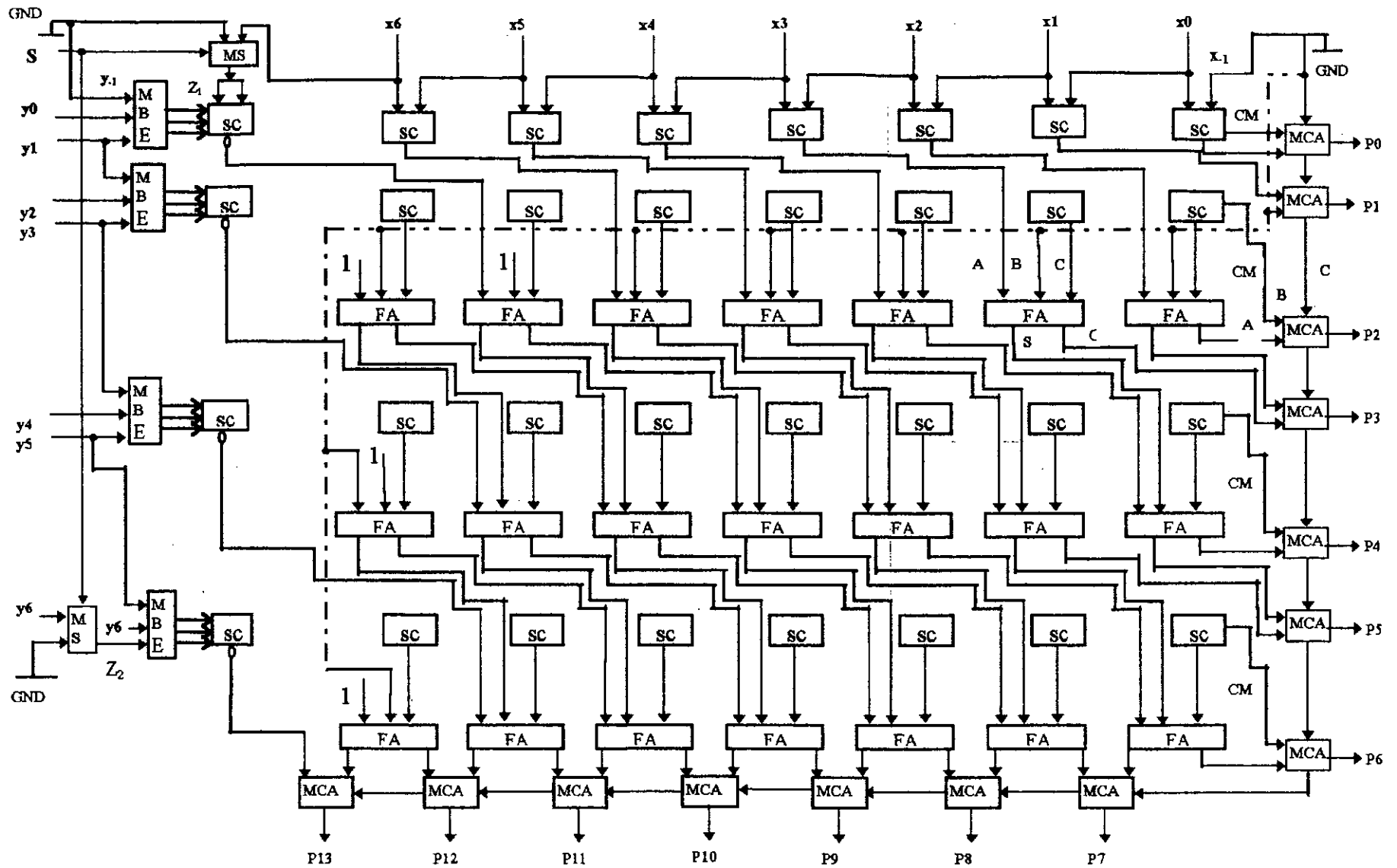


Fig 3.9 : A 7 by 7 bit generalized modified Booth multiplier

S= 0: Signed Multiplication
 S= 1: Unsigned Multiplication

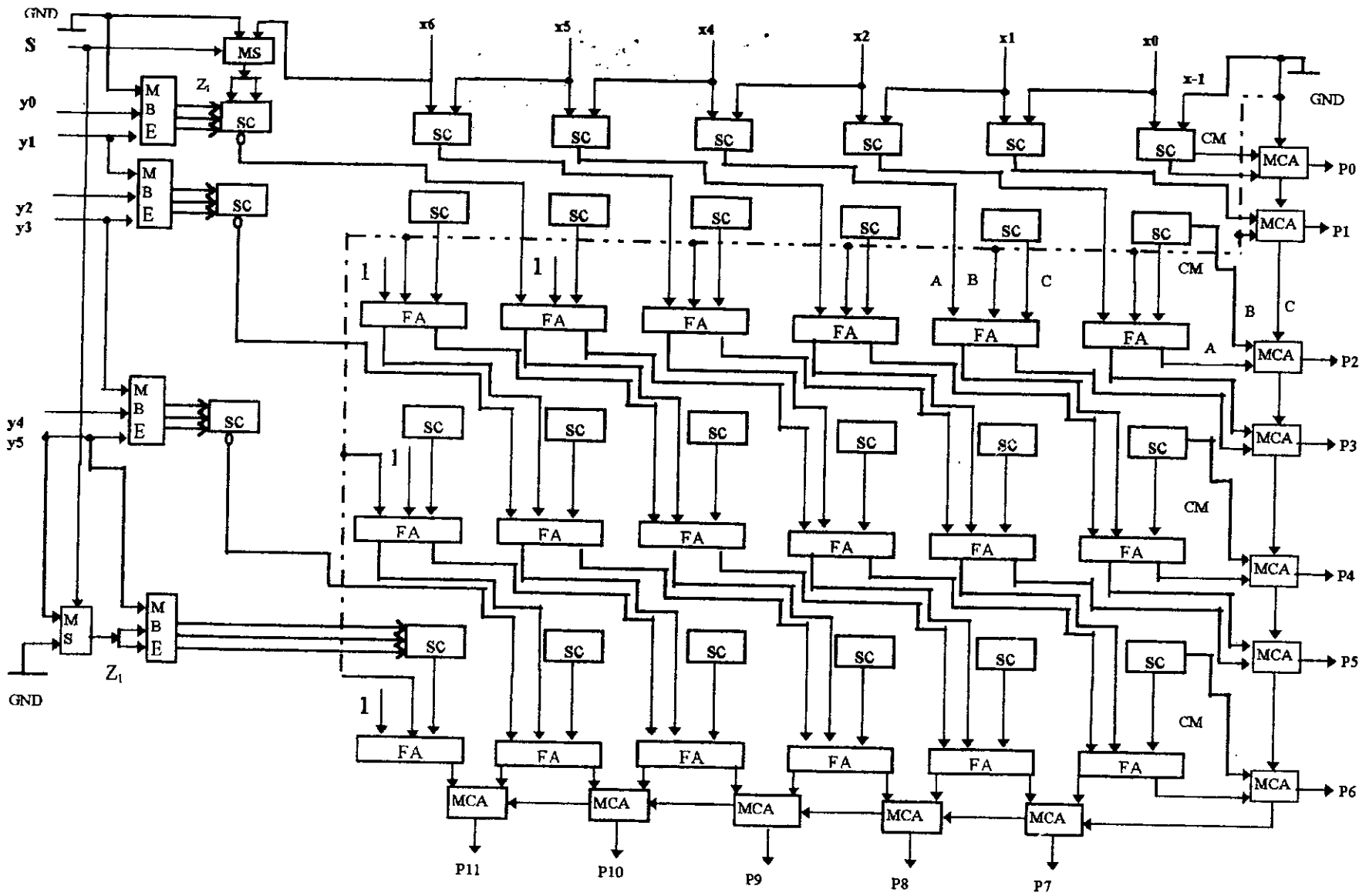
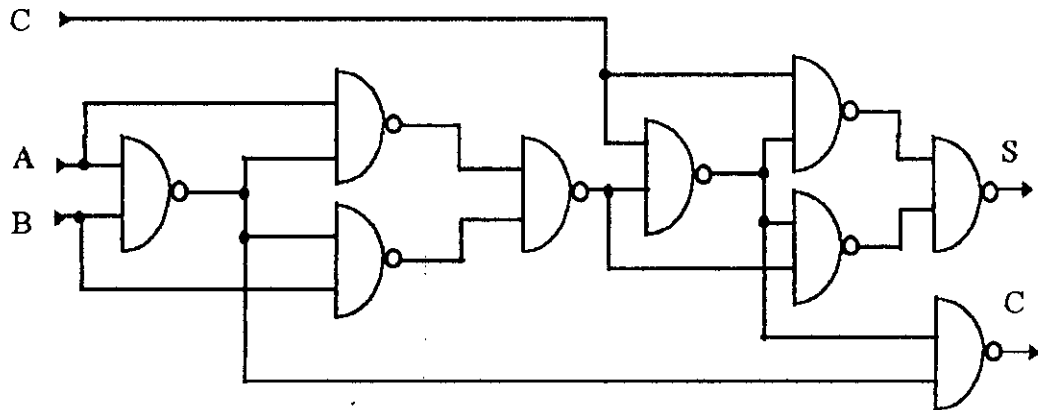


Fig. 3.10 A 6 by 6 bit generalized modified Booth multiplier

S= 0 Singed Multiplication
 S= 1 Unsigned Multiplication



3.11 Gate level design of Full-adder and Manchester carry adder circuits

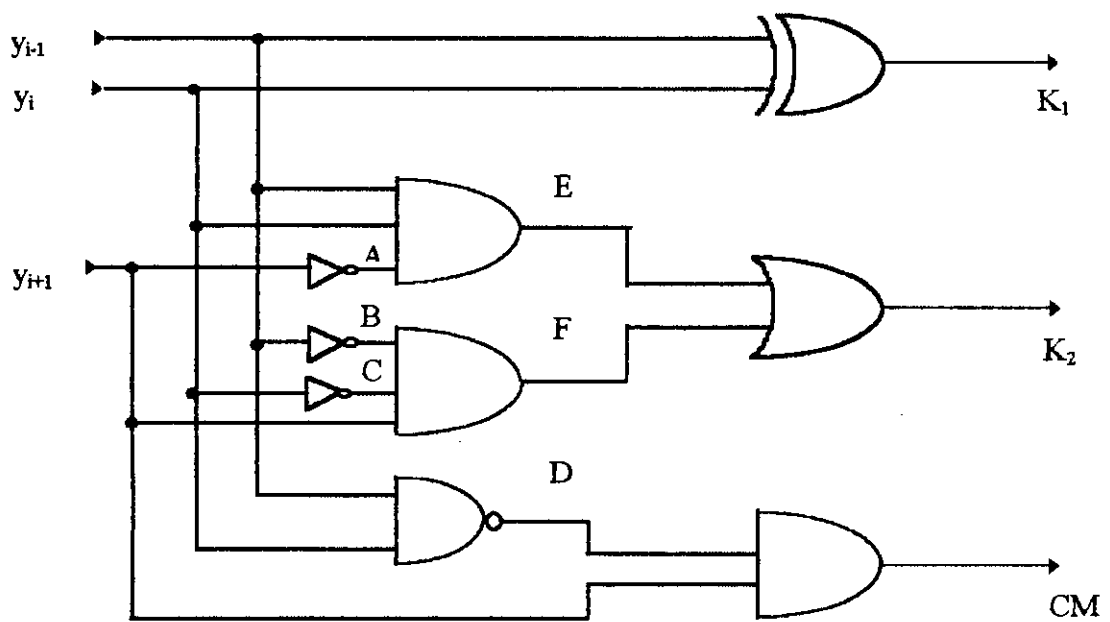


Fig. 3.12 Gate level design of the Modified Booth Encoder (MBE)

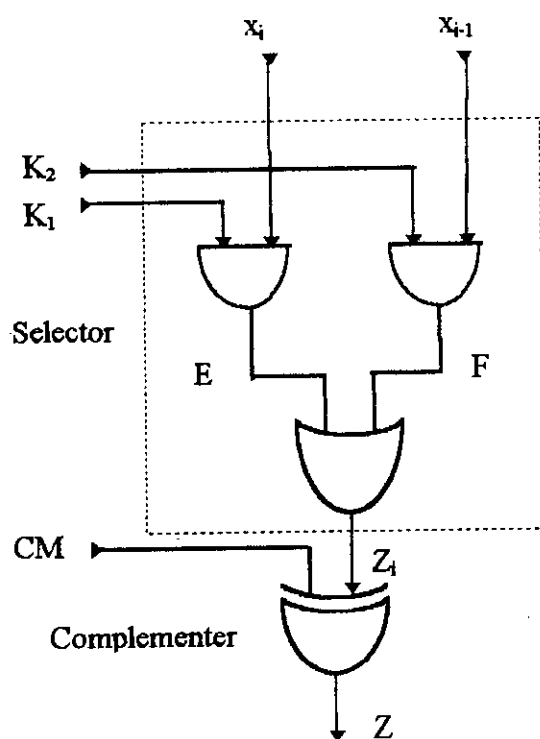


Fig. 3.13 Gate level design of the Selector-Complementer circuit

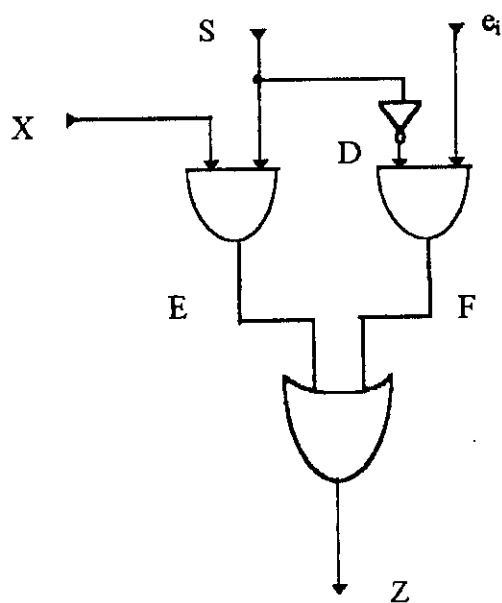


Fig. 3.14 Logic Diagram of Mode-Selector

Table 3.3: Hardware Requirements

Multiplier ($n \times n$)	No. of MBE	No. of SC	No. of FA	No. of MCA	No. of MS
Original Architecture n odd	$(n+1)/2$	$\{(n+1)/2\}(n+1)$	$[\{(n+1)/2\}-1]n$	$2n$	0
Generalized Architecture n odd	$(n+1)/2$	$\{(n+1)/2\}(n+1)$	$[\{(n+1)/2\}-1]n$	$2n$	2
Original Architecture n even	$n/2$	$(n/2)(n+1)$	$\{(n/2)-1\}n$	$2n-1$	0
Generalized Architecture n even	$(n/2+1)$	$\{(n/2+1)(n+1)\}-1$	$(n/2)n$	$2n$	2

Therefore:

$$\text{Hardware Overhead} = \frac{12}{[15n^2 + 47n + 14]} \times 100\% \quad (3.1)$$

Equation 3.1 gives hardware overhead of 1.11317% and 0.29311% for $n=7$ and $n=15$ respectively. This figure continues to decrease as n increases. Thus with large operand size, the hardware overhead is negligible.

Now when n is even, the original design has $[15n^2 + 32n - 18]/2$ basic gates. However, the generalized architecture requires one extra modified Booth encoder, n extra selector-complementers, n extra full-adders, 1 extra Manchester carry adder and 2 extra mode-selectors. In total $(15n+23)$ extra gates are required in the generalized architecture compared to the original signed multiplier.

Therefore,

$$\text{Hardware Overhead} = \frac{30n + 46}{[15n^2 + 32n - 18]} \times 100\% \quad (3.2)$$

This formula is valid as long as n is even. Equation 3.2 gives hardware overhead of 12.1365% and 6.146% for $n=16$ and $n=32$ respectively. Thus a 32 by 32 bit generalized multiplier does not have very large hardware overhead. The overhead reduces further as the operand size increases.

3.6.2 Delay Overhead

In order to calculate the delay overhead it is assumed that a basic logic gate has unit delay except that an inverter is assumed to have zero delay. When n is odd the only delay overhead that occurs in the generalized multiplier over the original one is due to the mode-selector circuit which introduces a delay of 2 time units (see Fig.3.14). However, when n is even the generalized multiplier has one extra row of selector-complementers and full-adders. Besides, there will be some extra delay in the mode-selector circuit. The selector-complementer circuit has a delay of 4 units while the full-adder introduces a delay of 6 time units. Therefore, the total delay overhead is $2 + 4 + 6 = 12$ time units when n is even.

Chapter 4

Testability of the Generalized Multiplier

4.1 Introduction

In general VLSI circuits are very difficult to test for several reasons. The high device-to-pin ratio severely limits the controllability and observability of internal signal lines in VLSI chip [14]. Also, there exists a large number of faults of various types, many of which cannot be modelled by the traditional stuck-at fault model. Test pattern generation and verification procedures are becoming very costly or even computationally infeasible to implement [15]. However, VLSI circuits like array multipliers having regular iterative structure have been shown to be easily testable by slight modification of the conventional design [16]. In this chapter the generalized multiplier architecture presented in Chapter 3 will be modified in order to convert it to an easily testable one.

4.2 Testing Approach

The objective of the testing approach adopted in this research is to exhaustively test the full-adders (FAs), Manchester carry adders (MCAs) and modified Booth encoders

(MBEs). Such a test set will be applicable to any arbitrary logic implementation of these cells. The fault model used in this research assumes:

- a) in an array multiplier, at most one basic cell is faulty at a time;
- b) the fault is a permanent fault (i.e. the fault permanently changes the circuit's logic characteristics);
- c) the fault may alter the cell's output functions in any arbitrary way, as long as the faulty cell remains combinational circuit.

In order to generate exhaustive test set for the selector-complementers (SCs) it is necessary to modify the design of the modified Booth encoders with a significant increase in complexity and gate count. However, Takach and Jha [18] have shown that hardware overhead reduction is possible for array multipliers if a fault model based on single (stuck at) faults is used instead of the single cell fault model. They have also shown that a set of test vectors which detect all single stuck-at faults in a gate level carry-save multiplier can be readily adopted to detect all detectable single stuck-at, transistor stuck-on and stuck-open faults in a DCVS implementation of the multiplier. Therefore, the selector-complementers will be tested for single stuck-at faults only. Moreover, although MBEs are eventually exhaustively tested, this testing does not guarantee the fault propagation to the primary outputs of the array. Due to this, equivalent gate level circuit for MBE and also mode-selector (MS) will be tested for single stuck-at faults.

4.3 Modification of the Architecture for Testability

The main challenge in testing array multipliers is the difficulty of controlling the inputs of internal adder cells from the primary inputs, namely the multiplier (Y) and multiplicand (X) inputs. In fact, some patterns cannot be applied to some adders cells. To

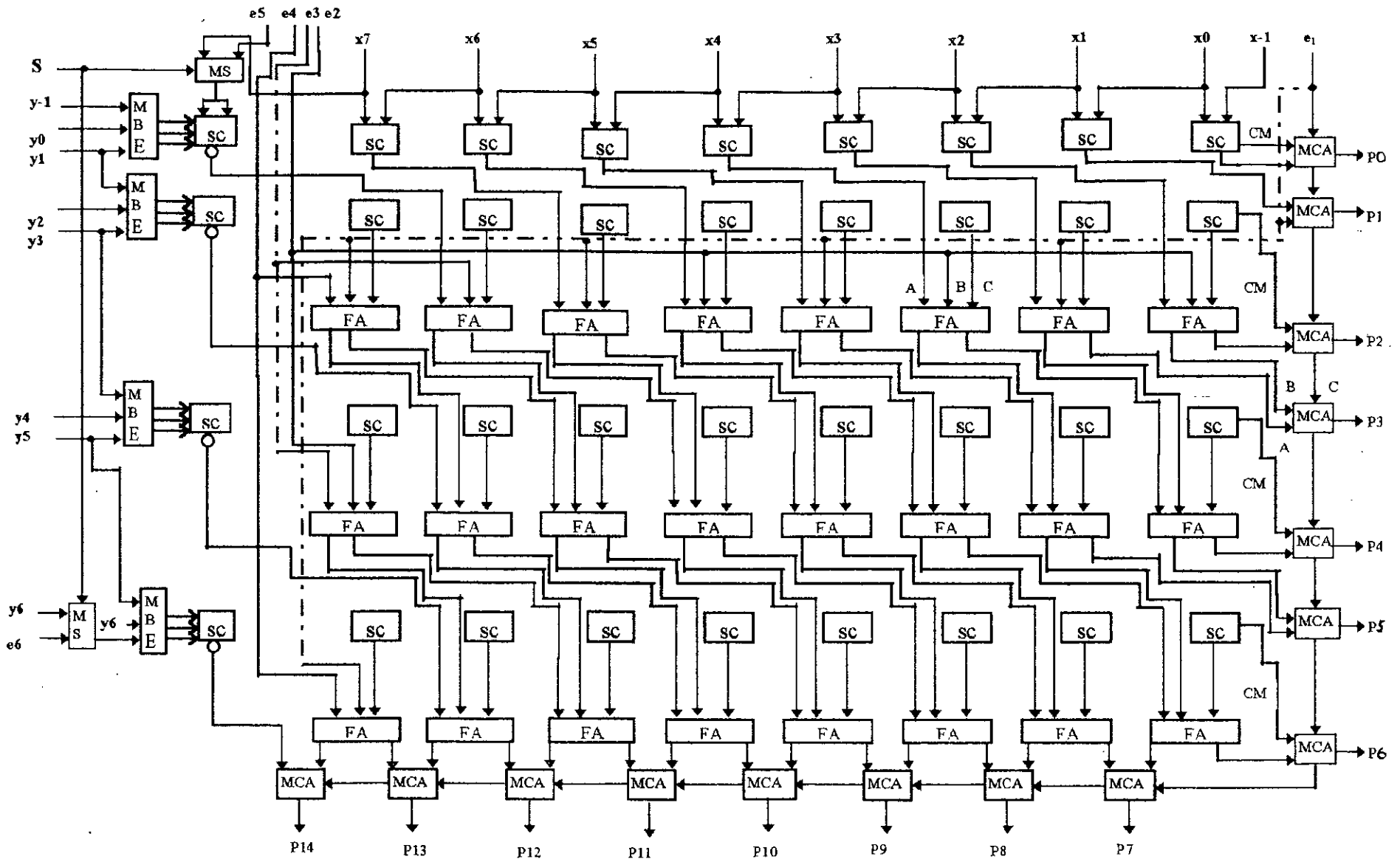


Fig 4.1: An 8 by 7 bit generalized testable modified Booth multiplier

circumvent this problem, extra inputs and sometimes extra hardware is added to enhance controllability and observability of the internal signal lines in a VLSI circuits.

A testable architecture for an 8×7 bit generalized (signed and unsigned) multiplier is shown in Fig. 4.1. Comparing to its non testable version, this architecture has 8 extra controllable inputs $e_1, e_2, e_3, e_4, e_5, e_6, x_{-1}$ and y_{-1} to enhance the controllability of various cells. For normal multiplication operation these extra inputs will have the following logic values: $e_1 = 0, e_2 = 0, e_3 = 1, e_4 = 1, e_5 = 0, e_6 = 0, x_{-1} = 0$ and $y_{-1} = 0$

4.4 Testing the Individual Cells

The patterns required for testing the various individual cells of the multiplier for single stuck-at faults are derived in this section.

4.4.1 Testing of MBEs for Single Stuck-at Fault

The logic diagram of the modified Booth encoder used in the generalized multiplier was shown in Fig. 3.12. This is repeated here in Fig. 4.2 for convenience.

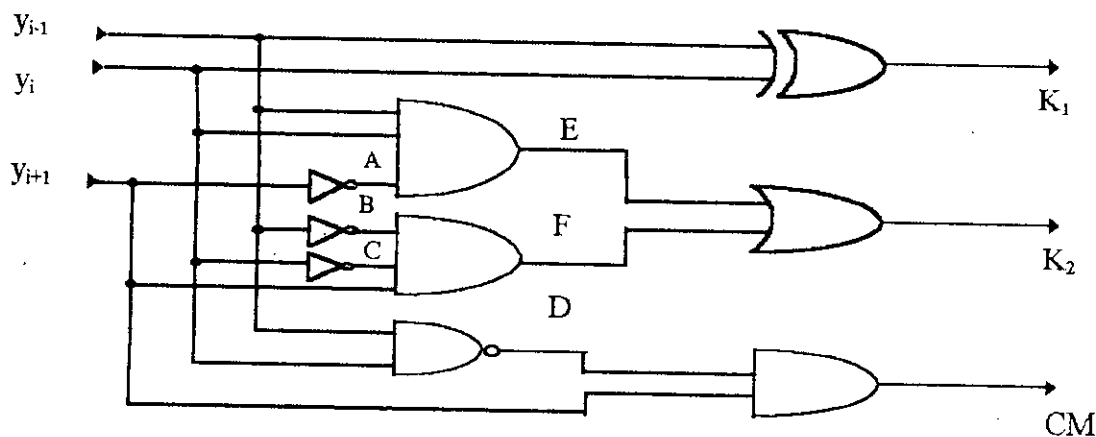


Figure: 4.2 Gate level design of the Modified Booth Encoder (MBE)

The circuit has twelve nodes and so twenty four possible stuck-at faults. For the three primary inputs there will be eight possible test vectors which will be identified as t_0 to t_7 , where the suffix is the decimal equivalent of the binary numbers $(y_{i-1}y_iy_{i+1})$. The fault coverage is conveniently displayed in the fault-matrix shown in Table 4.1. The tick against each test indicates the fault covered by that test.

Table 4.1 Fault matrix for the MBE logic circuit

Test	y_{i-1}	y_{i-1}	y_i	y_i	y_{i+1}	y_{i+1}	A	A	B	B	C	C	D	D	E	E	F	F	K_1	K_1	K_2	K_2	CM	CM
	/0	/1	/0	/1	/1	/1	/0	/1	/0	/1	/0	/1	/0	/1	/0	/1	/0	/1	/0	/1	/0	/1	/0	/1
t_0		✓		✓		✓										✓		✓		✓		✓		✓
t_1		✓		✓	✓				✓		✓		✓				✓			✓	✓			✓
t_2		✓	✓			✓										✓		✓	✓			✓		✓
t_3		✓	✓		✓						✓	✓				✓		✓	✓			✓	✓	
t_4	✓			✓		✓			✓							✓		✓	✓					✓
t_5	✓			✓	✓				✓			✓				✓		✓	✓			✓	✓	
t_6	✓		✓				✓								✓					✓	✓			✓
t_7	✓		✓					✓					✓		✓		✓			✓	✓			✓

From the above fault matrix it is seen that the test vectors t_1 , t_3 , t_6 and t_7 are the essential tests. These four essential tests cover all the faults except $y_{i+1}/1$ and $B/1$. A single test that covers both of these faults is t_4 . Hence a set of test patterns for the inputs $(y_{i-1}y_iy_{i+1})$ of the MBE of Fig. 4.2 that detect any single stuck-at fault in the MBE is $\{001, 011, 100, 110, 111\}$.

4.4.2 Testing of the SCs for Single Stuck-at Fault

The logic diagram of the selector-complementer block is repeated in Fig. 4.3. It has a total of five inputs. However, for testing of single stuck-at fault we will derive fault matrix for only the selector part. This is because the complementer part is nothing but an EX-OR gate whose one input is the complement signal CM and the other is output of selector circuit Z_i . Since output of an EX-OR gate inverts due to inversion of any one of its input so if we can test only the selector part for single stuck-at fault we may declare that this fault will propagate to the SC output due to that fault propagation property of EX-OR gate. This criterion will also reduce the number of input test vector of SC blocks from twenty five to sixteen. These will be identified as t_0 to t_{15} , where the suffix is the decimal equivalent of the binary number $(K_1 x_i K_2 x_{i-1})$. Table 4.2 shows the fault matrix .

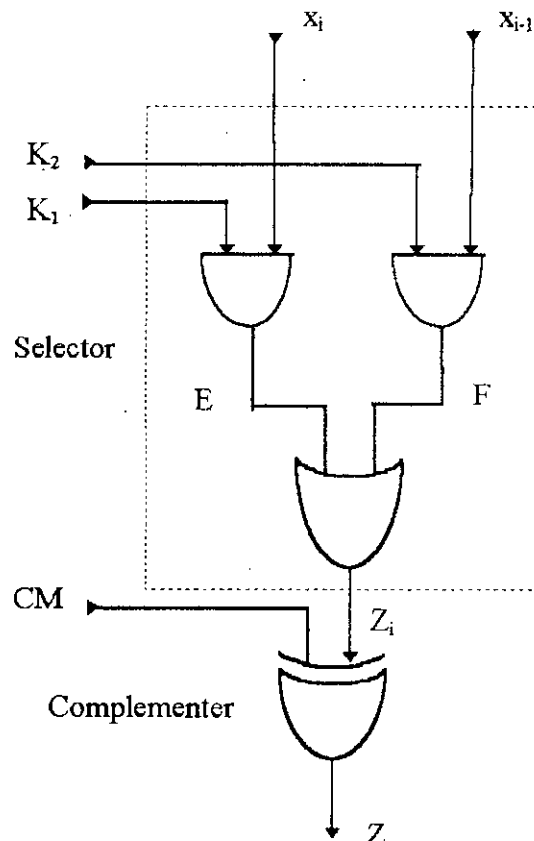


Fig. 4.3 Gate Level Design of the Selector-Complementer Block

Table 4.2 Fault Matrix for the Selector Block

Test	$K_1/0$	$K_1/1$	$K_2/0$	$K_2/1$	$x_i/0$	$x_i/1$	$x_{i-1}/0$	$x_{i-1}/1$	E/0	E/1	F/0	F/1	$Z_i/0$	$Z_i/1$
t_0										✓		✓		✓
t_1				✓						✓		✓		✓
t_2								✓		✓		✓		✓
t_3			✓				✓			✓	✓		✓	
t_4		✓								✓		✓		✓
t_5		✓		✓						✓		✓		✓
t_6		✓						✓		✓	✓	✓		✓
t_7		✓	✓				✓			✓			✓	
t_8						✓				✓		✓		✓
t_9				✓		✓				✓		✓		✓
t_{10}						✓		✓		✓		✓		✓
t_{11}			✓			✓	✓			✓	✓		✓	
t_{12}	✓				✓				✓			✓	✓	
t_{13}	✓			✓	✓				✓			✓	✓	
t_{14}	✓				✓			✓	✓			✓	✓	
t_{15}	✓		✓		✓		✓		✓		✓		✓	

Identifying the indistinguishable faults and dominant faults in the fault matrix of Table 4.2, it is found that the test vectors needed to test any single stuck-at fault are t_2 , t_7 , t_8 , and t_{13} . So the set of test pattern for the inputs $(K_1x_iK_2x_{i-1})$ of the selector circuit to detect any single stuck-at fault is $\{0010, 0111, 1000, 1101\}$.

4.4.3 Testing of the Mode-Selector for Single Stuck-at Fault

The logic circuit for the mode-selector block is shown in Fig. 4.4. It has three inputs X , S and e_i so there are eight possible input test vectors (t_0 - t_7) in the fault matrix. The circuit has seven nodes giving rise to a total of fourteen stuck-at faults. The fault matrix is shown in Table 4.3.

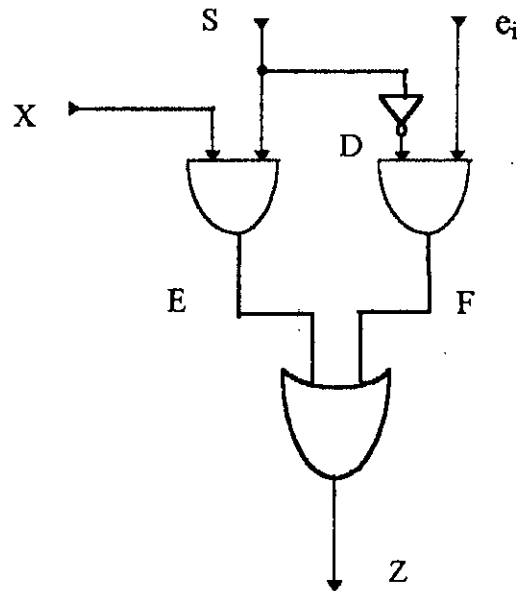


Fig. 4.4 Logic diagram of Mode-Selector block

From the fault matrix it is found that t_6 is the essential test. Using the concept of indistinguishable faults and fault dominance it is found that test vectors required to test any single stuck-at fault in the MS circuit are t_0 , t_4 , t_6 and t_7 . So the set of test patterns for the inputs ($X S e_i$) of the MS is $\{000, 100, 110, 111\}$.

Table 4.3 Fault matrix for the MS

test	X/0	X/1	S/0	S/1	e _i /0	e _i /1	D/0	D/1	E/0	E/1	F/0	F/1	Z/0	Z/1
t ₀		√								√		√		√
t ₁		√		√						√		√		√
t ₂						√				√		√		√
t ₃			√		√						√		√	
t ₄	√			√			√		√				√	
t ₅	√						√		√				√	
t ₆			√			√		√		√		√		√
t ₇					√						√		√	

4.5 Testing the Multiplier

A set of test vectors for testing the multiplier will be derived in this section. The vectors will cover the exhaustive testing of the FAs, MCAs, MBEs as well as the stuck-at faults in the selector-complementers and mode-selectors.

4.5.1 Test Vectors

Table 4.4 shows a set of test vectors for detecting all single stuck-at faults in the multiplier of Fig. 4.1. An 8-bit multiplicand X and a 7-bit multiplier Y are shown with their LSBs to the right most position. The underlined bits have to be replicated for generating the test vectors for multipliers with larger operand wordlengths.

Table 4.4 A set of test vectors for a 8×7 bit generalized multiplier

Vectors	X	x_{-1}	Y	y_{-1}	S	$e_4e_3e_2e_1$	e_6e_5
t ₁	0000 0000	0	000 0000	0	0	0000	xx
t ₂	1000 0000	0	101 0101	0	1	0000	01
t ₃	1111 1111	1	101 0101	0	1	1111	01
t ₄	1000 0000	0	010 1010	1	1	1111	11
t ₅	0000 0000	0	111 1111	0	0	1100	xx
t ₆	0000 0000	0	011 0011	1	0	0011	xx
t ₇	1111 1111	1	101 0011	0	1	0011	01
t ₈	1111 1111	1	100 0100	1	1	1100	01
t ₉	0101 0101	0	100 1100	1	0	0110	xx
t ₁₀	01010101	0	011 0011	0	0	1001	xx
t ₁₁	1111 1111	1	011 0011	0	0	0011	xx
t ₁₂	1111 1111	1	100 1100	1	0	1100	xx
t ₁₃	0000 0000	0	010 1010	1	1	1100	10
t ₁₄	1111 1111	1	010 1010	1	1	0000	11
t ₁₅	111 11111	0	001 1001	1	1	1111	10
t ₁₆	111 11111	0	110 0110	0	1	0000	01
t ₁₇	0000 0000	0	110 0110	0	1	1111	00
t ₁₈	1111 1111	1	001 1001	1	1	1001	11
t ₁₉	1111 1111	1	111 1111	1	0	0000	xx

4.5.2 Exhaustive Testing of the Full-Adders

The first twelve test vectors t_1 - t_{12} of Table 4.4 set up the patterns required for exhaustive testing of all the full-adders as explained in the following steps:

1) The test vector t_1 applies 000 to most of the full-adders. However, the full adders affected by the inverted sign bits of the partial products receive 100. Test vector t_2 applies 000 to these full-adders.

2) Application of pattern 111 to all the full-adders is accomplished with the vectors t_3 and t_4 .

3) The vector t_5 applies 100 to all the full-adders except the one labeled 'FA1' in the second row of Fig. 4.1 which receives the pattern 010. t_6 applies 100 to FA1.

4) The vector t_7 applies 011 to all the full-adders except the one labeled 'FA1' which receives the pattern 101. t_8 applies 011 to FA1.

5) t_9 applies 001 and 110 to alternate full-adders, t_{10} applies 110 and 001 to alternate full-adders in each row.

6) t_6 applies 010 to the full-adders in the even rows except FA1. It was seen in step 3 that FA1 gets 010 by t_5 . Application of 010 to the full-adders in the odd rows is accomplished with the test vector t_{11} .

7) t_8 applies 101 to the full-adders in the even rows except FA1. It was seen in step 4 that FA1 gets 101 by t_7 . Application of 101 to the full-adders in the odd rows is accomplished with the test vector t_{12} .

Table 4.5 shows the results of exhaustive testing of the full-adders.

Table 4.5 Exhaustive testing of the FAs

Pattern applied to FAs	Test vector required
000	t ₁ , t ₂
111	t ₃ , t ₄
100	t ₅ , t ₆
011	t ₇ , t ₈
001	t ₉ , t ₁₀
110	t ₉ , t ₁₀
010	t ₅ , t ₆ , t ₁₁
101	t ₇ , t ₈ , t ₁₂

Now let us consider how the effect(s) of a fault in a full-adder is transmitted to the primary outputs (observable outputs) of the multiplier. The sum output of a full-adder is the parity (EX-OR) of the three input bits. Therefore, if one of these three inputs is inverted due to appearance of a faulty signal then the sum output of the full-adder is also inverted. The carry output may or may not be inverted depending on the logic values of the other two inputs. This is also true for manchester carry adder (MCA), because it realizes the same logic function as a full-adder. Since all the full-adders are exhaustively tested, the effect of a fault in a full-adder is transmitted to its output(s). The two outputs of each full-adder of Fig. 4.1 are connected to the primary outputs of the multiplier through two

different chains of three input EX-OR gates (of FAs and final MCAs). Hence the effect of a fault in a full-adder is transmitted to the observable output(s).

4.5.3 Exhaustive Testing of the Manchester Carry Adders

All the manchester carry adders are exhaustively tested using a subset of test vectors from Table 4.4. The combinations of test vectors that apply various patterns to all the manchester carry adders are listed in Table 4.6. The effect of a fault in any MCA is transmitted to it's sum output which is a primary output of the multiplier.

Table 4.6 Exhaustive testing of the MCAs

Pattern applied to MCAs	Test vector required
000	t_1, t_2
111	t_3, t_4
101	t_3, t_5, t_7
011	t_7, t_{13}
010	t_6, t_8, t_{14}
100	t_6, t_8
001	$t_9, t_{10}, t_{11}, t_{12}, t_{15}, t_{16}$
110	$t_9, t_{10}, t_{11}, t_{12}, t_{15}, t_{16}$

4.5.4 Testing of the Modified Booth Encoders

The modified Booth encoders are tested in two ways. First they are exhaustively tested. However unlike the FAs and MCAs, exhaustive testing of the MBEs does not necessarily guarantee the transmission of the effect of a fault in an MBE to the primary outputs of the multiplier. That is why the MBE block is also tested for single stuck-at faults.

4.5.4.1 Exhaustive Testing

The modified Booth Encoders are exhaustively tested by the vectors shown in Table 4.7.

Table 4.7 Exhaustive testing of the MBEs

Pattern applied to MBEs	Test vector required
000	t_1
010	t_2
101	t_4
111	t_5, t_6
011	t_{11}, t_{12}
100	t_{11}, t_{12}
001	t_{15}, t_{16}
110	t_{15}, t_{16}

4.5.4.2 Testing for Single Stuck-at Fault

Unlike the full-adders and manchester carry adders, exhaustive testing of the MBEs does not necessarily guarantee the transmission of the effect of a fault in an MBE to the primary output of the multiplier. Fault propagation depends on the type of fault and its effect on the output of the MBE. Also, note from Figures 4.1 and 4.2 that the output of an MBE in any row, namely CM , K_1 , and K_2 are inputs to the selector-complementers (SC) in that row (fan-out nodes). In the rest of this sub-section, the test vectors which propagates any single stuck-at fault in an MBE to the primary outputs of the multiplier will be derived.

It was shown in Section 4.4.1 that a set of test patterns for the inputs $(y_{i-1}y_i y_{i+1})$ of the MBE of Fig. 4.2 that detect any single stuck-at fault in the MBE is $\{001, 011, 100, 110, 111\}$. Note from Fig. 4.2 that every input to the MBE has a fan-out of three. This means that the effect of a fault at one of the input nodes might propagates to more than one output of the faulty MBE. Because of the fan-outs at the outputs of the MBE these faulty signals from MBE might propagate through two different paths, i.e. the SC and the chains of carry-save adders, and then reconverge at the final adders (MCAs). Also, note from Fig. 4.1 that the adjacent MBEs share one multiplier bit. Therefore, a fault on one of the shared multiplier bits might affect both the MBE sharing that bit resulting in transmission of the fault through both the MBEs and subsequent reconvergence in the array of FAs and MCAs. It can be verified that because of these reasons some path sensitive patterns (mentioned above) applied to the MBEs for detecting some single stuck-at faults result in negative reconvergence [15] of the fault effects unless these patterns are accompanied by application of appropriate patterns to the multiplicand. It was extensively verified in this research that the test vectors t_{11} , t_{12} , t_{15} , t_{16} and t_{19} apply all the necessary patterns to all the MBEs along with appropriate multiplicand patterns so that any single stuck-at fault in the MBE is propagated to the primary outputs of the multiplier.

4.5.5 Test Vectors for SCs

As mentioned in Section 4.4.2, a set of test patterns for the inputs ($K_1x_iK_2x_{i-1}$) of the selector block of Fig. 4.3 that detect any single stuck-at fault in the selector is {0010, 0111, 1000, 1101}. The selector in any row have two common input nodes, namely K_1 , and K_2 (fan-out nodes). It is verified that the vectors t_2 , t_3 , t_5 , t_{17} and t_{18} sensitize the single stuck-at faults at these nodes and propagate them to the output of the selectors. Each of this selector output is passed through a complementer (an EX-OR gate) whose other input is the MBE output CM (complement signal). The effect of a single stuck-at fault at one of the multiplicand bits is propagated through the outputs of the SCs in that column to one of the inputs of the full-adders in that column. Now the sum output or both the sum and carry outputs of a full-adder are inverted due to a faulty input signal. Each output of every full-adder is connected to the primary outputs of the multiplier through a chain of 3-input EX-OR circuits (of FAs and final MCAs). Thus, these faults are essentially propagated to the primary outputs of the multiplier.

4.5.6 Test Vectors for MSs

As shown in article 4.4.3, a set of test patterns for the inputs ($X S e_j$) of the mode selector (MSs) block of Fig. 4.4 that detect any single stuck-at fault in the mode selector is {000, 100, 110, 111}. It is verified that the vectors t_0 , t_2 , t_5 , t_6 , t_{15} and t_{16} sensitize the single stuck-at faults at these nodes and propagate them to the output of the mode selectors.

4.6 Calculation of Overhead

4.6.1 Hardware Overhead

The testable design of the multiplier presented earlier does not require any extra logic compared to the original design presented in Chapter 3. However, the testable version requires eight extra inputs which increases the number of input pins of the multiplier chip. For a large multiplier, e.g., 32×32 bit multiplier the penalty in terms of extra pins will not be as severe as for a multiplier of small operand wordlengths, e.g., 8×8 bit. The lines carrying the above extra signals will increase the silicon area of the testable multiplier chip compared to the non-testable design.

4.6.2 Delay Overhead

Compared to the non-testable design, the testable multiplier will not have an extra logic gate delay. However, there will be some additional delay due to the extra wiring capacitances associated with various connected to the extra input.

4.7 Difficulties with Even Number of Multiplier Bits

It was discussed in Chapter 3 that when the multiplier (Y) has even number of bits then the generalized architecture needs hardware provision for one extra row of partial product. This extra row is only needed when unsigned mode of multiplication is desired. However, a problem arises during designing a testable generalized architecture whose multiplier has even number of bits. The partial view of the last row of the multiplier architecture for even number of multiplier (Y) bits is shown in Fig. 4.5. From the figure it is seen that application of all possible combination of inputs to the last row modified Booth encoder (MBE) is difficult. This particular problem arises due to the fact that two of the three inputs to the last row MBE are same and comes from mode selector (MS) block. Hence, only four possible combination of inputs namely 000, 100, 011 and 111 can be applied to this particular MBE. Therefore, with this configuration exhaustive testing of the last row full-adders and MBE is not possible.

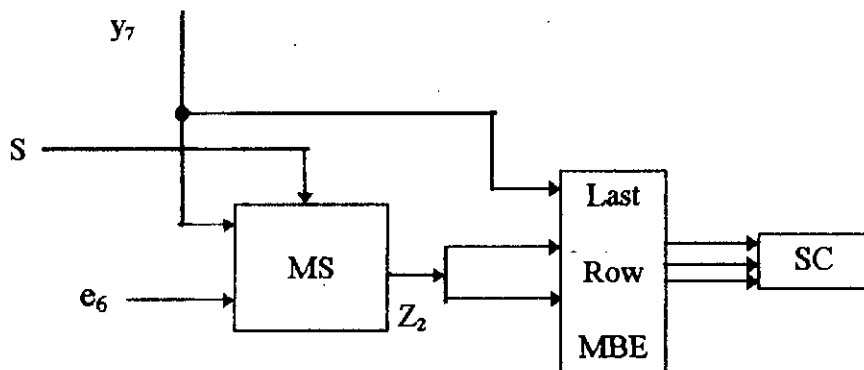


Fig. 4.5 Arrangement of MS and extra row MBE when Y got even number of bits (8 bits)

4.8 Difficulties With Odd Number of Multiplicand Bits

It was shown in Section 4.5 that the generalized multiplier architecture is fully testable as long as the multiplicand (X) has even number of bits and the multiplier (Y) has odd number of bits. In this case it was shown that a total of 19 test vectors were needed to detect all single stuck-at faults. However, this is not true when the number of bits in the multiplicand is odd. Unlike the previous case, it is found that no pair of test vector exists which can apply the patterns 001 and 110 to all alternate full-adders in any row of the array. For example, when the vector pair t_9 and t_{10} Table 4.4 is applied, most full-adders in the second row receives 001 and 110 alternately except the two most-significant full-adders. As a result, not all the full-adders in the subsequent rows receive alternate 001 and 110. The number of full-adders that do not receive the above patterns by the application of vector pair t_9 and t_{10} increases by the number 2 in the subsequent rows. This is illustrated in Fig. 4.6.

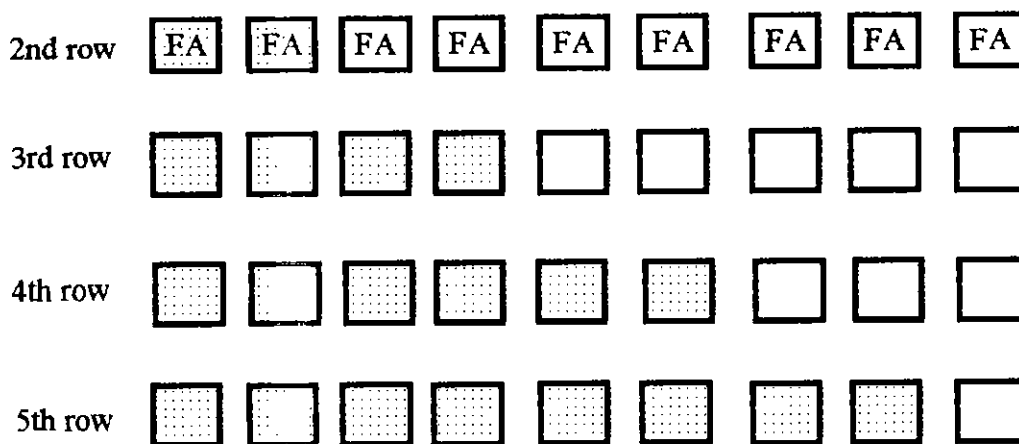


Fig. 4.6 Effect of vector pair t_9 and t_{10} applied to a 9×9 multiplier. The shaded FAs do not receive the patterns 001 and 110 alternately (only the FAs are shown for clarity)

In order to apply the patterns 001 and 110 to all the full-adders four extra test vectors t_{20} - t_{23} have been derived. These four extra vectors, shown in the Table 4.8, replaces the original vectors t_9 and t_{10} of Table 4.4 for the even multiplicand case.

Table 4.8 Extra test vectors for a 9×9 bit multiplier

Test Vectors	X (9-bit)	X_{-1}	Y (9-bit)	Y_{-1}	S	$e_4e_3e_2e_1$	e_6e_5
t_{20}	10101 0101	0	10011 0011	0	1	0001	10
t_{21}	10101 0101	0	01100 1100	1	1	1110	00
t_{22}	01010 1010	1	01100 1100	1	0	0110	xx
t_{23}	01010 1010	1	10011 0011	0	0	1001	xx

The vector pair t_{20} and t_{21} applies 001 and 110 to some of the alternate full-adders in each row in a manner shown in Fig. 4.7. The shaded FAs do not receive the above patterns.

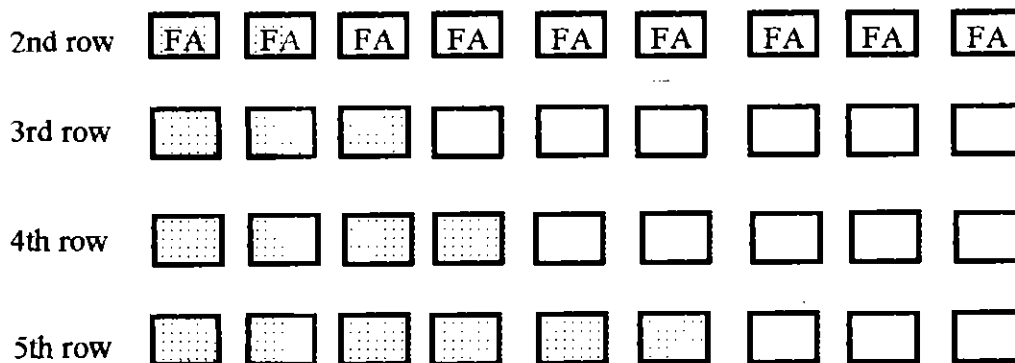


Fig. 4.7 Effect of vector pair t_{20} and t_{21} applied to a 9 by 9 multiplier. The shaded FAs do not receive the patterns 001 and 110 alternately.

It is clear from Fig. 4.7 that this vector pair (t_{20} , t_{21}) is not able to apply the appropriate patterns (001 and 110) to all the alternate full-adders in each row. However, this time the number of full-adders that do not receive the appropriate patterns by the application of vector pair t_{20} and t_{21} grows by the number 1 in the subsequent rows until the 5th row where the number increases by 2.

The vector pair t_{22} and t_{23} applies 001 and 110 only to those alternate full-adders in each row which do not receive these patterns by the application of vectors t_{20} and t_{21} . Fig 4.8 illustrates the results of application of t_{22} and t_{23} . It is seen that the hatched marked full-adder (4th from the LSB side of the 5th row) still does not receive the patterns 110 or 001 by application of any of the vectors t_{20} - t_{23} . Hence, it is clear that these four new vectors can not guarantee that all the full-adders in a multiplier array of any larger size will receive these two patterns, since the problem recurs in the 5th row when the number of bits in the multiplier (Y) is higher than eight. One possible solution to this particular problem is to arrange additional inputs so that the desired patterns (001 and 110) can be applied to all the full-adders using only two test vectors.

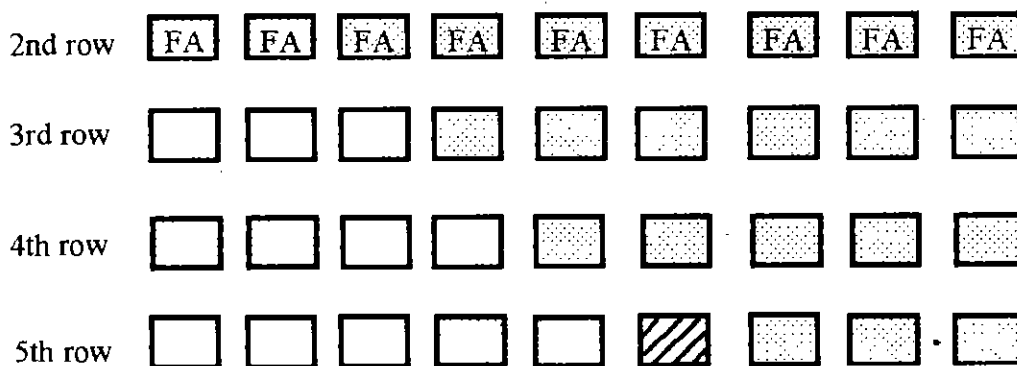


Fig. 4.8 Effect of vector pair t_{22} and t_{23} applied to a 9×9 multiplier. The hatched FA of 5th row still does not receive the desired patterns.

4.9 Summary

The testable design of the 8×7 -bit multiplier presented in this chapter requires only test vectors to test all single stuck-at faults. Also, the full-adders, Manchester carry adders and the modified Booth encoders are exhaustively tested. The numbers of test vectors for any larger multiplier will still remain the same, i.e., 19 as long as the multiplicand has even number of bits and the multiplier (Y) has odd number of bits. Therefore, this testable multiplier can be said to be C-testable [17] within the constraints specified earlier. Further extensive investigation would be required to design a testable generalized multiplier which would overcome the above constraints.

Chapter 5

Conclusions and Recommendations

5.1 Conclusions

A generalized multiplier architecture based on the modified Booth algorithm capable of performing the multiplication of binary numbers represented in two's complement as well as in sign-magnitude notation has been developed. The hardware and delay overhead of the proposed design are quite low. The architecture is regular and therefore suitable for VLSI implementation.

The testability of the generalized multiplier has been investigated in this research. A C-testable design has been developed when the multiplicand (X) and the multiplier (Y) has even and odd number of bits respectively. In this case the multiplier can be fully tested for all stuck-at faults using only 19 test vectors irrespective of the size of the operands. This C-testable version of the multiplier does not require any extra logic blocks. The delay overhead is also quite low compared to the non-C-testable version of the generalized architecture. If the multiplier (Y) has even number of bits then complex logic blocks would be required for generating the required test patterns at the inputs of the most-significant modified Booth encoder. When the number of bits in the multiplicand (X) is odd, the

existing architecture does not remain C-testable, i.e., it can no longer be tested using a constant number of test vectors, however, modifications can be made to the multiplier to make it C-testable.

5.2 Future Works

Further investigation can be carried out on the C-testability of the generalized multiplier for any number bits (odd or even) in both the operands. This may include insertion of full-adders in the first row of the multiplier array to enhance the controllability of the architecture. The design of a logic block to apply all required patterns to the most significant Booth encoder when the multiplier (Y) has even number of bits would be a crucial and interesting part of any future work.

With continuous advances in VLSI technology, the number of devices accommodated on a single chip continues to grow. This number has already reached tens of millions [24]. Designing such large and complex chips is time consuming. Commercial manufacturers tend to market their finished chip within a short time frame in order to capture a major share of the IC market and also to remain competitive. Automatic layout generation and logic synthesis tools reduce the design time. This approach to IC design is therefore becoming increasingly popular in the industry [25]-[27]. Future research into the automatic generation of multiplier layouts of various sizes would be a very challenging one. VHDL [28]-[29] can be used for behavioral design entry and simulation. Since VHDL is a high-level language, this approach would result in a process independent design which is very easily portable from an existing process to a future one. This would enormously reduce the time required to port a custom or semi-custom design to another process.

References

- [1] F. P. J. M. Welton, A. Delaruelle, "A 2- μ m CMOS 10-MHz Microprogrammable Signal Processing Core With an On-Chip Multiport Memory Bank" *IEEE J. of Solid-State Circuits*, Vol. SC-20, No.3, pp. 754-760, June 1985.
- [2] K. Takeda, F. Ishino, "A single-chip 80-bit floating point processor" *IEEE J. of Solid-State Circuits*, Vol. SC-20, No.5, pp. 986-991, Oct. 1985.
- [3] P. A. Lynn, W. Fuerst, "Introductory Digital Signal Processing with Computer Applications" *Jhon Willey & Sons*, 1992
- [4] A. V. Oppenheim and R. W. Schafer, "Discreet-time Signal Processing" *Prentice-Hall of India PTY. Ltd.*, Delhi, 1994
- [5] J. J. F. Cavanagh, "Digital Computer Arithmetic Design and Implementation" *McGraw-Hill Book Company*, New York, 1985.
- [6] N. Weste and K. Eshraghian, "Principle of CMOS VLSI Design" *Addision-Wesley Publishing Company*, Sydney, 1993.
- [7] A. Pucknell and K. Eshraghian, "Basic VLSI design" *Prentice-Hall of Australia PTY. Ltd.*, Sydney, 1994
- [8] C. S. Wallace, "A suggestion for a fast multiplier" *IEEE Trans. on Electronic Comput.*, Vol. EC-13, pp. 14-17, Feb. 1964.
- [9] L. Dadda, "Some schemes for parallel multipliers" *Alta Frequenza*, Vol. 34, No. 5, pp. 349-356, May 1965.

- [10] D. G. Crawley and G. A. J. Amaratunga, "8 × 8 bit pipelined Dadda multiplier in CMOS" *IEE Proceedings*, Vol. 135, Pt. G, No. 6, Dec. 1988, pp. 231-240.
- [11] C. R. Baugh and B. A. Wooley, "A two's complement parallel array multiplication algorithm" *IEEE Trans. Comput.*, Vol. C-22, No. 12, pp. 1045-1047, Dec. 1973.
- [12] L. P. Rubinfield, "A proof of the modified Booth's algorithm for multiplication" *IEEE Trans. Comput.*, pp. 1014-1015, Oct. 1975.
- [13] J. A. Starzyk and Z. S. R. Dandu, "Overlapped Multi-bit scanning multiplier" *Proceedings of IEEE Int. Conf. on Comp. Design: VLSI in Computers*, ICCD; 85, NY. Oct. 1985, pp. 363-366
- [14] T. Williams and K. Parker, "Design for Testability - a Survey" *IEEE Trans. Comput.*, Vol. C-31, pp. 2-15, Jan. 1982.
- [15] B. R. Wilkins, "Testing Digital Circuits: An Introduction" *Van Nostrand Reinhold (UK) Co. Ltd.*, 1986.
- [16] J. P. Shen and F. J. Ferguson, "The design of easily testable VLSI array multiplication" *IEEE Trans. Comput.*, Vol. C-33, No. 6, pp. 554-560, June 1984.
- [17] A. D. Friedman, "Easily testable iterative systems" *IEEE Trans. Comput.*, Vol. C-22, pp. 1061-1064, Dec. 1973.
- [18] A. R. Takach and N. K. Jha, "Easily testable gate-level and DCVS multipliers" *IEEE Trans. Computer-Aided Design*, Vol. 10, No. 7, pp. 932-942, July 1991.
- [19] R. Stans, "The testability of a modified Booth multiplier" *Proceedings of 1st European Test Conference*, 1989, pp. 286-2938.
- [20] S. M. Aziz, "A C-testable modified Booth's array multiplier" *8th International Conference on VLSI design*, New Delhi, India, Jan. 1995, pp. 278-282.
- [21] W. A. J. Waller and S. M. Aziz, "A C-testable parallel multiplier using differential cascode voltage switch (DCVS) logic" *International Conference on Very Large Scale Integration: VLSI '93*, Sept. 6-10, 1993, pp. 3.4.1-10.

- [22] M. Roorda, "Method to reduce the sign bit extension in a multiplier that uses the modified Booth algorithm" *Electronic Letters*, Vol. 22, No. 20, pp. 1061-1062, 25th Sept. 1986.
- [23] N. Burgess, "Removal of sign-extension circuitry from Booth's algorithm multiplier-accumulators" *Electronic Letters*, Vol. 26, No. 17, pp. 1413-1415, 16th Aug. 1990.
- [24] D. Alpert and D. Avnon, "Architecture of Pentium microprocessor" *IEEE micro*, pp 11-21, June 1993
- [25] D. D. Gajski, N. D. Dutt, C. H. Wu, Y. L. Lin, "High-level Synthesis, introduction to chip and system design" *Kluwer Academic Publishers*, 1991.
- [26] R. K. Gupta and G. Demicheli, " System-level Synthesis using Re-programmable Components" *Proceeding of the European Conference on Design Automation (EDAC)*, 1992, pp 2-7
- [27] R. Gupta and G. Demicheli, "Hardware-software cosynthesis for digital systems" *IEEE Desig & Test of Components*, pp29-41, Oct. 1993.
- [28] Z. Navabi, "VHDL Analysis and Modeling of Digital Systems" *McGraw-Hill Inc.*, NY, 1993.
- [29] J. R. Armstrong, "Chip Level Modeling with VHDL" *Prentice-Hall International Inc.*, USA, 1989.

