CHAPTER 4

ANALYSIS OF LOW POWER, AREA EFFICIENT AND HIGH SPEED MULTIPLIER TOPOLOGIES

4.1 INTRODUCTION

Multiplication is one of the basic functions used in digital signal processing. It requires more hardware resources and processing time than addition and subtraction. In fact, 8.72% of all instructions in a typical processing are multiplier as discussed by Asadi (2007). 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. Multiplier is such an important element which contributes substantially to the total power consumption of the system. In multiply and accumulate (MAC) unit, multiplication is the main function, so there is a need of high speed multiplier. Currently, multiplication time is still a dominant factor in determining the instruction cycle time of a DSP chip. 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 for many applications.

To perform an M-bit by N-bit multiplication as shown in Figure 4.1, the M-bit multiplicand A=a(M-1)a(M-2)…….a1a0 is multiplied by the N-bit multiplier B=b(N-1)b(N-2)……b1b0 to produce the M+N bit product P=p(M+N+1)p(M+N+2)…..p1p0. The unsigned binary numbers A and B can be expressed by Equations 4.1 and 4.2. The equation for the product is defined in Equation 4.3 as discussed by Kim (2010).

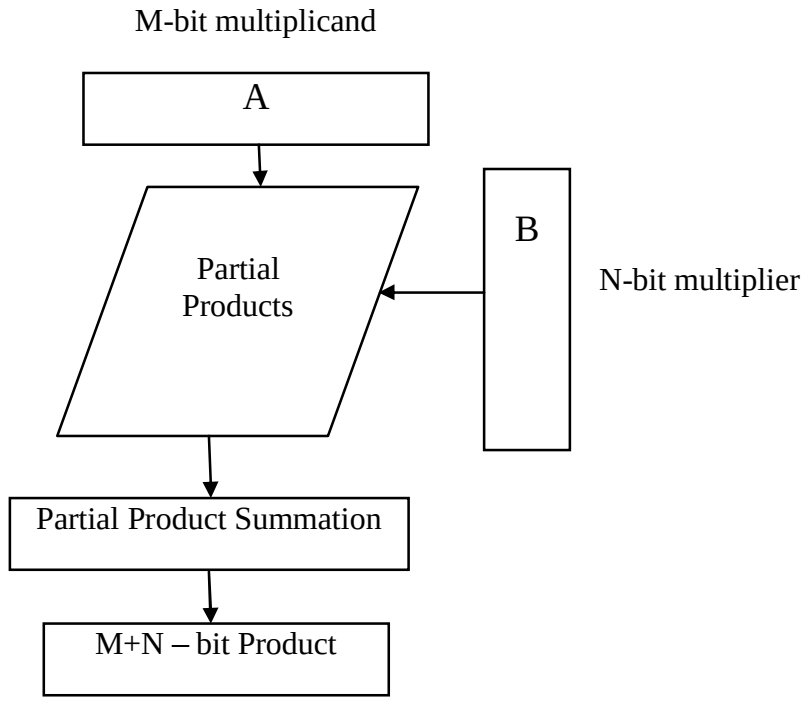

Figure 4.1 Generic multiplier block diagram

$$A = \sum_{i=0}^{M-1} a_i 2^i \quad (4.1)$$

$$B = \sum_{i=0}^{N-1} b_i 2^i \quad (4.2)$$

$$P=A.B=\sum_{i=0}^{M-1} a_i 2^i . \sum_{i=0}^{N-1} b_i 2^i = \sum_{i=0}^{M-1} \sum_{i=0}^{N-1} (a_i b_i 2^{i+j}) \quad (4.3)$$

With two's complement multiplication, both numbers are signed and the result is signed. If A and B is signed binary numbers, they are expressed by Equations 4.4 and 4.5. The equation for the product is defined in Equation 4.6.

$$A = -a_{M-1} 2^{M-1} + \sum_{i-0}^{M-2} a_i 2^i \quad (4.4)$$

$$B = -b_{N-1} 2^{N-1} + \sum_{i-0}^{N-2} b_i 2^i \quad (4.5)$$

$$P = A.B = (-a_{M-1} 2^{M-1} + \sum_{i-0}^{M-2} a_i 2^i).(-b_{N-1} 2^{N-1} + \sum_{i-0}^{N-2} b_i 2^i) \quad (4.6)$$

Any multiplier can be divided into three stages: Partial productions generation stage, partial products addition stage and the final addition stage. In the first stage, the multiplicand and the multiplier are multiplied bit by bit to generate the partial products. In this stage, a second-order Booth encoding algorithm is usually used instead to reduce the number of partial products to half. The second stage is the most important, as it is the most complicated and determines the speed of the overall multiplier. In the last stage, the two-row outputs of the tree are added using any high speed adder such as look-ahead adder to generate the output result.

Multipliers are categorized relative to their applications, architecture and the way the partial products are produced and summed up. They are Array multipliers and Tree multipliers.

In array multipliers, the counters and compressors are connected in a serial fashion for all bit slices of the Partial Product parallelogram as it can be seen in Figure 4.2, the array topology is a two-dimensional structure that fits nicely on the VLSI planar process.

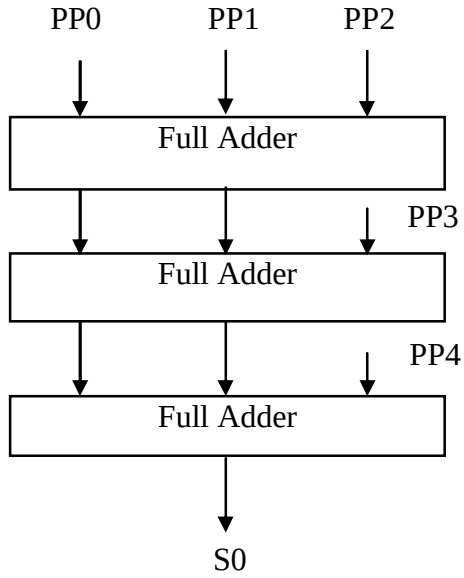

Figure 4.2 Array multiplier mechanism

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. An example is shown in Figure 4.3. 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.

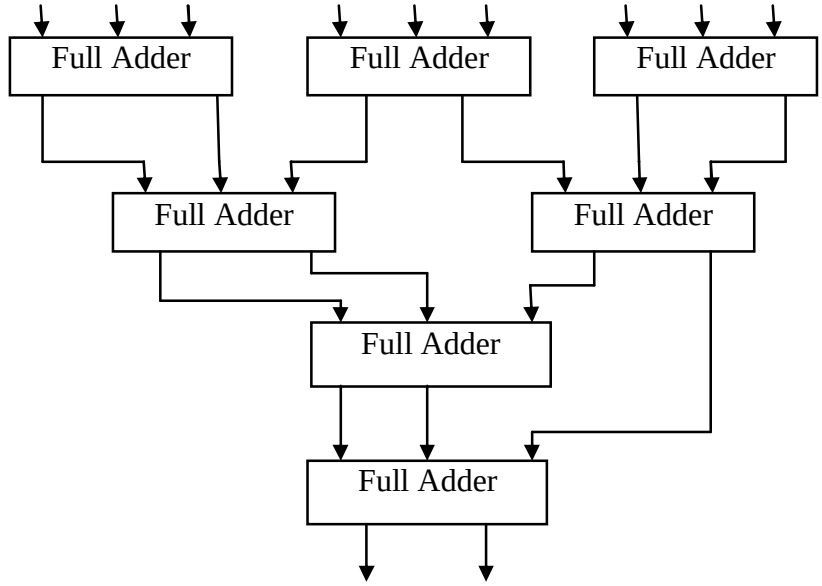

Figure 4.3 Partial product addition using tree topology

The first tree structure was introduced by Wallace. Wallace showed that PPs can be reduced by connecting (3:2) compressors are parallel in a tree topology. The regular trees include binary, balanced-delay and overturned-staircase trees as well as (9:2) compressors.

- Binary Tree
- Balanced-Delay tree
- Overturned-Staircase tree
- Wallace Tree

4.1.1 Compressors

Compressors are mostly used in multipliers to reduce the operands while adding terms of partial products. A compressor C_{iis} a combinatorial device that compresses N input lines in the position i to 2 output lines i.e. sum

and carry. In addition, there are L inputs lines coming to the compressor to different levels j. Figure 4.4 shows a simple compressor.

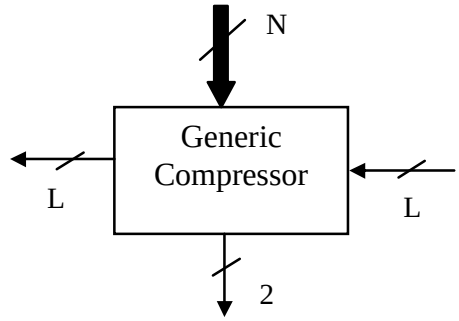

Figure 4.4 Generic compressor

A (3:2) compressor is basically a Full adder. It has 3 inputs i1, i2 and i3 to be summed up and provides 2 outputs (sum and carry). Gate level diagram of (3:2) compressor is shown in Figure 4.5.

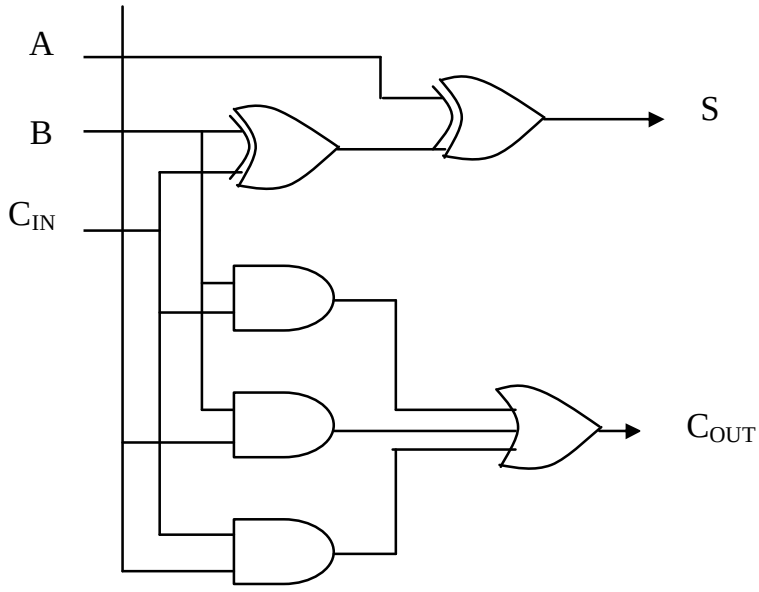

Figure 4.5 Gate level design of (3:2) compressor

A (4:2) compressor has 4 input lines i1, i2, i3and i4 that must be summed and has two output lines s and c, which are so called results of compression. The additional lines are input and output carries. The gate level design of a (4:2) compressor is shown in Figure 4.6.

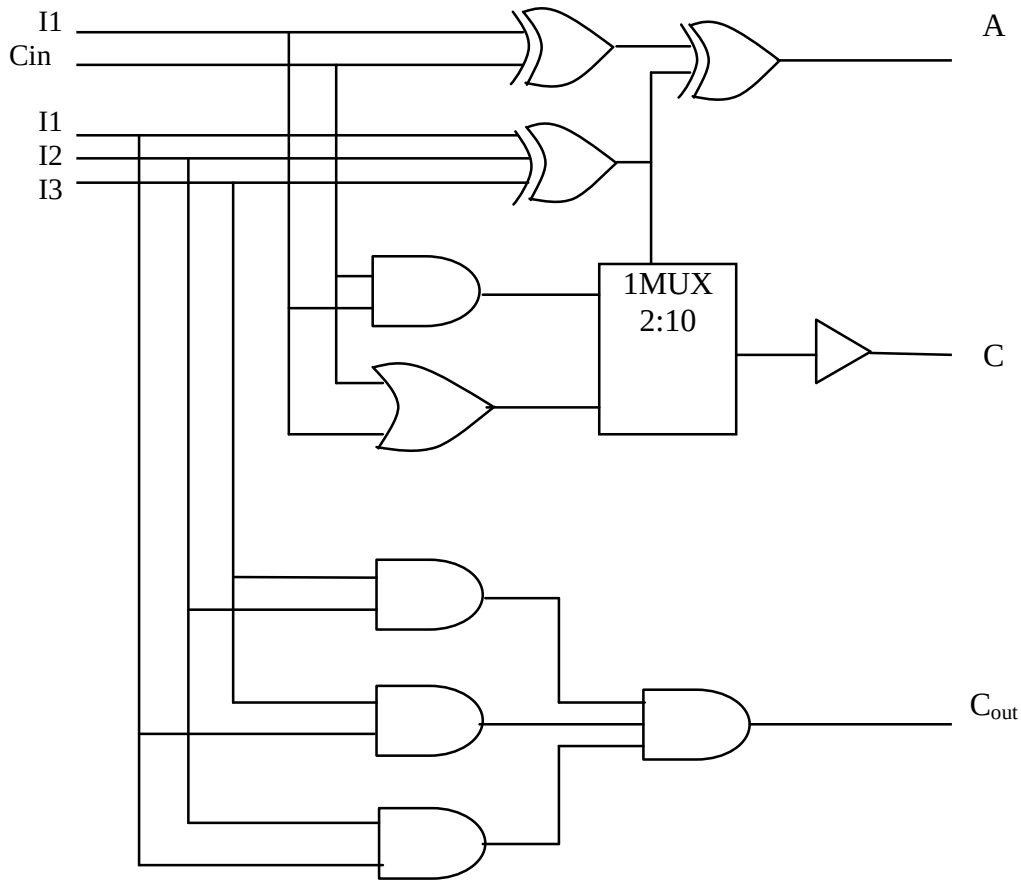

Figure 4.6 (4:2) Compressor logic diagram

A (4:2) compressor can also be designed using two (3:2) compressors as shown in Figure 4.7.

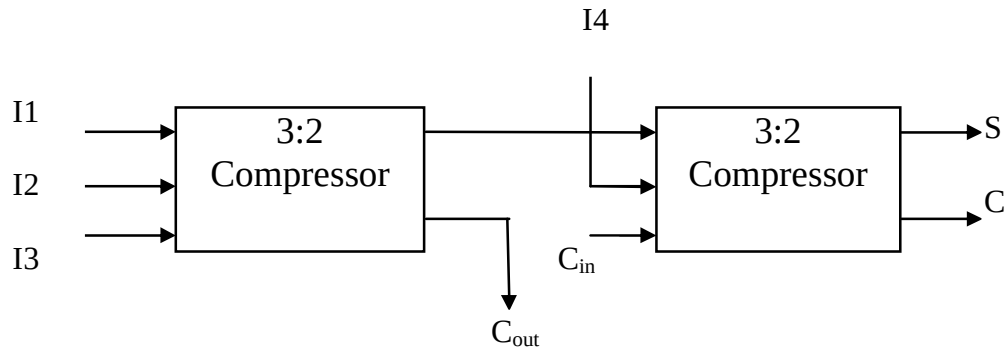

Figure 4.7 (4:2) Compressor using (3:2) compressor

4.2 MULTIPLIER TOPOLOGIES

The following section presents the design of multiplier topology. In this work the following multiplier structures are synthesized and analyzed for proposed MAC unit.

- Booth Multiplier
- Modified Booth Multiplier
- Wallace tree multiplier
- Booth Encoded Wallace Tree Multiplier.

4.2.1 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.

4.2.1.1 Booth recoding

Andrew D. Booth proposed the Booth recoding, or Booth algorithm in 1951. 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 4.1 shows the booth algorithm operation.

Table 4.1 Booth algorithm

Xi	Xi-1	Operations	Comments	Yi
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 avoids arithmetic, so this 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.

4.2.1.2 Booth example

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

22: 0010110, -22: 1101010

34: 0100010, -34: 1011110

Table 4.2 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 4.1. 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 4.2 Booth example

qiqi-1	Action	[*M*] *[A*]	0010110 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			
			0010110 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 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 worst-case standard multiplier.

4.2.2 Modified Booth Algorithm

The modified Booth encoding (MBE), or modified Booth's algorithm (MBA), was proposed by Macsorley (1961), as discussed by Liao (2002). 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 + log2r. Algorithm for modified booth is given below:

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

$$Y = -Y_{n-1}2^{n-1}+Y_{n-2}2^{n-2}+\ldots\ldots\ldots\ldots+Y_0 2^0 \quad (4.7)$$

$$Y=(-2Y_{n-1}+Y_{n-2}+Y_{n-3})2^{n-2}+(-2Y_{i-3}+Y_{i-4}+Y_{i-5})2^{i-4}+\ldots\ldots\ldots+(-2Y_1+Y0+Y_{-1})2^0 \quad (4.8)$$

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. Equation (4.8) can be represented by

$$Y = \sum_{i=0}^{i=\frac{n}{2}-1}(-2Y_{2i+1} + Y_{2i} + Y_{2i-1}).2^{2i} = \sum_{i=0}^{i=\frac{n}{2}-1}(Y_i 2^i) \quad (4.9)$$

$$X.Y = \left(-X_{n-1}2^{n-1} + \sum_{i=0}^{n-2} X_i 2^i\right)\left(-Y_{i-1}2^{n-1} + \sum_{j=0}^{n-2} Y_j 2^{ji}\right) \quad (4.10)$$

$$X.Y=\left(-X_{n-1}2^{n-1} + \sum_{i=0}^{n-2} X_i 2^i\right)\left(\sum_{j=0}^{\frac{n}{2}-1} Y_j\, 2^{2j}\right) \quad (4.11)$$

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 4.3 Modified booth algorithms

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.

Modified Booth Example

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

Multiplicand A= 34 = 00100010

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

A × B = -1428

Table 4.4 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		PP_1
				1	1	0	0	1	0	0	0	1	0	0				PP_2
		1	1	0	0	0	1	0	0	0	1	0						PP_3
1	0	1	1	1	0	1	1	1	1	0								PP_4
1	1	1	1	1	1	0	1	0	0	1	1	0	1	1	0	0		-1428

Table 4.4 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 has 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 and it is 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 significant block, so LSB of first block is always taken to be zero. Block diagram of an n×n–bit modified Booth multiplier is shown in Figure 4.8. 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 2 -bit addition.

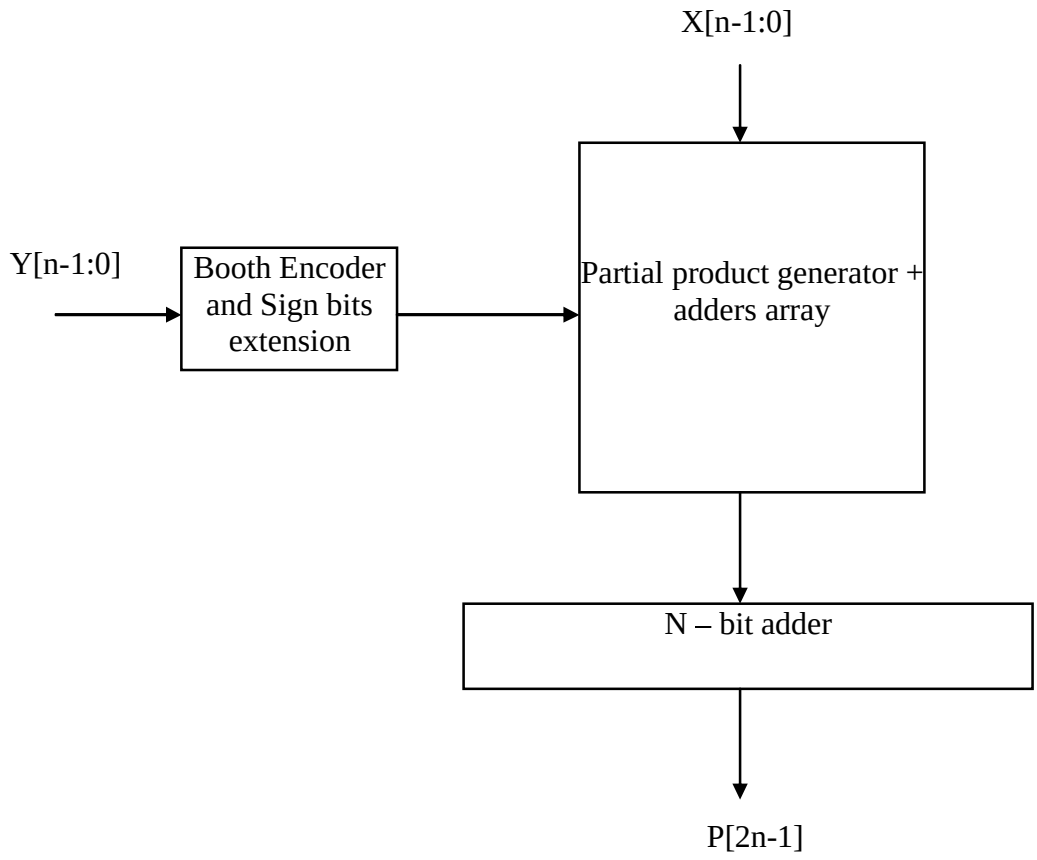

Figure 4.8 n × n modified booth multiplier

4.2.3 Wallace Tree Multiplier

A fast process for multiplication of two numbers was developed by Wallace. In 1964, C.S. Wallace 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". 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 bigger than 16 bits. 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.

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 one bit of the arguments, by each bit of the other, yielding n2 results. Depending on position of the multiplied bits, the wires carry different weights.
- Reduce the number of partial products by two 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 4.9 explains the various steps of Wallace tree multiplier.

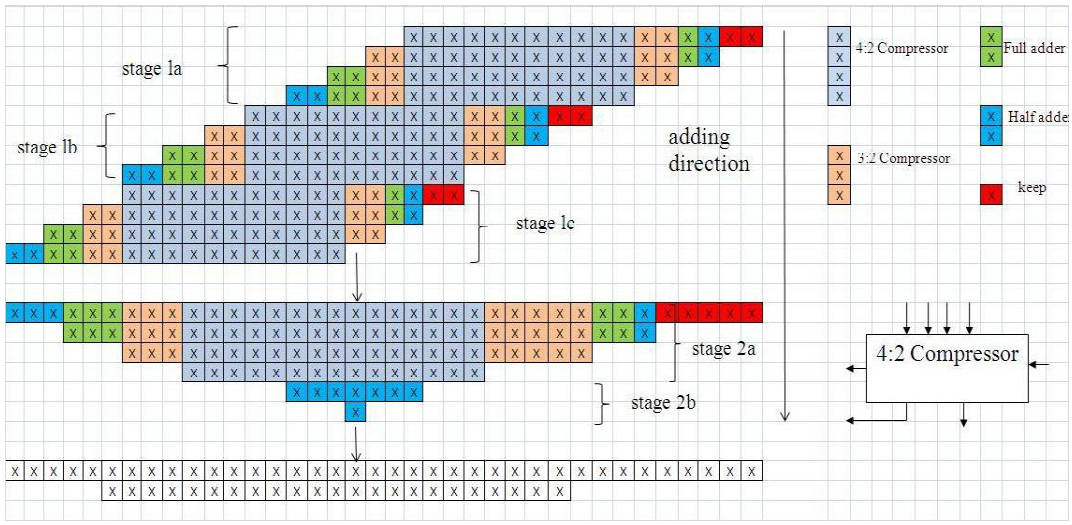

Figure 4.9 Wallace tree example

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

Advantage of Wallace Multiplier

Propagation delay in this multiplier is reduced in comparison to array multiplier.

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.

4.2.4 Booth Encoded Wallace tree Multiplier

In 16 X 16 bit multiplier, eight MBR generate eight partial products of 17-bit. 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 4.10.

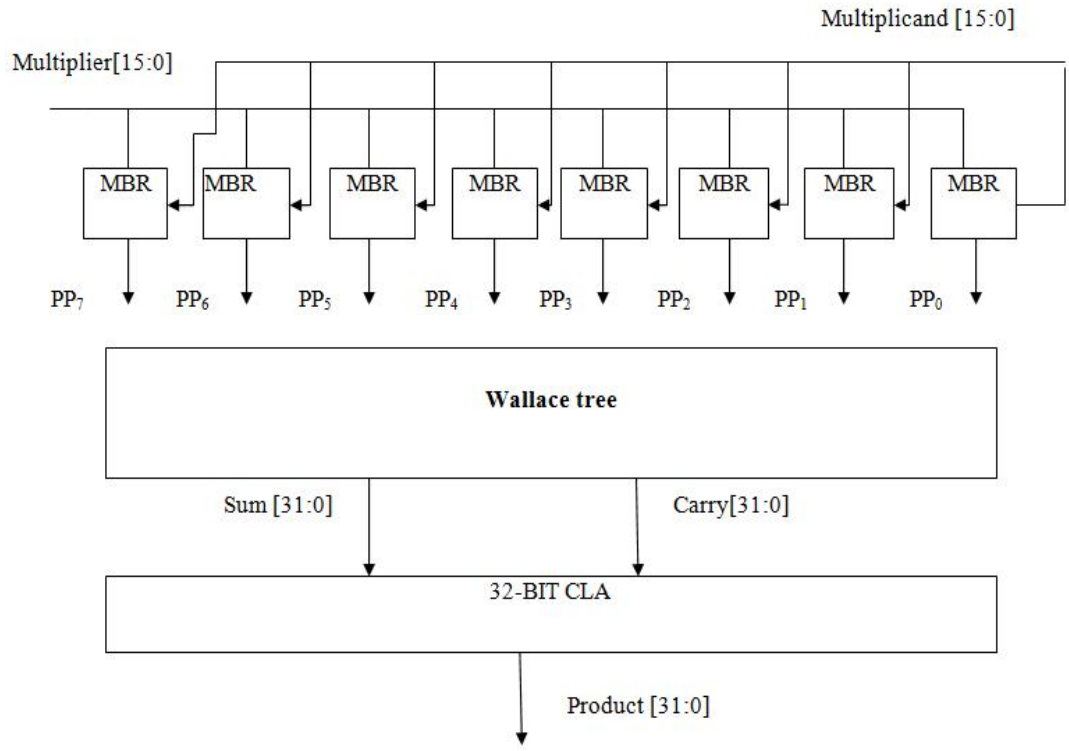

Figure 4.10 Block diagram of 16 × 16 bit booth encoded wallace tree

Wallace tree includes three rows of 4:2 compressors. The first row 4:2 compressors adds partial products PP0, PP1, PP2and PP3 , the second row adds partial products PP4, PP5, PP6 and PP7 and the third one add the sum outputs from the first rows and the carry bits from 4:2 compressors in the right column. Figure 4.11 shows the 4:2 compressor organizations for adding eight partial products. The carry and sum outputs of last row of

4:2 compressor are added with 32-bit CLA with the carry output bits shifted left one bit position to add with the sum bits. Figure 4.10 shows 32-bit CLA which requires three CLC levels and eleven carry look-ahead units. 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 C4, C8, C12, C20, C24 and C28. Third level CLC uses the group P and G outputs from the second level CLCs as inputs and provides the carry output C16. 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 C0 to produce carry output C32. Thus final 32-bit product is obtained.

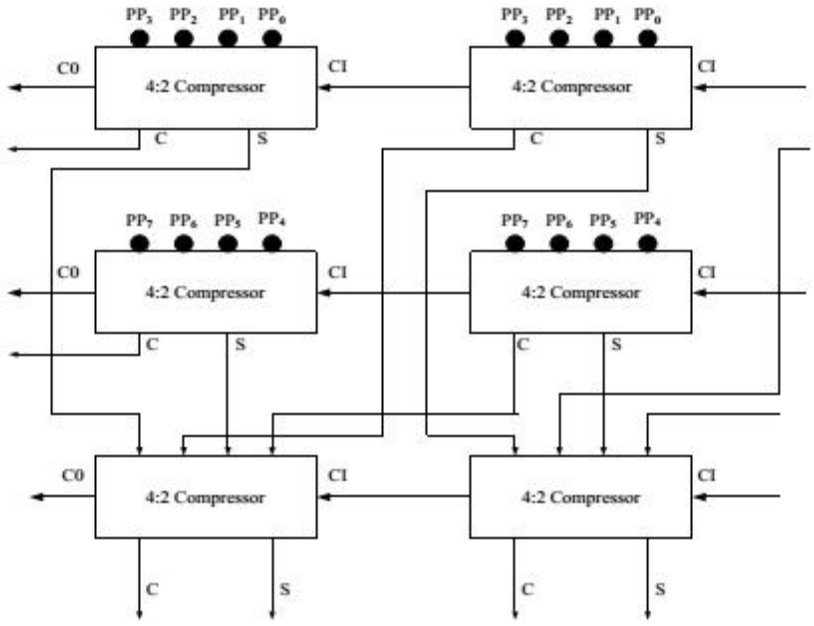

Figure 4.11 (4:2) Compressor organisation

4.3 SIMULATION RESULTS

The design of the above four 16x16 bit multipliers have been implemented on Spartan 3E (XC3S 500E). The table 4.5 shows the simulated

output of selected multipliers. All comparison based on the synthesis signifies keeping one common base for comparison which means, targeting the same FPGA device with same design constraints disguised for the synthesis of each multiplier.

Table 4.5 Area, delay and power dissipation comparison of multiplier topologies

Multipliers	Area	Delay	Power Dissipation
Booth Multiplier (BM)	4.53%	42.21	83.58 mW
Modified Booth Multiplier (MBM)	5.14%	44.96	85.39 mW
Wallace tree Multiplier (WTM)	7.32%	37.85	89.91 mW
Modified Booth Encoded Wallace tree Multiplier (MBWTM)	5.48%	39.39	85.55 mW

From the comparison as shown in Figure 4.12, the booth multiplier is having an area of 50.47% which is less when compared to other three multipliers. When coming to delay the Wallace tree is having 38.859 ns which is less when compared to other multipliers.

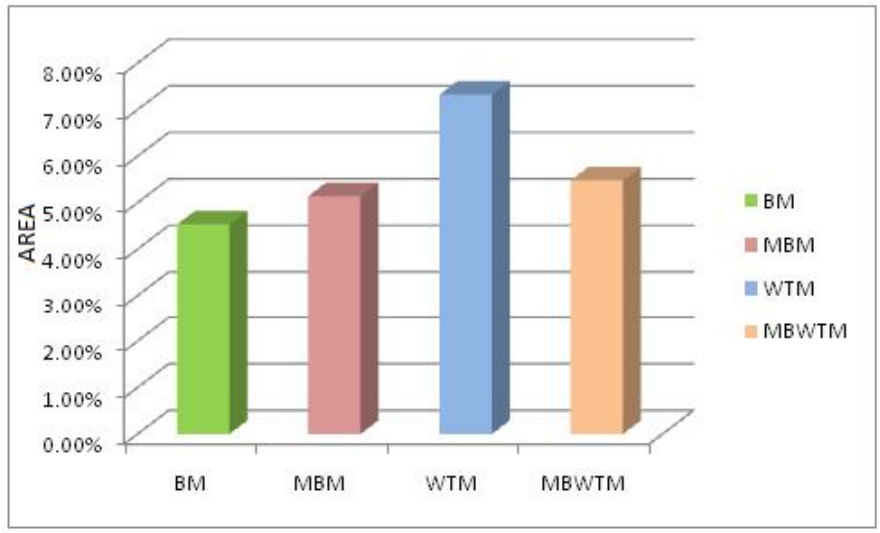

Figure 4.12 Area comparison of multiplier topologies

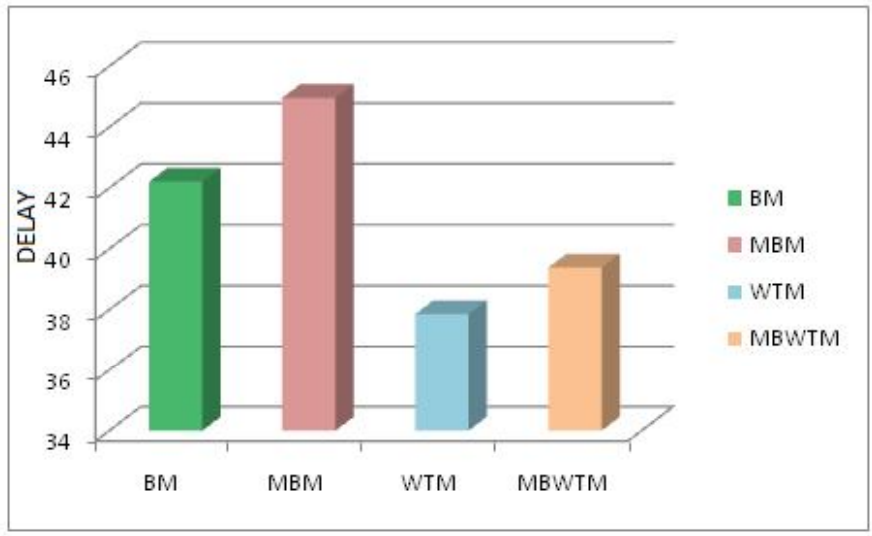

Figure 4.13 Delay comparison of multiplier topologies

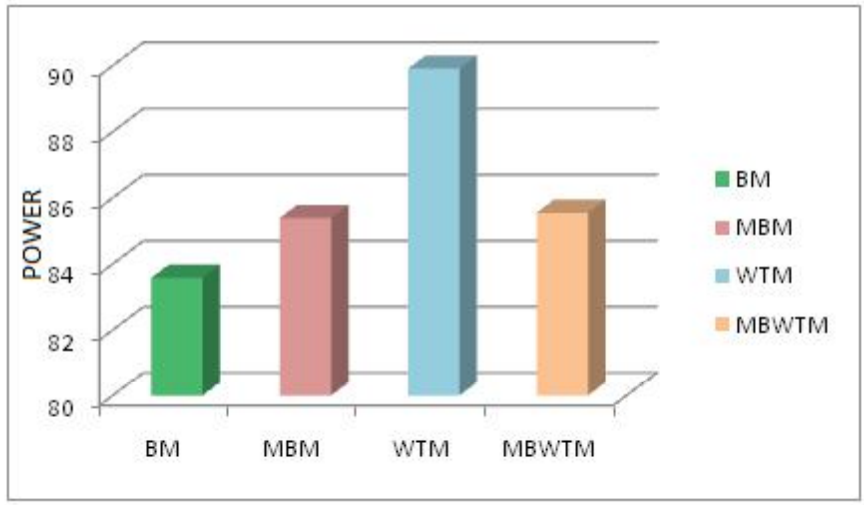

Figure 4.14 Power dissipation comparison of multiplier topologies

But the Modified Booth Encoded Wallace tree has an area of 54.87% which is slightly higher than Booth and Modified Booth multiplier. But when comparing the delay, Modified Booth Encoded Wallace tree is having 39.398 ns which is lower than Booth and Modified Booth multiplier and higher than Wallace tree multiplier. As there is tradeoff between area, delay and power consumption, the Modified Booth Encoded Wallace tree

multiplier is chosen for designing low power and high speed MAC Unit. The delay and Power dissipation comparison is shown in Figure 4.13 and 4.14.

After selecting the optimized multiplier i.e., Modified booth encoded Wallace tree multiplier, the different adders are combined with this multiplier and simulated. The Table 4.6 shows the comparison of the multiplier and adder combination.

Table 4.6 Area, delay and power dissipation comparison of modified booth encoded wallace tree multiplier with various adders

Multipliers	Area	Delay	Power Dissipation
Modified Booth Encoded Wallace tree Multiplier with Carry look ahead adder (MA1)	3.1%	20.20	80.21 mW
Modified Booth Encoded Wallace tree Multiplier with carry skip fixed block adder (MA2)	3.7%	17.69	81.58 mW
Modified Booth Encoded Wallace tree Multiplier with carry skip variable block adder (MA3)	3.1%	20.08	80.21 mW
Modified Booth Encoded Wallace tree Multiplier with Ripple Carry adder (MA4)	3.4%	20.33	80.95 mW
Modified Booth Encoded Wallace tree Multiplier with carry select adder (MA5)	3.5%	13.66	81.88 mW

From the comparison the area is approximately same for all the combinations. But the delay for carry select adder combination is less, when compared to all other combination. So modified booth encoded Wallace tree multiplier with carry select adder for partial product generation is chosen for proposed multiply and accumulate unit. The area, delay and power dissipation comparison is shown in Figures 4.15, 4.16 and 4.17 respectively.

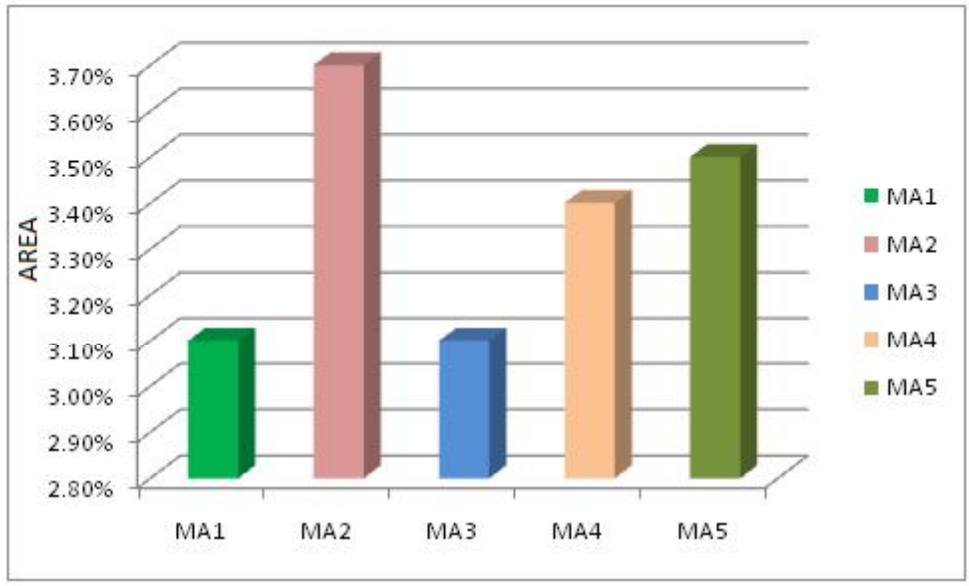

Figure 4.15 Area comparison of modified booth encd. wallace tree multiplier with various adder topologies

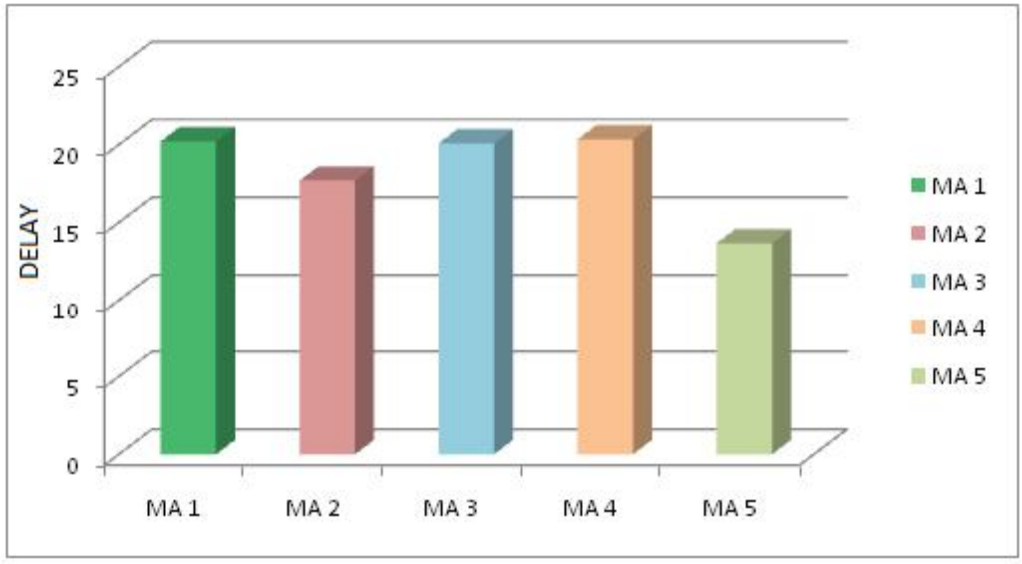

Figure 4.16 Delay comparison of modified booth encd. wallace tree multiplier with various adder topologies

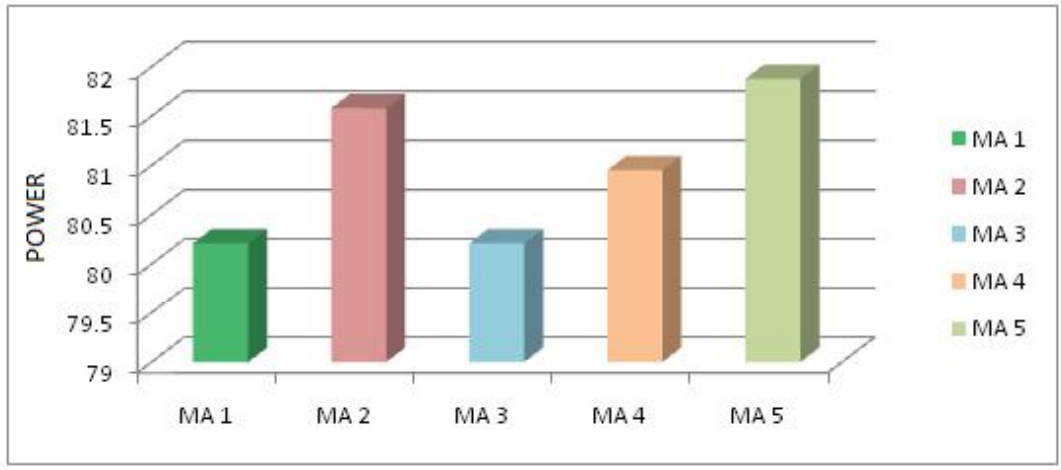

Figure 4.17 Power dissipation comparison of modified booth encd. wallace tree multiplier with Various adder topologies

4.4 CONCLUSION

The design of various multiplier and multiplier/ adder combination have been implemented on Spartan 3E (XC3S 500E). Amongst all the multipliers conferred in this chapter, the modified booth encoded Wallace tree multiplier is chosen for MAC unit. Analysis made using multiplier with different adders demonstrate that carry select adder combination is having maximum speed and efficiency. Even though carry select adder combination is having little area overhead, modified booth encoded Wallace tree multiplier with carry select adder can be chosen for designing the MAC unit because there is a tradeoff between area, delay and power.