Design and Analysis of Generic Architecture of Multipliers

Ms. Ritu Jain
M.Tech.*, VLSI Design
Poornima college of Engineering, Jaipur

Mr. Dinesh Chand Gupta
Asst. Professor, ECE Dept.
Poornima college of Engineering, Jaipur

Abstract:
Multipliers are becoming one of the most important building block in recent digital signal processors and other high performance systems. The area and delay of the multiplier often affects the overall area and speed performance of a VLSI system. In this paper we proposed the generic architecture of the four different multipliers. This architecture will give area, delay and other performance parameters of the multipliers for any number of input bits. The four multipliers include array multiplier, Column Bypass multiplier, Modified Booth multiplier, and Wallace tree Multiplier. By using the generic architecture we presented a comparative analysis in terms of area and delay offered by these multipliers for different number of input bits. The circuit is simulated using the Xilinx tool.

Index Terms – Array Multiplier, Column Bypass Multiplier, Modified Booth Multiplier, Wallace Tree multiplier

I. INTRODUCTION

Multipliers are the key components of many high performance systems such as RISC(reduced instruction set computing), microprocessors, digital signal processors and graphics engines. Design of portable battery operated multimedia devices also requires energy efficient multiplication circuits. So multipliers play a very important role in the designing of VLSI circuits for these high speed processors.

A VLSI system performance very highly depends on the performance of the multiplier because generally the multiplier is the most area consuming and the slowest element in the system. Thus after guaranteeing the correct digital functionality, the primary consideration for system designers has always been to optimize the area and the time delay of the multiplier circuit. Other factors may also have equal or greater importance for example power dissipation, yield/reliability issues etc. Nevertheless, in successive generations of integrated circuit technologies area and the time delay are also one of the primary considerations for the designers. Hence optimizing the time delay and area is a major design issue.

However, a major complication in microelectronic circuits is the fact that many design decisions involves a area – delay tradeoff. One cannot be lowered without raising the other. If we reduce the time delay to improve speed, that may result in larger area. In this paper, we will arrive at a better trade-off between the two area and the time delay by realizing the generic architectures of the multipliers and then comparing the results of all the four multipliers for different number of input bits.

The common multiplication method is “add and shift” algorithm. M*N bit multiplication can be viewed as forming N partial products of M bits each and then summing the appropriately shifted partial products to produce an M+N bit result. To generate N partial products we have to perform binary multiplication of multiplier with each multiplicand bit. Binary multiplication is equivalent to a logical AND operation. Each column of partial products must then be added and if necessary, carry values are passed to the next column.

There are a number of techniques that can be used to perform multiplication. In general the choice is based upon the factors such as area, time delay, latency, throughput and design complexity. The first multiplier which we are going to design is the Array Multiplier. Array multiplier is well known due to its regular structure. In this multiplier the addition can be performed using the normal carry save adder. But this adder affects the time delay of the multiplier. Total N-1 adders are required for multiplication where N is the multiplier length. The second multiplier which we are going to design is the Column Bypass multiplier. Column bypassing technique is another well known technique. In this technique the operations in a column can be disabled if the corresponding bit in the multiplicand is zero. This technique totally depends on the number of zeroes in the multiplicand bits. The third multiplier which we are going to design is the modified booth multiplier. The number of partial products to be added can be reduced using modified booth algorithm. The array multiplier and column bypass multiplier compute the partial products in a radix-2 manner, i.e. by observing one bit of the multiplier at a time. Radix 2^r multipliers produce N/r partial products, each of which depends on the r bits of the multiplier. For example, a radix-4 multiplier produces N/2 partial products. Fewer partial products leads to a smaller and faster carry save adder array. In this paper we are going to design a radix-4 Modified Booth algorithm. This algorithm is also a powerful algorithm for signed number multiplication, which treats both positive and negative numbers uniformly. The fourth multiplier which we are going to design is the Wallace tree multiplier. The number of logic levels required to perform the summation can be reduced using the Wallace tree algorithm. In array multipliers the column addition is slow because only one carry save adder is active at a time. But if we perform

summation of partial products in parallel rather than sequentially then we can speed up the column addition. Wallace tree algorithm uses this approach. The Wallace tree requires [$\log_{3/2}(N/2)$] levels of carry select adders to reduce N inputs down to 2 outputs. This also gives speed improvements.
In this paper we are going to design the generic architecture (n*n bit multiplication) of the four multipliers which are mentioned above and then we will calculate the area occupied and the time delay of all multipliers for different number of input bits.

II. ARRAY MULTIPLIER

Array multiplier is well known due to it is regular structure. It uses the fact multiplications form a recurring pattern. Multiplier circuit is based on add and shift algorithm. Each partial product is generated by the multiplication of the multiplicand with one multiplier bit. The partial product are shifted according to their bit orders and then added.

The generation of N partial products requires N*M two bit AND gates. Then the summation of N partial products requires N-1, M bit adders. Most of the area of the multiplier is devoted to the adding of N partial products. The shifting of the partial product for their proper alignment is performed by simple routing. The hardware circuit for 4 – bit multiplication using array multiplier is shown in figure 1.

One advantage of array multiplier comes from its regular structure. Since it is regular, it is easy to layout. So the design time of array multiplier is much less than that of a tree multiplier. Another advantage is its ease of design for a pipelined architecture. A fully pipelined array of the multiplier has been successfully designed for high speed DSP application.[3]

The main limitation of full linear array multipliers is that they are very large. As operand size increases, linear arrays grow in size at a rate equal to the square of the operand size. This is because the number of rows in the array is equal to the length of the multiplier, and the width of each row is equal to the width of the multiplicand. Another problem with array multiplier is that the hardware is underutilized. As the sum is propagated down through the array, each row of CSA's computes a result only once, when the active computation front passes that row. Thus, the hardware is doing useful work for a very small percentage of time.[3]

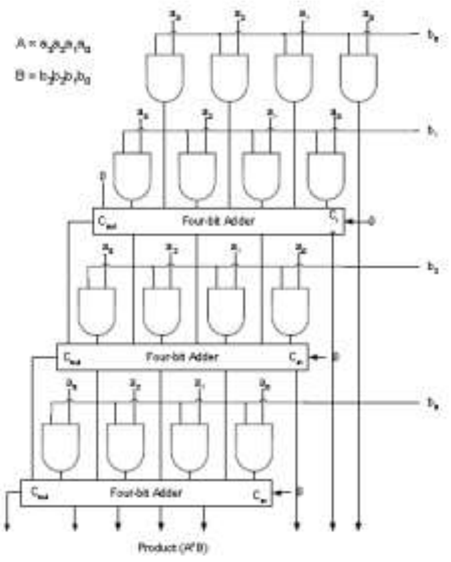


Fig. 1: 4*4 Array Multiplier

III. COLUMN BYPASS MULTIPLIER

The computation of multiplier manipulates the input data to generate the partial products for subsequent addition operations, which requires many switching activities in CMOS circuit design. These switching activities within the functional unit of a multiplier account for the majority of the power dissipation of a multiplier. So we introduce a parallel multiplier in which switching activities are reduced through architecture optimization using bypassing scheme. [4]

Column bypassing means turning off some columns in the multiplier array whenever certain multiplicand bits are zero. With this technique, the operations in a column can be disabled if the corresponding bit in the multiplicand is 0. It also uses additional tri state buffers and MUXs to skip the FA cells in the column of 0 bits. The modified full adder for column bypassing multiplier is shown in figure 2. Based on this, a 4*4 column bypass multiplier structure is developed as shown in figure 3.

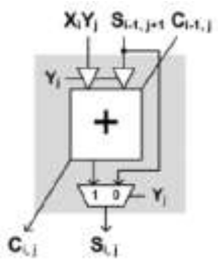


Fig. 2: The Modified FA Cell for Column Bypass Multiplier. [5]

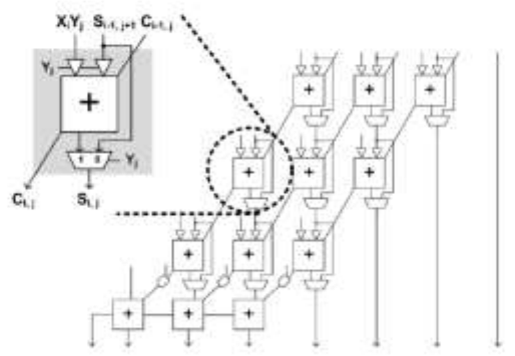


Fig 3: Structure of 4*4 Column bypassing multiplier [5]

IV. MODIFIED BOOTH MULTIPLIER

In the standard add and shift operation which was used in array multipliers, the partial products are computed in a radix-2 manner, i.e. by observing one bit of the multiplier at a time. Each multiplier bit generates one multiple of the multiplicand to be added to the partial product. But if the length of multiplier is very large, then a large number of partial products have to be added. In this case the delay of the multiplier will increase as the number of bits in the multiplier will increase because the delay in the multiplier is determined by the number of additions to be performed.

Booth algorithm is a method that will reduce the number of multiplicand multiples. A higher radix representation of a number, leads to fewer digits. For example, a k-bit binary number can be interpreted as k/2 digit radix-4 number, k/3 digit radix-8 number and so on. So we can deal with more than one bit of the multiplier in each cycle of using higher radix multiplication. Radix 2^r multiplier produces N/r partial products, each of which depends on group of r bits of the multiplier. For example a radix-4 multiplier produces N/2 partial products. Fewer partial products lead to a smaller and faster CSA array. For example a radix-4 multiplication is shown in the figure 4.

Multiplicand	A=	●●●●	
Multiplier	B=	(●●)(●●)	
Partial Product Bits		●●●●	$(B1B0)_2A$
		●●●●	$(B3B2)_2A$
Product	P=	●●●●●●●●	

Fig 4.: Radix-4 Modified Booth multiplication in dot notation

Initially grouping of multiplier bits is processed. The three bits are selected at a time starting from left and with overlapping of left most bit. Then the group of three bits of multiplier is encoded according to the radix-4 modified booth encoder. Booth encoder generates signals to control the selector to choose -2Y, -Y, 0, Y or 2Y. Figure 5 shows the design of a booth encoder and booth selector. Table 3.1 shows how the partial products are selected, based on bits of the multiplier. Since the booth method applies to 2's complement arithmetic, it is necessary to use sign extensions to obtain correct result. For sign extension, all the rows of the partial products have to be extended to 2*N, where N is the length of the multiplicand, with the use of the sign bit of the respective partial product. But it increases the capacitive load, the area, and the computational time.

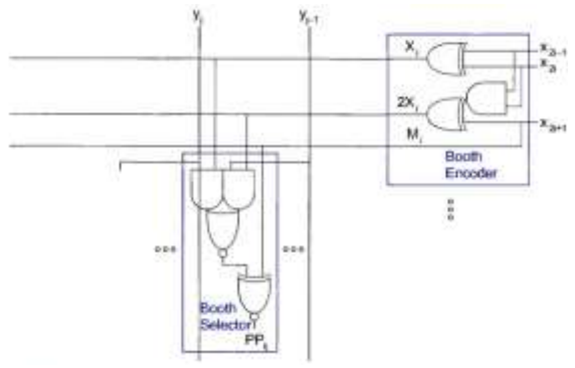


Fig 5: Radix-4 Booth Encoder and Selector

Inputs			Booth Selects			Partial Product
x_{2i+1}	x_{2i}	x_{2i-1}	X_i	$2X_i$	M_i	PP_i
0	0	0	0	0	0	0
0	0	1	1	0	0	Y
0	1	0	1	0	0	Y
0	1	1	0	1	0	2Y
1	0	0	0	1	1	-2Y
1	0	1	1	0	1	-Y
1	1	0	1	0	1	-Y
1	1	1	0	0	1	0

Table 1: Radix-4 Modified Booth Encoding values

V. WALLACE TREE MULTIPLIER

Several popular and well known schemes are developed, with the objective of improving the speed of the parallel multiplier. Tree multiplication is also a very important iterative realization of parallel multiplier for this purpose. Tree multipliers are faster and use less power than array multipliers. This advantage becomes more pronounced for multipliers bigger than 16-bits. The partial sum adders are rearranged in a tree like fashion, so reducing both the critical path and the number of adder cells needed.

All the tree algorithms use the matrix where the columns represent each bit weight and the number of rows represents how many partial products that bit weight has. The challenge is to realize the complete matrix with a minimum depth and a minimum number of adder elements. The algorithm reduces the matrix by adding FAs and HAs to it. The first type of operator that can be used to cover the array is a full adder, which takes three inputs and produces two outputs (a 3, 2 counter). The sum, located in the same column and the carry, located in the next one. For this reason, the FA is called a 3-2 compressor. It is denoted by a circle covering three bits. The other operator is half adder, which takes two input bits in a column and produces two outputs (a 2, 2 counter). It is denoted by a circle covering two bits.

To arrive at the minimal implementation, we iteratively cover the tree with FAs and HAs starting from its densest part. The

FAs and HAs produces an output matrix, which represents the partial products for the next stage of the algorithm. The output matrix contains the partial products that still need reduction. This task is repeated several times, until the output contains only columns with one or two partial products. Each bit weight will then only have two outputs, and the result from the tree can be put into a vector merging adder (VMA). By connecting the FAs and HAs from each reduction stage, we will get a structure that looks very similar to a tree, hence the name is tree multiplier. Figure 6 shows the transformation of a partial product tree into a Wallace tree. Figure 7 shows the hardware implementation of 4*4 Wallace tree multiplier.

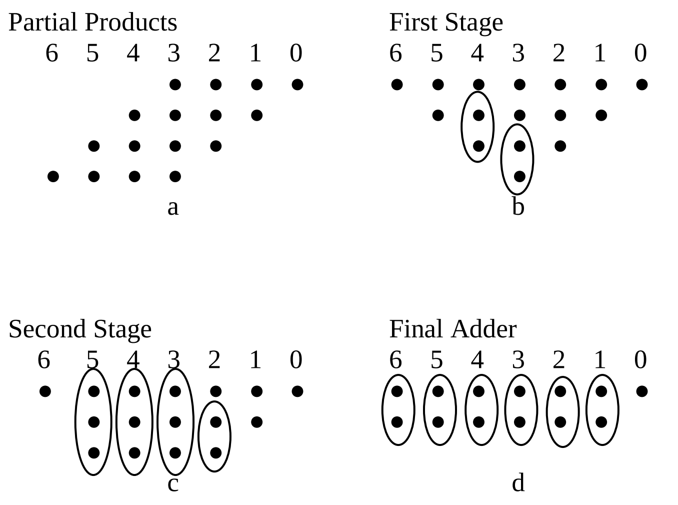


Fig 6: Transforming a partial product tree(a) into a Wallace tree(b,c,d)

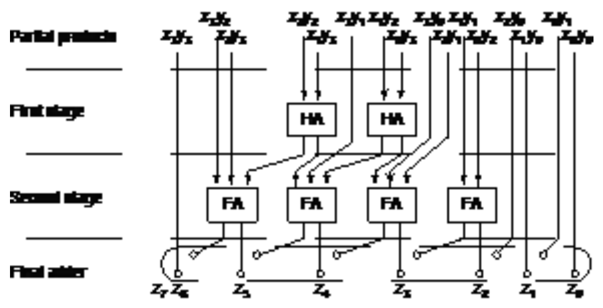


Fig 7: Hardware implementation of Wallace tree for four-bit multiplier[1]

The Wallace tree multiplier does not have a nice repetitive structure as the array multiplier. So it has the disadvantage of being irregular. So it uses a lot more area on wiring and also complicates the task of creating an efficient layout.

VI. PROPOSED DESIGN

Our objective is to find out which multiplier is best suitable with less area and less time delay for a particular application. Existing digital multipliers are designed for a fix number of input bits for example 8-

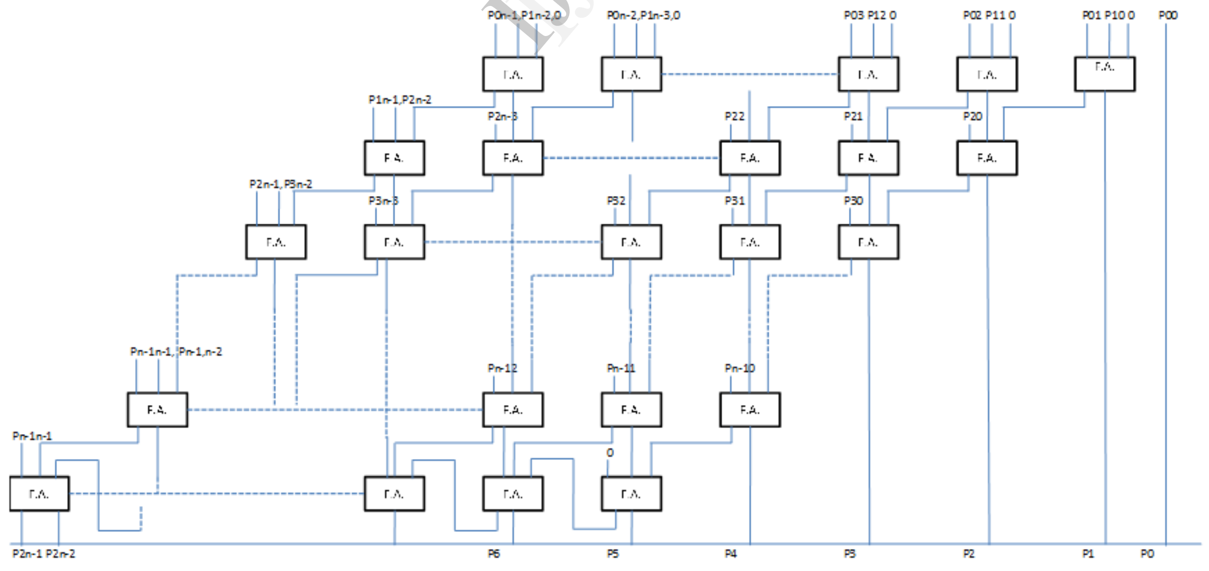


Fig 8: Generic Architecture of Carry Save Array Multiplier

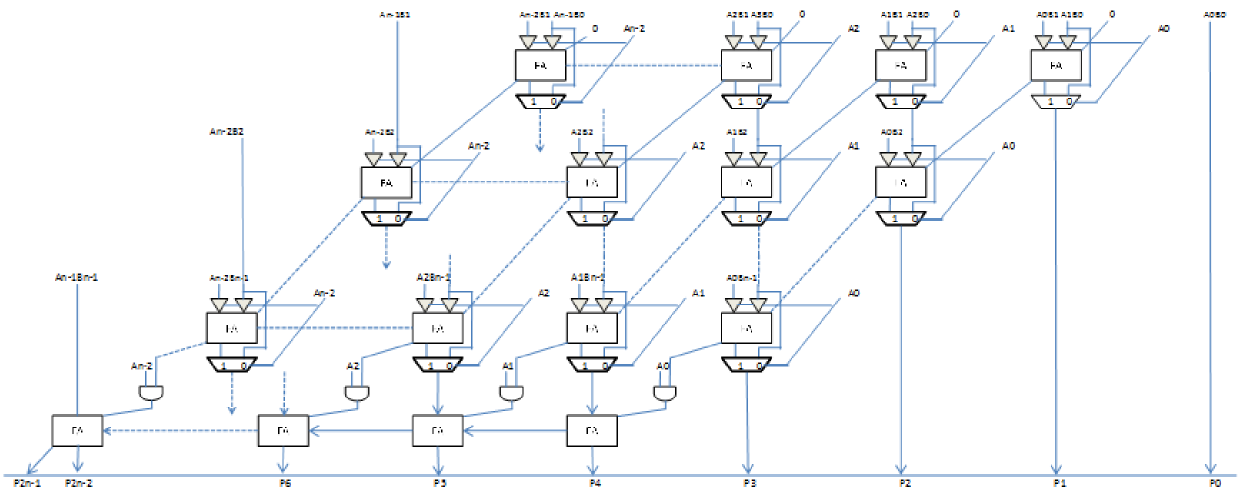
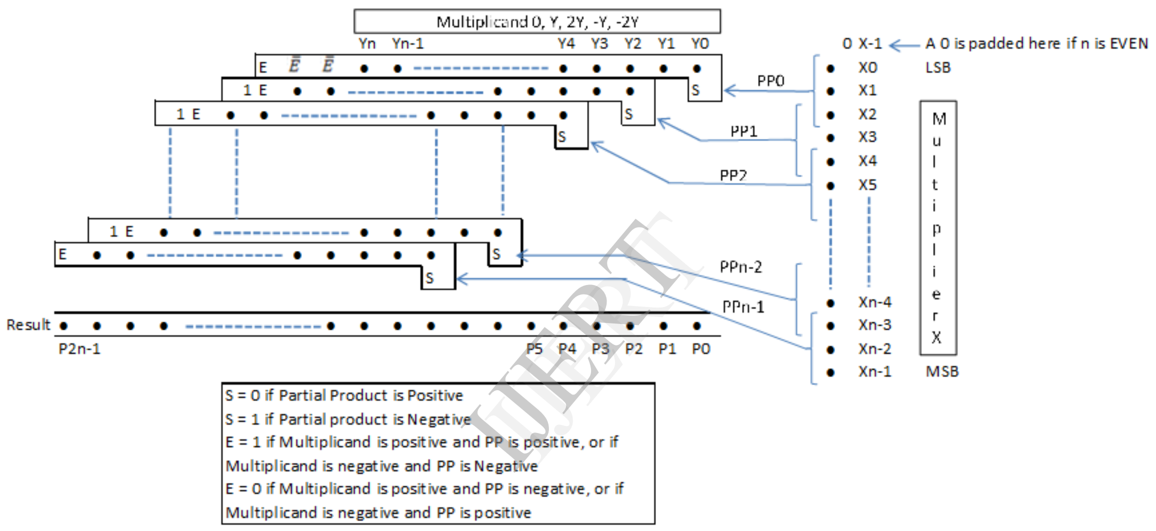


Fig 9: Generic Architecture of Column Bypass Multiplier



S = 0 if Partial Product is Positive
S = 1 if Partial product is Negative
E = 1 if Multiplicand is positive and PP is positive, or if Multiplicand is negative and PP is Negative
E = 0 if Multiplicand is positive and PP is negative, or if Multiplicand is negative and PP is positive

Fig 10: Generic architecture of Modified Booth multiplier in dot notation

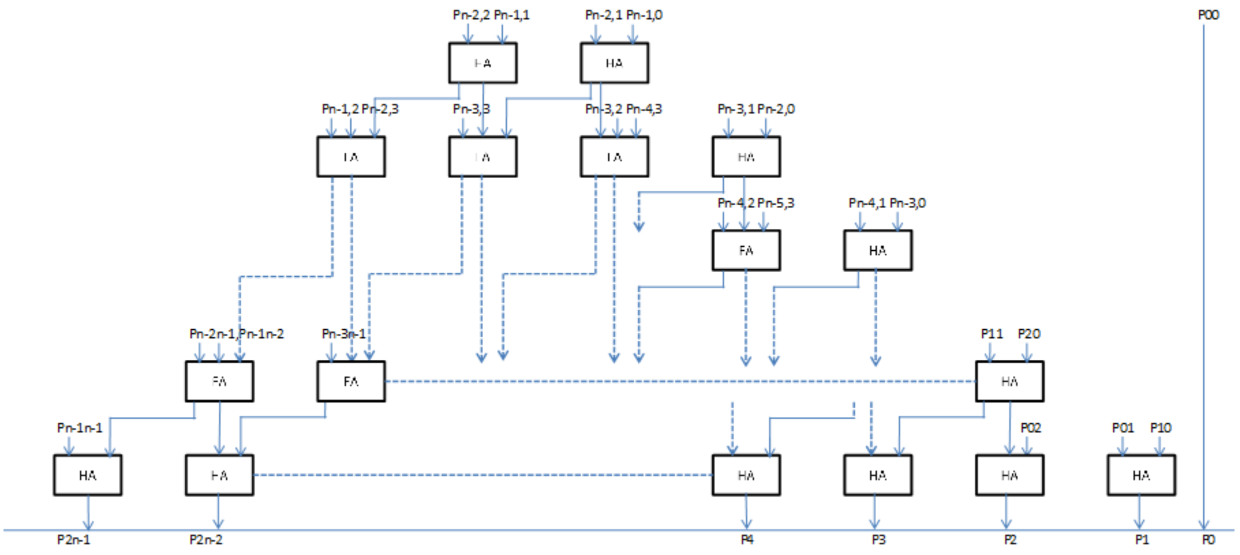


Fig 11: Generic Architecture of Wallace tree multiplier

bit, 64-bit, 128-bit etc. But in today's digital signal processors we can have any n*n bit multiplication. So it becomes a limiting factor for broad range of applications. As a result we have designed the generic architectures of the four multipliers named as array multiplier, column bypass multiplier, modified booth multiplier and Wallace tree multiplier. This will give the correct multiplication result along with the area occupied and the time delay of that multiplier for any number of input bits. In this way we can find out the best suitable multiplier which will occupy the less area and also has the lesser time delay for any number of input bits and thus optimization in the design can be obtained. Figure 8, 9, 10, 11 shows the generic architectures of Carry save Array multiplier, Column bypass multiplier, Modified Booth multiplier and Wallace tree multiplier respectively.

VII. RESULTS AND ANALYSIS

A. Propagation Delay

For the propagation delay the critical path delay is measured. The propagation delay of each architecture is measured and shown in table 1 and graphed in figure 12.

TABLE 1
Propagation Delay of multipliers (in ns)

Name of multiplier / No. of input bits	Array Multiplier	Column bypass Multiplier	Modified Booth Multiplier	Wallace Tree Multiplier
N=4	17.681	15.853	10.249	7.165
N=8	32.001	29.452	12.418	7.165
N=16	61.241	57.231	12.418	10.178

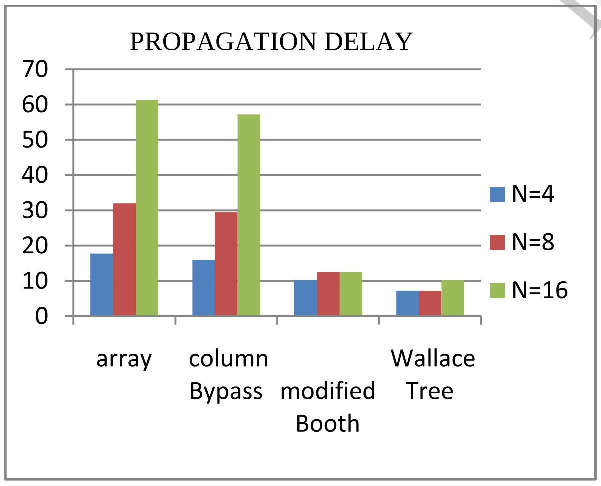


Figure 12: Propagation Delay test results

We can see from the graph that in terms of delay Wallace tree multiplier performs best while array multiplier and column bypass multiplier performs worst. As the number of input bits increases the significant difference between the delay of different multipliers also increases.

B. Area

The area of each multiplier is calculated in terms of no. of LUTs, no. of slices and gate count and it is tabulated in table 2 and graphed in figures 13, 14 and 15 respectively.

TABLE 2
Area occupied by multipliers

		Array Multiplier	Column bypass Multiplier	Modified Booth Multiplier	Wallace Tree Multiplier
No. of LUTs	N=4	33	24	39	46
	N=8	133	112	159	238
	N=16	525	480	623	438
No. of Slices	N=4	18	13	23	32
	N=8	69	58	84	140
	N=16	268	241	321	240
Gate Count	N=4	198	144	352	695
	N=8	798	672	1423	3407
	N=16	3150	2880	5563	6217

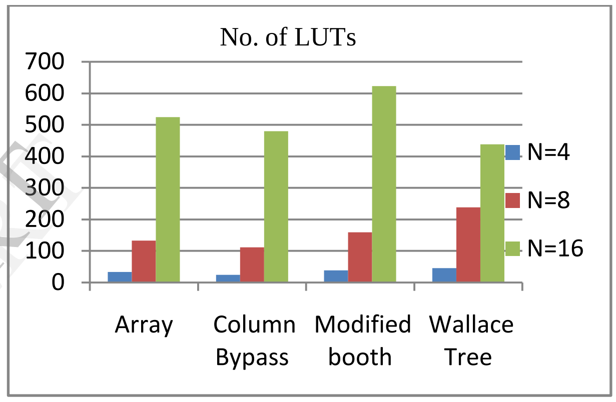


Fig 13: Total No. of LUTs test results

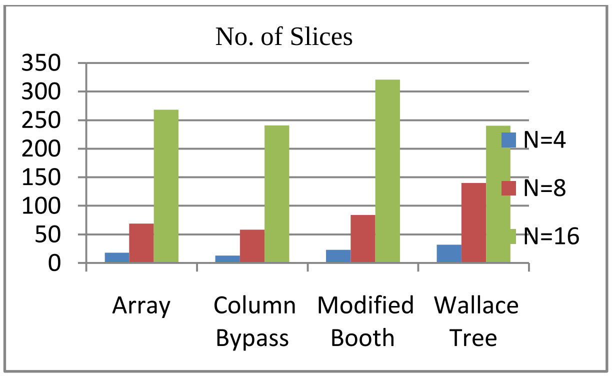


Fig 14: Total No. of slices test results

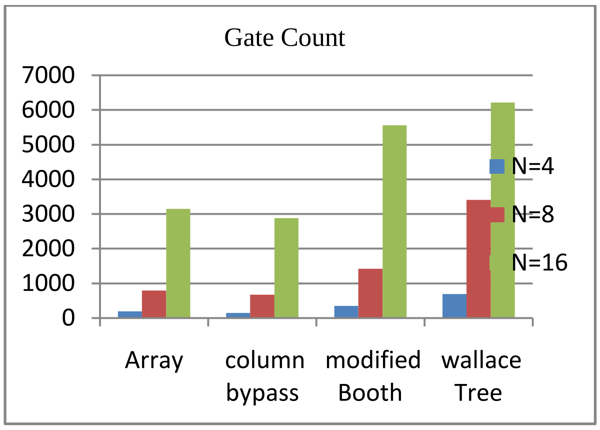


Fig 14: Total No. of Gate Count test results

We can see from the graph that the column bypass multiplier has the minimum gate count as compared to other multipliers while the Wallace tree multiplier occupies more area.

VIII. CONCLUSION

Each multiplier has its own advantages and disadvantages. The multipliers with bypassing scheme, booth encoding, and tree structure has the less delay as compared to the array multiplier but with the speed increment the gate count i.e. area of the multiplier also increases.

As we increase the no. of input bits the Wallace tree multiplier has the minimum delay and the array multiplier has the maximum delay but in terms of area the column bypass multiplier performs best and the Wallace tree multiplier performs worst.

REFERENCES

[1]Jan M. Rabaey, Anantha Chandraprakashan, Borivoje Nikolic, "Digital Integrated Circuits – A Design Perspective" , PHI learning private limited, 2nd edition.

[2] Neil H.E. Weste, David Harris, Ayan Banerjee, "CMOS VLSI Design – A circuits and systems Perspective", Pearson education, 3rd edition

[3] K.Z.Peckmestzi "Multiplexer based Array multipliers" IEEE Transactions on computers, Vol. 38, no. 1, January 1999

[4] T. Arunachalam and S. Kirubaveni, "Analysis of high speed multipliers", International conference on Communication and Signal Processing 978-1-4673-4866-9/13/$31.00 ©2013 IEEE

[5] Alvin Joseph J. Tang, Joy Alinda Reyes, "Comparative analysis of low power multiplier architectures" 2011 Fifth Asia Modelling Symposium ,978-0-7695-4414-4/11 $26.00 © 2011 IEEE

[6] Rutesh S. Lonkar, Pravin P. Ashtankar, S.S.Shriramwar, "Analysis of column bypass multiplier", The International Journal of Computer Science & Applications (TIJCSA), ISSN – 2278-1080

[7] Sunjoo Hong, Taehwan Roh, and Hoi-Jun Yoo, "A 145 uW 8*8 parallel multiplier based on optimized bypassing architecture", 978-1-4244-9474-3/11/$26.00 ©2011 IEEE