IOSR Journal of Electronics and Communication Engineering (IOSR-JECE)
e-ISSN: 2278-2834,p- ISSN: 2278-8735.Volume 10, Issue 1, Ver. III (Jan - Feb. 2015), PP 79-86
www.iosrjournals.org

Development of an Algorithm for 16-Bit WTM

[1]Sravanthi.kantamaneni, Asst Professor, [2]Dr.V.V.K.D.V.Prasad, Professor,
[3] Veera Vasantha Rao.Battula, Asst. Professor

Abstract: *Binary Multipliers plays an important role in digital circuits. There are many methods for generating a Simple binary multiplication and some of them are like Ripple carry array multipliers, Row adder tree multipliers, Partial product LUT multipliers, Wallace trees, Booth recoding etc.,. Our project mainly concentrates on 8x8 Wallace tree multiplier. It uses a famous Wallace tree structure which is an implementation of an adder tree designed for minimum propagation delay. Rather than completely adding the partial products in pairs like the ripple adder tree does, the Wallace tree sums up all the bits of the same weights in a merged tree. Usually full adders are used, so that 3 equally weighted bits are combined to produce two bits: one (the carry) with weight of* ***n+1*** *and the other (the sum) with weight n. Each layer of the tree therefore reduces the number of vectors by a factor of 3:2. A conventional adder is used to combine these to obtain the final product. The benefits of the Wallace tree is that there are only* ***O****(log****n****) reduction layers, and each layer has* ***O****(1) propagation delay. As making the partial products is* ***O****(1) and the final addition is* ***O****(log****n****), the multiplication is only* ***O****(log****n****), not much slower than addition (however, much more expensive in the gate count). Naively adding partial products with regular adders would require* ***O****(log^2****n****) time. Our project is to develop 8 x 8 Wallace tree multiplier using VHDL and will be simulated with the help of XILINX simulator and verified on Spartan-3E FPGA circuit board.*

I. Introduction

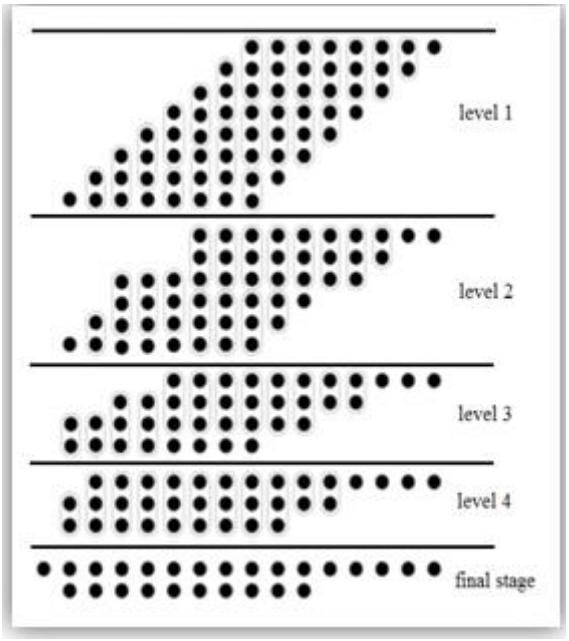
Recent advancements in mobile computing and multimedia applications demand for high performance and low-power consuming VLSI (very large scale integrated circuit) Digital Signal Processing (DSP) systems. One of the most important components of DSP systems is a multiplier. Multiplication is basically shift and add operation. Usually in a DSP system, multiplier units consume large amount of power and cause most of the delay compared to other units like adders. Depending on size of the inputs (2 X 2 bit, 4 X 4, 8 X 8 etc.,) the number of steps a normal binary multiplier takes to compute the product increases drastically. Larger the steps of calculation larger will be the delay as well as the power consumption. Also area occupied by the multiplier on a FPGA (Field Programmable Gate Array) increases. Hence various algorithms have been developed in order to achieve lesser complexity in computation involving minimum calculation steps, which in turn can reduce delay, power and area constraints of multipliers.

The Wallace tree has three computation steps:

1. Generation of Partial products – multiplying each bit of one binary input with every bit of the other binary input. If each input has **n**-bits the result of this step will give us $\mathbf{n^2}$ number of binary bits called 'Partial products' distributed in **n**-rows and **2n**-columns. This step is very same as what we do to multiply two numbers by hand.

2. Reduction of partial products – the partial products are to be added according to their place values (or 'weights') using half adders and full adders until only two rows of partial products are left.

3. Last stage addition – remaining two rows will be added using a conventional adder to get final result of multiplication.



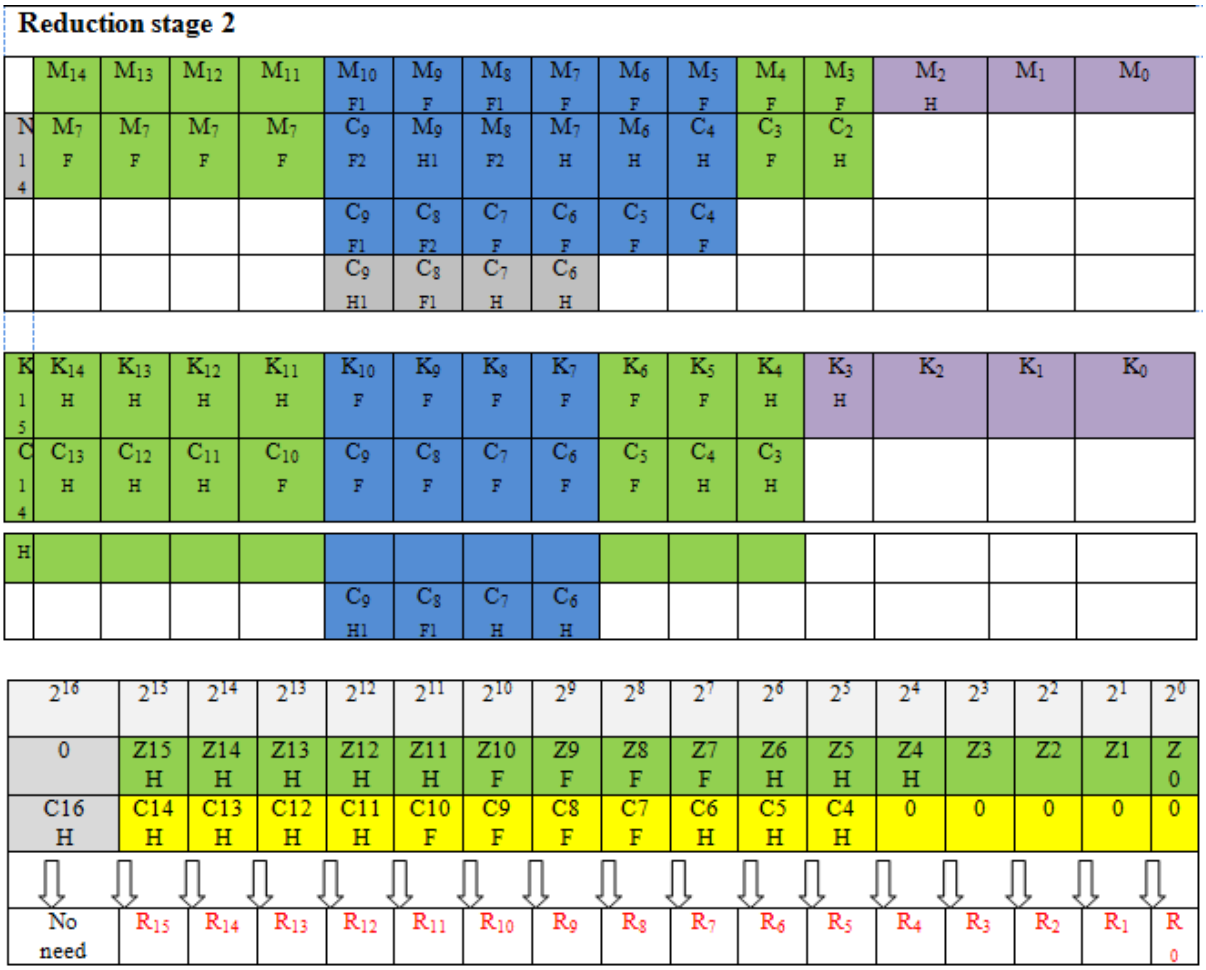
Wallace tree multiplier reduction stages for 8X8 multiplication [1]

Generation of the Partial Products	a_7	a_6	a_5	a_4	a_3	a_2	a_1	a_0
	b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0
	a_7b_0	a_6b_0	a_5b_0	a_4b_0	a_3b_0	a_2b_0	a_1b_0	a_0b_0
	a_7b_1	a_6b_1	a_5b_1	a_4b_1	a_3b_1	a_2b_1	a_1b_1	a_0b_1
	a_7b_2	a_6b_2	a_5b_2	a_4b_2	a_3b_2	a_2b_2	a_1b_2	a_0b_2
	a_7b_3	a_6b_3	a_5b_3	a_4b_3	a_3b_3	a_2b_3	a_1b_3	a_0b_3
	a_7b_4	a_6b_4	a_5b_4	a_4b_4	a_3b_4	a_2b_4	a_1b_4	a_0b_4
	a_7b_5	a_6b_5	a_5b_5	a_4b_5	a_3b_5	a_2b_5	a_1b_5	a_0b_5
	a_7b_6	a_6b_6	a_5b_6	a_4b_6	a_3b_6	a_2b_6	a_1b_6	a_0b_6
a_7b_7	a_6b_7	a_5b_7	a_4b_7	a_3b_7	a_2b_7	a_1b_7	a_0b_7	

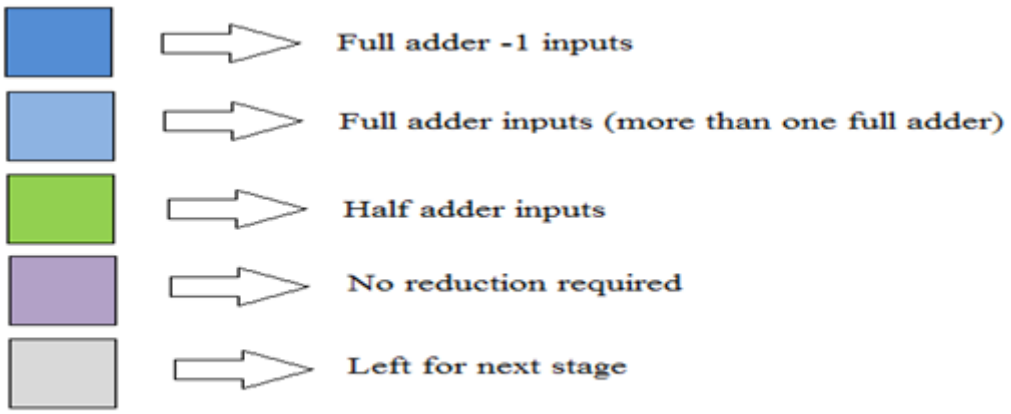
215	2^{14}	2^{13}	2^{12}	2^{11}	2^{10}	2^9	2^8	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0

Reduction stage 1

	S_{14}	S_{13} H	S_{12} F1	S_{11} F1	S_{10} F1	S_9 F1	S_8 F1	S_7 F1	S_6 F1	S_5 F1	S_4 F	S_3 F	S_2 F	S_1 H	S_0
	C_{13} H	C_{12} F1	C_{11} F1	P_4P_7	S_{10} H1	S_9 F2	S_8 F2	S_7 F2	S_6 F2	S_5 F2	C_3	C_2	C_1		
				C_{10} F1	C_9 F1	P_4P_5	P_1P_7	S_7 H	P_0P_6	C_4 F	S_4 H	P_0P_3			
				C_{10} H1	C_9 F2	P_3P_6	C_7 F1	C_6 F1	C_5 F1	C_4 H					
						P_2P_7	C_7 F2	C_6 F2	C_5 F2						
							C_7 H1								



Final Result Bits



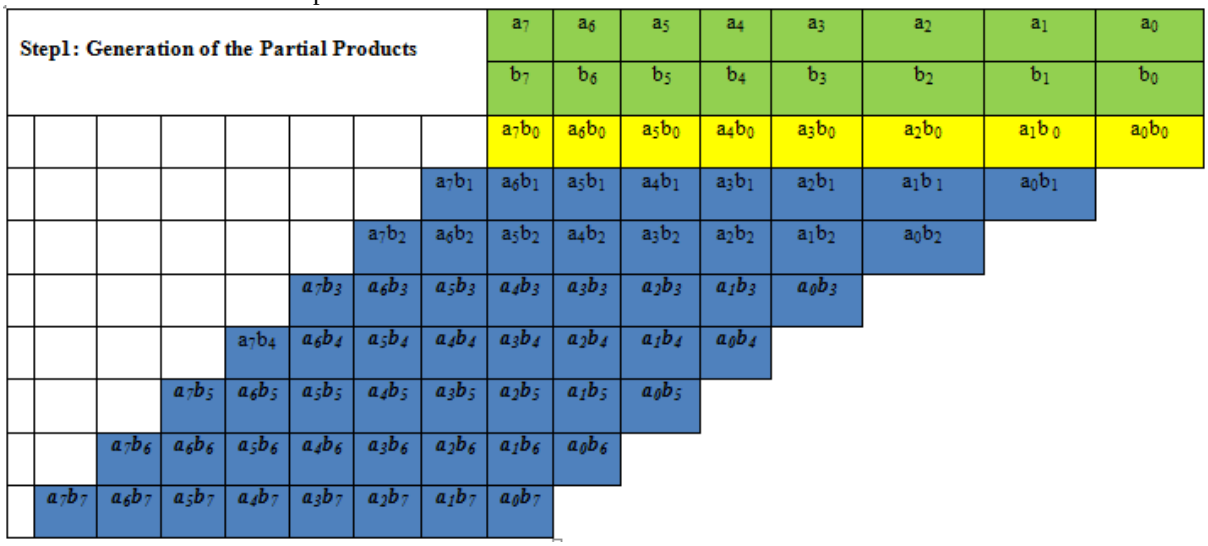
Wallace tree reduction stages for 8 bit multiplication.

Improvements to the Algorithm:

First stage:

It is evident that every multiplication in Wallace tree algorithm is done in three logical stages. They are the partial product generation, the reduction stages and the last stage addition using a conventional adder. If we recall the description for Partial product generation, the first step, we are doing it by use of AND gates. If two 8 bit numbers say A and B are to be multiplied, the algorithm starts at the least significant bits (LSBs) of A and B. LSB of B say b_0 will be AND with all the eight bits of A from a_0 to a_7. This gives us a single row of partial

products. The second row of PPs will be generated when all bits of A will be AND with b_1. Similarly, every row will be formed due to AND operations.



Rows of partial products of 8 bit multiplication

Every partial product of first row has b_0. Every second row element has b_1. Similarly every partial product of n^{th} row will have the common AND input b_{n-1}. So, we can write mathematically the first row as $(a_7\ a_6\ a_5\ a_4\ a_3\ a_2\ a_1)*b_0$. Any n^{th} row can be written as $(a_7\ a_6\ a_5\ a_4\ a_3\ a_2\ a_1)*b_{n-1}$.

We know that all the above bits are binary digits i.e. either 0 or 1. Hence two possibilities exist.

If $b_0 = 0$:

Then the first row $(a_7\ a_6\ a_5\ a_4\ a_3\ a_2\ a_1)*b_0$ will be equal to
$(a_7\ a_6\ a_5\ a_4\ a_3\ a_2\ a_1)*0 = (0\ 0\ 0\ 0\ 0\ 0\ 0\ 0)$

If $b_0 = 1$:

Then the first row $(a_7\ a_6\ a_5\ a_4\ a_3\ a_2\ a_1)*b_0$ will be equal to
$(a_7\ a_6\ a_5\ a_4\ a_3\ a_2\ a_1)*1 = (a_7\ a_6\ a_5\ a_4\ a_3\ a_2\ a_1) = A$
Hence any row will be equal to the Multiplicand A or it will be a row full of Zeros.

Keeping the above fact in mind we can use another way to generate partial products for our need without using n^2 number of AND gates (n is the size of inputs). This method is given below. Any n^{th} row will be 0 or A based on the value of common input b_{n-1} of that row.
Row 'n' = 0, if b_{n-1}=0
Row 'n' =A, if b_{n-1}=1

Consider the following example to comprehend this new logic in a better way.
Let A= 11111111 and B=10011011. A is multiplicand and B is multiplier as usual.
The rows of partial products for the multiplication A X B are:
Row 1 = b_0 AND (11111111) = 1 AND (11111111) =11111111 =A
Row 2 = b_1 AND (11111111) = 0 AND (11111111) =11111111 =A
Row 3 = b_2 AND (11111111) = 0 AND (11111111) =00000000 =O
Row 4 = b_3 AND (11111111) = 1 AND (11111111) =11111111 =A
Row 5 = b_4 AND (11111111) = 1 AND (11111111) =11111111 =A
Row 6 = b_5 AND (11111111) = 0 AND (11111111) =00000000 =O
Row 7 = b_6 AND (11111111) = 0 AND (11111111) =00000000 =O
Row 8 = b_7 AND (11111111) = 1 AND (11111111) =11111111 =A
(Or)
Simply we can write:
Row 1 = A since b0= 1
Row 2 = O since b1= 0
Row 3 = O since b2= 0
Row 4 = A since b3= 1
Row 5 = A since b4= 1

Row 6 = O since b5= 0
Row 7 = O since b6= 0
Row 8 = A since b7= 1

Advantage:

In case of previous method for a n-bit multiplier, the first stage will require a total of n^2 number of AND gates. But with the new method the task of partial product generation will be done by just n number of steps instead of n^2 steps. It is important to note that while describing the multiplier in VHDL code, each of the n^2 AND operations have to be written manually. For a 32 bit- multiplier it requires 1024 steps to be written for simple AND operations. Instead, with the new modification, only 32 steps are to be written which will save a lot of energy and time to the design engineer during development of the code. So, we have adopted the latter method in designing the WTM system.

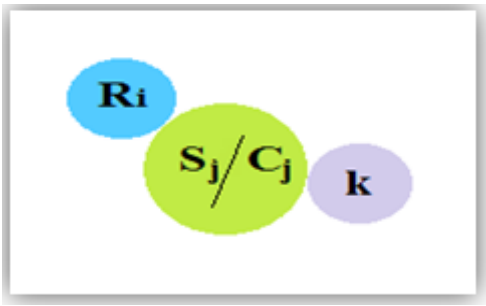
Representation of signals:

As described in an algorithm, we have to use different variables to indicate the partial products in different levels of reduction stages. For example, we used a, b in first stage, then P0, P1 in the next and later S, C, M, N etc. in the figures of chapter 2. These variables are of our choice and we must make sure that PPs in different levels of reduction do not have the same representation. It means that designer has to make a note of which variables he is using in what stage of reduction, clearly and without confusion.

There is another sound drawback of representing variables (or 'signals' with respect to VHDL coding) using normal alphabets like A, B, M or N etc. Let us assume that we have come across a signal N2 while verifying the design. We cannot readily identify which reduction stage this signal N2 belongs to. We must go through the code once again form start and locate where N2 has its origin. Imagine a 32 bit multiplier which will have a very large number of such signals. To go through the code every other time to know about a signal, it is a tremendous burden for the designer. So, it is of high importance that we have a proper representation scheme for signals or variables. We must be able to identify from the name of a signal or variable several aspects. They are:

1. The reduction stage to which it belongs
2. The column or the weight of the partial product
3. Whether it is a SUM bit or CARRY bit
4. If more than one sum and carry bits are present in the column, then position of that bit in the column.

Family	Spartan 3E
Category	General purpose
Device Type	XC3S250E
Package Type	PQ 208: pin Plastic Quad Flat Pack (PQFP)
Speed Grade	-5: High Performance



Scheme of representation of signals

To satisfy the above four requirements, we adopt the above representation scheme.

Ri ⇨ reduction stage number 'i' ; Eg: R1, R2, R4 etc.
Sj ⇨ sum bit of column 'j+1'
Cj ⇨ carry bit of column 'j+1'Weight of partial product = 2^j
k ⇨ place of the signal in the column.

If there are 4 sum bits in the column k takes the values of 1, 2, 3 and 4.

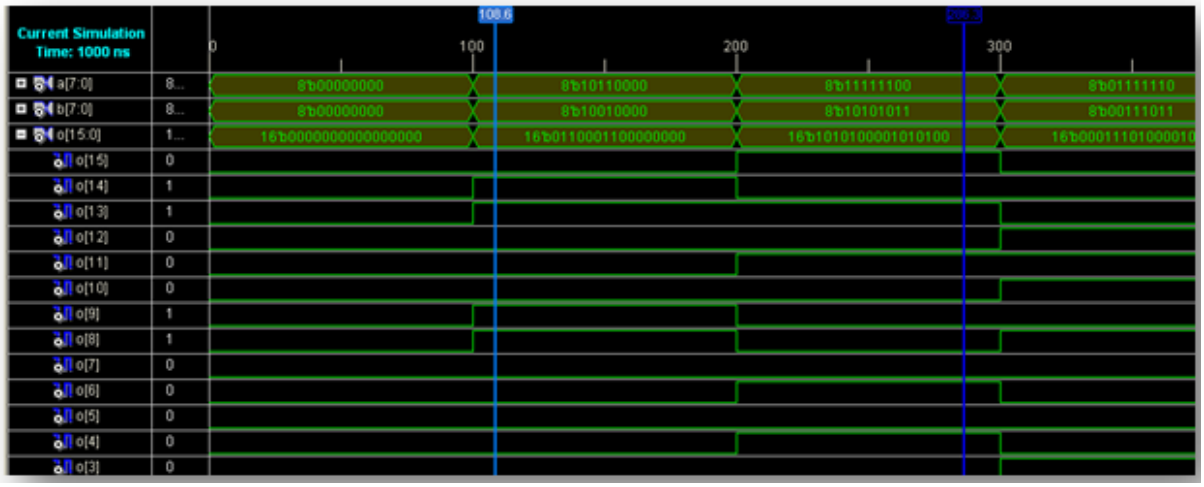
Always the sum bits are taken first and the carry bits are taken next to sums. Reverse will also give the same answer, but to avoid confusion sums are given first priority in any column. Let us consider a column having 4 sums and 3 carries. Let all the bits belong to column number 6 (j=5) of 3rd reduction stage. It will be represented as follows.

R3S5_1
R3S5_2
R3S5_3
R3S5_4
R3C5_1
R3C5_2
R3C5_3

A column of signals of 3rd reduction stage, 5th column

Designing, Synthesis and Results of WTM for the Spartan 3E family FPGA chip

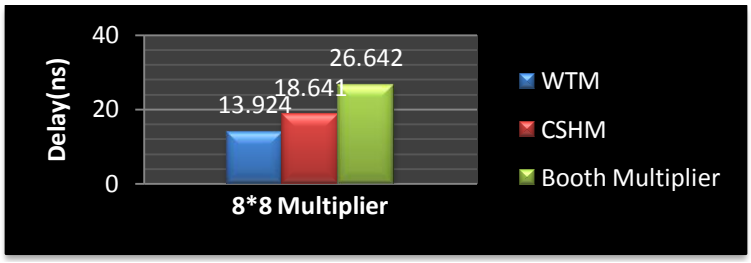
Simulated output of WTM



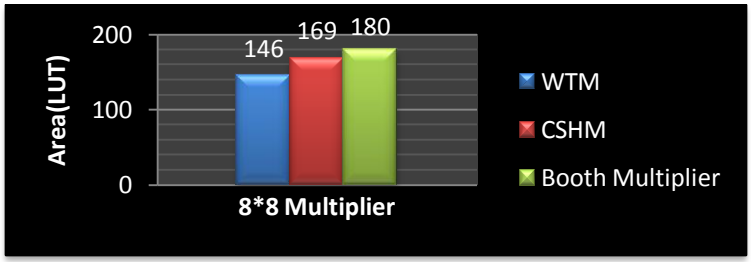
Maximum Combinational Path Delay

S. No	Size of multiplier	Maximum combinational path delay (Nano seconds)
1	4	10.426
2	6	11.921
3	8	13.924
4.	10	14.775
5.	12	14.798
6.	14	16.168
7.	16	16.476

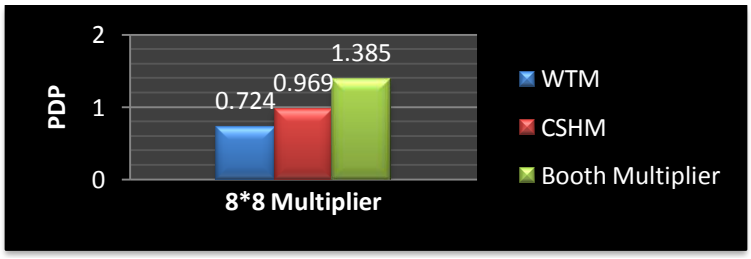
Comparison of Multipliers w.r.t Delay (ns)



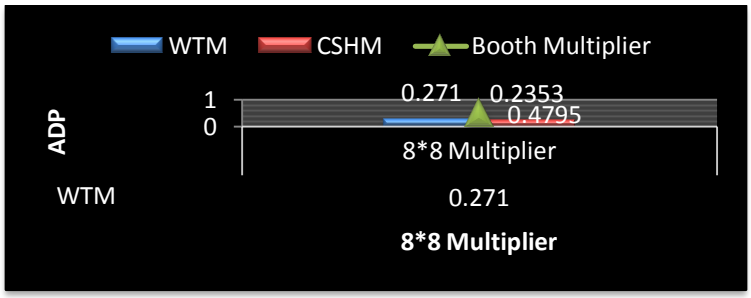
Comparison of Multipliers in terms of Area



Comparison of Multipliers in terms of PDP



Comparison of Multipliers in terms of ADP



Size of the multiplier Vs No. of reduction stages

S. No	Size of multiplier	No. of Reduction Stages	Including last stage
1	4	3	4
2	8	4	5
3	16	6	7

Features of WTM

Parameter	Used	Available	Pre Layout Values (or) Ratio
Number Of Slices 96	371	2448	15%
Number of 4-input LUTs 178	647	4896	13%
Number Of Bonded Input 32	64	158	40%
Number Of Bonded Output 32	64	158	40%
Delay(ns)		16.476	
Slice Utilization Ratio		100	
BRAM Utilization Ratio		100	

Conclusion and Discussion

It can be concluded that Wallace tree multiplier is superior in all respects like Delay, Area and speed. However array multiplier requires more power consumption and gives optimum number of components required, but it can provides a better delay .If we utilize this multiplier as a module of real time applications like FIR Filter ,it can pump up the Filtering action. Further the work can be extended for optimization of said multiplier to improve speed or to minimize the Power Consumption.

References

[1]. C. S. Wallace, A Suggestion for a Fast Multiplier, IEEE Transactions on Electronic Computers, February 1964, EC-13:14–17.
[2]. Vijaya Prakash A. M, Dr. MGR, K. S. Gurumurthy, A Novel VLSI Architecture for Low power FIR Filter, International Journal of Advanced Engineering & Application, January 2011, PP 218 - 224.
[3]. Gary W. Bewick, Fast multiplication algorithms and implementation, The Department Of Electrical Engineering and The Committee on Graduate studies of STANFORD UNIVERSITY, February 1994. PP 8 - 16.
[4]. J. Bhasker, AVHDL Primer, Third Edition, Pearson Education, 2007, PP-21 to 50, 88 to 101
[5]. John F. Wakerly, Digital Design Principle and Practices, fourth edition, Prentice Hall Pearson Education, 2009, PP 235-250, PP 786-795
[6]. http://en.wikipedia.org/wiki/Wallace_tree, http://en.wikipedia.org/wiki/FPGA.
[7]. Multiplication in FPGA's "The performance FPGA DESIGN specialist"