Implementation of Pipelined Booth Encoded Wallace Tree Multiplier Architecture

Rahul D Kshirsagar, Aishwarya.E.V., Ahire Shashank Vishwanath, P Jayakrishnan

School of Electronics Engineering, VIT University, Vellore, India

***Abstract*— The Booth multiplier is a very fast multiplier with minimum latencies. In this paper, a typical architecture of Booth Encoder and Wallace tree is presented, In which we have implemented pipelining at the intermediate nodes of the modules present in it. The architecture comprises of four modules, they are as follows, One's Complement generator, Booth Encoder, Partial product generator and Wallace tree adder accompanied by Ripple carry adder respectively. The Wallace tree adder and Booth multiplier are typically used for high speed computations. One such application is a DSP processor. In this paper we have designed a four stage pipelining at the intermediate nodes mentioned above. This will help in performing many arithmetic operations simultaneously and hence increase the speed as well as computation of simultaneous inputs. The design is implemented in Verilog HDL. The simulation is done on Cadence NC Sim while the synthesis is carried out in Cadence RTL Compiler using TSMC 45nm slow.lib.**

***Index Terms*— Booth Encoder, Partial product generator, Wallace structure.**

I. INTRODUCTION

In this paper a pipelined four stage Booth Encoded Wallace tree Multiplier implemented. The multiplication operation is one of the most complex arithmetic operation as it involves a lot of additions and carry propagations. There are four basic combinational units which do the computations in our module are as shown in Fig 1. For the given equation Z = MD × MR, The two operands MD and MR involved in multiplication are termed as Multiplicand and Multiplier respectively [1].

The conventional multiplication methodology involves generation of partial products and finally adding all these products in a regular matrix formation. While performing MD × MR multiplication it results in 2N bit result. Booth encoded multiplier provides high speed algorithm for multiplying large numbers. Also in a modified booth encoded multiplier the number of partial products can be reduced to (MD/2 -1) partial products. A modified booth multiplier version has been used for multiplier generation circuit. Various types of adders such as carry save adder, carry look ahead adder, Ripple carry adder, Wallace tree adder and Dadda tree adder can be used effectively for the generation of partial products [2].

The speed at which the addition of the partial products is carried out determines the speed of the circuit. We have used Wallace tree algorithm for high speed computation and Ripple carry adder is used at the end for final addition of the carry propagated due to various column additions taking place.

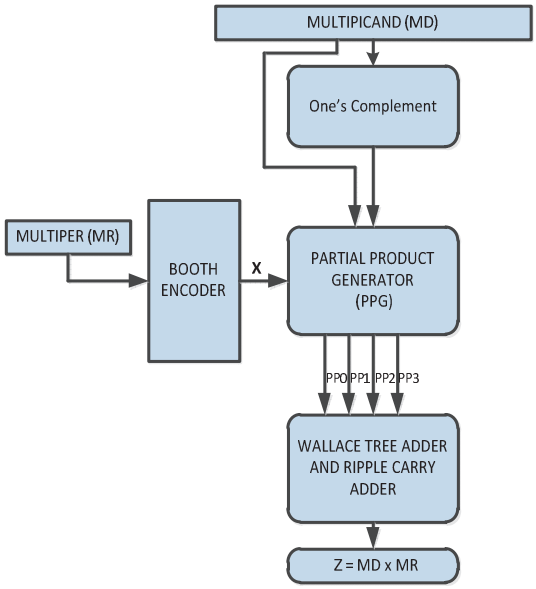

Figure 1. Booth Encoded Wallace Tree Multiplier

II. ARCHITECTURE OF BOOTH ENCODED WALLACE TREE

The Booth Encoded Wallace Tree Architecture is broken into four modules and each module has been explained in the following subsections.

i) One's Complement:

It is one of the basic requirements for signed multiplication and it is obtained by simply complementing the bits of MD. The Partial Product Generator requires this to differentiate the data required between the signed and unsigned multiplication.

ii) Booth Encoder:

The main purpose of using booth encoder is to decrease the number of partial products as compared to basic booth multiplier [3]. The conventional multiplication

Position	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PP_0						s_0'	s_0	s_0	$p_{0,7}$	$p_{0,6}$	$p_{0,5}$	$p_{0,4}$	$p_{0,3}$	$p_{0,2}$	$p_{0,1}$	$p_{0,0}$
PP_1					1	s_1'	$p_{1,7}$	$p_{1,6}$	$p_{1,5}$	$p_{1,4}$	$p_{1,3}$	$p_{0,2}$	$p_{1,1}$	$p_{1,0}$		cor_0
PP_2			1	s_2'	$p_{2,7}$	$p_{2,6}$	$p_{2,5}$	$p_{2,4}$	$p_{2,3}$	$p_{2,2}$	$p_{2,1}$	$p_{2,0}$		cor_1		
PP_3	1	s_3'	$p_{3,7}$	$p_{3,6}$	$p_{3,5}$	$p_{3,4}$	$p_{3,3}$	$p_{3,2}$	$p_{3,1}$	$p_{3,0}$		cor_2				
PP_4				MP			TP	major		cor_3			TP	Minor		
	P_15	P_14	P_13	P_12	P_11	P_10	P_9	P_8	P_7	P_6	P_5	P_4	P_3	P_2	P_1	P_0

Figure 2. 8*8 Partial Product matrix

methodology for Z = MD x MR yields a minimum of B (where B is the number of bits present in MR) partial products to be formed. With the use of Booth encoder one can multiply not only positive but also negative numbers and at the same time reduce the number of partial products [4]. Consider two N-bit numbers A and X. These are represented in two's complement form, and is shown in equation 1 and 2.

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

$$X = -x_{N-1}2^{N-1} + \sum_{i=0}^{N-2} X_i \cdot 2^i \quad (2)$$

In the equation (1) and (2), a_i and x_i denotes i-th bit of the multiplicand and the multiplier respectively. By using booth encoder whose encoding values are given in Table 1 the multiplier ' X ' can be represented as given in equation (3), by considering three bits (triplets) at a time. This triplet is formed by considering three bits of multiplier X, at a time this is shown in Figure 2. In equation (3), x-1=0 and Mi takes any value in {-2,-1, 0, 1, 2}. The output of normal multiplication will be as given in equation 4. The equation 4, P_i denotes the primary output product bit at i^{th} iteration, and this is also represented in equation (5) .

$$X = \sum_{i=0}^{\frac{MD}{2}-1} M_i \cdot 2^{2i}$$

$$X = \sum_{i=0}^{\frac{MD}{2}-1} (-2x_{2i+1} + x_{2i} + x_{2i-1}) \cdot 2^{2i} \quad (3)$$

$$Z = A*X = \sum_{i=0}^{2MD-1} P_i \cdot 2^i \quad (4)$$

The equation (4), P_i denotes the primary output product bit at i^{th} iteration, and this is also represented in equation (5)

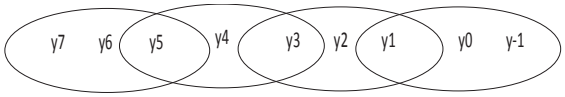

Figure 3. Group bits into triplets for Modified Booth Encoding with N=8

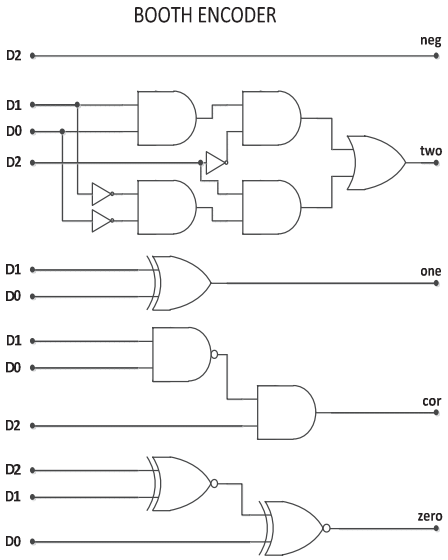

Figure 4. Modified Booth Encoder

$$Z = \sum_{i=0}^{(MD-2)/2} (-2x_{2i+1} + x_{2i} + x_{2i-1}) \cdot 2^{2i}$$

$$Z = \sum_{i=0}^{(MD-2)/2} S_I \quad (5)$$

The equation (5),

$S_i=(-2x_{2i+1}+x_{2i}+x_{2i-1})\,2^{2i}\,X$ and

S_i indicates the scanning of triplets which begins from y_{-1} to the MSB bit with one overlapping bit in each iteration.

The operation –X can be realized by the inversion of the multiplicand and adding 1 to the LSB. The operations 2X and -2X can be realized by shifting X and –X by one bit to left and filling the LSB with zero. In this paper for further discussion an N x N multiplication with N=8 is considered. While computing all the value the resultant partial product matrix for 8*8 modified booth multiplications is given in figure 2.

Table 1. Booth Encoded Values

D2(x_{2i+1})	D1(x_{2i})	D0(x_{2i-1})	Fn	neg	two	one	zero	cor
0	0	0	+0	0	0	0	1	0
0	0	1	+X	0	0	1	0	0
0	1	0	+X	0	0	1	0	0
0	1	1	+2X	0	1	0	0	0
1	0	0	-2X	1	1	0	0	1
1	0	1	-X	1	0	1	0	1
1	1	0	-X	1	0	1	0	1
1	1	1	-0	1	0	0	1	0

A simple gate equivalent circuit of booth encoded table is realized in Fig 4. The three bits of MR (multiplier) are selected at once. Neg bit indicates whether the multiplicand is multiplied with negative weight or not. zero indicates whether to be set as partial product. one can be interpreted as multiplicand that is used directly and finally two tells whether the left shift operation is to be done on the multiplicand or not. To add the final correction bit to the answer cor is the error correction signal that is added to the final partial product.

iii) Partial Product Generator:

The use of Modified Booth Encoder has led to a decrease in the number of partial products. Although the number has decreased but the bit width size of the partial products has increased [5]. The Fig 3 shows a basic multiplexer circuit which generates the first bit PP_0[0] of the partial product PP_0. The first pass is a multiplexer unit which has two inputs the multiplicand MD and its complement MDbar.

The neg signal generated by the Modified booth encoder circuit decides to pass either input or it's complementary to the next stage. The second stage is a one hot multiplexer ,as only one of the input select lines will be high at a time. The input select line is a combination of {two, one, zero}. If two is high then it will pass the last state of the input, if one is high it will pass the output from previous multiplexer to the final output stage else if zero is set high then it sets the output to '0'. Now for every combination of the triplet formed through Figure 3 a partial product PP_0[0] is formed. This entire logic is replicated for all bits of the partial products. Hence at the end we get a 14 bit PP_0. Similarly we get PP_1,PP_2,PP_3 respectively.

The neg signal generated by the Modified booth encoder circuit decides to pass either input or it's complementary to the next stage. The second stage is a one hot multiplexer ,as only one of the input select lines will be

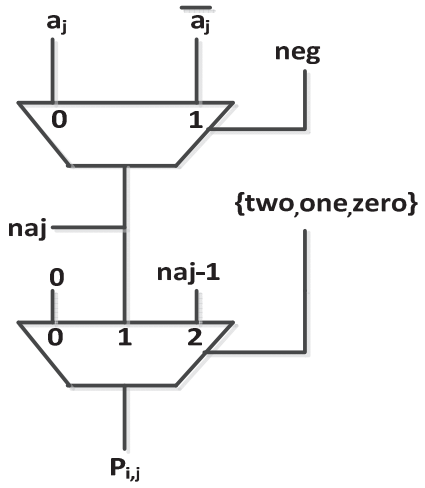

Figure 5. Generating Partial Product Logic Equivalent

high at a time. The input select line is a combination of {two, one, zero}.If two is high then it will pass the last state of the input, if one is high it will pass the output from previous multiplexer to the final output stage else if zero is set high then it sets the output to '0'. Now for every combination of the triplet formed through Fig 2 a partial product PP_0[0] is formed. This entire logic is replicated for all bits of the partial products. Hence at the end we get a 14 bit PP_0. Similarly we get PP_1,PP_2,PP_3 respectively.

iv) Wallace Tree Structure

A Wallace tree architecture implements a faster addition as the data bits need not wait for the previous data to appear at the input. Consider a particular column in which all the bits of the partial products present in that column are added together the carry which is generated is not propagated. A set of matrix will reduce the iterations to a minimum value. The final stage is a simple carry adder which will add all the carry propagated into a final value. At this point even the correction bits are also added so as to minimize the computing error.

The Wallace tree architecture when used in conjunction with booth encoder leads to high speed computation and yields higher efficiency [6].

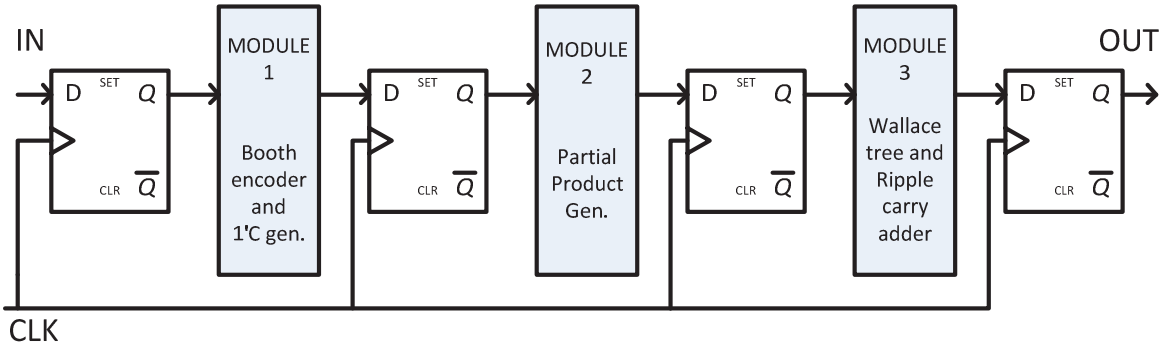

Figure 6. Linear pipelining implementation in Booth Encoded Wallace Tree Multiplier

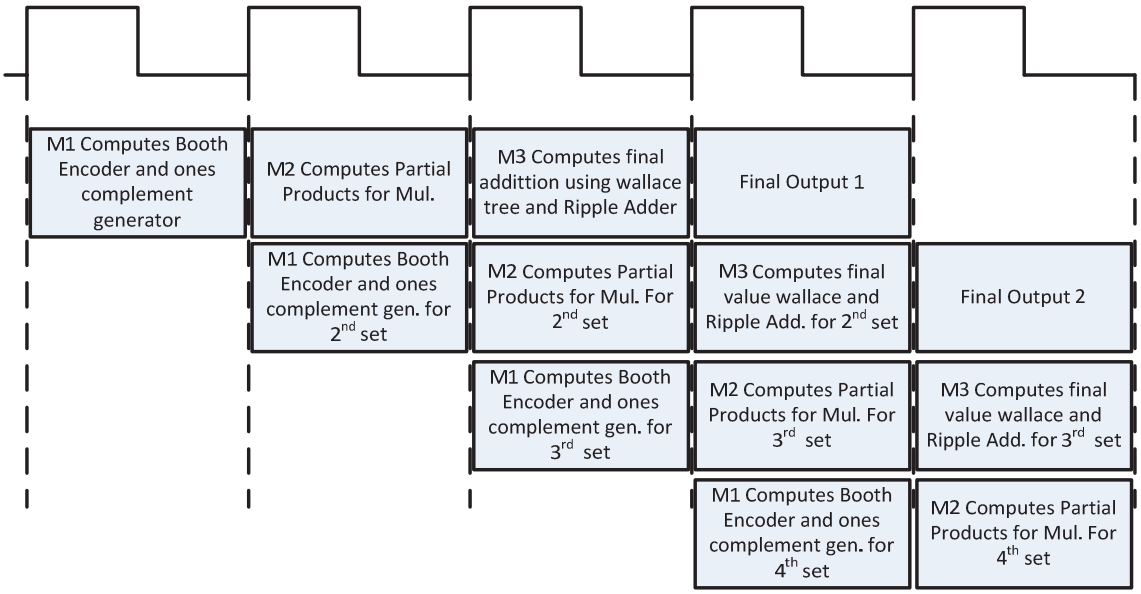

Figure 7. Typical flow in the architecture

III. LINEAR PIPELINING IMPLEMENTATION

Pipelining in Computer Architecture can be defined as Fetching the next instruction while executing the current instruction. Pipelining not only improves the efficiency of the system but also speeds up the process [7]. It is classified into two types as Linear pipelining and Non linear pipelining. In linear pipelining we do not have any feedback as it is a simple cascaded structure of modules where output of one module is drives the input of the next module. The Fig 6 shows how pipelining is incorporated.

The Non linear pipelining has feedback from one of the modules or a feed forward path to the next module. We have implemented a 4-stage linear pipelining by introducing latches at every intermediate stages to achieve temporal parallelism. These latches play a very crucial role. Firstly, they provide isolation. Due to isolation there will not be any glitches in the output. Secondly, a pure combinational logic circuit is converted to sequential circuit. Thus, any FSM Controller which is controlling this multiplier module need not wait for the entire combinational unit to complete its operation and yield output. The only care that needs to be taken is propagation delay of combinational logic circuit should be less than that of the clock period applied.

The Controller can give the next set of inputs on the next clock edge. In this way a series of inputs can be given continuously. For a K stage Pipelined architecture the first output is obtained after K clock cycles and the next output is obtained at K+1 clock cycles and so on. The Fig 7 shows a typical structure of linear pipelining and the corresponding fetch and decode operation which is relevant to our module has been explained.

Figure 7. Linear Pipelined Output

IV. Simulation and Results

The Booth Encoded Wallace tree Multiplier was successfully designed and simulated using Cadence NCsim for the functional correctness. The synthesis is done in Encounter® RTL Compiler [8] v09.10-s233_1 using 45nm TSMC slow.lib technology file. The constraints were applied from setting constraints file [9].

The RTL compiler has successfully created synthesized top level module booth_main this is shown in Figure 8. The corresponding synthesized netlist and design constraint file ie the .sdc were also obtained and it can be further used for the backend design and timing verification. The final Synthesis results are summarized below. The Table 2 provides power dissipation for top as well as individual modules. The corresponding cell area has been given in Table 3.

Table 2. Power Values for Pipelined Architecture

Instance	Cells	Leakage Power (nW)	Dynamic Power (nW)	Total Power (nW)
booth_main	256	44.368	161526.355	161570.723
df	57	9.794	1974.415	1984.208
p1	21	2.305	1180.353	1182.658
p2	21	2.305	1430.519	1432.824
p3	21	2.305	1719.009	1721.314
p4	21	2.305	1511.026	1513.331
e1	2	0.116	73.429	73.545
e2	8	0.592	566.938	567.530
e3	8	0.592	580.636	581.228
e4	8	0.592	563.213	563.804

The maximum operating frequency for pipelined architecture is found to be 256.4760195 Mhz.

Table 3. Area Values for Pipelined Architecture

Instance	Cells	Cell Area
booth_main	256	1008
df	57	229
p1	21	64
p2	21	64
p3	21	64
p4	21	64
e1	2	12
e2	8	12
e3	8	12
e4	8	2

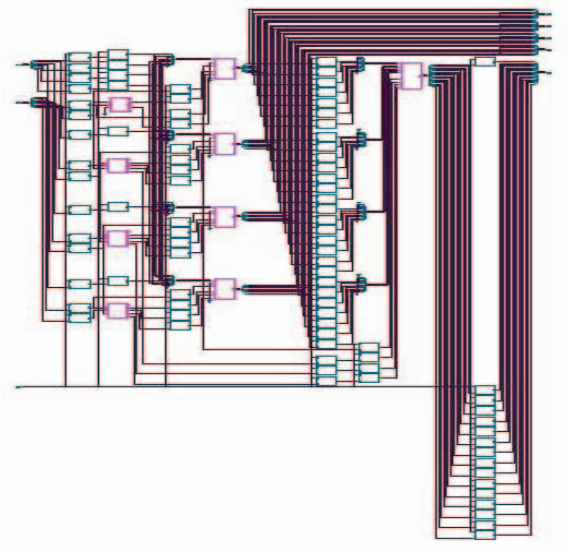

Figure 8. Synthesized Top Level Module

V. CONCLUSION

Thus a four stage Linear Pipelining was successfully implemented in Booth Encoded Wallace Tree Multiplier. It is found that after four clock cycles the first output appears and remaining results appear after every clock interval. Due to the introduction of pipelining every module is able to perform its own operation independent of other in a given clock period. Thus a series of arithmetic operations can now be done in consecutive clock cycles.

ACKNOWLEDGEMENT

We sincerely thank our fellow mates Babu Ramki S and Aravind Babu S for sharing their valuable inputs in implementing the Modified Booth Encoder Logic.

REFERENCES

[1] Jalil Fadavi-Ardekani, *M x N Booth Encoded Multiplier Generator Using Optimized Wallace Trees Using Optimized Wallace Trees,* IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol.1, no. 2, June 1993.

[2] Jagadeshwar Rao, Sanjay Dubey*, A High Speed Wallace Tree Multiplier Using Modified Booth Algorithm for Fast Arithmetic Circuits,* IOSR Journal of Electronics and Communication Engineering (IOSRJECE) ISSN: 2278-2834, ISBN No: 2278-8735, Vol. 3, No.1 (Sep-Oct 2012), pp 07-11. http:// www.iosrjournals.org.

[3] Sukanya B, Kothainachiar S, *Booth Encoded Area Efficient Parallel Tree Reduced Multipliers*, International Journal of Communications and Engineering, Volume 03– No.3, Issue: 04, March2012.

[4] Aviral Mittal, *Booth Multiplier Implementation of Booth's Algorithm using Verilog RTL*, VLSI IP :Booth's Multiplier, http://www.vlsiip.com

[5] Jiun-Ping Wang, Shiann-Rong Kuang ,Shish-Chang Liang**,** *High-Accuracy Fixed-Width Modified Booth Multipliers for Lossy Applications*, IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol.19, no. 1, Jan 2011.

[6] Jayant Chowdhary, Vivek Garg, Tushar Negi, Shreya Jain , *Realization Of An 8-bit Pipelined Microprocessor in Verilog HDL*, Computer Engineering and Intelligent Systems, ISSN 2222-1719 (Paper) ISSN 2222-2863 (Online), vol.3, no.7, 2012. http://www.iiste.org

[7] Keshab K. Parhi , David G. Messerschmitt, *Pipeline Interleaving and Parallelism in Recursive Digital Filters-Part I: Pipelining Using Scattered Look-Ahead and Decomposition*, IEEE Transactions on Acoustics, Speech and Signal Processing, Vol. 31,no.7,July 1989.

[8] *Encounter RTL Compiler Version 10.1*, Cadence Designs Systems, 25-April 2011.

[9] *Setting Constraints and Performing Timing Analysis Using Encounter RTL Compiler*, Cadence Designs Systems, April 2010.

BIOGRAPHY

Rahul D Kshirsagar obtained his BE degree in Electronics Engineering from Vidyalankar Institute of Technology affiliated to Mumbai University, Mumbai, India. Currently he is pursuing Mtech in VLSI design from Vellore Institute of Technology, Vellore, Tamil Nadu. Any queries regarding the above can be sent at rahulk3904@gmail.com.

Aishwarya.E.V. received her Bachelor of Engineering degree in Electronics and Communication Engineering from St.Peter's Engineering College, Anna University, India. She is currently pursuing Master of Technology in VLSI Design from VIT University, India.

Shashank Ahire received his Bachelor of Engineering degree in Electronics and Telecommunication Engineering from Maharashtra Institute of Technology University of Pune, India in 2010. Currently he is pursuing MTech in VLSI Design in School of Electronics Engineering at VIT University Vellore.