Dhanya Geethanjali Sasidharan, Aarathy Iyer / International Journal of Engineering Research and Applications (IJERA) ISSN: 2248-9622 www.ijera.com
Vol. 3, Issue 1, January -February 2013, pp.1513-1516

Comparison of Multipliers Based on Modified Booth Algorithm

Dhanya Geethanjali Sasidharan , Aarathy Iyer
Department of Electronics and Communication Engineering Mahathma Gandhi University Kolenchery,India

Abstract

In this paper comparison of different 16 x 16 and 4 x 4 multipliers based on booth algorithm has been presented. Different variations of booth algorithm for recording circuitry and the adder for final compression of partial products are implemented.. The proposed architecture was synthesized using Xilinx tool. Based on the theoretical and experimental estimation, analysis was carried on results such as the amount of hardware resources and delay . Proposed multipliers can be used for high performance applications like signal processing, image processing.

Keywords—Booth Algorithm,Carry Save Adder,Kogge Stone Adder ,Digitl Signal Processing

I. INTRODUCTION

Multiplication is an important fundamental function in arithmetic operations. Multiplication-based operations such as Multiply and Accumulate (MAC) is used in many Digital Signal Processing (DSP) applications such as convolution, Fast Fourier Transform (FFT), filtering and in the arithmetic and logic unit of microprocessors[1]. In many DSP algorithms, the multiplier lies in the critical delay path and ultimately determines the performance of algorithm. The speed of multiplication operation is of great importance in DSP as well as in general processor. Since multiplication dominates the execution time of most DSP algorithms and determines the speed of ALU, there is a need of high speed multiplier[2].

In general, a multiplier uses Booth's algorithm and array of full adders (FAs), or Wallace tree instead of the array of FAs., i.e., this multiplier mainly consists of three parts: Booth encoder, a tree to compress the partial products such as Wallace tree, and final adder. To reduce the number of calculation steps for the partial products, MBA algorithm has been applied mostly where Wallace tree has taken the role of increasing the speed to add the partial products

The rest of the paper is organized as follows. In section two, an introduction to the standard design is given . Section three deals with adders such as carry save adder and kogge stone adder. Section four shows implementation result and the characteristics of parallel multiplier based on both of the booth encodings. Finally, the conclusion will be given in section five in which provides a brief summary of the proposed approach and discussion on scope of future extensions

II. GENERAL STRUCTURE

A.Standard Design

In this section, we discuss basic MAC operation. Basically, multiplier operation can be divided into three operational steps. The first one is booth encoding to generate the partial products. The second one is adder array or partial product compression and the last one is final addition in which final multiplication result is produced. If the multiplication process is extended to accumulate the multiplied result, then MAC consists of four steps.

General multiplier executes the multiplication operation by multiplying input multiplier X and input multiplicand Y. After that current multiplication result is added to the previous multiplication result Z as accumulation step.

The N-bit 2's complement binary number X can be expressed as

$$Y = -2^{N-1}y_{N-1} + \sum_{I=0}^{n-2} 2^i \, y_i \qquad (1)$$

If "Eq. (1)" is expressed in base-8 type redundant sign digit form in order to apply the radix-8 booth's algorithm,
it would be

$$Y = \sum_{i=0}^{((N+1/3)-2)} d_i 8^i \qquad (2)$$

$$d_i \in (\pm 4, \pm 3, \pm 2, \pm 1, 0) \qquad (3)$$

If "Eq. (1)" is expressed in base - 4 type redundant sign digit form in order to apply the radix - 4 booth's algorithm ,it would be

$$Y = \sum_{i=0}^{\left(\frac{N}{2}-1\right)} d_i 4^i \qquad (4)$$

$$d_i \in \{\pm 2, \pm 1, 0\} \qquad (5)$$

Here di refers to the select signals for the partial product. If "Eq. (2)" is used, then multiplication can be expressed as

$$X \times Y = \sum_{i=0}^{((N+I/3)-2)} Y d_i \, 2^{3i} \qquad (6)$$

If these equations are used, then multiplication accumulation result can be expressed as

$$P = X \times Y + Z = \sum_{i=0}^{((N+1/3)-2)} d_i 2^{3i} + \sum_{j=0}^{2N-1} z_i 2^{3i} \quad (7)$$

Here the MAC architecture implemented by "Eq. (7)" is called standard design.

A.Booth Encoders

The modified Booth's algorithm based on a radix-4, generally called Booth-2 [7] is the most popular approach for implementing fast multipliers using parallel encoding [1]. It uses a digit set {0, ±1, ±2} to reduce the number of the partial products to n'= [(n+1)/ 2]. Radix- 4 encoding start by appending a zero to the right of multiplier LSB. Triplets are taken beginning at position x – 1 and continuing to the MSB with one bit overlapping between adjacent triplets

TABLE I
RADIX-2 CODING ALGORITHM

a	b	b'_i	Operation	Comments
0	0	0	0	String of 0's
0	1	+1	+Y	End of 1's
1	0	-1	-Y	Begin of 1's
1	1	0	0	String of 1's

This recoding scheme applied to a parallel multiplier halves the number of partial products so the multiplication time and the hardware requirements decrease. Radix-8 recoding [5], [7] applies the same algorithm as radix-4, but now in this we take quartets of bits instead of triplets. The Booth-3 scheme is based on a radix-8 encoding to reduce this number to n' = [(n+1)/3].

All digit sets {0, ±1, ±2, ±3, ±4} are obtained by simple shifting and complementary operations, except generation of the multiple 3X, which is computed by an adding and shifting operation, 3X = 2X+X and -3X can be generated by complement 3X. Table I depicts the partial product generating recording algorithm in case of Radix – 2 booth encoding and Table II corresponds to Radix - 4 booth encoding .

TABLE II
RADIX-4 CODING ALGORITHM

B2i+1	B2i	B2i-1	Operation	Comments
0	0	0	0	String of 0's
0	0	1	1	End of 1's
0	1	0	1	A single 1
0	1	1	2	End of 1's
1	0	0	-2	Beginning of 1's
1	0	1	-1	A single 0
1	1	0	-1	Beginning of 1's
1	1	1	0	String of 1's

III. FINAL ADDER

A.Carry Save Adder

There are many cases where it is desired to add more than two numbers together. The straightforward way of adding together m numbers (all n bits wide) is to add the first two, then add that sum to the next, and so on. This requires a total of m – 1 additions, for a total gate delay of O(m.log n). Instead, a tree of adders can be formed, taking only O(log m · log n) gate delays.

Using carry save addition, the delay can be reduced further still. The idea is to take 3 numbers that has to be added together, X + Y + Z, and convert it into two numbers 'carry + sum' (C + S) such that X + Y + Z = C + S, and do this in O(1) time. In carry save addition, we refrain from directly passing on the carry information until the very last step.

The important point is that c and s can be computed independently, and furthermore, each Ci (and Si) can be computed independently from all of the other C's (and S's). A carry save adder simply is a full adder with the Cin input renamed to Z, and the Cout output renamed to C

.

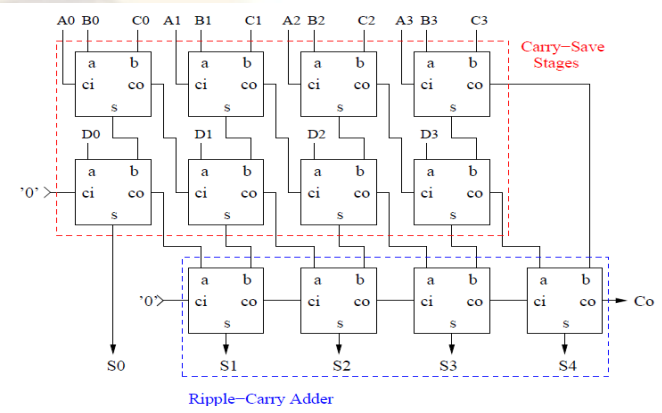

Figure 1.4 bit carry save adder

B.Kogge Stone Adder

KSA is a parallel prefix form carry look ahead adder. It generates carry in O (log n) time and widely considered as the fastest adder and is widely used in the industry for high performance arithmetic circuits. In KSA, carries are computed fast by computing them in parallel at the cost of increased area.

The complete functioning of KSA can be easily comprehended by analyzing it in terms of three distinct parts :

A = 1001 B = 1100 Sum = 10101

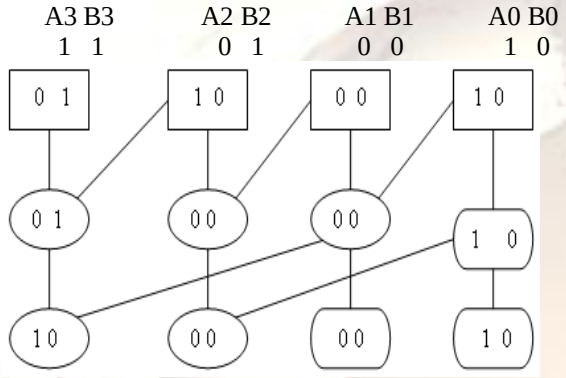

Figure 2. 4 bit kogge stone adder

Pre processing: This step involves computation of generate and propagate signals corresponding too eachpair of bits in A and B. These signals are given by the logic equations below:
pi = Ai *xor* Bi
gi = Ai *and* Bi

Carry look ahead network : This block differentiates KSA from other adders and is the main force behind its high performance. This step involves computation of carries corresponding to each bit. It uses group propagate and generate as intermediate signals which are given by the logic equations below:

Pi:j = Pi:k+1 *and Pk:j*
Gi:j = Gi:k+1 *or* (Pi:k+1 *andGk:j*)

Post processing: This is the final step and is common to all adders of this family (carry look ahead). It involves computation of sum bits. Sum bits are computed by the logic given below:
Si = pi xor Ci-1

IV. EXPERIMENTAL RESULTS

The 16×16 bit parallel MAC based on both the booth encodings (i.e. Radix-2 booth encoding and Radix-4 booth encoding) and some final adders (such as CSA adder and Kogge stone adder) is designed in Verilog and the functionalities of the algorithms are verified by XILINX ISE 13.2i .

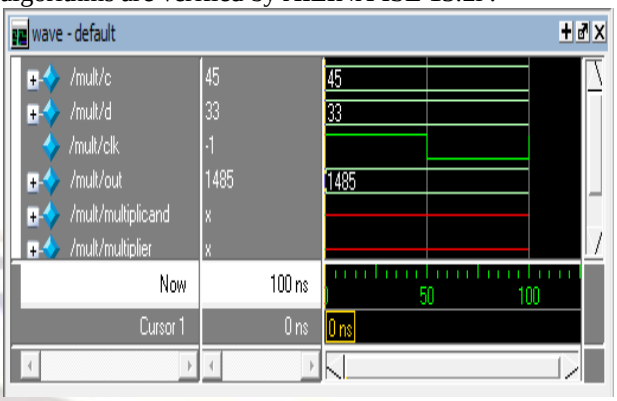

Figure 3. Simulation result of Pipelined 16-bit MAC based on radix-4 modified booth encoder

TABLE III
PERFORMANCE COMPARISON OF 16 BIT MULTIPLIERS

Performance Parameters	16 x 16 bit multiplier		
	array multiplier	*radix 2 multiplier*	*radix 4 multiplier*
Supply Power (mW)	42.10	42.10	42.10
Number of 4 input LUTs	505	525	621
Number of Occupied Slices	290	303	313

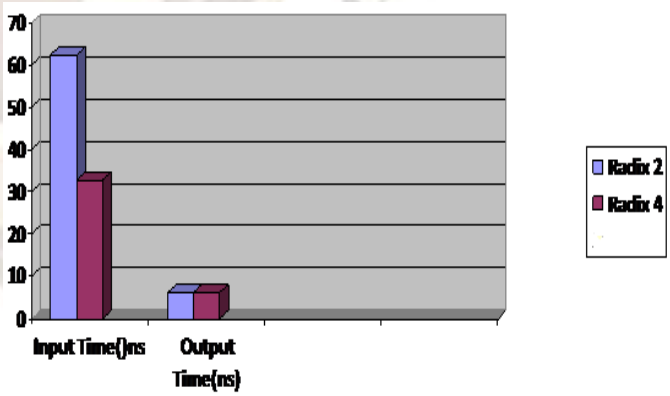

Figure 4 .Timing comparison of 16 X 16 multipliers

TABLE IV
PERFORMANCE COMPARISON OF 4 BIT MULTIPLIERS

Performance Parameters	4 x 4 bit multiplier		
	Radix 2 Multiplier With Kogge Stone Adder	*Radix 4 Multiplier With Kogge Stone Adder*	*Radix 4 Multiplier With Carry Save Adder*
Supply Power (mW)	42.10	42.10	42.10
Number of 4 input LUTs	34	24	62
Number of Occupied Slices	19	12	35

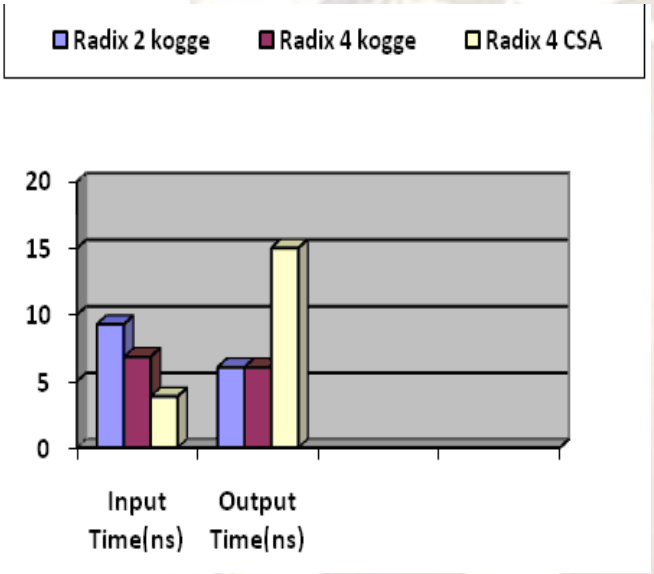

Figure 5 .Timing comparison of 4 X 4 multipliers

From Table III ,it can be seen that for the same supply power number of 4 input LUTs and the number of occupied slices is maximum for Radix 4 multiplier . But from the Figure 4 , for timing analysis it can be inferred that Radix 4 multiplier has better performance than Radix 2 multiplier in terms of speed of operation

Considering 4 x 4 multipliers with different adder ,it can be seen that multipliers with kogge stone adder has better performance in terms of speed of operation than carry save adder. From Table IV ,it can be inferred that multiplier using carry save adder consumes more hardware than than one using kogge stone adder.Here again from figure 5, radix 4 booth multiplier with kogge stone adder has the best performance comparing all the other multipliers in terms of speed of operation.

V. CONCLUSION

Proposed multiplier architecture designs using radix-2 and radix-4 booth algorithm were found to be faster with same amount of power consumption as compared to ordinary multiplier .Radix-4 MBA based multiplier is faster than radix-2 multiplier .

In the case of different adders used in the final addition for multiplication kogge stone adder is found to be faster than ordinary carry save adder . For high speed applications where little amount of logic utilisation over head is permissible the proposed modified booth algorithm architectures can be considered.These multipliers can be incorporated into existing applications like filters, MAC unit, etc. and the performance can be compared. This work can be utilized in any of the following such as in DSP applications, Numerical co-processor, Calculators (pocket, graphic etc), Filtering, Modulation & Demodulation etc.

REFERENCES

[1] Young-Ho Seo and Dong-Wook Kim, "A New VLSI Architecture of Parallel Multiplier Accumulator Based on Radix-2 ModifiedBooth Algorithm", IEEE trans. on VLSI Systems, Vol.18 No. 2, Feb. 2010. R.J. J. F. Cavanagh, "Digital Computer Arithmetic", New York: McGraw- Hill, 1984.

[2] F. Elguibaly, "A fast parallel multiplier–accumulator using the modified Booth algorithm", IEEE Trans. Circuits Syst., vol. 27, no. 9,

[3] C. S. Wallace, "A suggestion for a fast multiplier", IEEE Trans. Electron Comput., vol. EC-13, no. 1, pp. 14–17, Feb. 1964

[4] Fayed and M. Bayoumi, "A merged multiplier-accumulator for high speed signal processing application", Proc. ICASSP, vol. 3, pp.3212–3215, 2002.

[5] R. Cooper, "Parallel architecture modified Booth multiplier", Proc. Inst. Electr. Eng. G, vol. 135, pp. 125–128, 1988.

[6] J.Fadavi-Ardekani, "M x N Booth encoded multiplier generator usingoptimizedWallace trees," *IEEE Trans. Very Large Scale Integr. (VLSI)Syst.*, vol. 1, no. 2, pp. 120–125, Jun. 1993.

[7] Marc Hunger and Daniel Marienfeld, "New self checking booth multiplier", Int. J. Appl. Math, Comput. Sci., Vol.18, No.3, 319–328,2008