I.J. Information Engineering and Electronic Business, 2015, 4, 1-8
Published Online July 2015 in MECS (http://www.mecs-press.org/)
DOI: 10.5815/ijieeb.2015.04.01

A New Partial Product Reduction Algorithm using Modified Counter and Optimized Hybrid Network

Pouya Asadi
Department of Computer, College of Engineering, Varamin-Pishva Branch, Islamic Azad University, Varamin, Iran
p_asadi@iauvaramin.ac.ir

Abstract—In this paper, a new multiplier is presented which uses modified fourteen transistor adder and optimized hybrid counter for partial product reduction step. Conventional adder is modified to improve Wallace tree functionality. Reducing critical path in counter structure can reduce VLSI area in whole multiplier structure. This paper uses a new structure in partial product reduction step to increase speed. Four to two compressors are used in modified Wallace structure to minimize the critical path. In final addition step of algorithm a new carry lookahead network is presented which adds two final operands efficiently. It uses dynamic CMOS in transistor level to reduce power consumption. Proposed multiplier reduces critical path, increases speed and decreases wiring problems in compare with previous algorithms efficiently. A new Booth encoder is presented in radix 16 circuitry. It decreases number of partial products while hardware overhead is minimized.

Index Terms—Adder, Booth encoder, CMOS, Multiplier, VLSI.

I. INTRODUCTION

Multiplier is a central part of an ALU. Most high-performance CPUs rely on hardware multiplication to achieve high speed and low power consumption. This is especially true for DSPs in which the basic building modules are counters, compressors, and multipliers. We have developed a high-speed low power multiplier architecture that is well suited for ALUs [1] and is easily developed to higher order partial product reduction techniques. This allows high-speed multipliers to use a small amount of IC and thus the implementation of high-speed multipliers into a single chip area is possible. The critical path in a pipeline multiplier can be divided into three parts: partial product generator, partial product reduction tree and the final adder [2]. The delay introduced by the partial product generator is less than to the other two parts especially for the large width multipliers. This component latency is also relatively small and is independent of the size of the multiplier. The Dadda high-speed low-power multiplier uses full adders to decrease an M bit operand to an equivalent two M-bit final partial products that is added with a carry skip adder to give the result [1]. It is a fully pipeline algorithm of the multiplier used in the Intel processors [2]. The compressors are a kind of counters whose carries are not chained, so that three bits are taken in and two bits are outputs. As mentioned more clearly in [3], the Wallace multiplier also uses counters in the addition step. This paper shows a modified algorithm that greatly decreases the number of compressors in Dadda multipliers. In this paper, all input operands are assumed to be M-bit two's complement partial products. For the modified Dadda reduction method, once the partial product tree (of M^2 bits) is generated, adjacent lines are summed into different collections of tree. Each group of bits is decreased by using compressors applying a counter to each column that uses two bits, and removing any even rows to the next step without summing. This compression algorithm is applied to each step until only two partial products remain. The final two partial products are added with a carry skip counter. This algorithm is presented by the modified 8-bit-by-8-bit Dadda multiplier shown in Fig. 1. Squares show the two column collections. The summation is done in four steps (each with the wiring of one counter) with a total of 35 counters and 15 compressors. The second step will need an 8-bit carry lookahead adder. In comparison with Dadda algorithm, the Wallace method [4] does the fastest decrease needed at each step. To determine how much compression is needed, the wiring of each step is computed by summing from the previous step. It has a height of three columns. The delay introduced by the partial product reduction and the counter constitutes a large critical path in the multiplier, which the delay introduced in the rows of the tree is about half of the final carry skip adder. Therefore, major concern to the speed of a multiplier will result for improvement in partial product reduction and final addition steps. In this paper, we discuss the performance of various known algorithms for compressing the input operands. Modified Booth encoder block diagram is shown in Fig. 1. The number representation for the floating point multiplication used in the algorithm is based on [5]. Therefore, both input operands are double precision numbers. With the progress of electronic portable devices, the requirement of low power equipment is getting more concern in recent years [1]. The primary rule in electronic mobile devices is to extend functional hours without changing the battery technology. Although advanced technology improves battery life to operate for longer time, the complicated calculations in the high-end portable devices have high power consumption and are critical for low power architectures. Low power structure can be implemented at system, VLSI, technology, architecture and circuit implementation. Power reduction can be significant if the low power design is programmed in the earlier step at circuit level. Optimizing logic structure in circuit implementation is also critical in low power architectures. To reach this goal, software simulation requires to be extended. In section two to six we have presented a new multiplier architecture which has reduced three different electronic parameters. Presented multiplier reduces transistor count, latency and area efficiently in compare with conventional multipliers [1-4]. In section two, the partial

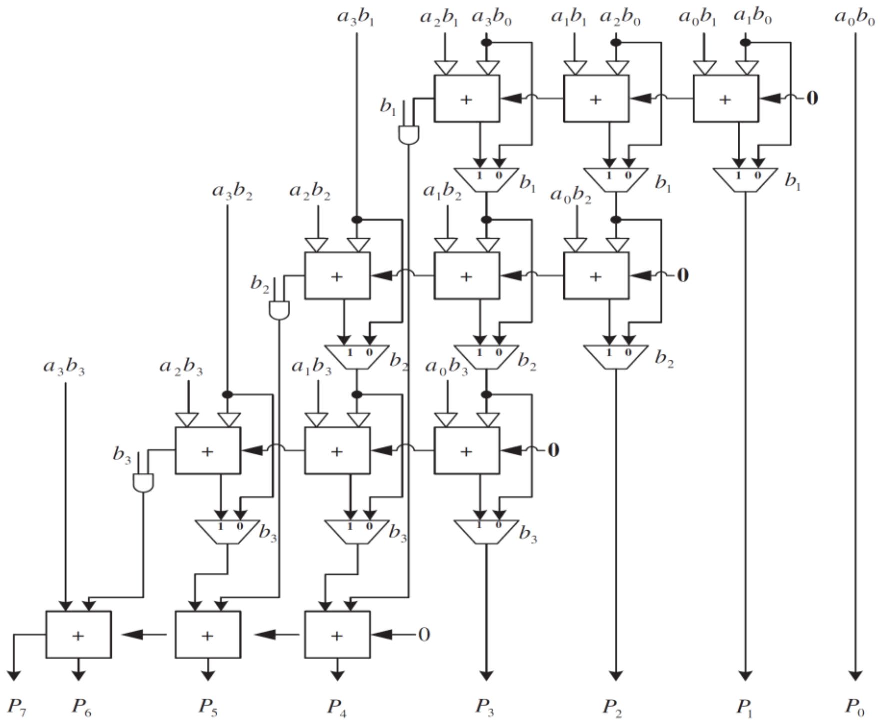

Fig.1. Modified Booth encoder block diagram

product reduction algorithm for presented multiplier is described. In section three and four, counter tree, which is needed for partial product reduction algorithm, and modified Booth generator, which is necessary, for partial product generation step is proposed. In section five, counter array is implemented using a dynamic architecture. In section six, hybrid addition using carry hybrid adder is designed.

II. The Partial Product Reduction Algorithm For Presented Multiplier

Because the floating-point multiplier is dedicated for a floating point DSP, the necessity for increasing speed and decreasing power is very high. For the floating-point multiplier to be synthesized to run consistently at the speed of 230 MHz the parallel method is selected in the algorithm, in which critical path is separated into several steps, and the operands are used between the steps to save the wiring path. Fig. 2 shows carry network for final addition. Because the wiring path length of the input operands through the tree structure is decreased in two-clock cycle, the clock intervals are increased significantly. The operational frequency is greater than conventional logic tree. The faster the critical path of tree, the shorter the signal path between the output operands, and the greater the partial product path dealt with, concludes higher efficiency. As a result, the operational speed of Wallace algorithm is increased significantly by the parallel algorithm. The structure is very appropriate for adoption of the parallel technique because of efficiency of their method. Let $a_{n-1}a_{n-2}\ldots a_0$ and $b_{n-1}b_{n-2}\ldots b_0$ be two m-bit binary numbers with a sum of $s_{n-1}s_{n-2}\ldots s_0$. The carry-hybrid algorithm computes the s_i's by

$$S_{i+1} = \bar{a}_i \oplus \bar{b}_i \oplus c_i$$

$$C_{i+1} = \bar{a}_i\bar{b}_i + (\bar{a}_i \oplus \bar{b}_i)c_i$$

$$= \bar{a}_i\bar{b}_i + (a_i\bar{b}_i + \bar{a}_i b_i)c_i$$

$$= \bar{a}_i\bar{b}_i + a_i\bar{b}_i c_i + \bar{a}_i b_i c_i$$

$$= \bar{a}_i\bar{b}_i + \bar{b}_i c_i + \bar{a}_i c_i \quad (1)$$

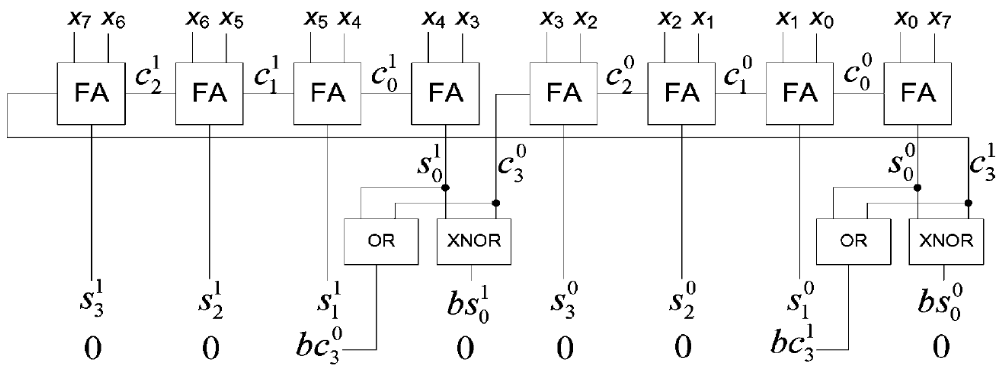

Fig.2. Carry network for final addition

For large m, the above carry calculation is difficult to implement due to the practical restrictions on fan-in and fan-out. In order to decrease the complexity, it is common practice to group outputs into adders [6].

III. Counter Tree

The counter suitable for use in a digital processing algorithm was implemented using these basic structures. In addition to normal add, subtract, multiplication and division needed for the ALU module, we also had to design an absolute value subtraction algorithm. Sign magnitude arithmetic is not particularly well designed for complex operations, since another computation stage is needed whenever the calculated outputs are not positive. We decided to use the two's complement method for doing the arithmetic calculations, which in turn computes the following extra stages. Fig. 3 shows six to two counter at transistor level.

1. Exor logic is needed prior to summation (converting from two's complement to sign magnitude).
2. Compute carry to perform two's complement addition.
3. Exnor logic is needed after summation (converting back to two's complement).

This was not a complex algorithm in our case since we were calculating 16-bit numbers. Taking the overall wiring and speed into computations, we decided to design the 16-bit two's complement adder. The computed carry introduced only four extra gate delays at the carry production step.

$$Z = -2^{2n-1} + x_{n-1}y_{n-1}2^{2n-2} + \sum_{j=0}^{n-2} \overline{x_{n-1}y_j}\,2^{n+j-1} \tag{2}$$

$$+\sum_{i=0}^{n-2} \overline{y_{n-1}x_i}\,2^{n+i-1} + \sum_{i=0}^{n-2}\sum_{j=0}^{n-2} x_i y_j 2^{i+j} + 2^n$$

After implementing gate expressions and control logic we found the counter to be slower than required by the frequency (clock speed = 17.68 MHz). The counter was originally implemented as a step in an ALU. Throughput is more important than delay. We decided to divide the counter in three steps. This led to an improved implementation of a parallel counter. Hybrid lookahead adder as shown in Fig. 5 for a Wallace tree of the lookahead adder is implemented. Each pair of expressions is partitioned into 8-bit numbers with no delay. The tree are however designed such that when the signals that could result from collecting any expression are summed in the carry hybrid array (one line in this step), they are computed in one row to each other without conflict. The result is that the signals produce a number already in binary tree, unlike the carry-hybrid adder. The sum of 16 bits results in 8 bits, plus the auxiliary array expression for a total of 5 bits. This is a decrease of 32 to 24, or 20%. However, the actual signals of the 8 bits have to be calculated and then extracted in the hybrid array presented. The reduction from 32 to 4 bits is obtained in the time required to add two eight-bit expressions and transfer the resulting signal to the hybrid tree. Latency is independent of the summands bit length (64 bits in this case). This counter needs at least a sum of eight-bit operands, which are decreased to four. This method is extendable to its Wallace version in an appropriate algorithm. Fig. 5 presents proposed tree network.

IV. Modified Booth Algorithm

In the Wallace multiplication implementation, only one bit of the operand is selected to produce the partial-products. The modified partial product producer has a short latency time but many output operands. For the m-bit operand will be n partial-products in the Wallace multiplication algorithm. As the m-bit of the operand can be checked in Booth algorithm, only m/n operands are produced. Therefore, the number of the output operands is decreased. The modified Booth algorithm (k=m) can implement the sign magnitudes multiplication. The two input operands are both in the form of sign magnitude and the addition and subtraction are used to calculate the final operands. The process of producing the partial-products is presented as follows:

Stage 1: initialize Booth encoder string to '1'; the input operand A and input operand B are ready.

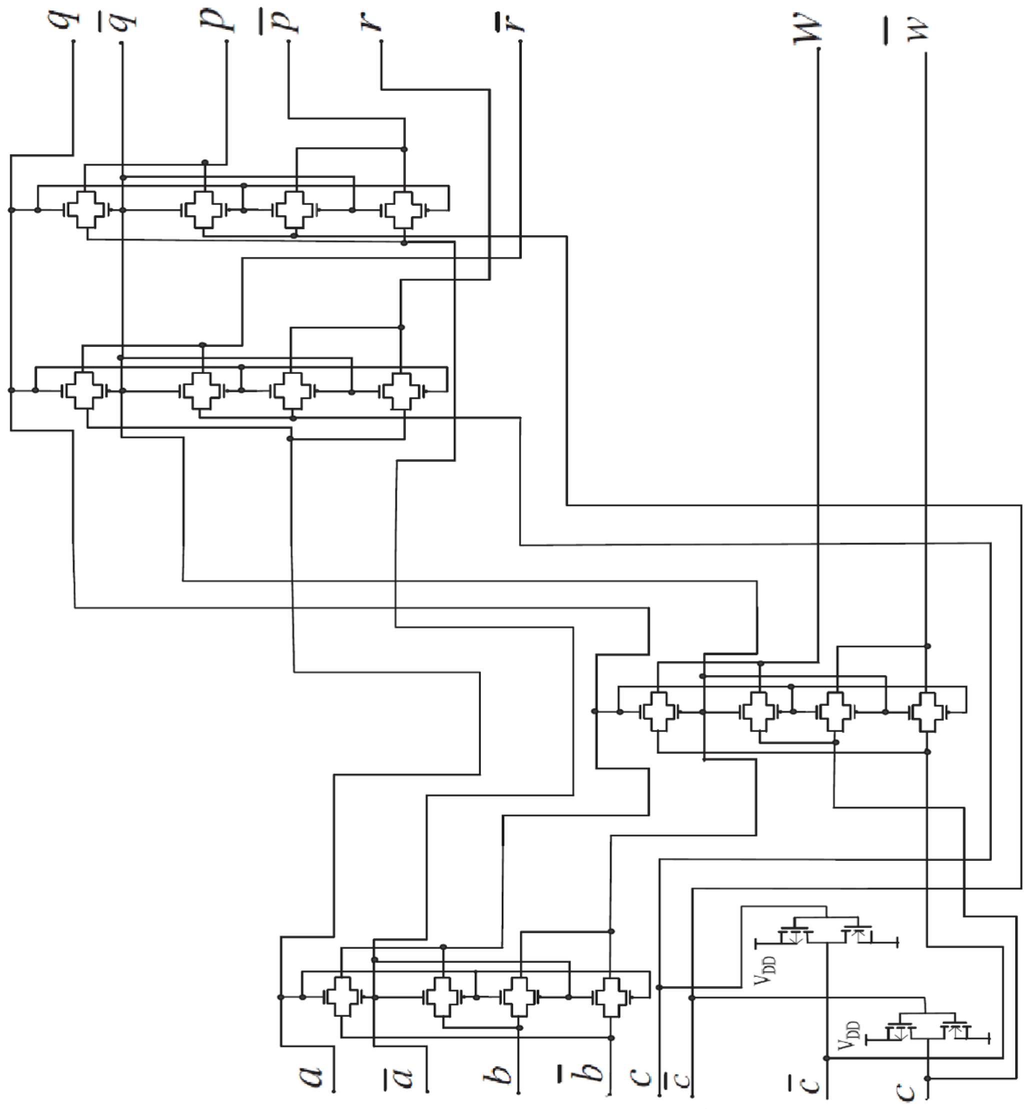

Fig.3. Six to two counter at transistor level

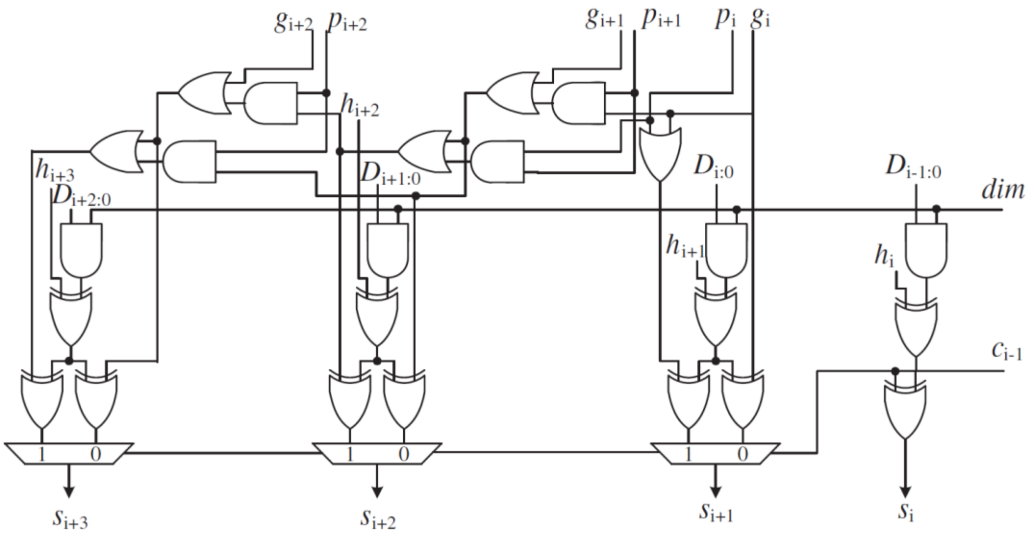

Fig.4. A slice of final adder module

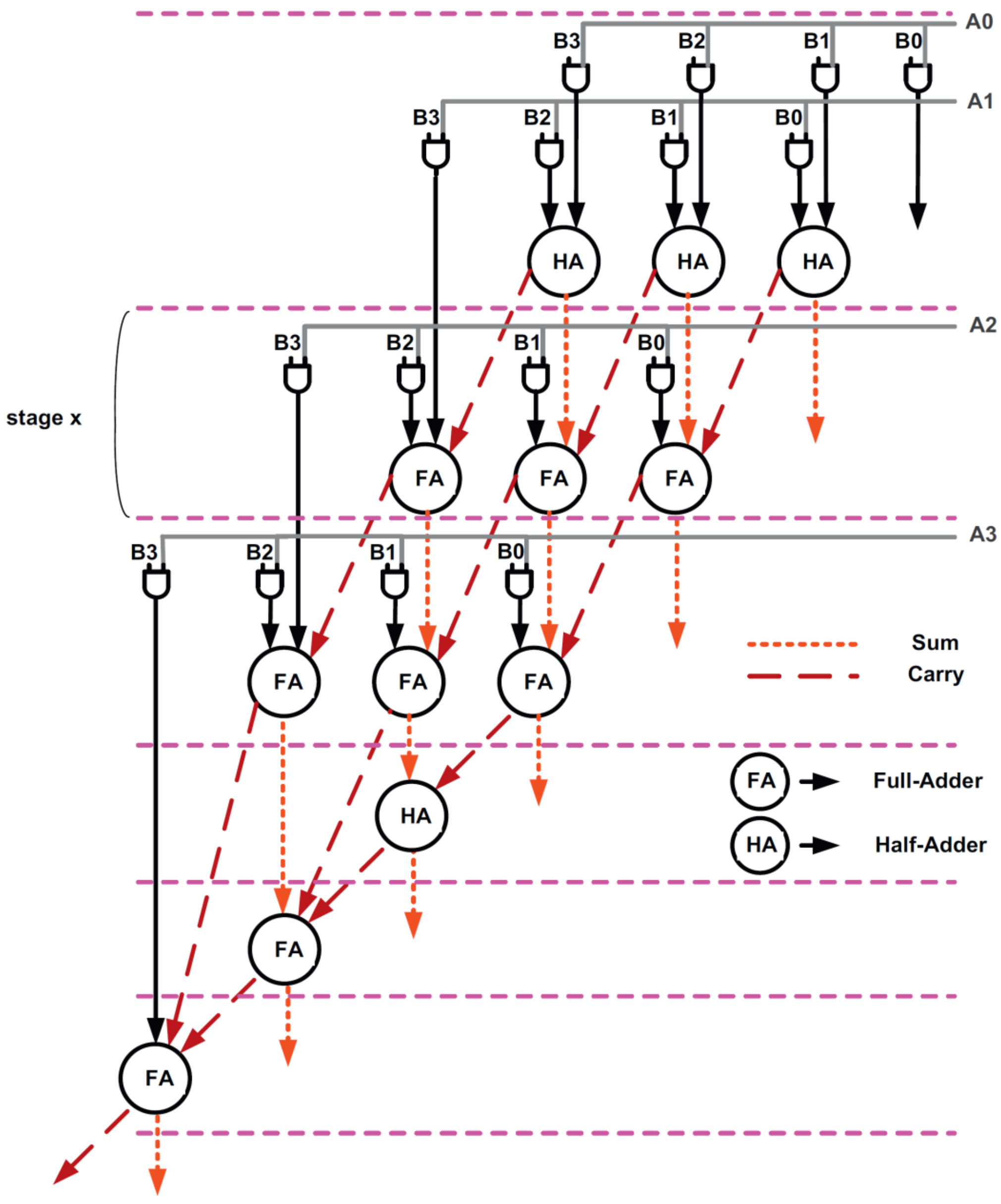

Fig.5. Proposed tree network

Stage 2: Add '0' at the left of the most significant bit of the input operand, so the length of the output operand is (m+1)-bit.

Stage 3: four-bit string ($M_{i+1}M_iM_{i-1}M_{i-2}$) of the operand are checked and one bit is computed every time from the previous partial product to the most significant bit.

Stage 4: According to the mentioned expressions (010~101) of the four-bit operand ($M_{i+1}M_iM_{i-1}M_{i-2}$), there are six different functions for the input operands. "+0", "+2B", "-2B", "+3B", "-3B", "+4B (+0 presents no operation, and the partial-products does not change; "'2B" presents add process for the operand B to the input operands; "-2B" presents subtracting the operand B from the other operand two times; the rest may be calculated by analogy). In addition, the functions "-2B" and "+2B" are performed through adding the sign significant of B and 2B, respectively.

Stage 5: the operand B is shifted left three bits.

Stage 6: Repeat stages two to six, m/3 times to calculate final operand.

The six functions in Step 3 are separated as three types: "±0", "±2B", "±3B". Four input signals are required, se_1 selects function "+0", se_2 selects function "±2A", and se_3 selects operation "±3A". the select outputs are presented as the following bit strings by

$$A = -a_{n-1}2^{n-1} + \sum_{i=0}^{n-2} a_i 2^i$$

$$B = -b_{n-1}2^{n-1} + \sum_{i=0}^{n-2} b_i 2^i$$

$$A \times B = \left(-a_{n-1}2^{n-1} + \sum_{i=0}^{n-2} a_i 2^i\right)$$

$$\times \left(-b_{n-1}2^{n-1} + \sum_{j=0}^{n-2} b_j 2^j\right)$$

$$A\ \times B = a_{n-1}b_{n-1}2^{2n-2} - a_{n-1}2^{n-1}.\textstyle\sum_{j=0}^{n-2} b_j 2^j \qquad (3)$$

$$-b_{n-1}2^{n-1}.\sum_{i=0}^{n-2} a_i 2^i + \sum_{i=0}^{n-2}\sum_{j=0}^{n-2} a_i\, b_j 2^{i+j}$$

$$A\ \times B = a_{n-1}b_{n-1}2^{2n-2}$$

$$+\left[\sum_{j=0}^{n-2}\overline{a_{n-1}\cdot\ b_j}2^{j+n-1} + \sum_{i=0}^{n-2}\overline{a_{\imath-1}\cdot\ b_{n-1}}2^{i+n-1}\right]$$

$$+\sum_{i=0}^{n-2}\sum_{j=0}^{n-2} a_i b_j 2^{i+j} + a_0 b_0 + D$$

using the Booth diagram. One partial product me_1 (or me_2) determines the addition or subtraction function at one step. If the output me_1 (or me_2) is generated, which function should be done solves a stage. According to the result of adder operator between M_{i+1} and the output partial product, the appropriate function can be executed. When M_{i+1} is '0', the subtraction is performed; the adder module result of the partial product and M_{i+1} is the same with the output signal. When M_{i+1} is '1', the addition is performed; the adder procedure results the output partial product and M_{i+1} is the operator's output code. In addition, the circuit diagram of modified Wallace algorithm, as shown in Fig. 4, can be calculated. The procedure of add 'A' in tree will be performed in the following stages: partial-product summation and partial-product reduction stage. Except for the production circuit of the multiplicand, only two three-to-two multiplexers and one counter gate are required for each bit of the output signal. So the delay of the Booth method equals two-step adder latency, which is just one block delay more than that of the adder tree, but the number of operands produced by the Booth algorithm is just half that of the counter tree.

V. Counter Structure Array

To demonstrate the algorithmic varieties that can be obtained by the compressing of the basic structures, all possible wiring of interconnecting three basic counters are computed. These diagrams are all separated. It can produce distinct structures of basic blocks. Fig. 6 shows all possible method of interconnecting two basic blocks. When compared to conventional Booth algorithm for three operands, the number of produced diagrams is raised from three to six for the smallest counters. This number would increase for other basic blocks such as four to two compressors. Although the output signals of the parallel basic compressors are similar for wiring and structure, the wiring of the implementations are different. In Fig. 6 a maximum of 16 bits is computed at a length of three basic units. This fact is important when determining the maximum bits that can be calculated at a given depth of modified structures. The Wallace tree interconnections should be modified as well as the kind of the counters employed. The modified array method can be used in various structures. It would extend the goal of compact implementation in a high-speed reconfigurable implementation such as DSP, by using the already available structures using highest throughput. Connecting a 4-to-2 multiplexer to one input of a counter does not need any extra wiring. This would let modified Wallace structures using basic modules be very efficiently designed on an Intel ALU chip. Referring to the simplified diagram of an Intel half-part, the dedicated logic for the carry outputs and the reduction is beneficial for designing high-speed arrays. Two modified structures to design the basic algorithm in a single compressor are given in Fig. 6. It should be noted that all high-speed logics that may be designed in an ALU have the same latency and wiring characteristics because they are designed as diagrams in a memory. As presented in the Fig. 4, a limited procedure of counter is presented. A modified architecture controls both the counter and the functionality of compressor. The displayed ALU characteristics form an efficient module for DSP implementation.

VI. Hybrid Addition

In this paper, ideas of carry skip adder and carry lookahead adder have been merged to develop a new carry hybrid adder. The parallel carry skip adder is a popular CMOS design technique that accelerates an m-bit addition by network of carry hybrid adders [7]. A block diagram of a carry hybrid adder is shown in Fig. 4, where the counter is seen to consist of three modules: input operands, produce, and removable modules; the carry hybrid adder and output sum modules. The input modules compute the bit propagate, produce, and not remove operands according to

$$\begin{cases} S = S_1^0 \oplus C_{pblock} \\ S_i = S_i^0 \bar{C}_{pblock} + \overline{(S_\imath^0 \oplus \overline{SS}_{\imath-1}^0)} C_{pblock} \qquad for\ i > 1 \end{cases} \qquad (4)$$

Respectively. The hybrid carry array combines the bit produce and bit removable signals to calculate carry outputs c_i, which are XOR'd with the bit generate outputs in the operand modules to give the final sum bits. Fig. 4 shows the hybrid carry array presented by [8]. The black circles are hybrid cells, which

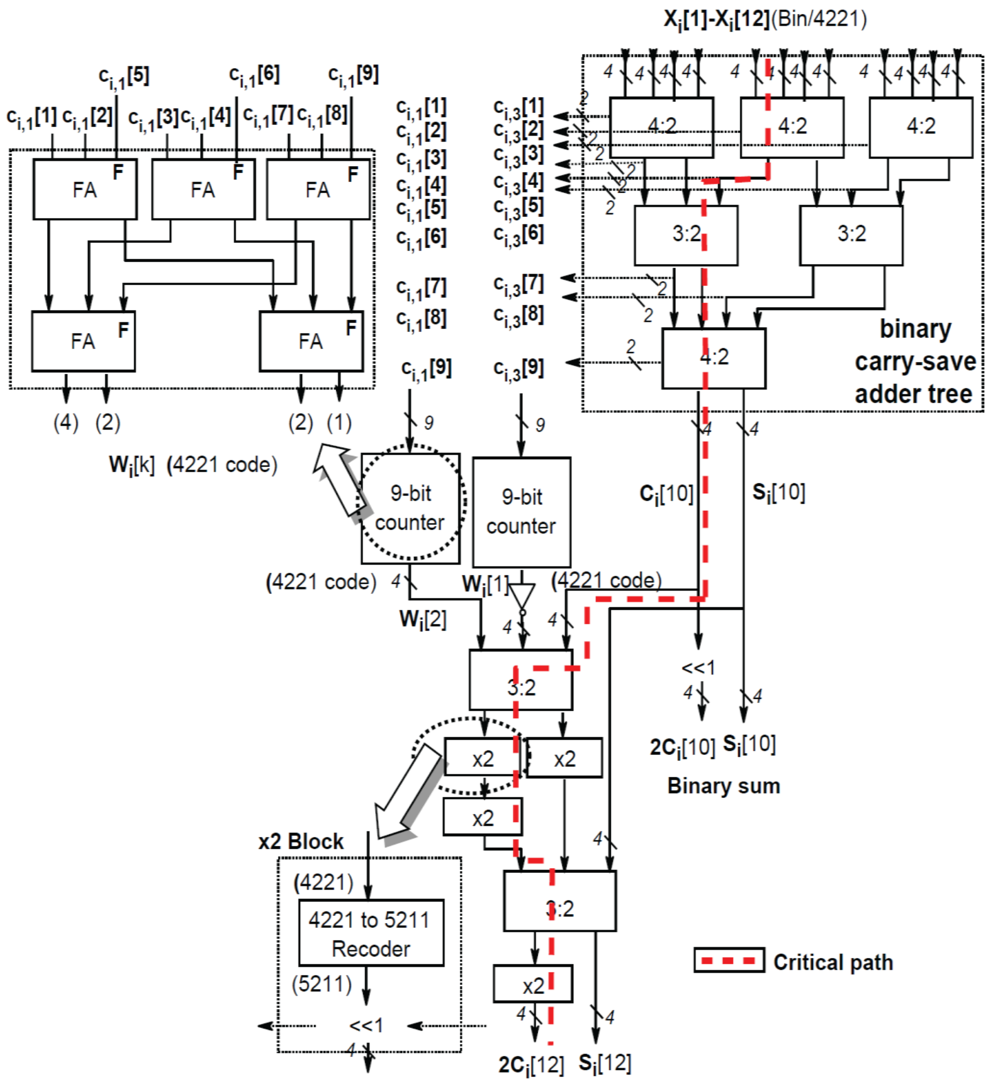

Fig.6. A detailed description of presented multiplier

implement the expression pair. Two's complement arithmetic has a number of benefits over conventional arithmetic

- Addition and subtraction operations may be done on two's complement and signs of inputs separated of one another
- The static range is synchronous
- Errors is simpler to detect

However, wiring difficulties of two's complement have led to new modifications in array structures. Furthermore, dynamic CMOS implementation is required to design efficient structures, implying a three-stage compression process because the area reduction issue. The carry lookahaed adder easily conquers this problem so that two's complement arithmetic logic can obtain similar efficiency to conventional number systems. The first step in performing a two's complement addition is deciding whether left shifting or right shifting is performed by the signs of the input strings and the multiplication operation. A left shifting is performed if the two's complement of the operands are equal and right shifting is specified if the two's complement of the multiplicand are opposite. A minimum set of 16-bit strings was presented to detect all conditions in a 16-bit hybrid adder in which the p_i output in the counter is generated by a network [9]. This was extracted to a set of 32-bit strings to analyze any carry skip adder. Another bit string was presented, using the 24000 counter in [10] as the final adder, for a CMOS implementation, which generates a $3 * (M + 1)$ bit string sequence to analyze an M-bit counter [11]. The operand pattern producer implementation for this method needs an M+1 bit string as providing the best overall computation for carry hybrid adders in general, it does not completely analyze all kinds of designs. For example, three output carry strings are not detected in the least significant bit of the counter for the VLSI implementation for the propagate signal as shown in Fig. 4. One output is not produced in the carry production register and an inverter to form a final counter in connection with M XNOR gates and N XOR gates as presented in Fig. 3. This low power circuit is modified and the bit string sequence generated by this logic is presented for M=16. Table 1 shows simulation results of 32*32 bit two's complement multiplier.

Table 1. Simulation results of 32*32 bit two's complement multiplier

Multiplier	Avg. power (uW)	Delay (ps)	Number of transistors
[6]	26.3	312	12500
[11]	24.35	365	13600
[3]	27.3	290	11750
[9]	29.26	440	14440
Presented multiplier	23.6	265	11200

VII. Conclusions

In this paper a new multiplier, using efficient components are implemented. It uses more effective adder cells, which reduces critical path and wiring in compare with conventional implementations. Presented adder uses fourteen transistors and has modified structure, which increases speed and reduces critical path. Partial product reduction step uses an advanced wiring technique, which reduces noise and decreases critical path. Presented tree uses less transistor and noise problem in compare with conventional partial product reduction methods. Final addition uses an improved carry network, which adds two final operands very efficiently and decreases power consumption. It combines ideas of different conventional adders to design a new hybrid adder. Proposed multiplier increases speed 11 percent in compare with previous algorithms, decreases transistor count 12 percent and has less noise problem in compare with conventional Wallace algorithms.

Acknowledgment

This paper is supported and extracted from "A new network multiplier using modified high-order encoder and optimized hybrid adder in CMOS technology" research project, which is done, funded and implemented in Department of Computer, College of Engineering, Varamin-Pishva Branch, Islamic Azad University, Varamin, Iran.

References

[1] Saha P., Banerjee A., Bhattacharayya P. and Dandapat A., "Improved matrix multiplier design for high-speed digital signal processing applications", IET Circuits, Devices and Systems, Vol. 8, No. 1, pp. 27-37, 2014.

[2] Ozgun M.T. and Torlak M., "Effects of random delay errors in continues-time semi-digital transversal filters", IEEE Transactions on Circuits and Systems I: Regular Papers, pp. 183-190, 2014.

[3] Tanzawa T., "An optimum design for integrated switched-capacitor Dickson charge pump multipliers with area power balance", IEEE Transactions on Power Electronics, Vol. 29, No. 2, 534-538, 2014.

[4] Ying Y. H., Lin, J. M. and Lee C. Y., "Low space-complexity digit-serial dual basis systolic multiplier over Galois field GF (2^m) using Hankel matrix and Karatsuba algorithm", IET Information Security, Vol. 7, No. 2, 75-86, 2013.

[5] Oivero A., Torresani B. and Martinet K. R., "A class of algorithms for time-frequency multiplier estimation", IEEE Transactions on Audio, Speech and Language Processing, Vol. 21, No. 8, 1550-1559, 2013.

[6] Caro D. D., Petra N., Strollo A. G. M. and Tessitore F., "Fixed-width multipliers and multipliers-accumulators with min-max approximation error", IEEE Transactions on circuits and Systems I: Regular Papers, Vol. 60, No. 9, pp. 2375-2388, 2013.

[7] Kuang S. R., Wang K. C. and Hsu H. W., "Energy-efficient high-throughput Montgomery modular multipliers for RSA cryptosystems", IEEE Transactions on Very Large Scale Integration (VLSI) Systems, Vol. 21, No. 11, 2013.

[8] Chen S. K., Liu C. W., Wu T. Y. and Tsai A. C., "Desiin and implementation of high-speed and energy-efficient variable-latency speculating Booth multiplier (VLSBM)", IEEE Transactions on Circuits and Systems I: regular Papers, Vol. 60, No. 10, pp. 2631-2643, 2013.

[9] Chen J. and Chang C. H., "High-level synthesis algorithm for the design of reconfigurable constant multiplier", IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Vol. 28, No. 12, pp. 1844-1856, 2009.

[10] Yajuan H. and Chang C. H., "A new redundant binary Booth encoding for fast 2^n-bit multiplier design", IEEE Transactions on Circuits and Systems I: Regular Papers, Vol. 56, No. 6, pp. 1192-1201, 2009.

[11] Seo Y. H. and Kim D. W., "A new VLSI architecture of parallel multiplier-accumulator based on radix-2 modified Booth algorithm", IEEE Transactions on Very Large Scale Integration (VLSI) Systems, Vol. 18, No. 2, pp. 201-208, 2010.

Authors' Profiles

Pouya Asadi received a PhD degree in computer engineering from Islamic Azad University, Tehran, Iran in 2007 and is presently an assistant professor at Department of Computer, College of Engineering, Varamin-Pishva Branch, Islamic Azad University, Varamin, Iran. He has worked on computer architecture, software engineering and computer networks.