Considerations for Design of a Complex Multiplier

Vojin G. Oklobdzija
Department of Electrical and Computer Engineering
University of California
Davis, CA 95616
vojin@eecs.ucdavis.edu
(916) 752-5634

David Villeger, Thierry Soulas
Ecole Superieure d'Ingenieurs d'Electrotechnique et d'Electronique
2 Boulevard Blaise Pascal, BP 99
93162 Noisy le Grand CEDEX
FRANCE
33-1-45-92-65-00

Abstract

IIn this paper we consider a design of a multiplier for the multiplication of complex numbers. The numbers are represented by two 13-bit parts with the same 6-bit exponent. Multiplication of complex numbers was examined from the perspectives of performance, complexity and silicon area. The design shares the Booth encoding for the Real and Imaginary parts including only one Wallace tree of 4:2 adders for each part. The number of adders used in the multiplier is also reduced. We consider VLSI CMOS technology and the relevant characteristics as they impact the implementation and performance. The circuit has been designed and laid out using 1.5um standard CMOS process

1. Introduction

The need for Digital Signal Processors (DSP) has increased as a reflection of the increase in processing power and capabilities, and also as a result of switching from analog to digital signals in a wide range of applications.

In order to increase their dynamic range, there is a need to represent signals in floating-point number representation and perform the operations as floating-point operations. Given the fact that we represent signals as complex numbers, operations on complex numbers represent a large part of DSP operations. Since multiplication of two complex numbers is a very frequent operation in many signal processing algorithms, we have concentrated on an efficient hardware implementation [1,2].

Straight forward implementation of complex number multiplication for example,

$$P = (a+ib).(c+id) = (ac-bd)+i(ad+bc) \quad \ldots\ldots\ (1)$$

requires 4 multipliers (each contains one adder) and 2 adders. A frequently used technique to reduce the number of multipliers, is to share multiplications with the real and Imaginary part by representing complex product using the following formula:

$$P = (a+ib).(c+id) = [(a-b)d+a(c-d)] + i[(a-b)d+b(c+d)] \ldots(2)$$

This yields:

$$Re[P]=(a-b)d+a(c-d) \text{ and } Im[P]=(a-b)d+b(c+d) \quad \ldots\ldots(3)$$

where the term (a-b)d is shared. In spite of saving on one multiplication operation (either in hardware or software), this method requires four extra adders. In addition, attention needs to be given to normalization problems after each multiplication, at the expense of both: time and accuracy. Although advantageous in the case of a software implementation, this method does not yield a clear advantage when a hardware implementation is required. Other methods such as CORDIC [3] are not well suited for parallel implementations in hardware. Therefore it might be better to consider the "straightforward" multiplication of complex numbers and concentrate on the implementation issues instead.

This paper describes a new design of a fast complex number multiplier. By using a specific final adder optimized for the signal arrival from the Wallace tree and by sharing the Booth encoding and Wallace tree, we can improve size, speed and accuracy.

2. Architecture

The multiplier is to operate on two complex numbers and produce the result in one cycle. The complex numbers are packed in one 32-bit word with the real and imaginary parts sharing the same exponent. This allow us to store the complex number in just one word in memory; yet there is sufficient precision and dynamic range for the class of signal processing applications that we are considering. The representation of complex numbers is given in Fig. 1.

Fig. 1. Representation of the complex number in one 32-bit word

Multiplication of two complex numbers is performed by first separating exponent parts and adding them together.

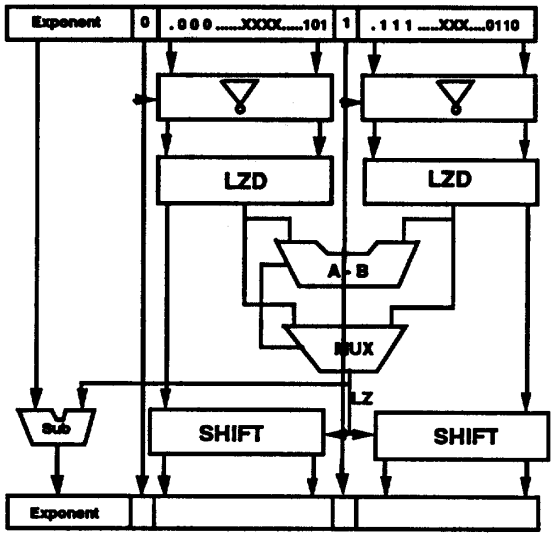

Fig.2. Post Normalization Unit Organization

The numbers are assumed to be normalized to the greater of the two: real and imaginary parts. The sum of the exponents becomes the exponent of the result. It is clear, given one common exponent for both the real and the imaginary part, that the product could result in a non-normalized number. Further, when normalizing the product term with the common exponent it is only possible to normalize it with the respect to one of the parts: real or imaginary. We have chosen to always normalize to the bigger one, i.e., we will shift the result left until one of the parts becomes normalized. Given that this is not a sequential process (one that we want to accomplish in one cycle) the post-normalization process can be quite elaborate. First, we have to count the leading zeros (ones) in both parts. Second, we need to determine the smaller of the two leading zeroes and use this number to left-shift both, the real and imaginary part and subtract this number from the exponent. This operation needs to be accomplished in one cycle and it involves comparison, leading zero detection, shift and subtract, Fig.2.

The multiplication algorithm that we used operates in a rather standard fashion. It uses Booth encoding to reduce the number of partial products and the Wallace Tree method to sum the partial products, reducing them to two operands which are added in a fast carry-propagate adder in the final stage. Normalization is performed in the following cycle. As it has been shown by M. Santoro [4,5], the distribution of the silicon area occupied by the multiplier can be divided into approximately four quarters. One quarter of the layout area is devoted to the Booth encoding logic, one for the Wallace Tree, and the other two for the final adder and wiring channels, respectively.

Therefore, we decided to share Booth encoders for the real and imaginary parts by Booth encoding terms a and b where a as well as b are used to form a product with c and d. The real part of the complex product is form by subtraction, ac - bd, while the imaginary part is formed by summation, ad + bc. We have also decided to perform this summation / subtraction as a part of the Wallace Tree, saving on two adders as well as reducing the time it takes to generate the product. The fact that both Booth encoded operands a and b are each being multiplied by c and d allows us to lay the multiplier out in a compact quadrant structure. The main structure of the multiplier is shown in Fig. 3. By sharing Booth encoders and merging the addition / subtraction into a combined Wallace tree, we estimate (using Santoro's rule) that at least 10% in area has been saved. In addition multiplier speed has been increased.

Handling the sign in this multiplier has been achieved by proper encoding of sign bits and carrying 4 extra bits: two for the encoded sign, and two for the carry in associated with the complementation of the number. An additional bit is inserted in the 14th bit position of each of the product trees to be summed. This leads to an elegant way of performing parallel multiplication of positive and negative numbers using Booth encoding without using sign extension nor a correction term.

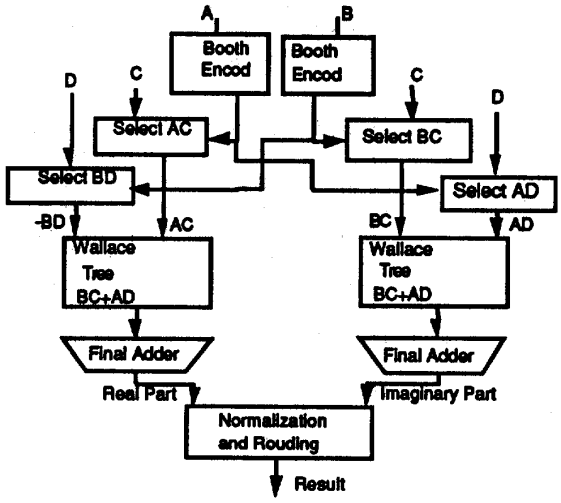

Fig.3. Organization of the Multiplier

3. Implementation of the Wallace Tree

The best known technique for summation of the partial products has been the use of Wallace trees [6]. They are usually implemented by using full adders. They are also referred to as 3:2 counters because they take 3 inputs of the same "weight" and produce two outputs of two different "weights". By "weight", we refer to the value associated to the position of the digit. This is the

simplest and most straightforward technique. However, the layout of such a tree is not regular. It takes more space and utilizes wires of different and irregular lengths. The layout of Wallace tree implemented using full adders is shown in Fig.4.

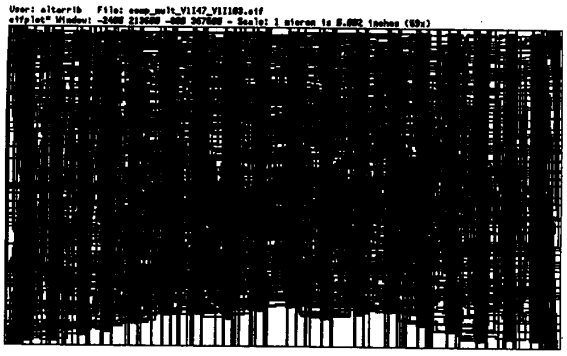

Fig. 4. Layout of the Wallace tree

The Wallace tree can be implemented using variety of counters and several schemes have been proposed, the best known being Dadda's scheme Dadda [7][8]. The 7:3 adder is similar to the use of the full adder in concept, but it results in more efficient implementation. The 7:3 and 9:3 adders are more efficient, but their advantage begins to show only with Wallace trees bigger than the one used in this particular case.

Recently, Santoro used 4:2 counters [5]. They are a special case of using 3:2 counters or full adders which differ only in the ways they are implemented (from the full adders) and in the way those adders are interconnected inside the 4:2 counter. The 4:2 counter has 5 inputs (4 partial products and a carry input) and 3 outputs (2 carry outputs and the sum). The interesting feature of this counter is that the carry output signal from the counter does not depend on the carry input to the counter. Therefore, carry is propagating only inside the counter through the two adders used to implement the counter. We used the 4:2 adder whose circuit is shown in Fig. 5. The implementation using 4:2 counters turned out to be more regular.

Comparing both 3:2 and 4:2, trees, the tree of 4:2 counters has half of the levels compared to the one using 3:2 counters. Signal propagation in the 4:2 counter involves propagation through 4 XOR gates compared to a 3:2 counter involving 2 XOR gates in the critical path. This makes those two implementations almost equivalent, except for the advantage in layout regularity exhibited by the scheme using 4:2 counters. However, we were able to redesign the 4:2 counter in such a way that the worse case propagation through 4:2 counter involves 3 instead of 4 XOR gates.We have chosen to implement Wallace tree consisting of full adders, and a tree of 4:2 adders and compare only those two choices and exclude 7:3 and higher order counters from our considerations.

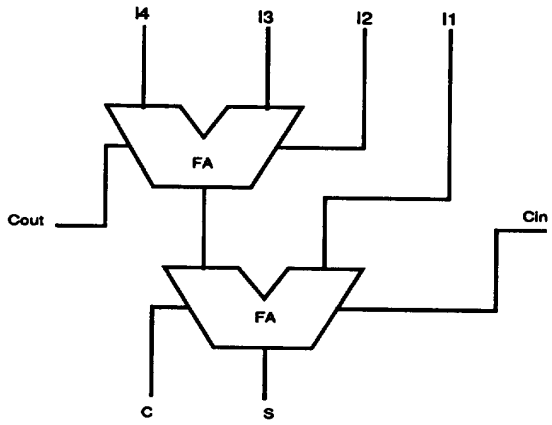

Fig. 5. Structure of 4:2 counter

3.1. Signal Arrival Time in the Multiplier

The speed of Wallace trees is critical; it depends on the number of levels and the way the counters are interconnected. We have two solutions to add the partial products. The first is to have Wallace trees for **ac, -bd, ad** and **bc**, and the second to add the results separately. This would involve two carry propagation operations instead of one. In both cases reducing both partial products to two operands for the final addition would add a level or two (if we used 4:2 or 3:2 counters respectively)

Therefore we decided to design one common Wallace tree for both partial products and performing addition / subtraction in one common Wallace tree. This does not necessarily result in a speed increase; however, it does result in more compact layout and more regular wiring, which is reflected in the speed of the multiplier.

In terms of speed, signals originating from the Wallace tree arrive at the output in very different times. It was observed that they arrive sooner at the ends of the multiplier tree, while the signals in the middle of the tree are arriving last [11]. The ideal situation would be if we can trim some of the delay from the middle of the tree and distribute it toward the ends. This optimization process would yield a more balanced tree and shorten the longest path.

We have implemented two types of Wallace trees: one using 3:2 (RWT) and other using 4:2 (MWT) counters and compared their delays and signal arrival profile. They are shown in Fig.6.

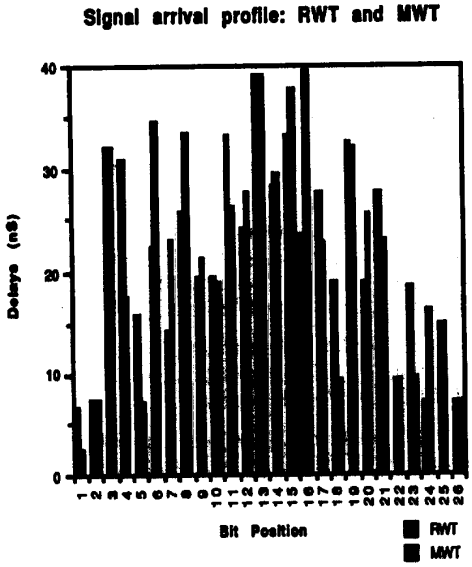

Fig.6. Signal Arrival Profile for RWT and MWT

We can observe that the use of 4:2 counters results in a signal arrival profile that is more balanced compared to the one using 3:2 counters. The total delay of the tree utilizing 4:2 counters is t = 53.3 nS while for the one using 3:2 counters the delay is t = 56.3 nS. This is not a substantial difference, however, for multipliers with wider operands this difference would be larger. This is attributed to the change in the signal paths in the tree. If we use 4:2 counters or even ones with higher compression ratios, for example 6:3 or 8:4, the critical paths are shifted more horizontaly, thus adding the delay to the end bits and taking it away from the middle bits. With the proper mix of counters, we believe it is possible to design a Wallace tree exhibiting a balanced signal arrival [work in progress].

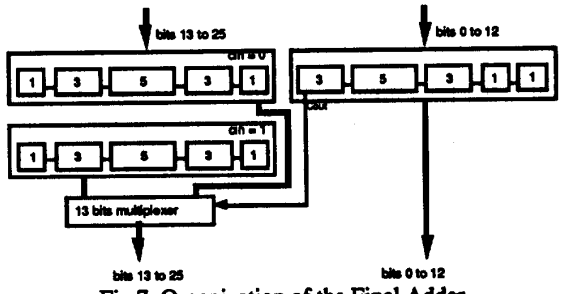

Fig.7. Organization of the Final Adder

4. The Final Adder

The final adder used in the last stage of the multiplier is a Variable Carry Skip Adder [9,10], because this scheme

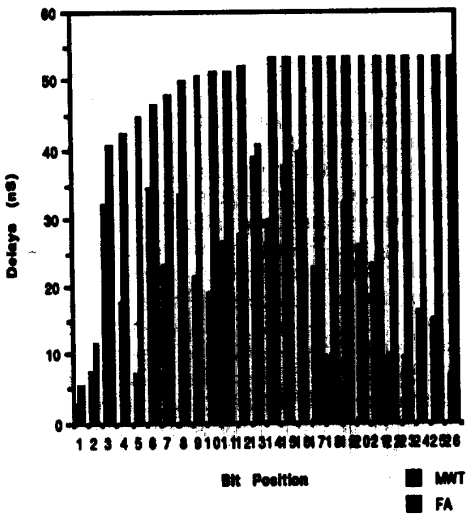

Fig.8. Signal Arrival Profile from the Multiplier

makes it possible to achieve good performance without much additional logic. Another reason for using a VBA adder is that the size of the individual blocks in the VBA adder is fine-tuned to minimize the difference in delays introduced by the carry-paths of the different length. However, this optimization is done under the assumption that all of the input bits to the adder arrive at the same time. In our case this is not true, and we have modified this scheme by applying the signal arrival profile originating from the Wallace tree.

We have only implemented a 13-bit VBA adder, and used the Conditional Sum scheme for the upper 13-bits [12]. The adder is shown in Fig.7. The critical path of the adder, i.e. the carry signal, is implemented as a string of multiplexers. This is because they are implemented from the pass transistors in the CMOS library, making the multiplexer even faster than a simple gate of the standard cell library. This adder performs addition of two 26-bit numbers in t = 23.2 nS, assuming that all of the inputs arrive in the same time. However, this is not a realistic assumption in our case, and its delay is measured under the signal arrival profiles obtained from the Wallace tree implemented from the 3:2 and 4:2 counters. The delays obtained are t = 39.2 nS for the regualr Wallace tree of 3:2 counters (RWT) and t = 39.1 nS for the modified one MWT of the 4:2 counters. This shows the advantage of using 4:2 instead of 3:2 counters. The signal arrival profile for the multiplier (including the final adder) is shown in Fig.8. The layout of the final adder is shown in Fig.9. It turned out to be quite regular.

6. Conclusion

The structure of the complex multiplier presented in this paper takes advantage of using one common Wallace tree to perform summation of the partial products and performing one add/subtract operation resulting in the real / imaginary part of the complex number product. Considering that normalization and routing are slow operations, we gained a great advantage by requiring only one routing and normalization instead of two. In designing the Wallace tree we achieved a careful balance by using 4:2 instead the 3:2 counters. Should the word-length in signal processors increase, further advantage could be taken by using counters of higher order such as 8:4 or even 12:6. It should be easier to construct a more efficient tree with more bits available, simply because there are more possibilities. Also, different adders could be used, including Carry Look Ahead adder optimized for the signal arrival time. The Conditional Sum adder seems to be a better choice when the adder size is considerably increased. Further performance improvement can be obtained by fine tuning the gates knowing that no gate is symmetric, even the XOR. There is always a possibility of improving the design of the tree, but partly at the expense of the gain achieved by the asymmetric adder (optimized for the different signal arrival).

We believe that the multiplier for complex numbers will find its application not only in signal processors, but also in the general and scientific computation environment.

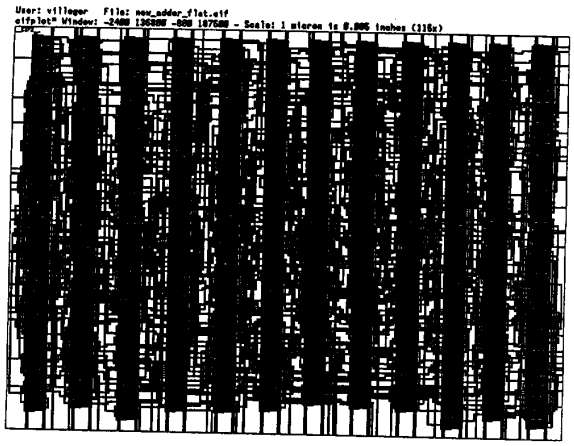

Fig.9. Layout of the Final Adder

References

[1] N. M. Marinovich, V. G. Oklobdzija, , *"VLSI Chip Architecture for Real Time Ambiguity Function Computation "*, 25th Asilomar Conference on Signals, Systems and Computers, November 4-6, Pacific Grove, 1991.

[2] N. M. Marinovich, V. G. Oklobdzija, , *"A VLSI Architecture for Real-Time Computation of The Cross-Ambiguity Surface "*, submitted for publication to IEEE Transactions on Signal Processing, April 1992.

[3] J.E.Volder, *"The CORDIC trigonometric computing technique "*, IEEE Transactions on Electronic Computers, Vol EC-8, p. 330, September 1959.

[4] J.S.Walther, *"A Unified Algorithm for Elementary Functions "*, SJCC, p. 379, April 1971.

[5] Mark R. Santoro, Mark A. Horowitz, *"SPIM: A Pipelined 64X64-bit Iterative Multiplier"*, IEEE Journal of Solid State Circuits, Vol. 24, No. 2, April 1989.

[6] C.S. Wallace, *"A suggestion for a fast Multiplier"*, IEEE Transaction on Electronic Computers, Vol. EC-13, No. 1, February 1964.

[7] W. J. Stenzel, W J. Kubitz, *"A Compact High-Speed Parallel Multiplication Scheme"*, IEEE Transaction on Computers, Vol. C-26, No. 10, October 1977.

[8] Luigi Dadda, *"Some Schemes for Parallel Multipliers"*, Alta Frequenza, Vol. 34, No. 5, March 1965.

[9] V. G. Oklobdzija, E. R. Barnes, *"Some Optimal Schemes for ALU Implementation in VLSI Technology"*, 7th Symposium on Computer Arithmetic ARITH-7, June 4-6, 1985, Urbana, Illinois.

[10] V. G. Oklobdzija, E. R. Barnes, *"On Implementing Addition in VLSI Technology"*, IEEE Journal of Parallel Processing and Distributed Computing, No.5, p. 716 1988.

[11] K.F.Pang et al, *"Generation of High Speed CMOS Multiplier -Accumulators"*, Proceedings of the International Conference on Computer Design, Rye, New York, October 1988.

[12] J. Sklansky, *"Conditional-Sum Addition Logic"*, IRE Transactions on Electronic Computers, Vol. EC-9, No.2., June 1960.