Journal of VLSI Signal Processing, 7, 213–222 (1994)
© 1994 Kluwer Academic Publishers, Boston. Manufactured in The Netherlands.

An Integrated Multiplier for Complex Numbers

VOJIN G. OKLOBDZIJA
Department of Electrical and Computer Engineering, University of California, Davis, CA 95616

DAVID VILLEGER AND THIERRY SOULAS
Ecole Superieure d'Ingenieurs d'Electrotechnique et d'Electronique, 2 Boulevard Blaise Pascal, BP 99, 93162 Noisy le Grand CEDEX, France

Received February 2, 1993; Revised June 3, 1992.

Abstract. In this article we consider a design of a multiplier for the multiplication of complex numbers. The complex numbers are packed into one 32-bit word. They are represented by two 13-bit parts with the same 6-bit exponent. Multiplication of complex numbers is examined from the perspectives of performance, complexity and silicon area. The design is unique and combines shared Booth encoding for the real and imaginary parts including only one combined modified Wallace tree of 4:2 adders for each part. The regular Wallace tree is compared with the tree of 4:2 adders. This design results in a more compact wiring structure and balanced delays resulting in a faster multiplier circuit. 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.

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. Our particular application is based on real time ambiguity function computation as discussed in [1].

However, the need for specific high-speed floating-point operations, long numbers and accuracy is present in many other DSP specific applications. Complex number operations have been usually performed as a sequence of common operations. For example, complex number multiplication of two complex numbers $W = (a + ib)$ and $Z = (c + id)$ is usually done with 4 multipliers (each contains one adder) and 2 final adders (FA) as:

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

or it is performed sequentially passing the operands through the available arithmetic units in the data path (usually one multiplier and one adder).

To reduce the number of multipliers, it is proposed to share multiplication by calculating the real and imaginary parts in the following way:

$$Re[P] = (a - b)d + a(c - d)$$
$$Im[P] = (a - b)d + b(c + d) \quad (2)$$

However, both methods need one adder and require the resolution of normalization problems after each multiplication, wasting both time and accuracy.

Multiplication of complex numbers using a parallel multiplier has been investigated by Pekmestzi [3]. In his work Pekmestzi represents complex numbers by merging them into two bits (for each bit position) representing real and imaginary part. His complex number adder is designed with this representation in mind, which essentially results in an integrated network of two adders merged into one. He uses the same approach for multiplication where he realizes the subtraction function inside the Wallace tree. His multiplier is a parallel Wallace tree multiplier composed of two multipliers of the same type merged into one. The final adder is a ripple-carry implemented in the last stage of the parallel multiplier network. The complex number multiplier by Pekmestzi does not take advantage of higher order counters such as 4:2, nor does it consider possibilities of speeding up the final addition step.

Another complex number multiplier was reported by Akinwande et al. [4]. It is a very fast implementation in GaAs heterostructure FET technology and therefore the level of integration is fairly low, resulting in only partial implementation of the multiplier. It accomplishes the complex number multiplication operation in several alternative steps each of which consists of 3 pipeline stages. Their throughput is high, producing a complex product every 8 nS and operating at the clock rate of 520 MHz. However, their implementation uses a regular Wallace tree and CLA adder in the final stage. Therefore the speed of this multiplier is achieved only due to the application of very special technology and not to any use of sophisticated multiplication scheme.

The other approach reported by Shyu et al. [5] accomplishes complex number multiplication with only two integer multiplications. Their scheme is based on the use of a Quadratic-Polynomial Fermat Residue Number System (QFNS) which in our case has very little practical value.

It is also possible to use the CORDIC algorithm for the computation of the product of the two complex numbers by properly initializing the sequence of the shift and add operation with the table-lookup [6]–[8]. We have not considered CORDIC for this implementation, though this algorithm warrants further studies for the DSP computation due to its specific properties.

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 is done to efficiently utilize memory. It is also considered that this format provides an ample range and sufficient precision for most DSP real-time applications. In our particular implementation [1]–[2] we eliminated normalization after each multiplication, becuase normalization and rounding are slow processes. Our approach is advantageous because it uses only one rounding and normalization after the final adder.

The representation of complex numbers is given in figure 1. Multiplication of two complex numbers is performed by first separating exponent parts and adding them together.

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 produce 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 (that can be accomplished 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 zeros 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, shifting and subtraction, figure 2.

The multiplication algorithm that we used operates in a rather standard fashion. It uses Booth encoding [9] to reduce the number of partial products and the Wallace Tree method [10] 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.

M. Santoro has shown that the distribution of the silicon area occupied by the multiplier can be divided into approximately four quarters [14]–[15]. 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. In our case, we 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 formed by subtraction, **ac** − **bd**, while the imaginary part is formed by summation, **ad** + **bc**. The most direct solution would be to implement Wallace trees separately for partial products and add the results separately as shown in

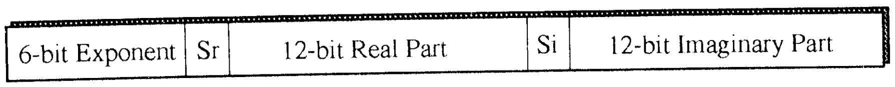


Fig. 1. Representation of the complex number in one 32-bit word.

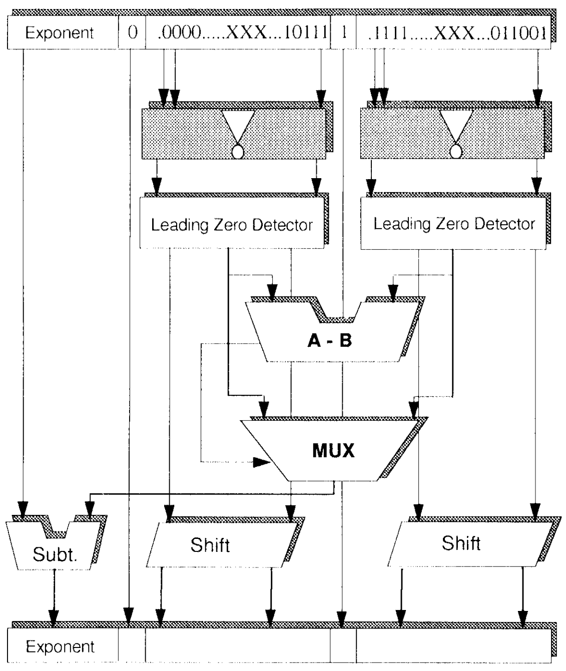


Fig. 2. Post normalization unit organization.

figure 3a. The better solution is however, to use one merged Wallace tree and perform addition (subtraction) inside the Wallace Tree as a part of bit reduction process, as shown in figure 3b, thus avoiding the increase in the number of levels [2]. 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 figure 4. By sharing Booth encoders and merging the addition (subtraction) into a combined Wallace tree, we estimate that about 10% in area (wiring not included) 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 3 extra bits: two for the encoded sign, and one 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.

Finally, the reduced partial products from that Wallace tree are added in the fast Final Adder (FA). This adder is designed to take advantage of the specific profile of the signal arrival times that are origination from the Wallace tree.

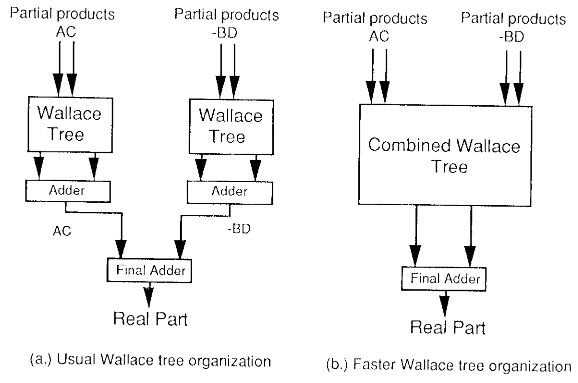


Fig. 3. Structure of the Wallace trees.

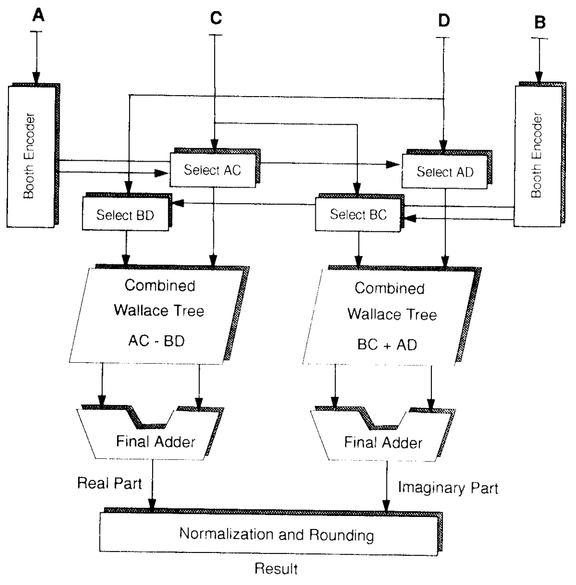


Fig. 4. Organization of the multiplier.

3. Implementation of the Wallace Tree

The best known technique for summation of the partial products has been the use of Wallace trees [10], which 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 Wallace tree can be implemented using a variety of counters and several schemes have been proposed, the best known being Dadda's scheme [11]. The use of larger size counters and combination of different types has been discussed in [12]. The 7:3 adder is similar to the use of the full adder in concept, but it results in more efficient implementation. Though more efficient, their advantage begins to show only with Wallace trees bigger than the one used in this particular case. We considered only a Wallace tree built from full adders and a tree of 4:2 adders as proposed by Santoro.

Recently, Santoro used 4:2 counters [14] and the same approach has been followed by Nagamatsu et al. from Toshiba Corporation in their 54×54b multiplier implementation [16]. It should be pointed out that the idea of using 4:2 counter (compressor) was disclosed by A. Weinberger in 1981 [13]. They are a special case of 3:2 counters or full adders differing only in the implementation and in the way 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). A good feature of the 4:2 counter is that the carry output does not depend on the carry input and that the two carries have the same weight. Therefore, there is no carry propagation from far right to left but inside only two adders. The 4:2 counter used by Santoro results in the longest path equivalent to 4 XOR gate delays. Multiplier design by authors from Toshiba Corp. [16] uses a specially designed 4:2 counter with reduced delay.

The Wallace tree is not regular and is difficult to be wired. Santoro has shown that the Wallace tree can be better in terms of wireability by using 4:2 adders. We have implemented both Wallace trees: Regular Wallace Tree (RWT) using 3:2 counters and Modified Wallace Tree (MWT) using 4:2 counters. The wiring complexity of each is illustrated in the wiring diagrams of RWT (figure 5a) and MWT (figure 5b).

Comparing both trees, we realized that the tree of 4:2 adders has almost one half of the levels (of used counters) compared to the RWT built from the full adders. However, in terms of the gate delays each 4:2 counter stage of MWT is similar to a delay equivalent to 4 XOR gates. On the other hand, the Wallace tree stage built from the full adders, RWT, is comparable to 2 XOR gate delays. Therefore, the use of 4:2 counters is considered almost equivalent to RWT in terms of the delay. The only recognized advantage of 4:2 counters is in their use resulting in more regular layout. To take full advantage of 4:2 counters, we designed a special 4:2 counter cell using available ASIC cells, as it was done by the authors from Toshiba [16]. We call it Reduced Delay Counter (RDC), resulting in only 3 XOR equivalent gate delays. The RDC cell is shown in figure 6. The resulting RDC 4:2 cell is larger (in terms of equivalent gates, 30 versus 20) for the 25% increase in speed of the resulting Wallace tree. The signal arrival time resulting from such a Modified Wallace Tree (MWT) shows more even profile which is the result of the optimization and passing of the carry signals. The signals from MWT tree arrive sooner than the ones from the RWT. The signal arrival profiles for the RWT and MWT are shown in figure 7.

The resulting multiplier shows an overall advantage in speed over the one using RWT. We think that for the longer operands one could further optimize the FA given that the end effects of the different signal arrival times will still exist. The speed advantage of such a modified tree in terms of the delay over a regular Wallace tree will increase with the increase of the operand length giving an additional advantage to our approach.

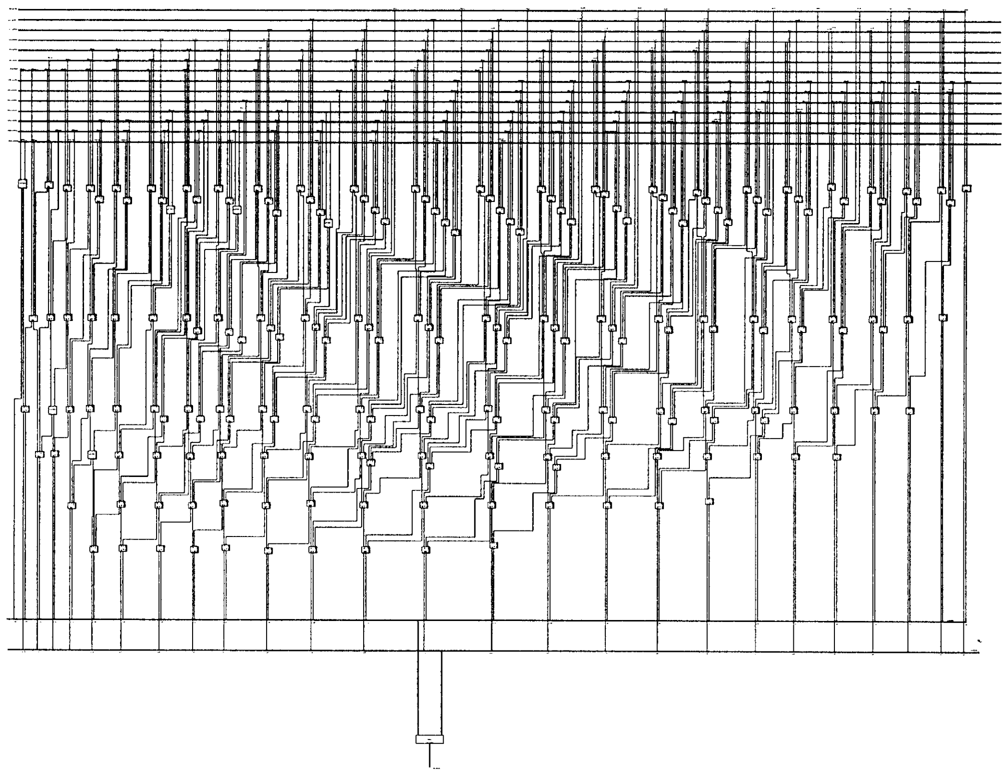
We also realized that the delay of the RDC shows sensitivity to the inputs, i.e., that the delay is different for different input to output path. This opens an opportunity for further speed optimization by selecting which input to output combination to use in a particular MWT realization. However, this would require either a sophisticated placement and wiring tool or "manual intervention," and it was not considered for that reason.

3.1. Signal Arrival Time in the Multiplier

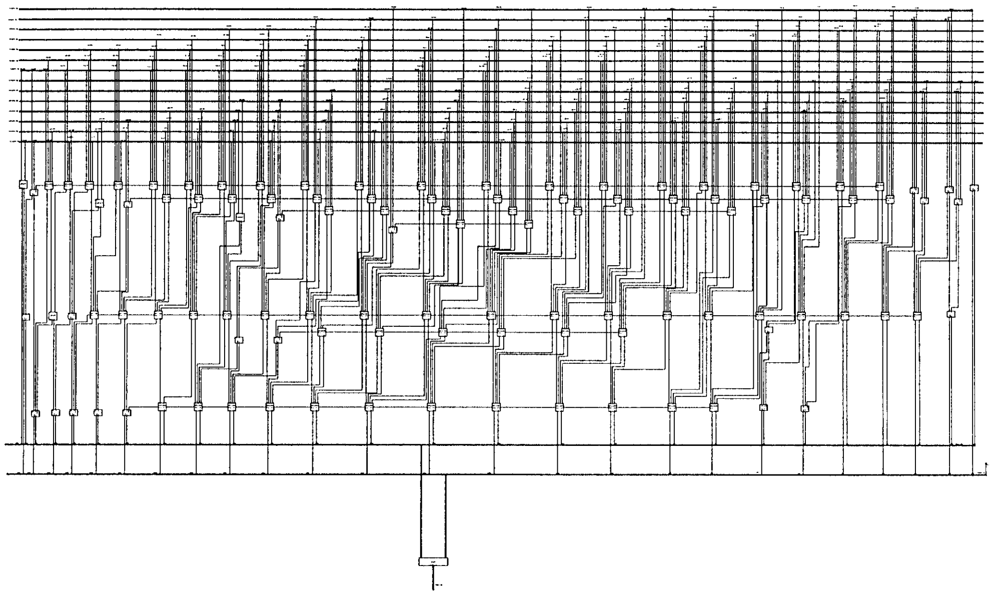
Signals originating from the Wallace tree arrive at the output in very different times. It was observed that they arrive sooner at the ends of the multiplier tree, while the signals in the middle of the tree arrive last [17]. The ideal situation would be to trim some of the delay from the middle of the tree and distribute it toward the ends. This optimization process would yield a more balanced tree and shorten the longest path. Such an approach has been first taken by K. Pang et al. of LSI Logic Corporation in the implementation of their MACGEN tool. Their tool automatically generates optimized netlist and compact layout for multiplier-accumulator implementations [17].

On the least-significant bit end not much can be done and the resulting signal arrival profile will always have a positive slope starting from the least-significant bit. In the middle part we can reroute some of the signals, so that they end up in the most-significant part of the multiplier tree. This can be achieved by using limited carry propagation. Such an approach is described in [22].

As far as the use of 4:2 counters is concerned, the in-out carry signal does just that for one bit position.



(a) RWT (consisting of 3:2 counters)



(b.) MWT (consisting of 4:2 counters)

Fig. 5. Wiring diagram of the Wallace tree.

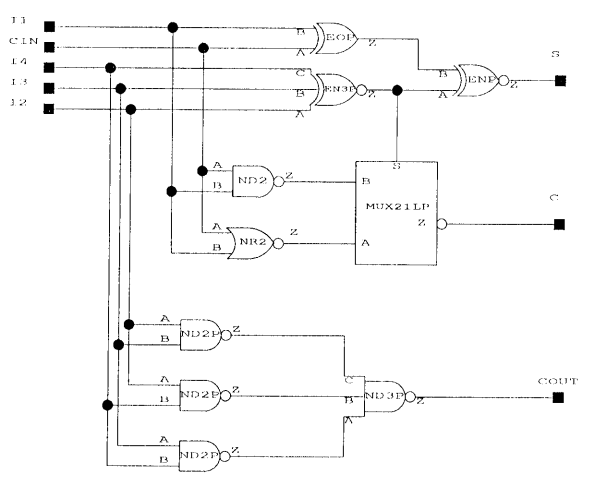


Fig. 6. Logic of the Reduced Delay Counter RDC.

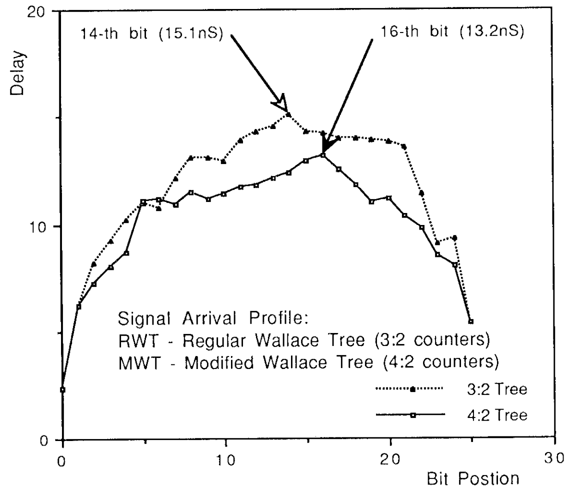


Fig. 7. Signal arrival profile for RWT and MWT.

The carry path is faster than the sum and this explains why the *peak* (maximum) of the resulting signal arrival profile is shifted to the right and slightly faster in MWT, as compared to RWT. This applies for the use of regular 4:2 counters. Use of the RDC will result in an even faster MWT.

In our implementation two types of Wallace trees: one using 3:2 (RWT) and the other using using 4:2 (MWT) counters are compared with respect to their delays and signal arrival profile. Their corresponding wiring diagrams are shown in figure 5. The use of 4:2 counters results in a more balanced signal arrival profile compared to the one using 3:2 counters (figure 7).

The distribution of delays in our multiplier design shows 13.2 nS from Booth encoding to the FA input (15.1 nS using RWT) as shown in figure 7. For multipliers with wider operands this difference would be larger. The difference is attributed to the change in the signal paths in the tree. If we use 4:2 counters or design special counters with higher compression ratios, the critical paths will be moving from the vertical direction toward horizontal, 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 as shown in [22].

4. The Final Adder

In the last stage of the multiplier we used a Variable Block Adder (VBA) [19]–[20]. The VBA scheme uses carry-skip technique to pass the carry over the group of bits (blocks) which are adjusted in size in such a way that the resulting delay is minimized. This scheme 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 can be fine-tuned to minimize the difference in delays introduced by the carry-paths of the different length. Sufficiently fast VBA sections are constructed and combined into a two section Carry Select Adder [18] as shown in figure 8. In our case, this optimization is done under the assumption that all of the input bits to the adder to not arrive at the same time. Obviously, the adder delay is dependent on the input arrival profile. The FA delays for the signals, arriving all in the same time (a) and for the input arrival profile resembling that from the Wallace tree (b) are shown in figure 9. We have optimized the Final Adder (GA) for the signal arrival profile originating from the Wallace tree.

Since the least significant bits are provided very soon, we can even use ripple adders without time penalty. In this case low-order VBA section provides sufficient speed. The results from the high-order section are needed as soon as the slowest bit (14) arrives. Therefore we use *select* on this portion of the FA. The bits after bit 14 arrive sooner than the 14-th bit. From that point on we need to use the fastest available scheme for addition. Therefore, we combined VBA addition in a 13-bit section with Carry Select Addition over the remaining 12 + 1 bit section. (The most significant portion of the adder is duplicated assuming the carry in signal equals either 0 or 1.) Carry out of the least significant section controls selection of the proper sum. The increase in size due to the duplicated part is not large because of the use of the VBA scheme. Determin-

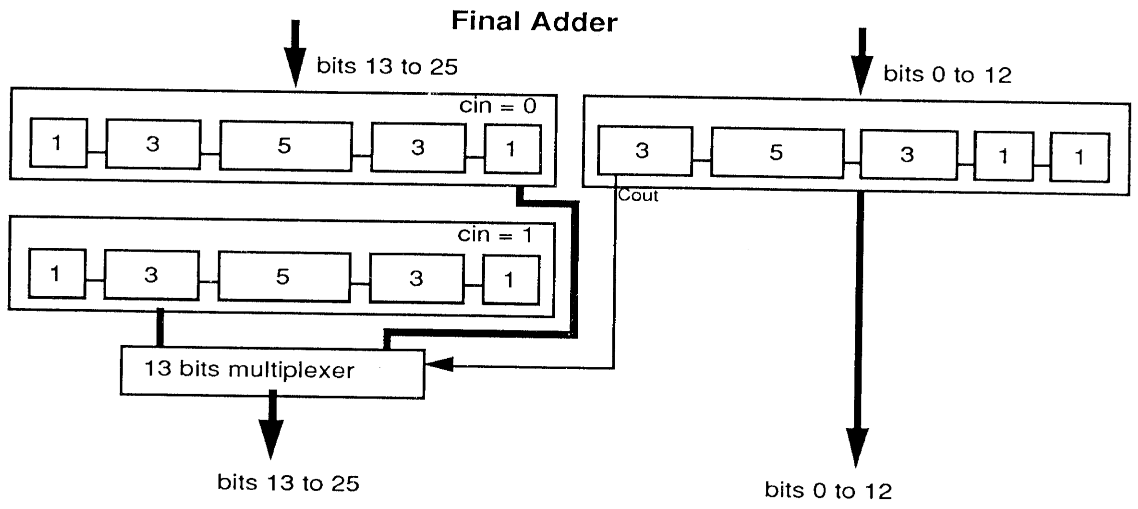


Fig. 8. Final adder for MWT multiplier.

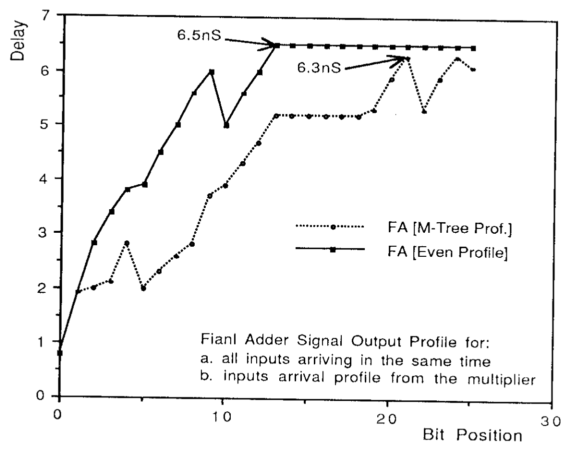


Fig. 9. Final adder delay for different signal arrival profiles to FA inputs.

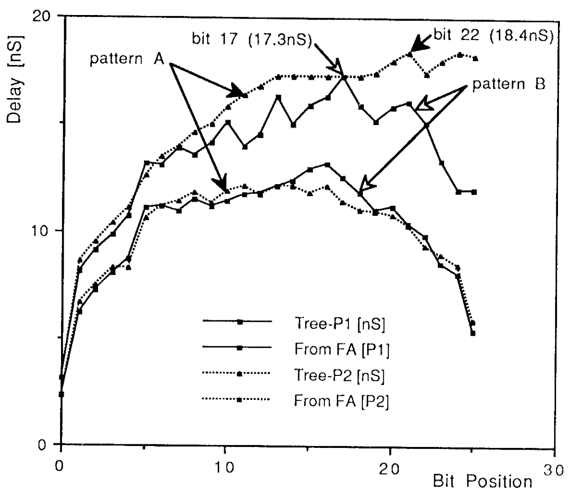


Fig. 10. Delay profile from the MWT and FA for two different input patterns A and B.

ation as how to divide the two sections of the adder is based on the following:

(a) first, we must obtain the delay profile resulting from the worst case delay of the multiplier. This is not necessarily the input combination which produces the worst case delay from the tree. We can observe this from figure 10, showing delay from the multiplier tree and final adder for the two patterns A and B. The input pattern B results in faster signal arrival profile from the multiplier tree, yet the delay after the FA is worse than the pattern A for which the signals from the multiplier tree arrive later.

(b) second, we divide the FA into two sections in such a way that the signals from the *higher-order* section are already in the same time the carry out of the *low-order* section arrives. This will produce flat output profile for the signals carrying the result. If they arrive after the carry signal from the least significant section, that means that the low-order portion is too small, which in turn increases the complexity of the FA. The right dividing point is the point at which those signals arrive at the same time (not before and not after).

As shown in [20] for short operand sizes, simple carry propagation schemes are the most efficient in terms of speed and size, and the use of more complex addition schemes is not warranted.

The critical path of the adder, i.e., the carry signal, is implemented as a string of multiplexers. This is

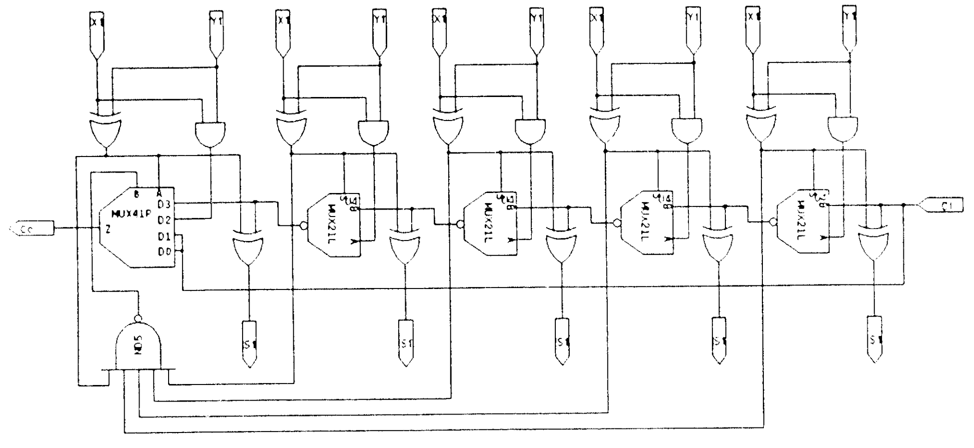


Fig. 11. Section of VBA final adder [20].

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 [20]. A section of VBA adder is shown in figure 11.

The associated signal delay profile from the MWT and from the FA are shown in figure 10. The worst case delay of such an adder is 6.5 nS (6.3 nS for the adder adjusted for the MWT). Given that the portion of the signal propagation is already absorbed in the MWT (RWT) delay, the FA in our design introduces an additional 6.3 nS to the multiplication time. This represents one third of the total multiplication time. Our total multiplication time is 18.4 nS.

The advantage of using 4:2 instead of 3:2 counters is self evident.

5. Improvements

Should the word-length in signal processors increase, further advantage could be taken by using counters of higher order [22]. An increase in the operand size should not result in a much slower circuit. Moreover, in this case, it is easier to find a better tree with more bits, simply because there are more possibilities. Also, in the last significant section of the final adder, different adders could be used, including Carry Look Ahead adder optimized for the signal arrival time [21]. The Carry-Select adder seems to be the best choice for the most significant section of the final adder. The appropriate step would be to design a complex multiplier tree, compare different adders for the trees and improve the Final Adder for longer operands. The 4:2 counter can be improved using the fact that no gate is symmetric in terms of delays of their inputs with respect to the output. This property applies even in the case of the XOR gate. There is also a possibility of improving the design of the tree in that way, but this would be partly at the expense of gain achieved by the asymmetric counter.

6. Conclusion

The structure of the complex multiplier presented in this article 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 and imaginary part of the complex number product. Considering that normalization and rounding are slow operations, we gained a great advantage by requiring only one rounding and normalization instead of two. In designing the Wallace tree we achieved a careful balance by using 4:2 instead of 3:2 counters.

The circuit has been designed using a 1.0 μ CMOS process and contains 10758 gates in LSI 100K technology. This technology was available to us and implementation using a current ASIC family would yield much better results. However, the basic findings and observations will still remain. We compared the results of our design with an ASIC implementation of a 16-bit multiplier-accumulator implemented in the LSI 10K ASIC technology [17]. Although we are not comparing the same design (complex number multiplier) and the size of the mantissas differ (13 vs. 16 bit), the differences can be estimated. The best implementation of the reported 16×16 integer multiplier-accumulator yields a result in 19.7 nS [17]. Our multiplier multiplies two complex numbers with 13-bit mantissas and yields a complex multiplication result in 20.8 nS in the same LSI 10K ASIC technology. Given that our Wallace tree is more complex, summing the partial products of two

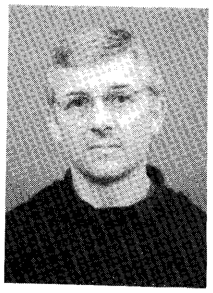
13×13-bit product terms instead of one those numbers are comparable. The advantage of our approach is due to the use of a better and optimized Wallace tree (using 4:2 adders) and also in optimization which is done across the Wallace tree and Final Adder (tuning the adder to the Wallace tree). We have not optimized the paths by *powering* the critical paths via *buffering* as it was done in [17]. However, powering the critical paths would give us an additional advantage in speed. Finally, we feel that we have not only demonstrated a concept but have provided a useful macro-cell component which is a core cell in the implementation of many Digital Signal Processing algorithms and applications.

Acknowledgment

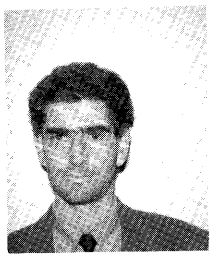
Generous support from LSI Logic Corporation, Robert Ulicki in particular, is gratefully acknowledged. Assistance of Simon Liu and careful reading of Lance Halsted as well as comments provided by the reviewer No. 3 improved the context of this article very much. This work has been supported by a Faculty Research Grant from the Office of Research, University of California Davis.

References

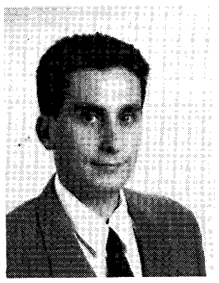
1. N.M. Marinovich, and V.G. Oklobdzija, "VLSI Chip Architecture for Real Time Ambiguity Function Computation," 25th Asilomar Conference on Signals, Systems and Computers, November 4–6, Pacific Grove, 1991.
2. E.E. Swartzlander, Jr., "Merged Arithmetic," *IEEE Transactions on Computers*, Vol. C-29, No. 10, 1980, pp. 946–950.
3. K.Z. Pekmestzi, "Complex Number Multipliers," IEE Proceedings (Computers and Digital Technology), Vol. 136, No. 1, 1989, pp. 70–75.
4. Akinwande et al., "A 500 MHz 16×16 Complex Multiplier Using Self-Aligned Gate GaAs Heterostructure FET Technology," *IEEE Journal of Solid-State Circuits*, Vol. 24, No. 5, 1989.
5. H.C. Shyu et al., "A Complex Integer Multiplier Using the Quadratic-Polynomial Residue Number System with Numbers of Form $2^{2n} + 1$," *IEEE Transactions on Computer*, Vol. C-36, No. 10, 1987, pp. 1255–1258.
6. J.E. Volder, "The CORDIC Trigonometric Computing Technique," *IEEE Transactions on Electronic Computers*, Vol. EC-8, 1959, p. 330.
7. J.S. Walther, "A Unified Algorithm for Elementary Functions," *SJCC*, 1971, p. 379.
8. H. Yoshimura et al., "A 50 MHz CMOS Geometrical Mapping Processor," *IEEE Transaction on Circuits and Systems*, 1989, p. 1360.
9. A.D. Booth, "A Signed Binary Multiplication Technique," *Q.J. Mech. Appl. Math.*, 1951, pp. 236–240.
10. C.S. Wallace, "A Suggestion for a Fast Multiplier," *IEEE Transaction on Electronic Computers*, Vol. EC-13, No. 1, 1964.
11. Luigi Dadda, "Some Schemes for Parallel Multipliers," *Alta Frequenza*, Vol. 34, No. 5, 1965.
12. W.J. Stenzel, and W.J. Kubitz, "A Compact High-Speed Parallel Multiplication Scheme," *IEEE Transaction on Computers*, Vol. C-26, No. 10, 1977.
13. A. Weinberger, "4:2 Carry-Save Adder Module," IBM Technical Disclosure Bulletin, Vol. 20, 1981.
14. Mark R. Santoro, "Design and Clocking of VLSI Multipliers," Technical Report No. CSL-TR-89-397, Stanford University, 1989.
15. Mark R. Santoro, and Mark A. Horowitz, "SPIM: A Pipelined 64×64-Bit Iterative Multiplier," *IEEE Journal of Solid State Circuits*, Vol. 24, No. 2, 1989.
16. J. Mori et al., "A 10 nS 54×54-b Parallel Structured Full Array Multiplier with 0.5-u CMOS Technology," *IEEE Journal of Solid State Circuits*, Vol. 26, No. 4, 1991.
17. K.F. Pang et al., "Generation of High Speed CMOS Multiplier-Accumulators," Proceedings of Int'l. Conference on Computer Design, Rye, New York, 1988.
18. O.J. Bedrij, "Carry-Select Adder," *IRE Transactions on Electronic Computers*, Vol. EC-11, No. 3, 1962, pp. 340–346.
19. V.G Oklobdzija, and E.R. Barnes, "Some Optimal Schemes for ALU Implementation in VLSI Technology," *7th Symposium on Computer Arithmetic ARITH-7*, 1985, Urbana, Illinois.
20. V.G. Oklobdzija, and E.R. Barnes, "On Implementing Addition in VLSI Technology," *Journal of Parallel Processing and Distributed Computing*, No. 5, 1988, p. 716.
21. B.D. Lee, and V.G. Oklobdzija, "Delay Optimization of Carry-Lookahead Adder Structure," *Journal of VLSI Signal Processing*, Vol. 3, No. 4, 1991.
22. V.G. Oklobdzija, and David Villeger, "Multiplier Design Using Improved Column Compression Tree and Optimized Final Adder in CMOS Technology," 1993 International Symposium on VLSI Technology, Systems and Applications, 1993, Taipei, Taiwan.



Vojin G. Oklobdzija joined the department of Electrical and Computer Engineering, University of California, Davis in July 1991. Previously he was with the IBM T.J. Watson Research Center in New York where he worked on development of RISC and super-scalar RISC architecture and processors. In 1986 he was one of the initiators of the super-computer project at IBM. From 1988–90 he was teaching courses in computer architecture, computer arithmetic and computer design at the University of California, Berkeley while on leave from IBM. He obtained Ph.D. and M.Sc. in Computer Science from the University of California, Los Angeles in 1982 and 1978, respectively, where he came as a Fulbright scholar in 1976. In 1991 he spent time in Peru lecturing and assisting in creation of their university programs as a Fulbright professor. He obtained Dipl.Ing. (M.Sc.EE) degree in electronics and telecommunication from the Electrical Engineering Department, University of Belgrade, Yugoslavia in 1971 and stayed on the faculty until 1976. His industrial experience includes positions at the microelectronics center of Xerox Corporation, consulting positions at Sun Microsystems Laboratories and various others. His interest is in VLSI and fast circuits, in particular efficient implementations of algorithms and computation and his work on fast ALU scheme has been widely referenced. He has also worked on high performance architectures and is a co-holder of a patent on the IBM RS/6000 RISC super-scalar architecture. He holds four U.S.A. and European patents in the area of computer design and has published in the areas of: circuits and technology, computer arithmetic and computer architecture. He has over 40 technical publications and has given over 50 invited talks in the U.S.A., Europe, Latin America, Australia and Japan.



David Villeger studied electronics at Ecole Superieure d'Ingenieurs en Electrotechnique et Electronique, Paris, France. He chose to specialize in microelectronics and received "Diplome d'Ingenieur" in July of 1993 after a 6 month project at the University of California, Davis, U.S.A. He spent time in Japan where he gave presentations at Yokogawa Denki Corporation in Tokyo and at Osaka University, Japan. His current interest is in computer arithmetic and VLSI. He worked on square root and multiplication algorithm in France and U.S.A. and he has published three papers on the subject.



Thierry Soulas received Diplome d'Ingenieur degree in July of 1993 from the Ecole Superieure d'Ingenieurs en Electrotechnique et Electronique, Paris, France. He spent the Summer of 1992 at the Electrical and Computer Engineering Department, University of California, Davis where he worked on design of a fast parallel multiplier for complex numbers. Since January of 1993 he has been working at Yokogawa Electric Corporation in Tokyo, Japan where he has been applying HDL methodology in design of arithmetic circuits.