An ASIC Macro Cell Multiplier for Complex Numbers

Thierry Soulas, David Villeger
Ecole Superieure d'Ingenieurs en Electrotechnique et Electronique
93162 Noisy le Grand CEDEX, FRANCE
soulas@apo.esiee.fr, villeger@apo.esiee.fr
33-1-45-92-65-00

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

Abstract

An architecture for ASIC macro-cell implementing a complex number multiplier with applications in a digital signal processing ASIC chip is described. The complex numbers are packed into one 32-bit word. The design is unique and combines shared Booth encoding for the real and imaginary parts including only one combined modified Wallace tree. We compared the regular Wallace tree and the tree of 4:2 adders for the complex multiplier implementation. We took advantage of 4:2 adders in implementing the combined bit compression tree for each part. This design resulted in a more compact wiring structure and balanced delays resulting in faster multiplier circuit. The number of adders was also decreased.

1. Introduction

Flexibility and rapid turn-around design offered by ASIC technology has made implementation of Digital Signal Processing (DSP) algorithms in hardware very viable and popular. Using advanced CAD tools, especially through the use of VHDL and Logic Synthesis, it became relatively easy to map a particular DSP algorithm into a specific hardware. ASIC technology is almost an ideal platform for implementation of these algorithms in hardware, since it is relatively easy to modify and recompile the design, due to the fact that ASIC requires fewer mask levels than a custom design process. However, to be viable, it is very important for an ASIC technology to be supplemented and supported with a comprehensive library of proven, efficiently designed high-speed macro cells used in the design process. One of very frequent operations in DSP algorithms is multiply-add or simply multiplication of complex numbers. In our specific case, we have concentrated on the multiplication of complex numbers, and design of a multiplier for complex numbers in particular. This specific multiplier is designed to be a macro-cell in a particular implementation of a VLSI chip that evaluates the cross-ambiguity function [1,2].

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:

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

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 multiplications by calculating the real and imaginary parts in the following way:

$$\mathrm{Re}[P] = (a-b)d + a(c-d) \quad \text{and} \quad \mathrm{Im}[P] = (a-b)d + b(c+d) \quad \text{...(2)}$$

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

It is also possible to use 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. [3,4,5]. We have not considered CORDIC for this implementation, though this algorithm warrants further studies for the DSP computation due to its specific properties.

In this paper, we describe a new design for a fast complex number multiplier. By using a specific final adder optimized for the specific signal arrival time from the Wallace tree and sharing both, the Booth encoding and Wallace tree, we can improve size, speed and accuracy. The architecture of the complex multiplier is described in Section II. We compare different trees in Section III. Section IV describes the specific adders. We suggested some improvements in Section V. Section VI is the conclusion.

2. Architecture

The specifics of a design of this complex multiplier require that 32-bit inputs are used with the specific floating point number representation given in Fig. 1. In order to efficiently pack a complex number representation into a 32-bit word, real and imaginary parts share the same exponent. This is done to efficiently utilize memory and provide compatibility with the available standard buses and memory hardware. It is also considered that this format

1066-1409/93 $03.00 © 1993 IEEE

provides an ample range and sufficient precision for most DSP real time applications. By using this representation and the IEEE standard normalization, we eliminated the need to normalize after each multiplication. Because normalization and routing are slow processes, our approach is advantageous because it uses only one routing and normalization after the final adder.

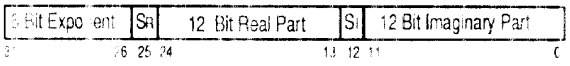


Figure 1 Representation format

The layout and wireability of the complex number multiplier were one of our main concerns. This is because a lot of the hidden cost in terms of the multiplier delay is contained in these areas. A study done by Santoro has provided useful guidelines for the case of the CMOS technology [6,7]. His findings can be summarized that in terms of the size the layout area of the multiplier is roughly divided into four quarters: Booth encoding logic, Wallace tree, Final Adder and Wires.

In our approach, we have used a straightforward multiply-add operation as outlined in [1,2] and savings in the number of components are achieved by sharing the common product terms and therefore the Booth encoders for the operands **a** and **b**. In addition, we combined the Wallace trees for the two partial products **ac** and **bd**, and **bc** and **ad** respectively. By the measures provided by Santoro, our design resulted in a greater than 10% reduction in terms of the size and more regular layout of the multiplier.

The multiplication process of the two complex numbers is performed in a standard procedure for the multiplication of the floating point numbers. The exponents and mantissas are separated and the exponent parts are added together. The real and imaginary part mantissas are multiplied sharing Booth encoders and using a common Wallace tree. We chose to use Booth encoding of the multiplier because this allow as to share the common terms. 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 originating from the Wallace tree. The main structure of the multiplier is shown in Fig. 2.

Wallace trees are used for the bit compression of both partial products, thus avoiding the use of separate adders as shown in Fig.3. The structure of the Wallace tree is not regular and the signals arrive to FA in different times depending on the bit position. It has been known that the bits closer to the ends of the Wallace tree will arrive sooner. In our design the final adder is specific to the data arrival.

The speed of the encoder does not depend on the size of the operands, but only on different fan-outs. However, the speed of the Wallace tree is critical and depends on the number of levels which is related to the size of the operands. In our specific case, we had two solutions for adding the partial products.

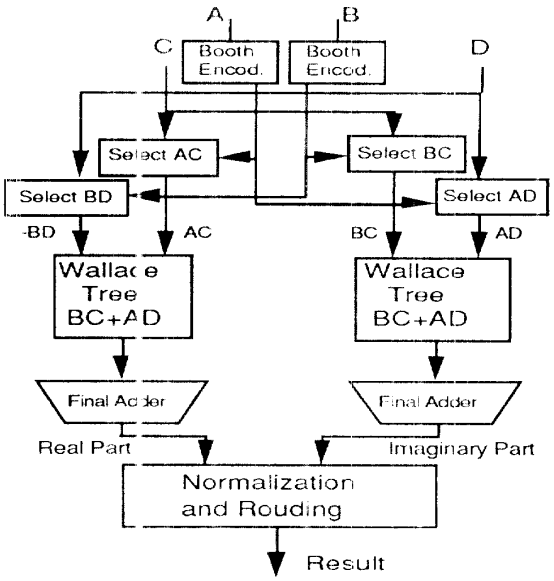


Figure 2: Multiplier Organization

The most direct solution would be to implement Wallace trees separately for partial products and add the results separately as shown in Fig.3a. The better solution is however, to use separate Wallace trees, as shown in Fig.3b, thus avoiding large increase in the number of levels.

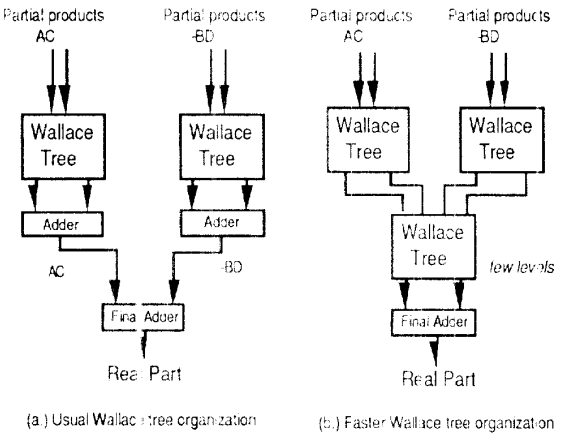


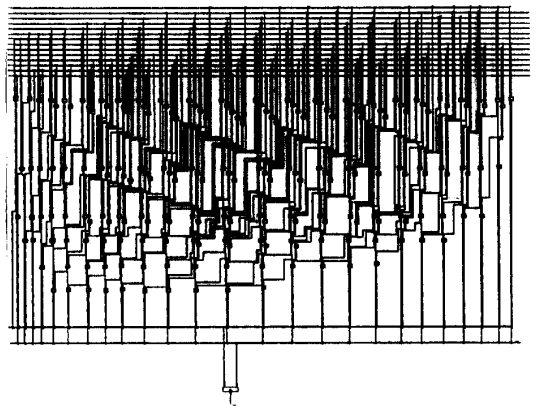
Figure 3 : Structure of the Wallace trees.

As shown by Dadda, the ability to reduce the number of levels is determined by the type of the general counter chosen [8,9]. Because our operands are only 13 bits long, the second solution is not advantageous. Therefore, we

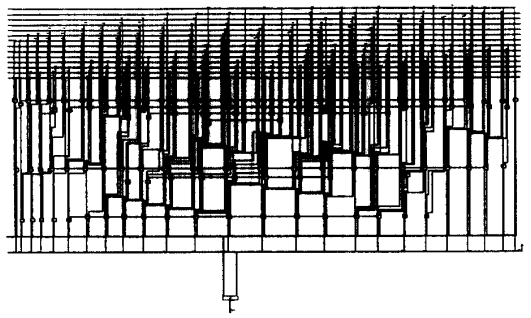
implemented one common Wallace tree for bit reduction of both partial products, as shown in Fig. 2. The final adder is also critical because of the carry propagation time. Indeed, the time required for final addition is almost equal to the third of total multiplication time. A common way to speed up addition is the well-known Carry-Look-Ahead (CLA) method. However, as shown in [10], it is possible to implement a Variable Block Adder (VBA) of the speed comparable to the CLA and complexity of the Carry Skip Adder by varying block sizes and optimizing for the different arrival times of the carry signal in the critical path. We used this method in order to adjust the final adder block sizes to the specific signal arrival time profile originating from the Wallace tree.

3. The Different Trees

Usually Wallace tree is implemented from the full adders. The trees are not regular and they are difficult wire, as shown in Fig.4.a. Santoro has shown that the Wallace tree can be better in terms of wireability by using 4:2 adders (ST), as shown in Fig.4.b [6]. This adder has actually 5 inputs (4 partial products and a carry input) and 3 outputs (2 carry outputs and the sum).



(a) RWT



(b.) MWT

Figure 4: Wiring diagram of the Wallace Tree

The interesting feature of this adder is that the carry output does not depend on the carry input and 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 (SC) results in the longest path equivalent to 4 XOR gate delays. Another multiplier design from Toshiba Corp. [12] uses a specially designed 4:2 counter with reduced delay. The 7:3 adder is similar to the use of full adders, but it is more efficient. The 9:3 adder is similar to 4:2 adder however, its use is difficult because it requires more bits for its efficient utilization. Therefore, we considered only Wallace tree built from full adders and a tree of 4:2 adders as proposed by Santoro.

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 Regular Wallace Tree (RWT) built from the full adders. However, in terms of the gate delays each 4:2 counter stage of SC is similar to a delay equivalent to 4 XOR gates. On the other hand, 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 [6]. To take full advantage of 4:2 counters, we designed a special 4:2 counter cell called Reduced Delay Counter (RDC) using available ASIC cells, resulting in only 3 XOR equivalent gate delays. RDC cell is shown in Fig.5. The resulting RDC 4:2 cell is only slightly 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. The signals at the end bit positions from MWT tree arrive almost at the same time. The signal arrival profiles for the RWT and MWT are shown in the Fig.6.

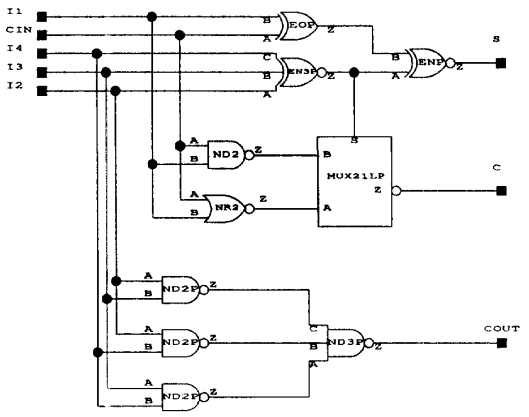


Figure 5: Logic of the Reduced Delay Counter RDC

On the other hand, the final adder can not be tuned as well as with RWT and advantage achieved by optimizing the FA for the specific signal arrival profile is somewhat diminished. The resulting multiplier show an overall advantage in speed over the one using RWT as shown in Fig.8. 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 modified tree in terms of the delay over 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 RDC shows sensitivity to the inputs, ie. 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.
The distribution of delays in our multiplier design shows 15.6 nS from Booth encoding to the FA input (17.4 nS using RWT) as shown in Fig. 6.

4. The Final Adder

For the final adder, we used Variable Block Adder [10] because this scheme achieves good performance without using much additional logic. We construct sufficiently fast VBA sections which are combined into a two section Conditional Sum Adder [11]. The low order bit VBA section is tuned for the signal arrival time from bits 0-12 of the MWT while the higher order section is optimized for signal arrival from bit positions 13-25. The optimized FA resulted in two equal 13-bit sections applied to MWT and 12 and 14 bit sections for the RWT (12-bit section being the least significant portion of the product). This reduces the critical path and permits high speed calculation of the sum.

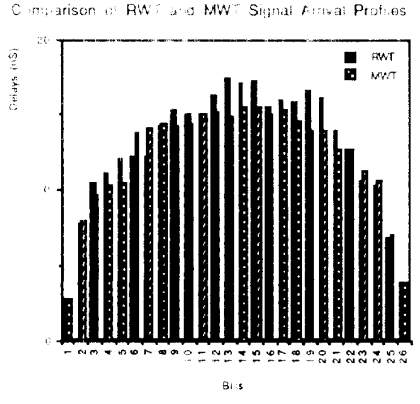


Figure 6: Signal Arrival Profile for RWT and MWT

The signal arrival times before entering the FA are shown in Fig.6. Since the least significant bits are provided very soon, we can use ripple adders without time penalty. The bits after the bit 13 arrive sooner than the 13th bit. Therefore, we combined VBA addition in a 12-bit section with Conditional-Sum Addition over the remaining 12+1 bit section. (The most significant portion of the adder is duplicated assuming carry in signal equals either 0 or 1.) Carry out of the least significant section controls selection of the proper sum. The size of the duplicated part is not large due to the use of the VBA scheme. Also, as shown in [11] 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 Final Adder is shown in Fig.7.

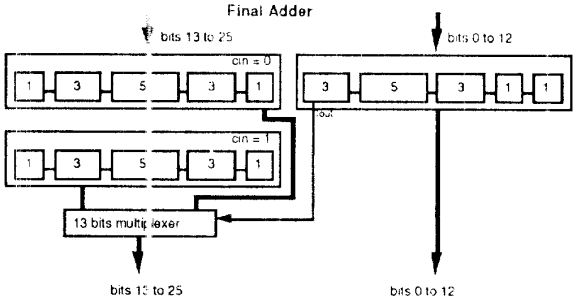


Figure 7: Final Adder for MWT Multiplier

The associated signal delay profile from the MWT and from the FA are shown in Fig.8. The worse case delay of such adder is 8.2nS (8.3nS for the adder adjusted for the RWT). Given that the portion of the signal propagation is already absorbed in the MWT (RWT) delay, the FA in our design introduces only an additional 5.5 nS to the multiplication time. This represents a quarter of the total multiplication time. Our total multiplication time is 20.8 nS.

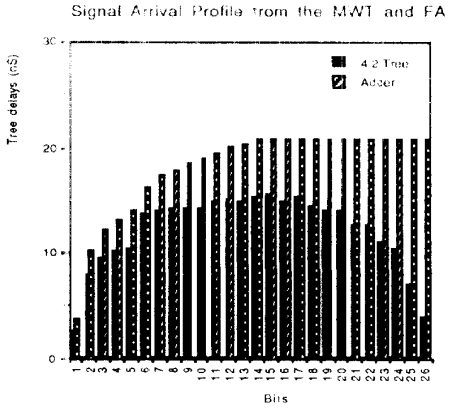


Figure 8: Delay profile from the MWT and FA

5. Improvements

In case of the long operands, increase in the operand size, should not be resulting in much slower circuit. Moreover, in this case, it is easier to find a better tree with more bits, simply because there are more possibilities. Therefore, the appropriate step would be to design a complex multiplier tree, compare different adders for the trees and improve the Carry Skip Adder blocks with longer operands. The 4:2 adders can be improved using the fact that no gate is symmetric in terms of delays of their inputs with the respect to the output. This property applies even in case of the XOR gate.

In the post normalization stage, improvement should not be difficult given that the last step consists of the adjustment of the exponent and the shift of the fractions. It is also possible to use the Conditional Sum Adder for this operation. We can reduce the size of the final adder by using only the carry propagation on the right part. Indeed, we know that the result do not need to be shifted more than five bits.

6. Results and Conclusion

The circuit has been designed using a 1.5μ CMOS process and contains 10758 gates in LSI 10K technology. This technology was available to us and implementation using 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 16-bit multiplier-accumulator implemented in the same ASIC technology [13]. 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 reported [13] 16X16 integer muliplier-accumulator yields a result in 19.7 nS. Our multiplier multiplies two complex numbers with 13-bit mantissas and yields a complex multiplication result in 20.8 nS in the same ASIC technology. This is substantially faster, given that our Wallace tree is more complex, summing the partial products of two 13X13-bit product terms. The advantage of our approach is due to the use of 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 into the Wallace tree). We have not optimized the paths by powering the critical paths "buffering" as it was done in [13]. However, powering the critical paths would give us an additional advantage in speed. Finally, we feel that we have not only demonstrated a concept for designing fast parallel multiplier in ASIC technology but have provided an useful macro-cell component which is a core cell in the implementation of many Digital Signal Processing algorithms and applications.

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 Funmctions "*, SJCC, p. 379, April 1971.

[5] H. Yoshimura et al, *"A 50 MHz CMOS Geometrical Mapping Processor "*, IEEE Transaction on Circuits and Systems, p. 1360, October 1989.

[6] Mark R. Santorio, *"Design and Clocking of VLSI Multipliers"*, Technical Report No. CSL-TR-89-397, Stanford University, October 1989.

[7] 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.

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

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

[10] 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.

[11] 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.

[12] J. Mori et al, *"A 10nS 54X54-b Parallel Structured Full Array Multiplier with 0.5-u CMOS Technology"*, IEEE Journal of Solid State Circuits, Vol. 26, No. 4, April 1991.

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