A low logic depth complex multiplier using distributed arithmetic

Berkeman, Anders; Öwall, Viktor; Torkelson, Mats

Published in:
IEEE Journal of Solid-State Circuits

DOI:
10.1109/4.839928

2000

A Low Logic Depth Complex Multiplier Using Distributed Arithmetic

Anders Berkeman, Viktor Öwall, and Mats Torkelson

***Abstract*—A combinatorial complex multiplier has been designed for use in a pipelined fast Fourier transform processor. The performance in terms of throughput of the processor is limited by the multiplication. Therefore, the multiplier is optimized to make the input-to-output delay as short as possible. A new architecture based on distributed arithmetic, Wallace-trees, and carry-lookahead adders has been developed. The multiplier has been fabricated using standard cells in a 0.5-μm process and verified for functionality, speed, and power consumption. Running at 40 MHz, a multiplier with input wordlengths of 16+16 times 10+10 bits consumes 54% less power compared to an distributed arithmetic array multiplier fabricated under equal conditions.**

***Index Terms*—Complex multiplier, digital CMOS, distributed arithmetic.**

I. Introduction

COMPLEX multiplication is one of the most time-critical and area-consuming operations in a digital signal processor. Therefore, effort has to be made to decrease the number of multipliers and to increase their speed. A pipelined fast Fourier transform (FFT) processor has been designed for use in an orthogonal frequency-division multiplex (OFDM) system. In the designed FFT processor the critical path consists of a complex multiplier in series with a butterfly unit performing addition and subtraction. A part of the FFT pipeline is shown in Fig. 1. Since the butterfly processors are much faster than the complex multiplier, the maximum clock frequency of the processor strongly depends of the multiplier delay. This paper presents a novel multiplier architecture based on distributed arithmetic and Wallace trees. The multiplier is fully parameterized, so any configuration of input and output wordlengths could be elaborated.

II. The FFT Processor

In the early versions of the FFT processor, a complex array multiplier was used [1]. The array multiplier is a highly regular structure resulting in a minimal wire length, which is important for high-speed design in submicrometer processes where wiring delay gives a significant contribution to the overall delay. However, in a process where cell delay dominates wire delay, the logic depth of the design is more important than regularity. In the complex array multiplier the logic depth is $\mathcal{O}(\mathcal{N})$, where N is the input wordlength. In the adder tree multiplier, on the other hand, the depth is $\mathcal{O}(\log \mathcal{N})$ [2]. Even for short wordlengths, this leads to a substantial reduction in delay.

Manuscript received July 6, 1999; revised November 5, 1999.

The authors are with the Department of Applied Electronics, Lund University, Lund SE-221 00 Sweden (e-mail: abn@tde.lth.se; vikt@tde.lth.se; torkel@tde.lth.se).

Publisher Item Identifier S 0018-9200(00)02865-1.

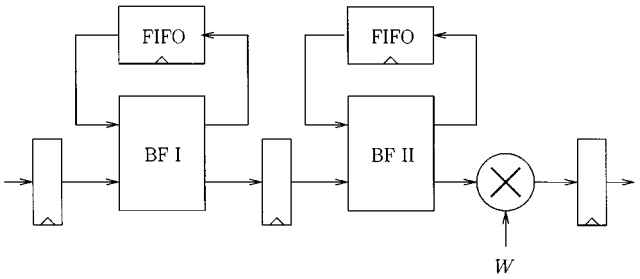

Fig. 1. Part of the R2^{2}DIF FFT processor pipeline. The butterfly processors are named "BF I" and "BF II". Shaded boxes are combinatorial blocks without registers.

A way to decrease the critical path of the FFT processor would be to pipeline the multiplier into two or more stages. However, due to the pipelined structure of the FFT processor, complexity of the controlling hardware would increase [3]. Furthermore, the wordlengths of the datapaths are wide, due to the application of the processor, and all operators use complex arithmetic. A multiplier in this application has between 44 and 52 input bits, and a pipeline register inserted somewhere in the middle of the multiplier would need a wordlength of more than a hundred bits, due to the internal "carry save" number representation. This would increase area, routing, and clock load and is not a preferable solution. Instead, the multiply operation is entirely combinatorial.

The FFT processor is implemented using the R2^{2}DIF FFT-algorithm [3]. In this algorithm, every second multiplication can be exchanged to a multiply by $-j$, which for an 8192-point FFT leaves only six complex multipliers. This is to be compared to 12 using a straightforward radix-2 implementation. The multiplication by $-j$ is realized without a multiply by real-imaginary swap and negation of the imaginary part. This is the reason for the two different butterfly processors, "BF I" and "BF II," in Fig. 1. By using this algorithm, the number of instantiated multipliers is minimized compared to an ordinary radix-2 FFT without any loss in throughput.

III. Multiplier Algorithm

A complex multiplier calculates two inner products

$$\begin{cases} Z_R = A_R W_R - A_I W_I \\ Z_I = A_R W_I + A_I W_R. \end{cases} \tag{1}$$

In the case of the FFT processor, $W = W_R + jW_I$ are the twiddle factors stored in a ROM. The wordlength of W_R and W_I is denoted M. According to (1), four real multiplications and two additions are required. With the exception of logic minimization, there are two methods to decrease multiplication logic depth if it is assumed that multiplication is performed by summation of partial products. The first is to reduce the number of partial products and the second is to use a faster adder strategy

TABLE I
RELATION BETWEEN THE BITS a_{R_i}, a_{I_i} AND THE PARTIAL PRODUCTS

α_{R_i}	α_{I_i}	a_{R_i}	a_{I_i}	$W_I\alpha_{R_i} + W_R\alpha_{I_i}$	$W_R\alpha_{R_i} - W_I\alpha_{I_i}$
−1	−1	0	0	$-W_\Sigma$	$-W_\Delta$
−1	1	0	1	W_Δ	$-W_\Sigma$
1	−1	1	0	$-W_\Delta$	W_Σ
1	1	1	1	W_Σ	W_Δ

to sum all the partial products together [2]. Both methods have been combined in the presented architecture. Distributed arithmetic [4]–[6] was chosen as a means to reduce the number of partial products. A Wallace tree adder was selected for adding the partial products together, since it has a low logic depth resulting in fast addition.

By using distributed arithmetic, the complex multiplication is treated as two independent inner products Z_R and Z_I. Each of the two inner products will be calculated using one distributed arithmetic multiplier, as compared to a direct realization of (1), which requires four real multipliers. In the equations that follow, a bit variable is treated as an integer variable holding the arithmetic value of 0 or 1. In this way, bits can be used together with arithmetic variables and operators. If A is an N-bit fractional number in two's complement, the value of A is calculated according to

$$A = -a_0 + \sum_{i=1}^{N-1} a_i 2^{-i}. \tag{2}$$

By using the identity

$$A = \frac{1}{2}(A - (-A)) \tag{3}$$

and the rule for negating a two's complement number, $-A = \bar{A} + 2^{-(N-1)}$, (2) can be written as

$$A = -(a_0 - \overline{a_0})2^{-1} + \sum_{i=1}^{N-1}(a_i - \overline{a_i})2^{-i-1} - 2^{-N}. \tag{4}$$

Introduce $\alpha_0 = (\overline{a_0} - a_0)$, and for $k \neq 0$, $\alpha_k = (a_k - \overline{a_k})$, note that all $\alpha_k \in \{-1, +1\}$. Using this notation, A can be written as

$$A = \sum_{i=0}^{N-1} \alpha_i 2^{-i-1} - 2^{-N}. \tag{5}$$

The relationship between a_i and α_i is shown in Table I. Using this encoding, the complex product can be expressed as

$$Z_R = \sum_{i=0}^{N-1} (W_R\alpha_{R_i} - W_I\alpha_{I_i})\, 2^{-i-1} - (W_R - W_I)2^{-N} \tag{6}$$

and

$$Z_I = \sum_{i=0}^{N-1} (W_I\alpha_{R_i} + W_R\alpha_{I_i})\, 2^{-i-1} - (W_I + W_R)2^{-N} \tag{7}$$

where $\alpha_i = \alpha_{R_i} + j\alpha_{I_i}$. The expressions $W_R\alpha_{R_i} - W_I\alpha_{I_i}$ and $W_I\alpha_{R_i} + W_R\alpha_{I_i}$ are for $i \neq 0$ examined in Table I, where the twiddle factors have been transformed from W_R and W_I to W_Σ and W_Δ according to

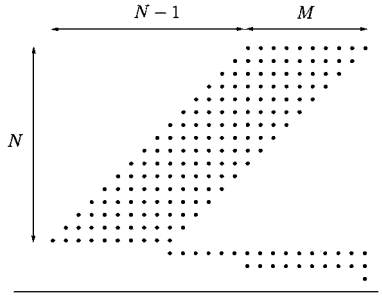

Fig. 2. Partial product bits by significance for Z_R or z_I. Input wordlength is N and coefficient wordlength is M. Each dot represents one partial product bit.

$$\begin{cases} W_\Sigma = W_R + W_I \\ W_\Delta = W_R - W_I. \end{cases} \tag{8}$$

This transformation does not cause any problems in the implementation, since the twiddle factors are precalculated in the W_Σ and W_Δ format before realization. From Table I, it is clear that $p_i = (a_{R_i} \oplus a_{I_i})$ can be used to select W_Σ or W_Δ. Treating p_i as integers holding the values 0 or 1 and $\vee$ as a bitwise inclusive-OR operator, (6) and (7) can be written as

$$Z_R = \sum_{i=0}^{N-1} (\overline{a_{R_i}} \oplus (p_i W_\Sigma \vee \overline{p_i} W_\Delta) + \overline{a_{R_i}})2^{-i-1} - W_\Delta 2^{-N} \tag{9}$$

and

$$Z_I = \sum_{i=0}^{N-1} (\overline{a_{I_i}} \oplus (p_i W_\Delta \vee \overline{p_i} W_\Sigma) + \overline{a_{I_i}})2^{-i-1} - W_\Sigma 2^{-N}. \tag{10}$$

When evaluating the sums, the powers $\overline{a_{R_i}}$ and $\overline{a_{I_i}}$ should be replaced with a_{R_i} and a_{I_i} for the case $i = 0$, since these bits have negative weight in two's-complement representation. The partial inner products

$$\overline{a_{I_i}} \oplus (p_i W_\Delta \vee \overline{p_i} W_\Sigma) + \overline{a_{I_i}} \tag{11}$$

and

$$\overline{a_{R_i}} \oplus (p_i W_\Sigma \vee \overline{p_i} W_\Delta) + \overline{a_{R_i}} \tag{12}$$

are suitable for hardware mapping. They are realized as multiplexers selecting W_Σ or W_Δ, depending on the value of p_i. The bits a_{R_i} and a_{I_i} conditionally negate the outputs of the multiplexers by inverting and adding a '1' in the least significant position. Fig. 2 shows all the partial product bits that has to be added to generate Z_R or Z_I. The wordlength for the twiddle factor W is M bits, and for the data A it is N bits, in this case 10 and 16 bits, respectively. The top 16 lines in the figure are the partial products generated inside the sum of (9) or (10), and the third line from the bottom is the ones that form the corresponding two's-complement of these products. The last two lines are the $-W_\Sigma$ or $-W_\Delta$ times 2^{-N} term.

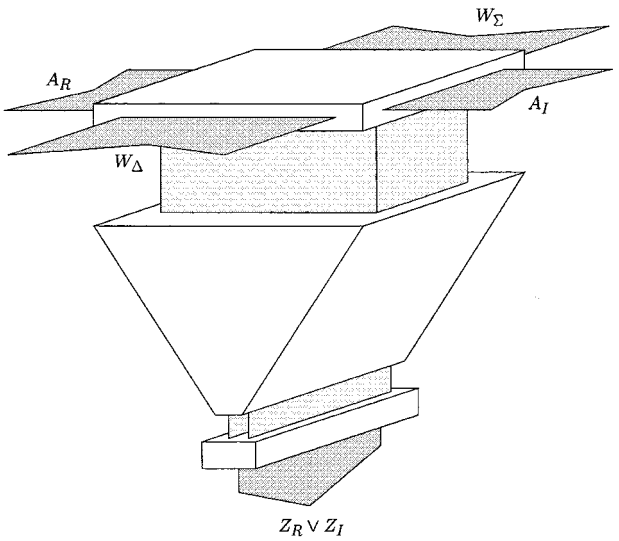

Fig. 3. The multiplier for Z_R or Z_I; the complete complex multiplier consists of two of these. Partial inner product generator at the top, adder tree in the middle, and fast carry-lookahead adder at the bottom.

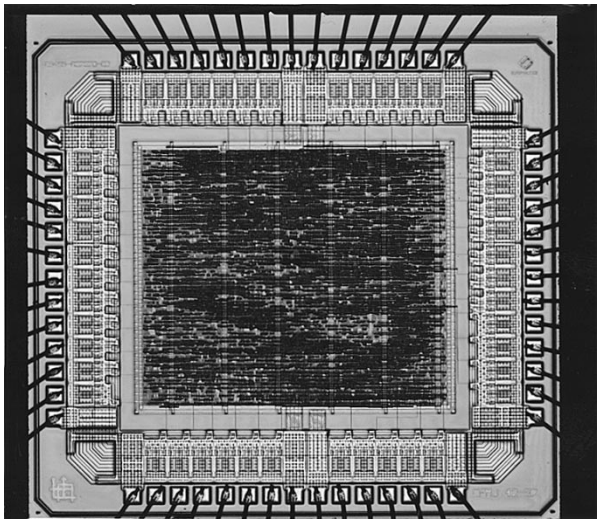

Fig. 4. Chip microphotograph of the proposed multiplier. The pad frame is 3.2 $\times$ 2.9 mm^2 and equal for both designs. Core size is 3.59 mm^2.

As an alternative to distributed arithmetic, modified Booth encoding was considered [7], [8]. However, as the number of partial products are about the same for both methods, modified Booth encoding requires more logic gates to implement. This is due to the fact that in the modified Booth algorithm, three variables have to be decoded to select the proper partial product. In a complex multiplier based on distributed arithmetic, a simple two-input XOR-gate implements the selection.

IV. Implementation

The proposed multiplier consists of two distributed arithmetic blocks, one calculating Z_R and the other Z_I. The two blocks are similar, and the difference is basically the minus in (1). Each block is divided into three parts: partial inner product generator, adder tree, and carry lookahead adder; see Fig. 3.

When designing the adder tree, a generic tree generator was used. This generator produces a tree with y inputs of wordlength x, that is, a rectangle of x by y input bits. This rectangle has to be large enough to cover all the partial product bits of Fig. 2, i.e., $x = M+N-1$ and $y = N+3$. For certain sizes of N and M, the two last lines in Fig. 2 can be joined with two of the N first lines, minimizing y to $N+1$. Unfortunately, almost 50% of the inputs to the adder tree are unused or used for sign extension only, and extra logic will be generated. Therefore, the area for the tree multiplier is approximately 75% larger than for the array multiplier. The number of gates for the array multiplier is 3000, while the tree multiplier uses 6200 gates, of which 4400 belong to the two adder trees. Theoretically, the area for a dedicated tree generator should be only slightly larger than for the array multiplier.

When data flows through the pipeline of the FFT processor, the wordlength has to increase to keep accuracy in the calculations. For the current application, the input wordlength is 12 + 12 bits (real + imaginary) and the output wordlength is 16 + 16 bits. The twiddle factors are kept constant at 10 + 10 bits at all stages of the pipeline. Different wordlengths in the datapath means that a set of multipliers of different wordlengths has to be instantiated if the longest wordlength is not to be used for all multipliers with a corresponding increase in area. Also, as FFT processors will be built for different applications, the wordlength is subject to change. Therefore, the multiplier is fully parameterized and a multiplier of specific wordlength can be elaborated when needed.

For the FFT application, the output wordlength should equal the input wordlength, i.e., some of the least significant bits of the result are cut away. A simple rounding scheme is applied to lower the distortion when the output is truncated. A rounding bit is added to the right of the rightmost bit to be kept after truncation, causing a carry to propagate when the most significant position of the bits cut away is a one. A feature of the adder tree is that this bit can be inserted together with the partial inner products at the top of the tree; see Fig. 2. In the array multiplier, an additional row of half-adders had to be included to handle rounding. As rounding includes addition of a one with the product, arithmetic overflow at the output is possible. Therefore, a saturation unit is placed at the output of the carry-lookahead adder. This unit checks the most significant bits of the result and saturates the output if an overflow has occurred.

Both the proposed multiplier and a multiplier based on a regular adder array have been fabricated under equal conditions for comparison. Chip microphotograph for the tree multiplier is shown in Fig. 4. The multipliers were fabricated using a three-metal-layer, 0.5-μm cell library that does not contain any dedicated half or full adder cells. Such cells could be beneficial in a multiplier architecture due to the large amount of additions in the algorithm. Instead, adders are realized using basic gates such as two-input XOR-gates.

Simulations show that 55% of the total delay in the critical path is due to the adder tree. The partial inner product generator contributes with 20% and the carry-lookahead adder 25% of the total delay. Simulation results have been presented in [9]. Most of the delay is spent in the adder tree, and by using dedicated adder cells, this delay can be decreased. However, the target cell

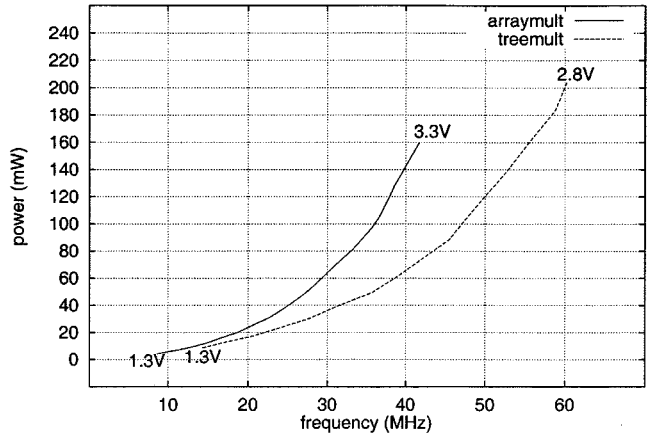

Fig. 5. Power versus frequency for the tree and the array multipliers.

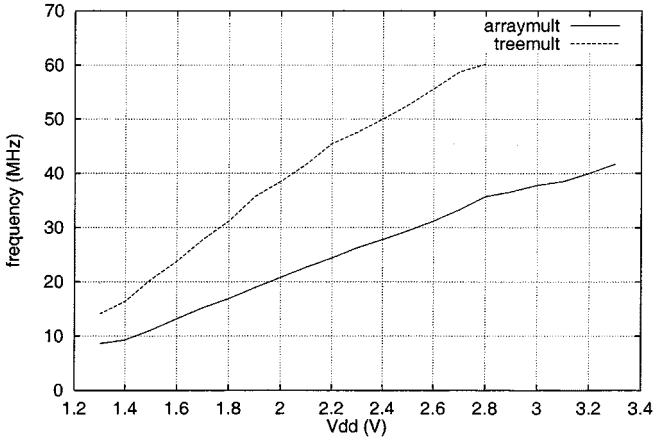

Fig. 6. Frequency versus supply voltage for the tree and array multipliers.

library does not contain any such cells and such improvements have not been implemented, which is the case for both designs.

V. Results

Both the array multiplier [1] and tree multiplier were fabricated on the same wafer in a 3.3-V, 0.5-μm, three-metal-layer CMOS process using standard cells. This process is neither a dedicated low-power nor high-speed process, but the enhanced performance is due to the architectural solution and should be applicable to more advanced processes as well. To keep the number of IO pads to a reasonable amount, the input was chosen to 16 real plus 16 imaginary times 10 real plus 10 imaginary bits. The most significant 16 real plus 16 imaginary product bits were output.

Power consumption and maximum operating frequency were measured over a supply voltage ranging from 1.3 to 3.3 V in steps of 100 mV. None of the circuits worked below 1.3 V. Power consumption was measured as the power dissipated in the core when the chip ran at maximum operational frequency for a given voltage. The measured results are plotted in Figs. 5 and 6. Fig. 5 shows power consumption versus operating frequency, and Fig. 6 shows maximum operating frequency versus supply voltage.

Maximum speed for the array multiplier was 41 MHz at 3.3 V, while the tree multiplier exceeded the 60-MHz limit of the test instrument when driven with 2.8 V. Due to the tradeoff between power and speed [10], the proposed multiplier is either faster than the array multiplier or, at equal speed, it is less power consuming.

VI. Conclusion

A Wallace-tree-based complex multiplier using distributed arithmetic has been designed, fabricated, and varified for functionality, speed, and power consumption. The multiplier is compared to a complex multiplier based on a regular array adder fabricated under equal conditions. The multipliers were fabricated in a three-metal-layer, 0.5-μm process on the same wafer using a standard cell library. This library does not contain any full or half adder cells that could be beneficial in a multiplier architecture, but all adders are built with basic gates.

At a frequency of 40 MHz, the proposed tree multiplier consumes 66 mW, which is 54% less than the array multiplier. Operating at the same voltage, the tree multiplier is 54% faster than the array multiplier. Maximum speed for the proposed multiplier is beyond 60 MHz at 3.3 V. Enhanced performance comes from a novel architecture and is transferable to more advanced processes. Performance could be further improved by using a dedicated adder tree. Since the multiplier dominates the critical path, the delay contribution from the adder or subtracter can be ignored, and throughput of the FFT processor is expected to increase by approximately 80%. The multiplier is fully parameterized so any configuration of input and output wordlengths can be elaborated and synthesized.

References

[1] S. He and M. Torkelson, "A complex array multiplier using distributed arithmetic," in *Proc. IEEE Custom Integrated Circuits Conf.*, 1996, pp. 71–74.

[2] C. S. Wallace, "A suggestion for a fast multiplier," *IEEE Trans. Electron. Comp.*, vol. EC-13, pp. 114–117, Feb. 1964.

[3] S. He and M. Torkelson, "A new approach to pipeline FFT processor," in *Proc. IEEE Int. Parallel Processing Symp.*, 1996, pp. 766–770.

[4] A. Croisier, D. J. Esteban, M. E. Levilion, and V. Riso, "Digital Filter for PCM Encoded Signals,", Dec. 4, 1973.

[5] S. G. Smith and P. B. Denyer, "Efficient bit-serial complex multiplication and sum-of products computation using distributed arithmetic," in *Proc. IEEE Int. Conf. Acoustics Speech and Signal Processing*, 1986, pp. 271–276.

[6] K. K. Parhi, *VLSI Digital Signal Processing Systems: Design and Implementation*. New York: Wiley, 1999.

[7] A. D. Booth, "A signed binary multiplication technique," *Q. J. Mech. Appl. Math.*, vol. 4, pp. 236–240, 1951.

[8] L. P. Rubinfield, "A proof of the modified Booth algorithm for multiplication," *IEEE Trans. Comput.*, vol. C-24, pp. 1014–1015, Oct. 1975.

[9] A. Berkeman, V. Öwall, and M. Torkelson, "A low logic depth complex multiplier," in *Proc. 24th IEEE Eur. Solid-State Circuits Conf.*, 1998, pp. 204–207.

[10] A. P. Chandrakasan and R. W. Brodersen, "Minimizing power consumption in digital CMOS circuits," *Proc. IEEE*, vol. 83, pp. 498–523, Apr. 1995.