ALGORITHM AND TWO EFFICIENT IMPLEMENTATIONS FOR COMPLEX MULTIPLIER

Yousf B. Mahdy *Associate Professor*

Samia A. Ali *Assistant Professor*

Khaled M. Shaaban *Assistant Professor*

Electrical Engineering Department
Faculty of Engineering
Assiut University, Egypt

ABSTRACT

Complex number operations are the backbones for many Digital Signal Processing (DSP) algorithms and several other scientific applications. In this paper, a complex multiplication algorithm and two of its realizations are provided. The first realization is a direct implementation of the algorithm. The second realization is complex but more efficient both hardware and time wise. This realization processes information in a pipeline fashion thus it is valuable for applications with long sequences of complex operations. The hardware and time saving of the second realization over the simple one is approximately forty percent. Furthermore, the two proposed implementations could be used to perform cyclic convolutions of two point sequences just by negating one of its inputs.

1. INTRODUCTION

Real number multiplication is one of the most critical and resource intensive computation operation. Thus, it has been and still is the center of attention of large number of research efforts [1-5]. Other types of multipliers such as complex number multiplication have not been receiving enough attention in the literature. The main reason for that is, these types of operations can be performed using real-number multipliers. However, a higher performance could be achieved using hardware designed specially for these operations. Such design is particularly useful for applications employing algorithms based on complex operations. Special multipliers have been proposed in the literature for residue number system arithmetic [6,7], modulo multiplication [8], convolutions of small tables [9].

The direct implementation of a complex multiplier of two given points (X_0+jY_0) and (X_1+jY_1) makes use of four multiplications as follows:

$$(X_0 + jY_0)(X_1 + jY_1) = R + jI \quad (1)$$

where $R = X_0X_1 - Y_0Y_1$ and $I = X_0Y_1 + X_1Y_0$

Another arrangement is to make use of the real part, which is given by:

$$R = X_0X_1 - Y_0Y_1$$

$$I = (X_0 + Y_0)(X_1 + Y_1) - X_0X_1 - Y_0Y_1 \quad (2)$$

This arrangement has one less multiplication but three more additions. There is a similarity between complex multiplication and cyclic convolution of two points; each requires four multiplications and two additions/subtractions. One of the most recent approaches for cyclic convolution of two points is the so-called one over eight squared algorithm [9]. This algorithm mainly defines four variables *(a, b, c, and d)* as follows:

$$a = X_0 + Y_0 + X_1 + Y_1 \quad (3)$$

$$b = -X_0 + Y_0 - X_1 + Y_1 \quad (4)$$

$$c = -X_0 - Y_0 + X_1 + Y_1 \quad (5)$$

$$d = -X_0 + Y_0 + X_1 - Y_1 \quad (6)$$

Then squaring both sides of Eqs. (3-6) one derives:

$$X_0X_1 + Y_0Y_1 = \frac{a^2+b^2-c^2-d^2}{8} \quad (7)$$

$$X_0Y_1 + X_1Y_0 = \frac{a^2+d^2-b^2-c^2}{8} \quad (8)$$

The left sides of Eqs.(7) and (8) are the two terms of the cyclic convolution performed on (X_0, Y_0) and (X_1, Y_1). Thus, this algorithm is able to transfer the four required multiplications of size n into only four squaring operations but of size $(n+3)$. This transformation was done at the expense of ten extra addition operations. However, it is a common knowledge that both the timing and cost of addition are far less than for multiplications.

In this paper a new algorithm and two implementations for complex number multiplier are developed. The implementations utilize special squaring lookup table. The goal of the design is to reduce the size and the number of the lookup tables as well as the number of required additions.

2. THE PROPOSED ALGORITHM

In order to illustrate the proposed algorithm let us consider the two points (X_0+jY_0) and (X_1+jY_1) and define variables, *A, B, C, D, E,* and *F*, to be as follows:

$$A = X_0 + X_1 \quad (9)$$

$$B = Y_0 - Y_1 \quad (10)$$

$$C = X_0 + Y_1 \quad (11)$$

$$D = X_1 + Y_0 \quad (12)$$

$$E = X_0 - Y_1 \quad (13)$$

$$F = X_1 - Y_0 \quad (14)$$

Squaring both sides of Eqs. (9) through (14) the following equations can be obtained:

$$X_0X_1 - Y_0Y_1 = [2(A^2+B^2)-(C^2+D^2+E^2+F^2)]/4 \quad (15)$$

$$X_0Y_1 + X_1Y_0 = [2(C^2+D^2)-(C^2+D^2+E^2+F^2)]/4 \quad (16)$$

$$X_0Y_1 + X_1Y_0 = [C^2+D^2-(E^2+F^2)]/4 \quad (17)$$

The left sides of Eqs. (15) and either (16) or (17) are the real and the imaginary parts respectively of the complex multiplication performed on $(X_0 + jY_0)$ and $(X_1 + jY_1)$. The right sides of these same equations demonstrate that only squaring operations and additions are required for the computation of the complex multiplication. It is worth noting that multiplication by two is nothing more than left shift and dividing by four is equivalent to dropping the two least significant bits of a number. Furthermore, replacing the variable B in Eq. (10) by B' (B' = $Y_0 + Y_1$) then substituting B' for B in Eqs.(15) through (17) yields the two terms of the cyclic convolution performed on (X_0, Y_0) and (X_1, Y_1).

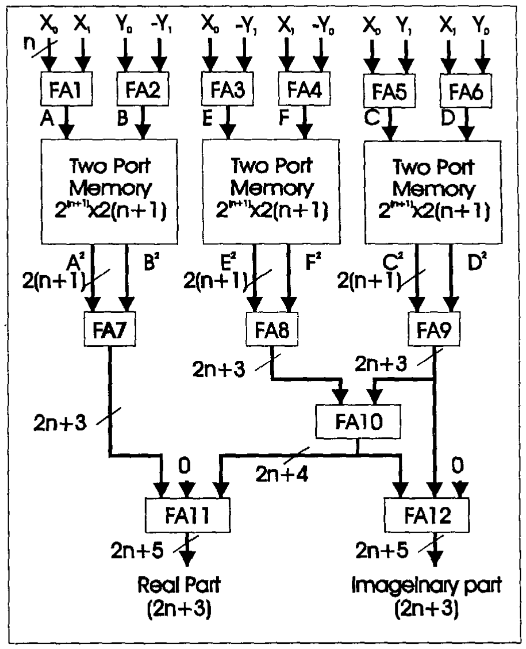
3. ALGORITHM IMPLEMENTATIONS

Both of the proposed implementations are based on the squaring operation. It is well recognized that the squaring operation for n-bit numbers is far less complex and consumes far less time than the multiplication of two n-bit numbers [10]. Moreover, squaring can be easily performed using look-up table [11].

3.1 Direct Implementation

The direct implementation of the Eqs. (15) and (16) is shown in Fig.(1). The inputs to this system are the two complex numbers to be multiplied, $(X_0 + jY_0)$ and $(X_1 + jY_1)$. All of the four inputs are n bits and they are applied to the input in parallel. The first stage of the system consists of six full adders named: FA1 through FA6. FA2, FA3, and FA4 work in the subtraction mode. The outputs of this stage, as shown in Fig. (1), are the terms *A, B, C, D, E,* and *F* described in Eqs. (9) through (14). The second stage consists of three two-port memories each contains $2^{(n+1)}$ locations. Each of these locations is 2(n+1) bits containing the lookup value for the squaring operation. In the proposed implementation, a two-port memory was preferred over a single port one in order to save in cost and space [12]. The lookup outputs are applied simultaneously to the three full adders FA7, FA8, and FA9. The outputs of the adding operations are the sum of the squared values as shown in Fig. (1). Each of these outputs is (2n+3) bits.

The term $(C^2+ D^2+ E^2+ F^2)$ is generated using a full adder, FA10, then added to the outputs of FA7 and FA9 in parallel using FA11 and FA12. The (2n+3) output lines of FA7 and FA9 are connected to the most significant 2n+3 inputs lines of each of FA11 and FA12 receptively. A zero is inserted to the least significant bit (LSB) of each of FA11 and FA12. These special connections are equivalent to multiplying each of the terms $C^2+ D^2$ and $A^2+ B^2$ by two before being added to the term $(C^2+ D^2+ E^2+ F^2)$ as required by Eqs. (15,16). The outputs of FA11 and FA12 are the real and imaginary parts of the multiplication scaled by four. Fortunately, division by four is equivalent to dropping the two LSB's.



X_0 X_1 Y_0 $-Y_1$ X_0 $-Y_1$ X_1 $-Y_0$ X_0 Y_1 X_1 Y_0
n
FA1
FA2
FA3
FA4
FA5
FA6
A
B
E
F
C
D
Two Port Memory 2^{n+1}x2(n+1)
Two Port Memory 2^{n+1}x2(n+1)
Two Port Memory 2^{n+1}x2(n+1)
A^2
B^2
E^2
F^2
C^2
D^2
2(n+1)
2(n+1)
2(n+1)
FA7
FA8
FA9
2n+3
2n+3
2n+3
FA10
0
2n+4
0
FA11
FA12
2n+5
2n+5
Real Part (2n+3)
Imageinary part (2n+3)

Fig. 1: Direct Implementation for Complex Multiplier of Two n-bit Complex Numbers.

3.2 Enhanced Implementation

This implementation processes information in a pipeline manner thus it is valuable for applications with long sequences of complex operations. The enhanced implementation of Eqs. 15 and 16 (or 17) is shown in Fig.(2). The inputs to this system are the two complex numbers to be multiplied, $(X_0 + jY_0)$ and $(X_1 + jY_1)$, and two external control signals C1 and C2, see Fig. (3). All of the four inputs (X_0, Y_0, X_1, and Y_1) are n bits and they are applied to the input in parallel. The first stage of the system consists of two multiplexers, MUX1 and MUX2, and three full adders named: FA1 through FA3. Each of these full adders works in either the subtraction or the addition mode depending on the value of the control signal C1. When C1 is low, these full adders are in the addition mode and when C1 is high, they are in the subtraction mode. The outputs of this stage, as shown in Fig. (2), are the terms C/E, D/F and A/B for the ADD/SUB mode. The second stage consists of two memories, one is two-port (M2) and the other is a single port (M1). Each of M1 and M2 contains $2^{(n+1)}$ locations of 2(n+1) bits each. Moreover, the contents of both M1 and M2 are identical; they are actually the lookup values for the squaring operation. The outputs of M2 are inputted to FA4, while the output of M1 is inputted to FA6 and loaded to R3 according to specific values of the control signals C1 and C2. The output of FA4 is loaded to either R1 (the value of $C^2 + D^2$) or R2 (the value of $E^2 + F^2$) according to specific values of the control signals C1 and C2. The contents of R3 (the value A^2) are added to the output of M1 (the value B^2) using FA6. Also, the contents of R1 and R2 are either added (to produce the value $C^2 + D^2 + E^2 + F^2$) or subtracted (to produce the value $C^2 + D^2 - (E^2 + F^2)$) using FA5 according the specific values of C1 and C2. The output of FA5 is either outputted as the imaginary part of the complex multiplication scaled by four (the output during the SUB mode) or loaded to R4 (the output during the ADD mode) according to the values of C1 and C2. The (2n+4) output lines of R4 and the (2n+3) of FA6 plus a zero at the LSB position are connected to the input lines of FA7. The zero is inserted to the LSB in order to multiply the term $A^2+ B^2$ by two before adding it to the term $(C^2+ D^2+ E^2+ F^2)$ as required by Eqs. (15). The output of FA7 is the real part of the complex multiplication multiplied by four. Fortunately, division by four is equivalent to dropping the two LSB's.

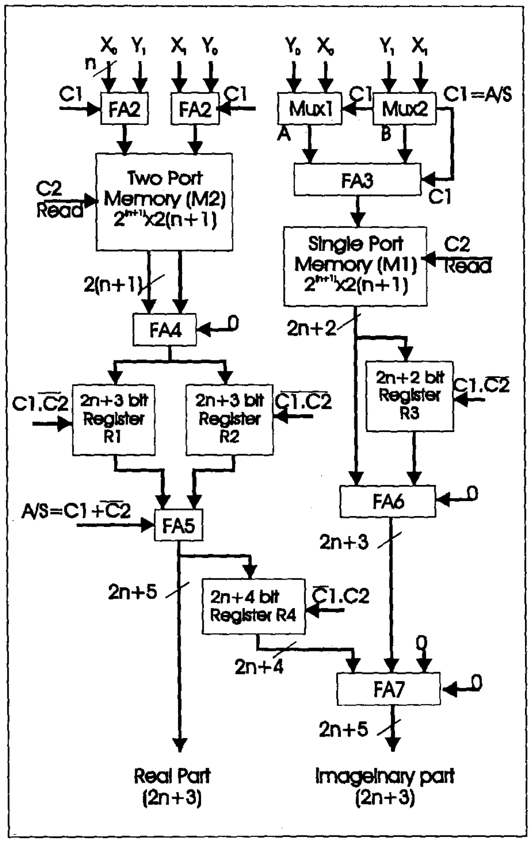


Fig. 2: Enhanced Implementation for Complex Multiplier of Two n-bit Complex Numbers.

The specific sequence of operations (refer to Figs. (2) and (3)) for this implementation can be summarized as follows:

1-The control signal C1 becomes low so that MUX1, and MUX2 select X_0 and X_1 respectively and FA1, FA2, and FA3 start operating in the add mode (i.e., ADD/SUB signal is low in order to compute the variables C, D, and A). Let us call the time to perform this addition $T_{A/S}$. The starting and the ending of this step are indicated by points a, and b respectively in the timing diagram of Fig. (3).

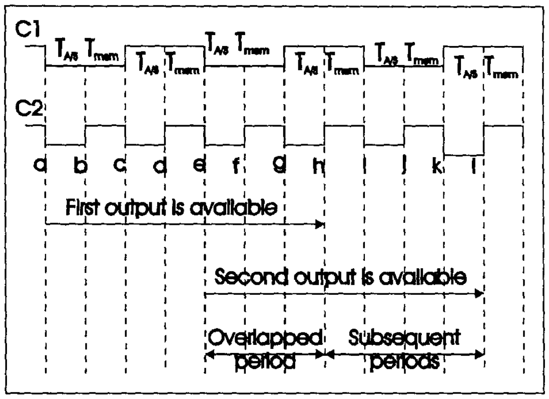


Fig. 3: Timing Diagram

2-The other control signal, C2, becomes high at point b (in Fig. (3)) to enable the two read signals of the two memories (M1 and M2) in order to obtain C^2, D^2, and A^2. The control signal C1 remains low during this time slot in order to have stable values out of FA1, FA2, and FA3. Denote the memory reading time by T_{MEM}. The time slot (b to c) in Fig. (3) marks the time required for this step.

3-The control signal C1 turns high at point c to switch FA1, FA2, and FA3 to the subtraction mode. At the same time the control signal C2 becomes low in order to disable the reading from M1 and M2 until the outputs of the FA1, FA2, and FA3 are stable. Also, during this time slot the two outputs of M2, (C^2 and D^2) are added and loaded to the (2n+3)-bit register, R1, in Fig. (2). In addition, the output of M1 (A^2) is loaded to the (2n+2)-bit register, R3. The time for this step is marked (c to d) (see Fig. (3)).

4-The control signal C2 turns high in order to read the squares of E, F, and B out of M2, and M1 respectively. In addition, the control signal C1 remains high during the period (d to e), see Fig. (3), in order to have stable inputs at M1 and M2.

5-Both of the control signals C1 and C2 turn low for a time equals to $T_{A/S}$ in order to be able to perform $(E^2 + F^2)$ and $(A^2 + B^2)$ and load R2 with $(E^2 + F^2)$. Also during this time slot, (e to f) in Fig. (3), FA1, FA2, and FA3 have added the next batch of inputs.

6- The control signal C2 turns high while the control signal C1 remains low for the time period (f to g) in Fig. (3). Thus, FA5 performs $(C^2 + D^2 + E^2 + F^2)$ and loads it to R4. Also, FA6 performs the addition $(A^2 + B^2)$. At the same time the values (C^2, D^2, and

A^2 for the next two numbers in the sequence) are read from M2 and M1 respectively.

7- The control signal C1 turns high while C2 becomes low for the time period (g to h). Thus, FA5 and FA7 compute the imaginary and the real parts of the two given n-bit complex operands. Simultaneously, the outputs of M2, and M1 are added and loaded into R1 and R3 respectively, of course these values are for the next batch of two complex numbers. Furthermore, the real and imaginary parts of the next two complex numbers are available at the output of FA5 and FA7 respectively after only four extra time slots, see Fig. (3).

Note that, the time required for the first operation in a sequence is equal to four times the delay of (2n+4)-bit full adder plus three times the delay of reading from $2^{(n+1)}$ x 2(n+1) memory. However, for any subsequent two numbers the time required is only two times the delay of (2n+4)-bit full adder plus two times the delay of reading from $2^{(n+1)}$ x 2(n+1) memory.

4. COMPARISONS WITH OTHER IMPLEMENTATIONS

Both of the two implementations require less time and hardware than the quarter squared algorithm [11] and the one over eight algorithm. The quarter square algorithm multiples two numbers using a reduced lookup table (i.e., for n-bit numbers it reduces the lookup table from 2^{2n} to 2^n). The hardware required for real number multiplication using this algorithm, is two memories, three full adders, and one negator for a single multiplication. Thus, complex multiplication using this algorithm requires a total of eight memories, fourteen additions, and four negations. On the other hand, one of the popular implementation for cyclic convolution [9], requires four n-bit full adders, four (n+1)-bit full adders, two (2n+4)-bit full adders, two (2n+5)-bit full adders, five negators, and four $2^{(n+3)}$ x 2(n+3) memory. It is worth noting that the size of the memory[9] is four times the size of the memory in either of the two proposed implementations. In addition, the number of negators is reduced to only three. Furthermore, the operating time using the proposed enhanced implementation is at least forty percent less than the time required using either the proposed simple implementation or the one presented for cyclic convolution in [9]. It is worth noting that, the complexity of cyclic convolution is the same as complex multiplication, since each of those requires four multiplications and two additions/substractions. Furthermore, the two proposed implementations have distinguishable capability unlike other implementations in the literature; both implementations could be used to perform both complex multiplication and cyclic convolution.

5. CONCLUSIONS

A new algorithm for computing complex multiplication has been developed. The presented algorithm requires only additions and lookup operations but no real multiplication. Thus, it saves both the hardware and time required for performing complex multiplication. Two implementations are presented for the proposed algorithm. The first is a straightforward implementation of the algorithm and is shown to be more efficient than the popular implementations in the literature [9, 11]. The enhanced implementation for complex multiplication has reduced the hardware by approximately fifty percent over the straightforward implementation. Moreover, the enhanced implementation saves over forty percent of the time for multiplication of sequences of complex numbers. Furthermore, the proposed algorithm and the two implementations could be used to perform cyclic convolution.

6. REFERENCES

[1] A. Bewrmak , D. Martinez, J. L. Noullet, "High-density 16/8/4-bit configurable multiplier", *IEE Proceedings-Circuits, Devise and Systems, Vol. 144, (No.5), IEE*, pp. 272-6, Oct. 1997.

[2] M. Santoro, "Design and clocking of VLSI m ultipliers", Ph.D. thesis, Stanford University, Oct. 1989.

[3] C.S. Wallace, "A suggestion for a fast m ultiplier", *IEEE Trans. On Electronic Computer, Vol .. EC-13*, pp.14-17, *1964*.

[4] Kyung-Wook Shin; Bang-Sup Song "A complex multiplier architecture based on redundant binary arithmetic", *Proceedings of 1997 IEEE International Symposium on Circuits and Systems. Circuits and Systems in the Information Age. ISCAS '97*, pp. 1944-1947, Vol.3.

[5] G. Goto, T. Sato, M. Nakajima, and T. Sukemura, "A 54X54-bit regularly structured tree multiplier", *IEE J. of Solid State Circuits*, 27(9), *pp. 1229-1236, Sep. 1992*.

[6] M. A. Soderstrand, and E. L. Fields, "Multipliers for residue number arithmetic digital filters", *Electron. Lett.*, 13, (6), pp. 164-166, 1977.

[7] M. A. Soderstrand, and C. Vernia, "A high-speed low-cost modulo pi multiplier with RNS arithmetic applications", *Proceedings of the IEEE, 68, (4)*, pp. 529-532, *1980*.

[8] Taylor, F.J., "Large moduli multipliers for signal processing", *IEEE Trans, CAS- 28, (7)*, pp. 731-736., *1981*.

[9] A. Skavatzos ,"Noval approach for implementing convolutions with small tables ", *IEE Proceedings-e* Vol. 138, No. 4, pp. 255-259, July 1991.

[10] B. W. Y. Wei; He Du; Honglu Chen (Edited by: Knowles, S., McAllister, W.H.) "A complex-number multiplier using radix-4 digits", *Proceedings of the 12th symposium on Computer Arithmetic*, 1995.

[11] E.L. Johnson, "A digital quarter square multiplier", *IEE Trans. On Comp.*, Vol. C-29 No. 3, pp. 258-261, Mar. 1980.

[12] H. Shininohara, N. Matsumoto, K. Fujimori, Y. Tsujihashi, H. Nakao, S. Kato, Y. Horiba, and A. Tada, "A flexible multiport RAM compiler for data path', *IEEE J. Solid-State Circuits. Vol. 26, pp. 343-349, March 1991*.