On-line Algorithms for Complex Number Arithmetic

Robert McIlhenny
rmcilhen@cs.ucla.edu

Miloš D. Ercegovac
milos@cs.ucla.edu

Computer Science Department
University of California
Los Angeles, CA 90024

Abstract

A class of on-line algorithms for complex number arithmetic is presented. These algorithms adopt a redundant complex number system (RCNS) to represent complex numbers as a single number. Such a scheme simplifies the specification of the design, and has the additional effect that single-precision complex arithmetic can be easily reconfigured for double-precision real arithmetic. We present cost×delay comparisons with the more conventional approach to show a significant improvement, demonstrating that the presented algorithms are attractive for VLSI systems demanding complex number operations.

1. Introduction

Algorithms for various on-line operations have been developed, analyzed, and implemented in actual arithmetic units over the past two decades. These include algorithms for various arithmetic operations, such as addition [5], multiplication [11], and division [11]. A design methodology is presented in [6], and a tutorial overview is given in [5].

These algorithms assume operations in real number systems. However, complex number computations also have imporantant roles in various signal processing and scientific applications such as complex orthogonal transformations, convolutions, correlations, and filtering [4]. Recent papers have demonstrated the emergence of on-line algorithms for complex number arithmetic. In [7], the computability of complex addition by an on-line finite state automaton is demonstrated. In [9], various on-line operators for complex arithmetic assuming radix 2 and a borrow-save encoding of the digits are presented. In this paper, algorithms for various complex number operations are presented, assuming a redundant complex number system (RCNS), which is presented next.

2. Redundant Complex Number System

A RCNS, as proposed in [2], is a radix-(rj) system, in which digits are in the set $\{-\rho, \cdots, 0, \cdots, \rho\}$, where $r \geq 2$ and $\lceil r^2/2 \rceil \leq \rho \leq r^2 - 1$. This allows a unified representation of the real and imaginary components.

Redundancy is essential in on-line operations, since in conventional number systems, the most significant digits of the result cannot be determined until all operand digits have been inputed and any carries have been allowed to propagate fully. Redundancy is achieved by using a signed-digit number representation [3]. The redundancy factor K in a RCNS is:

$$K = \frac{\rho}{r^2 - 1} \tag{1}$$

A RCNS with $r = 2, \rho = 2$, and hence $K = \frac{2}{3}$ is adopted, since, for multiplication, producing vector multiples $2x$ and $-2x$ is merely a matter of left shifting the vector x appropriately. This has an advantage over the digit set where $\rho = 3$, which has to consider multiples $3x$ and $-3x$, and has to incorporate an extra addition stage to add $\pm 2x$ and $\pm x$.

The actual representation of digits will be done using an extension of borrow-save encoding [10]. Using this encoding, the value of a digit x_k=$(x^+_{k,1}, x^-_{k,1}, x^+_{k,0}, x^-_{k,0})$ is: $2x^+_{k,1} - 2x^-_{k,1} + x^+_{k,0} - x^-_{k,0}$.

Conversion from a sign-magnitude representation consisting of a sign bit s and two binary digits: $(y_{k,1}, y_{k,0})$ to a radix-(2j) digit of weight $(2j)^{-k}$ using borrow-save encoding: $(x^+_{k,1}, x^-_{k,1}, x^+_{k,0}, x^-_{k,0})$ has to consider four cases of k:

Case 1: k mod 4=0 (Real, positive weight)

$$(x^+_{k,1}, x^-_{k,1}, x^+_{k,0}, x^-_{k,0}) = (\overline{s}y_{k,1}, sy_{k,1}, \overline{s}y_{k,0}, sy_{k,0})$$

Case 2: k mod 4=1 (Imaginary, negative weight)

$$(x^+_{k,1}, x^-_{k,1}, x^+_{k,0}, x^-_{k,0}) = (\overline{\overline{s}y_{k-1,0}}, \overline{sy_{k-1,0}}, \overline{\overline{s}y_{k,1}}, \overline{sy_{k,1}})$$

Case 3: k mod 4=2 (Real, negative weight)

$$(x^+_{k,1}, x^-_{k,1}, x^+_{k,0}, x^-_{k,0}) = (\overline{\overline{s}y_{k,1}}, \overline{sy_{k,1}}, \overline{\overline{s}y_{k,0}}, \overline{sy_{k,0}})$$

Case 4: k mod 4=3 (Imaginary, positive weight)

$$(x^+_{k,1}, x^-_{k,1}, x^+_{k,0}, x^-_{k,0}) = (\overline{s}y_{k-1,0}, sy_{k-1,0}, \overline{s}y_{k,1}, sy_{k,1})$$

Assuming fractional operands, the value of a number $X = (x_1, \ldots, x_m)$, is:

$$X = \sum_{k=1}^{m} x_k (2j)^{-k} \tag{2}$$

Alternatively, X can be considered as having radix (-4) real and imaginary components, X_r and X_i, respectively, where:

$$X_r = \sum_{k=1}^{\lfloor \frac{n}{2} \rfloor} x_{2k}(-4)^{-k} \tag{3}$$

$$X_i = (2j) \sum_{k=1}^{\lfloor \frac{n+1}{2} \rfloor} x_{2k-1}(-4)^{-k} \tag{4}$$

and $X = X_r + jX_i$.

3. Basic operations

The basic operations for complex on-line algorithms include (1) complex addition; (2) complex vector by digit multiplication; and (3) shifting by 2j, which will be described next.

3.1. Complex Addition

A radix (2j) on-line adder can be constructed as a modification of the radix-2 on-line adder presented in [8] consisting of PPM and MMP adder cells. The details of the adder cells are shown in Figure 1, and the radix-2 on-line adder is shown in 2. Modifying the design for radix (2j) consists of: (1) considering two bit slices, (2) complementing the carry-out, and (3) introducing extra delay registers due to the interleaving of real and imaginary digits. The complex on-line adder with an on-line delay $\delta = 2$, is shown in Figure 3.

3.2. Complex Vector by Digit Multiplication

Vector by digit multiplication for radix (2j) is defined such that, given input vector $X = (x_0 = 0, x_1, x_2, \ldots, x_m, x_{m+1} = 0, x_{m+2} = 0)$, where each digit $x_k = (x^+_{k,1}, x^-_{k,1}, x^+_{k,0}, x^-_{k,0})$ and digit y_i, then to compute $Z = X\,y_i$, where $Z = (z_0, z_1, \ldots, z_m)$, each output digit z_k has the following functionality:

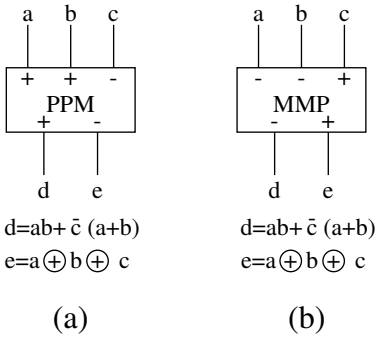

Figure 1. (a) PPM adder cell; (b) MMP adder cell

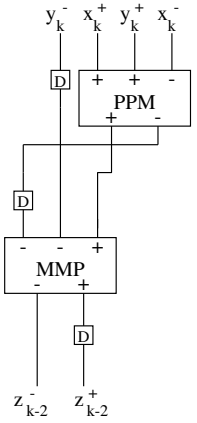

Figure 2. Radix-2 On-line Adder

$$z_k = \begin{cases} (\overline{x^+_{k,0}}, \overline{x^-_{k,0}}, x^+_{k+2,1}, x^-_{k+2,1}) & \text{if } y_k = \overline{2} \\ (x^+_{k,1}, x^-_{k,1}, x^+_{k,0}, x^-_{k,0}) & \text{if } y_k = \overline{1} \\ (0,0,0,0) & \text{if } y_k = 0 \\ (x^+_{k,1}, x^-_{k,1}, x^+_{k,0}, x^-_{k,0}) & \text{if } y_k = 1 \\ (x^+_{k,0}, x^-_{k,0}, \overline{x^+_{k+2,1}}, \overline{x^-_{k+2,1}}) & \text{if } y_k = 2 \end{cases} \tag{5}$$

3.3. Shifting by 2j

Shifting a vector (multiplying) by 2j for radix (2j) is defined such that to compute $Z = (2j)X$, for each digit of Z, $z_k = (z^+_{k,1}, z^-_{k,1}, z^+_{k,0}, z^-_{k,0})$:

$$\begin{array}{ll} z^+_{k,1} = x^+_{k+1,1} & z^-_{k,1} = x^-_{k+1,1} \\ z^+_{k,0} = x^+_{k+1,0} & z^-_{k,0} = x^-_{k+1,0} \end{array} \tag{6}$$

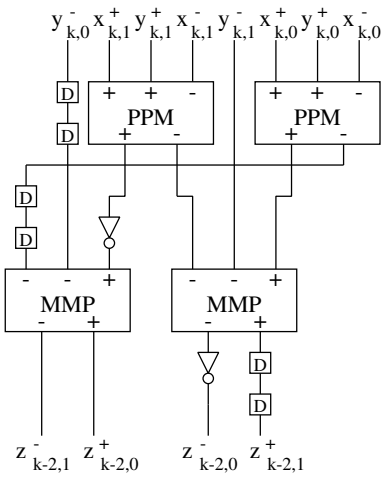

Figure 3. Complex On-line Adder

4. Complex On-line Algorithms

In this section, on-line algorithms for complex number arithmetic are presented, along with their on-line delay. The *on-line delay* of an operation is defined as the number δ such that the kth result digit is produced after $\delta + k$ digits of an input are available.

The algorithms consider inputs X and Y, and output Z, where

$$X = \sum_{i=1}^{m} x_i (2j)^{-i} \tag{7}$$

$$Y = \sum_{i=1}^{m} y_i (2j)^{-i} \tag{8}$$

$$Z = \sum_{i=1}^{m} z_i (2j)^{-i} \tag{9}$$

In on-line form,

$$X_k = \sum_{i=1}^{k+\delta} x_i (2j)^{-i} = X_{j-1} + x_{k+\delta}(2j)^{-k+\delta} \tag{10}$$

$$Y_k = \sum_{i=1}^{k+\delta} y_i (2j)^{-i} = Y_{j-1} + y_{k+\delta}(2j)^{-k+\delta} \tag{11}$$

$$Z_k = \sum_{i=1}^{k+\delta} z_i (2j)^{-i} = Z_{j-1} + z_{k+\delta}(2j)^{-k+\delta} \tag{12}$$

Operations that are considered include multiplication: $Z = X * Y$ and division $Z = X/Y$, which are presented next.

4.1. Complex On-line Multiplication

We want to compute $Z = XY$. At the kth step:

$$Z_k = X_{k-1}Y_{k-1} + (X_k y_{k+\delta} + Y_{k-1}x_{k+\delta})(2j)^{-k-\delta}$$

Letting $D_k = (X_k Y_k)(2j)^k$, define the kth residual as:

$$W_k = D_k - Z_{k-1}(2j)^k$$

Then the residual recurrence is:

$$W_k = fract((2j)W_{k-1} + (X_k y_{k+\delta} + Y_{k-1}x_{k+\delta})(2j)^{-\delta})$$

The algorithm for complex on-line multiplication is shown below:

Algorithm COLMUL

Step 1. [Initialization]
$W_0 = 0; Y_0 = 0;$
Step 2. [Recurrence]
for $k = 1, \ldots, m$ **do:**
$W_k = fract((2j)W_{k-1} + (X_k y_{k+\delta} + Y_{k-1}x_{k+\delta})(2j)^{-\delta})$
$z_k = int((2j)W_{k-1} + (X_k y_{k+\delta} + Y_{k-1}x_{k+\delta})(2j)^{-\delta})$

The implementation consists of appropriate registers to store X and Y and append digits x_k and y_k, vector-by-digit multipliers for computing $X_k y_k$ and $Y_{k-1}x_k$, and a borrow-save adder to add $X_k y_k$, $Y_k x_k$, and W_{k-1}. The design is shown in Figure 4. The cost and delay of each module for general m-digit precision is shown in Table 2.

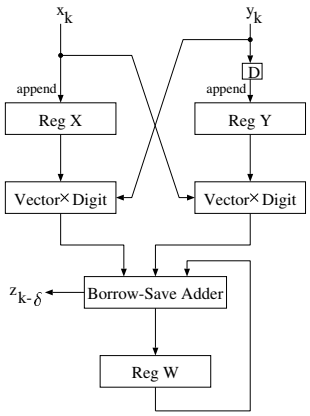

Figure 4. Complex On-line Multiplier

Module	Cost equiv. gates	Delay
Reg	$6m-5$	$t_{XOR}+2t_{DFF}$
Vector×Digit	$12m+1$	$t_{MUX(4\times1)}$
BS Adder	$24m+8$	$6t_{XOR}$

Table 1. Parameters of Modules

4.2. Complex On-line Division

We want to compute $Z = X/Y$. At the kth step:

$$X_k/Y_k = X_{k-1}/Y_{k-1} + x_{k+\delta}(2j)^{-k-\delta} - Z_{k-1}y_{k+\delta}(2j)^{-k-\delta}$$

Letting Q_k be the scaled partial quotient at step k:

$$Q_k = (X_k/Y_k)(2j)^k$$

Then:

$$Q_k = (X_{k-1}/Y_{k-1})(2j)^k + x_{k+\delta}(2j)^{-\delta} - Z_{k-1}y_k(2j)^{-\delta}$$

Letting z_k be the kth computed quotient digit, then the kth residual is:

$$W_k = Q_k - Y_{k-1}Z_{k-1}(2j)^k$$

The residual recurrence is:

$$W_k = (2j)(W_{k-1} - Y_{k-1}z_{k-1}) + (x_{k+\delta} - Z_{k-1}y_{k+\delta})(2j)^{-\delta}$$

The algorithm for complex on-line division is shown below:

Algorithm COLDIV

Step 1. [Initialization]

$z_0 = 0; Z_0 = 0; W_0 = X_0;$

Step 2. [Recurrence]

for $k = 1, \ldots, m$ **do:**

$W_k = (2j)(W_{k-1} - z_{k-1}Y_{k-1}) + (x_{k+\delta} - Z_{k-1}y_{k+\delta})(2j)^{-\delta}$

$z_k = SEL(\widehat{W_k})$

where the selection function $SEL(\widehat{W_k})$ is computed according to a lookup table that takes a 2-digit (8-bit) truncated value of W_k and returns the appropriate quotient digit $z_{k-\delta}$.

The implementation consists of appropriate registers to store W, Y, and Z and append digits x_k, y_k and z_k, vector-by-digit multipliers for computing $z_{k-1}Y_{k-1}$ and $Z_{k-1}y_k$, a borrow-save adder to add $z_{k-1}Y_k$, $Z_{k-1}y_k$, and W_{k-1}, and a lookup table to compute z_k. The design is shown in Figure 5.

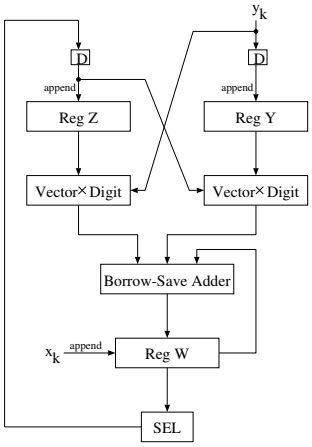

Figure 5. Complex On-line Divider

5. Evaluation

The proposed designs (Prop) were compared to the conventional (Conv) approach consisting of a network of real on-line operators. In the conventional approach, the real and imaginary components are computed according to the equations in Table 2, assuming $X = (a+bj)$, $Y = (c+dj)$, and $Z = (e+fj)$, where Z=Xop$Y$. The implementations were mapped onto Altera 10K-20 FPGA's [1] and measured for cost in terms of the number of logic cells, and critical delay, as shown in Table 3, assuming a total precision of 8 digits (4 for the real component, 4 for the imaginary component).

For multiplication, two conventional approaches were considered, including the standard four-multiplications and two additions (4M2A) approach, and the three-multiplications and five additions (3M5A) approach presented in [12]. For division, four real on-line multipliers, two real on-line dividers, two square units, and three real on-line adders were used.

Op	Result	
	e	f
Mul (4M2A)	$(ac-bd)$	$(ad+bc)$
(3M5A)	$(a-b)d+a(c-d)$	$(a-b)d+b(c+d)$
Div	$\frac{ac+bd}{a^2+b^2}$	$\frac{ad+bc}{a^2+b^2}$

Table 2. Conventional Complex Equations

Op		Cost	δ	Total Delay (ns)
Mul	Prop	418	2	190.8
	4M2A	810	4	210.0
	3M5A	656	6	279.9
Div	Prop	587	4	233.2
	Conv	1076	10	275.6

Table 3. Comparison of Approaches for precision $m = 8$

6. Conclusion

On-line algorithms were presented for complex number arithmetic using a redundant complex number system (RCNS) to represent complex numbers as a single number. Implementations were compared to a network of real on-line adders, to demonstrate a significant reduction in cost and in delay.

Such algorithms can be further developed for other operations such as complex square-root, complex square, and composite algorithms can be designed for actual applications involving complex numbers, such as the Fast Fourier Transform (FFT), recursive digit filters, and various other applications.

Acknowledgments. This research has been supported in part by the MICRO Grant 98037 "Reconfigurable Hardware for Numerically Intensive Computations", Raytheon Systems Company and Xilinx

References

[1] Altera-Corporation. *Altera Max+Plus II Manual*. San Jose, CA, 1996.

[2] T. Aoki, Y. Ohi, and T. Higuchi. Redundant complex number arithmetic for high-speed signal processing. *IEEE Workshop on VLSI Signal Processing*, pages 523–532, October 1995.

[3] A. Avizienis. Signed digit number representations for fast parallel arithmetic. *IRE Transactions on Electronic Computers*, EC-10:389–400, 1961.

[4] B. Barazesh, J. Michalina, and A. Picco. A VLSI signal processor with complex arithmetic capability. *IEEE Transactions on Circuits and Systems*, 35(5):495–505, May 1988.

[5] M. Ercegovac. On-line arithmetic: an overview. *Real Time Signal Processing VII, SPIE-495*, pages 86–93, 1984.

[6] M. Ercegovac and T. Lang. On-line arithmetic: a design methodology and applications. *1988 IEEE Workshop on VLSI Signal Processing*, pages 252–263.

[7] C. Frougny. Parallel and on-line addition in negative base and some complex number systems. *Euro-Par '96 Parallel Processing*, 2:175–182, 1996.

[8] A. Guyot, B. Hochet, and J.-M. Muller. JANUS, an on-line multiplier/divider for manipulating large numbers. *9th IEEE Symposium on Computer Arithmetic*, pages 106–111, September 1989.

[9] A. Nielsen. Number systems and digit serial arithmetic (Ph.D. dissertation). *Dept. of Mathematics and Computer Science, Odense University, Denmark*, 1996.

[10] A. Nielsen and J.-M. Muller. Borrow-save adders for real and complex number systems. *2nd conference on real numbers and computers*, April 1996.

[11] K. Trivedi and M. Ercegovac. On-line algorithms for division and multiplication. *IEEE Transactions on Computers*, C-26:681–687, July 1977.

[12] B. Wei, H. Du, and H. Chen. A complex-number multiplier using radix-4 digits. *Proceedings of the 12th Symposium on Computer Arithmetic*, pages 84–90, 1995.