A Hardware-Oriented Method for Evaluating Complex Polynomials

Miloš D. Ercegovac
Computer Science Department
University of California at Los Angeles
Los Angeles, CA 90095, USA
milos@cs.ucla.edu

Jean-Michel Muller
CNRS-Laboratoire LIP, projet Arénaire
Ecole Normale Supérieure de Lyon
69364 Lyon Cedex 07, FRANCE
Jean-Michel.Muller@ens-lyon.fr

Abstract

A hardware-oriented method for evaluating complex polynomials by solving iteratively a system of linear equations is proposed. Its implementation uses a digit-serial iterations on simple and highly regular hardware. The operations involved are defined over the reals. We describe a complex-to-real transform, a complex polynomial evaluation algorithm, the convergence conditions, and a corresponding design and implementation. The latency and the area are estimated for the radix-2 case. The main features of the method are: the latency of about m cycles for an m-bit precision; the cycle time independent of the precision; a design consisting of identical modules; and a digit-serial connections between the modules. The number of modules, each roughly corresponding to serial-parallel multiplier without a carry-propagate adder, is $2(n+1)$ for evaluating an n-th degree complex polynomial. The method can also be used to compute all successive integer powers of the complex argument with the same latency and a similar implementation cost.

1 Introduction

In this paper we describe a new method for evaluating complex polynomials, suitable for hardware implementation. It has a latency of m cycles for m-bit precision and a repetitive implementation which corresponds roughly to $2(n+1)$ serial-parallel multipliers for polynomials of degree n. The coefficients and argument are fixed-point complex numbers. The proposed method is a generalization of a polynomial evaluation method over the reals introduced as the E-method [2, 3], and recently overviewed in [5]. This paper is based on the report [6] where the complex E-method is introduced and discussed in general terms.

The method uses the following approach: (i) a polynomial is mapped onto a system of linear equations, (iii) a transform is applied to change the complex domain to the real domain, and (iii) the system is solved in a digit-by-digit manner, the most-significant digit first. The proposed approach is suitable for hardware implementation. The main characteristics of the method are: (i) m-digit solution is computed in about m steps, each step consisting of a sum of number-by-digit products, (ii) the cycle time does not depend on the precision m (if redundant additions are used), (iii) for a system of order n, the shortest latency requires n elementary units (modules) for the real part, and n units for the imaginary part, and (iv) the elementary units are interconnected with digit-wide links.

We first introduce the transform which allows the method to be used in the complex field $\mathbf{C}$. Then we show how to use the complex evaluation method (CE-method) in evaluating complex polynomials. Simultaneous evaluation of consecutive powers of a complex argument is a special case of polynomial evaluation and can be performed on the same hardware.

Complex polynomials appear in many areas such as digital signal and image processing, control systems, and applied mathematics, in general. A Horner type method for evaluating complex polynomials is proposed in [1] at the algorithm level, implicitly assuming a software implementation. The method uses $O(n)$ multiplications and $O(n)$ additions for a complex polynomial of degree n. If these multiplications and additions are performed in a sequential order, the latency of the method is about $n \times T_{MULT-ADD}$ which is significantly slower than our method. If a parallel algorithm for polynomial evaluation is used, the total time is about $\log n \times T_{MULT-ADD}$ which is still slower than our method. The evaluation of complex polynomials on equispaced arguments [7], error analysis [8], and a complexity analysis [9] are examples of research involving complex polynomials.

In the next section we describe the transformation which maps computation from the complex to the real domain. In Section 3 we show the CE-method for polynomials. In Section 4 iterations and convergence conditions are considered. Implementation aspects are discussed in Section 5.

1-4244-1027-4/07/$25.00 ©2007 IEEE

2 Complex-Real (CR) Transforms

Complex numbers can be represented by 2×2 skew-symmetric matrices

$$x + iy \leftrightarrow \begin{pmatrix} x & -y \\ y & x \end{pmatrix} \tag{1}$$

This isomorphism holds for complex addition and multiplication which are used in the proposed method :

$$\begin{aligned} (a+ib)+(c+id) &\leftrightarrow \begin{pmatrix} a & -b \\ b & a \end{pmatrix} + \begin{pmatrix} c & -d \\ d & c \end{pmatrix} \\ &= \begin{pmatrix} a+c & -b-d \\ b+d & a+c \end{pmatrix} \\ &\leftrightarrow (a+c)+i(b+d) \end{aligned} \tag{2}$$

$$\begin{aligned} (a+ib)\times(c+id) &\leftrightarrow \begin{pmatrix} a & -b \\ b & a \end{pmatrix} \times \begin{pmatrix} c & -d \\ d & c \end{pmatrix} \\ &= \begin{pmatrix} ac-bd & -(ad+bc) \\ ad+bc & ac-bd \end{pmatrix} \\ &\leftrightarrow (ac-bd)+i(bc+ad) \end{aligned} \tag{3}$$

Consequently, an $m \times n$ matrix of complex numbers can be represented as a $2m \times 2n$ matrix of real numbers. For $n \times n$ complex matrices, considered in this paper, the transform from the complex domain to the real domain is defined next.

Definition 1 *The* CR-transform *of the n-dimensional* complex *linear system*

$$\begin{pmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,n} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,n} \\ a_{3,1} & a_{3,2} & \cdots & a_{3,n} \\ \vdots & \vdots & \cdots & \vdots \\ a_{n,1} & a_{n,2} & \cdots & a_{n,n} \end{pmatrix} \times \begin{pmatrix} s_1 \\ s_2 \\ s_3 \\ \vdots \\ s_n \end{pmatrix} = \begin{pmatrix} t_1 \\ t_2 \\ t_3 \\ \vdots \\ t_n \end{pmatrix} \tag{4}$$

is the $2n$-dimensional real *linear system*

$$\begin{pmatrix} a^r_{1,1} & -a^i_{1,1} & a^r_{1,2} & -a^i_{1,2} & \cdots & a^r_{1,n} & -a^i_{1,n} \\ a^i_{1,1} & a^r_{1,1} & a^i_{1,2} & a^r_{1,2} & \cdots & a^i_{1,n} & a^r_{1,n} \\ a^r_{2,1} & -a^i_{2,1} & a^r_{2,2} & -a^i_{2,2} & \cdots & a^r_{2,n} & -a^i_{2,n} \\ a^i_{2,1} & a^r_{2,1} & a^i_{2,2} & a^r_{2,2} & \cdots & a^i_{2,n} & a^r_{2,n} \\ a^r_{3,1} & -a^i_{3,1} & a^r_{3,2} & -a^i_{3,2} & \cdots & a^r_{3,n} & -a^i_{3,n} \\ a^i_{3,1} & a^r_{3,1} & a^i_{3,2} & a^r_{3,2} & \cdots & a^i_{3,n} & a^r_{3,n} \\ \vdots & \vdots & \vdots & \vdots & \cdots & \vdots & \\ a^r_{n,1} & -a^i_{n,1} & a^r_{n,2} & -a^i_{n,2} & \cdots & a^r_{n,n} & -a^i_{n,n} \\ a^i_{n,1} & a^r_{n,1} & a^i_{n,2} & a^r_{n,2} & \cdots & a^i_{n,n} & a^r_{n,n} \end{pmatrix} \times \begin{pmatrix} s^r_1 \\ s^i_1 \\ s^r_2 \\ s^i_2 \\ s^r_3 \\ s^i_3 \\ \vdots \\ s^r_n \\ s^i_n \end{pmatrix}$$
$$= \left(t^r_1, t^i_1, t^r_2, t^i_2, t^r_3, t^i_3, \ldots, t^r_n, t^i_n\right)^T \tag{5}$$

where $a_{j,k} = a^r_{j,k} + ia^i_{j,k}$, $s_j = s^r_j + is^i_j$ and $t_j = t^r_j + it^i_j$. These two linear systems are *equivalent*.

In other words, the real linear system (5) is obtained from the complex linear system (4) by replacing each element $x+ix$ by the 2×2 matrix defined in (1). In the next section we consider a hardware-oriented method for solving such a system.

3 CE-method: An Overview

The E-method [2, 3] provides an iterative approach to solving diagonally dominant linear systems in real domain. The method has characteristics desirable for efficient hardware implementation: the basic operators are digit-vector multiplexers, and redundant adders of $[q : 2]$ type ($q \in \{3, 4, 6\}$), and registers. The overall structure consists of n elementary units, interconnected digit-serially. The method computes one digit of each element of the solution vector per iteration in the MSDF (Most Significant Digit First) manner which allows digit-serial communication. The modules operate concurrently. The time to obtain the solution to m digits of precision is about m cycles (iterations). The amount of hardware required is roughly related to the number of nonzero terms of the matrix of the system, which makes the proposed method very efficient in hardware resources when the matrix of the system is sparse. Typical applications of the method are evaluation of polynomial and rational functions, since these correspond to such sparse linear systems. The solution of the linear system

$$\begin{pmatrix} 1 & -x & 0 & 0 & 0 & \cdots & 0 \\ 0 & 1 & -x & 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \cdots & \vdots \\ 0 & 0 & \cdots & 0 & 0 & 1 & -x \\ 0 & 0 & 0 & \cdots & 0 & 0 & 1 \end{pmatrix} \times \begin{pmatrix} y_0 \\ y_1 \\ \vdots \\ y_{n-1} \\ y_n \end{pmatrix}$$
$$= (p_0, p_1, \ldots, p_{n-1}, p_n)^T$$

is

$$y = \begin{pmatrix} p_0 + p_1 x + p_2 x^2 + \cdots + p_n x^n \\ \vdots \end{pmatrix}$$

that is, the first element of the solution vector y is

$$y_0 = p_0 + p_1 x + p_2 x^2 + \cdots + p_n x^n = p(x)$$

Now consider the evaluation of polynomials with complex coefficients and argument:

$$p(z) = p_0 + p_1 z + p_2 z^2 + \ldots + p_n z^n$$

where the p_j's and z are complex numbers. As in the real case, the desired value $p(z)$ is clearly equal to the first component of the solution of the linear system

$$\begin{pmatrix} 1 & -z & 0 & 0 & 0 & \dots & 0 \\ 0 & 1 & -z & 0 & 0 & \dots & 0 \\ 0 & 0 & 1 & -z & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \dots & 0 & 1 \end{pmatrix} \times \begin{pmatrix} s_0 \\ s_1 \\ s_2 \\ \vdots \\ s_n \end{pmatrix}$$

$$= (p_0, p_1, p_2, p_3, \dots, p_n)^T \quad (6)$$

The method cannot directly solve the linear system (6). If we define real numbers x and y as $x + iy = z$, and p_j^r and p_j^i as $p_j = p_j^r + ip_j^i$, then we can apply the CR-transform to (6), and get the linear system with the coefficient matrix

$$A = \begin{pmatrix} 1 & 0 & -x & y & 0 & 0 & 0 & 0 & \cdots & 0 \\ 0 & 1 & -y & -x & 0 & 0 & 0 & 0 & \cdots & 0 \\ 0 & 0 & 1 & 0 & -x & y & 0 & 0 & \cdots & 0 \\ 0 & 0 & 0 & 1 & -y & -x & 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \cdots & 0 & 0 & 0 & 1 & 0 & -x & y \\ 0 & 0 & \cdots & 0 & 0 & 0 & 0 & 1 & -y & -x \\ 0 & 0 & \cdots & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & \cdots & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

The first two components of the solution s of the linear system

$$A \times \left(s_0^r, s_0^i, s_1^r, s_1^i, \dots, s_{n-1}^r, s_{n-1}^i, s_n^r, s_n^i\right)^T \quad (7)$$

$$= \left(p_0^r, p_0^i, p_1^r, p_1^i, \dots, p_{n-1}^r, p_{n-1}^i, p_n^r, p_n^i\right)^T$$

are equal to the real and imaginary parts of

$$p_0 + p_1 z + p_2 z^2 + \cdots + p_n z^n.$$

For instance, in the case $n = 3$, we get the solutions shown in Figure 1. The linear system (7) is easily solved by the proposed method, provided that it is diagonally dominant. The iterations and convergence conditions are discussed in the next section. Note that the method does not evaluate directly the expressions given for the solution s_0. These would require at least 16+16 full-precision multiplications which, assuming enough multipliers are available, would take at least 3 consecutive multiply times. Moreover, the reduction of product terms would require a [10:2] reduction. Of course, all the interconnections are of full precision. Instead, as explained later, the proposed method computes s_0 on 14 serial-parallel (left-to-right) multipliers, including the additions, in about *one serial-parallel multiplication time*. Moreover, the interconnections are digit-serial which simplifies routing and reduces power dissipation.

4 Iteration and convergence conditions

For simplicity, we discuss here radix-2 iterations. Adaptation to higher radices is rather straightforward. The radix-2 method consists in solving the n-dimensional linear system

$$As = p$$

by using the following basic recursion on residuals:

$$w^{(j)} = 2 \times \left[w^{(j-1)} - Ad^{(j-1)}\right] \quad (8)$$

with $w^{(0)} = [p_0, p_1, \dots, p_n]^T$, and $d^{(j)} = [d_0, d_1, \dots, d_n]^T$ where the digits $d_k^{(j)}$ are in $\{-1, 0, 1\}$. Define the number $D_k^{(j)} = d_k^{(0)}.d_k^{(1)}d_k^{(2)} \dots d_k^{(j)}$ (the $d_k^{(j)}$ are the digits of a radix-2 signed-digit representation of $D_k^{(j)}$). By induction, we easily get,

$$w^{(j)} = 2^j \left[w^{(0)} - AD^{(j-1)}\right]. \quad (9)$$

Using (9), one can show that if the residuals $|w_k^{(j)}|$ are bounded, then for all k, $D_k^{(j)}$ goes to s_k as j goes to infinity. The problem at step j is to find a *selection function* that produces a value of the digits $d_k^{(j)}$ from the residuals $w_k^{(j)}$ such that the values $w_k^{(j+1)}$ remain bounded. In [3], the following selection function as a form of rounding is proposed

$$SEL(x) = \begin{cases} \text{sign } x \times \lfloor |x + 1/2| \rfloor, & \text{if } |x| \le 1 \\ \text{sign } x \times \lfloor |x| \rfloor, & otherwise, \end{cases} \quad (10)$$

We apply the selection function to an estimate of the residual to avoid carry-propagate addition: $d_k^{(j)} = SEL(\hat{w}_k^{(j)})$, where $\hat{w}_k^{(j)}$ is a low-precision approximation to $w_k^{(j)}$.

Consider next in more detail evaluation of an n-degree complex polynomial.

$$p_n z^n + p_{n-1} z^{n-1} + \cdots + p_0$$

at the complex point $z = x + iy$, with $p_k = p_k^r + ip_k^i$. The matrix of the CR-transform is shown in Section 3.

We slightly modify the notations w and d of iteration (8) to adapt them to the complex case. The residual vector $w^{(j)}$ is

$$w^{(j)} = [w_{0,r}^{(j)}, w_{0,i}^{(j)}, w_{1,r}^{(j)}, w_{1,i}^{(j)}, \cdots, w_{n,r}^{(j)}, w_{n,i}^{(j)}],$$

$$s = \begin{pmatrix} -3\,xy^2p_3^r + x^3p_3^r - 3\,yx^2p_3^i + x^2p_2^r - 2\,xyp_2^i + xp_1^r + y^3p_3^i - y^2p_2^r - yp_1^i + p_0^r \\ -y^3p_3^r + 3\,yx^2p_3^r - 3\,y^2xp_3^i + 2\,yxp_2^r - y^2p_2^i + yp_1^r + x^3p_3^i + x^2p_2^i + xp_1^i + p_0^i \\ \vdots \end{pmatrix}$$

Figure 1. Solutions to system (7)

and its initial value are given by

$$w_{k,r}^{(0)} = p_k^r, \quad w_{k,i}^{(0)} = p_k^i$$

The digit-vector $d^{(j)}$ will be denoted

$$d^{(j)} = [d_{0,r}^{(j)}, d_{0,i}^{(j)}, d_{1,r}^{(j)}, d_{1,i}^{(j)}, \cdots, d_{n,r}^{(j)}, d_{n,i}^{(j)}].$$

Therefore, iteration (8) becomes

- for $k = 0, \ldots, n-1$,

$$\begin{aligned} w_{k,r}^{(j)} &= 2\left[w_{k,r}^{(j-1)} - d_{k,r}^{(j-1)} + xd_{k+1,r}^{(j-1)} - yd_{k+1,i}^{(j-1)}\right] \\ w_{k,i}^{(j)} &= 2\left[w_{k,i}^{(j-1)} - d_{k,i}^{(j-1)} + yd_{k+1,r}^{(j-1)} + xd_{k+1,i}^{(j-1)}\right] \end{aligned} \tag{11}$$

- for $k = n$,

$$\begin{cases} w_{n,r}^{(j)} &= 2\left[w_{n,r}^{(j-1)} - d_{n,r}^{(j-1)}\right] \\ w_{n,i}^{(j)} &= 2\left[w_{n,i}^{(j-1)} - d_{n,i}^{(j-1)}\right] \end{cases}$$

We now examine the convergence conditions. The iterations converge to the desired result if residual vector $w^{(j)}$ is bounded. Define constants ξ, α and Δ (with $0 \le \Delta < 1$) such that

1. $|x| + |y| \le \alpha$;
2. for any k between 0 and n, $|p_k^r| \le \xi$, $|p_k^i| \le \xi$, $|w_{k,r}^{(j)} - \hat{w}_{k,r}^{(j)}| \le \frac{\Delta}{2}$, and $|w_{k,i}^{(j)} - \hat{w}_{k,i}^{(j)}| \le \frac{\Delta}{2}$

Since $|d_{k,r}^{(j-1)} - \hat{w}_{k,r}^{(j-1)}| \le 1/2$ and $|d_{k,i}^{(j-1)} - \hat{w}_{k,i}^{(j-1)}| \le 1/2$, from (11) we find

$$|w_{k,r}^{(j)}| \le 2\left(\frac{1}{2} + \frac{\Delta}{2} + \alpha\right) = 1 + \Delta + 2\alpha. \tag{12}$$

The same bound holds for $|w_{k,i}^{(j)}|$. For this bound to be feasible, we must assure that a suitable choice of $d_{k,r}^{(j)}$ and $d_{k,r}^{(j)}$ in $\{-1, 0, 1\}$ is possible. This requires that $|w_{k,r}^{(j)}|$ and $|w_{k,i}^{(j)}|$ should be less than $3/2$. This immediately gives the following condition

$$\Delta + 2\alpha \le \frac{1}{2}. \tag{13}$$

Now, let us turn to the initial values. Since $|w_{k,r}^{(0)}|$ and $|w_{k,i}^{(0)}|$ must also be less than $3/2$, we get

$$\xi \le \frac{3}{2}. \tag{14}$$

Consider the following example: we wish to evaluate

$$p(z) = (1+i)\,z^3 - (0.5 + 1.25\,i)\,z^2 + z + 1.$$

at point

$$z = \frac{1}{100} + \frac{i}{10}.$$

We assume that $\Delta = 0$ (that is, we use non-redundant residuals). We get:

- Initialization:

$$\begin{aligned} w^{(0)} &= [p_0^r, p_0^i, p_1^r, p_1^i, p_2^r, p_3^i, p_4^r, p_4^i]^t \\ &= [1, 0, 1, 0, -0.5, -1.25, 1, 1]^t \end{aligned}$$

- Step 1: from $w^{(0)}$ and the selection function, we get

$$s^{(0)} = [1, 0, 1, 0, 0, -1, 1, 1]^t,$$

which gives

$$w^{(1)} = [0.02, 0.2, 0.2, -0.02, -1.18, -0.28, 0, 0]^t.$$

- Step 2: from $w^{(1)}$ and the selection function, we get

$$s^{(1)} = [0, 0, 0, 0, -1, 0, 0, 0]^t,$$

which gives

$$w^{(2)} = [0.04, 0.4, 0.38, -0.24, -0.36, -0.56, 0, 0]^t.$$

- After 20 iterations, the number

$$d_{0,r}^{(0)}.d_{0,r}^{(1)}d_{0,r}^{(2)}\cdots d_{0,r}^{(20)} + i \times d_{0,i}^{(0)}.d_{0,i}^{(1)}d_{0,i}^{(2)}\cdots d_{0,i}^{(20)}$$

is equal to

$$\frac{533789}{524288} + \frac{57727}{524288}\,i \approx 1.018121719 + 0.110105514\,i$$

whereas the exact value of $p(z)$ is

$$p(z) = 1.018121 + 0.110106\,i$$

Exactly as in the real case, even if polynomial p and point z do not satisfy the convergence constraints, one can easily "transform" them using mere shifts, so that $p(z)$ can be computed using the proposed method. Once Δ is chosen, and α is defined as $\frac{1}{4} - \Delta/2$, this is done as follows:

1. Find the smallest integer k such that $|\Re(z/2^k)| + |\Im(z/2^k)|$ should be less than α.
2. Now, $p(z) = \pi(t)$, where the degree-m coefficient of polynomial π is $2^{mk}p_m$. If at least one of the coefficients of π has the absolute value of its real or imaginary part greater than $\xi = 3/2$, then divide π by 2^ℓ, where ℓ is the smallest integer such that $\rho = \pi/2^\ell$ has the absolute value of the real and imaginary parts of its coefficients less than ξ.
3. What we actually compute using the method is $\rho(z/2^k)$. This result will then be multiplied by 2^ℓ (a simple left-shift) to get $p(z)$.

4.1 Implementation

In this section we discuss implementation of the proposed method. The main difference from implementation of a real domain evaluation E-method is that the number of non-zero off-diagonal elements doubles: for the polynomial case from one to two. This has two consequences. First, the bounds on the elements are smaller by a factor of two, and second, the cycle time is increased as explained later in this section. The corresponding implementations considered for the real domain method are discussed in [3].

A scheme for evaluation of complex polynomials is shown in Figure 2 for $n = 3$ and the corresponding elementary unit (PEU) is illustrated in Figure 3. A bit-parallel bus transmits x and y values in a broadcast mode, while the real and imaginary coefficients p^r and p^i are loaded in separate cycles. Note that the initialization cycles could be shorter than the iteration cycles.

A block diagram of an elementary unit PEU0r (real part only) for polynomial evaluation is shown in Figure 3. The modules are:

- Registers (4)
- Multiple generators MG (2), producing $\{-1, 0, 1\} \times x$ and $\{-1, 0, 1\} \times y$, with buffers
- Multiplexer MUX for initializing the residual
- A [4:2] adder
- Output digit selection SEL (a table or a gate network)

The digit-serial outputs of $PEU0$ can be converted into digit-parallel form using on-the-fly converters $OFCr$ and

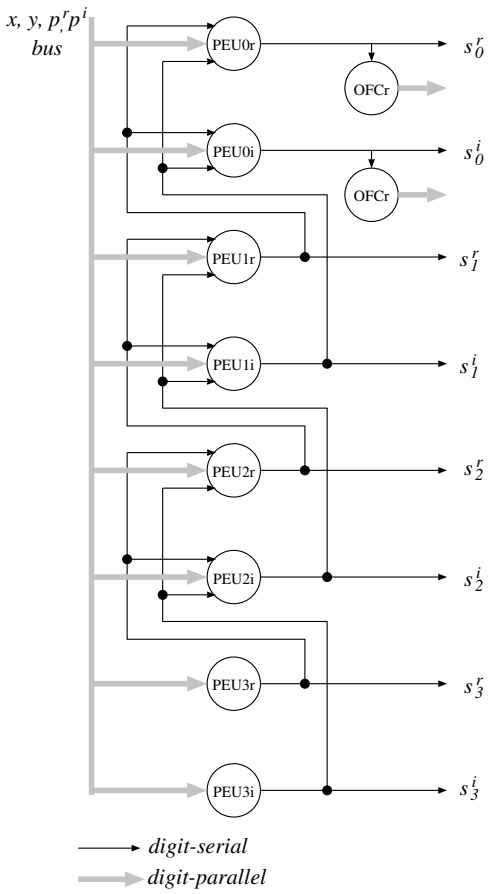

Figure 2. Scheme for evaluating complex polynomial ($n = 3$).

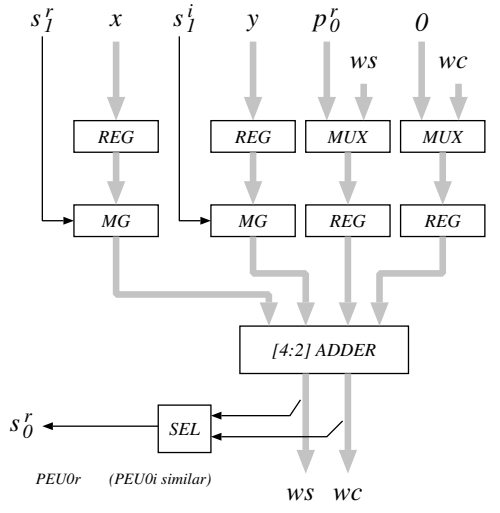

Figure 3. Block diagram of Elementary Unit.

$OFCi$.The cycle time, in terms of a full adder (complex gate) delay t, is estimated as

$$\begin{aligned} T_{PEU} &= t_{BUFF} + t_{MG} + t_{SEL} + t_{[4:2]} + t_{REG} \\ &\approx (0.4 + 0.3 + 1 + 1.3 + 0.9)t = 3.9t \quad (15) \end{aligned}$$

The cost, again in terms of area of a full adder A_{FA}, is estimated as

$$\begin{aligned} A_{PEU}(m) &= A_{SEL} + 2A_{BUFF} + 2A_{MG} \\ &+ A_{MUX} + A_{[4:2]} + 4A_{REG} + 2A_{OFC} \\ &\approx [5 + 2 \times 0.4 + (m+2)(3 \times 0.45 \\ &+ 2.3 + 4 \times 0.6 + 2 \times 2.1)]A_{FA} \\ &\approx 16 + 10mA_{FA} \quad (16) \end{aligned}$$

The cost is estimated as area occupied by modules using the area of a full-adder A_{FA} as the unit. The areas of primitive modules are: Register $A_{REG} = 0.6A_{FA}$; buffer $A_{BUFF} = 0.4A_{FA}$; MUX $A_{MUX} = 0.45A_{FA}$; multiple generator MG $A_{MG} = 0.45A_{FA}$; [4:2] adder $A_{[4:2]} = 2.3A_{FA}$; SEL $A_{SEL} = 5A_{FA}$, and on-the-fly converters $A_{OFC} = 2A_{MUX} + 2A_{REG} = 2.1A_{FA}$.

4.2 Possible applications

In [1] the author recalls that in control system theory, the frequency response of a linear continuous time invariant system is obtained by substituting $s = i\omega$, in the system transfer function $H(s) = q(s)/p(s)$, where q and p are one-variable polynomials. Evaluating such polynomials would be easily done using our method. Moreover, we could easily take into account the fact that s has no real part: Eq. 11 would be simplified, since variable x of that equation would disappear.

In a paper [4] a digit-recurrence algorithm for performing complex divisions is presented. The algorithm requires a prescaling step to get an approximation to the reciprocal of the divisor. In that paper, this is done using a table that can be very large. Instead of that, we propose to approximate the reciprocal of $1 + z$ (with, say, $|z| \leq 1/2$, with reduction to this rather large domain easily done) by the truncated series

$1 - z + z^2 - z^3 + \cdots + (-1)^k z^k$

Using k terms, we have an error bounded by 2^{-k}. Moreover, a brief examination of [4] and this paper shows that much hardware can be common to both methods.

4.3 Summary

We presented a method for evaluating complex polynomials by solving diagonally-dominant linear systems in complex domain by a digit-recurrence algorithm. This is a generalization of the real-domain method. The latency is roughly m cycles for m bits of precision and independent of the order of the system. This does not take into account potentially needed scaling steps. The cycle time is independent of m. We discussed the transform from real to complex numbers, the iteration and convergence conditions. Implementation is given at a high level with estimates of the cost and latency.

With the exception of complex addition and multiplication, complex operations are typically not implemented in hardware. Recently, hardware-oriented methods for complex division and square root have been introduced [4]. The method proposed in this report extends complex arithmetic to complex polynomials and complex powers. In a similar manner, the real method can be adapted to evaluation of complex rational functions.

References

[1] K. Benmahammed, Evaluation of Complex Polynomials in One and Two Variables. Multidimensional Systems and Signal Processing, 5, 245-261, 1994.

[2] M.D. Ercegovac. *A general method for evaluation of functions and computation in a digital computer*. PhD thesis, Dept. of Computer Science, University of Illinois, Urbana-Champaign, 1975.

[3] M.D. Ercegovac. A General Hardware-oriented Method for Evaluation of Functions and Computations in a Digital Computer. *IEEE Trans. Comp.*, C-26(7):667–680, 1977.

[4] M.D. Ercegovac and J.-M. Muller. Complex Division with Prescaling of Operands. *IEEE International Conference on Application-Specific Systems, Architectures and Processors*, pp. 293-303, 2003.

[5] M.D. Ercegovac and T. Lang. *Digital Arithmetic*, Morgan Kaufmann Publishers - an Imprint of Elsewier Science, San Francisco, 2004.

[6] M.D. Ercegovac and J.-M. Muller, Solving Systems of Linear Equations in Complex Domain : Complex E-Method. LIP Report No. 2007-2, École Normale Supérieure de Lyon, France.

[7] A.H. Nutall, Efficient Evaluation of Polynomials and Exponentials of Polynomials for Equispaced Arguments, IEEE Trans. Acoust., Speech, Signal Process., vol. ASSP-35, pp. 1486-1487, 1987.

[8] F. W. J. Olver, Error Bounds for Polynomial Evaluation and Complex Arithmetic, IMA Journal of Numerical Analysis 6, 373-379, 1986.

[9] J.H. Reif, Approximate Complex Polynomial Evaluation in Near Constant Work Per Point, STOC 97, pp. 30-39, 1997.