Journal of VLSI Signal Processing 2007
© 2007 Springer Science + Business Media, LLC Manufactured in the United States.
DOI: 10.1007/s11265-006-0029-2

Complex Square Root with Operand Prescaling

MILOŠ D. ERCEGOVAC
Computer Science Department, University of California at Los Angeles, 4732 Boelter Hall, Los Angeles, CA 90095, USA

JEAN-MICHEL MULLER
CNRS-Laboratoire CNRS-ENSL-INRIA-UCBL LIP Ecole Normale Supérieure de Lyon, 46 Allée d'Italie, 69364 Lyon Cedex 07, France

Received: 1 March 2005; Revised: 25 January 2006; Accepted: 23 October 2006

Abstract. We propose a radix-*r* digit-recurrence algorithm for complex square-root. The operand is prescaled to allow the selection of square-root digits by rounding of the residual. This leads to a simple hardware implementation of digit selection. Moreover, the use of digit recurrence approach allows correct rounding of the result if needed. The algorithm, compatible with the complex division presented in Ercegovac and Muller ("Complex Division with Prescaling of the Operands," in *Proc. Application-Specific Systems, Architectures, and Processors (ASAP'03)*, The Hague, The Netherlands, June 24–26, 2003), and its design are described. We also give rough estimates of its latency and cost with respect to implementation based on standard floating-point instructions as used in software routines for complex square root.

Keywords: computer arithmetic, complex square-root, digit-recurrence algorithm, operand prescaling

1. Introduction

1.1. Complex Square-root

Complex square-root appears in numerical computations such as complex Givens rotation [3], complex singular value decomposition [1, 13, 25], and in applications such as principal component analysis [7], quantum defect theory [19] and wave propagation [23].

Complex square-root operation is commonly implemented in software based on various algorithms developed for reliable and accurate evaluation in languages like FORTRAN 90 [14]. There are also collections of Fortran routines for multiple-precision complex arithmetic which include complex square-root [24]. These implementations rely on standard floating-point instructions and, consequently, their execution time is significantly longer than that of a single arithmetic instruction. In today's processors it is quite common to have hardware implementation of all basic operations on real fixed/floating-point operands. To our knowledge, there are no implementations of complex square-root at the hardware level on conventional processors. The only hardware implementation of complex arithmetic, including square-root, we are aware of is an FPGA implementation due to McIlhenny [16], who used an adaptation of on-line arithmetic to Eq. (1). With a rapid increase in capacity of integrated circuits, it is timely to consider hardware-based implementation of an extended set of operations. In this research we focus on hardware-oriented algorithms and implementations of operations on operands in the complex domain. In [11] we proposed and developed an algorithm and its

This is an expanded version of a paper presented at the ASAP'2004 conference.

implementation for complex division compatible with a standard radix-r digit-recurrence division scheme. In this paper we extend our approach to complex square-root operation.

Since a nonzero complex number has two square roots, we will define $\sqrt{z}$ as the square root whose real part is positive when z is not a negative real number. The simplest algorithm for evaluating a complex square root $u + iv$ of $x + iy$, based on real square roots, consists in successively computing

$$\begin{aligned} \ell &= \sqrt{x^2 + y^2} \\ u &= \sqrt{(\ell + x)/2} \\ v &= \pm\sqrt{(\ell - x)/2} \end{aligned} \tag{1}$$

with sign(v) = sign(y) [2]. This algorithm is optimal in the *algebraic* sense, i.e., in the number of exact operations $+, -, \times, \div, \sqrt{\cdot}$. However, it suffers from several drawbacks:

- $x^2 + y^2$ can overflow or underflow, even if the exact square root is representable, leading to very poor results;
- 3 real square roots, 2 squares and 3 additions/subtractions are required. Even if this is "algebraically optimal," this is quite costly and one could hope that a direct hardware implementation of complex square root could be better.

Another solution [21] is to first compute

$$w = \begin{cases} 0 & \text{if} \quad x = y = 0 \\ \sqrt{|x|}\sqrt{\dfrac{1 + \sqrt{1 + (y/x)^2}}{2}} & \text{if} \quad |x| \geq |y| \\ \sqrt{|y|}\sqrt{\dfrac{|x/y| + \sqrt{1 + (x/y)^2}}{2}} & \text{if} \quad |x| < |y|. \end{cases}$$

and then obtain

$$\begin{aligned} u + iv &= \sqrt{x + iy} \\ &= \begin{cases} 0 & \text{if} \quad w = 0 \\ w + i\frac{y}{2w} & \text{if} \quad w \neq 0 \text{ and } x \geq 0 \\ \frac{|y|}{2w} + iw & \text{if} \quad w \neq 0 \text{ and } x < 0 \text{ and } y \geq 0 \\ \frac{|y|}{2w} - iw & \text{if} \quad w \neq 0 \text{ and } x < 0 \text{ and } y < 0. \end{cases} \end{aligned}$$

This avoids intermediate overflows at the cost of more computation including several tests, divisions and real square roots which make the complex square root evaluation quite slow compared to a single arithmetic instruction. Also, estimating the final accuracy is very difficult.

Kahan [15] gives a better solution, that also correctly handles all special cases (infinities, zeros, NaNs, etc.), also at the cost of significantly more computation than the naive method (1).

This paper presents an algorithm similar to the usual digit-recurrence real square-root algorithm [9, 10, 20], suitable for hardware implementation. The original approach was presented in [12]. For computing $\sqrt{x}$, this algorithm uses the recurrence

$$w[j+1] = rw[j] - 2s_{j+1}S[j] - s_{j+1}^2 r^{-j-1} \tag{2}$$

where $w[0] = x$, and the square root digits s_j's are chosen in a radix-r redundant digit-set, so that the residual $w[j]$ is bounded.

The main problem of digit-recurrence algorithms is to find a practical result-digit selection function for higher radices. Several approaches have been suggested for higher radix square-root digit selection. In [4] the use of digit selection tables is analyzed and applied to the radix-4 case. The hardware complexity of this approach grows rapidly with the radix. An alternative, applied to higher radix digit-recurrence algorithms for division [9, 10], uses prescaling of the operands and rounding of the truncated residual to achieve a feasible digit-selection function for higher radices. This approach using prescaling and rounding has been also developed for higher radix square rooting [17, 18]. It consists of multiplying x by a constant K so that Kx is close to 1, and using the standard residual recurrence to compute $\sqrt{Kx}$. The prescaling allows the selection of s_{j+1} by rounding the residual $w[j]$ (or, merely, an approximation made up with a few most significant digits of $w[j]$) to the nearest integer. For fast implementation, a truncated residual is used. K is deduced from a few most significant bits of x. To simplify the multiplication $K \times x$, it is desirable to choose a low-precision value of K.

Throughout the paper, i is $\sqrt{-1}$, and if z is a complex number, then $\Re(z)$ and $\Im(z)$ denote the real and imaginary parts of z. The norm $||z||_\infty$ denotes $\max\{|\Re(z)|, |\Im(z)|\}$, whereas $|z|$ denotes the usual complex absolute value

$$\sqrt{(\Re(z))^2 + (\Im(z))^2}.$$

Since $|\Re(z)|$ and $|\Im(z)|$ are both less than or equal to $||z||_\infty$, we deduce: $|z| \leq \sqrt{||z||_\infty^2 + ||z||_\infty^2} = \sqrt{2}||z||_\infty$.

In the next section we describe the basic recurrence and prescaling of the argument. In Section 3 we discuss implementation of the method and give some rough measures of its latency and cost. We also compare with a conventional implementation of complex square root operation.

2. Complex Square-root Algorithm

2.1. Basic Iteration

Assume we wish to compute $\sqrt{x}$, where x is a complex number satisfying $||x||_\infty < 2$. We consider a digit-recurrence algorithm that produces a radix-r representation of $\sqrt{x}$ in the form $s_0.s_1s_2s_3s_4\dots$ with $s_j = s_j^{\mathcal{R}} + is_j^{\mathcal{I}}$, where $s_j^{\mathcal{R}}$ and $s_j^{\mathcal{I}}$ are in the redundant digit set $\mathcal{S} = \{-a, -a+1, \dots, a\}$, $a \leq r-1$, and r is the radix (in practice, r is a power of 2).

Assume that we have already computed $S[j]$ represented by $s_0.s_1s_2s_3s_4\dots s_j$. The jth residual is defined as

$$W[j] = r^j\left(x - (S[j])^2\right) \qquad (3)$$

Using Eq. (3), we obtain the residual recurrence

$$W[j+1] = rW[j] - 2s_{j+1}S[j] - s_{j+1}^2 r^{-j-1} \qquad (4)$$

which is the same recurrence as in the real case. It is important to note that, from Eq. (3), any choice of the s_j's for which the $W[j]$'s remain bounded will ensure that $S[j]^2 \rightarrow x$. After separating the real and imaginary parts of $W[j+1]$ in Eq. (4), we get

$$\begin{cases} W^{\mathcal{R}}[j+1] &= rW^{\mathcal{R}}[j] - 2s_{j+1}^{\mathcal{R}}S^{\mathcal{R}}[j] + 2s_{j+1}^{\mathcal{I}}S^{\mathcal{I}}[j] \\ & \quad -\left((s_{j+1}^{\mathcal{R}})^2 - (s_{j+1}^{\mathcal{I}})^2\right)r^{-j-1} \\ W^{\mathcal{I}}[j+1] &= rW^{\mathcal{I}}[j] - 2s_{j+1}^{\mathcal{I}}S^{\mathcal{R}}[j] - 2s_{j+1}^{\mathcal{R}}S^{\mathcal{I}}[j] \\ & \quad -2s_{j+1}^{\mathcal{R}}s_{j+1}^{\mathcal{I}}r^{-j-1}. \end{cases} \qquad (5)$$

Selection of digits $s_{j+1}^{\mathcal{R}}$ and $s_{j+1}^{\mathcal{I}}$ so that $W^{\mathcal{R}}[j+1]$ and $W^{\mathcal{I}}[j+1]$ remain bounded is not obvious and we now discuss our approach in obtaining a selection function. Indeed, Fig. 1 shows that, at least in some cases, choosing the "complex digits" s_{j+1} cannot be done at a reasonable cost.

However, we notice that:

- The terms $\left((s_{j+1}^{\mathcal{R}})^2 - (s_{j+1}^{\mathcal{I}})^2\right)r^{-j-1}$ and $2s_{j+1}^{\mathcal{R}}s_{j+1}^{\mathcal{I}}r^{-j-1}$ decrease rapidly as j increases, so that their influence can be neglected when choosing s_{j+1} after, possibly, a few first iterations.
- If $S[j]$ is close to 1 (i.e., if $S^{\mathcal{R}}[j]$ is close to one and $S^{\mathcal{I}}[j]$ close to zero), then $W^{\mathcal{R}}[j+1]$ would be close to $rW^{\mathcal{R}}[j] - 2s_{j+1}^{\mathcal{R}}$ and $W^{\mathcal{I}}[j+1]$ would be close to $rW^{\mathcal{I}}[j] - 2s_{j+1}^{\mathcal{I}}$, so that a "natural" choice for $s_{j+1}^{\mathcal{R}}$ would be the integer closest to $\frac{1}{2}rW^{\mathcal{R}}[j]$, and a "natural" choice for $s_{j+1}^{\mathcal{I}}$ would be the integer closest to $\frac{1}{2}rW^{\mathcal{I}}[j]$.

To make $S[j]$ close to 1, we perform prescaling of the operand. This allows the iterations (4) to start at step $j_0 > 1$ and to use for the selection the "natural" choices presented above. For a fast implementation of the iteration, we will use low-precision estimates of the shifted real and imaginary residuals. Of course, we have to make sure that this suffices to ensure convergence.

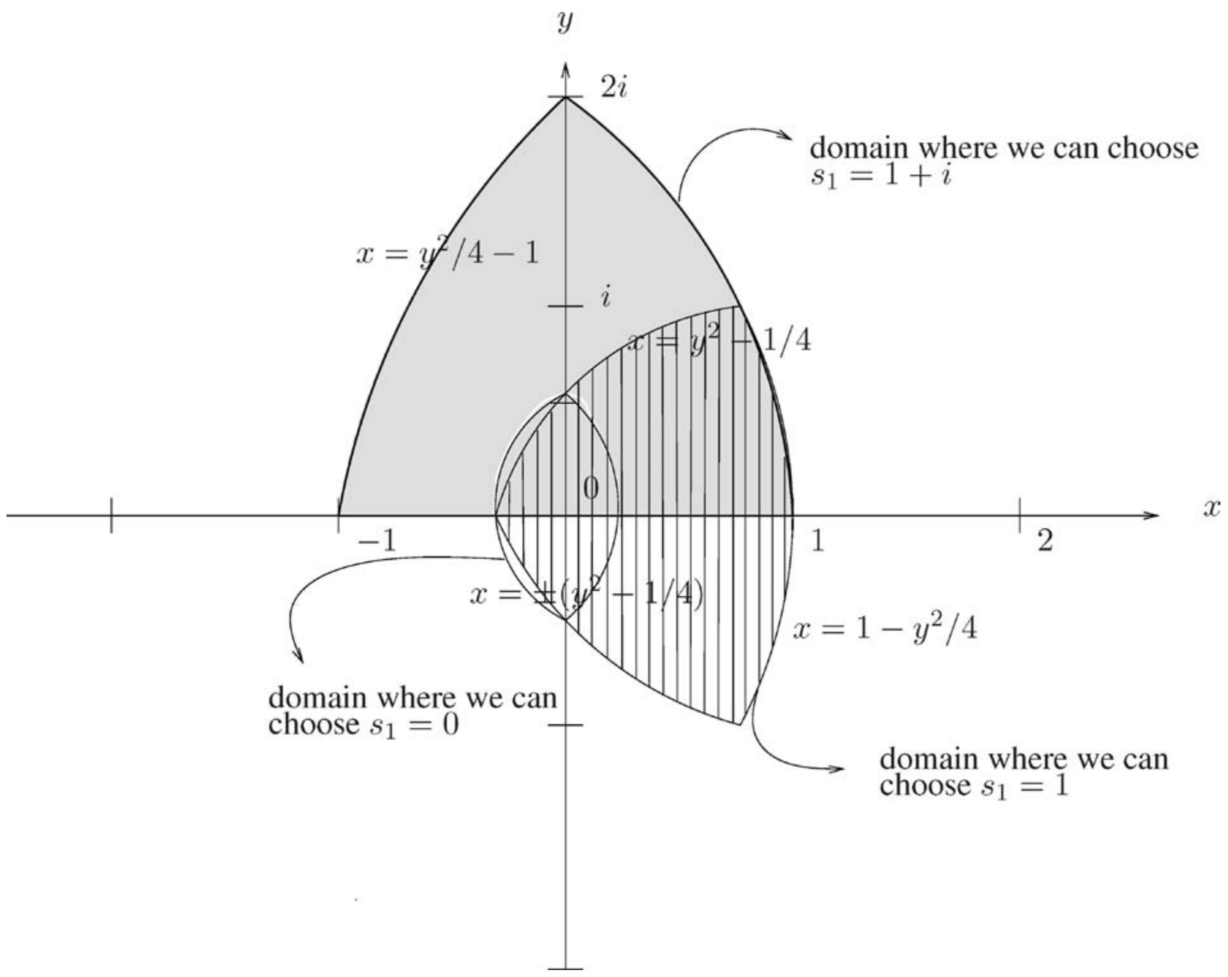
2.2. Prescaling the Operand

2.2.1. How Prescaling the Operand Simplifies the Selection. The prescaling part of the algorithm is very similar to that of complex division [11]. Before discussing different ways of performing the prescaling, let us show how having prescaled the input operand simplifies our problem.

Using an input number x (for simplicity, we assume $1/2 \leq ||x||_\infty < 1$), we first obtain (from a look-up table, either by direct look-up or using a method presented in Section 2.4) a complex number K such that $||Kx - 1||_\infty < 2^{-q}$, where q is a parameter of the square-root algorithm. We then obtain $d = Kx$ and use the digit-recurrence algorithm with selection by rounding to compute $\sqrt{d}$. The table stores also precomputed values $1/\sqrt{K}$, so that at the end of the computation we can obtain the final result $\sqrt{x}$ as

$$\sqrt{x} = \sqrt{d} \times \left(1/\sqrt{K}\right).$$

The multiplication by $1/\sqrt{K}$ can be performed in parallel with the recurrence: as we compute a new complex digit s_j of $\sqrt{d}$, we accumulate $s_j \times 1/\sqrt{K}$ in



y
$2i$
domain where we can choose $s_1 = 1 + i$
$x = y^2/4 - 1$
i
$x = y^2 - 1/4$
0
x
-1
1
2
$x = \pm(y^2 - 1/4)$
$x = 1 - y^2/4$
domain where we can choose $s_1 = 0$
domain where we can choose $s_1 = 1$

Figure 1. The domains where digit $s_1^{\mathcal{R}}$ can be chosen equal to 1, $1 + i$ or 0, assuming $s_0 = 0$. The complexity of these shapes shows that prescaling is needed to simplify the digit selection.

two registers, one for the real and another for the imaginary part.

After the prescaling step, we have obtained $d = Kx$ such that $||\delta||_\infty < 2^{-q}$, where $\delta = d - 1$. Let us try to bound $\left|\left|\sqrt{d} - 1\right|\right|_\infty$. We use the Taylor expansion

$$\sqrt{d} = 1 + \frac{\delta}{2} - \frac{\delta^2}{8} + \frac{\delta^3}{16} - \frac{5\delta^4}{128} + \cdots \qquad (6)$$

Define R and θ by $\delta = Re^{i\theta}$. From $\delta^k = R^k e^{ik\theta}$, one gets $||\delta^k||_\infty \leq R^k$, and since $R \leq \sqrt{2}||\delta||_\infty$, we finally obtain $||\delta^k||_\infty \leq 2^{k/2}||\delta||_\infty^k \leq 2^{k/2}2^{-qk}$. Combining the last result with the power series (6), we get

$$\begin{aligned} ||\sqrt{d} - 1||_\infty &\leq \frac{2^{-q}}{2} + \frac{2 \times 2^{-2q}}{8} + \frac{2\sqrt{2} \times 2^{-3q}}{16} \\ &\quad + \frac{5 \times 4 \times 2^{-4q}}{128} + \cdots \\ &\leq \frac{2^{-q}}{2} + \frac{2 \times 2^{-2q}}{8} + 2^{-3q}. \end{aligned}$$

In particular, for $q > 2$:

$$\left|\left|\sqrt{d} - 1\right|\right|_\infty < \frac{9}{16} 2^{-q}. \qquad (7)$$

Hence, there exists a radix-2 representation of $\sqrt{d}$ that starts with

- For the real part:

$$1.\underbrace{000\cdots 00}_{q\text{ zeros}}$$

- For the imaginary part:

$$0.\underbrace{000\cdots 00}_{q\text{ zeros}}$$

These digits can be used to initialize $S[j_0]$ for some j_0. More precisely, assume that the digits s_j are radix-$r = 2^k$ digits in the set $\{-a, \ldots, +a\}$. From Eq. (7), one can easily show that there exists a representation of $\sqrt{d}$ in this radix-r system that starts with

- For the real part:

$$1.\ \underbrace{000\cdots 00}_{j_0=\lfloor q/k \rfloor\text{ zeros}}$$

- For the imaginary part:

$$0.\ \underbrace{000\cdots 00}_{j_0=\lfloor q/k \rfloor\text{ zeros}}$$

if $a\sum_{i=j_0+1}^{\infty} r^{-i} \geq \frac{9}{16}2^{-q}$, i.e.,

$$\frac{ar^{-j_0}}{r-1} \geq \frac{9}{16}2^{-q}. \tag{8}$$

Assume $q = kj_0 + \ell$, with $0 \leq \ell \leq k-1$. Eq. (8) gives

$$\frac{a}{r-1} \geq \frac{9}{16}2^{-\ell} \tag{9}$$

The ratio $a/(r-1) = \rho$ is the redundancy factor which, for a redundant digit set $\{-a, \ldots, +a\}$, satisfies $1/2 < \rho \leq 1$. If $\ell \geq 1$ the condition (9) can be satisfied for any r and a. For $\ell = 0$, the condition requires that $a > r/2$, i.e., the minimally redundant system cannot be used.

2.3. *Making the Iterations Work*

We assume that the radix of the iteration is $r = 2^k$. We start the iterations from step $j_0 = \lfloor q/k \rfloor$ since Eq. (8) is satisfied in all practical cases. The initial values are

$$\begin{cases} S^{\mathcal{R}}[j_0] &= 1 \\ S^{\mathcal{I}}[j_0] &= 0 \\ W^{\mathcal{R}}[j_0] &= r^{j_0}(d^{\mathcal{R}} - 1) \\ W^{\mathcal{I}}[j_0] &= r^{j_0}(d^{\mathcal{I}}). \end{cases} \tag{10}$$

The selection function that returns the s_j's is as follows. We will choose $s^{\mathcal{R}}_{j+1}$ as the integer closest to the number constituted by $1/2rW^{\mathcal{R}}[j]$ truncated after the σ-th borrow-save position.[1] We choose $s^{\mathcal{I}}_{j+1}$ as the integer closest to the value $1/2rW^{\mathcal{I}}[j]$ truncated after the σ-th borrow-save position. This implies that

$$\left|\left|s_{j+1} - \frac{1}{2}rW[j]\right|\right|_{\infty} \leq \frac{1}{2} + 2^{-\sigma}. \tag{11}$$

For convergence of the algorithm, we have to :

1. Bound $||W[j+1]||_{\infty}$, that is, bound $|W^{\mathcal{R}}[j+1]|$ and $|W^{\mathcal{I}}[j+1]|$;
2. Choose q and σ so that the selection function returns digits in the set $\{-a, \ldots, +a\}$.

Let us first bound $||W[j+1]||_{\infty}$. Denote $S[j] = 1 + \epsilon = 1 + \epsilon^{\mathcal{R}} + i\epsilon^{\mathcal{I}}$. From Eq. (10), we get $||\epsilon||_{\infty} = \max\{|\epsilon^{\mathcal{R}}|, |\epsilon^{\mathcal{I}}|\} < r^{-\lfloor q/k \rfloor}$. Since

$$W^{\mathcal{R}}[j+1] = \left(rW^{\mathcal{R}}[j] - 2s^{\mathcal{R}}_{j+1}\right) - 2s^{\mathcal{R}}_{j+1}\epsilon^{\mathcal{R}} + 2s^{\mathcal{I}}_{j+1}\epsilon^{\mathcal{I}} \\ - \left((s^{\mathcal{R}}_{j+1})^2 - (s^{\mathcal{I}}_{j+1})^2\right)r^{-j-1}$$

we find

$$\begin{aligned} |W^{\mathcal{R}}[j+1]| &< 1 + 2^{-\sigma+1} + 4ar^{-\lfloor q/k \rfloor} + a^2r^{-j-1} \\ &< 1 + 2^{-\sigma+1} + 4ar^{-\lfloor q/k \rfloor} + a^2r^{-\lfloor q/k \rfloor-1} \end{aligned}$$

Similarly, from $W^{\mathcal{I}}[j+1] = (rW^{\mathcal{I}}[j] - 2s^{\mathcal{I}}_{j+1}) - 2s^{\mathcal{I}}_{j+1}\epsilon^{\mathcal{R}} - 2s^{\mathcal{R}}_{j+1}\epsilon^{\mathcal{I}} - 2s^{\mathcal{R}}_{j+1}s^{\mathcal{I}}_{j+1}r^{-j-1}$ we find $|W^{\mathcal{I}}[j+1]| < 1 + 2^{-\sigma+1} + 4ar^{-\lfloor q/k \rfloor} + 2a^2r^{-\lfloor q/k \rfloor+1}2^{-q}$.

Therefore, we get the following bound

$$||W[j+1]||_{\infty} < 1 + 2^{-\sigma+1} \\ + \left(4a + 2\frac{a^2}{r}\right)r^{-\lfloor q/k \rfloor} = \Omega \tag{12}$$

Let us now determine conditions to assure that the real and imaginary parts of computed digits s_{j+1}

belong to $\{-a, \ldots, +a\}$. These digits are obtained by rounding to the nearest integer an estimate with accuracy $\pm 2^{-\sigma}$ of a number whose absolute value can be as large as $\frac{1}{2} r\Omega$. To satisfy $|s_{j+1}| \leq a$ we must have

$$\frac{1}{2} r\Omega + 2^{-\sigma} \leq a + \frac{1}{2} \tag{13}$$

Combining Eqs. (12) and (13), we get:

Property 1 *If*

$$r\left(\frac{1}{2} + 2^{-\sigma} + \left(2a + \frac{a^2}{r}\right) r^{-\lfloor q/k \rfloor}\right) + 2^{-\sigma} \leq a + \frac{1}{2} \tag{14}$$

then the recurrence (5), initialized at step $j_0 = \lfloor q/k \rfloor$ with the values defined in Eq. (10) returns a representation of $\sqrt{d}$ in radix r with digits in $\{-a, \ldots, +a\}$ which can be selected by rounding the residual estimate of σ fractional bits.

2.4. Two Ways of Prescaling the Argument

Our method requires that, from a given argument x, we obtain a complex scaling factor K such that $||Kx - 1||_\infty < 2^{-q}$ by a table-lookup. We must also get $1/\sqrt{K}$. Let us examine two approaches to obtaining K.

2.4.1. First Approach: Direct Table-lookup.

We assume that

$$\frac{1}{2} \leq ||x||_\infty < 1.$$

This is obtained by a mere shift of the argument x. We can also assume that $\Re(x)$ and $\Im(x)$ are both nonnegative, and that $\Re(x) > \Im(x)$. This comes from the elementary properties:

$$\begin{cases} \frac{1}{a+ib} & = & i \times \frac{1}{-b+ia} \\ \frac{1}{\overline{a+ib}} & = & \overline{(1/(a+ib))}. \end{cases}$$

and

$$\begin{cases} K' = iK & \Rightarrow & i \times \frac{1}{\sqrt{K'}} = \frac{\lambda}{\sqrt{K}} \\ \sqrt{\overline{a+ib}} & = & \overline{\sqrt{a+ib}}. \end{cases}$$

where $\overline{a+ib} = a - ib$ is the complex conjugate and[2] $\lambda = \frac{\sqrt{2}}{2}(1-i)$.

Now, if we write $x = a + ib$, a and b can be represented as binary fixed-point numbers

$$\begin{cases} a & = & 0.a_1 a_2 a_3 a_4 \cdots \\ b & = & 0.b_1 b_2 b_3 b_4 \cdots, . \end{cases}$$

where $a_1 = 1$. Define $\hat{a}$ and $\hat{b}$ as a and b rounded to the nearest m-fractional-bit number. Our first solution consists in looking-up

$$K = \frac{1}{\hat{a} + i\hat{b}}$$

in a table with $2m - 1$ address bits.[3] Now, by denoting $\hat{x} = \hat{a} + i\hat{b}$, we easily find

$$\left|\left|\frac{x}{\hat{x}} - 1\right|\right|_\infty \leq 2||x - \hat{x}||_\infty \left|\left|\frac{1}{\hat{x}}\right|\right|_\infty \leq 4||x - \hat{x}||_\infty \leq 2^{-m+1}$$

Therefore, to assure that $||Kx - 1||_\infty$ will be less than 2^{-q}, it suffices to choose $m = q + 1$. Hence the lookup table will have $2q + 1$ address bits.

2.4.2. Second Approach: Two-step Table-lookup.

Another solution is to first use the previous method with a much smaller value of q, so that from a and b we get a_1, b_1, K_1 and $1/\sqrt{K_1}$ that satisfy

$$\begin{array}{ccc} |1 - a_1| & < & 2^{-q_1} \\ |b_1| & < & 2^{-q_1} \\ a_1 & = & K_1 a \\ b_1 & = & K_1 b \end{array}$$

To do that, we need a table with $2q_1 + 1$ address bits. Then, we use the same method again (with a different table), to find from a_1 and b_1, values a_2, b_2, K_2 and $1/\sqrt{K_2}$ that satisfy

$$\begin{array}{ccc} |1 - a_2| & < & 2^{-q} \\ |b_2| & < & 2^{-q} \\ a_2 & = & K_2 a_1 \\ b_2 & = & K_2 b_1 \end{array}$$

Since the first q_1 fractional bits of a_1 are zeros or ones, and since the first q_1 fractional bits of b_1 are zeros, the second table lookup requires a tables with

Table 1. Parameters q and σ and number of address bits as function of r and a.

Case	r	a	q	σ	P1 (# address bits)	P2 (# address bits)
1.	2	1	4	4	9	7
2.	2	1	6	3	13	9
3.	4	2	6	5	13	9
4.	4	3	6	3	13	9
5.	8	4	9	5	19	12
6.	8	5	9	3	19	12
7.	8	6	6	5	13	9
8.	8	7	6	4	13	9
9.	16	8	12	6	25	15
10.	16	9	8	9	17	11
11.	16	10	8	5	17	11
12.	16	11	8	4	17	11
13.	16	12	8	3	17	11
14.	16	15	8	2	17	11

$2m$ address bits, where $m = (q+1) - (q_1 - 1)$ (the "-1" in "$q_1 - 1$" comes from the fact that we need to know the sign of b_1 and whether the first bits of a_1 are all zeros or ones). After that, we compute $K = K_1K_2$ and $1/\sqrt{K} = 1/\sqrt{K_1} \times 1/\sqrt{K_2}$.

Hence we need to lookup a table with $2q_1 + 1$ address bits and a table with $2((q+1) - (q_1 - 1))$ address bits. The best solution is obtained by equating the sizes of both tables, which gives $q_1 = \frac{2q+3}{4}$, i.e.,

$$q_1 = \lceil q/2 + 3/4 \rceil$$

Therefore, the tables have $q + 3$ address bits each.

2.5. *Relations among Algorithm Parameters*

Table 1 gives parameters q and σ that satisfy Eq. (14), depending on r and a. The table required by the prescaling step has $2q + 1$ address bits if we perform a direct table look-up. With the two-step prescaling method, the tables use $q + 3$ address bits. Table 1 also indicates the number of address bits for the direct (P1) and the two-step prescaling (P2) approaches. Note that Case 4 (radix 4 max. redundancy) and Case 8 (radix 8 max. redundancy) have the same table size requirements. Since the prescaling for the complex square root is similar to the one for the complex division algorithm [11], a combined scheme is possible, which makes hardware implementation more attractive.

3. Implementation and Comparison

We are considering only the computation of the significand, ignoring exponent handling and related adjustments to the argument. The argument is $x = (x^{\mathcal{R}}, x^{\mathcal{I}})$ and the result is $z = \sqrt{x} = (z^{\mathcal{R}}, z^{\mathcal{I}})$ with the real and imaginary components in a fixed-point format. The overall scheme for computing complex square root is outlined in Fig. 2.

There are four main parts in the scheme: prescaling, recurrence evaluation, postscaling, and on-the-fly conversion with rounding. The prescaling part uses a table lookup to determine the scaling factor $K = (K^{\mathcal{R}}, K^{\mathcal{I}})$ and the postscaling factor $1/\sqrt{K} = (C^{\mathcal{R}}, C^{\mathcal{I}})$, and a complex multiplier to obtain the scaled argument $d = x \times K$. The postscaling factor is needed to obtain z from the computed result as $\sqrt{d}$ as $z = \sqrt{d} \times (C^{\mathcal{R}}, C^{\mathcal{I}})$.

The real and imaginary residual recurrences can be implemented using a modified conventional square-root recurrence. As discussed in [10], it is convenient to define the residual recurrence as

$$w[j+1] = rw[j] + F[j] \tag{15}$$

where

$$F[j] = -2S[j]s_{j+1} - s_{j+1}^2 r^{-(j+1)} \tag{16}$$

Since $s[j]$ digits are produced in signed-digit form, the partial result $S[j]$ is also in signed-digit form. Depending on the adder used, $S[j]$ is converted to adapt to the adder. If a carry-save adder is used, F has to be in two's complement form. This conversion can be done on-the-fly. In this paper we assume that the adder is of a carry-save type. We now apply the form of Eq. (15) to the complex residual recurrence 5:

$$\left\{\begin{array}{rcl} F^{\mathcal{R}}[j] & = & -2s_{j+1}^{\mathcal{R}}S^{\mathcal{R}}[j] - (s_{j+1}^{\mathcal{R}})^2 r^{-j-1} \\ G^{\mathcal{R}}[j] & = & 2s_{j+1}^{\mathcal{I}}S^{\mathcal{I}}[j] + (s_{j+1}^{\mathcal{I}})^2 r^{-j-1} \\ W^{\mathcal{R}}[j+1] & = & rW^{\mathcal{R}}[j] + F^{\mathcal{R}}[j] + G^{\mathcal{R}}[j] \\ F^{\mathcal{I}}[j] & = & -2s_{j+1}^{\mathcal{R}}S^{\mathcal{R}}[j+1] \\ G^{\mathcal{I}}[j] & = & -2s_{j+1}^{\mathcal{I}}S^{\mathcal{I}}[j] \\ W^{\mathcal{I}}[j+1] & = & rW^{\mathcal{I}}[j] + F^{\mathcal{I}}[j] + G^{\mathcal{I}}[j] \ . \end{array}\right. \tag{17}$$

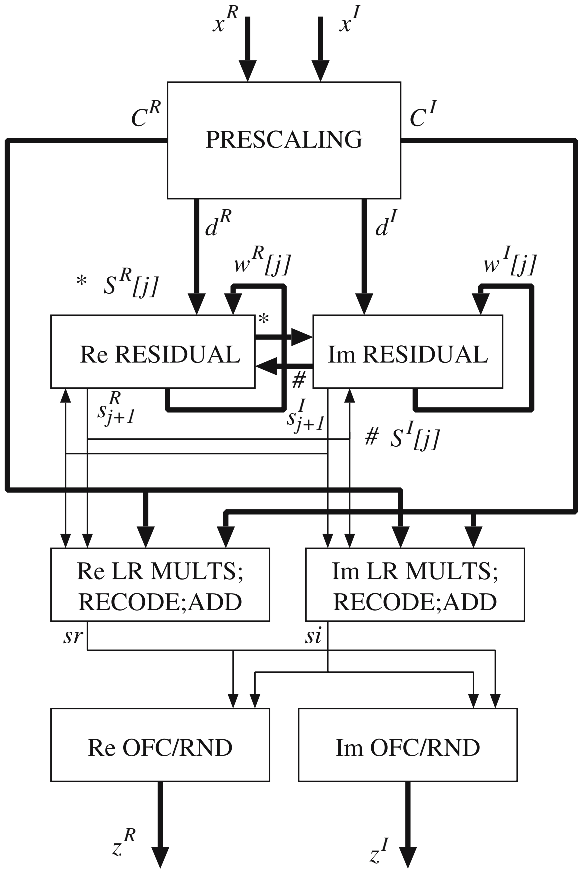


Figure 2. Overall scheme for computing complex square root.

The real $s^{\mathcal{R}}_{j+1}$ and imaginary $s^{\mathcal{I}}_{j+1}$ digits are selected by rounding of the corresponding shifted residual estimates $r\widehat{W^{\mathcal{R}}}[j]$ and $r\widehat{W^{\mathcal{R}}}[j]$, respectively:

$$\begin{aligned} s^{\mathcal{R}}_{j+1} &= SEL(r\widehat{W^{\mathcal{R}}[j]}) \\ s^{\mathcal{I}}_{j+1} &= SEL(r\widehat{W^{\mathcal{I}}[j]}) \end{aligned} \tag{18}$$

where *SEL* function consists of rounding the residual estimate and recoding of the result digit into radix-4 signed digits.

The block-diagram of implementation of the complex recurrence is shown in Fig. 3.

The postscaling to obtain $z = \sqrt{d} \times (C^{\mathcal{R}}, C^{\mathcal{I}})$ is performed using four sequential left-to-right carry-free (LRCF) multipliers without final adders [8]. Each LRCF multiplier produces one product digit per step. Since these digits are in an over-redundant digit set, they are recoded and added to produce the real and imaginary digits in signed-digit form. These are then used sequentially by on-the-fly converters to produce the final real and imaginary parts of the result in a conventional form. The implementation combines postscaling with on-the-fly conversion. Moreover, as discussed in [10], the rounding can be integrated with the on-the-fly conversion. These multipliers do not have final adders since the product digits are used left-to-right in redundant form.

We now discuss the postscaling and conversion in more detail. In step j the following increments are added to the accumulated real and imaginary partial products:

$$\begin{gathered} s^{\mathcal{R}}_{j+1}C^{\mathcal{R}} - s^{\mathcal{I}}_{j+1}C^{\mathcal{I}} \\ s^{\mathcal{R}}_{j+1}C^{\mathcal{I}} - s^{\mathcal{I}}_{j+1}C^{\mathcal{R}} \end{gathered} \tag{19}$$

These are implemented using four LRCF multipliers and the corresponding recurrences are

$$\left\{ \begin{aligned} U^{\mathcal{R}}[j+1] &= r(frac(U^{\mathcal{R}}[j] + C^{\mathcal{R}}s^{\mathcal{R}}_{j+1})) = r(frac(UR)) \\ V^{\mathcal{R}}[j+1] &= r(frac(V^{\mathcal{R}}[j] - C^{\mathcal{I}}s^{\mathcal{I}}_{j+1})) = r(frac(VR)) \\ pr1_{j+1} &= int(UR) \\ pr2_{j+1} &= int(VR) \\ U^{\mathcal{I}}[j+1] &= r(frac(U^{\mathcal{I}}[j] + C^{\mathcal{I}}s^{\mathcal{R}}_{j+1})) = r(frac(UI)) \\ V^{\mathcal{I}}[j+1] &= r(frac(V^{\mathcal{I}}[j] - C^{\mathcal{R}}s^{\mathcal{I}}_{j+1})) = r(frac(VI)) \\ pi1_{j+1} &= int(UI) \\ pi2_{j+1} &= int(VI) \end{aligned} \right. \tag{20}$$

where $frac(g)$ and $int(g)$ are the fractional and integer parts of g, respectively. Since

$$|UR| \leq r + (r-1) \times a \tag{21}$$

the range of $pr1_{j+1}$ exceeds one radix r digit. Similarly for the other output digits. Dividing the postscaling factors by r, we obtain the output digits in the set $\{-(r+a-1), \ldots, (r+a-1)\}$ which are recoded to the set $\{-(r-1), \ldots, r-1\}$ to simplify the remaining modules. After recoding the digits are added using on-line addition to produce the real (imaginary) radix-r signed-digits of the result. These digits are then converted using on-the-fly conversion to obtain the final result in conventional radix-r representation. The design details are omitted.

The proposed scheme has an estimated latency

$$T_{proposed} = t_{prescal} + t_{iter} + t_{postscale} + t_{recode} + t_{OL-add} + t_{convert-rnd} \tag{22}$$

For $r \leq 16$ we estimate the delays of the terms in the expression for T_{prop} as follows. $t_{prescal}$ is the time to perform the table lookup to get K and C and perform $d = x \cdot K$ which we estimate to be $\leq 2t_{cycle}$. The iteration time is $t_{iter} = (n - j_0)t_{cycle}$. The postscaling and conversion/rounding are overlapped with the residual recurrence: $t_{recode} = 2t_{cycle}$ because of the scaling of C as discussed above; the remaining stages have single cycle delay. We estimate that the cycle time t_{cycle} of the recurrence loop (see Fig. 3), measured in full-adder delays (t_{FA}), is

$$t_{cycle} = t_{SEL} + t_{F,G} + t_{[6:2]} + t_{reg} = 6t_{FA} \tag{23}$$

where $t_{SEL} = 1.5t_{FA}$ is the delay of the selection function, $t_{F,G} = 1.5t_{FA}$ is the delay of the F (G) network, $t_{[6:2]} = 3t_{FA}$ is the delay of the redundant adder, and $t_{reg} = 0.5t_{FA}$. For example, for $r = 16$ and 54-bit significand, we estimate that $T_{proposed} = (2 + 11 + 2 + 1 + 1 + 1)t_{cycle} = 18t_{cycle} = 108t_{FA}$.

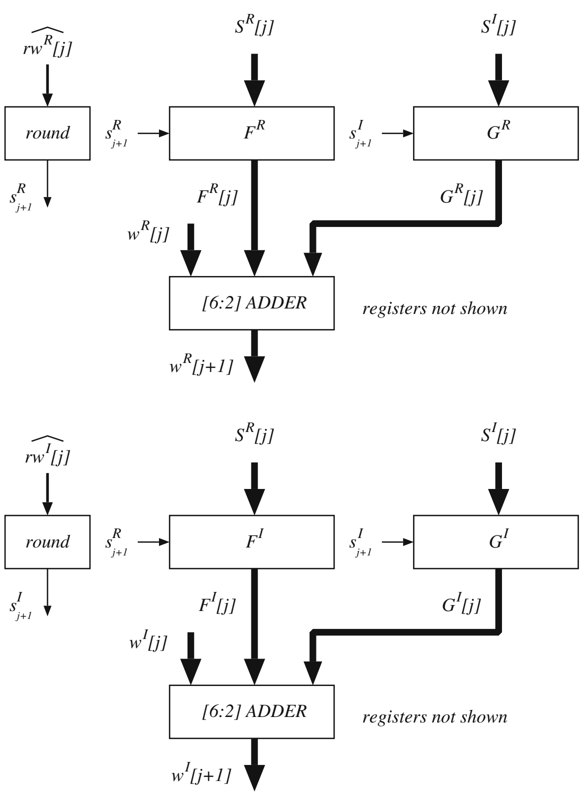


Figure 3. Block-diagram of implementation of the real and imaginary residual recurrences.

Table 2. Area of primitive modules (in A_{FA} units).

Module	Area
Register	$A_{reg} = 0.6n$
2-to-1 MUX	$A_{21mux} = 0.4n$
k-to-1 MUX, k=3, 4	$A_{kmux} = 0.8n$
[2:1] adder	$A_{[2:1]} = 0.5nlogn$
[3:2] adder.	$A_{[3:2]} = n$
[4:2] adder	$A_{[4:2]} = 2.3n$
[6:2] adder	$A_{[6:2]} = 4.3n$
SEL (round and recode	$A_{SEL} = 6$
On-the-fly converter	$A_{ofc} = 2A_{21mux} + 2A_{reg} = 2n$
OFC with rounding	$A_{ofcrnd} = 2A_{21mux} + 3A_{reg} = 3n$
LR multiplier	$A_{LRmul} = A_{kmux} + A_{[3:2]} + 2A_{reg} = 3n$
Complex rect. multiplier	$A_{Cmul} = 4(4A_{kmux} + A_{[3:2]} + A_{[4:2]}) + 2A_{[2:1]} + 2A_{reg} = (27.2 + 0.5logn)n$
Divider $r16$	$A_{div} = A_{sel} + 2A_{kmux} + A_{[3:2]} + 2A_{reg} + A_{ofcrnd} = 120 + 1.6n + 2n + 1.2n + 3n = 120 + 7.8n$
SQRT $r16$	$A_{sqrt} \approx A_{div}$

To get a rough comparison of the latency of the proposed scheme with a conventional complex square-root software implementation, we consider the algorithm defined in [21] which uses 3 real square-root and 2 real division operations in the floating-point format. There are also several comparison and absolute value operations. The estimated latency of this implementation is

$$T_{conv} = t_{DIV} + 2t_{SQR} + t_{DIV} \approx 4t_{SQR} \tag{24}$$

To achieve this latency two square-root units in parallel are required. We ignore exponent processing and rounding delays. Moreover, assuming a radix-16 digit-recurrence implementation of significand computation, we estimate

$$t_{SQR} \approx t_{prescal} + (n/4) \times t_{cycle} + t_{postscale} + t_{round} \tag{25}$$

where $t_{cycle} \approx t_{SEL} + t_{F-net} + t_{[4:2]} + t_{reg} = 4t_{FA}$. We assume that prescaling and postscaling is done for radix 16 so that the selection function is simplified.

For a 54-bit significand, we estimate that $T_{conv} = 4 \times (2 + 14 + 1 + 1)t_{cycle} = 288t_{FA}$ without taking into account comparison and absolute value operations. We conclude that the proposed scheme is at least 2.5 times faster than a conventional one.

To implement the real part of the proposed scheme we use the following main components: 2 multiple generators in F and G networks, a [6:2] adder, two registers, two sequential left-to-right multipliers without final adder, one on-line adder, and three registers for on-the-fly conversion with rounding. Similarly for the imaginary part. In addition we need a lookup table of $2q + 1$ inputs and a complex rectangular multiplier. The cost is measured as area occupied by modules using the area of a full-adder (A_{FA}) as the unit. The area of primitive modules is given in Table 2.

We also assume $r = 16$, $a = 10$, which has prescaling tables of size $2^{11} \times (3 \times 9)$ bits. equivalent to $A_{tbl} = 1.9KA_{FA}$. The area of the real (imaginary) recurrence, including postscaling and on-the-fly conversion/rounding, is

$$A_{real} = A_{SEL} + 2(A_{ofc} + A_{kmux} + A_{[3:2]}) + A_{[6:2]} + 2A_{reg}$$

Similarly for the imaginary part A_{imag}. The total area of the proposed scheme for $r = 16$ and $n = 54$ is estimated as

$$A_{CSQR} = 2A_{tbl} + A_{Cmul} + A_{real} + A_{imag} + 4 \times A_{LRmult} + 2 \times A_{ofcrnd} \approx 7.5KA_{FA}$$

We estimate that the cost of a conventional implementation using two FLPT square-units and a FLPT divider would be

$$A_{conv} = A_{div} + 2A_{sqr} \approx 2.5KA_{FA}$$

which is much smaller than the implementation of the proposed approach.

4. Summary

We proposed a new algorithm for complex square root. It uses two digit-recurrences and prescaling of the operand to allow result-digit selection by rounding. This makes the proposed scheme suitable for higher radices. The prescaling is more complicated than in the real case leading to larger lookup tables. Since the same prescaling is applicable to the digit-recurrence complex division proposed in [11], these two algorithms can be combined. A rough comparison with a conventional implementation based on floating-point instructions indicates a significant speedup of the proposed scheme at a higher cost. The proposed scheme allows correct rounding.

Acknowledgements

This research was supported in part by the State of California and STMicroelectronics MICRO Grant. We thank reviewers for their helpful comments.

Notes

1. Adaptation to carry-save notation is straightforward.
2. Multiplication by λ is straightforward: each time we get a new digit of the square root, we accumulate λ times that digit.
3. A straightforward implementation would require $2m$ address bits, but we use the fact that $a_1 = 1$.

References

1. G. Adams, A. M. Finn, and M. F. Griffin, "A Fast Implementation of the Complex Singular Value Decomposition on the Connection Machine," *IEEE Trans. Acoust. Speech Signal Process.*, 1991, pp. 1129–1132.
2. T. Ahrendt, "Fast High-precision Computation of Complex Square Roots," in *Proceedings of ISSAC'96*, Zurich, Switzerland, 1996.
3. D. Bindel, J. Demmel, W. Kahan, and O. Marques, "On Computing Givens Rotations Reliably and Efficiently," *ACM Trans. Math. Softw.*, vol. 28, no. 2, June 2002, pp. 206–238.
4. L. Ciminiera and P. Montuschi, "Higher Radix Square Rooting," *IEEE Trans. Comput.*, vol. 30, no. 10, October 1990, pp. 1220–1231.
5. D. Das Sarma and D. W. Matula, "Faithful Bipartite ROM Reciprocal Tables," in *Proceedings of the 12th IEEE Symposium on Computer Arithmetic*, Bath, UK, S. Knowles and W. McAllister (Eds.), IEEE Computer Society Press, Los Alamitos, CA, July 1995.
6. F. de Dinechin and A. Tisserand, "Some Improvements on Multipartite Table Methods," in *Proceedings of the 15th IEEE Symposium on Computer Arithmetic*, Vail, Colorado, L. Ciminiera and N. Burgess (Eds.), IEEE Computer Society Press, Los Alamitos, CA, June 2001.

7. M. A. Elliot, G. A. Walker, A. Swift, and K. Vandenborne, "Spectral Quantitation by Principal Component Analysis using Complex Singular Value Decomposition," *Magn. Reson. Med.*, vol. 41, 1999, pp. 450–455.
8. M. D. Ercegovac and T. Lang, "Fast Multiplication without Carry-propagate Addition," *IEEE Trans. Comput.*, vol. C-39, no. 11, November 1990, pp. 1385–1390.
9. M. D. Ercegovac and T. Lang, *Division and Square Root: Digit-recurrence Algorithms and Implementations*, Kluwer, Boston, MA, 1994.
10. M. D. Ercegovac and T. Lang, *Digital Arithmetic*, Morgan Kaufmann, San Mateo, CA, 2004.
11. M. D. Ercegovac and J.-M. Muller, "Complex Division with Prescaling of the Operands," in *Proc. Application-Specific Systems, Architectures, and Processors (ASAP'03)*, The Hague, The Netherlands, June 24–26, 2003.
12. M. D. Ercegovac and J.-M. Muller, "Complex Square Root with Operand Prescaling," in *Proc. Application-Specific Systems, Architectures, and Processors (ASAP'04)*, Galveston, TX, 27–29 September, 2004, pp. 52–62.
13. N. Hemkumar and J. Cavallaro, "*A Systolic VLSI Architecture for Complex SVD*, *Proc. of the IEEE International Symposium on Circuits and Systems*," 1992, pp. 1061–1064.
14. T. E. Hull, T. F. Fairgrieve, and P. T. P. Tang, "Implementing Complex Elementary Functions Using Exception Handling," *ACM Trans. Math. Softw.*, vol. 20, no. 2, 1994, pp. 215–244.
15. W. Kahan, "Branch Cuts for Complex Elementary Functions, or Much Ado About Nothing's Sign Bit," in *The State of the Art in Numerical Analysis*, Clarendon Press, Oxford, 1987.
16. R. D. Mcilhenny, *Complex Number On-line Arithmetic for Reconfigurable Hardware: Algorithms, Implementations, and Applications*. PhD thesis, University of California at Los Angeles, 2002.
17. T. Lang and P. Montuschi, "Higher Radix Square Root with Prescaling," *IEEE Trans. Comput.*, vol. 41, no. 8, 1992, pp. 996–1009.
18. T. Lang and P. Montuschi, "Very High Radix Square Root with Prescaling and Rounding and A Combined Division/Square Root Unit," *IEEE Trans. Comput.*, vol. 48, no. 8, 1999, pp. 827–841.
19. J. Mitroy and I. A. Ivallov, "Quantum Defect Theory for the Study of Hadronic Atoms," *J. Phys., G, Nucl. Part. Phys.*, vol. 27, 2001, pp. 1–13.
20. P. Montuschi and M. Mezzalama, "Survey of Square Rooting Algorithms," *IEE Proceedings E: Computers and Digital Techniques*, vol. 137, no. 1, pp. 31–40.
21. W. Press, S. A. Teukolski, W. T. Vetterling, and B. F. Flannery, *Numerical Recipes in C*, 2nd Edition, Cambridge University Press, 1992.
22. M. J. Schulte and J. E. Stine, "Approximating Elementary Functions with Symmetric Bipartite Tables," *IEEE Trans. Comput.*, vol. 48, no. 8, Aug. 1999, pp. 842–847.
23. J. Salo, J. Fagerholm, A. T. Friberg, and M. M. Salomaa, "Unified Description of Nondiffracting X and Y Waves," *Phys. Rev.*, vol. E 62, 2000, pp. 4261–4275.
24. D. M. Smith, "Algorithm 768: Multiple-Precision Complex Arithmetic and Functions," *ACM Trans. Math. Softw.*, vol. 24, no. 4, December 1994, pp. 359–367.
25. R. D. Susanto, Q. Zheng, and X.-H. Yan, "Complex Singular Value Decomposition," *J. Atmos. Ocean. Technol.*, vol. 15, no. 3, 1998, pp. 764–774.

Dr. Miloš D. Ercegovac is a Professor and a former Chair in the Computer Science Department of the Henry Samueli School of Engineering and Applied Science, University of California at Los Angeles, where he has been on the faculty since 1975. He earned his MS (1972) and PhD (1975) in computer science from the University of Illinois, Urbana-Champaign, and BS in electrical engineering (1965) from the University of Belgrade, Serbia. Dr. Ercegovac has specialized for over 30 years in research and teaching in digital arithmetic, digital and computer system design, and parallel architectures. His life-long dedication to teaching and research has also resulted in several co-authored books: two in the area of digital design (*Digital Systems and Hardware/Firmware Algorithms*, Wiley & Sons, 1985, and *Introduction to Digital Design*, Wiley & Sons, 1999), and two in digital arithmetic (*Division and Square Root: Digit-Recurrence Algorithms and Implementations*, Kluwer Academic Publishers, 1994, and *Digital Arithmetic*, Morgan Kaufmann Publishers—a Division of Elsevier, 2004.) Dr. Ercegovac has been involved in organizing the IEEE Symposia on Computer Arithmetic since 1978. He served as an associate editor of the IEEE Transactions on Computers 1988–1992 and as a subject area editor for the Journal of Parallel and Distributed Computing 1986–1993. Dr. Ercegovac's work has been recognized by his election in 2003 to IEEE Fellow and to Foreign Member of the Serbian Academy of Sciences and Arts in Belgrade, Serbia. He is also a member of the ACM and of the IEEE Computer Society.

Dr Jean-Michel Muller was born in Grenoble, France, in 1961. He received his PhD degree in 1985 from the Institut National Polytechnique de Grenoble. He is

Directeur de Recherches (senior researcher) at CNRS, France, and he is the former head of the LIP laboratory (LIP is a joint laboratory of CNRS, Ecole Normale Supérieure de Lyon, INRIA and Université Claude Bernard Lyon 1). His research interests are in Computer Arithmetic. Dr Muller was co-program chair of the 13th IEEE Symposium on Computer Arithmetic (Asilomar, USA, June 1997), general chair of the 14th IEEE Symposium on Computer Arithmetic (Adelaide, Australia, April 1999). He is the author of several books, including *Elementary Functions, Algorithms and Implementation* (2nd edition, Birkhäuser Boston, 2006). He served as associate editor of the IEEE Transactions on Computers from 1996 to 2000. He is a senior member of the IEEE.