Complex Square Root with Operand Prescaling

Miloš D. Ercegovac
Computer Science Department, 3732 Boelter Hall
University of California at Los Angeles
Los Angeles, CA 90095, USA
milos@cs.ucla.edu

Jean-Michel Muller
CNRS-Laboratoire CNRS-ENSL-INRIA-UCBL LIP
Ecole Normale Supérieure de Lyon
46 Allée d'Italie, 69364 Lyon Cedex 07, FRANCE
Jean-Michel.Muller@ens-lyon.fr

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. Moreover, the use of digit recurrence approach allows correct rounding of the result. The algorithm, compatible with the complex division, and its design are described at a high-level. We also give rough comparisons of its latency and cost with respect to implementation based on standard floating-point instructions as used in software routines for complex square root.

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, 11, 23], and in applications such as principal component analysis [7], quantum defect theory [17] and wave propagation [21].

Complex square-root operation is commonly implemented in software based on various algorithms developed for reliable and accurate evaluation in languages like FORTRAN 90 [12]. There are also collections of Fortran routines for multiple-precision complex arithmetic which include complex square-root [22]. 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 [14], who used an adaptation of on-line arithmetic to (1). With a rapid increase in capacity of integrated circuits, it is timely to consider hardware-based implementation of an extended

Proceedings of the 15th IEEE International Conference on Application-Specific Systems, Architectures and Processors (ASAP'04)
1063-6862/04 $ 20.00 IEEE

set of operations. In this research we focus on hardware-oriented algorithms and implementations of operations on operands in the complex domain. In [10] we proposed and developed an algorithm and its 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 $+, -, \times, \div, \sqrt{}$. 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 [19] 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

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

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

The objective of this paper is to develop an algorithm similar to the usual digit-recurrence real square-root algorithm [18, 8, 9], suitable for hardware implementation. 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 quotient digits q_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 [8, 9], 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 [15, 16]. 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 $w[j]$ 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 $\sqrt{(\Re(z))^2 + (\Im(z))^2}$.

2 Complex square-root algorithm

2.1 Basic iteration

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

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

$$W[j] = r^j \left(d - (S[j])^2\right) \tag{3}$$

From (3), we obtain the residual recurrence

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

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

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

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:

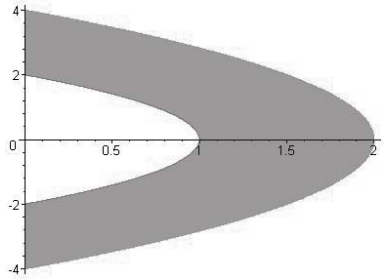

Figure 1. *The domain where the real part $s_1^{\mathcal{R}}$ of digit s_1 can be chosen equal to* 1*, assuming* $s_0 = 1 + 0 \times i$*. This shows that prescaling is needed to simplify the digit selection.*

- 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.

2.2 Prescaling

The prescaling part of the algorithm is similar to that of complex division [10]. Using an input number x (for simplicity, we assume $1/2 \leq ||x||_\infty < 1$), we obtain from a look-up table 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 two registers, one for the real and another for the imaginary part.

Using a direct table lookup, the prescaling step requires a table with $2q+1$ address bits (see [10] for details). For larger values of q the use of the bipartite method [5, 20] adapted to complex functions [10] results in smaller tables.

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

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

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

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

In particular, for $q > 2$:

$$\left|\left|\sqrt{d}-1\right|\right| < \frac{9}{16}2^{-q}. \tag{7}$$

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

- for the real part:
$$1.\underbrace{000\cdots00}_{q \text{ zeros}}$$
- for the imaginary part:
$$0.\underbrace{000\cdots00}_{q \text{ zeros}}$$

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 (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:
$$1.\ \underbrace{000\cdots00}_{j_0=\lfloor q/k \rfloor \text{ zeros}}$$
- for the imaginary part:
$$0.\ \underbrace{000\cdots00}_{j_0=\lfloor q/k \rfloor \text{ zeros}}$$

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 start the iterations from step $j_0 = \lfloor q/k \rfloor$ since (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 $s_{j+1}^{\mathcal{R}}$ 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 $s_{j+1}^{\mathcal{I}}$ as the integer closest to the value $1/2rW^{\mathcal{I}}[j]$ truncated after the σ-th borrow-save position. This implies that

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

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 (10), we get $||\epsilon||_\infty = \max\{|\epsilon^{\mathcal{R}}|, |\epsilon^{\mathcal{I}}|\} < r^{-\lfloor q/k \rfloor}$. Since

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

we find

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

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

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 (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 (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.

Table 1 gives parameters q and σ that satisfy (14), depending on r and a. Since the table required by the prescaling step has $2q + 1$ address bits, radices 2 and 4 are feasible. Higher radices will require bipartite tables as discussed in [10]. The prescaling for the complex square root is the same as for the complex division algorithm [10], thus simplifying implementation of a combined scheme.

[1] Adaptation to carry-save notation is straightforward.

Table 1. Parameters q and σ that satisfy (14), depending on r and a.

r	a	q	σ
2	1	4	4
2	1	6	3
4	2	6	5
4	3	6	3
8	4	9	5
8	5	9	3
8	6	6	5
8	7	6	4
16	8	12	6
16	9	8	9
16	10	8	5
16	11	8	4
16	12	8	3
16	15	8	2

3 Implementation and Comparison with Other Methods

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 Figure 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 of the scaling factor $K = (K^{\mathcal{R}}, K^{\mathcal{I}})$ and of 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 [9], 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 will assume that the adder is of a signed-digit type (i.e., borrow-save). We now apply the form of (15) to the complex residual recurrence 5:

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

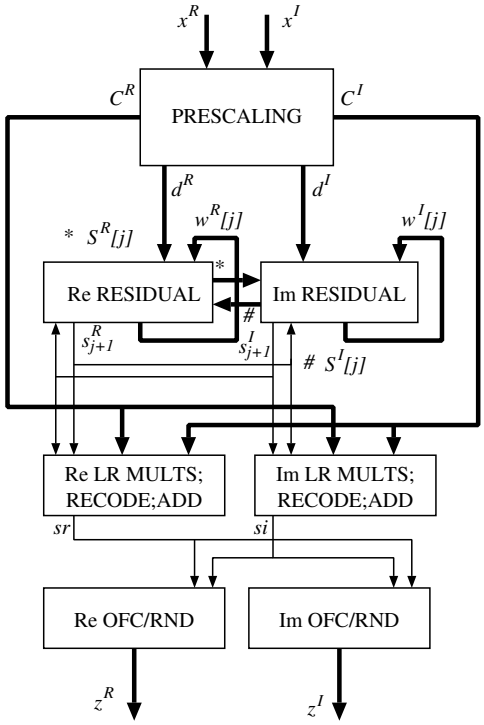

Figure 2. *Overall scheme for computing complex square root.*

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:

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

The block-diagram of implementation of the complex recurrence is shown in Figure 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. 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 [9], 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:

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

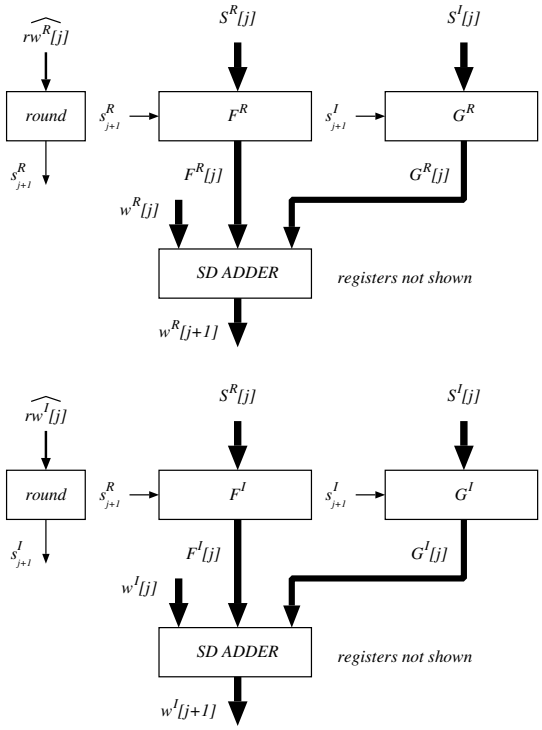

Figure 3. *Block-diagram of implementation of the real and imaginary residual recurrences.*

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

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

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 Figure 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 $T_{proposed} = (2 + 11 + 2 + 1 + 1 + 1)t_{cycle} = 18t_{cycle} = 108t_{FA}$.

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 [19] 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

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

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. We estimate that the cost of a conventional implementation using two FLPT square-units and a FLPT divider would be no less than the cost of the proposed scheme. A detailed design remains to be done.

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 [10], 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 similar cost. Moreover, the proposed scheme allows correct rounding.

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 Transactions on Acoustics, Speech and Signal Processing*, pp.1129-1132, 1991.

[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 Transactions on Mathematical Software*, 28(2):206-238, June 2002.

[4] L. Ciminiera and P. Montuschi. Higher Radix Square Rooting. *IEEE Transactions on Computers*, 30(10):1220-1231, October 1990.

[5] D. Das Sarma and D. W. Matula. Faithful bipartite ROM reciprocal tables. In S. Knowles and W. McAllister, editors, *Proceedings of the 12th IEEE Symposium on Computer Arithmetic*, Bath, UK, July 1995. IEEE Computer Society Press, Los Alamitos, CA.

[6] F. de Dinechin and A. Tisserand. Some Improvements on Multipartite Table Methods. In L. Ciminiera and N. Burgess, editors, *Proceedings of the 15th IEEE Symposium on Computer Arithmetic*, Vail, Colorado, June 2001. IEEE Computer Society Press, Los Alamitos, CA.

[7] M.A. Elliot, G.A. Walker, A. Swift, and K. Vandenborne. Spectral Quantitation by Principal Component Analysis using Complex Singular Value Decomposition. *Magnetic Resonance in Medicine*, 41:450-455, 1999.

[8] M. D. Ercegovac and T. Lang. *Division and Square Root: Digit-Recurrence Algorithms and Implementations*. Kluwer Academic Publishers, Boston, 1994.

[9] M. D. Ercegovac and T. Lang. *Digital Arithmetic*. Morgan Kaufmann Publishers, 2004.

[10] 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.

[11] N. Hemkumar and J. Cavallaro. A Systolic VLSI Architecture for Complex SVD. *Proc. of the IEEE International Symposium on Circuits and Systems*, pp.1061-1064, 1992.

[12] T. E. Hull, T. F. Fairgrieve, and P. T. P. Tang. Implementing Complex Elementary Functions Using Exception Handling. *ACM Transactions on Mathematical Software*, 20(2):215-244, 1994.

[13] 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.

[14] R.D. Mcilhenny. *Complex Number On-line Arithmetic for Reconfigurable Hardware: Algorithms, Implementations, and Applications*. PhD thesis, University of California at Los Angeles, 2002.

[15] T. Lang and P. Montuschi. Higher Radix Square Root with Prescaling. *IEEE Transactions on Computers*, 41(8):996–1009, 1992.

[16] T. Lang and P. Montuschi. Very High Radix Square Root with Prescaling and Rounding and A Combined Division/Square Root Unit. *IEEE Transactions on Computers*, 48(8):827–841, 1999.

[17] J. Mitroy and I.A. Ivallov. Quantum Defect Theory for the Study of Hadronic Atoms. *Journal of Physics*, G: Nuclear and Particle Physics, 27, pp. 1-13, 2001.

[18] P. Montuschi and M. Mezzalama. Survey of Square Rooting Algorithms. *IEE Proceedings E: Computers and Digital Techniques*, 137(1): 31–40.

[19] W. Press and S. A. Teukolski and W. T. Vetterling and B. F. Flannery. *Numerical Recipes in C*, 2nd Edition. Cambridge University Press, 1992.

[20] M.J. Schulte and J.E. Stine. Approximating elementary functions with symmetric bipartite tables. *IEEE Transactions on Computers*, 48(8):842–847, Aug. 1999.

[21] J. Salo and J. Fagerholm and A. T. Friberg and M. M. Salomaa. Unified description of nondiffracting X and Y waves. *Phys. Rev.* E 62, pp. 42614275, 2000.

[22] D. M. Smith. Algorithm 768: Multiple-Precision Complex Arithmetic and Functions. *ACM Transactions on Mathematical Software*, 24(4):359-367, December 1994.

[23] R.D. Susanto, Q. Zheng, and X-H. Yan. Complex Singular Value Decomposition. *Journal of Atmospheric and Oceanic Technology*, 15(3):764-774, 1998.