# Design of a complex divider

Miloš D. Ercegovac[a] and Jean-Michel Muller[b]

[a]Computer Science Department, University of California, Los Angeles, California, U.S.A.;
[b]CNRS-Laboratoire CNRS-ENSL-INRIA-UCBL LIP, Ecole Normale Superieure de Lyon, Lyon, France

## ABSTRACT

We describe a hardware-oriented design of a complex division algorithm proposed in .[1] This algorithm is similar to a radix-$r$ digit-recurrence division algorithm with real operands and prescaling. Prescaling of complex operands allows efficient selection of complex quotient digits in higher radix. The use of the digit-recurrence method allows hardware implementation similar to that of conventional dividers. Moreover, this method makes correct rounding of complex quotient possible. On the other hand, the proposed scheme requires the use of prescaling tables which are more demanding than tables in similar dividers with real operands. In this paper we present main design ideas, implementation details, and give a rough estimate of the expected latency. We also make a comparison with the estimated latency of the Smith's algorithm used in software routines for complex division.

**Keywords:** Complex division, digit-recurrence algorithms, prescaling, radix $r$, hardware implementation

## 1. INTRODUCTION

As mentioned in,[1] complex division is used in diverse applications such as earthfault distance protection,[2] acoustic pulse reflectometry,[3] astronomy,[4] non-linear RF measurement,[5] as well as in many digital signal processing applications such as the complex SVD . While hardware implementations of complex multipliers are common, complex division has been largely provided in software.

A software routine using the conventional formula

$$\frac{a+ib}{c+id} = \frac{ac+bd+i(bc-ad)}{c^2+d^2}. \tag{1}$$

may lead to overflows during the intermediate computations although the final result is representable in the floating-point format. This drawback has been discussed in [6,7] and by Smith[8] who proposed a more robust algorithm:

$$\frac{a+ib}{c+id} = \begin{cases} \dfrac{a+b(d/c)}{c+d(d/c)} + i\dfrac{b-a(d/c)}{c+d(d/c)} & (\text{ if } |c| \geq |d|) \\ \dfrac{b+a(c/d)}{d+c(c/d)} + i\dfrac{a-b(c/d)}{d+c(c/d)} & (\text{ if } |c| \leq |d|) \end{cases} \tag{2}$$

Stewart[9] proposes further improvements to the Smith's algorithm resulting in even more complicated algorithm. In spite of their robustness, these algorithms cannot guarantee correct rounding of the real and imaginary parts of the quotient. A direct hardware implementation of the Smith's or Stewart's algorithms is not attractive since it would require costly hardware implementation without significant performance gain over software-based solutions.

Hardware algorithms and implementations for complex division have not been studied frequently. A hardware implementation of complex division has been developed in[10] using a radix-2 on-line algorithm and implemented

Further author information: (Send correspondence to M.D.E.)
M.D.E.: E-mail: milos@cs.ucla.edu, Telephone: 1 310 825 5414
J.-M.M.: E-mail: Jean-Michel.Muller@ens-lyon.fr



Advanced Signal Processing Algorithms, Architectures, and Implementations XIV, edited by Franklin T. Luk, Proc. of SPIE Vol. 5559 (SPIE, Bellingham, WA, 2004)
0277-786X/04/$15 · doi: 10.1117/12.560154

(1) on a reconfigurable hardware. This approach, however, is not suitable for higher radices because it uses quotient-digit selection function with selection constants.

The key idea of the approach proposed in[1] is to apply a high-radix (real) digit-recurrence division algorithm with prescaling of operands[11, 12] to complex division. That is, the proposed division algorithm uses a standard residual recurrence with complex variables

$$w[j+1] = rw[j] - q_{j+1}y \tag{3}$$

where $w[0] = x$ is the complex dividend, $y$ is the complex divisor, and the complex quotient digits $q_j$'s are chosen in a radix-$r$ redundant digit-set, so that the complex residuals $w[j]$ remain bounded.

Several approaches to quotient-digit selection in high radix division algorithms have been discussed in the literature.[12] Among them, the *prescaling* technique is most suitable for higher radix: it consists of multiplying the dividend $x$ and the divisor $y$ by a constant $K$ such that $Ky$ is close to 1. Consequently, one can choose $q_{j+1}$ by rounding $rw[j]$ (or its low-precision estimate) to the nearest integer.

In the rest of the paper we first review the algorithm for complex division. Then we discuss a design and implementation details followed by a rough estimate of the expected cycle time and overall latency. We compare these estimates with delay characteristics of compatible dividers operating on real operands.

Throughout the paper, $i$ is $\sqrt{-1}$, and if $z$ is a complex number, then $zR = Re(z)$ and $zI = Im(z)$ denote the real and imaginary parts of $z$. The norm $||z||_\infty$ denotes $\max\{|zR|, |zI|\}$, whereas $|z|$ denotes the usual complex absolute value $\sqrt{(zR)^2 + (zI)^2}$.

## 2. COMPLEX DIVISION ALGORITHM - AN OVERVIEW

### 2.1. General scheme

The algorithm for complex division developed in[1] computes

$$q = qR + iqI = z/d = \frac{zR + izI}{dR + idI} \tag{4}$$

by performing the following steps:

**Prescaling.** Obtain a complex scaling factor $K = K1 + iK2$ as a short approximation of $1/d$ such that

$$||Kd - 1||_\infty < 2^{-p} \tag{5}$$

where the integer $p$ depends on the radix $r$. Then, compute scaled complex dividend and divisor

$$\begin{cases} w[0] &= zK \\ y &= dK \end{cases} \tag{6}$$

Prescaling of the divisor and dividend allows simplified and separate selection of the real and imaginary parts of the complex quotient digits.

**Iterations.** Perform radix-$r = 2^k$ iterations to produce $n + 1$ quotient digits

$$w[j+1] = rw[j] - q_{j+1}y \tag{7}$$

where $r$ is the radix of the division, $q_{j+1} = qR_{j+1} + iqI_{j+1}$, and $qR_{j+1}$ and $qI_{j+1}$ belong to the redundant digit-set $\mathcal{DS} = \{-a, \ldots, a\}$.

Any choice of the $q_j$'s for which the $||w[j]||_\infty$'s remain bounded 1 suffices to produce the quotient digits satisfying

$$0.qR_1qR_2qR_3 \cdots qR_n + i0.qI_1qI_2qI_3 \cdots qI_n \rightarrow z/d$$

### 2.2. Prescaling expressions

The expressions for the prescaling of complex operands using $K = K1 + iK2$ are

$$\begin{cases} xR &= zR \cdot K1 - zI \cdot K2 \\ xI &= zI \cdot K1 + zR \cdot K2 \\ & \\ yR &= dR \cdot K1 - dI \cdot K2 \\ yI &= dI \cdot K1 + dR \cdot K2 \end{cases} \tag{8}$$

which can be implemented with rectangular multipliers.

### 2.3. The complex residual recurrences

The recurrences for computing real and imaginary residuals, given below, can be computed in parallel. They contain one additional term compared to the residual recurrence of conventional division.

$$\begin{cases} wR[j+1] &= rwR[j] - qR_{j+1}yR + qI_{j+1}yI \\ wI[j+1] &= rwI[j] - qI_{j+1}yR - qR_{j+1}yI \end{cases} \tag{9}$$

### 2.4. Selection of the quotient digits

Since the prescaled divisor is close to 1, the $(j+1)$-st quotient digit can be obtained by rounding the residual to its integer part. Moreover, a short-precision estimate of the residual suffices for this rounding so that residuals can be obtained in redundant form (e.g., carry-save or signed-digit). That is, the selection is performed by rounding to the nearest integer the real and imaginary parts of $\widehat{w}[j]$, where $\widehat{w}[j]$ is obtained by truncating a redundant $w[j]$ after some fractional position $t$. Specifically,

$$\begin{aligned} qR_{j+1} &= Sel\left(\widehat{rwR[j]}\right) \\ qI_{j+1} &= Sel\left(\widehat{rwI[j]}\right) \end{aligned} \tag{10}$$

where the quotient digits $qR_{j+1}$ and $qI_{j+1}$ are from the redundant digit-set $\mathcal{DS} = \{-a, \ldots, a\}$ with $2a + 1 > r$. Due to rounding of a truncated residual the selection function $Sel$ satisfies

$$|Sel(x) - x| < \frac{1}{2} + 2^{-t} \tag{11}$$

One can meet this requirement by truncating a carry-save representation of $x$ after the $(t+1)$-st fractional position, and rounding the obtained result to the nearest integer. If $x$ is represented in borrow-save form, it suffices to truncate it after the $t$-th fractional position before rounding it to the nearest. In this paper we use a carry-save representation.

The containment condition of the residual guarantees that the selection function (10) always produces a digit from the set $\mathcal{DS}$, is

$$||w[j]||_\infty \leq \frac{1}{r}\left(a + \frac{1}{2} - 2^{-t}\right) = \Omega \tag{12}$$

Since $\epsilon_y = y - 1$. From (7) and (11), we get

$$\begin{aligned} w[j+1] &= -\epsilon_y qR_{j+1} - \left[Sel(\widehat{rwR[j]}) - rwR[j]\right] \\ &\quad -i\epsilon_y qI_{j+1} - i\left[Sel(\widehat{rwI[j]}) - rwI[j]\right] \end{aligned} \tag{13}$$

| $r$ | $a$ | $p$ | $t$ | $\Omega$ |
|---|---|---|---|---|
| 2 | 1 | 4 | 4 | 23/32 |
| | 2 | 3 | 3 | 19/16 |
| 4 | 2 | 6 | 5 | 79/128 |
| | 3 | 5 | 3 | 27/32 |
| | 4 | 4 | 4 | 71/64 |
| 8 | 4 | 8 | 6 | 287/512 |
| | 7 | 6 | 3 | 59/64 |
| | 8 | 5 | 5 | 271/256 |
| 16 | 8 | 10 | 7 | 1087/2048 |
| | 15 | 7 | 3 | 123/128 |
| | 16 | 6 | 6 | 1055/1024 |

**Table 1.** Values of the parameters $p$, $t$ and $\Omega$ of the algorithm, depending on $r$ and $a$.

or,

$$w[j+1] = \epsilon_y q_{j+1} + A + iB \tag{14}$$

where $A$ and $B$ are real numbers of absolute value less than $\frac{1}{2} + 2^{-t}$, $||q_{j+1}||_\infty \leq a$ and $||\epsilon_y||_\infty < 2^{-p}$. From that, we get

$$||w[j+1]||_\infty < 2^{-p+1}a + \frac{1}{2} + 2^{-t} \tag{15}$$

Finally, the residual bounds (12) and (15) define the relation between the parameters $a$, $p$, $r$ and $t$ which guarantees the convergence of the complex division algorithm:

$$2^{-p+1}a + \frac{1}{2} + 2^{-t} \leq \frac{1}{r}\left(a + \frac{1}{2} - 2^{-t}\right) \tag{16}$$

Examples of parameters that meet these constraints are given in Table 1.

## 3. PRESCALING OF OPERANDS

As mentioned above, the method obtains a scaling factor $K$ from the divisor $d$ such that $||Kd-1||_\infty < 2^{-p}$. We consider a table lookup approach and determine the number of address bits and the precision of entries in the table. We assume that the divisor $d = dR + idI$ is normalized, i.e.,

$$\frac{1}{2} \leq ||d||_\infty < 1. \tag{17}$$

Consequently, $dR_1 = 1$ or $dI_1 = 1$. Let $a = \widehat{dR} = \{dR\}_q$ and $b = \widehat{dI} = \{dI\}_q$ be the estimates of the real and imaginary parts of the divisor rounded to the nearest $q$-fractional bit number, respectively. The scaling factor, defined as

$$K = \frac{1}{a+ib} = \frac{a}{a^2+b^2} - \frac{ib}{a^2+b^2} = K1 + iK2 \tag{18}$$

is obtained from a table addressed by $2q-1$ address bits[1] where $q = p+1$. As derived in ,[1] this guarantees that $y = dK$ satisfies $||y-1||_\infty < 2^{-p}$. Since the scaling factor $K$ is an approximation of $1/d$ with $p$ bits of relative accuracy, we represent $K1$ and $K2$ with $p+1$ bits of precision after rounding.

For the parameter values shown in Table 1, we obtain table sizes in Table 2. Note that over-redundant digit set reduces the table size. A direct table lookup approach, except for $r = 2$ and $r = 4$ with $a = 4$, is not attractive. To reduce the table size at expense of additional computation, one may consider various interpolation

techniques. Alternatively, a possible solution to this problem is, as suggested in,[1] to generalize the bipartite table method of [13, 14] to functions of two variables. In this approach, instead of a single table with $2q-1$ address bits, we use three tables, each of them with $4q/3$ address bits. Consequently, in all cases listed in Table 2, lookup tables do not require more than $1K$ words except for $r = 16$ with $a = 15$. We do not consider further these techniques and details of their implementation in this paper.

| $r$ | $a$ | $p$ | $2p+1$ | $Size$ $(single)$ | $Size$ $(bipartite)$ |
|---|---|---|---|---|---|
| 2 | 1 | 4 | 9 | $2^9 \times 10$ | $(2^7 \times 10) \times 3$ |
| | 2 | 3 | 7 | $2^7 \times 8$ | $(2^6 \times 8) \times 3$ |
| 4 | 2 | 6 | 13 | $2^{13} \times 14$ | $(2^{10} \times 14) \times 3$ |
| | 3 | 5 | 11 | $2^{11} \times 12$ | $(2^8 \times 12) \times 3$ |
| | 4 | 4 | 9 | $2^9 \times 10$ | $(2^7 \times 10) \times 3$ |
| 8 | 7 | 6 | 13 | $2^{13} \times 14$ | $(2^{10} \times 14) \times 3$ |
| | 8 | 5 | 11 | $2^{11} \times 12$ | $(2^8 \times 12) \times 3$ |
| 16 | 15 | 7 | 15 | $2^{15} \times 16$ | $(2^{11} \times 16) \times 3$ |
| | 16 | 6 | 13 | $2^{13} \times 14$ | $(2^{10} \times 14) \times 3$ |

**Table 2.** Examples of table sizes for single and bipartite table lookup.

## 4. DESIGN DESCRIPTION

In the following sections we discuss the design of the complex divider and its main modules.

### 4.1. General organization

The overall scheme is shown in Figure 1. It consists of a table-lookup module TBL-K producing the real ($K1$) and imaginary ($K2$) parts of the complex scaling factor $K$, two shared modules for performing the scaling multiplications and for performing real/imaginary recurrences including quotient-digit selection, and two modules for converting the redundant quotient representation to the conventional form combined with rounding (ROFC/RND, IOFC/RND).

### 4.2. Scaling factor module

As discussed above, the scaling factor is produced by a table-lookup (direct or bipartite) and, if necessary, combined with a suitable interpolation scheme. We do not discuss design of this module since we assume the use of a table-lookup. Based on work in,[15] we estimate that the complex scale factor $K = kR + iKI$, with $2p+1$ input bits and $2 \times (p+1)$ output bits, is obtained in time shown in Table 3. The access time is expressed in $\tau$'s where $\tau$ corresponds to the delay of a complex gate such as full adder. The area is expressed in $\alpha$'s where $\alpha$ corresponds to the area of a complex gate.[15]

### 4.3. Scaling/Recurrence modules

Figure 2 shows the modules for performing scaling of the operands and for the the residual recurrence evaluation. There are two modules, one for the real part and another for the imaginary part computations. During the operands scaling, the modules in Figure 2 first compute in one cycle the carry-save form of the real $(yRc, yRs)$ and imaginary $(yIc, yIs)$ parts of the scaled divisor $y$. These redundant forms are converted to $yR$ and $yI$ using a 2-cycle CPA. During the second cycle the dividend is scaled. The produced carry-save forms of the scaled dividend are used to initialize the residual registers in the real and imaginary recurrences. The multiple generators (M-GEN) are implemented as rectangular multipliers without final carry-propagate addition. The short operand corresponds to a radix $r$ digit (recoded into radix-4 signed digit form) of the scaling factor K1 (K2) or the quotient digit $qR_{j+1}$ $(qI_{j+1})$.

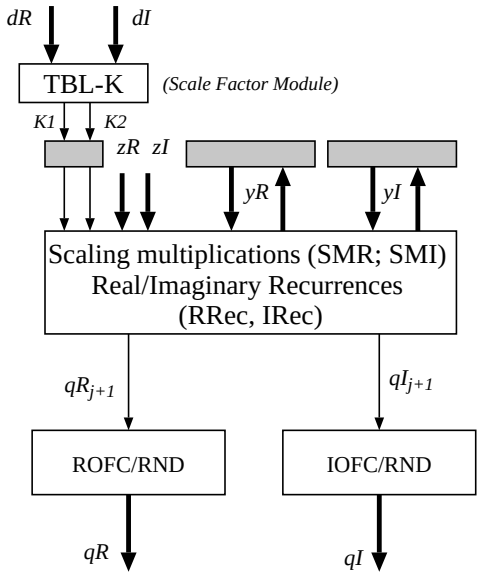


**Figure 1.** Overall scheme for complex division.

| $Address$ $(bits)$ | $Delay$ $\tau$ | $Area$ $\alpha$ |
|---|---|---|
| 6 | 2.5 | $40/Kbit$ |
| 7 | 3 | $35/Kbit$ |
| 8 | 3.5 | $35/Kbit$ |
| 9 | 4 | $35/Kbit$ |
| $10-11$ | 4.5 | $35/Kbit$ |
| $12-13$ | 5 | $30/Kbit$ |
| $14-15$ | 6 | $25/Kbit$ |

**Table 3.** Lookup tables: Access time and area.

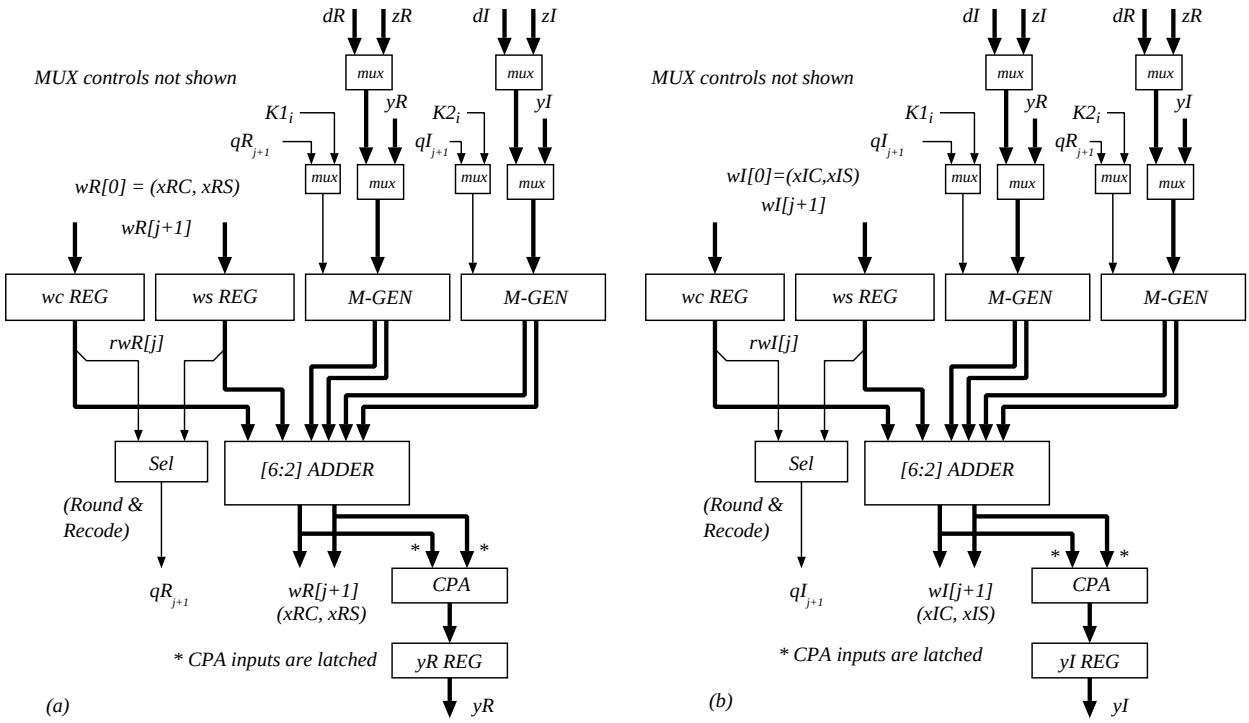


**Figure 2.** Modules for performing scaling and recurrence (real and imaginary parts).

During the execution of the recurrence, the M-GEN modules use one radix $r = 2^k$ quotient-digit per iteration. This digit is recoded into $k'$ radix-4 signed-digits where $k'$ is roughly $k/2$ depending on $k$ and the redundancy in the digit sets. Therefore, the M-GEN modules provide a reduction of $k'$ operands. We assume that a $[k'{:}2]$ adder is used since producing a non-redundant multiple would increase the cycle time more than having an extra operand accommodated in the recurrence redundant adder. During the scaling, the real and imaginary factors $K1$ and $K2$ are recoded into $p'$ minimally recoded radix-4 signed signed digits where $p'$ is roughly $p/2$ as stated above for $k'$. Since $p' > k'$, the number of operands forming a product during scaling is larger than in the recurrence computation. In order to limit the number of additional cycles in the prescaling multiplication, one possibility is to apply extra operands via multiplexed $wR$ ($wI$) inputs (not shown). The recurrence adder, as indicated in the figure, is a [6:2] adder which can be decomposed into [3:2] adders followed by a [4:2] with a delay of $\tau + 1.5\tau = 2.5\tau$.

We now estimate the cycle time in the scaling/recurrence module:

$$t_{cycle} = t_{Sel} + t_{4:1mux} + t_{[k':2]} + t_{[6:2]} + t_{reg} \tag{19}$$

where

| | |
|---|---|
| $t_{Sel} \approx 2\tau$ | - the delay of the selection function based on rounding |
| $t_{4:1mux} = 1.5\tau$ | - the delay of 4-to-1 multiplexer including the buffering of control |
| $t_{k':2} = 1\tau$ | - the delay $[k'{:}2]$ adder for $k' = 3$; $1.5\tau$ for $k' = 4$; $2.5\tau$ for $k' = 5,\ 6$, etc. |
| $t_{[6:2]} = 2.5\tau$ | - the delay of [6:2] adder |
| $t_{reg} = 1\tau$ | - the delay of a register |

For example, for $r = 16$ and $a = 15$, $t_{cycle} \approx 8\tau$.

### 4.4. On-the-fly conversion and rounding module

The redundant quotient digits (real and imaginary) are converted using on-the-fly conversion to produce a quotient in the conventional form. As discussed in,[16] the rounding can be incorporated into the conversion.

The conversion causes no additional delay during iterations. The rounding, however, requires obtaining one additional quotient digit and a determination of the sign of the last residual. The CPAs used in scaling of the operands produce the sign of the last residuals. We estimate that this can be accomplished in two cycles.

## 5. EVALUATION AND COMPARISON

We now estimate latency of the complex division implementation for a particular radix and compare it with an implementation based on the Smith's formula.

For $r = 16$ and $n = 54$-bit result, the latency of the proposed scheme can be estimated as

$$\begin{aligned} T_{CDIV} &= t_{prescal} + t_{iter} + t_{terminate} \\ &= [(1 + 2 + 2) + (\lceil 54/4 \rceil + 1) + 2] t_{cycle} = 22 \times 8\tau = 176\tau \end{aligned} \tag{20}$$

A radix-16 division on real operands with two overlapped radix-4 stages[16] has a cycle time $t_{cycle-conv} \approx 6.5\tau$ and a total time of

$$T_{RDIV} = (14 + 1 + 2) \times 6.5\tau \approx 111\tau \tag{21}$$

Therefore, $T_{CDIV}/T_{RDIV} \approx 1.6$

To get a rough comparison of the latency of the proposed scheme with a conventional complex division, we consider the complex division operation defined by (2) which uses 6 real divisions and 6 real multiplication operations in the floating-point format. There is also a comparison of absolute values. Assuming two floating-point units and a radix-16 digit-recurrence implementation of 54-bit division with the latency $T_{RDIV}$, a 4-cycle multiply, and 2 cycles for absolute value comparison, the estimated latency of this implementation, is

$$T_{S-CDIV} = 3T_{RDIV} + 6 \times t_{cycle-conv} \approx 372\tau \tag{22}$$

We ignored exponent processing and rounding delays. We conclude that the proposed scheme is at least two times faster than a hypothetical implementation based on formula (2).

The complex division in[10] uses a radix-2 on-line algorithm. It implements the selection function using selection constants and consequently, unlike the proposed algorithm which uses rounding, it is restricted to small radices. In the case of radix 2, the critical path in the recurrence and its cost are similar in both schemes.

Unlike a conventional implementation, the method with prescaling requires a table lookup as discussed earlier and two complex (rectangular) multiplications to perform scaling of the operands. As described above, these can be merged with the recurrence implementation. The need for a table lookup is the main cost disadvantage of the proposed method. To reduce the overall cost, instead of having two residual recurrence modules, a single two-stage pipelined module can be used at an expense of increasing the iteration time.

### Summary

We have discussed implementation of a digit-recurrence algorithm with prescaling of operands for complex division. The method are suitable for higher radix. The prescaling is more complicated than in the real case and the table size is the limiting factor for the choice of radix. The proposed algorithm is estimated to be at least twice as fast as a direct implementation of the formula used in software routines for complex division. Moreover, the method always allows easy faithful rounding while correct rounding can be performed at a reasonable cost.

## REFERENCES


1. M. D. Ercegovac and Muller, J.-M. Complex division with prescaling of operands *Proc. IEEE ASAP03*, 2003.
2. M. Igel, H.J. Koglin, and P. Schegner. New algorithms for earthfault distance protection in insulated and compensated networks. *ETEP*, 1(5), 1991.
3. A. Li, D.B. Sharp, and B.J. Forbes. Improving the high frequency content of the input signal in acoustic pulse reflectometry. In *Proc. of the International Symposium on Musical Acoustics*, pages 391–394, 2001.
4. S.R. Dicker et al. Cbm observations with the Jodrell Bank - iac interferometer at 33 Ghz. *Mon. Not. R. Astron. Soc.*, 00:1–12, 2000.
5. G. Vandersteen et al. Comparison of arithmetic functions with respect to Boolean circuits. In *58th ARFTG Conference Digest RF Measurements for a Wireless World*, 2001.
6. 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, 2002.
7. X. Li et al. Design, implementation and testing of extended and mixed precision BLAS. *ACM Transactions on Mathematical Software*, 28(2):152–205, 2002.
8. R.L. Smith. Algorithm 116: Complex division. *Communications of the ACM*, 5(8):435, 1962.
9. G.W. Stewart. A note on complex division. *ACM Transactions on Mathematical Software*, 11(3):238–241, 1985.
10. R.D. Mcilhenny. *Complex Number On-line Arithmetic for Reconfigurable Hardware: Algorithms, Implementations, and Applications*. PhD thesis, University of California at Los Angeles, 2002.
11. M. D. Ercegovac, Lang, T., and Montuschi, P. Very-high radix division with prescaling and rounding. *IEEE Transactions on Computers*, 43(8):909–918, 1994.
12. M. D. Ercegovac and T. Lang. *Division and Square Root: Digit-Recurrence Algorithms and Implementations*. Kluwer Academic Publishers, Boston, 1994.
13. 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.
14. M.J. Schulte and J.E. Stine. Approximating elementary functions with symmetric bipartite tables. *IEEE Transactions on Computers*, 48(8):842–847, Aug. 1999.
15. J. A. Pineiro, *Algorithms and Architectures for Elementary Function Computation*, PhD Dissertation, Department of Electronics and Computation, University of Santiago de Compostela, 2003.
16. M. D. Ercegovac and Lang, T. *Digital Arithmetic*. Morgan Kaufmann, 2004.