Variable Radix Real and Complex Digit-Recurrence Division

Miloš D. Ercegovac
Computer Science Department, 4732 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 digit-recurrence algorithm for division in real and complex number domains using a variable radix. The objective of the approach is to simplify the prescaling of the operands by using a suitable low radix, and switch to higher radices in the remaining iterations to reduce their number. The prescaling is used to allow a simple quotient-digit selection by rounding of the residual. We discuss the algorithm, its implementation, and estimate its time and cost characteristics with respect to fixed high-radix division algorithms.

1 Introduction

As shown by Oberman and Flynn [10], although division is a rather infrequent arithmetic operation, ignoring its implementation may result in significant performance degradation. Digit-recurrence algorithms are frequently used to implement division and square root of real (fixed-point or floating-point) operands in hardware [11, 3, 4]. They also have been generalized to complex division [7] and square-root [6]. For computing x/y (assuming $|x| \leq y$) the radix-r (real) digit-recurrence division algorithm uses the recurrence

$$w[j+1] = rw[j] - q_{j+1}y \qquad (1)$$

where $w[0] = x$, and the radix-r quotient digits q_j's are selected so that the residuals $w[j]$'s remain bounded. This ensures the convergence of the algorithm:

$$(0.q_1q_2\cdots q_n)_r = q_1r^{-1} + q_2r^{-2} + \cdots + q_nr^{-n} \rightarrow x/y.$$

The quotient-digit set is redundant which causes the quotient-digit selection intervals to overlap which, in turn, allows the use of (i) low-precision estimates of the residuals (i.e., the residual can be computed using a redundant adder) and, (ii) simple selection constants (for small radices). At each iteration, one new radix-r digit of the quotient is generated. Hence, to minimize the number of iterations, it is advantageous to use a high radix: for instance a radix-2^k algorithm generates k bits of quotient per iteration.

However, selecting the q_{j+1}'s becomes more difficult as the radix increases. The usual way is to perform the selection either by comparing the residual $w[j]$ to some selection constants, or through table look-up (with a few most significant bits of $w[j]$ and y as address bits). For radices higher than 2, these approaches also require the use of the divisor [4]. However, the number of selection constants increases with r (roughly linearly in the real case and quadratically in the complex case). The size of the lookup table also increases with r. In the real case, if the table takes the residuals $w[j]$ in carry-save or borrow-save form, the increase will roughly be quadratic. If a short adder is used to convert the most significant part of $w[j]$ to nonredundant representation, the table will be smaller, but the delay of the quotient digit selection will be increased by the delay of the adder. All this makes very-high radix digit recurrence division too difficult to implement using the selection function with constants.

To have a feasible selection function for very-high radix division, *prescaling* of operands is used. It consists in multiplying x and y by a constant M chosen so that first, My is close to 1, and second, the precision of M is short so that

Proceedings of the16th International Conference on Application-Specific Systems, Architecture and Processors (ASAP'05)
1063-6862/05 $20.00 © 2005 IEEE

the products Mx and My are computed exactly, and using a few additions only). Then the digit-recurrence is used to compute $(Mx)/(My)$. Since My is very close to 1, one can choose q_{j+1} by rounding $w[j]$ (or, merely, an approximation to $w[j]$, made up with a few most significant digits of $w[j]$) to the nearest integer, provided that some conditions are satisfied. Therefore, the quotient-digit selection has a delay proportional to the $log_2 r$ and it is independent of the divisor. The idea of scaling operands to make the quotient-digit selection independent of the divisor was proposed in [14] for a decimal computer. This scheme was extended by in [15] to an arbitrary radix. A general derivation and analysis of prescaling is given in [3].

Finding an adequate value for M is usually done through table look-up. Again, the size of the lookup table increases with the radix, making very-high radix division difficult to implement (this is even worse for complex division, where radices greater than or equal to 8 are not implementable with current technology [7]). One can make the table smaller using the bipartite method [12, 13]. Alternatively, the scaling constant can be computed in few steps, using a suitable approximation, such as a linear interpolation, as suggested by Ercegovac, Lang and Montuschi [5, 9] (the coefficients of the linear approximation that generates M are obtained using smaller tables). In this paper, we explore the idea of using a variable radix to simplify the prescaling, and exploit the idea of a higher radix in later steps to reduce the overall time.

Since M is an approximation to $1/y$, a solution to generate it would be to perform a few iterations of a low-radix division (we will call it the "preliminary division"), and then, once M is obtain to enough accuracy, to start the high-radix of Mx by My. Unfortunately, this would increase the delay of the computation of x/y (the "main division"). A solution we wish to analyse here is to try to somewhat overlap the preliminary and main division.

2 Division, real case

Assume we wish to compute x/y, with $1 \leq y < 2$. First, we perform δ radix-2 iterations for $1/y$ (adequate values of δ will be suggested later). This gives

$$M = 0.m_1 m_2 m_3 \cdots m_\delta \tag{2}$$

with $m_i \in \{-1, 0, 1\}$ and

$$\frac{1}{y} - 2^{-\delta} < M < \frac{1}{y} + 2^{-\delta}. \tag{3}$$

In parallel with the δ radix-2 iterations, we compute

$$\begin{aligned} x^* &= Mx \\ y^* &= My. \end{aligned}$$

These multiplications are overlapped with the division iterations: each time we get a new quotient bit m_i we accumulate $m_i y 2^{-i}$ and $m_i x 2^{-i}$. The accumulations can be done using a redundant adder (e.g., carry-save or borrow-save). with the y^* being converted on-the-fly to non redundant representation using the techniques introduced in [1]. However, this process produces only b most significant bits of the scaled divisor. Regarding the redundant scaled divisor, one alternative is to use a fast CPA to obtain the remaining bits of y^*. Another alternative is to use the scaled divisor in the redundant form. As discussed later, this adds another level of reduction stages in the rectangular multiplier. The scaled dividend can also be used in the redundant form. We will focus on this alternative. Note that it is also possible to use the on-line division method which handles the remaining digits of the scaled divisor as they become available.

Using (3) we derive

$$1 - 2^{-\delta+1} < y^* = My < 1 + 2^{-\delta+1}, \tag{4}$$

hence,

$$\left| \frac{1}{y^*} - 1 \right| < 2^{-\delta+2} \tag{5}$$

Assume that $\delta - 2$ is a multiple of k, say $\delta - 2 = k\Delta$. From (5), we deduce that there is a radix-2 signed-digit representation of $1/y^*$ that starts with the sequence

$$1.\underbrace{00000\cdots00}_{\delta-2 \text{ zeros}}$$

Hence, there is a radix-2^k maximally redundant signed-digit representation of $1/y^*$ that starts with the sequence

$$1.\underbrace{0000\cdots0}_{\Delta \text{ zeros}}.$$

Denote $r = 2^k$. We suggest to start radix-r division iterations for computing $1/y^*$, namely iterations of the form

$$W[i+1] = rW[i] - m'_{i+1} y^* \tag{6}$$

from step $i = \Delta$, with

$$m'_0 = 1, m'_1 = m'_2 = m'_3 = \cdots = m'_\Delta = 0$$

and

$$W[\Delta] = r^\Delta (1 - y^*).$$

In parallel with (6), we will perform (for $i \geq \Delta$)

$$Q[i+1] = Q[i] + m'_{i+1} r^{-i-1} x^*$$

with

$$Q[\Delta] = x^*$$

Proceedings of the16th International Conference on Application-Specific Systems, Architecture and Processors (ASAP'05)
1063-6862/05 $20.00 © 2005 IEEE

using a left-to-right multiplication with on-the-fly conversion which does not require a final adder to produce the most significant n bits of the result [1, 2].

Note that if (6) is a "valid" digit-recurrence division (that is, if the $W[j]$ are bounded), then

$$Q[n] \to \frac{x}{y},$$

since

$$Q[j] = x^*(1.000 \cdots m'_{\Delta+1} m'_{\Delta+2} m'_{\Delta+3} \cdots m'_j)$$

and

$$1.000 \cdots m'_{\Delta+1} m'_{\Delta+2} m'_{\Delta+3} \cdots = \frac{1}{y^*}$$

and

$$\frac{x}{y} = x^* \times \frac{1}{y^*}.$$

Now, since y^* is very close to 1, we can select digit m'_{i+1} by rounding $rW[i]$ to the nearest integer (this is standard in high-radix digit-recurrence division algorithms with prescaling [5, 3]). More precisely, from (4), we get

$$1 - \frac{r^{-\Delta}}{2} < y^* < 1 + \frac{r^{-\Delta}}{2}.$$

Hence, if $-r + 1 \le m'_{i+1} \le r - 1$ then

$$-(r-1)\frac{r^{-\Delta}}{2} < m'_{i+1} - m'_{i+1} y^* < (r-1)\frac{r^{-\Delta}}{2},$$

Since

$$rW[i] - m'_{i+1} y^* = \left(rW[i] - m'_{i+1}\right) + \left(m'_{i+1} - m'_{i+1} y^*\right),$$

we find

$$|W[i+1]| < \frac{1}{2} + (r-1)\frac{r^{-\Delta}}{2}. \tag{7}$$

Assuring that $rW[i]$ rounded to the nearest integer is always between $-r + 1$ and $r - 1$ requires

$$|W[i]| \le 1 - \frac{1}{2r}. \tag{8}$$

Therefore, based on (7) and (8), the proposed scheme will converge (i.e., the $W[j]$'s will be bounded) provided that

$$\frac{1}{2} + (r-1)\frac{r^{-\Delta}}{2} \le 1 - \frac{1}{2r},$$

i.e.,

$$(r-1)r^{-\Delta} + \frac{1}{r} \le 1. \tag{9}$$

A quick look on (9) reveals that all values $\Delta \ge 1$ are a solution. Taking $\Delta = 1$, from $\delta - 2 = k\Delta$, we deduce that we can start radix $r = 2^k$ iterations after $k + 2$ radix-2 iterations.

Moreover, we can continue to increase the radix. For instance, after iteration number δ_1 of the radix $r_1 = 2^{k_1}$ division (that is, when we have obtained quotient digit number δ_1 of $1/y^*$), defining $M' = 0.m'_1 m'_2 \cdots m'_{\delta_1}$ and assuming we have continued to compute $y \times 0.m'_1 m'_2 \cdots m'_{\delta_1}$ by successive accumulations, we have

$$\frac{1}{y} - 2^{-k_1\delta_1} < M' < \frac{1}{y} + 2^{-k_1\delta_1}. \tag{10}$$

This last equation is the same as equation (3), with δ replaced by $k_1\delta_1$. This means that we can start a radix 2^{k_2} iteration from step 1, where $k_2 = k_1\delta_1 - 2$.

For instance, after 4 radix-2 iterations, we can start the radix 4 iterations, after 4 radix 4 iterations, we can start radix 16 iterations, and after 4 radix 16 iterations, we can start radix 256 iterations. We now consider an implementation of variable-radix division for 53-bit operands. An initial prescaling of the divisor y into $Y0 \in (7/8, 9/8)$ uses a separate [3:2] adder with inputs selected from the set $\{y, y/2, y/4, y/8, -y/8\}$ based on the three MS bits of the divisor y. The radix 4 digits of the short reciprocal $1/Y0$ are in the set $\{-2, -1, 0, 1, 2\}$, selected by rounding of the shifted residual. We perform four iterations in radix 4. In parallel, we compute a first scaled dividend $X0$ using a left-to-right multiplier, producing the product in a redundant (carry-save) form. Next we perform four radix 4 iterations, producing 4 radix-4 digits (8 bits) of the scaled reciprocal y^*. These digits are used in parallel to obtain scaled dividend x^* and the MS digits of the quotient. Next 4 iterations are performed in radix-16, producing next 4 radix-16 digits of the reciprocal, and, in parallel, a scaled dividend, and additional digits of the quotient. Finally, we perform 4 iterations in radix 256, producing remaining digits in a similar manner. A general scheme is indicated in Figure 1.

We now briefly describe the main modules shown in Figure 1. The RR Module produces five radix-4 digits by rounding and recoding (parallel) of the corresponding number of leading WS and WC bits. The outputs of the parallel radix-4 recoder are selectively enabled according to the radix of the iteration. So, during the radix-4 iterations only two MS recoded digits are used; three during the radix-16 iterations, and five during the radix-256 iterations. The central module is a rectangular multiplier-accumulator ROMAC. We assume that the multiplicand (i.e., the progressively scaled divisor y^*) is the redundant carry-save form. The design of operand reduction array is based on the design of a fully redundant multiplier [8]. The ROMAC is designed for radix-256 and it is used for radix 4 and radix 16 iterations to keep the cycle time uniform. The scaling left-to-right (LR) multipliers in the scheme do not use CPA adders so ROMAC accommodates redundant multiplicands. The quotient produced by another LR multiplier is converted to its conventional form using on-the-fly conversion approach [1].

Proceedings of the16th International Conference on Application-Specific Systems, Architecture and Processors (ASAP'05)
1063-6862/05 $20.00 © 2005 IEEE

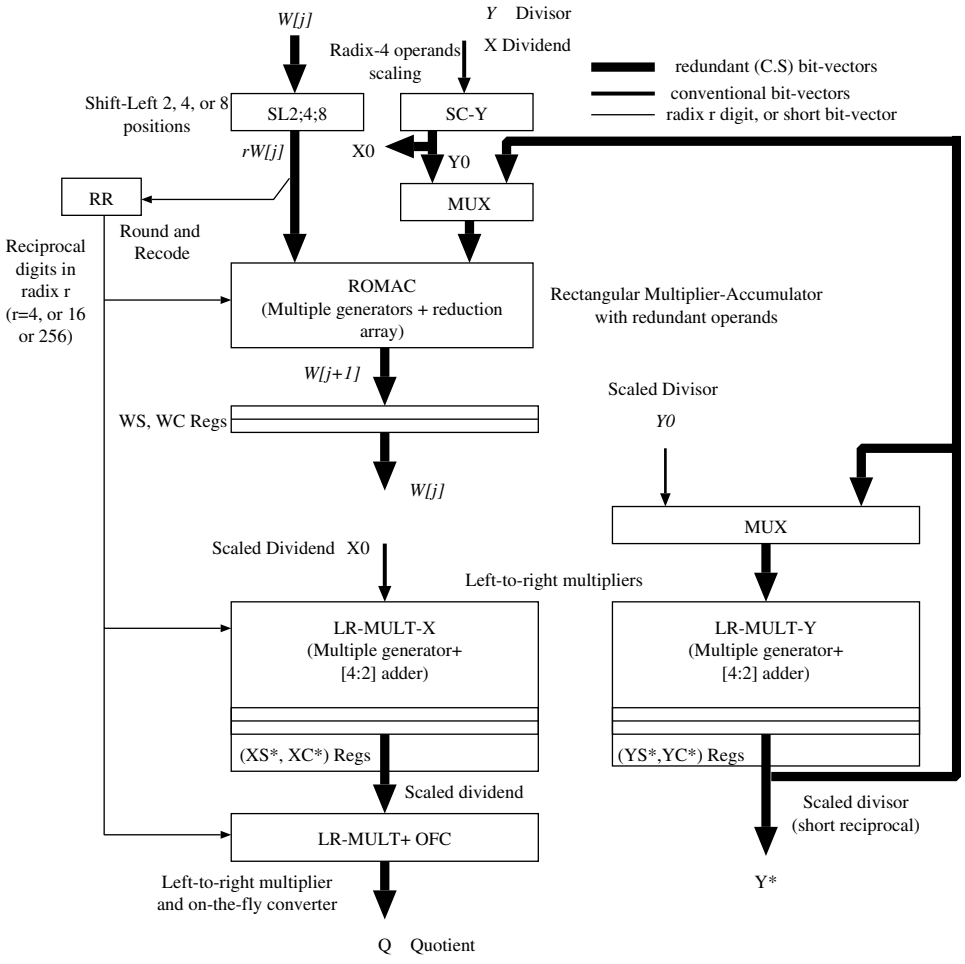

Figure 1. *Variable-Radix 4-16-256 Division Scheme.*

Proceedings of the16th International Conference on Application-Specific Systems, Architecture and Processors (ASAP'05)
1063-6862/05 $20.00 © 2005 IEEE

Rough estimates of delay and area are based on using the delay and area of a full-adder t_{FA} and A_{FA}, respectively. We use $n = 53$. We estimate that the radix-256 cycle time is

$$t_{256} = t_{RR} + t_{MG} + t_{RA} + t_{SL} + t_{reg}$$
$$\approx (1.5 + 0.5 + 3.5 + 0.5 + 1)t_{FA} = 7t_{FA}$$

The number of iterations (cycles) for the variable-radix scheme is

$$N_{4-16-256}$$
$$= prescale + r4cycles + r16cycles + r256cycles$$
$$= 1 + 4 + 4 + 4 = 13$$

The latency of the proposed scheme is $T_{4-16-256} = 13 \times 7t_{FA} = 91t_{FA}$. This is not the shortest latency that one can achieve. By decoupling radix-4, radix-16, and radix-256 iterations, the latency could be reduced to $1 + 4 \times (4 + 5 + 7) = 65t_{FA}$ which is similar to the latency of a radix-256 divider which is estimated to have a cycle time of $7t_{FA}$ and it takes 2 cycles for operands scaling and 7 cycles for computation of the quotient, resulting in a total latency of $63t_{FA}$.

We now estimate the cost of the proposed scheme and compare it with a cost of a radix-256 divider roughly assuming internal precision of 64 bits. The cost of the proposed scheme, using area of a full-adder A_{FA} as unit, is shown in Table 1.

Table 1. Proposed scheme

Module	Area [A_{FA}]
RR	70
SL2;4;8	50
2 2-1 MUXES	50
ROMAC	700
3 LR-MULT and OFC	765
MISC	50
TOTAL:	1750

Using a similar analysis of the cost, we estimate that a radix-256 divider has the costs shown in Table 2.

These rough cost estimates indicate a slight advantage of the proposed scheme.

3 Division, complex case

Throughout this section, 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 $\max\{|\Re(z)|, |\Im(z)|\}$.

In [7], we adapted the radix-r digit-recurrence division algorithm to complex division. The iterations for computing

Table 2. Radix-256 scheme

Module	Area [A_{FA}]
Tables and multiplier	700
RR	70
2-1 MUXES	50
Fast CPA	400
MAC	600
OFC	100
MISC	50
TOTAL:	1880

x/y are of the form

$$w[j+1] = rw[j] - q_{j+1}y \tag{11}$$

with $q_{j+1} = q^{\mathcal{R}}_{j+1} + iq^{\mathcal{I}}_{j+1}$, where $q^{\mathcal{R}}_{j+1}$ and $q^{\mathcal{I}}_{j+1}$ belong to the digit set $\mathcal{S}$ of a redundant radix-r representation (a typical example is the *maximally redundant set* $\mathcal{S} = \{-r+1, -r+2, \ldots, r-2, r-1\}$). We assume $1 \le ||y||_\infty < 2$. To make the selection of the quotient digits simple we need to prescale the input operands. In the complex case, the prescaling step, if implemented using tables, requires much larger tables than in the real case. In practice, this limits our complex division method to radices less than 8. Hence it is of interest to adapt the variable-radix method presented in the previous section to the complex case.

As in the real case, we can start radix-2 iterations[1], so that at step δ of these iterations, we get

$$M = 0.m_1 m_2 m_3 \cdots m_\delta \tag{12}$$

with $\Re(m_i), \Im(m_i) \in \{-1, 0, 1\}$ and

$$\begin{aligned} \Re(\tfrac{1}{y}) - 2^{-\delta} < \Re(M) < \Re(\tfrac{1}{y}) + 2^{-\delta} \\ \Im(\tfrac{1}{y}) - 2^{-\delta} < \Im(M) < \Im(\tfrac{1}{y}) + 2^{-\delta} \end{aligned} \tag{13}$$

In parallel with the δ radix-2 iterations, we compute

$$\begin{aligned} x^* &= Mx \\ y^* &= My. \end{aligned}$$

The major difference with what we did for the real case is that now, instead of (4), we have:

$$||1 - My||_\infty < 2^{-\delta+2}$$

Hence, compared to the real case, we have $(-\delta + 2)$ instead of $(-\delta + 1)$ which comes from

$$||1 - My||_\infty < 2||y||_\infty ||1/y - M||_\infty$$

[1] They will be prescaled: this is necessary in the complex case. Prescaling in radix 2 is done easily [7].

Proceedings of the16th International Conference on Application-Specific Systems, Architecture and Processors (ASAP'05)
1063-6862/05 $20.00 © 2005 IEEE

Assume $r = 2^k$, and define Δ as $\delta - 3 = k\Delta$ (which is not the same definition as in the real case). We wish to know when (i.e., at which step Δ) we can start the radix-r iterations. Very similar calculations to the ones performed in the real case give

$$||m'_{i+1} - m'_{i+1}y||_\infty \leq 2(r-1)\frac{r^{-\Delta}}{2}, \qquad (14)$$

which gives

$$||W[i+1]||_\infty < \frac{1}{2} + 2(r-1)\frac{r^{-\Delta}}{2} \qquad (15)$$

instead of (7). Assuring that the real and imaginary parts of $rW[i]$ rounded to the nearest integer are always between $-r+1$ and $r-1$ requires

$$||W[i]||_\infty \leq 1 - \frac{1}{2r}. \qquad (16)$$

so that we get, instead of (9):

$$2(r-1)r^{-\Delta} + \frac{1}{r} \leq 1. \qquad (17)$$

A quick look at (17) reveals that all values $\Delta \geq 2$ are a solution. Taking $\Delta = 2$, from $\delta - 3 = k\Delta$, we deduce that we can start radix $r = 2^k$ iterations after $2k+3$ radix-2 iterations. We can continue to increase the radix as in the real case.

Consequently, we can avoid the demand for rather large prescaling tables by starting with a small radix and gradually increasing it as we did in real division.

Summary

We have considered a variable radix approach to division. To have a practical arbitrary radix division, the selection of the quotient digits has to be done via rounding of the residual. This requires scaling of the operands. The scaling is typically accomplished using tables for the coefficients of a suitable linear or quadratic approximation of the short reciprocal. Our objective is to eliminate these tables by using gradually increasing radix. Such an approach would be of particular importance in implementing division in the complex domain. We presented the derivation of the method and preliminary results regarding an implementation. The latency and cost comparisons with these of a radix-256 divider indicate a good potential of the proposed approach. Detailed designs remain to be developed.

References

[1] M. D. Ercegovac and T. Lang. On-the-fly conversion of redundant into conventional representations. *IEEE Transactions on Computers*, C-36(7), July 1987. Reprinted in E. E. Swartzlander, *Computer Arithmetic*, Vol. 2, IEEE Computer Society Press Tutorial, Los Alamitos, CA, 1990.

[2] M. D. Ercegovac and T. Lang. Fast multiplication without carry-propagate addition. *IEEE Transactions on Computers*, 39(11):1385–1390, November 1990.

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

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

[5] M. D. Ercegovac, T. Lang, and P. Montuschi. Very-high radix division with prescaling and selection by rounding. *IEEE Transactions on Computers*, 43(8):909–918, August 1994.

[6] M. D. Ercegovac and J.-M. Muller. Complex square root with operand prescaling. In *Proc. 15th IEEE International Conference on Application-Specific Systems, Architectures and Processors, Galveston, Texas*. IEEE Computer Society Press, September 2004.

[7] M. D. Ercegovac and J.-M. Muller. Complex division with prescaling of the operands. In E. Deprettere, S. Bhattacharyya, J. Cavallaro, and A. Darte, editors, *Proceedings of ASAP'03: 14th IEEE Conference on Application-Specific Systems, Architectures and Processors*, pages 304–314. IEEE Computer Society, June 2003.

[8] M. I. Ferguson, and M. D. Ercegovac. A multiplier with redundant operands *Proc. 33rd Asilomar Conference on Signals, Systems, and Computers*, pages 1322 - 1326 vol.2, 1999.

[9] P. Montuschi and T. Lang. Boosting very-high radix division with prescaling and selection by rounding. *IEEE Transactions on Computers*, 50(1):13–27, jan 2005.

[10] S. F. Oberman and M. J. Flynn. Design issues in division and other floating-point operations. *IEEE Transactions on Computers*, 46(2):154–161, February 1997.

[11] J. E. Robertson. A new class of digital division methods. *IRE Transactions on Electronic Computers*, EC-7:218–222, 1958. Reprinted in E. E. Swartzlander, *Computer Arithmetic*, Vol. 1, IEEE Computer Society Press Tutorial, Los Alamitos, CA, 1990.

[12] D. Das Sarma and D. W. Matula. Faithful bipartite ROM reciprocal tables. In Knowles and McAllister, editors, *Proceedings of the 12th IEEE Symposium on Computer Arithmetic (ARITH-12)*, pages 17–28. IEEE Computer Society Press, June 1995.

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

[14] A. Svoboda. An algorithm for division. *Inf. Process. Mach.*, 9:25–32, 1963. Reprinted in E. E. Swartzlander, *Computer Arithmetic*, Vol. 1, IEEE Computer Society Press Tutorial, Los Alamitos, CA, 1990.

[15] C. Tung. A division algorithm for signed-digit arithmetic. *IEEE Transactions on Computers*, C-17, 1968.

Proceedings of the16th International Conference on Application-Specific Systems, Architecture and Processors (ASAP'05)
1063-6862/05 $20.00 © 2005 IEEE