The M-log-Fraction Transform (MFT) for Computer Arithmetic

Oskar Mencer, Michael J. Flynn, and Martin Morf

Technical Report : CSL-TR-99-784

December 1999

This research is supported by DARPA Grant No. DABT63-96-C-0106.

The M-log-Fraction Transform (MFT) for Computer Arithmetic

by

Oskar Mencer, Michael J. Flynn, and Martin Morf

Technical Report : CSL-TR-99-784

December 1999

Computer Systems Laboratory

Department of Electrical Engineering and Computer Science

Stanford University

William Gates Computer Science Building, 4A-408

Stanford, California 94305-9040

Email: pubs@shasta.stanford.edu

Abstract

State-of-the-art continued fraction(CF) arithmetic enables us to compute rational functions so that input and output values are represented as simple continued fractions. The main problem of previous work is the conversion between simple continued fractions and binary numbers.

The M-log-Fraction Transform(MFT), introduced in this work, enables us to instantly convert between binary numbers and M-log-Fractions. Conversion is related to the distance between the '1's of the binary number. Applying M-log-Fractions to continued fraction arithmetic algorithms reduces the complexity of the CF algorithm to shift-and-add structures, and more specifically, digit-serial arithmetic algorithms for a family of rational functions.

We show two applications of the MFT:

(1) a high radix rational arithmetic unit computing (ax+b)/(cx+d) in a shift-and-add structure.

(2) the evaluation of rational approximations (or continued fraction approximations) in a multiplication-based structure.

In (1) we obtain algebraic formulations of the entire computation, including the next-digit-selection function. For high radix operation, we can therefore partition the selection table into arithmetic blocks, making high radix implementations feasible.

(2) overlaps the final division of a rational approximation with the multiply-add iterations.

The MFT bridges the gap between continued fractions and the binary number representation, enabling the design of a new class of efficient rational arithmetic units and efficient evaluation of rational approximations.

Key Words and Phrases: Computer Arithmetic, Continued Fractions, Rational Arithmetic

Contents

List of Figures

1 Introduction

Rational approximation offers efficient evaluation of elementary functions (see [12]). We investigate the design of rational arithmetic units for VLSI based on the M-log-Fraction Transform (MFT), which we introduce in this paper. The M-log-Fraction connects binary numbers and continued fractions. In order to explain the theory behind the M-log-Fraction, and the relationship of the M-log-Fraction to the distances between the '1's of a binary number, we require a basic understanding of continued fractions and the state-of-the-art in continued fraction arithmetic algorithms.

1.1 Introduction to Continued Fractions

We first define some continued fraction(CF) forms. *Finite continued fractions* are rational numbers that are constructed as follows: for $a_i, b_i \in \mathcal{R}$

$$\frac{A_n}{B_n} = a_0 + \frac{b_1}{a_1 + \frac{b_2}{a_2 + \cdots \frac{b_n}{a_n}}} = a_0 + \frac{b_1|}{|a_1} + \frac{b_2|}{|a_2} + \cdots + \frac{b_n}{a_n} \tag{1}$$

Simple continued fractions form a special case with partial quotients $b_i = 1$. We use the series notation $[a_0; a_1, \ldots a_n]$ to denote simple continued fractions. The core transform of the simple continued fraction form is

$$F(x) = a_i + \frac{1}{x} \tag{2}$$

Regular continued fractions are simple continued fractions with all $a_i \in \mathcal{N}^+$, except $a_0 \in \mathcal{N}$. *Logarithmic continued fractions* lead to a logarithmic representation of partial quotients. Early results on continued fraction algorithms are presented in [7]. Trivedi[8] investigates logarithmic continued fractions, and Kornerup[9] uses $\{-2, -\frac{1}{2}, 0, \frac{1}{2}, 2\}$ as the digit set for partial quotients of simple continued fractions.

A finite continued fraction with i partial quotients can always be transformed into a ratio $\frac{A_i}{B_i}$ with:

$$\begin{aligned} A_i &= a_i A_{i-1} + b_i A_{i-2} && (3) \\ B_i &= a_i B_{i-1} + b_i B_{i-2} && (4) \end{aligned}$$

where $\frac{A_{i-1}}{B_{i-1}}$ corresponds to the value of the same continued fraction without the i^{th} partial quotient. Initial conditions are $A_0 = a_0$, $B_0 = 1$, $A_{-1} = 1$, and $B_{-1} = 0$.

Every number representation has its natural set of operations. Residue numbers favor addition, while logarithmic numbers favor multiplication. Continued fractions favor $f = 1/x$, which is computed by $[0; a_0, a_1, \ldots]$ if $a_0 \neq 0$, and $[a_1; a_2, \ldots]$ if $a_0 = 0$. Negation is achieved by negating all partial quotients. A simple continued fraction multiplied by a constant c becomes:

$$[0; \frac{a_0}{c}, c \cdot a_1, \frac{a_2}{c}, c \cdot a_3, \frac{a_4}{c}, c \cdot a_5, \ldots] \tag{5}$$

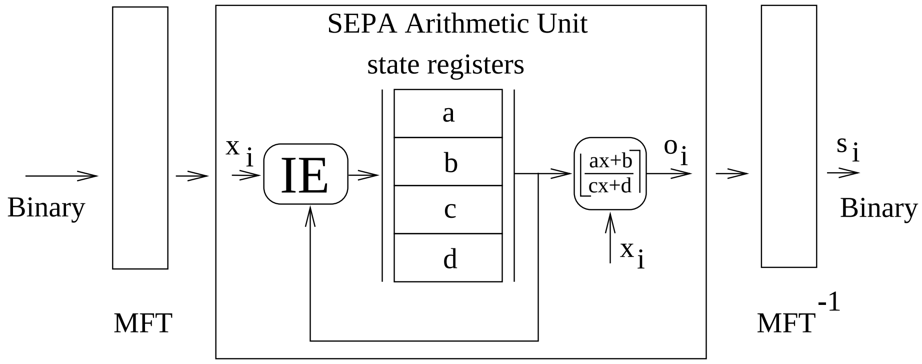


Figure 1: *The figure shows the state machine for the iteration equations (IE) of the SEPA algorithm. The rounding operation for the output digit ($\lfloor\rceil$) stands for choosing an output digit close to the exact value. The MFT is introduced in section 2.*

In summary, continued fractions are equivalent to rational approximations, converge faster than polynomial approximations with a much larger interval of convergence. For a thorough introduction to continued fractions see [1].

1.2 The SEPA Algorithm

The basic algorithms for continued fraction arithmetic are based on the work of Hurwitz[2], Hall [3], and later Gosper[4] and Raney[5]. The Semi-Exact Positional Algebraic (SEPA) algorithm for regular continued fractions is analyzed in [11]. In summary, the SEPA algorithm is a finite implementation of Vuillemin's positional algebraic algorithm[17] for exact arithmetic. In the simplest case we compute a linear fractional transformation $o = [o_1, o_2, o_3, \ldots] = T(x)$ with $x = [x_1, x_2, x_3, \ldots]$.

$$T(x) = \frac{A \cdot x + B}{C \cdot x + D} = \frac{a_0 \cdot x + b_0}{c_0 \cdot x + d_0} = T_0(x) \tag{6}$$

with a_0, b_0, c_0, d_0 as initial state. The iterative algorithm consumes one input digit x_i and produces one output digit o_i at each iteration step, based on the simple continued fraction core transform $F(x)$ from equation 2. The transformation of the function T_i at each iteration is:

$$T_{i+1}(x) = \frac{1}{T_i(x_i + \frac{1}{x}) - o_i} = F(x_i) \circ T_i(x) \circ F^{-1}(o_i) \tag{7}$$

The *homographic* function $T(x) = \frac{ax+b}{cx+d}$ is invariant over the required transformation and can therefore be computed by using a finite state machine and the following iteration (state transition) equations for state variables a, b, c, d shown in figure 1:

$$\begin{aligned} a_{i+1} &= \mathtt{c_i x_i + d_i} & b_{i+1} &= c_i \\ c_{i+1} &= a_i x_i + b_i - o_i(c_i x_i + d_i) & d_{i+1} &= a_i - o_i c_i \end{aligned} \tag{8}$$

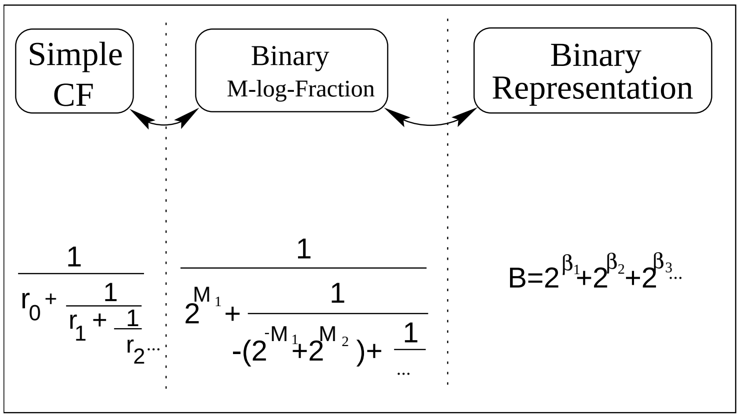


Figure 2: *The block-diagram above shows the binary M-log-fraction. The binary M-log-fraction serves as a link between simple continued fraction expansions and binary numbers.*

The algorithm is self-correcting, i.e. we can choose an arbitrarily approximate o_i close to $\hat{o}_i = \frac{a_i x_i + b_i}{c_i x_i + d_i}$ at each iteration, and the algorithm still converges to the right result. Ideally, we choose o_i to make the final continued fraction simple to evaluate to a binary number, which is the key observation leading to the MFT proposed in this paper.

In general, a SEPA algorithm can be developed for many rational functions, i.e. functions $T(x)$ which preserve their form in transformation 7. For more details on how to design higher order SEPA algorithms see [17] and [11].

2 The M-log-Fraction Transformation (MFT)

One major difficulty with continued fractions as a tool for computer arithmetic is the amount of effort needed to convert binary numbers to continued fractions. We introduce the *M-log-fraction* – a special case of a logarithmic simple continued fraction – to bridge the gap between simple continued fraction expansions and binary numbers (see figure 2):

Theorem 1 *Binary M-log-Fraction: A binary number B with p binary digits, and n '1's is equivalent to a simple continued fraction with n partial quotients called the M-log-Fraction:*

$$B = b_1 b_2 b_3 \cdots b_p = 2^{\beta_1} + 2^{\beta_2} + 2^{\beta_3} + 2^{\beta_4} + 2^{\beta_5} + \ldots + 2^{\beta_n} \tag{9}$$

$$\equiv [0; 2^{M_1}, -(2^{-M_1} + 2^{M_2}), (2^{-M_2} + 2^{M_3}), -(2^{-M_3} + 2^{M_4}), (2^{-M_4} + 2^{M_5}) \ldots \pm (2^{-M_{n-1}} + 2^{M_n})] \tag{10}$$

$$\equiv < M_1, M_2, M_3, M_4, M_5, \ldots, M_n > \tag{11}$$

where M_i are related to α_i, the distances between the '1's of the binary number B, as follows:

$$M_1 = \alpha_1 = -\beta_1, \quad M_2 = \alpha_2 - M_1, \quad M_i = \alpha_i - M_{i-1} \tag{12}$$

For a proof of theorem 1 see Appendix A. We refer to the encoding of binary numbers with a sequence of M_is as shown above as *M-log-Fraction Transformation (MFT)*. In fact, the first digit M_1 is the integer part of the logarithmic representation of B. Thus, the MFT is a variant of the logarithmic number system[16] with the advantage that conversion just requires counting the distance between the '1's.

We now have a direct correspondence between binary numbers and the MFT space. Applying the M-log-Fraction to the input and output of the SEPA algorithm described above leads to the first practical rational arithmetic units. However, in it's simplest form, the MFT creates difficulties for the selection function of the SEPA output digit. The sign of each digit is fixed. As a consequence we would have to choose each ouput digit in a way that would force the right sign on the next output digit. In order to avoid this difficulty we introduce the signed-digit M-log-Fraction. The signed-digit M-log-Fraction shown below connects signed-digit binary numbers (e.g. digits $\{-1, 1\}$) with the MFT.

Corollary 2 *Signed-Digit Binary M-log-Fraction: A binary number B_R with n digits, and $s_i \in \{+1, -1\}$ is equivalent to a simple continued fraction with n partial quotients as follows:*

$$B_R = s_1 2^{\beta_1} + s_2 2^{\beta_2} + s_3 2^{\beta_3} + s_4 2^{\beta_4} + \ldots + s_n 2^{\beta_n}$$
$$\equiv [0; s_1 2^{M_1}, -(s_1 2^{-M_1} + s_2 2^{M_2}), (s_2 2^{-M_2} + s_3 2^{M_3}), -(s_3 2^{-M_3} + s_4 2^{M_4}), \ldots \pm (s_{n-1} 2^{-M_{n-1}} + s_n 2^{M_n})]$$

For faster algorithms, the digits s_i are extended to $s_i \in \{-(r-1), \ldots, r-1\}$ (higher radix-r). Given the signed-digit MFT and the SEPA algorithm we show some hardware implementations of rational arithmetic units.

3 Algorithm Class 1: A Rational Arithmetic Unit

The signed-digit binary M-log-Fraction combined with the SEPA algorithm explained in the Introduction above results in a simple next-digit selection function(see figure 1). For simplicity reasons we show the case $T_1 = \frac{\mathtt{ax+b}}{\mathtt{cx+d}}$ with $\mathtt{a} = \mathtt{0}, \mathtt{c} = \mathtt{0}$. The SEPA algorithm simplifies to a division algorithm for $\frac{\mathtt{b}}{\mathtt{d}}$. As there are no input digits x_i, the transformation $T_{i+1}(x) = \frac{1}{T_i(x) - o_i}$ produces iteration equations:

$$d_{-1} = \mathtt{b} \qquad d_0 = \mathtt{d} \qquad d_{i+1} = d_{i-1} - o_i d_i \tag{13}$$

with

$$o_1 = s_1 2^{M_1} \qquad o_i = (-1)^{i-1}(s_{i-1} 2^{-M_{i-1}} + s_i 2^{M_i}) \tag{14}$$

selected according to corollary 2.

In fact, the iterations are reduced to shift-and-add operations. We fix the series of shifts $M = \{1, 0, 1, 0, 1, 0, \ldots\}$. Fixing the shift-sequence (Ms) results in regular hardware. By unrolling the iterations, the shifts can be hardwired and we obtain a pipeline of adders. A comparison of the resulting structure to state-of-the-art SRT-division(e.g. [14]) is shown in figure 4.

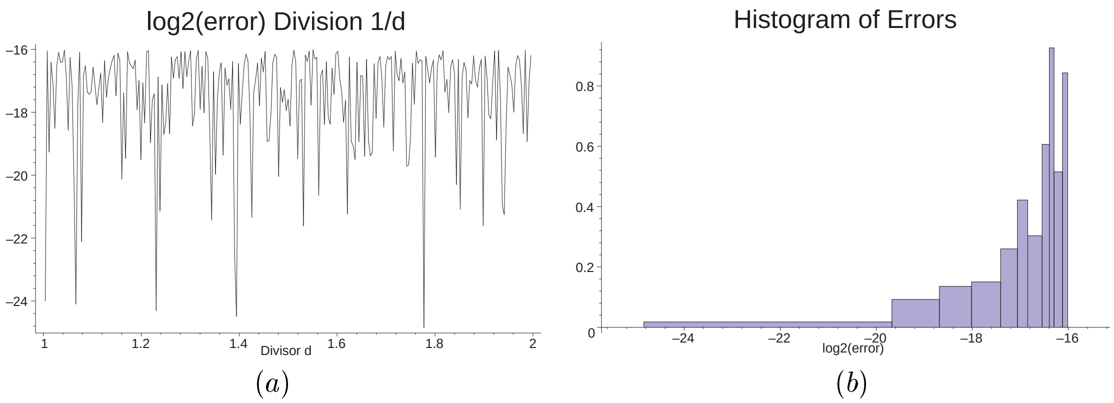


Figure 3: *The graphs plot the error for computing* $1/\mathtt{d}$ *after 16 iterations with 16-bit precision at each iteration, not including exact results. (a) shows* $\log_2(\mathtt{error})$ *for divisor* $\mathtt{d}$ *in the range of* $[1,2[$*. (b) shows the distribution of error values with a histogram.*

Next, we compute the signed-digits $s_i = \{-1, 1\}$ as follows:

$$s_i = sign\left((-1)^{i-1} \cdot d_{i-1} - s_{i-1} \cdot 2^{-M_{i-1}} \cdot d_i\right) \cdot sign(d_i) \tag{15}$$

Equation 15 enables us to select the next signed digit with little effort. The self-correcting feature of the SEPA algorithm guarantee convergence to the right result. Using a redundant digit set $\{+1, 0, -1\}$ gives us some flexibility. As a consequence, a real-world radix-2 SRT divider has a delay of 1 CSA plus a few gates to select the next digit. A similar approach can get rid of the CPAs and tables for the MFT-based unit.

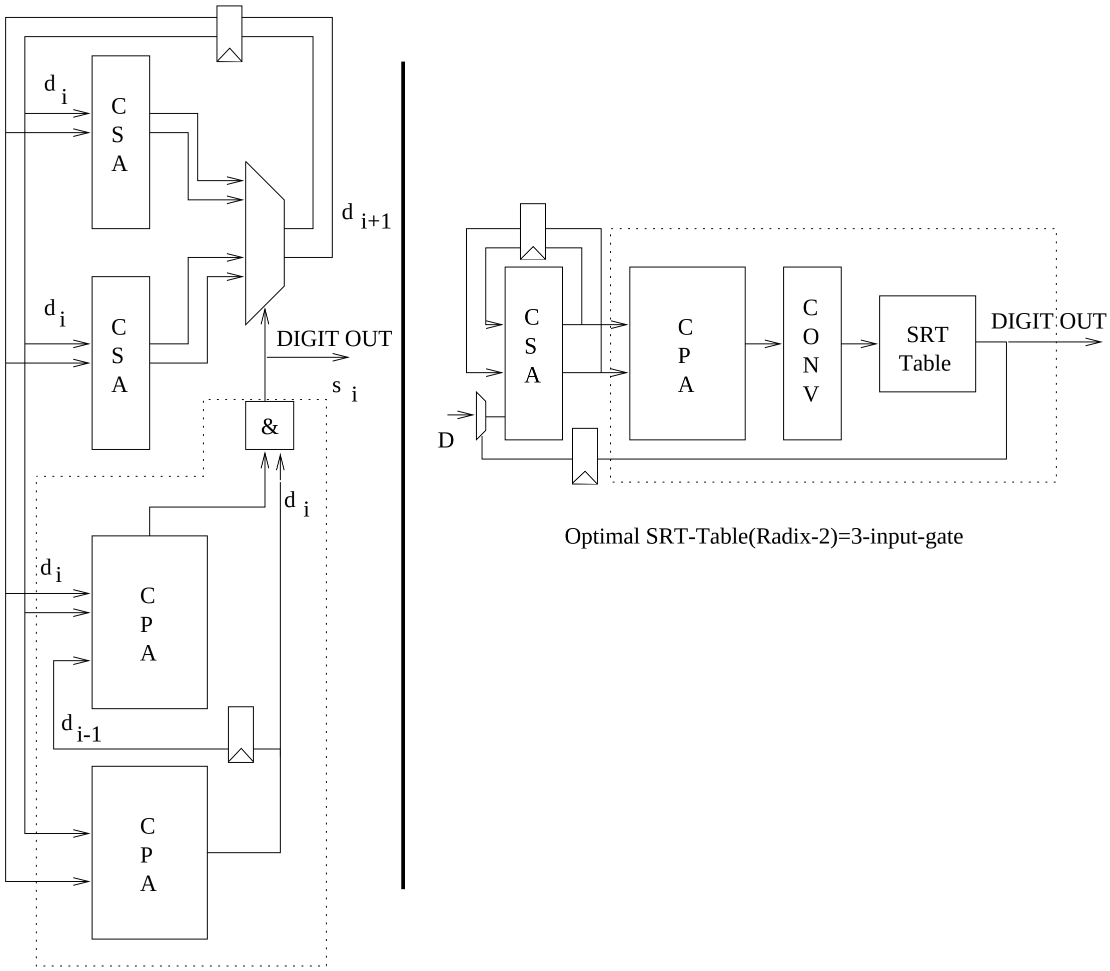
Figure 3 shows simulation results for the simple case $\mathtt{b} = \mathtt{1}, \frac{\mathtt{b}}{\mathtt{d}} = \frac{\mathtt{1}}{\mathtt{d}}$. We observe the interval $[1, 2[$ that is of interest for floating point division. The figures show results for a fixed algorithm with 16 iterations, 16-bit accurate iteration computations, targeting 16-bit results.

The proposed algorithm is similar to SRT division (Digit-Recurrence Algorithms [18]). The major difference lies in the digit selection function. The derivation of the proposed algorithm from continued fractions leads to a simpler and well defined selection function based solely on sign comparisons in case of radix-2.

3.1 Higher Radix Rational Arithmetic

The radix-r algorithm uses a signed digit-set, e.g. $s = \{-(r-1), \ldots, r-1\}$. Higher radix algorithms reduce the number of iterations at the cost of additional area and, possibly, a longer delay per iteration. Current state-of-the-art in SRT-division does not scale to efficient dividers beyond radix-8. The MFT enables us to design relatively simple higher radix algorithms based on an algebraic formulation of the selection functions, as shown for radix-2 in the previous sections.

As before, we first obtain an M-log-Fraction for the desired radix, and then design the iterative algorithm based on the SEPA algorithm and the signed-digit radix-r M-log-



(a) MFT-based Radix-2 Divider (b) SRT-Table-based Radix-2 Divider

Figure 4: *The figure shows (a) a regular, pipelined radix-2 divider based on the MFT, and (b) a general SRT-Table divider based on results from [14]. Broken lines en-capsule logic that can be collapsed to a few gates if we use a redundant digit representation.*

Fraction for the result. A radix-r number(rrn) results in the following signed-digit radix-r M-log-Fraction:

$$
\begin{aligned}
rrn &= 1 + s_1 r^{-1} + s_2 r^{-2} + s_3 r^{-3} + s_4 r^{-4} + \cdots = && (16)\\
&= [1; Q_1, Q_2, Q_3, Q_4, \ldots] && (17)\\
Q_1 &= \frac{r}{s_1} \qquad p_1 = s_1^{-1} \qquad \overline{p_1} \hat{=} \frac{1}{p_1} = s_1 && (18)\\
Q_i &= (-1)^{i-1} \cdot \left(p_i + \frac{1}{r \cdot p_{i-1}}\right) \cdot q_i && (19)\\
p_i &= \overline{p_{i-1}} \frac{s_{i-2}}{s_{i-1}} \qquad \overline{p_i} \hat{=} \frac{1}{p_i} = p_{i-1} \frac{s_{i-1}}{s_{i-2}} && (20)\\
q_i &= \begin{cases} r & \mathtt{i} = \mathtt{odd} \\ 1 & \mathtt{i} = \mathtt{even} \end{cases} && (21)
\end{aligned}
$$

We obtain a next digit selection function for the SEPA algorithm by choosing the output digit in the form of a digit of the radix-r M-log-Fraction as follows:

$$o_i = Q_i \sim \frac{a_i x_i + b_i}{c_i x_i + d_i} \tag{22}$$

Combining equations 22,19 we obtain the next digit s_i as a function of:

$$s_i = f(a_i, b_i, c_i, d_i, Q_i) \tag{23}$$

The general development of a next digit equation for the SEPA algorithm can be used to implement a digit-serial algorithm for a rational arithmetic unit capable of computing many rational functions.

We use the simplest case, division of two numbers, to show the details of an actual radix-r algorithm.

Radix-r Division

In case of radix-r division the function from equation 23 becomes:

$$s_i \sim \frac{X}{Y} = \begin{cases} \frac{d_i s_{i-1} r}{-p_i d_{i-1} r - d_i} & i = \mathtt{even} \\ \frac{d_i s_{i-1} r}{p_i d_{i-1} - d_i} & i = \mathtt{odd} \end{cases} \tag{24}$$

with p_i computed by equation 20. Figure 5 shows the structure of an implementation of the proposed algorithm for radix-r division units. Tables $T1$ and $T2$ use the most significant bits of X and Y to choose the next output digit $s_i \sim \frac{X}{Y}$. With increasing radix-r we require more levels of CSAs and increased precision in tables $T1$ and $T2$. However, as in the case of radix-2, a redundant digit representation can enable us to reduce the complexity of choosing the next digit s_i to a few gates.

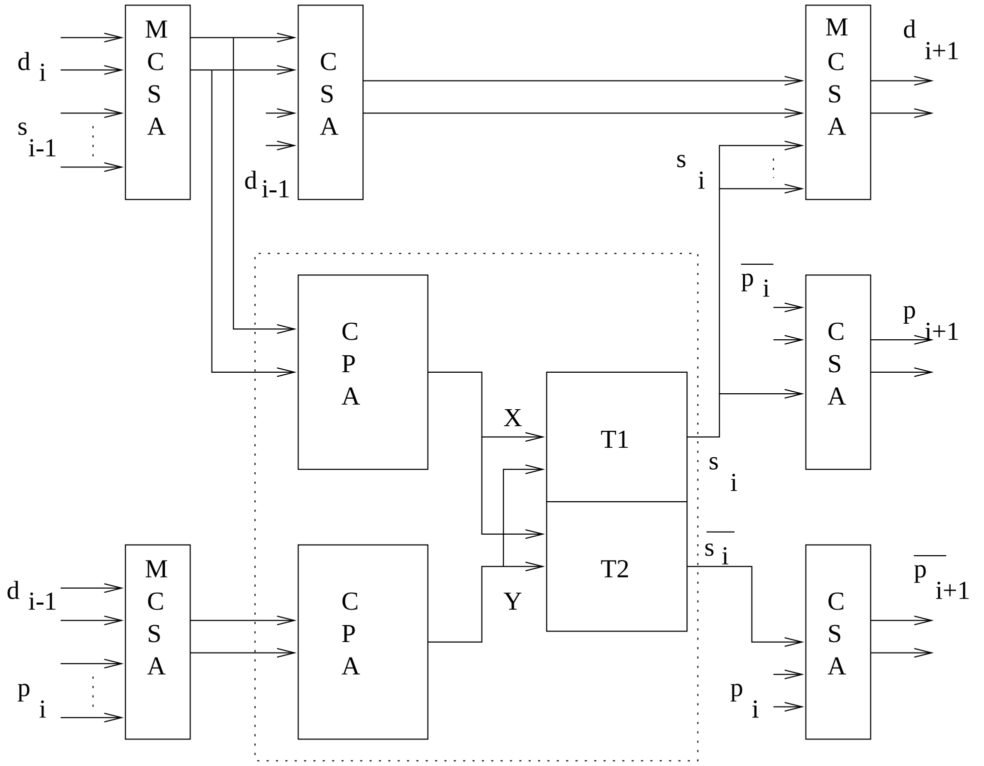


Figure 5: *The figure shows a radix-r base-implementation of the proposed division unit.* $\overline{s_i}\hat{=}\frac{1}{s_i}, \overline{p_i}\hat{=}\frac{1}{p_i}$*, MCSAs denote* $(\log r)$*-level CSA structures. T1, T2 are small* $\log r$ *by* $\log r$ *tables. However, as in the case of radix-2, a redundant digit representation enables us to reduce the complexity of choosing the next digit* s_i *to a few gates. The broken line en-capsules the area that can be simplified.*

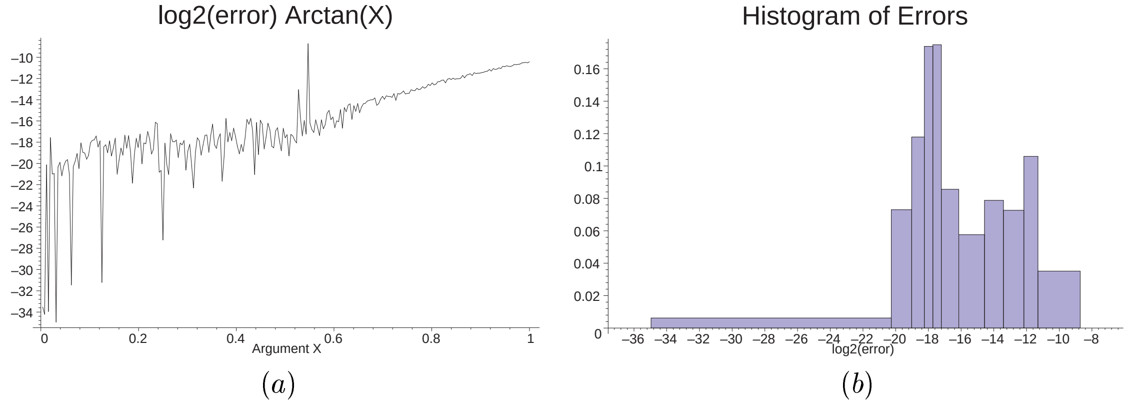


Figure 6: *The graphs show the error for computing arctan(x) after 16 iterations with 16-bit precision at each iteration. (a) shows* $\log_2(\texttt{error})$ *for arguments x in the range of* $[0, 1[$. *(b) shows the distribution of error values with a histogram.*

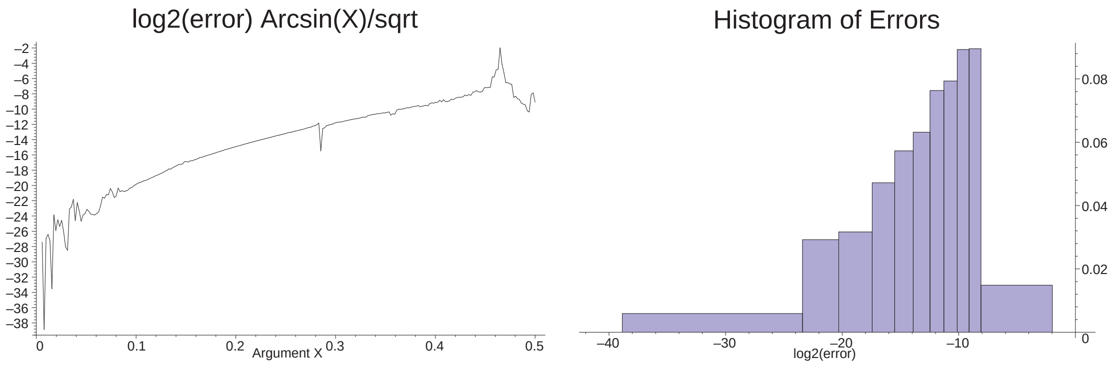
In summary, we have shown in this section the details on how to combine the MFT and the SEPA algorithm to design regular, pipelineable radix-2, and higher radix division units. Using the full power of the SEPA algorithm, we can compute a whole family of functions such as $T(x) = \frac{ax+b}{cx+d}, T(x) = \frac{ax^2+bx+c}{dx^2+ex+f}$, but also functions of more variables such as $T(x, y) = \frac{axy+bx+cy+d}{exy+fx+gy+h}$. For details on the algorithm for these higher order functions see [11].

The following section explores another application of the MFT method. We combine the division-like algorithm with the evaluation of a rational approximation in the form of a simple continued fraction.

4 Algorithm Class 2: Rational Approximations of Elementary Functions

In this section we explore a multiplication-based algorithm based on the MFT and the SEPA algorithm. By feeding a rational approximation in simple continued fraction form into the MFT/SEPA algorithm, we evaluate the rational approximation in a digit-serial fashion, obtaining one digit per iteration.

We focus here on the left step in figure 2: converting simple continued fraction expansions (rational approximations) with rational elements r_i to binary numbers using a digit-serial algorithm. We use the trivial *linear fractional transformation* $\frac{1 \cdot x+0}{0 \cdot x+1}$ and the **SEPA** algorithm to convert simple continued fraction expansions to a binary M-log-Fraction. In the examples below we use the approach from the previous section with a small modification to increase the speed of convergence. Instead of fixing the shifts (M_is) we compute the next digit and the next shift (M_i). Given the iteration equations 8 for the linear fractional SEPA algorithm we *choose* the following output digit of the redundant binary M-log-Fraction.



(a) (b)

Figure 7: *The graphs show the error for computing* $\frac{arcsin(x)}{\sqrt{1-x^2}}$ *after 16 iterations with 16-bit precision at each iteration. (a) shows* $\log_2(\texttt{error})$ *for arguments* x *in the range of* $[0, 0.5[$. *(b) shows the distribution of error values with a histogram.*

$$\begin{aligned} M_i &= \left\lfloor \log_2 \left((-1)^{i-1} \left(\frac{ax_i + b}{cx_i + d} - s_{i-1} 2^{-M_{i-1}} \right) \right) \right\rfloor \\ &\sim \lfloor \log_2 (ax_i + b - s_{i-1} 2^{-M_{i-1}} (cx_i + d)) \rfloor - \lfloor \log_2 (-1)^{i-1} (cx_i + d) \rfloor \\ &\quad \texttt{using } o_i = (-1)^{i-1} (s_i 2^{M_i} + s_{i-1} 2^{-M_{i-1}}) \end{aligned} \tag{25}$$

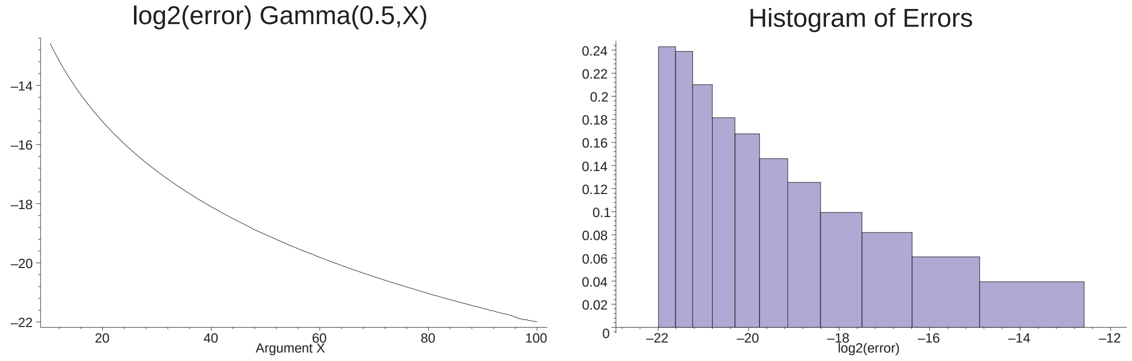
We avoid computing $\log(x)$ by using a *leading one detect*(LOD) circuit computing $\lfloor \log(x) \rfloor$. By computing an approximation to the optimal next shift (M_i) we improve the convergence of the algorithm to above 1 bit per iteration. The actual results of the examples below show the convergence of the algorithm to the exact function, and thus include also the convergence behavior of the rational approximation. As a consequence, for some arguments X, overall convergence drops below 1 bit per iteration.

Jones and Thron ([6], p. 202) give $\arctan(1)$ as an example of a function that converges much faster when approximated by a continued fraction compared to a Taylor approximation. In order to show the versatility of continued fraction approximations, we also show a composite function, and the incomplete Gamma function. For simplicity we use approximations with integer coefficients. For optimal results over an interval a minimax continued fraction which can be computed with Remez's method [15],[10] is preferable.

Example 1 *We approximate* $\arctan(x)$ *for* $x \in [0 \ldots 1]$ *with the continued fraction approximation given in [6], p. 202.*

$$arctan(x) = \left[0; \frac{1}{x}, \frac{3}{x}, \frac{5}{x} \left(\frac{1}{2} \right)^2, \frac{7}{x} \left(\frac{2}{3} \right)^2, \frac{9}{x} \left(\frac{3}{4} \right)^2, \ldots \right] \tag{26}$$

Figure 6 shows the results for 16 iterations with 16-bit precision at each iteration.



(a) (b)

Figure 8: *The graphs plot the error for computing the incomplete Gamma function* $\Gamma(0.5, x)$ *after 16 iterations with 16-bit precision at each iteration. (a) shows* $\log_2(\texttt{error})$ *for arguments* x *in the range of* $[10, 100]$*. (b) shows the distribution of error values with a histogram.*

Example 2 *We approximate the normalized function* $\frac{\arcsin(x)}{\sqrt{1-x^2}}$ *for* $x \in \{0, 0.5\}$ *with the continued fraction approximation given in [6], p. 203.*

$$\frac{arcsin(x)}{\sqrt{1-x^2}} = \left[0; \frac{1}{x}, -\frac{3}{1\cdot 2}\cdot\frac{1}{x}, \frac{5}{1\cdot 2}\cdot\frac{1}{x}, \frac{7}{3\cdot 4}\cdot\frac{1}{x}\cdots\right] \tag{27}$$

Figure 7 shows the results for 16 iterations with 16-bit precision at each iteration.

Example 3 *We approximate the incomplete Gamma function* $\Gamma(a, x)$ *for* $a = \frac{1}{2}, x \in [10 \ldots 100]$ *with the continued fraction approximation given in [6], p. 348, and [13].*

$$\Gamma(0.5, x) = e^{-x} \cdot x^{0.5} \cdot \left[0; x, (0.5)^{-1}, x, (1.5)^{-1}, x, (2.5)^{-1}, x, (3.5)^{-1} \ldots\right] \tag{28}$$

Figure 8 shows the results for 16 iterations with 16-bit precision at each iteration. We observe increasing precision with increasing distance from $x = 0$*. Intuitively, the even quotients of the continued fraction,* x*, lead to pseudo-divisions by zero. In fact,* $\Gamma(0.5, x)$ *has a pole at zero.*

In general, average precision is dependent on the precision for the iteration computations, while the standard deviation of the error in figure 6(a) varies with the number of iterations.

5 Conclusions

In conclusion, the MFT bridges the gap between continued fractions and the binary number representation, enabling the design of a new class of efficient rational arithmetic units. In this paper we show the contribution of the MFT to two application areas: (high radix) rational arithmetic units and the digit-serial evaluation of rational approximations.

From a distance the M-log-Fraction and MFT behave like a redundant representation of binary numbers by the integer distances between the '1's, just like 0-runlength encoding.

This relatively simple transformation allows us to exploit the symmetries in the continued fraction space while computing and storing binary numbers.

In this paper we showed some possible applications of the MFT/SEPA symbiosis. Future work consists of investigating other possible algorithms, implementations, and their competitiveness to state-of-the-art implementations of arithmetic units.

Acknowledgments

We thank D.W. Bump for discussions on the M-log-Fraction, and J.T. Gill and S. Oberman for helpful comments and suggestions.

Appendix A: Proof of Theorem 1

Throughout the appendix $\overline{M_i} = \sum_{j=1}^{i} M_j$ and $< M_1, M_2, M_3, \ldots >$ denotes the M-log-Fraction with parameters M_i.

Before proving theorem 1 we have to show that for any $\frac{A_i}{B_i}$ (see equations 3,4) of a binary M-log-Fraction (equation 10), $i = 1, 2, 3, \ldots$,

$$\boxed{B_i = (-1)^{\lfloor 0.5 \cdot i \rfloor} \cdot 2^{\overline{M_i}}} \tag{29}$$

Proof by induction

$$B_1 = a_1 = 2^{M_1} \qquad B_2 = B_1 a_2 + B_0 = -2^{M_1+M_2} \tag{30}$$

from equation 4 follows

$$B_{i+1} \quad = \quad (-1)^i (2^{M_{i+1}} + 2^{-M_i}) B_i + B_{i-1} \tag{31}$$

using equation 29 for B_i and B_{i-1} we obtain

$$\begin{aligned} B_{i+1} &= (-1)^i \left((-1)^{\lfloor 0.5 \cdot i \rfloor} 2^{\overline{M_{i+1}}} + (-1)^{\lfloor 0.5 \cdot i \rfloor} 2^{\overline{M_{i-1}}} \right) + (-1)^{\lfloor 0.5 (i-1) \rfloor} 2^{\overline{M_{i-1}}} = & (32) \\ &= (-1)^{\lfloor 0.5 (i+1) \rfloor} \cdot 2^{\overline{M_{i+1}}} & (33) \end{aligned}$$

q.e.d.

In addition to the proof above, we require identity(e.g. see [6] equation 2.1.9),

$$\boxed{A_{i+1} B_i - A_i B_{i+1} = (-1)^i} \tag{34}$$

for simple continued fractions.

Proof of Theorem 1

An M-log-Fraction of length n is equivalent to n powers of 2 for any $n \in \mathcal{N}$.

Direction 1: Binary M-log-Fraction of length $n \rightarrow n$ powers of 2.

We show by induction on the length of the M-log-Fraction that for any $n \in \mathcal{N}$:

$$\begin{aligned} <M_1, M_2, M_3, M_4, \ldots M_n> &\rightarrow 2^{-M_1} + 2^{-2M_1 - M_2} + 2^{-2M_1 - 2M_2 - M_3} + \cdots + 2^{-2\overline{M_{n-1}} - M_n} \\ &\rightarrow 2^{\beta_1} + 2^{\beta_2} + 2^{\beta_3} + \cdots + 2^{\beta_n} \end{aligned} \tag{35}$$

First, for length $n = 1$ and $n = 2$ we obtain by inspection:

$$< M_1 >= 2^{-M_1} \qquad < M_1, M_2 >= 2^{-M_1} + 2^{-2M_1 - M_2} \tag{36}$$

Given n,

$$< M_1, M_2, \ldots, M_n >= 2^{-M_1} + 2^{-2M_1 - M_2} + \cdots + 2^{-2\overline{M_{n-1}} - M_n} \tag{37}$$

we show that

$$< M_1, M_2, \ldots, M_{n+1} >=< M_1, M_2, \ldots, M_n > + 2^{-2\overline{M_n} - M_{n+1}} \tag{38}$$

$$< M_1, M_2, \ldots, M_{n+1} > - < M_1, M_2, \ldots, M_n > \;=\; \frac{A_{n+1}}{B_{n+1}} - \frac{A_n}{B_n} = \tag{39}$$

$$= (-1)^n \frac{1}{B_{n+1} B_n} \tag{40}$$

using identity 34.

Finally, we apply equation 29 and obtain

$$(-1)^n \frac{1}{B_{n+1} B_n} = 2^{-2\overline{M_n} - M_{n+1}} \tag{41}$$

Direction 2: n powers of 2 $\rightarrow$ Binary M-log-Fraction of length n, for any $n \in \mathcal{N}$.

$$\begin{aligned} 2^{\beta_1} + 2^{\beta_2} + 2^{\beta_3} + \cdots + 2^{\beta_n} &\rightarrow 2^{-M_1} + 2^{-2M_1 - M_2} + 2^{-2M_1 - 2M_2 - M_3} + \cdots + 2^{-2\overline{M_{n-1}} - M_n} \\ &\rightarrow < M_1, M_2, M_3, \ldots M_n > \end{aligned} \tag{42}$$

First, for length $n = 1$ and $n = 2$:

$$2^{\beta_1} =< -\beta_1 > \qquad 2^{\beta_1} + 2^{\beta_2} =< -\beta_1, 2\beta_1 - \beta_2 > \tag{43}$$

Given n,

$$2^{\beta_1} + 2^{\beta_2} + \cdots + 2^{\beta_n} =< M_1, M_2, \ldots, M_n > \tag{44}$$

we show that

$$2^{\beta_1} + 2^{\beta_2} + \cdots + 2^{\beta_{n+1}} =< M_1, M_2, \ldots, M_{n+1} > \tag{45}$$

By using equations (29,34,39,40,44) we conclude that

$$< M_1, M_2, \ldots, M_{n+1} > \quad - \quad < M_1, M_2, \ldots, M_n >= \frac{A_{n+1}}{B_{n+1}} - \frac{A_n}{B_n} = \tag{46}$$

$$= \quad 2^{-2\overline{M_n} - M_{n+1}} = 2^{\beta_{n+1}} \tag{47}$$

Thus, an M-log-Fraction of length n is equivalent to n powers of 2, with the relation between Ms and the powers of 2 as shown in equation 35.

q.e.d.

References

[1] O. Perron, *Die Lehre von den Kettenbrüchen,* Band I,II , Teubner Verlag, Stuttgart, 1957.

[2] A. Hurwitz, *Über die Kettenbrüche, deren Teilnenner arithmetische Reihen bilden,* Vierteljahrsschrift d. naturforsch. Gesellschaft, Zürich, Jahrgang 41, 1896.

[3] M. Hall, *On the sum and product of continued fractions,* Ann. of Math. 48, 966-993, 1947.

[4] R.W. Gosper, R. Schroeppel, M. Beeler, *HAKMEM,* Continued Fraction Arithmetic, MIT AI Memo 239, Feb. 1972.

[5] G.N. Raney, *On Continued Fractions and Finite Automata,* Math. Ann. 206, 265-283, 1973.

[6] W.B. Jones, W.J. Thron, *Continued Fractions: Analytic Theory and Applications,* Encyclopedia of Mathematics and its Applications, Vol. 11, Addison-Wesley, Reading, Mass., 1980.

[7] A. Bracha-Barack, *Application of Continued Fractions for Fast Evaluation of Certain Functions on a Digital Computer,* IEEE Trans. on Computers, March 1974.

[8] J.E. Robertson, K.S. Trivedi, *On the Use of Continued Fractions for Digital Computer Arithmetic,* IEEE Trans. on Computers, July 1977.

[9] P. Kornerup, D.W. Matula, *An algorithm for redundant binary bit-pipelined rational arithmetic,* IEEE Trans. on Computers, Vol. 39, No. 8, Aug. 1990.

[10] Waterloo Maple Inc., *Maple V,* http://www.maplesoft.com/.

[11] O. Mencer, M. Morf, M. J. Flynn, *Precision of Semi-Exact Redundant Continued Fraction Arithmetic for VLSI,* SPIE Advanced Signal Processing Algorithms, Architectures, and Implementations IX, Denver, July 1999.

[12] I. Koren, O. Zinaty, *Evaluating Elementary Functions in a Numerical Coprocessor Based on Rational Approximations,* IEEE Trans. on Computers, Vol. 39, No. 8, Aug. 1990.

[13] A.N. Khovanskii, *The application of continued fractions and their generalizations to problems in approximation theory,* 1956, translated by Peter Wynn, Groningen, Netherlands, P. Noordhoff, 1963.

[14] S. Oberman with M.J. Flynn, *Design issues in high performance floating point arithmetic units* PhD thesis, Dept. of Electrical Engineering, Stanford, Nov. 1996.

[15] E.Y. Remez, *General Computational Methods of Chebyshev Approximation,* Kiev, 1957. (see also L.A. Lyusternik et al, *Computing Elementary Functions,* Pergamon Press, 1965.)

[16] E.E. Schwartzlander, A.G. Alexpoulos *The sign-logarithm number system,* IEEE Trans. on Computers, Dec. 1975.

[17] J.E. Vuillemin, *Exact Real Computer Arithmetic with Continued Fractions,* IEEE Trans. on Computers, Vol. 39, No. 8, Aug. 1990.

[18] M.D. Ercegovac, T. Lang, *Division and Square Root, Digit-Recurrence Algorithms and Implementations,* Kluwer Academic Pubs, Massachusetts, 1994.