BKM: a new hardware algorithm for complex elementary functions
Jean-Michel Muller, Jean-Claude Bajard, Sylvanus Kla

To cite this version:
Jean-Michel Muller, Jean-Claude Bajard, Sylvanus Kla. BKM: a new hardware algorithm for complex elementary functions. IEEE Transactions on Computers, Institute of Electrical and Electronics Engineers, 1994, 43 (8), pp.955-963. <10.1109/12.295857>. <ensl-00086894>

HAL Id: ensl-00086894
https://hal-ens-lyon.archives-ouvertes.fr/ensl-00086894
Submitted on 20 Jul 2006

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L’archive ouverte pluridisciplinaire HAL, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d’enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.
Abstract—A new algorithm for computing the complex logarithm and exponential functions is proposed. This algorithm is based on shift-and-add elementary steps, and it generalizes some algorithms by Briggs and De Lugish, as well as the Cordic algorithm. It can easily be used to compute the classical real elementary functions (sin, cos, arctan, ln, exp). This algorithm is more suitable for computations in a redundant number system than the Cordic algorithm, since there is no scaling factor when computing trigonometric functions.

Index Terms—Computer arithmetic, elementary functions, cordic, redundant number systems.

I. INTRODUCTION

The point at stake in this paper is the search for algorithms that rapidly compute elementary functions. Many methods have been proposed in the literature: approximation by polynomials or rational functions [3], [5], [6], [10], use of Newton’s method, approximation by continued products, continued fractions, E-Method [8], [9] and shift-and-add methods. The algorithms presented in this paper belong to the class of shift-and-add methods. These methods are based on simple elementary steps: additions, and shifts (i.e., multiplications by a power of the radix of the number system used), and they go back to the 17th century: Briggs, a contemporary of Neper, invented an algorithm that made it possible to build the first tables of logarithms. For instance, to compute the logarithm of $x$ in radix-2 arithmetic, numerous methods (including that of Briggs, adapted to this radix) [4], [12], [13], [15] broadly consist of finding a sequence $d_k = -1, 0, 1$, such that $x \prod_{k=1}^{n} (1 + d_k 2^{-k}) = 1$. Then $\ln(x) \approx -\sum_{k=1}^{n} \ln(1 + d_k 2^{-k})$. Another method belonging to the shift-and-add class is the Cordic algorithm, introduced in 1959 by J. Volder [17] and then generalized by J. Walther [18]. CORDIC allows computation of many functions [18]. For instance, $e^x$ is obtained as $\cosh x + \sinh x$, while $\ln x$ is obtained as $2 \tanh^{-1} \left( \frac{1-x}{1+x} \right)$. CORDIC has been implemented in many pocket calculators and floating-point coprocessors. Its major drawback arises when performing the iterations using a redundant (e.g., carry save or signed-digit) number system. Such systems are advantageous for quickly-performed arithmetic, since they make it possible to perform carry-free additions [2]. In these systems, $d_n$ is difficult to evaluate. For instance, assume that we are in the rotation mode of CORDIC (see Table I), and that numbers are represented by $p$ digits, in radix 2 with digits $-1, 0$ or 1. $d_n$ is equal to the sign of the most significant nonzero digit of $z_n$: to find its value, we must examine some number of digits which may be close to $p$. Thus, the advantage of the redundant representation (a constant time elementary step) would be lost. An alternative is to accept $d_n = 0$: this makes it possible to examine a few digits of $z_n$ only. Unfortunately, with such a method the scale factors $K$ and $K'$ are not constants. $K$ is equal to $\prod_{k=0}^{\infty} \sqrt{1 + d_k^2 2^{-2n}}$, it is a constant if the $d_k$’s are all equal to $\pm 1$, but it is no longer a constant if the $d_k$’s are allowed to be zero. Many solutions have been suggested to overcome this problem. Broadly speaking, they lead to a repetition of iterations in time [16], [1], or in space [7]. In order to avoid this wasteful repetition, we need to work out a new algorithm.

In the following, assume that we are using a radix-2 classical or signed-digit number system. Extension to binary carry-save representation is straightforward. Let us consider the basic step of CORDIC in trigonometric mode (i.e., iteration (1) with $n = 1$). If we define the complex number $L_n$ as $L_n = x_n + iy_n$, we obtain $L_{n+1} = L_n (1 + id_n 2^{-n})$, this relation is close to the basic step of Briggs’ algorithm. This remark brings us to a generalization of that algorithm: we could perform multiplications by terms of the form $(1 + d_n 2^{-n})$, where the $d_n$’s are complex numbers, chosen such that a multiplication by $d_n$ can be reduced to a few additions. In this paper, we study the following iteration, called BKM:

$$\begin{align*}
L_{n+1} &= L_n (1 + d_n 2^{-n}) \\
E_{n+1} &= E_n - \ln(1 + d_n 2^{-n})
\end{align*}$$

with $d_n \in \{-1, 0, 1\}$, and whose imaginary part is between $-\pi$ and $\pi$.

- If we are able to find a sequence $d_n$ such that $L_n$ goes to 1, then we will obtain $E_n \to E_1 + \ln(L_1)$. We call this iteration mode the $L$-mode of the BKM algorithm.

Manuscript received September 17, 1993; revised February 23, 1994.

J.-C. Bajard is with Lab. LIM, Université de Provence, Marseille, 3 Place Victor Hugo, 13331 Marseille Cedex 03, France.
S. Kla is with INSET, Dept. FS, BP 1093 Yamussukro, Ivory Coast.
J.-M. Muller is with CNRS, Lab. LIP, Ecole Normale Supérieure de Lyon, 46 Allée d’Italie, 69364 Lyon Cedex 07, France; e-mail: jmmuller@lip.ens-lyon.fr.

IEEE Log Number 9203096.
If we are able to find a sequence $d_n$ such that $E_n$ goes to 0, then we will obtain $L_n \rightarrow L_1 e^{E_1}$. We call this iteration the E-mode of BKM. 

So, in the next sections, we will focus on the problem of finding sequences $d_n$ such that $E_n$ goes to 1 or such that $E_n$ goes to 0. From this study, we will deduce convenient algorithms for computing the complex logarithm and exponential functions.

### II. COMPUTATION OF THE EXPONENTIAL FUNCTION (E-MODE)

As pointed out at the end of the previous section, for computing $e^{E_1}$ using BKM, one needs to find a sequence $d_n$, $d_n = -1, 0, 1, -i, 1, i, -1, i, -1, -i, +1$, such that $\lim_{n \to \infty} E_n = 0$. Let us define $d_n^r$ and $d_n^i$ as the real and imaginary parts of $d_n$ (they belong to $\{-1, 0, 1\}$) and $E_n^r$ and $E_n^i$ as the real and imaginary parts of $E_n$. We easily find:

$$
\begin{align*}
E_{n+1}^r &= E_n^r + \frac{1}{2} \ln [1 + d_n^{2(n+1)} + (d_n^2 + d_n^2)^2 - 2n] \\
E_{n+1}^i &= E_n^i - d_n \arctan \left( \frac{2n}{1 + d_n^2 - 2^{2n}} \right)
\end{align*}
$$

(3)

In this section, we give an algorithm which computes the sequence $d_n$ for any $E_1$ belonging to a rectangular set $R_1 = [-s_1^r, r_1^r] + i[-r_1^i, r_1^i]$. The proof of our algorithm is based on the construction of a sequence $R_n = [-s_n^r, r_n^r] + i[-r_n^i, r_n^i]$ of rectangular sets, whose length goes to zero as $n$ goes to infinity, and such that for any $E_n \in R_n$, $d_n$ is such that $E_{n+1}^r \in R_{n+1}$. $d_n^r$ is chosen by examining a few digits of $E_n^r$ and $d_n^i$ is chosen by examining a few digits of $E_n^i$. These properties allow a simple and fast implementation of the choice of $d_n$.

#### A. Choice of $d_n^r$

The diagram presented in Fig. 1 shows the different parameters involved in determining $d_n^r$. This figure is close to the Robertson Diagrams that appear in many division algorithms [14]. In the following, we call such a diagram a Robertson diagram. The diagram is constructed as follows.

1) we assume that $E_1^r$ belongs to the interval $[-s_1^r, r_1^r]$, which is the real part of $R_n$.
2) $E_{n+1}^r$ is equal to $E_n^r - k \ln [1 + d_n^{2(n+1)} + (d_n^2 + d_n^2)^2 - 2n]$, so the value of $E_{n+1}^r$ vs. $E_n^r$ is given by various straight lines parameterized by $d_n^r$ and $d_n^i$.
3) $d_n^r$ must be such that for any possible value of $d_n^i$, $E_{n+1}^r \in [-s_n^r, r_n^r]$.

$r_n^r$ must be the largest value of $E_{n+1}^r$ corresponding to the straight line $d_n^r = 1$ (see Fig. 1). That is to say, $r_n^r$ must satisfy:

$$r_n^r = r_{n+1}^r - \ln (1 + 2^{-n})$$

Similarly, the lowest possible value for $E_n^r$ must correspond to the value obtained with $d_n^r = -1 \pm i$. This gives: $s_n^r = -\frac{1}{2} \ln (1 - 2^{-k} + 2^{-2k+1})$.

The terms $A_n$, $A_n$, $B_n$, and $B_n$ appearing in the diagram shown in Fig. 1 are equal to:

\[
\begin{align*}
A_n &= s_{n+1}^r + \frac{1}{2} \ln (1 - 2^{-n}) \\
B_n &= -s_{n+1}^r + \frac{1}{2} \ln (1 + 2^{-2n}) \\
A_n &= -s_{n+1}^r + \frac{1}{2} \ln (1 + 2^{-n+1} + 2^{-2n+1}) \\
B_n &= r_{n+1}^r.
\end{align*}
\]
One can prove that if $E_n^* < -B_n$ then $d_n^* = -1$.

We use the relation $E_{n+1}^* = E_n^* - d_n^* \arctan \left( \frac{2^{-n}}{1+2^{-n}} \right)$. Fig. 2 shows the Robertson diagram associated to the choice $d_n$. From this, we deduce: $r_n = \sum_{k=0}^{\infty} \arctan \left( \frac{2^{-k}}{1+2^{-k}} \right)$. The terms $C_n$ and $D_n$ appearing in the diagram are: $C_n = -r_{n-1}^* + \arctan \left( \frac{2^{-n}}{1+2^{-n}} \right)$ and $D_n = r_{n+1}^*$. One can prove that $C_n < D_n$. Thus, for any $E_n^* \in [-s_n^*, r_n^*]$, the following choices will give a value of $E_{n+1}^*$ between $-s_{n+1}^*$ and $r_{n+1}^*$:

$$
\begin{align*}
&\text{if } E_n^* < -B_n \text{ then } d_n^* = -1 \\
&\text{if } -B_n \leq E_n^* < A_n \text{ then } d_n^* = -1 \text{ or } 0 \\
&\text{if } A_n \leq E_n^* < B_n \text{ then } d_n^* = 0 \\
&\text{if } B_n < E_n^* \text{ then } d_n^* = 1.
\end{align*}
$$

The convergence domain $R_1$ of the algorithm is:

$$
-0.8298023738 \cdots < -s_1^* \leq E_1^* \leq r_1^* = 0.8688766517 \cdots \\
-0.7497803020 \cdots < -r_1^* \leq E_1^* \leq r_1^* = 0.7497803020 \cdots
$$

C. The Algorithm

Relations (4) and (5) make it possible to find a sequence $d_n$ such that, for $E_1 \in R_1$, $\lim_{n \to \infty} E_n = 0$. Now, let us try to simplify the choice of $d_n$; (4) and (5) involve comparisons that may require the examination of all the digits of the variables, we want to replace these comparisons by the examination of a small number of digits. The parameters $\hat{A} = -1/2$, $A = 1/4$, $C = 3/4$, $p_1 = 3$ and $p_2 = 4$ satisfy, for every $n$:

$$
\begin{align*}
2^p B_n \leq \hat{A} &< 2^{p_1} \hat{A} \\
2^p A_n \leq A &< A + 2^{p_1} \leq 2^p B_n \\
2^p C_n \leq C &< C + 2^{p_1} \leq 2^p D_n.
\end{align*}
$$

Therefore, if we denote $\hat{E}_n^*$ the number obtained by truncating $2^n E_n^*$ after its $p_1$th fractional digit, and $E_n^*$ the number obtained by truncating $2^n E_n^*$ after its $p_1$th fractional digit, we obtain, from (4), (5) and (6):

- if $\hat{E}_n^* \leq \hat{A} - 2^{-p_1}$ then $\hat{E}_n^* < \hat{A}$, therefore $d_n^* = -1$ is a valid choice.
- if $\hat{A} \leq \hat{E}_n^* \leq \hat{A}$ then $\hat{E}_n^* < \hat{A}$, therefore $d_n^* = 0$ is a valid choice.
- if $\hat{A} + 2^{-p_1} \leq \hat{E}_n^*$ then $\hat{E}_n^* < \hat{A}$, therefore $d_n^* = 1$ is a valid choice.
- if $\hat{E}_n^* \leq \hat{A} - 2^{-p_1}$ then $\hat{E}_n^* < \hat{A}$, therefore $d_n^* = -1$ is a valid choice.
- if $\hat{C} - 2^{-p_1} \leq \hat{E}_n^*$ then $\hat{E}_n^* < \hat{C}$, therefore $d_n^* = 0$ is a valid choice.
- if $\hat{C} + 2^{-p_1} \leq \hat{E}_n^*$ then $\hat{E}_n^* < \hat{C}$, therefore $d_n^* = 1$ is a valid choice.

From this, we deduce the $E$-mode of the BKM algorithm.

BKM Algorithm—$E$-mode

- Start with $E_1 \in R_1 = [-0.829801 \cdots, 0.868877 \cdots] + i[-0.749780 \cdots, 0.749780 \cdots].$
- Iterate:

$$
\left\{ \begin{array}{l}
L_{n+1} = L_n (1 + d_n 2^{-n}) \\
E_{n+1} = E_n - \ln (1 + d_n 2^{-n})
\end{array} \right.
$$

with $d_n = d_n^* + id_n^*$, chosen as follows:

- define $\hat{E}_n^*$ as the number obtained by truncating the real part of $2^n E_n$ after its 3rd fractional digit, and $E_n^*$ as the number obtained by truncating the imaginary part of $2^n E_n$ after its 4th fractional digit.

- if $\hat{E}_n^* \leq \hat{A} - 2^{-p_1}$ then $d_n^* = -1$
- if $\hat{A} \leq \hat{E}_n^* \leq \hat{A}$ then $d_n^* = 0$
- if $\hat{A} + 2^{-p_1} \leq \hat{E}_n^*$ then $d_n^* = 1$

- if $\hat{E}_n^* \leq \hat{A} - 2^{-p_1}$ then $d_n^* = -1$
- if $\hat{A} \leq \hat{E}_n^* \leq \hat{A}$ then $d_n^* = 0$
- if $\hat{A} + 2^{-p_1} \leq \hat{E}_n^*$ then $d_n^* = 1$

- Result: $\lim_{n \to \infty} L_n = L_1 e^E_1.$

In practice, instead of computing $E_{n+1} = E_n - \ln (1 + d_n 2^{-n})$ and examining the first digits of $\alpha_n = 2^n E_n$, one would directly compute the sequence $\alpha_{n+1} = 2 \alpha_n - 2^{n+1} \ln (1 + d_n 2^{-n}).$

D. Number of Iterations

Now, let us roughly estimate the number of iterations required to obtain a given accuracy. We want to compute $L_1 e^E_1.$ The sequence $d_n$ defined by the algorithm satisfies $L_1 e^E_1 = L_1 \prod_{i=1}^{\infty} (1 + d_i 2^{-1}).$ After $n$ iterations of the $E$-mode, we have computed $L_1 \prod_{i=1}^{n-1} (1 + d_i 2^{-1}).$ The relative error made by approximating $L_1 e^E_1$ by this value is $|1 - \prod_{i=1}^{n-1} (1 + d_i 2^{-1})|$, which is bounded by a term equivalent to $2^{-n}.$ Therefore, after $n$ iterations of the $E$-mode, we obtain a relative error approximately equal to $2^{-n}.$

1 Since $\hat{A}, A,$ and $\hat{E}_n^*$ have at most $p_1$ fractional digits, if $\hat{E}_n^* > \hat{A} - 2^{-p_1},$ then $E_n^* \geq \hat{A}.$
E. Number of Constants Stored

This algorithm requires the storage of the constants 
\(\ln(1 + d_k 2^{-i+1} + (d_k^2 + d_k^3 2^{-2i}))\), \(d_k^2 = -1, 0, 1\) and 
\(\arctan \left( \frac{1}{d_k^2 + d_k^3 2^{-2i}} \right)\). So, we need to store 9 constants. This result can be improved by observing that if \(i > n/2\), then 
\(\ln(1 + d_k 2^{-i+1} + (d_k^2 + d_k^3 2^{-2i}))\) and \(\arctan \left( \frac{1}{d_k^2 + d_k^3 2^{-2i}} \right)\) can be replaced by \(d_k^2 2^{-i+1}\) and \(2^{-i}\) with accuracy \(2^{-n}\). Thus, we only need to store \(9^n\) constants.

III. COMPUTATION OF THE LOGARITHM FUNCTION (L-MODE)

As shown in the introduction, computing the logarithm of a complex number \(L_1\) using BKM requires the calculation of a sequence \(d_n, d_n = -1, 0, 1, -i, i, i - 1, i + 1, -i - 1, -i + 1\), such that \(\lim_{n \to \infty} L_n = 1\), with \(L_{n+1} = L_n (1 + d_n 2^{-n})\).

A. A Straightforward Strategy

In the following, we use the norm \(\|\|\) defined as \(\|a + ib\| = \max\{|a|, |b|\}\). Let us define a sequence \(\epsilon_n\) as \(\epsilon_n = 2^n (L_n - 1)\). We deduce:

\[\epsilon_{n+1} = 2(\epsilon_n + d_n) + d_n \epsilon_n 2^{-n+1}.\]  

(7)

If we find a sequence \(d_n\) such that the sequence \(\epsilon_n\) is bounded, then \(L_n\) will go to 1. An intuitive solution is to choose \(d_n = -\epsilon_n\). So, in this section, we consider the following straightforward strategy which consists of building a sequence \(\|\epsilon_n\| \leq 3/2\) as follows:

* at step \(i\), we examine the value \(\epsilon_i\) obtained by truncating the real and imaginary parts of \(\epsilon_i\) after their \(p\)th fractional digits, where \(p\) is a very small integer.
* \(d_i\) is obtained by rounding the real and imaginary parts of \(-\epsilon_i\) to the nearest integer. Since \(p\) is small, this operation is easily performed. If \(\|\epsilon_i\| \leq 3/2\), this choice will give \(d_i \in D\).

If this algorithm actually gives \(\|\epsilon_n\| \leq 3/2\) for any \(n\), then \(L_n\) will go to 1. From \(\|\epsilon_i - \epsilon_i\| \leq 2^{-p}\) and \(\|d_i + \epsilon_i\| \leq 1/2\), using (7), we deduce:

\[\|\epsilon_{n+1}\| \leq 1 + 2^{1-p} + 2^{-n+1}\|\epsilon_n\|.\]  

(8)

The norm \(\|\|\) satisfies \(\|zz'\| \leq 2\|z\|\|z'\|\), therefore:

\[\|\epsilon_{n+1}\| \leq 1 + 2^{1-p} + 2^{-n+2}\|\epsilon_n\|.\]  

(9)

If \(n \geq 4, p \geq 4\), and \(\|\epsilon_n\| \leq 3/2\), then, using (9), one can prove that for any \(k \geq n\), \(\|\epsilon_k\| \leq 3/2\). Therefore, if we start the iteration (7) at step 4, from \(\epsilon_4\) satisfying \(\|\epsilon_4\| \leq 3/2\), then the strategy presented above will give a sequence \(d_n\) which fulfills \(\lim_{n \to \infty} L_n = 1\). This strategy allows computation of the logarithm in a very tiny domain: we can use it to compute the logarithm of \(L_4\) such that \(\|\epsilon_4\| = \|\ln(L_4 - 1)\| \leq 3/2\). So the convergence domain of this algorithm is \(L_4 \in [1 - \frac{3}{32}, 1 + \frac{3}{32}]\), and \(L_4 = [\frac{3}{32}, \frac{3}{32}]\).

B. Computation of the Logarithm in a Larger Domain

As in Section III-A, we study the sequence \(\epsilon_{n+1} = 2(\epsilon_n + d_n) + d_n \epsilon_n 2^{-n+1}\), where \(d_n\) is defined as \(2^n (L_k - 1)\). Our purpose is to start the evaluation at step \(k\) with \(\epsilon_k\) belonging to a domain \(T\) that will be given later, and to obtain, after a few steps, say \(n\) steps \((n \geq 3)\), a value \(\epsilon_{n+1}\) satisfying \(\|\epsilon_{n+1}\| \leq 3/2\). After this, the strategy presented in the previous section can be used. Our goal is to obtain a convergence domain \(L_1\) \(\frac{1}{2} T + 1\) larger than the previous one. The following algorithm was found through simulations, before being proved.

BKM Algorithm—L-Mode:

* Start with \(L_1\) belonging to the trapezoid \(T\) delimited by the straight lines \(x = 1/2, x = 1.3, y = x/2, y = -x/2\). \(T\) is the domain where the convergence is proven, but experimental tests show that the actual convergence domain of the algorithm is larger.

Iterate:

\[\begin{align*}
L_{n+1} &= L_n (1 + d_n 2^{-n}) \\
E_{n+1} &= E_n - \ln(1 + d_n 2^{-n})
\end{align*}\]

with \(d_n = d_k^n + id_k^n\), chosen as follows:

- define \(\epsilon_k^n\) and \(\epsilon_k^n\) as the real and imaginary parts of \(\epsilon_n = 2^n (L_n - 1)\), and \(\epsilon_k^n\) and \(\epsilon_k^n\) as the values obtained by truncating these numbers after their \(4\)th fractional digits.

* At step 1, we have the equation found at the bottom of the page.

* At step \(n, n \geq 2\):

- if \(\epsilon_k^n \leq -1/2\) then \(d_k^n = 1\)
- if \(-1/2 < \epsilon_k^n < 1/2\) then \(d_k^n = 0\)
- if \(1/2 < \epsilon_k^n\) then \(d_k^n = -1\)

* result: \(\lim_{n \to \infty} E_n = E_1 + \ln(L_1)\).

In a practical implementation, instead of computing \(L_n\) and examining the first digits of \(\epsilon_n = 2^n (L_n - 1)\), one would directly compute the sequence \(\epsilon_n\) using (7).

Proof of the Algorithm: Our goal is to show that if \(L_1 \in T\), then there exists \(n \geq 4\) such that \(\|\epsilon_n\| \leq 3/2\). Thus the proof of section 3.1 will hold. In order to prove...
this, we build a sequence $\beta_k$ of bounding sets, such that for any $L_1 \in T$, $\epsilon_k \in \beta_k$. Our problem is reduced to show that there exists $n \geq 4$ such that $\beta_n$ is included in the square $||z|| \leq 3/2$. Let us explain how the sequence $\beta_n$ is computed.

The first bounding set $\beta_1$ is equal to $2(T-1)$. At step $k$, $\beta_k$ is an aggregate of convex polygons, represented by their vertices. A step of the algorithm can be represented by a splitting of the complex plane into 9 convex $d$-areas. $d$-area associated with $\epsilon \in \{\pm 1, \pm i\}$ is the domain $D(\epsilon)$ such that if $\epsilon_k \in D(\epsilon)$, then the algorithm gives $d_k = \epsilon$. For instance, if $k \geq 2$, then $D(-1-i)$ is the set of the complex numbers whose real and imaginary parts are greater than 1/2. In $D(\epsilon)$, the transformation $\epsilon_{k+1} = 2(\epsilon_k + \epsilon) + d_k 2^{k+1}$ is a similarity, i.e., the combination of a rotation and the multiplication by a real factor. A similarity transforms a convex polygon into another convex polygon.

Each convex polygon of $\beta_k$ is split into sub-convex polygons, obtained by intersecting it with the $d$-areas. Fig. 3 presents the bounding step at step $k$, the various $d$-areas (for $k \geq 2$), and the splitting of the polygons of $\beta_k$.

Broadly speaking, $\beta_{k+1}$ is obtained by computing the transformation of each sub-convex polygon obtained after the splitting (the image of a polygon is obtained by computing the image of its vertices). However, we have to take into account the fact that the choice of $d_k$ is based on the examination of $\epsilon_k$ and $\epsilon_k^2$, which are obtained by truncating the real and imaginary parts of $\epsilon_k$ after their 4th fractional digits. For instance, if $\epsilon_k = \epsilon_k^2 + i\epsilon_k^2$ belongs to $D(-1)$, this does not prove that $\epsilon_k$ actually belongs to $D(-1)$. Therefore, to each sub-convex polygon, a "ribbon" of length $2^{-4}$ is added, so that if $\epsilon_k$ belongs to the "old" sub-convex polygon, then $\epsilon_k$ belongs to the "new" sub-convex polygon. After this, for each "new" sub-polygon, we compute its image by the similarity defined by the value of $d_k$ associated with the polygon (Fig. 4). This gives the new bounding set $\beta_{k+1}$.

The proof by induction that for any $L_1 \in T$, $\epsilon_k \in \beta_k$ is straightforward. If we find $n \geq 4$ such that all the vertices of the sub-convex polygons of $\beta_n$ are in the square $||z|| \leq 3/2$, then the algorithm is proven. The adequate value of $n$ is 6: this leads to a number of vertices which is much too large to be examined by a paper-and-pencil method. We have used a program for computing all the vertices of $\beta_n$, this program is written in ML, and uses exact rational arithmetic. Fig. 5 shows the bounding sets $\beta_1$, $\beta_4$, and $\beta_6$. Using this program, we have verified that all the vertices of $\beta_6$ are included in the square $||z|| \leq 3/2$.

C. Number of Iterations

Let us estimate the number of iterations required to obtain a given accuracy. The sequence $d_l$ satisfies $\ln(L_1) \approx -\sum_{k=1}^{\infty} \ln(1 + d_k 2^{-k})$. After $n$ iterations of the $L$-mode, we have computed $E_1 - \sum_{k=1}^{n} \ln(1 + d_k 2^{-k})$. The absolute error made by approximating $E_1 + \ln(L_1)$ by this value is $\sum_{k=n+1}^{\infty} \ln(1 + d_k 2^{-k})$, which is bounded by a term equivalent to $2^{-n} \sqrt{2}$. From this we deduce that, to obtain an absolute error equal to $2^{-n}$, one needs to perform $n + 1$ iterations.

IV. RANGE REDUCTION

The algorithms for computing elementary functions generally converge in some bounded domain. For computing $f(x)$ with an arbitrary value $x$, one usually needs to find a value
A. Range Reduction for the Complex Exponential Function

We assume that if \( I \) is an interval containing zero, whose length is greater than \( \rho \), we can compute, from any real \( x \), an integer \( k \) such that \( x - k \rho \in I \). This can be done by performing a few steps of a SRT-like division algorithm. Assume that we want to compute \( e^{x+i\gamma} \). BKM allows the evaluation of the exponential function in \( R_1 = [-0.82980237\ldots, +0.86887665\ldots] + i [-0.749780302\ldots, +0.749780302\ldots] \). The range reduction can be performed as follows:

1. Compute \( k_y \) such that \( y - k_y \frac{\pi}{2} \in [-\pi, \pi] \). Define \( y^* = y - k_y \frac{\pi}{2} \).
2. We have: \( e^{x+i\gamma} = e^{(k_y \mod 8) \frac{\pi}{2} + i\gamma} \). The multiplication by \( e^{(k_y \mod 8) \frac{\pi}{2}} \) looks difficult to reduce to a small amount of additions and shifts. Fortunately, this problem is easily overcome. As an example, let us consider the case \( k_y \mod 8 = 1 \). The term \( e^{\frac{\pi}{2}} \) is equal to \( i \sqrt{2} \). A multiplication by \( 1+i \) is easily reduced to two additions. A similar trick can be used for the other possible values of \( k_y \mod 8 \).

Assume that we want to compute the logarithm of \( z_{\text{init}} = x_{\text{init}} + iy_{\text{init}} \). In order to do this, from \( z_{\text{init}} \) we obtain \( z_{\text{Bkm}} = 2^k (d^x + iy^x) z_{\text{init}} \), with \( d^x, d^y \) not simultaneously equal to zero, such that \( z_{\text{Bkm}} \) belongs to \( T \). After this, \( \ln(z_{\text{Bkm}}) \) is computed using the L-mode, then \( k \ln(2) + \ln(d^x + iy^x) \) is subtracted from the result (we just need to store the 8 possible values of \( \ln(d^x + iy^x) \)). This reduction is performed in three steps:

- **Prescaling**: We find \( k_1 \) such that \( x = x + iy = \pm 2^k z_{\text{init}} \) satisfies \( \frac{1}{2} \leq x < 1 \).
- **Search for a Cone \( C_p \) Containing \( z \)**: Now, let us find \( p \) such that \( z \in C_p \). Since \( x > 1/2 \), the only possible values of \( p \) are \( 0, 1, 2, 6, 7 \). From the definition of the cones \( C_p \), we easily find (see Fig. 6):

\[
\begin{align*}
\text{If} & \quad |y| \leq x/2 & \quad \text{then} & \quad z \in C_0 \\
\text{If} & \quad -3x \leq y \leq -x/3 & \quad \text{then} & \quad z \in C_7 \\
\text{If} & \quad x/3 \leq y \leq 3x & \quad \text{then} & \quad z \in C_1 \\
\text{If} & \quad y \leq -2x & \quad \text{then} & \quad z \in C_6 \\
\text{If} & \quad y \geq 2x & \quad \text{then} & \quad z \in C_2.
\end{align*}
\]

By taking into account the overlapping of the cones \( C_p \), we can replace these comparisons by comparisons involving only a few digits of \( x \) and \( y \). If we define \( \bar{x} \) and \( \bar{y} \) as the numbers obtained by truncating \( x \) and \( y \) after their 5th fractional digits (as previously, we assume a binary number system), we easily deduce: \( |x - \bar{x}| \leq 1/2 \) and \( |y - \bar{y}| \leq 1/2 \). Therefore we have the following:

1. **Range reduction** for the complex logarithm function

Let us define the cone \( C_0 \) as the set of the numbers \( x + iy \) such that \( |y| \leq x/2 \) (see Fig. 6), and \( C_1, C_2, C_3, C_4, C_5, C_6, \) and \( C_7 \) as \( C_0 \pm 1/2 \). For each nonzero element \( z \) of \( C_0 \) there exists an integer \( n \) such that \( 2^n C_0 \) belongs to the convergence domain \( T \) of the L-mode. Since \( C_0 = \lambda C_0 \) for any nonnegative real \( \lambda \), we easily find:

\[
\begin{align*}
C_1 &= (1+i)C_0 = \frac{C_0}{1-i} & C_4 &= (-1+i)C_0 = \frac{C_0}{1+i} \\
C_2 &= iC_0 = C_7 & C_5 &= -iC_0 = C_6 \\
C_3 &= (-1+i)C_0 = \frac{C_0}{1-i} & C_7 &= (1-i)C_0 = \frac{C_0}{1+i} \\
C_4 &= -C_0.
\end{align*}
\]
V. COMPUTATION OF ELEMENTARY FUNCTIONS

As shown in the previous sections, BKM makes it possible to compute the following functions:

- in E-mode, \( L_1E_1 \), where \( E_1 \) is a complex number belonging to \( R_1 \),
- in L-mode, \( E_1 + \ln(L_1) \), where \( L_1 \) belongs to the trapezoid \( T \).

A. Functions Computable Using One Mode of BKM

1) Real sine and cosine functions: In the E-mode of BKM, one can compute the exponential of \( E_1 = 0 + j\theta \), and obtain \( L_n = \cos \theta + j\sin \theta \pm 2^{-n} \).

2) Real Exponential Function: If \( E_1 \) is a real number belonging to \([-0.8298023738, +0.8688766517]\), the E-mode will give a value \( L_n \) equal to \( L_1e^{E_1} \pm 2^{-n} \).

3) Real Logarithm: If \( L_1 \) is a real number belonging to \([0.5, 1.3]\), the E-mode will give a value \( E_n \) equal to \( L_1 + \ln(L_1) \pm 2^{-n} \). Furthermore, in this case, the BKM iteration is reduced to Briggs’ algorithm, and it is possible to show that the algorithm gives a correct result if and only if \( L_1 \in [\Pi_{n=1}(1 + 2^{-n})^{-1}, \Pi_{n=\infty}(1 - 2^{-n})^{-1}] \approx [0.419422, 1.73137] \).

4) 2-D Rotations: As pointed out in [11], performing rotations is useful for Fast Fourier Transformation, Digital Filtering, and Matrix Computations. The 2-D vector \((c, d)\) obtained by rotating \((a, b)\) of an angle \( \theta \) satisfies: \( c + id = (a + ib)e^{i\theta} \) therefore, \((c, d)\) is computed using the E-mode, with \( L_1 = a + ib \) and \( E_1 = i\theta \) (see Fig. 7).

5) Real arctan Function: From the relation:

\[
\ln(x + iy) = \begin{cases} 
\frac{1}{2}\ln(x^2 + y^2) + i \arctan \frac{y}{x} \mod (2\pi) & \text{if } x > 0 \\
\frac{1}{2}\ln(x^2 + y^2) + i(\pi + \arctan \frac{y}{x}) \mod (2\pi) & \text{if } x < 0 
\end{cases}
\]

one can easily deduce that, if \( x + iy \) belongs to the convergence domain of the L-mode of BKM, then \( \arctan(y/x) \) is the limit value of the imaginary part of \( E_n \), assuming that the L-mode is used with \( E_1 = 0 \) and \( L_1 = x + iy \) (see Fig. 7).
Fig. 9. Computation of \( x \sqrt{a} \) and \( y \sqrt{a} \) in parallel.

Fig. 10. Computation of lengths and normalization.

B. Functions Computable Using Two Consecutive Modes of BKM

Using two BKM operations, one can compute many functions. Some of these functions are the following.

1) Complex Multiplication and Division: The product \( zt \) is evaluated as \( z.e^{j\theta} \), while \( z/t \) is evaluated as \( z.e^{-j\theta} \) (see Fig. 8). One can compute \((ab)e^z\) or \((\bar{z})e^z\), where \( a, b \) and \( z \) are complex numbers, using the same operator, by choosing \( E_t^\theta \) equal to the real part of \( z \), and \( E_y^\theta \) equal to the imaginary part of \( z \).

2) Computation of \( x \sqrt{a} \) and \( y \sqrt{a} \) in Parallel (\( x, y \) and \( a \) are real numbers): we use the relation \( \sqrt{a} = e^{j\pi/4} \), as shown previously, the L-mode allows the computation of \( F = \frac{1}{2} \ln(a^2 + b^2) = \ln \sqrt{a^2 + b^2} \), where \( a \) and \( b \) are real numbers. Using the E-mode, we can compute \( e^F \) or \( e^{-F} \) (see Fig. 10).

VI. COMPARISON WITH CORDIC

In order to obtain \( p \) significant bits, CORDIC and BKM roughly need \( p \) iterations. BKM requires more hardware than CORDIC: BKM needs the storage of \( 2^p \) constants, while CORDIC needs the storage of \( p \) constants to compute trigonometric and hyperbolic functions. Since these constants are represented by \( p \) digits, both algorithms need a \( O(p^2) \) area for storage of the constants. Both algorithms need a shifter able to perform an \( n \)-position shift at step \( n \). A barrel shifter makes it possible to perform a \( n \)-position shift (for any \( n \leq p \)) in constant time, and lies in an area \( O(n^2) \). Since the area complexity of most adders is better than \( O(n^2) \), the area complexity of CORDIC and BKM is \( O(n^2) \). The computations performed during a BKM iteration are:

- For the variable:
  \[
  \begin{align*}
  \alpha_n &= 2 \alpha_n^{n+1} - 2^n \ln [1 + (d_n^2 + d_n^2)^{2-n} + (2^n + d_n^2)^{2-n}]
  
  \alpha_n^{n+1} &= 2 \alpha_n^p - 2^{n+1} + d_n^2 \text{ with } (d_n^2 + d_n^2)^{2-n} + (2^n + d_n^2)^{2-n}
  
  \text{if, instead of computing } \alpha_n \text{ and examining the first digits of } \alpha_n = 2^n \alpha_n, \text{ we directly compute } \alpha_n.
  
  \text{For the variable:}
  
  \begin{align*}
  L_n &= 2 \left[ (c_n^2 + d_n^2) + (d_n^2 + d_n^2)^{2-n} + (2^n + d_n^2)^{2-n} \right]
  
  L_n^{n+1} &= 2 \left[ (c_n^2 + d_n^2) + (d_n^2 + d_n^2)^{2-n} + (2^n + d_n^2)^{2-n} \right]
  
  \text{if, instead of computing } L_n \text{ and examining the first digits of } c_n = 2^n (L_n - 1), \text{ we directly compute } c_n.
  
  \text{So BKM looks more complicated than CORDIC. As a matter of fact, in order to compare CORDIC and BKM, we have to assume that we use a redundant number system. Using such a system, the time complexities of both algorithms are } O(p).
  
  \text{As pointed out in many papers dealing with CORDIC, efficient use of CORDIC with such a number system requires a doubling of the iterations in space [7] or in time [16], [1]. Doubling the CORDIC iterations in time gives:}
  
  \begin{align*}
  x_{n+1} &= x_n - d_n y_n 2^{-n} - d_n^2 x_n 2^{-2n-2}
  
  y_{n+1} &= y_n + d_n x_n 2^{-n} - d_n^2 x_n 2^{-2n-2}
  
  z_{n+1} &= z_n - 2d_n \text{arc}t \text{an}2^{-n-1}.
  
  \end{align*}

  \text{This iteration is at least as complex as the BKM iteration (at step } n \text{ one needs to perform an } n \text{-position shift and a } 2n - 2\text{-position shift: this requires a larger shifter, or several consecutive shifts). Doubling the iterations in space requires more control: in the branching CORDIC method [7], one needs to compare at each step the values given by two CORDIC modules. Furthermore, doubling iterations makes it possible to obtain a constant scale factor, but this scale factor remains different from 1, therefore, for computing many functions, one needs to perform a multiplication after the CORDIC iterations. So, although both methods have the same time and space complexities, BKM looks more interesting when using a redundant number system.}

VII. CONCLUSION

We have proposed a new algorithm for the computation of many elementary functions (complex exponential and logarithm functions, complex multiplication, complex functions \((ab)e^z\) and \((\bar{z})e^z\), real functions sin, cos, arctan\( \frac{x}{y} \), \( \ln(x^2 + y^2) \), \( x \sqrt{a^2 + b^2} \), \( x \sqrt{a^2 + b^2} \), and 2-D rotations). This algorithm matches the CORDIC algorithm, since it allows the use of a redundant number system without any scale factor problem and allows the computation of more functions.

ACKNOWLEDGMENT

We thank C. Paulin for her precious knowledge of ML and her careful reading of the manuscript.
REFERENCES


Jean-Claude Bajard was born in Saint-Etienne, France, in 1957. He received the Ph.D. degree in computer science in 1993 from the Ecole Normale Supérieure de Lyon.

He taught mathematics in high school from 1979 to 1990, and served as an assistant professor at Ecole Normale Supérieure de Lyon in 1993. He joined the Lab. I.I.M, Université de Provence, Marseille, France in 1993.

Dr. Bajard’s research interests include computer arithmetic and VLSI design.

Jean-Michel Muller (M’87) was born in Grenoble, France, in 1961. He received the Engineer degree in civil engineering from Ecole Nationale Supérieure des Travaux Publics, Yamoussoukro, Ivory Coast, in 1988, the M.S. degree in applied mathematics from Grenoble University, France, in 1989, and the Ph.D. degree from Lyon University, France, in 1993.

His research interests are in computer arithmetic and computer architecture.

Sylvanus Kla was born in Ivory Coast in 1963. He received the Engineer degree in civil engineering from Ecole Nationale Supérieure des Travaux Publics, Yamoussoukro, Ivory Coast, in 1988, the M.S. degree in applied mathematics from Grenoble University, France, in 1989, and the Ph.D. degree from Lyon University, France, in 1993.

His research interests are in computer arithmetic and computer architecture.

Dr. Muller served as General Chairman of the 10th Symposium on Computer Arithmetic (Grenoble, France, June, 1991).