(M, p, k)-friendly points: a table-based method for trigonometric function evaluation

Nicolas Brisebarre
CNRS-Laboratoire LIP
(CNRS, ENS Lyon, Inria, UCBL)
Ecole Normale Supérieure de Lyon
46 Allée d'Italie
69364 Lyon Cedex 07, France
Email: Nicolas.Brisebarre@ens-lyon.fr

Miloš D. Ercegovac
4731H Boelter Hall
Computer Science Department
University of California at Los Angeles
Los Angeles, CA 90024, USA
Email: milos@cs.ucla.edu

Jean-Michel Muller
CNRS-Laboratoire LIP
(CNRS, ENS Lyon, Inria, UCBL)
Ecole Normale Supérieure de Lyon
46 Allée d'Italie
69364 Lyon Cedex 07, France
Email: Jean-Michel.Muller@ens-lyon.fr

***Abstract*—We present a new way of approximating the sine and cosine functions by a few table look-ups and additions. It consists in first reducing the input range to a very small interval by using rotations with "(M, p, k) friendly angles", proposed in this work, and then by using a bipartite table method in a small interval. An implementation of the method for 24-bit case is described and compared with CORDIC. Roughly, the proposed scheme offers a speedup of 2 compared with an unfolded double-rotation radix-2 CORDIC.**

I. Introduction

We propose and investigate a new way of reducing the input argument, so that a bipartite method and networks of redundant adders can be employed for evaluating trigonometric functions with a reasonably large precision. We have illustrated the method applicability by describing a fully combinational scheme for computing 24-bit cos and sin functions.

A. The bipartite method

Sunderland et al. [1] considered approximating the sine of a 12-bit number x less than $\pi/2$ using tables. They proposed to split the binary representation of x into three 4-bit words, and to approximate the sine of $x = A + B + C$, where $A < \pi/2$, $B < 2^{-4}\pi/2$ and $C < 2^{-8}\pi/2$, using

$$\sin(A + B + C) \approx \sin(A + B) + \cos(A)\sin(C). \quad (1)$$

By doing that, instead of one table with 12 address bits (i.e., with 2^{12} elements), one needed two tables (one for $\sin(A + B)$ and one for $\cos(A)\sin(C)$), each of them with 8 address bits only. In 1995, DasSarma and Matula [2] introduced a new method for evaluation of reciprocals by table look-up and addition, and used it to generate seed values for computing reciprocals using the Newton–Raphson iteration. They named it the *bipartite method*. Generalized to other functions [3], [4], the bipartite method turned out, when applied to the trigonometric functions, to be the same as Sunderland et al. method.

Although the bipartite method is a fairly general method of approximating functions by table lookup and addition, in this section, we only focus on the problem of approximating $\sin(\theta)$ and $\cos(\theta)$, where θ is a small $(p - j)$-bit value less than 2^{-j}:

$$\theta = 0.0000\cdots0\theta_{j+1}\theta_{j+2}\theta_{j+3}\cdots\theta_p, \qquad \theta_i = 0, 1.$$

To simplify the presentation, assume that $p - j$ is a multiple of 3, say $p - j = 3q$ (typical practical values are $p = 24$ and $q = 6$), and write $\theta = \rho_1 + \rho_2 + \rho_3$, with

$$\rho_1 = \overbrace{0.0\cdots0}^{j \text{ bits}} \ \overbrace{\theta_{j+1}\cdots\theta_{j+q}}^{q \text{ bits}},$$

$$\rho_2 = \overbrace{0.0\cdots0}^{j \text{ bits}} \ \overbrace{0\cdots\cdots0}^{q \text{ bits}} \ \overbrace{\theta_{j+q+1}\cdots\theta_{j+2q}}^{q \text{ bits}},$$

and

$$\rho_3 = \overbrace{0.0\cdots0}^{j \text{ bits}} \ \overbrace{0\cdots\cdots0}^{q \text{ bits}} \ \overbrace{0\cdots\cdots0}^{q \text{ bits}} \ \overbrace{\theta_{j+2q+1}\cdots\theta_p}^{q \text{ bits}}.$$

We have:

$$\begin{aligned}\sin(\theta) &= \sin(\rho_1 + \rho_2)\cos(\rho_3) + \cos(\rho_1 + \rho_2)\sin(\rho_3)\\ &\approx \sin(\rho_1 + \rho_2) + \cos(\rho_1)\sin(\rho_3),\end{aligned} \quad (2)$$

and

$$\cos(\theta) \approx \cos(\rho_1 + \rho_2) - \sin(\rho_1)\sin(\rho_3). \quad (3)$$

Define four tables T_1, T_2, T_3, and T_4, each one with $2q$ address bits, as

$$\begin{array}{rcl} T_1(\rho_1, \rho_2) &=& \sin(\rho_1 + \rho_2),\\ T_2(\rho_1, \rho_3) &=& \cos(\rho_1)\sin(\rho_3),\\ T_3(\rho_1, \rho_2) &=& \cos(\rho_1 + \rho_2),\\ T_4(\rho_1, \rho_3) &=& -\sin(\rho_1)\sin(\rho_3).\end{array} \quad (4)$$

Then, according to the bipartite method,

$$\begin{array}{rcl} \sin(\theta) &\approx& T_1(\rho_1, \rho_2) + T_2(\rho_1, \rho_3),\\ \cos(\theta) &\approx& T_3(\rho_1, \rho_2) + T_4(\rho_1, \rho_3).\end{array} \quad (5)$$

Elementary calculation shows that the error of the first approximation is bounded by $3 \cdot 2^{-3j-3q-1} + 2^{-3j-4q-1}$, and the error of the second approximation is bounded by $2^{-2j-3q} + 2^{-2j-4q-1} + 2^{-4j-4q-1}$. Of course, tables T_1,

T_2, T_3, and T_4 store their values with limited precision: functions $\sin(\rho_1 + \rho_2)$, $\cos(\rho_1)\sin(\rho_3)$, $\cos(\rho_1 + \rho_2)$, and $\sin(\rho_1)\sin(\rho_3)$ are rounded to some precision, and these rounding errors must be added to the approximation errors given above. For instance, if $j = q = 6$ and if each value in the tables T_1, T_2, T_3 and T_4 is rounded to the nearest 28-bit number, then the error on $\sin(\theta)$ is bounded by $0.0313 \cdot 2^{-24}$ and the error on $\cos(\theta)$ is bounded by $0.047 \cdot 2^{-24}$.

The major advantage of the bipartite method, compared to a straightforward tabulation of $\sin(\theta)$ and $\cos(\theta)$ is that instead of two tables with $3q$ address bits, we need four tables with $2q$ address bits only. Also, no multiplication is required: the bipartite method just uses tabulation and addition.

However, unless j is large (that is, unless θ is small), we cannot tackle large precisions (i.e., large values of p) with this method. Variants have been suggested (see, e.g., [5]), and yet, since the bipartite method is intrinsically a linear-approximation method, it has an inherent limitation: to be able to evaluate functions with large precisions, we need to reduce the input arguments to very small values.

Recently, Matula and Panu suggested to "prescale" the input value before using the bipartite algorithm to obtain a single-precision ulp accurate reciprocal [6]. Concerning trigonometric functions, a prescaling (i.e., a preliminary multiplication by some value) would not help. However, given an input value x, subtracting from x an adequately chosen value $\hat{x}$ could make it possible to use the bipartite method for single-precision evaluation of sines and cosines.

This is what we address in Section II.

B. Canonical recoding

To simplify implementation of the proposed sin/cos computation, we minimize the number of non-zero digits in table entries by using the canonical recoding [7]. It has the property that any n-bit integer $D \in \{0, \ldots, 2^n - 1\}$ in radix 2 can be recoded into its canonical form:

$$D = f_n f_{n-1} \cdots f_0, \qquad f_i = \pm 1 \quad \text{or} \quad 0,$$

such that the number of non-zero digits is $n/3$ on average. Moreover, there can be no two consecutive non-zero digits, so that the maximum number of nonzero digits is always less than or equal to $\lceil \frac{n+1}{2} \rceil$. The canonical recoding is an improvement of the Booth recoding [8] (where a string of v consecutive 1s, starting in position u, is replaced by a pair $(1 \cdot 2^{u+v}, (-1)2^u)$).

From an input binary number $d_{n-1}d_{n-2}\cdots d_0$, we obtain its canonical representation $f_n f_{n-1} \cdots f_0$ using the following expressions [7], where $f_n = c_n$, with $c_0 = 0$ and $d_n = 0$:

$$\begin{array}{rcl} c_{i+1} & = & \lfloor (d_{i+1} + d_i + c_i)/2 \rfloor \\ f_i & = & d_i + c_i - 2c_{i+1} \end{array} \tag{6}$$

or, in a tabular form as shown Fig. 1.

c_i	d_{i+1}	d_i	f_i	c_{i+1}
0	0	0	0	0
0	0	1	1	0
0	1	0	0	0
0	1	1	$\overline{1}$	1
1	0	0	1	0
1	0	1	0	1
1	1	0	$\overline{1}$	1
1	1	1	0	1

Figure 1. Canonical recoding.

II. (M, p, k)-FRIENDLY POINTS AND ANGLES

Assume an input angle x between 0 and $\pi/2$. We wish to evaluate $\sin(x)$ and $\cos(x)$. Ideally, we would like to write $x = \hat{x} + \theta$, where θ is small enough, so that $\sin(\theta)$ and $\cos(\theta)$ are easily approximated, with high accuracy, using the bipartite method, and where $c = \cos(\hat{x})$ and $s = \sin(\hat{x})$, which would be tabulated, fit in a very small number of bits only, so that multiplications by c and s are reduced to a very small number of additions. We would then obtain $\cos(x)$ and $\sin(x)$ as $c \cdot \cos(\theta) - s \cdot \sin(\theta)$ and $s \cdot \cos(\theta) + c \cdot \sin(\theta)$, respectively. However, such values $\hat{x}$ do not exist: one easily shows that the only numbers whose sine and cosine fit in a finite number of bits are the multiples of $\pi/2$.

Hence, we propose to do something slightly different. We will look for numbers $\hat{x}$ such that $\cos(\hat{x})$ and $\sin(\hat{x})$ are of the form $a \cdot z$ and $b \cdot z$, respectively, where a and b are small integers (so that a multiplication by a and b reduces to a very small number of additions—that can be performed in carry save or borrow save), and so that a multiplication by $z = 1/\sqrt{a^2 + b^2}$ can be performed approximately, with a very good approximation, by a very small number of additions or subtractions. More precisely, if we aim at implementing precision-p arithmetic, we will require that z could be rewritten

$$2^e \cdot 1.z_1 z_2 z_3 \cdots z_p z_{p+1} z_{p+2} \cdots = \sum z_i 2^{e-i},$$

with $z_i = 0$ or ± 1 and $z_1 \neq -1$, where the number of nonzero values z_i for $i \leq p$ is less than some very small bound k. Hence, multiplying by z with relative error less than 2^{-p} will require less than k additions/subtractions. An adequate value of k is found by a trial-and-error process: if k is too large, the multiplications by z will be costly, and if k is too small, there won't be many (M, p, k)-friendly points, so that the largest distance between two consecutive (M, p, k)-friendly points (which directly determines the accuracy of the algorithm) will be too large.

Hence, we are interested in pairs of integers (a, b) that satisfy the following definition.

Definition 1: A pair of integers (a, b) is an (M, p, k)-friendly point if:

1) $0 \leq a \leq M$ and $0 \leq b \leq M$;
2) the number

$$z = \frac{1}{\sqrt{a^2 + b^2}}$$

can be written

$$2^e \cdot 1.z_1 z_2 z_3 \cdots z_p z_{p+1} z_{p+2} \cdots = \sum z_i 2^{e-i},$$

where e is an integer, $z_i \in \{-1, 0, 1\}$, $z_1 \neq -1$, and the number of terms z_i such that $1 \leq i \leq p$ and $z_i \neq 0$ is less than or equal to k.

An example of (M, p, k)-friendly point is the following. Assume $M = 255$, $p = 24$, $k = 7$, and consider $a = 72$ and $b = 106$. We have

$$z = \frac{1}{\sqrt{a^2 + b^2}} = 0.00000001111111111011100000011110 \cdots,$$

which can be recoded into its canonical form:

$$z = 0.0000001000000000\bar{1}00\bar{1}0000010000\bar{1}0,$$

where "$\bar{1}$" stands for the digit "-1". A multiplication by z reduces to 3 subtractions and one addition. Hence (a, b) is a $(255, 24, 7)$-friendly point. Moreover, it is a $(127, 24, 5)$-friendly point.

Definition 2: The number α, $0 \leq \alpha \leq \pi/2$ is an (M, p, k)-friendly angle if either $\alpha = 0$ or

$$\alpha = \arctan \frac{b}{a},$$

where (a, b) is an (M, p, k)-friendly point.

For instance, Figure 2 presents the (M, p, k)-friendly points, with $M = 64$, $p = 24$, and $k = 7$, and Figure 3 presents the (M, p, k)-friendly points, with $M = 256$, $p = 24$, and $k = 5$.

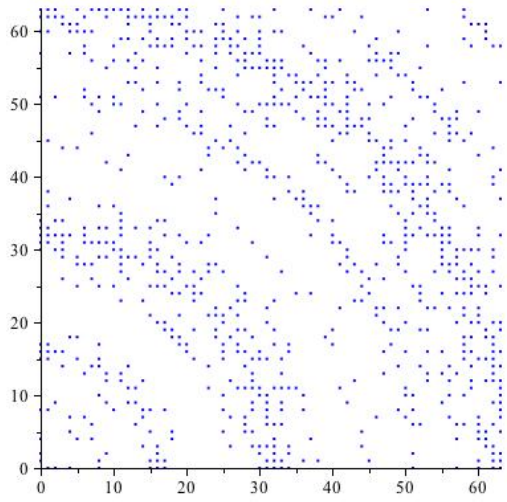

Figure 2. The (M, p, k)-friendly points, with $M = 63$, $p = 24$, and $k = 7$. The maximum distance between two consecutive angles is $0.0156237 \cdots < 2^{-6.00011}$.

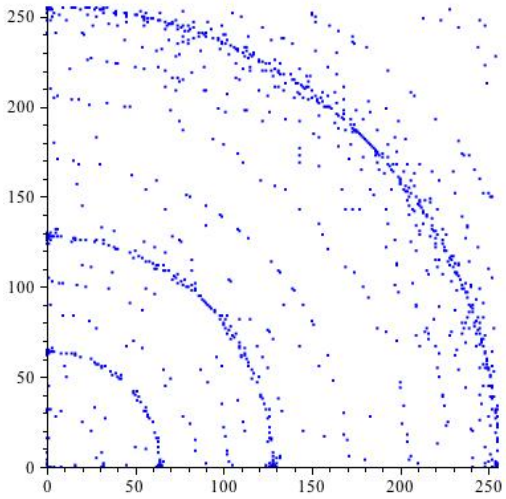

Figure 3. The (M, p, k)-friendly points, with $M = 255$, $p = 24$, and $k = 5$. The maximum distance between two consecutive angles is $0.0130652 \cdots < 2^{-6.258}$.

III. The algorithm

A. General sketch of the algorithm

Following the previous discussion, our algorithm, given an input angle x and parameters p, k and M, with (M, p, k)-friendly points precomputed and stored in a table, consists of:

- looking up in a table, addressed by a few leading bits of x, an (M, p, k)-friendly angle $\hat{x}$ and the associated values a, b, and z (in canonical form);
- computing, using the bipartite method, $\sin(\theta)$ and $\cos(\theta)$, according to Eqs. (4) and (5), where $\theta = x - \hat{x}$;
- computing $C = a\cos(\theta) - b\sin(\theta)$ and $S = b\cos(\theta) + a\sin(\theta)$ using a very few additions/subtractions—since a and b are less than M, multiplying by a and b requires at most $\frac{1}{2}\lceil \log_2 M \rceil$ additions/subtractions, that can be performed without carry propagation using redundant (e.g., carry-save) arithmetic;
- finally, multiplying C and S by z by adding a very few (at most k) multiples of C (S), using $[k : 2]$ adders followed by a carry-propagate adder (which may be omitted if the results can be used in redundant form).

The efficiency of the method essentially relies on how small θ can be for not-too-large values of the parameters M and k. An asymptotic study when p grows remains to be done, but we can see through some examples that it can work well.

B. Choosing adequate parameters

Consider the case, exemplified by Figure 3, where $M = 255$, $p = 24$, and $k = 5$. First, we have generated all (M, p, k)-friendly angles. Then, for any 7-bit number, less

than $\pi/2$:

$$x_0.x_1x_2x_3x_4x_5x_6$$

we have stored in a table T_0 the angle that is closest to $x_0.x_1x_2x_3x_4x_5x_61$. We will name $T_0(x_0, x_1, \cdots x_6)$ that closest value (in fact, we have also stored the corresponding values of a, b, and $1/\sqrt{a^2+b^2}$ in canonical form). The first and last entries of Table T_0 are given in Figure 4. The largest distance between $x_0.x_1x_2x_3x_4x_5x_61$ and $T_0(x_0, x_1, \cdots x_6)$ is $2^{-7.82181}$. Therefore, for any number $x = x_0.x_1x_2\cdots x_{24}$, x will be at a distance less that

$$2^{-7} + 2^{-7.82181} < 2^{-6.353}$$

from $T_0(x_0, x_1, \cdots x_6)$. We will choose

$$\hat{x} = T_0(x_0, x_1, \cdots x_6),$$

so that the corresponding value of θ will have absolute value less than $2^{-6.353}$: we can then use the bipartite method shown above with $k = q = 6$ for evaluating $\cos(\theta)$ and $\sin(\theta)$, where $\theta = x - \hat{x}$. The largest value of $1/\sqrt{a^2+b^2}$ stored in the table is around 0.01562.

C. Error bounds

Let us assume that $M = 255$, $p = 24$, and $k = 5$, and let us assume that we use the bipartite algorithm, with (as we did in the introduction) $j = q = 6$. From the error of the bipartite algorithm ($0.047 \cdot 2^{-24}$ for $\cos\theta$, and $0.0313 \cdot 2^{-24}$ for $\sin\theta$) one easily deduces that if a and b are the values selected in Table T_0, and if $z = 1/\sqrt{a^2+b^2}$, the difference between the exact and the computed value of $\cos(x)$ or $\sin(x)$ is upper-bounded by

$$2^{-25} + h(a, b, z),$$

where the 2^{-25} comes from the last rounding, and

$$h(a, b, z) = 0.047 \cdot 2^{-24} \cdot z \cdot (a+b) + (a + b \cdot 2^{-6}) \cdot 2^{-24} \cdot z.$$

The largest value of $h(a, b, z)$ for the 100 entries of the table is less than 1.049×2^{-24}, so that the total (absolute) error of the algorithm is less than 1.549×2^{-24}.

D. Critical path

Assuming $M = 255$, $p = 24$, and $k = 5$, we have on the critical path:

- lookup in a 7-address bit table (that contains 100 elements), to obtain $\hat{x}$, a, and b;
- computation of $\theta = \hat{x} - x$;
- bipartite method for $\sin(\theta)$ and $\cos(\theta)$: 1 table lookup in a 12-address-bit table followed by an addition;
- multiplication by a (and b: done in parallel), followed by one addition (e.g., for $a\cos(\theta) - b\sin(\theta)$). This can be done with 4 carry save additions in the critical path followed by one carry-propagate addition;
- final multiplication by z: 3 carry-save additions followed by one carry-propagate addition.

In the proposed implementation, we consider variants in the reductions and use of redundant representations until the end of computation thus avoiding carry-propagate additions until the end of the algorithm.

IV. Implementation of the proposed method

An implementation of the method is shown in Fig. 5. It consists of several modules which we now describe in some detail. The modules correspond to the steps of the algorithm. The argument is $x = x_0.x_1 \ldots x_{24}$. The outputs are $SIN = \sin(x)$ and $COS = \cos(x)$: 24 bits, rounded. For $(M, p, k) = (255, 24, 5)$, the main modules are characterized as follows.

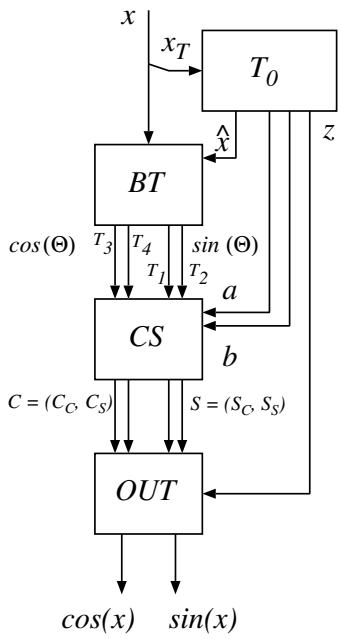

Figure 5. Overall block diagram.

Module T_0: A look-up table (Fig. 6), requiring 100 words, is addressed by the 7-bit truncated argument $x_T = (x_0, x_1, \ldots, x_6)$. The memory stores: a, b (8-bit wide, stored recoded to radix-4 with digit set {-2,-1,0,1,2} using a total of 2+4×3= 14 bits for each), the angle $\hat{x}$ (24 bits), and z represented with five fields $F0, F1, F2, F3, F4$, each field (except the first one) consisting of a sign and an index of 4 bits, for a total of 2 + 4×5 =22 bits. The index with value j indicates the position of a non-zero z_j bit. Consequently, a field Fi defines a signed multiple $Mi = C \times 2^{-j}$, for $i = 0, 1, 2, 3, 4$, generated in Module OUT with barrel shifters and complementers. Each field has an offset $2^{-offset}$ so that the shifts within a field are relative to the offset. For example, field $F0$ covers positions 6, 7, and 8. Consequently, its offset is 6 and relative shifts within the field are 0, 1, and 2. The offsets for fields $F1,\ F2,\ F3,\ F4$

$x_0x_1\cdots x_t$	a	b	$\hat{x}$	$\lvert\hat{x}-x_0x_1\cdots x_t1\rvert$	z
0	128	1	$.7812341e-2$	$.159e-6$	$0.00000010000000000000\overline{1}000000000$
1	128	3	$.2343321e-1$	$.429e-5$	$0.00000010000000000\overline{1}00\overline{1}000000001$
10	130	5	$.3844259e-1$	$.620e-3$	$0.0000001000\overline{1}0000\overline{1}0000000010100$
11	255	14	$.5484690e-1$	$.159e-3$	$0.00000001000000001010001\overline{1}00001000$
100	204	14	$.6852002e-1$	$.179e-2$	$0.00000001010000001000000000100010$
101	255	22	$.8606141e-1$	$.124e-3$	$0.00000001000000000001001\overline{1}01000000$
110	226	23	.1014207	$.142e-3$	$0.00000001001000001000001\overline{1}00000001$
111	127	15	.1175656	$.378e-3$	$0.00000010000000001000\overline{1}000001010$
1000	255	34	.1325515	$.261e-3$	$0.00000001000000001\overline{1}01\overline{1}0000000\overline{1}00\overline{1}00$
1001	126	19	.1496660	$.123e-2$	$0.00000010000000100101000000001\overline{1}$
1010	245	40	.1618374	$.223e-2$	$0.00000001000010000000000\overline{1}010\overline{1}0$
1011	247	45	.1802098	$.522e-3$	$0.00000001000001010000100000000\overline{1}$
1100	251	50	.1966293	$.132e-2$	$0.00000001000000000001001\overline{1}01000000$
1101	125	27	.2127318	$.179e-2$	$0.00000010000000001000\overline{1}000001010$
1110	247	57	.2267988	$.236e-3$	$0.00000001000000101000100011000000$
1111	221	55	.2439137	$.173e-2$	$0.00000001001000000\overline{1}0001000000\overline{1}000$
10000	231	62	.2622183	$.441e-2$	$0.00000001000100100000001000000001\overline{1}$
10001	246	69	.2734610	$.235e-4$	$0.00000001000000001000001000\overline{1}0001$
10010	162	48	.2880554	$.101e-2$	$0.0000001\overline{1}0000100000\overline{1}000000001000$
...	...	...	...	...	...
1010010	38	131	1.288468	$.594e-3$	$0.00000010001\overline{1}0000010001\overline{1}0000000\overline{1}0$
1010011	67	247	1.305915	$.123e-2$	$0.00000001000000000001010\overline{1}0000001$
1010100	63	248	1.322026	$.171e-2$	$0.00000001000000000010000\overline{1}000010$
1010101	60	249	1.334340	$.160e-2$	$0.00000001000000000\overline{1}00000\overline{1}0001\overline{1}0$
1010110	56	252	1.352127	$.565e-3$	$0.00000001000000\overline{1}000\overline{1}0000\overline{1}00\overline{1}00000$
1010111	53	253	1.364296	$.289e-2$	$0.00000001000000\overline{1}01\overline{1}000100001000000$
1011000	44	237	1.387232	$.442e-2$	$0.00000001000100000\overline{1}0000001010000$
1011001	39	224	1.398417	$.204e-4$	$0.00000001001000000100\overline{1}\overline{1}000100000$
1011010	39	251	1.416650	$.259e-2$	$0.00000001000000100000000100\overline{1}\overline{1}010$
1011011	35	246	1.429468	$.219e-3$	$0.00000001000010000\overline{1}000000001\overline{0}\overline{1}00$
1011100	32	254	1.445472	$.160e-3$	$0.00000001000000000000000\overline{1}0000000000$
1011101	25	227	1.461106	$.169e-3$	$0.00000001001000\overline{1}0000\overline{1}0000100000$
1011110	24	255	1.476955	$.393e-3$	$0.00000001000000000\overline{1}00000\overline{1}0001\overline{0}\overline{1}0$
1011111	19	254	1.496132	$.394e-2$	$0.00000001000000010101\overline{0}\overline{1}000000000$
1100000	16	255	1.508133	$.321e-3$	$0.00000001000000001000000000\overline{1}000\overline{1}$
1100001	3	64	1.523956	$.518e-3$	$0.0000010000000000\overline{1}00\overline{1}00000100\overline{0}\overline{1}0$
1100010	2	64	1.539556	$.494e-3$	$0.0000010000000000\overline{1}0000000000101$
1100011	2	126	1.554925	$.237e-3$	$0.00000010000010000001000000\overline{1}000\overline{1}$
1100100	0	254	1.570796	$.484e-3$	$0.00000001000000100000001000000100$

Figure 4. The first and last values of Table T_0, for $M = 255$, $p = 24$, and $k = 5$. There are 100 entries in the table. For each entry $(x_0, x_1, x_2, x_3, x_4, x_5, x_6)$, the distance between $\hat{x}$ and $x_0.x_1x_2\cdots x_61$ is less than $2^{-7.82181}$.

are 9, 11, 15, and 18, respectively. Shift value j selects a multiplicand shifted right j places using a barrel shifter. The offsets are done by wiring. Note that any field can represent any bit position, i.e., the order is irrelevant. This is used to minimize the width of the fields. We illustrate the z fields (before encoding) for the first 5 words of Table T_0 in Fig. 7. The field value (0,0), (0,j), and (1,j) indicate that no multiple, a positive multiple, or a negative multiple is selected, respectively.

The total width of T_0 word is 2×14+24+22 = 64 bits and the size of the memory is $2^7 \times 2^6 = 8K$ bits. The effective use is less since only 100 out of 128 words are needed.

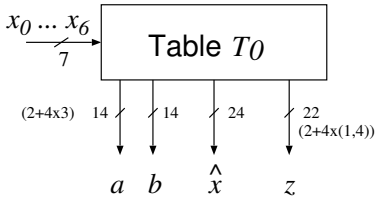

Figure 6. Table T_0.

$Address$	$F0$	$F1$	$F2$	$F3$	$F4$
0	$(0,7)$	$(1,22)$	$(0,0)$	$(0,0)$	$(0,0)$
1	$(0,7)$	$(1,22)$	$(1,19)$	$(0,31)$	$(0,0)$
2	$(0,7)$	$(1,13)$	$(1,18)$	$(0,27)$	$(0,29)$
3	$(0,8)$	$(1,17)$	$(1,19)$	$(1,23)$	$(0,28)$
4	$(0,8)$	$(0,17)$	$(0,10)$	$(0,26)$	$(0,30)$

Figure 7. Illustration of fields. (The fields store binary encodings relative to offsets.)

Module BT: It consists of four $(2^{12} \times 28 - bit)$ tables, for a total of $16K \times 28$ bits, which are used to produce $\sin(\theta) = (T_1, T_2)$ and $\cos(\theta) = (T_3, T_4)$ in redundant form according to Eqs. (4, 5). This is shown in Fig. 8. Leaving the outputs in redundant form reduces the overall delay at expense of doubling the number of rows to be reduced in Module CS. Alternatively, two CPAs can be used to produce $\sin(\theta)$ and $\cos(\theta)$ in conventional form. A direct table lookup would require two tables of $2^{24} \times 28$ bits for a total of $32M \times 28$ bits which is 2^{11} times larger than the tables in the proposed method.

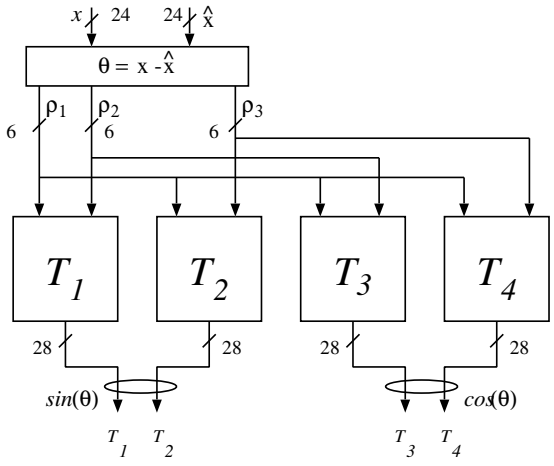

Figure 8. Module BT: Bipartite tables Table T_1, T_2, T_3, T_4.

Module CS: It produces

$$\begin{aligned} C &= a\cos(\theta) - b\sin(\theta) \\ &= (T_3, T_4) \times a - (T_1, T_2) \times b = (C_C, C_S) \\ S &= b\cos(\theta) + a\sin(\theta) \\ &= (T_3, T_4) \times b + (T_1, T_2) \times a = (S_C, S_S) \end{aligned}$$

in redundant form where index C and S denotes carry and sum bit-vectors. The short operands a, b are stored in T_0, recoded in 5 radix-4 digits {-2,-1,0,1,2}. Module BT produces $\cos(\theta)$ and $\sin(\theta)$ in redundant form (T_3, T_4) and (T_1, T_2), respectively. The multiplication $a\cos(\theta) = (T_3, T_4) \times a$ is performed as reductions by two [5:2] adders in parallel followed by a [4:2] adder. Similarly, for $\underline{s}in(\theta)$, $\underline{c}os(\theta)$, and $a\sin(\theta)$. Then the outputs $C = (C_C, C_S)$ and $S = (S_C, S_S)$ in redundant form are obtained with another [4:2] adder, respectively. The overall scheme of the module is shown in Fig. 9.

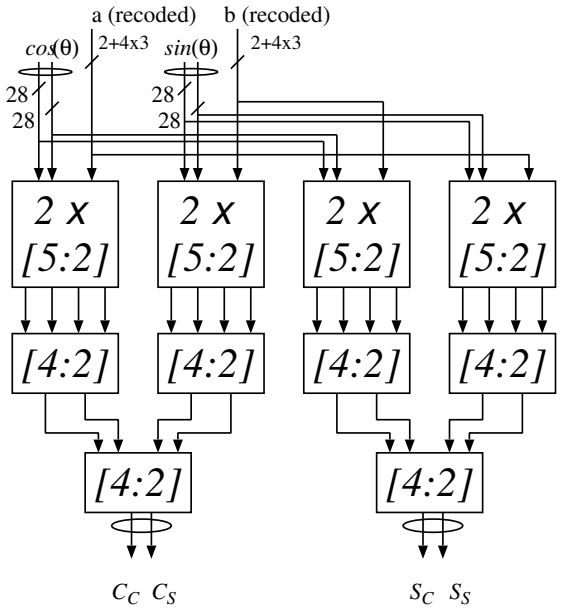

Figure 9. Module CS: Computation of $C = (C_C, C_S)$ and $S = (S_C, S_S)$.

Module OUT: This module produces the final results $\cos(x) = C \times z$ and $\sin(x) = S \times z$, rounded to 24 bits. Instead of performing multiplications of C and S by z by multipliers, we propose to use reduction networks on up to five shifted/negated multiples of C and S, respectively. The shift distances are stored in T_0 as fields $F0, \ldots, F4$. Multiples of C (S) to two bit-vectors, followed by fast carry propagate adders (CPA). The inputs are in redundant form, produced by Module CS as $C = (C_C, C_S)$ and $S = (S_C, S_S)$. So we perform $C_C \times z$ and $C_S \times z$ to obtain redundant output of $cos(x)$ as shown in Fig 10. There are four blocks identical to the block shown in the figure. Each block consists of one 2-stage barrel shifter $BS2$ that shifts (with respect to the wired-in offset) 0,1, or 2 positions to the right. The four remaining barrel-shifters $BS2, \ldots, BS4$ are four-stage shifters, shifting 0, 1, ..., up to 15 positions relative to the offset of a field. The barrel shifters consists of 2-input multiplexers for each position in each stage. These are controlled directly (no decoding) by the index bits in the corresponding field. If a field indicates a negative shifted multiple, the corresponding complementer $CMPL$ is activated and the related LSB carries are inserted into the reduction network. The reduction is performed using [5:2] adders, followed by a [4:2] adder. If the result is needed in a conventional form, a CPA is used.

We now estimate the delay of the implementation. Following the path through the main modules, we have

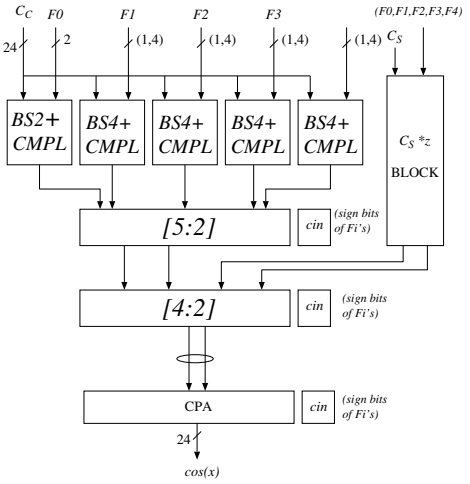

Figure 10. Computation of $\cos(x)$. BSv denotes v-level barrel shifter. (Similar block computes $\sin(x)$).

$$\begin{aligned} T_{COS-SIN} &\approx t_{T_0} + t_{BT} + t_{CS} + t_{OUT} + t_{CPA} \\ &\approx t_{ROM}(2^7) + t_{ROM}(2^{12}) \\ &\quad +(t_{mgen} + t_{buff} + t_{5:2} + 2 \times t_{4:2}) \\ &\quad +(t_{buff} + 4 \times t_{mux2} + t_{XOR} + t_{5:2} + t_{4:2}) \\ &\quad +t_{CPA} \end{aligned}$$

An optimized [5:2] adder has a critical path equivalent $4t_{XOR} \approx 2t_{FA}$ [9]. A [4:2] adder has a critical delay of about $3t_{XOR} \approx 1.5t_{FA}$. We assume that t_{buff}, t_{mgen} and t_{mux2} are roughly $0.5t_{FA}$, $t_{ROM}(2^7) \approx t_{FA}$, and $t_{ROM}(2^{12}) \approx 2t_{FA}$. The carry-propagate adder of parallel-prefix type over 24 bits is estimated to have a delay of $4t_{FA}$. Then we get

$$\begin{aligned} T_{COS-SIN} &\approx t_{FA}(1 + 2 + 0.5 + 0.5 + 2 + 2 \times 1.5 \\ &\quad +0.5 + 4 \times 0.5 + 0.5 + 2 + 1.5) + 4 \\ &= 19.5t_{FA} \end{aligned}$$

A rough comparison with a fully-unfolded CORDIC scheme [10] for computing $\cos$ and $\sin$ functions, using double rotations, and having a stage delay of about $2T_{FA}$, indicates that the proposed method is roughly twice as fast for 24-bit case. We make no comments on relative cost at this time.

V. Summary and future work

We have introduced a new way of reducing the input argument, so that a bipartite method can be employed for evaluating trigonometric functions with a reasonably large precision. We have investigated its properties in special cases. We have illustrated the methods applicability by describing a fully combinational scheme for computing 24-bit $\cos$ and $\sin$ functions. The method can be implemented in several alternative ways to achieve desired delay-cost trade-offs. More detailed implementations and their realizations in particular technologies remain to be done. A preliminary rough comparison with a CORDIC approach indicates a potential speedup of 2. In general, we would like to be able to predict values of M and k, as well as number of bits of address for the first table T_0, that will be of interest for a given precision p. This requires solving several theoretical problems such as predicting the gap between two consecutive friendly angles, which is linked to the probability that a p-bit chain can be recoded into canonical form with at most k nonzero digits. We plan to address these problems in the near future.

References

[1] D. A. Sunderland, R. A. Strauch, S. W. Wharfield, H. T. Peterson, and C. R. Cole, "CMOS/SOS frequency synthesizer LSI circuit for spread spectrum communications," *IEEE Journal of Solid State Circuits*, vol. SC-19, no. 4, pp. 497–506, 1984.

[2] D. D. Sarma and D. W. Matula, "Faithful bipartite ROM reciprocal tables," in *Proceedings of the 12th IEEE Symposium on Computer Arithmetic (ARITH-12)*, Knowles and McAllister, Eds. IEEE Computer Society Press, Los Alamitos, CA, Jun. 1995, pp. 17–28.

[3] M. J. Schulte and J. Stine, "Symmetric bipartite tables for accurate function approximation," in *Proceedings of the 13th IEEE Symposium on Computer Arithmetic*, I. T. Lang, J. Muller, and N. Takagi, Eds. IEEE Computer Society Press, Los Alamitos, CA, 1997.

[4] M. J. Schulte and J. E. Stine, "Accurate function evaluation by symmetric table lookup and addition," in *Proceedings of the IEEE International Conference on Application-Specific Systems, Architectures and Processors (Zurich, Switzerland)*, Thiele, Fortes, Vissers, Taylor, Noll, and Teich, Eds. IEEE Computer Society Press, Los Alamitos, CA, 1997, pp. 144–153.

[5] F. de Dinechin and A. Tisserand, "Multipartite table methods," *IEEE Transactions on Computers*, vol. 54, no. 3, pp. 319–330, Mar. 2005.

[6] D. W. Matula and M. T. Panu, "A prescale-lookup-postscale additive procedure for obtaining a single precision ulp accurate reciprocal," in *IEEE Symposium on Computer Arithmetic*, 2011, pp. 177–183.

[7] G. W. Reitwiesner, *Binary arithmetic*, ser. Advances in Computers, 1960, vol. 1, pp. 231–308.

[8] A. D. Booth, "A signed binary multiplication technique," *Quarterly Journal of Mechanics and Applied Mathematics*, vol. 4, no. 2, pp. 236–240, 1951, reprinted in E. E. Swartzlander, *Computer Arithmetic*, Vol. 1, IEEE Computer Society Press, Los Alamitos, CA, 1990.

[9] R. Menon, and D. Radhakrishnan,"High performance 5 : 2 compressor architectures", IEE Proceedings - Circuits, Devices and Systems, 153(5):447-452, 2006.

[10] J. Duprat and J.-M. Muller, "The CORDIC algorithm: New results for fast VLSI implementation." IEEE Trans. Computers, 42(2):168-78,1993.