Efficient algorithms for binary logarithmic conversion and addition

Y. Wan and C.L. Wey

Abstract: The logarithm number system is an attractive alternative to the conventional number systems when data needs to be manipulated at a very high rate over a wide range. The major problem is deriving logarithms and antilogarithms quickly and accurately enough to allow conversions to and from conventional number representations. Efficient algorithms that convert the conventional number representation to binary logarithm representations are proposed. The algorithms adopt a factorisation approach to reduce the look-up table size and a nonlinear approximation method to reduce the computational complexity. Simulation results on IEEE single precision (24 bits) conversion are presented, and the conversion requires only one ROM table with $2^{13} \times 26$ bits, one with $2^{13} \times 14$ bits, and one with $2^{13} \times 5$ bits, or a total of 360 kbits.

1 Introduction

The logarithm number system (LNS) is an attractive alternative to the conventional number systems when the data needs to be manipulated at a very high rate over a wide range. The LNS can simplify multiplication, division, roots, and powers [1, 2]. When logarithms are used, multiplication and division are reduced to addition and subtraction, respectively, and powers and roots are reduced to multiplication and division, respectively. On the other hand, the add and subtract operations become more complex. Another major problem is deriving logarithms and antilogarithms quickly and accurately enough to allow conversions to and from the conventional number representations. These conversions always involve approximations, resulting in inaccuracies. Therefore, binary logarithms can be useful only in arithmetic units dedicated to special applications, where very few conversions are required but many multiplications and divisions are executed, e.g. real-time digital filters.

The objective of this paper is to develop an efficient algorithm that converts the conventional number representation to a binary logarithm representation. More specifically, given a fractional binary number X which can be normalised as

$$X = .1x_1x_2..x_n^*2^{-k} = 1.x_1x_2..x_n^*2^{-(k+1)}$$

where it is assumed that X contains k's 0 right after the radix point $k \geq 0$. Thus, $\log_2(X) = \log_2(1+x) - (k+1)$, where $x = .x_1x_2..x_n$. For simplicity of notation, we use "log" instead of "$\log_2$" in the following discussion. This study is to efficiently and accurately evaluate log $(1+x)$. Therefore the problem can be restated as: given an n-bit fractional binary number $x = .x_1x_2..x_n$, i.e., $x \in [0,1)$, it is to generate a precise binary logarithmic value of $(1+x)$, i.e.

$$y = \log(1+x) \qquad (1)$$

where $y = .y_1y_2..y_n$ is also an n-bit fractional binary number, i.e. $y \in [0,1)$. On the other hand, for logarithm addition, let $a = \log A$ and $b = \log B$, i.e. $A = 2^a$ and $B = 2^b$. Suppose that $a < b$, i.e. $0 < r = (a/b) < 1$, then $a + b = a[1 + (a/b)] = a\ (1+r)$, and $\log\ (a+b) = \log\ [a(1+r)] = \log\ (a) + \log\ (1+r)$. The problem of logarithm addition can be stated as: given a n-bit binary number $r \in [0,1)$, it is to generate the binary logarithmic value of $(1+r)$. Therefore the efficient algorithm developed for conversion in eqn. 1 can also be used for the logarithmic addition problem.

A number of binary logarithm conversion algorithms [1–13] have been developed. The existing conversion algorithms can be roughly classified into two categories: *look-up table approach* and *computation approach*. The former approach adopts various approximation schemes such as linear approximation methods so that moderate size look-up table can be implemented [1]. The look-up table approach generally has the advantage of fast speed. However, the look-up table size often increases drastically as the word length increases. The latter approach converts the binary logarithm with multiplication and/or division operations [2, 3]. However, the applicability of such an approach is limited by its slow operation speed. The trade-off between both approaches is speed and precision. Thus, a feasible solution is the combination of both approaches. This paper presents efficient conversion algorithms which adopt a factorisation approach to reduce the look-up table size and an nonlinear approximation method to reduce the computational complexity.

2 Algorithm development

The basic problem in the look-up table approach is the use of a simple linear function $y = x$ to approximate $y = \log\ (1+x)$, as shown in Fig. 1 [1]. Thus, the approximation error is $\Delta = \log\ (1+) - x$, and the maximal approximation error $\Delta_{max} = 0.086071$ occurs at $x = 0.442695$. That

IEE Proceedings online no. 19990530
DOI: 10.1049/ip-cdt:19990530

Paper first received 22nd September 1997, and in revised form 22nd October 1998

The authors are with the Department of Electrical Engineering, Michigan State University, East Lansing, MI 48824-1226
E-mail: wey@egr.msu.edu

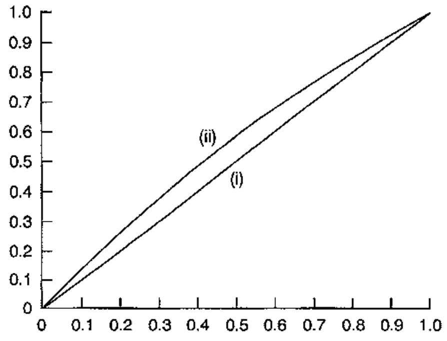


Fig. 1 *Linear approximation [1]*
(i) $y = x$
(ii) $y = \log(1 + x)$

approach needs 23 different partitioned groups for eight-bit word length and a ROM table with eight inputs and five outputs was used [1]. However, as word length increases, the number of partitioned groups increases exponentially and thus requiring a very large ROM table.

In this study, efficient algorithms that generate a precise binary logarithm $\log(1+x)$ for an n-bit fractional binary number $x = .x_1x_2..x_n$ is developed. Without loss of generality, n is assumed to be an even number, i.e. $n = 2m$. The term $(1+x)$ can be factorised as

$$\begin{aligned}1 + x &= 1 + .x_1x_2..x_n = 1 + .x_1x_2..x_mx_{m+1}x_{m+2}..x_{2m}\\ &= (1 + .x_1x_2..x_m)(1 + .00\ldots 0c_1c_2..c_m)\end{aligned}$$

where

$$.c_1c_2..c_m = .x_{m+1}x_{m+2}..x_{2m}/(1 + .x_1x_2..x_m) \quad (2)$$

Let $a = .x_1x_2..x_m$, $b = .x_{m+1}x_{m+2}..x_{2m}$, $b' = b \bullet 2^{-m}$, $c = .c_1c_2..c_m$, and $c' = c \bullet 2^{-m}$, the term $(1+x)$ can then be expressed as

$$1 + x = 1 + a + b' = (1 + a)(1 + c') \quad (3)$$

where, by eqn. 2,

$$c = b/(1 + a) \quad (4)$$

Therefore by eqn. 3, the logarithmic value of $(1+x)$ is derived as

$$\log(1 + x) = \log(1 + a) + \log(1 + c') \quad (5)$$

A ROM look-up table approach has been an efficient way to generate binary logarithms [2]. For generating the logarithm of an n-bit binary number, a look-up table is implemented with a 2^n-by-n ROM. By eqn. 5, two look-up tables are used: one table, ROM1, for $\log(1 + .x_1x_2..x_m)$ and the other, ROM2, for $\log(1 + .00\ldots 0c_1c_2..c_m)$, where each table is implemented with a 2^m-by-n ROM. In other words, instead of using 2^{2m}-by-n ROM in the conventional implementation, this approach uses two 2^m-by-n ROMs, i.e. $2^{(m-1)}n$ bits are reduced. For $n = 16$ and $m = 8$, the ROM table size is reduced from 2^{16}-by-16, or 1M bits, to two 2^8-by-16, or 8k bits. The reduction is significant. Table 1 summarises the ROM size reduction for various values of n. However, their approach requires the computation of c, as in eqn. 4, which involves a slow division process. As a result, the speed improvement gained by the use of look-up tables for eqn. 5 may be offset by the slow division process. Therefore attempts are made to avoid the slow division process for improving the speed performance. Taking the advantage of existing ROM1, i.e. the ROM table for $\log(1 + .x_1x_2..x_m)$, the slow division process can be simplified. More specifically, by eqn. 4, c can be expressed as

$$\begin{aligned}c &= b/(1 + a) = (1 + b)/(1 + a) - 1/(1 + a)\\ &= 2^{\log(1+b)-\log(1+a)} - 2^{-\log(1+a)} = 2^{B-A} - 2^{-A} \quad (6)\end{aligned}$$

where both $A = \log(1 + a)$ and $B = \log(1 + b)$ can be quickly evaluated from ROM1. Therefore the only problem remaining is the evaluation of the function 2^z.

Interestingly, both functions $(2^z - 1)$ and $[1 - \log(2 - z)]$, as shown in Fig. 2, are very close to each other when $0 \le z < 1$, i.e. $(2^z - 1) \simeq [1 - \log(2 - z)]$, or

$$2^z \simeq [2 - \log(2 - z)], \quad \text{for } z \in [0, 1) \quad (7)$$

In other words, the function 2^z can be approximated by the nonlinear function $[2 - \log(2 - z)]$ for all $z \in [0,1)$. The value of $\log(2 - z)$, $= \log[1 + (1 - z)]$, can be found from ROM1 because $(1 - z) \in (0,1)$ for all $z \in [0,1)$. Fig. 3 plots the approximation errors, i.e. $[2^z - 2 + \log(2 - z)]$, for all $z \in [0,1)$. Results show that the maximum error is 0.004198 which is much better than 0.086071 with a linear function approximation in [1]. Note that the maximum approximation error of 0.004198 is equivalent to seven-bit accuracy. To further reduce the maximum approximation error, we use the function

$$[1 - \log(2 - z) + 2^{-12} + 2^{-13}] \quad (8)$$

to approximate $2^z - 1$. As such, the maximum approximation error can be reduced as 0.0038038 which is less than

Table 1: Comparison

n	m	$2^n \times n$ (bits)	$2 \times 2^m \times n$ (bits)
10	5	10k	640
12	6	48k	1.5k
14	7	112k	3.5k
16	8	1M	8k
18	9	4.5M	18k
20	10	2M	40k

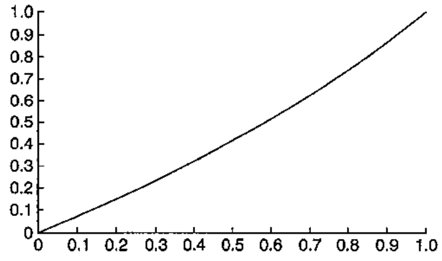


Fig. 2 *Nonlinear approximation plots for* $(2^z - 1)$ *and* $[1 - \log(2 - z)]$

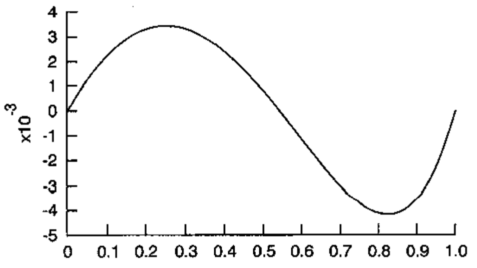


Fig. 3 *Approximation errors for* $(2^z - 1)$ *and* $[1 - \log(2 - z)]$

2^{-8}. Thus, $m=8$ is the appropriate value suggested with this implementation.

Since the approximation to 2^z in eqn. 7 is valid for all $z \in [0,1)$, the exponents in both terms of eqn. 6 must be ranged between 0 and 1. Therefore the following lemma holds.

Lemma 1: The value c can be derived as follows

$$c = \begin{cases} 2^{B-A} - 2^{1-A}/2 & \text{if } A < B \quad (9a) \\ 2^{1+B-A}/2 - 2^{1-A}/2 & \text{if } A \geq B \quad (9b) \end{cases}$$

Proof: Both $A=\log\ (1+a)$ and $B=\log\ (1+b)$ range between 0 and 1. If $A<B$, the term $B-A$ in eqn. 6 ranges between 0 and 1. Since the term $-A$ is not within the range, the term 2^{-A} is modified as $(2^{1-A})/2$ so that $1-A$ lies within the range. Hence, eqn. 9a results for $A<B$. Similarly, for $A \geq B$, the term 2^{B-A} is modified $(2^{1+B-A})/2$ so that $(1+B-A)$ lies within the range. With $2^{-A}=(2^{1-A})/2$, eqn. 9b holds.

Let $p=1-B+A$. Clearly, $p<1$ if $A<B$, and $p \geq 1$, otherwise. Let $p=p^0+p^f$, where p^0 and p^f are the integral and fractional parts of p, respectively. Thus, the following theorem results:

Theorem 1: The value c can be evaluated as follows

$$c = \begin{cases} (1-P) + A'/2 & \text{if } p^0 = 0 \quad (10a) \\ (-P) + A'/2 & \text{if } p^0 = 1 \quad (10b) \end{cases}$$

where $P=\log\ (1+p^f)$ and $A'=\log\ (1+A)$.

Proof: If $A<B$, then $p<1$, or $p^0=0$. By eqn. 7, both terms in eqn. 9a can be approximated as follows

$$2^{1-A} \simeq 2 - \log(1+A) = 2 - A' \text{ and } 2^{B-A} \simeq 2$$
$$- \log[2-(B-A)] = 2 - \log(1+p) = 2 - P$$

Therefore by eqn. 9a, $c=2^{B-A}-2^{1-A}/2 \simeq (2-P)-(2-A')/2$, or $c=(1-P)+A'/2$. On the other hand, if $A \geq B$, then $p \geq 1$, or $p^0=1$, the term 2^{1+B-A} in eqn. 9b is approximated as

$$2^{1+B-A} \simeq 2 - \log[2-(1+B-A)] = 2 - \log p$$
$$= 2 - \log(1+p^f) = 2 - P$$

Therefore $c=2^{1+B-A}/2 - 2^{1-A}/2 \simeq (2-P)/2-(2-A')/2$, or $c=(-P)/2+A'/2$

Algorithm 1 summarises the procedure described, where the comparison of B and A is not necessary in this implementation.

```
Algorithm 1
Step 0.  Generate look-up tables $2^m \times n$,
         ROM1($x$) for log $(1.x_1x_2..x_m)$ and
         ROM2($c$) for log $(1.00\ldots 0c_1c_2..c_m)$
         $x=.x_1x_2..x_mx_{m+1}x_{m+2}..x_{2m}=a+b \bullet 2^{-m}$;
         $c=.c_1c_2..c_m$
Step 1.  $A=\log\ (1+a)=$ROM1($a$) and $B=\log$
         $(1+b)=$ROM1($b$);
Step 2.  calculate $c$ from theorem 1
  2.1  $p=1-B+A=p^0+p^f$.
  2.2  $P=$ROM1($p^f$); $A'$ROM1($A$);
  2.3  IF $p^0=0$, THEN $c=1-P$; ELSE
         $c=(-P)/2$;
  2.4  $c=c+A'/2$.
Step 3.  $C=\log\ (1+c \bullet 2^{-m})=$ROM2($c$);
  $\log\ (1+x)=\log\ (1+a)+\log\ (1+c \bullet 2^{-m})=$
  $A+C$
```

The following examples illustrate the stepwise procedure of algorithm 1.

Example 1: Consider a 16-bit binary number $x=.10111011\ 11101010$.

$$a = .10111011 \text{ and } b = .11101010$$

From ROM1,

$$A = \log(1+a) = .11001011$$
$$A' = \log(1+A) = .11011000$$
$$B = \log(1+b) = .11110000$$

Thus $p=1-B+A=1-\log\ (1+b)+\log\ (1+a)=.11011011$. Note that $p^0=0$ and $p^f=p$. Since $p^0=0$, $c=.10001000$. By ROM2,

$$C = \log(1+c \bullet 2^{-m}) = \text{ROM2}(c) = .00000000\ 11000100$$

This results in

$$\log(1+x) = .1100101101001110$$

Note that the actual value is

$$\log(1+x) = .1100101101001101$$

The approximation error is 1 ULP (unit in the last position).

Example 2: Consider a 16-bit binary number $x=.10110100\ 01011011$.

$$a = .10111011 \text{ and } b = .01011011$$

From ROM1,

$$A = \log(1+a) = .11000101$$
$$A' = \log(1+A) = .11010011$$
$$B = \log(1+b) = .01110000$$

Since $p=1-\log(1+b)+\log(1+a)=1.01010101$, we have $p^0=1$ and $p^f=p-1$. Since $p^0=1$, $c=.00110100$. By ROM2,

$$C = \log(1+c \bullet 2^{-m}) = \text{ROM2}(c) = .00000000\ 01001011$$

This results in

$$\log(1+x) = .1100010011110011$$

The actual value is

$$\log(1+x) = .1100010011110101$$

The approximation error is -2 *ULP*s.

To demonstrate the effectiveness of algorithm 1, approximation errors have been analysed by comparing the approximated value with the actual value of $\log(1+x)$. This experiment assumes that $m=8$ and $n=16$, and $x=.x_1x_2..x_{16}$. All possible $(2^{17}-1)$ combinations of $x_1x_2..x_{16}$ are simulated for both approximated values and the actual values. Let *DIFF* denote the absolute value of

Table 2: Simulation results

DIFF (ulp)	No. of cases	Percentage
0	22426	34.2%
1	31388	47.9%
2	10417	15.0%
3	1278	11.95%
4	27	0.041%
5	0	0%

the difference between the approximated value and the actual value. Simulation results are shown in Table 2. Results show that the maximum approximation error is 4 ULP, or 2^{-14}. The number of combinations with $DIFF = 2^{-14}$ is 27 out of $(2^{17} - 1)$, which is 0.041%. Because of the nonlinear approximation error and the truncation error of the values in the look-up table, the computation of c can have an error of up to 1 to 2 ULP. The error can be magnified by a factor of $(1 + a)$, as shown in eqn. 3, and be further carried into the final stage of taking algorithm from the look-up tables. Results also show that all $DIFF < 2^{-13}$. This concludes that the developed algorithm achieves an accuracy of 13 bits, where two tables ROM1 and ROM2 are used and each has a size of $2^8 \times 16$, or 4k bits. Since $\log(1 + x) < 2x$, for all $x > 0$, the first seven bits of the output of ROM2 must be 0 s. So instead of 4k bits ROM2 needs only $2^8 \times 9$, or 2.25k bits.

3 Improvement

The slow division process in eqn. 4 is improved by approximating 2^z using an existing ROM table. However, the nonlinear approximation approach developed in this algorithm is limited to $m = 8$ and the accuracy of $\log(1 + x)$ to 13 bits. The accuracy can be improved by either selecting better nonlinear approximation functions, or alternatively, using additional ROM table for the values of 2^z, where $z \in [0,1)$. The former alternative seems difficult due to the fact that any nonlinear approximation function to the function 2^z itself needs to be effectively evaluated first. In this study, a connection between 2^z and available logarithmic function is discovered and hence there is not much hardware overhead involved. But in general it is not easy to find such an approximation function.

Consider the following approximation:

$$2^z \simeq [2 - \log(2 - z)] + \Delta(z) \text{ for } z \in [0, 1) \qquad (11)$$

where $\Delta(z)$ is the first m bits of the difference between 2^z and $[2 - \log(2 - z)]$ in eqn. 7. Thus, the difference can be stored into a $2^m \times m$ ROM table, ROM3. Based on the approximation in eqn. 11, the following theorem results:

Theorem 2: The value c can be evaluated as follows

$$c = \begin{cases} [(1 - P) + \Delta(1 - p^f)] + [A' + \Delta(1 - A)]/2 \\ \text{if } p^0 = 0 & (12a) \\ [-P + \Delta(1 - p^f)]/2 + [A' + \Delta(1 - A)]/2 \\ \text{if } p^0 = 1 & (12b) \end{cases}$$

where $P = \log(1 + p^f)$ and $A' = \log(1 + A)$.

Proof: For $p^0 = 0$, both terms in eqn. 9a can be approximated by eqn. 11 as follows

$$2^{1-A} \simeq 2 - \log[2 - (1 - A)] + \Delta(1 - A) = 2 - \log(1 + A)$$
$$+ \Delta(1 - A) = (2 - A') + \Delta(1 - A) \text{ and}$$
$$2^{B-A} \simeq 2 - \log[2 - (B - A)] + \Delta(B - A) = [2 - \log(1 + p)]$$
$$+ \Delta(1 - p) = (2 - P) + \Delta(1 - p^f)$$

By eqn. 9a, $c = 2^{B-A}\,2^{1-A}/2 \simeq [(2 - P) + \Delta(1 - p^f)] - [(2 - A') + \Delta(1 - A)]/2 = [(1 - P) + \Delta(1 - p^f)] + [A' + \Delta(1 - A)]/2$. Similarly, for $p^0 = 1$, the term 2^{1+B-A} in eqn. 9b is approximated by eqn. 11 as

$$2^{1+B-A} \simeq 2 - \log[2 - (1 + B - A)] + \Delta(1 + B - A)$$
$$= (2 - P) + \Delta(2 - p) = (2 - P) + \Delta(1 - p^f)$$

Therefore $c = 2^{1+B-A}/2 - 2^{1-A}/2 \simeq [(2 - P) + \Delta(1 - p^f)]/2\text{-}[(2 - A') + \Delta(1 - A)]/2$, and eqn. 12b holds.

Algorithm 2 summarises the procedure for this implementation.

```
Algorithm 2
Step 0.  (ROM1($x$) and ROM2($c$) are the same as in
         algorithm 1)
         ROM3($x$) for $2^x - 2 + \log(2 - x)$ size of
         $2^m \times m$.
Step 1.  (Same as algorithm 1)
Step 2.  calculate $c$
   2.1   $p = 1 - B + A = p^0 + p^f$.
   2.2   $P = $ROM1 $(p^f)$; $A' = $ROM1$(A)$;
   2.3   IF $p^0 = 0$, THEN
         $c = 1 - P + $ROM3$(1 - p^f) +$
         ROM3$(1 - A)/2$;
         ELSE $c = (-P + $ROM3$(1 - p^f)$
         $+$ROM3$(1 - A))/2$;
  2.4   $c = c + A'/2$.
Step 3.  (Same as algorithm 1)
```

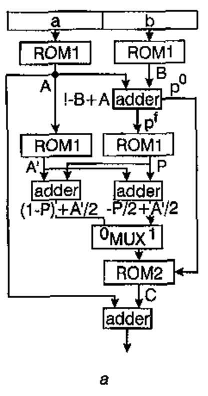
Algorithm 2 has been developed and simulated with IEEE single precision conversion (23-bit), where 24-bits are employed with two internal guarding digits, making the total number of bits in a word to be 26, which corresponds to $m = 13$. In this case, ROM1 has size $2^{13} \times 26$ bits, ROM2 has size $2^{13} \times 14$ bits since the first 12 bits of $\log(1 + c')$ are all 0, and ROM3 has size $2^{13} \times 5$ bits since by the previous analysis $\Delta(z)$ is less than 2^{-8} and hence only the last five bits need to be stored.

Simulation results show that the maximum conversion error is 2^{-24} after truncating the two guarding digits. Note that the look-up table size can be further reduced by using PLA (programmable logic array). The use of PLA reduces the size to about 16% of the original ROM size [1], similar effect can also be achieved by using PLA in this scenario.

4 Conclusions

The paper proposed a fast algorithm for evaluation of binary logarithms. Due to the discovery of a new nonlinear approximation function, not only is the conversion speed greatly improved, but also the hardware implementation is simple compared with existing approaches. Simulation results on IEEE single precision conversion have been presented, and the conversion requires only one ROM table with $2^{13} \times 26$ bits, one with $2^{13} \times 14$ bits, and one with $2^{13} \times 5$ bits.

Fig. 4 shows the hardware implementation of both algorithms 1 and 2. The circuits are currently being designed. In general, for a $2^n \times w$ ROM, its delay can be estimated as approximately $2nT_g$, where T_g is the delay of a gate [14, 15]. There exists a design trade-off between area and delay. For example, ROM1 table is used to compute A, B, P, and A', as shown in Fig. 4a. If multiple ROM1 tables are used, such values can be computed in parallel to reduce the delay, but at the cost of extra ROMs. In addition, the adders can be implemented using carry-lookahead adder to reduce the delay, but again at the cost of extra gates. Both architectures take three units of ROM delay, three units of adder delay, and one unit of MUX (multiplexer) delay. Careful implemention of the circuit is necessary. For example, an adder is used to calculate $(1 - B + A)$,



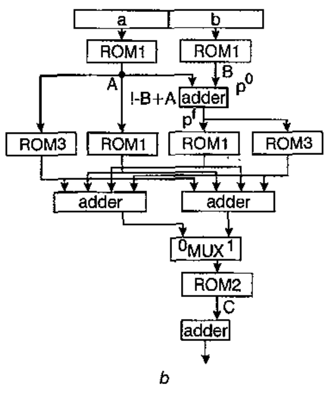


Fig. 4 *Hardware implementation*
a Algorithm 1
b Algorithm 2

where A and B are fractional numbers. Thus $(1 - B + A) = (1 + A) - B$ and $-B$ can be implemented using ones complement, where the least significant bit need for the twos complement can be ignored owing to the use of guided bits. The actual delay and area will be measured from the fabricated chips.

To extend this algorithm to even higher precision there appears to be a difficulty with approximation error with reasonable available ROM size. This algorithm uses one-step factorisation, namely, to factor $1 + x$ as $(1 + a)(1 + c')$. Multiple factorisations [16] have been explored and it appears that there is a large inherent approximation error involved. So to further increase conversion accuracy it seems that some kind of multiplication and/or division operations have to be used.

8 References

1 LO, H.Y., and AOKI, Y.: 'Generation of a precise binary logarithm with difference grouping programmable logic array,' *IEEE Trans.*, 1985, **C-34**, (8), pp. 681–691
2 KOREN, I.: 'Computer arithmetic algorithms' (Prentice Hall, 1993)
3 WONG, W.F., and GOTO, E.: 'Fast evaluation of the elementary functions in single precision,' *IEEE Trans.* 1995, **C-44**, (3), pp. 453–457
4 HUANG, S.-C., and CHEN, L.G.: 'The chip design of a 32-b logarithmic number system', Proceedings of IEEE international symposium on *Circuits and Systems*, 1994, pp. 167–170
5 ARNOLD, M.G., BAILEY, T.A., COWLES, J.R., and WINKEL, M.D.: 'Applying features of IEEE 754 to sign/logarithm arithmetic,' *IEEE Trans.*, 1992, **C-41**, (8), pp. 1040–1049
6 LAI, F.-S., and WU, C.-F.E.: 'A hybrid number system processor with geometric and complex arithmetic capabilities,' *IEEE Trans.*, 1991, **C-40**, (8), pp. 952–962
7 KOSTOPOULOS, D.K.: 'An algorithm for the computation of binary logarithms,' *IEEE Trans*, 1991, **C-40**, (11), pp. 1267–1270
8 KOREN, I., and ZINATY, O.: 'Evaluating elementary functions in a numerical coprocessor based on rational approximations,' *IEEE Trans.*, 1990, **C-39**, (8), pp. 1030–1037
9 LEWIS, D.M.: 'An architecture for addition and subtraction of long word length numbers in the logarithmic number system,' *IEEE Trans.* 1990, **C-39**, (11), pp. 1325–1336
10 STOURAITIS, T., and TAYLOR, F.J.: 'Floating-point to logarithmic encoder error analysis,' *IEEE Trans.*, 1989, **C-37**, (7), pp. 858–863
11 KUROKAWA, T., PAYNE, J., and LEE, S.: 'Error analysis of recursive digital filters implemented with logarithmic number systems,' *IEEE Trans.*, 1980, **ASSP-28**, pp. 706–715
12 TAYLOR, F.J.: 'A hybrid floating-point logarithmic number system processor', *IEEE Trans.*, 1985, **CAS-32**, pp. 92–95.
13 TAYLOR, F.J., GILL, R., JOSEPH, J., and RADKE, J.: 'A 20-bit logarithmic number system processor,' *IEEE Trans.*, 1988, **C-37**, pp. 190–199
14 DICLAUDIO, E.D., PIAZZA, F., and ORLANDI, G.: 'Fast combinational RNS processors for DSP applications,' *IEEE Trans.*, 1995, **C-44**, pp. 624–633
15 LEWIS, D.M.: 'Interleaved memory function interpolated with application to an accurate LNS arithmetic unit,' *IEEE Trans.*, 1994, **C-43**, pp. 974–982
16 WAN, Y.: 'Fast algorithms for generating high precision binary logarithm'. 1997. MS thesis, Department of Electrical Engineering, Michigan State University