システムＬＳＩ設計技術　104－13
(2002. 1. 24)

指数関数演算回路における性能／面積間のトレードオフに関する評価

波多江秀典† 橋本 浩二†† 村上 和彰†††

† 九州大学 大学院システム情報科学府 情報工学専攻
†† 九州大学 大学院システム情報科学府 情報理学専攻
††† 九州大学 大学院システム情報科学研究院 情報理学部門

E-mail: †arch@c.csce.kyushu-u.ac.jp

あらまし トランジスタ集積度の飛躍的な向上に伴い，LSIに搭載される算術演算回路の規模，複雑度が増加している．すなわち，今までソフトウェアで実現してきた指数関数，対数関数，三角関数，等の複雑な算術演算を算術演算回路，すなわち，ハードウェアとして実装できるようになってきた．本稿では，倍精度浮動小数点数に関する指数関数を対象に，新しいハードウェア・アルゴリズムを提案し，その性能／面積間のトレードオフを評価する．

キーワード 算術演算回路，ハードウェア・アルゴリズム，指数関数，LSI

Consideration on Tradeoff for Performance/Area of Exponential Function Circuits

Hidenori HATAE†, Koji HASHIMOTO††, and Kazuaki MURAKAMI†††

† Department of Computer Science and Communication Engineering, Kyushu University
†† Department of Informatics, Kyushu University
††† Department of Informatics, Kyushu University

E-mail: †arch@c.csce.kyushu-u.ac.jp

Abstract Ever increasing density of transistors on LSI's allows us to implement large and complex arithmetic circuits on board, and therefore some complex functions, such as exponential functions, which have been implemented by software are now potentially possible to be implemented by hardware. The paper surveys some existing hardware algorithm to implement exponential function, and then proposes a new hardware algorithm. The paper also tries to evaluate the tradeoff between performance and area of the algorithm.

Key words computer arithmetic, hardware algorithm, exponential function, LSI

1. はじめに

トランジスタ集積度の飛躍的な向上に伴い，LSIに搭載される算術演算回路の規模，複雑度が増加している．すなわち，データ形式に関しては固定小数点数から単精度浮動小数点数，倍精度浮動小数点数へと，また，算術演算内容に関しては加減算から乗算，除算，開平へと，算術演算回路で実現される機能の範囲が広がっている[2],[6]．今後は更に，指数関数，対数関数，三角関数，等のより複雑な算術演算を算術演算回路，すなわち，ハードウェアとして実装するものと予想される．これら算術演算を算術演算回路として実現するに当たっては，一般にその性能，面積，ならびに，消費電力／エネルギーの間にトレードオフ関係が成立する．たとえば，n ビットの2入力加算回路の場合，

- 順次桁上げ加算器（ripple carry adder: RCA）：$O(n)$ の遅延，$O(n)$ の面積．n が小さい範囲では消費電力，消費エネルギーともに下記の桁上げ先見加算器（CLA）よりも小さいが，n が大きくなると遅延時間の影響で消費エネルギーがCLAよりも大きくなる．
- 桁上げ先見加算器（carry lookahead adder: CLA）：$O(\log n)$ の遅延，$O(n \log n)$ の面積．

といったトレードオフ関係が存在する．

本稿では，倍精度浮動小数点数（IEEE754浮動小数点数形式）に関する指数関数を対象に，新しいハードウェア・アルゴリズムを提案し，既存の各種ハードウェア・アルゴリズムも含めて，それらの性能／面積間のトレードオフを評価する．次の2章で既存の指数関数アルゴリズムについて紹介し，3章で本稿で提案するハードウェア・アルゴリズムについて述べる．4章でこれらのアルゴリズムの性能および面積について議論し，5章でまとめを行う．

2. 既存の指数関数アルゴリズム

指数関数の計算アルゴリズムとして，STL (Sequential Table Lookup) 法[7],[3]，多項式近似を用いた解法[1]，Tang-Priest法[5],[8]，等が提案されている．

2.1 STL (Sequential Table Lookup) 法

STL法は，テーブルを逐次引きながら指数関数の計算を行う．任意の数 P，$Q(>0)$，$A(>0)$ に対して，

$$P + \ln Q = (P - \ln A) + \ln(Q \cdot A) \tag{1}$$

が成り立つ（ln は自然対数を示す）．STL法では，

表1　多項式近似を用いた解法の係数

係数	
A_0	1.4142137
A_1	9.8025821×10^{-1}
A_2	3.3972390×10^{-1}
A_3	7.8493057×10^{-2}
A_4	1.3683982×10^{-2}
A_5	1.8964611×10^{-3}

$$P_0 + \ln Q_0 = P_1 + \ln Q_1 = \cdots = P_n + \ln Q_n \tag{2}$$

という式の変換を用いる．この変換は，次の漸化式で表される．

$$\begin{aligned} P_j &= P_{j-1} - \ln A_j \\ Q_j &= Q_{j-1} \cdot A_j \end{aligned} \tag{3}$$

A_j $(j = 1, \ldots, 53)$ の取り得る値を1または $1+2^{-j}$ とし，$\ln(1+2^{-j})$ の値をテーブルに記憶しておくことで，上記の漸化式の計算がテーブル引き，シフト，および，加減算で行える．入力 $X(0 \leqq X \leqq \ln 2)$ に対して $\exp(x)$ を計算する場合，初期値を $P_0 = x$，$Q_0 = 1$ とし，各ステップでは，P_j が0に近づくように A_j を選ぶ．すなわち，

$$A_j = \begin{cases} 1 + 2^{-j} & (P_{j-1} - \ln(1+2^{-j}) \geqq 0) \\ 1 & (P_{j-1} - \ln(1+2^{-j}) < 0) \end{cases} \tag{4}$$

となる．ここで，

$$2^{-j-1} < \ln(1+2^{-j}) < 2 \cdot \ln(1+2^{-j-1}) < 2^{-j}$$

であるから，$0 \leqq P_j < 2^{-j}$ となり，一次の収束が保証される．小数点以下 n ビットの精度を得るには，このステップを n 回繰り返せばよい．この時，$P_n \approx 0$ となり，$x + \ln 1 \approx 0 + \ln Q_n$，すなわち，$x \approx \ln Q_n$ が成り立ち，$\exp(x) \approx Q_n$ となる．

2.2 多項式近似を用いた解法

本解法では，$\exp(x) = 2^y$ となる y を考える．この時，$y = \frac{x}{\ln 2}$ である．y を，その小数部 y_r が $-0.5 \leqq y_r \leqq 0.5$ を満たすように，整数部 y_i と小数部 y_r に分ける．すると，

$$\exp(x) = 2^y = 2^{y_i} \cdot 2^{y_r} \tag{5}$$

となる．ここで，2^{y_r} を多項式で近似すると，

$$\begin{aligned} &2^{y_r} \\ &\approx A_0 + y_r(A_1 + y_r(A_2 + y_r(A_3 + y_r(A_4 + y_r A_5)))) \end{aligned} \tag{6}$$

となる．係数 $A_0 \sim A_5$ は表1に示す値をとる[1]．2^{y_i} の部分は浮動小数点数の指数部にあたり，2^{y_r} の部分は仮数部にあたる．

表2　Priest 法の係数

係数	
A_0	$4.999999999999994 \times 10^{-1}$
A_1	$1.666666800631239 \times 10^{-1}$
A_2	$4.166666954215025 \times 10^{-2}$

2.3 Tang-Priest 法

Tang-Priest 法 [5] [8] では，$\exp(x)$ を以下のように変換する．

$$\begin{aligned} &\exp(x) = 2^k \cdot 2^{\frac{i}{256}} \cdot \exp\left(x - j \cdot \frac{\ln 2}{256}\right) \\ &j = \left[\frac{256x}{\ln 2}\right] = 256k + i \end{aligned} \tag{7}$$

ただし，i $(0 \leqq i < 256)$，kは整数．そして，2^k，$2^{\frac{i}{256}}$，$\exp(x - j\frac{\ln 2}{256})$ をそれぞれ求める．2^k は浮動小数点数における指数部となる．$2^{\frac{i}{256}}$ はテーブル参照で求める．また，$\exp\left(x - j \cdot \frac{\ln 2}{256}\right)$ は，以下の近似多項式により求める．

$$\begin{aligned} &\exp\left(x - j\frac{\ln 2}{256}\right) = \exp(r) \\ &= 1 + r(1 + r(A_0 + r(A_1 + rA_2))) \end{aligned} \tag{8}$$

係数 $A_0 \sim A_2$は表 2に示す値をとる [5]．

2.4 各アルゴリズムのコスト比較

2.1～2.3 節で紹介したすべてのアルゴリズムは，$\exp(x)$ の入力 x の定義域をある範囲内に限定する「レンジ・リダクション」という操作を必要とする意味では共通である．

次に，各アルゴリズムで必要とするテーブルサイズ（表3参照）ならびに演算回数（表4参照）を比較する．

- STL 法：$\ln(1 + 2^{-j})$ の値 $(j = 1, \ldots, 53)$ を保持するテーブルに 56 ビット× 53 語が必要．また，テーブル参照，シフト，および，加算がそれぞれ 53 回必要．
- 多項式近似を用いた解法：近似多項式近似の係数を保持するテーブルに 56 ビット× 6 語が必要．また，テーブル参照が 6 回，乗算および加算がそれぞれ 5 回必要．
- Tang-Priest 法：$2^{\frac{i}{256}}$の値 $(i = 0, \ldots, 255)$ を保持するテーブルに 56 ビット× 256 語，また，近似多項式近似の係数を保持するテーブルに 56 ビット× 3 語が必要．前者のテーブル参照が 1 回，後者のテーブル参照が 3 回必要．また，乗算が 5 回，加算が 4 回必要．

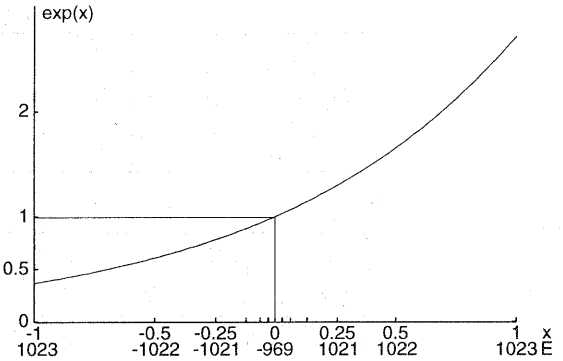

図1　x と Eと $\exp(x)$ の関係

3. 提案する指数関数アルゴリズム

3.1 アルゴリズム

3.1.1 準　　備

指数関数 $\exp(x)$ の入力 x が $x \geqq 710$ の場合，$\exp(x)$ の値は無限大（∞）となる．これは，倍精度浮動小数点数の表現可能最大値が 1.79×10^{308}であり，$\exp(x)$ の値がこの表現可能最大値よりも大きくなることによる．同様に，$x \leqq -746$ の場合，倍精度浮動小数点数の表現可能最小値が 4.94×10^{-324}であることから，$\exp(x)$ の値はこの表現可能最小値よりも小さくなり，結果的に $\exp(x) = 0$ となる．これから，$\exp(x)$ の値域が倍精度浮動小数点数で表現可能とするような入力 x の定義域は$-746 < x < 710$ となる．ここで，x を IEEE754 倍精度浮動小数点数形式で表現した場合の符号，指数部，仮数部をそれぞれ S，E，Fとすると，$x = (-1)^S \cdot 2^{E-1023} \cdot (1 + F)$ となる．上記の通り $|x| < 746$ であるから，$E \leqq 1032$ となる．

一方，$x \simeq 0$ の場合は $\exp(x) \to 1$ となる．$\exp(x) = 1$ を出力するような入力 x の値の範囲は $-\frac{1}{2^{54}} \leqq x \leqq \frac{1}{2^{53}}$ である．これより，Eの取り得る値の範囲は，$E \geqq 969$ となる．

以上から，Eの取り得る値の範囲は，$969 \leqq E \leqq 1032$ となる．図 1 に x と Eと $\exp(x)$ の関係を示す．

3.1.2 整数部と小数部への分離

$\exp(x)$ を以下のように変形する．

$$\begin{aligned} \exp(x) &= \exp((-1)^S \cdot 2^{E-1023} \cdot (1 + F)) \\ &= 2^{(-1)^S \cdot \frac{2^{E-1023}}{\ln 2} \cdot (1+F)} \end{aligned} \tag{9}$$

ここで，$D = (-1)^S \cdot \frac{2^{E-1023}}{\ln 2} \cdot (1 + F)$ を次の手順で計算して，整数部 D_iと小数部 D_rに分離する．なお，

表3　必要なテーブルサイズ

アルゴリズム	テーブルサイズ	利用目的
STL 法	56 ビット×53 語	$\ln(1+2^{-j})$ の値 $(j=1,\ldots,53)$
多項式近似を用いた解法	56 ビット×6 語	近似多項式の係数
Tang-Priest 法	56 ビット×256 語	$2^{\frac{i}{256}}$ の値 $(i=0,\ldots,255)$
	56 ビット×3 語	近似多項式の係数

表4　必要な演算回数

アルゴリズム	テーブル参照回数	乗算回数	シフト回数	加算回数
STL 法	53	0	53	53
多項式近似を用いた解法	6	5	0	5
Tang-Priest 法	1+3	5	0	4

計算はすべて固定小数点数表現で行う．

（1）　$(1+F)$ と $\frac{1}{\ln 2}$ との乗算．

（2）　上記乗算結果に 2^{E-1023} を乗ずる．これは，上記乗算結果を $(E-1023)$ ビットだけ（左方向へは最大 9 ビット，右方向には最大 54 ビット）シフトすることに相当する．

（3）　上記シフト結果に対して，小数点の左側を整数部 D_i，右側を小数部 D_r と分離する．なお，$S=1$ の場合は，上記シフト結果の 2 の補数表現を求めた後で分離する（付録参照）．

以上の分離の結果，

$$D=(-1)^S\cdot\frac{2^{E-1023}}{\ln 2}\cdot(1+F)=D_i+D_r$$
$$\exp(x)=2^D=2^{D_i+D_r}=2^{D_i}\cdot 2^{D_r}\quad(10)$$

となる．2^{D_i} は $\exp(x)$ の指数部に，また，2^{D_r} が仮数部にそれぞれ該当する．

3.1.3　2^{D_r} の計算

次に 2^{D_r} を計算する．これは前章で紹介した文献 [5] の方法を用いる．小数点以下のビット幅が 52 ビット，テーブルが 56 ビット× 2^8 の計算過程の例を示す．

D_r を x_i $(x_i\in\{0,1\}|\ 0\leqq i\leqq 51)$ の 2 進数で表現すると，

$$D_r=\sum_{i=0}^{51}2^{(i-52)}\cdot x_i\quad(11)$$

となり，

$$2^{D_r}=2^{\frac{i}{256}}\cdot 2^{D_r-\frac{i}{256}}\quad(12)$$
$$=2^{(\sum_{i=44}^{51}2^{i-52}\cdot x_i)}\cdot 2^{(\sum_{i=0}^{43}2^{i-52}\cdot x_i)}$$

となる．$x_{51}x_{50}\cdots x_{44}$ を入力とする $2^{(\sum_{i=44}^{51}2^{i-52}\cdot x_i)}$ の値を持つテーブルを引く．また，$2^{\sum_{i=0}^{43}2^{i-52}\cdot x_i}$ は，論文 [5] で紹介されている近似式で求める．

また，テーブルを用いて値を求める部分を二つに分割して表すと，

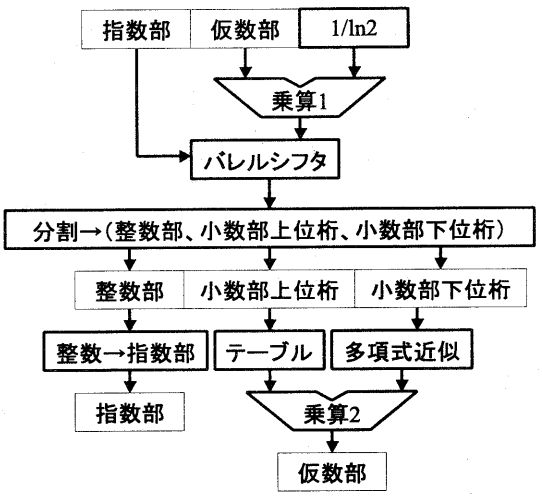

図2　指数関数演算回路の構成

$$2^{(\sum_{i=44}^{51}2^{i-52}\cdot x_i)}=2^{(\sum_{i=48}^{51}2^{i-52}\cdot x_i)}2^{(\sum_{i=44}^{47}2^{i-52}\cdot x_i)}\quad(13)$$

となる．$2^{(\sum_{i=48}^{51}2^{i-52}\cdot x_i)}$，$2^{(\sum_{i=44}^{47}2^{i-52}\cdot x_i)}$ の二つのテーブルそれぞれで求めた値を掛け合わせることにより，分割する前の $2^{(\sum_{i=44}^{51}2^{i-52}\cdot x_i)}$ のテーブルの値を求めることができる．この時，分割する前のテーブルサイズは 56×2^8 であるのに対して，分割したテーブルサイズは $56\times 2^4\times 2=56\times 2^5$ となる．このテーブルサイズ削減手法は乗算とテーブルサイズのトレードオフが存在する．つまり，n ビットを入力として持つテーブルのサイズは $O(2^n)$ で増加するが，この方法を用いれば，分割数 m とすると，乗算回数 $(m-1)$ 回必要となるが，テーブルサイズは $O(m\times 2^{\frac{n}{m}})$ となる．

3.2　ハードウェア構成

図 2 に，提案する指数関数アルゴリズムを実現する演算回路の構成を示す．これは下記の構成要素から成る．

- テーブル 1：入力の指数部を用いてテーブル参照．

- バレルシフタ:入力の指数部に基づいて仮数部をシフト.
- 乗算1：テーブル1から出力された値とバレルシフタから出力された値を乗ずる.
- 分割：「乗算1」の乗算結果を整数部，小数部上位桁，小数部下位桁に分割.
- テーブル2:小数部上位桁を用いてテーブル参照.
- 多項式近似：小数部下位桁を用いて多項式近似.
- 乗算2：テーブル2から出力された値と多項式近似結果を乗ずる.

本稿にて提案した方法をフローにすると以下のようになる.

（1） 入力変数に$\frac{1}{\ln 2}$を乗ずる
（2） 負ならば2の補数に変換
（3） 乗算結果を指数部に応じてシフト
（4） 整数部，小数部上位桁，小数部下位桁に分割
（5） 分割した値を並列に処理
（a） 整数部を出力の指数部へ変換
（b） 小数部上位桁のテーブル参照
（c） 小数部下位桁の多項式近似
（6） 上記小数部の結果を乗算

4. 現在の指数関数の実装状況と性能

現在の指数関数はソフトウェアで実装されているために，全ての行程を浮動小数点命令で実行しなければならい．これは大きなオーバーヘッドとなっている．表5はいくつかのCPUでの指数関数のサイクル数である．表5でのサイクル数は平均である．これは，入力された変数を有効なものか無効なものかを判断して有効なものしか計算しないためである．また，入力される変数の絶対値が小さければ小さいほど計算が早くなるためである.

LSIにおいて本稿で提案した例で実装をするならば,

- 乗算：2クロック
- バレルシフトと整数部，小数部分割：1クロック
- テーブル参照，多項式近似：(乗算4回パイプライン一段，加算4回パイプラインなし)，12クロック
- 最終乗算，2クロック

となり，指数関数の計算は18クロックで終了する．実際にはこのあとに丸め，正規化の処理が入る.

また，前章にて本稿で提案した指数関数のブロック図を示した．しかし，ブロック図2では，乗算のブロックが二つ，また，多項式近似のブロックの中に乗算が存在する．ここで，指数関数でのを見ると，データに依存関係を利用する．図2のブロックから，複数の乗算が同時に必要とされることはない．よって，LSIに実装する際には，一つの乗算器を共有することができる.

乗算器を共有して指数関数を実装すれば，面積的には浮動小数点演算器とさほど変わらない面積コストでLSI上に指数関数を実装できる．その例を図3で示す．図3では，Tang法による計算ブロックを展開している．テーブル3はTang法の多項式近似式における係数を係数を保持する．また，Tang法の多項式近似を実現するには定数加算器が必要になる.

4.1 テーブルと多項式近似のトレードオフ

小数部の計算をする方法として，多項式近似のテーラー展開を用いると，53ビットの精度を確保するには15次まで求める必要がある．しかし，テーブルを組み合わせることにより入力される範囲を狭めて多項式近似の次数を減らしていた．図3ののブロック図において，乗算がパイプライン化されているとすると，多項式近似の乗算をしている時間はデータに依存関係があるので他の演算が実行できないそこで，前章にて述べたテーブル分割を考える．テーブル分割により生じる乗算を多項式近似の乗算と並行して実行する．テーブル分割による演算回数とテーラー展開による多項式近似による演算回数のトレードオフを考える.

小数部のビット幅が56ビットだとする．テーブルで求めるビット幅をxビットとすると，多項式近似で求めるのは$56-x$ビットとなる．多項式近似の乗算回数は入力ビット幅により一意に決定する．そして，テーブル分割による乗算の回数と多項式近似の乗算回数が同じになるようにテーブルサイズを決定すればよい．表6に乗算回数とテーブルサイズの関係を示す.

また，計算が終了したあと，丸めの処理が必要になるが，それは全ての演算器に共通のことであるのでここでは触れていない.

5. おわりに

本稿では，RRをした後の処理の整数部と小数部の分割の際に演算結果が浮動小数点形式になるように分割する方法を示した．分割後の小数部の計算について，小数部上位桁の計算においてテーブルを分割するとテーブルサイズと乗算回数にトレードオフがあることを示た．また，小数部下位桁については多項式近似法，Tang法を用いることで計算する事を示した．本稿にて提案した方法で指数関数をLSI

表5　指数関数のクロックサイクル数

CPU	クロックサイクル	gcc version
UltraSPARC-II(450MHz)	85.09	2.95.2
UltraSPARC-II(296MHz)	89.94	2.7.2.3
PentiumIII(500MHz)	257.86	egcs-2.91.66

表6　乗算回数とテーブルサイズの関係

乗算回数	2	3	4	5
テーブルサイズ	56 ビット×192	56 ビット × 40	56 ビット×20	56 ビット×14

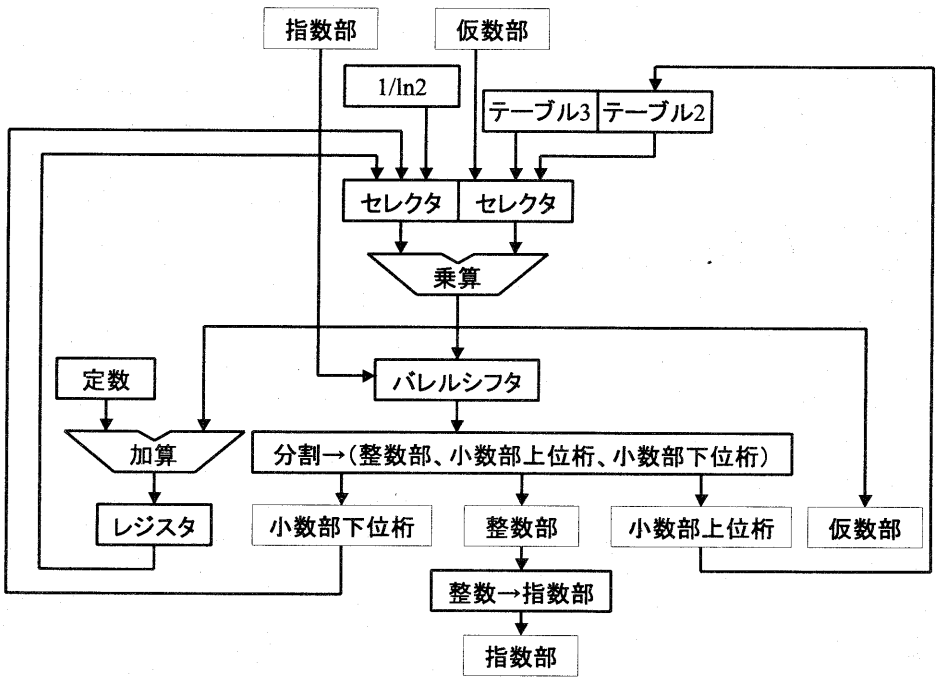

図3　指数関数ブロック図（乗算器共有型）

に回路を実装すれば指数関数を多用するアプリケーションで高速化が望める．

今後の課題としては，誤差解析，多項式近似の係数解析が残っている．

謝辞　本研究は一部，文部科学省 科学技術振興調整費 総合研究「科学技術計算専用ロジック組込み型プラットフォーム・アーキテクチャに関する研究」，日本学術振興会 科学研究費補助金 基盤研究 (A)(2) 展開研究「システム LSI 向けカスタム化可能 IP コアのアーキテクチャおよび設計支援技術の開発」（課題番号：12358002），日本学術振興会 科学研究費補助金 基盤研究 (A)(2) 一般研究「ハードウェア構成を動的に最適化する「ハードウェア・モーフィング」技術の開発」（課題番号：13308015）による．日頃からご討論頂く九州大学 安浦・村上・松永研究室の諸氏，ならびに，上記科学技術振興調整費プロジェクトの専用 LSI 開発チームの諸氏に感謝致します．本稿で紹介した「多項式近似を用いた解法」に関して有益なご助言を頂戴した東京大学 小柳義夫 教授に深く感謝致します．

文　献

[1] 小柳義夫, Private Communication, 2001.
[2] 高木直史, “算術演算回路のアルゴリズム,” 情報処理学会誌, vol.37, no.1–5, 1996 年 1 月–5 月.
[3] T. C. Chen, “Automatic Computation of Exponentials, Logarithms, Rations and Square Roots,” *IBM J. Research and Development*, vol. 16, no. 4, pp.380–388, July 1972.
[4] J. M. Muller, “Elementary Functions – Algorithms and Implementations –,” Birkhäuser, 1997.
[5] D. M. Priest, “Fast Table-Driven Algorithms for Interval Elementary Functions,” *Proc. 13th IEEE Symposium on Computer Arithmetic*, pp.168–174, 1997.
[6] P. Soderquist and M. Leeser, “Division and Square Root: Choosing the Right Implementation,” *IEEE Micro*, vol.17, no.4, pp.56–66, July/Aug. 1997.

[7] W. H. Specker, "A Class of Algorithms for $\ln x$, $\exp x$, $\sin x$, $\cos x$, $\tan^{-1} x$ and $\cot^{-1} x$," IEEE Trans. Electronic Computers, vol.EC-14, no.1, pp.85–97, Jan. 1965.
[8] P. T. P. Tang, "Table-Lookup Algorithmes for Elementary Functions and Their Error Analysis," *Proc. 10th IEEE Symposium on Computer Arithmetic*, pp.232–236, June 1991.

付　　録

一般的に n ビットの 2 の補数は二進数 i 桁目を b_i とすると，

$$-2^{n-1} \cdot b_{n-1} + \sum_{i=0}^{n-2} 2^i \cdot b_i \qquad \text{(A·1)}$$

と書ける．ある桁 j で上位桁と下位桁に分割すると，

$$-2^{n-1} \cdot b_{n-1} + \sum_{i=j}^{n-2} 2^i \cdot b_i + \sum_{i=0}^{j} 2^i \cdot b_i \qquad \text{(A·2)}$$
$$((n-1) > i > 0)$$

となる．この時

$$-2^{b_{n-1}} + \sum_{i=j}^{n-2} 2^i \cdot b_i < 0 \qquad \text{(A·3)}$$

$$\sum_{i=0}^{j} 2^i \cdot b_i \geqq 0 \qquad \text{(A·4)}$$

となり，任意の場所で分割しても，下位桁は正になる．

整数と小数に分割する前に 2 の補数にしておくと，分割した際に $0 < D_r < 1$ となるので，$1 \leqq 2^{D_r} < 2$ となる．