社団法人　電子情報通信学会
THE INSTITUTE OF ELECTRONICS,
INFORMATION AND COMMUNICATION ENGINEERS

信学技報
TECHNICAL REPORT OF IEICE.

二次近似法に基づくプログラマブル数値計算回路の構成とその合成法

永山　　忍† 　笹尾　　勤†† 　JonT. Butler†††

† 広島市立大学 情報工学科　〒 731–3194 広島市 安佐南区 大塚東 3–4–1
†† 九州工業大学 電子情報工学科　〒 820–8502 福岡県 飯塚市 川津 680–4
††† 海軍大学院大学　アメリカ カリフォルニア州 モントレー

あらまし　本稿は, 三角関数, 対数関数, 平方根演算, 逆数演算などの関数を計算する数値計算回路の構成とその自動合成法を提案する. 本数値計算回路は, LUT (Look-Up Table) カスケードを用いて, 与えられた定義域を不等区間に分割し, 数値関数を各区間毎に二次多項式で近似する. 不等区間分割, LUT カスケード, そして二次近似法を用いることで, 変化の激しい多様な関数に対しても, 従来法よりコンパクトに実現できる. 実験により以下を示す: 1) 本数値計算回路は, 線形近似法 (不等区間分割) に基づく数値計算回路の 4% のメモリ量で実現できる. 2) 本数値計算回路は, 5 次近似法 (等区間分割) に基づく数値計算回路の 22% のメモリ量で実現できる. 3) 本合成法では, 高精度 (24 ビット精度) 数値計算回路を従来法より小規模な FPGA で実現できる.

キーワード　不等区間分割, LUT カスケード, Chebyshev 二次近似多項式, 数値計算回路, 自動合成, FPGA.

Programmable Numerical Function Generators Based on Quadratic Approximation: Architecture and Synthesis Method

Shinobu NAGAYAMA†, Tsutomu SASAO††, and Jon T. BUTLER†††

† Department of Computer Engineering, Hiroshima City University
Ozuka-Higashi 3–4–1, Asa-Minami-Ku, Hiroshima, 731–3194 Japan
†† Department of Computer Science and Electronics, Kyushu Institute of Technology
Kawazu 680–4, Iizuka, Fukuoka, 820–8502 Japan
††† Department of Electrical and Computer Engineering, Naval Postgraduate School
Monterey, CA 93943-5121 USA

Abstract　This paper presents an architecture and a synthesis method for programmable numerical function generators (NFGs) for trigonometric, logarithmic, square root, and reciprocal functions. Our NFG partitions a given domain of the function into non-uniform segments using an LUT cascade, and approximates the given function by a quadratic polynomial for each segment. By using non-uniform segmentation, LUT cascade, and quadratic approximation, we can implement more compact NFGs than the existing methods for a wide range of functions. Implementation results on an FPGA show that: 1) our NFGs require only 4% of the memory needed by NFGs based on the linear approximation with non-uniform segmentation; 2) our NFGs require only 22% of the memory needed by NFGs based on the 5th-order approximation with uniform segmentation; and 3) our high-precision NFGs can be implemented with a compact and low-cost FPGA. Our automatic synthesis system generates such compact NFGs quickly.

Key words　Non-uniform segmentation, LUT cascades, 2nd-order Chebyshev approximation, numerical function generators (NFGs), automatic synthesis, FPGA.

1. はじめに

三角関数, 対数関数, 平方根演算, 逆数演算などの関数は, デジタル信号処理, 通信, ロボット工学, 天体物理学などの様々な分野で広く利用されている. 高性能 CPU は, これらの関数のための数値計算コプロセッサを備えているが, 安価な組込み CPU や FPGA 上の CPU は, 数値計算コプロセッサを備えていないため, 組込みシステムで高速な数値計算が要求される場合, ハードウェア実装による高速化が必要になる. 低精度 (例えば, 関数の入出力が 8 ビット) で数値関数を計算する場合, 関数表を単一メモリ

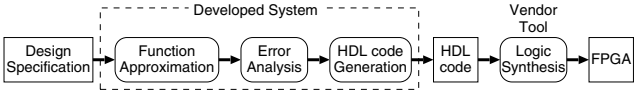


図 1　数値計算回路の合成フロー

でそのまま実装する方法が最も単純で高速であるが, 高精度で計算を行う場合には, 関数表が大きくなりすぎ, メモリでの実装が困難になる. そのため, 高精度の計算には, CORDIC (COordinate Rotation DIgital Computer) アルゴリズム [1], [22] に代表される繰り返しアルゴリズムがしばしば利用される. 代表的な FPGA ベンダーも, 数値計算 IP (Intellectual Property) として CORDIC を用意している. CORDIC アルゴリズムは, 加減算, シフト演算, およびテーブル参照の単純な演算の繰り返しにより, 関数を高精度で計算できるため, ハードウェア向きのアルゴリズムとして広く知られているが, 収束した解を得るには, 精度に比例した繰り返しを要する. そのため, CORDIC アルゴリズムは, 電卓などのように, 高速な計算が要求されないアプリケーションに適している [13] が, 高精度の計算を高速に行なうアプリケーションには, 不向きである.

高速な数値計算回路の実現法として, 関数を複数の多項式で近似し, その多項式を実現する手法が提案されている [9], [10], [20], [21]. 線形近似や二次近似を用いた数値計算回路は, 様々な関数を比較的高精度で高速に計算できるため, 有望な手法であるが, 統一的な実現法および合成法は, ほとんど提案されていない. 本稿では, LUT カスケード [8] を用いた不等区間生成, 二次近似に基づく数値計算回路の構成, およびその合成法を提案する. LUT カスケードの使用により, 定義域を任意の不等区間に分割でき, 様々な関数に対して, 高速でコンパクトな数値計算回路の自動合成が可能となる. 数値計算回路の自動合成フローを図 1 に示す. 本合成システムは, MATLAB 等の数値計算ソフト(注1) で記述された設計仕様から自動的に HDL コードを生成する. 設計仕様として, 数値関数 $f(x)$, x の定義域 $[a, b]$, および設計する数値計算回路の許容誤差を用いる. 本合成システムは, まず, 与えられた定義域を幾つかの区間へ分割し, 各区間を二次近似する. 次に, 数値計算回路の誤差を解析し, 計算に用いる演算器の精度 (ビット数) を算出する. 最後に, 誤差解析で算出された精度をもとに, HDL コードを生成する.

本稿は, 以下のように構成されている. 第 2. 節で用語の定義を行なう. 第 3. 節で, 不等区間への分割アルゴリズムと関数の二次近似法を示し, 第 4. 節で, 近似式を計算する数値計算回路の構成を示す. 第 5. 節で, 本数値計算回路の FPGA を用いた実装法について述べ, 第 6. 節では, いくつかの関数を本合成法を用いて合成し, 提案した合成法の効率を示す. 本数値計算回路の誤差解析は, ページ数削減のため, [15] に掲載する. 本稿は, [18] を二次近似へ拡張したものである.

2. 諸 定 義

［定義 2.1］　二進固定小数点で表現された数値 r を

$$r = d_{n\_int-1}\, d_{n\_int-2}\, \ldots\, d_1\, d_0 . d_{-1}\, d_{-2}\, \ldots\, d_{-n\_frac}$$

と表記する. ただし, $d_i \in \{0, 1\}$, n_int は整数部のビット数, n_frac は小数部のビット数である. このとき r は, 以下の式で計算できる.

$$r = -2^{n\_int-1} d_{n\_int-1} + \sum_{i=-n\_frac}^{n\_int-2} 2^i d_i$$

本稿では, 負数を 2 の補数で表現するため, 特に指定がない限り, n_int は符号ビットを含む.

［定義 2.2］　**誤差**とは, もとの値と近似値の差分絶対値を意味し, 特に, 本稿では, 関数近似で生じる誤差を**近似誤差**, 値を有限桁の二進固定小数点で表現した際に生じる誤差を**丸め誤差**という. **許容誤差**とは, 仕様で与えられる許容できる誤差の最大値である. 特に, **許容近似誤差**は, 許容できる近似誤差の最大値である.

［定義 2.3］　**精度 (precision)** とは, 実数計算における有効桁数のことである. 特に, n **ビット精度**とは, 実数計算において有効桁数が n ビット, すなわち, $n\_int + n\_frac = n$. 本稿で, n ビット精度の数値計算回路は, n ビット入力の回路を意味する.

［定義 2.4］　**確度 (accuracy)** とは, 実数計算における小数点以下の有効桁数のことである. 特に, m **ビット確度**とは, 実数計算において小数点以下の有効桁数が m ビット, すなわち, $n\_frac = m$. 本稿で, m ビット確度の数値計算回路は, 小数部 m ビット入力, 小数部 m ビット出力でなおかつ, 出力値の誤差が 2^{-m} の回路を意味する. m ビット確度の出力値を得るためには, 計算途中では, それ以上の確度で計算する必要がある.

3. 二次近似アルゴリズム

関数 $f(x)$ を効率良く二次近似するために, まず, x の定義域 $[a, b]$ をいくつかの区間に分割する. そして, その各区間において, $f(x)$ を二次関数 $g(x) = c_2x^2 + c_1x + c_0$ で近似する. このときの近似誤差は, 定義域の区間への分割法と係数 c_2, c_1, c_0 の値に依存する.

従来法の多くは, 定義域を等区間へ分割し関数近似を行う [2], [6], [20]. そのような等区間分割法は, 単純であり, 高速な回路を生成できるが, 関数によっては, 区間数が大きくなりすぎ実現できない場合がある. 一方, 定義域を不等区間に分割する不等区間分割法は, 同じ近似誤差の下で, 等区間分割法より少ない区間数で関数を近似できる [9], [18]. しかしながら不等区間分割法は, しばしば, 複雑な区間指定回路 (4. 節参照) が必要になり, 結果的に数値計算回路の面積や速度が劣化する. この問題点を解決するために, [9] は, 特殊な不等区間分割法を提案した. この手法は, 定義域を分割する点を制限することで, 簡単な区間指定回路を生成する. 結果的に, 関数を少ない区間数で近似でき, 高速でコンパクトな数値計算回路を生成できるが, アドホックな方法であるため, 自動合成に適していない. 本アーキテクチャでは, LUT カスケードの使用により, 任意の不等区間分割を高速でコンパクトに実現できる [18] ため, 本稿では, 自動合成に適した不等区間分割アルゴリズムを示す.

近似式の選定法は, 近似誤差だけでなく不等区間の個数にも影響する. 本稿では, 少ない不等区間で関数を近似するために, Chebyshev 二次近似式を用いて区間を生成し, 近似値の計算を行う. Chebyshev 近似式は, 近似誤差が小さく, 多項式の係数を容易に計算できるため, 区間分割アルゴリズムの計算時間を短くでき, 高速な自動合成に適している.

3.1 区間分割アルゴリズム

関数 f の区間 $[s, e]$ における Chebyshev 二次近似の最大近似誤差 $\epsilon_2(s, e)$ は, 以下の式で与えられる [11].

(注1)：現在の合成システムでは, フリーソフト Scilab [19] で仕様を記述できる.

入力　数値関数 $f(x)$, 定義域 $[a,b]$, 許容近似誤差 ϵ.
出力　不等区間 $[s_0,e_0],[s_1,e_1],\ldots,[s_{t-1},e_{t-1}]$.
処理
1. $s_0=a, i=0$ とする.
2. $\epsilon_2(s_i,p)=\epsilon$ を満たす $p(\geqq s_i)$ を探す.
3. $p>b$ ならば, $p=b$ とする.
4. $e_i=p, i=i+1$ とする.
5. $p=b$ ならば, $t=i$ として処理を終了する.
6. $p\neq b$ ならば, $s_i=p$ とし, 処理 2. へ戻る.

図 2　定義域の不等区間への分割アルゴリズム

$$\epsilon_2(s,e)=\frac{(e-s)^3}{192}\max_{s\leqq x\leqq e}|f^{(3)}(x)| \tag{1}$$

ここで $f^{(3)}(x)$ は, 元の数値関数の 3 次導関数を表す. (1) より, $\epsilon_2(s,e)$ は区間の幅 $e-s$ の単調増大関数である. 本分割アルゴリズムは, この性質を利用した貪欲解法により不等区間への分割を行う. 図 2 は, 本分割アルゴリズムを示す. このアルゴリズムは, 入力として, 関数 $f(x)$, 定義域 $[a,b]$ そして許容近似誤差 ϵ を与えると, 分割により生成された区間の数 t と各区間 $[s_0,e_0],[s_1,e_1],\ldots,[s_{t-1},e_{t-1}]$ を出力する. 生成された全ての区間において, 近似誤差は, ϵ 以下である. 図 2 の処理 2. において, $\epsilon_2(s_i,p)=\epsilon$ を満たす点 p を計算機で精確に求めることは難しい. そこで実際には, $\epsilon_2(s_i,p')$ を ϵ 以下で最大にする点 p' を計算機で求める. このような点 p' は, n ビット精度の入力 x の値を順に調べることで見つけることができる. しかしながら, それは $O(2^n)$ の探索を必要とする. 本アルゴリズムでは, $\epsilon_2(s_i,p')\leqq\epsilon$ を満たすように上位ビットから順に 0, 1 を決定していくことで, 点 p' の最大値を得る. これは, $O(n)$ の探索で見つけることができる. $\epsilon_2(s_i,p')$ の計算において, $\max_{s_i\leqq x\leqq p'}|f^{(3)}(x)|$ は非線形計画法 [7] により計算される.

3.2　近似値の計算

各区間 $[s_i,e_i]$ において, 関数 $f(x)$ をそれぞれ異なる二次関数 g_i で近似するため, ある値 x における関数 $f(x)$ の近似値 y は, x を含む区間 $[s_i,e_i]$ の二次関数

$$y=g_i(x)=c_{2i}x^2+c_{1i}x+c_{0i} \tag{2}$$

で計算する. ここで c_{2i}, c_{1i} および c_{0i} は, Chebyshev 二次近似多項式から導出される [11]. q_i を任意の値とし, (2) を以下のように変形する.

$$\begin{aligned} g_i(x) &= c_{2i}(x-q_i)^2+(c_{1i}+2c_{2i}q_i)(x-q_i) \\ &\quad +c_{0i}+c_{1i}q_i+c_{2i}q_i^2 \end{aligned} \tag{3}$$

(3) において, $c'_{1i}=c_{1i}+2c_{2i}q_i$ および $c'_{0i}=c_{0i}+c_{1i}q_i+c_{2i}q_i^2$ とすると,

$$g_i(x)=c_{2i}(x-q_i)^2+c'_{1i}(x-q_i)+c'_{0i} \tag{4}$$

が得られる. (4) は, (2) よりビット数の少ない乗算器で実現できるため, 本アーキテクチャは, (4) を採用する.

4.　数値計算回路のアーキテクチャ

(4) は, 図 3 に示すように, x を含む区間番号 i を計算する区間指定回路, $-q_i$, c_{2i}, c'_{1i} および c'_{0i} の係数表, 二乗回路, 二つの乗算器, そして二つの加算器の 7 つの構成要素で実現できる.

区間指定回路は, x を入力として, x を含む区間 $[s_i,e_i]$ の区間番号 i を出力する. x を n ビット精度とすると, 区間指定回路

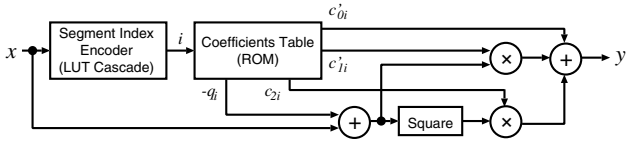
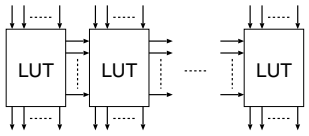


図 3　数値計算回路のアーキテクチャ

区間	区間番号
$s_0\leqq x\leqq e_0$	0
$s_1<x\leqq e_1$	1
$\vdots$	$\vdots$
$s_{t-1}<x\leqq e_{t-1}$	$t-1$

(a) 区間指定関数



(b) LUT カスケード

図 4　区間指定回路

は, 図 4(a) に示されている区間指定関数 $seg\_func(x):B^n\rightarrow\{0,1,\ldots,t-1\}$ を実現する回路である. ここで, $B=\{0,1\}$ であり, t は区間数を表す. 本稿では, 関数 $seg\_func(x)$ を図 4(b) に示されている LUT カスケード [8], [16], [17] で実現する. 関数 $seg\_func(x)$ を BDD (Binary Decision Diagram) で表現し, その BDD を用いて関数分解することにより, LUT カスケードが得られる. LUT カスケードの詳細な実現法については, [8], [16] を参照されたい. LUT カスケードでは, 隣接する LUT 間の信号線をレールといい, その信号線数をレール数と呼ぶ. コンパクトな LUT カスケードを実現するためには, レール数の削減が重要である. レール数が大きいと, LUT のサイズが大きくなり, 実現が困難になる. しかし, 区間指定関数 $seg\_func(x)$ は, [17] で示された定理により, レール数の小さいコンパクトな LUT カスケードで実現できることが, 理論的に保証されている.

［定理 4.1］　[17] 区間数を t としたとき, レール数が高々$\lceil\log_2 t\rceil$ の LUT カスケードで, 区間指定関数 $seg\_func(x)$ を実現できる.

本合成では, コンパクトな LUT カスケード実現のために, ヘテロジニアス MDD (Multi-valued Decision Diagram) [14] を用いる. また, LUT カスケードは, 各 LUT の並列動作が可能であるため, パイプライン処理による高速化が容易に実現できる. 従って, LUT カスケードの使用は, 高速でコンパクトな区間指定回路の実現を可能にする.

5.　FPGA での実装法

現在の主な FPGA は, 論理素子 (LE, CLB) の他に, メモリブロックや高速な乗算器 (DSP) を備えている. 本合成システムは, これらのハードウェア資源を有効利用し, 数値計算回路を生成する. 本アーキテクチャの区間指定回路 (LUT カスケード) と係数表はメモリブロック, 乗算器は DSP ユニット, 二乗回路および加算器は論理素子でそれぞれ実装する.

5.1　乗算器のサイズ削減

現在の FPGA は, 高速な乗算器を備えているが, 乗算器のサイズが大きくなるとそれに伴い遅延時間も大きくなるため, 高速な数値計算回路を得るためには, 乗算器のサイズ削減が重要である. 乗算器のサイズは, 係数 c_{2i}, c'_{1i} および $(x-q_i)$ のビット数に依存するため, 本節では, それらのビット数を削減する手法を述べる.

まず, 係数 c_{2i}, c'_{1i} の絶対値が大きい場合を考える. c_{2i}, c'_{1i} の絶対値が大きいとき, それらに必要なビット数も大きくな

表 1　数値計算回路のパイプライン段数

演算ユニット	パイプライン段数
1. 区間指定回路 (LUT カスケード)	n_cas
2. 係数表	1
3. 加算器: $x - q_i$	1
4. 二乗回路	1
5. 乗算器 (並列動作)	1
6. シフト演算器 (オプション)	0 or 1
7. 加算器	1
合計パイプライン段数	$n\_cas + 5 \sim n\_cas + 6$

n_cas: LUT カスケードの段数

る. このときのビット数を効率良く削減するために, 本合成システムは,[9] で示されたスケーリング手法を用いる. $|c_{2i}|$, $|c'_{1i}|$ の値が大きいとき, c_{2i} および c'_{1i} を $c_{2i} = c_{2i} \times 2^{-l_{2i}} \times 2^{l_{2i}}$, $c'_{1i} = c'_{1i} \times 2^{-l_{1i}} \times 2^{l_{1i}}$ で表現し, 係数表には, $c_{2i} \times 2^{-l_{2i}}$, $c'_{1i} \times 2^{-l_{1i}}$ の値と l_{2i}, l_{1i} の値をそれぞれ格納する. $c_{2i} \times 2^{-l_{2i}}$ および $c'_{1i} \times 2^{-l_{1i}}$ の値は, 元の c_{2i}, c'_{1i} より l_{2i}, l_{1i} ビットそれぞれ小さく表現できる. そして, 乗算器で, $c_{2i} \times 2^{-l_{2i}} \times (x - q_i)^2$, $c'_{1i} \times 2^{-l_{1i}} \times (x - q_i)$ を計算した後, $2^{l_{2i}}$, $2^{l_{1i}}$ をシフト演算することでもとの値を得る. シフト演算を用いる手法は, 乗算器のサイズ削減に有効であるが, 用いない手法に比べ, 誤差が大きくなる. 本合成システムでは, 誤差解析を行ない, 許容誤差以内で最適な l_{2i} および l_{1i} の値を算出する. 算出した結果, 全ての区間において, l_{2i}, l_{1i} の値が 0 ならば, シフト演算器を実装しない.

次に, $(x - q_i)$ について考える. $(x - q_i)$ のビット数は, 二乗回路と乗算器のサイズに影響するため, $(x - q_i)$ の値の範囲の削減は, 二乗回路と乗算器のサイズを削減する. また, $(x - q_i)$ の値の範囲の削減は, 二乗回路と乗算器のサイズだけでなく, 誤差も削減できる. (4) より, q_i は任意の値を設定できるため, 各区間 $[s_i, e_i]$ において, $(x - q_i)$ の値が小さくなるように q_i を区間の中央値, すなわち $q_i = (s_i + e_i)/2$ とする. 従って, $(x - q_i)$ の値の範囲は,$|x - q_i| \leqq (e_i - s_i)/2$ になる. そのため, 区間の幅 $(e_i - s_i)$ の削減は, $(x - q_i)$ の値の範囲を削減することがわかる. しかし, 区間の幅の削減は, 区間数の増加につながり, 結果的にメモリ量の増加につながる. そこで, 本節でメモリ量を増加させずに区間の幅を削減する方法を示す.

図 3 の係数表は, 2^k ワードの ROM で実現される. ただし t を区間数としたとき, $k = \lceil \log_2 t \rceil$ である. これは, 区間数を 2^k まで増加させても実装される ROM のサイズは増加しないことを示している. また, 区間指定回路の LUT カスケードにおいても, 定理 4.1 より, そのサイズは, k に依存するため, 区間数を 2^k に増加しても LUT カスケードのサイズに大きな変化はない. 本合成システムでは, 区間数が 2^k になるまで, 区間幅の大きな順に区間を二等分割し続けることで, 区間幅の最大値を削減する. この区間幅の削減は, メモリ量を増やすことなく, ビット数と誤差の両方を削減できる.

5.2　パイプライン処理による高速化

本数値計算回路は, スループットを高めるために, 回路中の各演算ユニット間にパイプライン・レジスタを挿入し, 全ての演算ユニットを並列動作させる. 各演算ユニットの遅延時間が短いため, 本数値計算回路は, 高いスループット (動作周波数) を達成できる. 図 3 に示した数値計算回路の各演算ユニットおよびパイプライン段数を表 1 に示す. 表より, 本数値計算回路のパイプライン段数は, 最小で LUT カスケードの段数 +5, 最大で LUT カスケードの段数 +6 になる.

表 3　線形近似法 (不等区間分割) とのメモリ量の比較

関数 $f(x)$	16 ビット精度			24 ビット精度		
	メモリ量 [bit]		比率	メモリ量 [bit]		比率
	線形	二次	[%]	線形	二次	[%]
2^x	20992	1112	5	696320	19072	3
$1/x$	21248	2432	11	700416	19136	3
$\sqrt{x}$	43776	5536	13	1425408	86784	6
$1/\sqrt{x}$	10176	1104	11	343040	19008	6
$\log_2(x)$	20864	2464	12	694272	19072	3
$\ln(x)$	20096	2448	12	700416	19136	3
$\sin(\pi x)$	19456	2336	12	661504	38656	6
$\cos(\pi x)$	19584	2336	12	663552	38784	6
$\tan(\pi x)$	19712	2304	12	667648	38272	6
$\sqrt{-\ln(x)}$	74240	11264	15	2662400	173056	7
$\tan^2(\pi x) + 1$	37632	4960	13	1290240	39040	3
Entropy	106496	10688	10	3768320	83968	2
Sigmoid	21120	2432	12	702464	40320	6
Gaussian	4416	444	10	156672	8384	5
平均	31415	3704	11	1080905	45906	4

線形: 線形近似法 [18].　　二次: Chebyshev 二次近似法.

6. 実 験 結 果

6.1　区間数と分割アルゴリズムの計算時間

本節では, 本分割アルゴリズムの効率を示すために, 計算時間と区間数を比較する. 表 2 は,[17] の実験で使用した関数に対して, 様々な区間分割法を適用したときの区間数を比較している. 表中の *Entropy*, *Sigmoid*, *Gaussian* は, 以下のように定義される関数である.

$$\textit{Entropy} = -x\log_2 x - (1-x)\log_2(1-x),$$
$$\textit{Sigmoid} = \frac{1}{1+e^{-4x}}, \qquad \textit{Gaussian} = \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}}$$

表 2 で, “線形近似 不等区間” は, [18] で提案された線形近似に基づく不等区間分割法, “二次近似 等区間” と “二次近似 不等区間” は, Chebyshev 二次近似に基づく等区間分割法と不等区間分割法をそれぞれ表す. “計算時間 [msec]” は, 本手法を関数に適用したときの CPU 時間をミリ秒単位で表している.

表 2 から, 多くの関数において, Chebyshev 二次近似に必要な区間数は線形近似より遙かに少ないことがわかる. しかしながら, $\sqrt{-\ln(x)}$ などの関数において, 等区間分割は, 二次近似を用いても, 線形近似の不等区間分割法より多くの区間数が必要になる. 同様に, 多くの従来法は, 等区間分割法であるため, 三角関数や指数関数は, 比較的少ない区間数で近似できるが, $\sqrt{-\ln(x)}$ などの関数では, 過度に多くの区間数が必要になる. 一方, 本分割アルゴリズムは, 二次近似に基づく不等区間分割なので, 様々な関数を少ない区間数で近似できる. また, 表 2 から, 本分割アルゴリズムの計算時間は, 区間数に依存することがわかる. 許容近似誤差を小さくすると, 区間数が増えるため, それに伴い計算時間も長くなる. しかしながら, 表 2 より, 許容近似誤差を 2^{-25} にしても, 本分割アルゴリズムの計算時間は, 実験に用いた全ての関数に対して, 1 秒以下だった. 以上の結果より, 本分割アルゴリズムは, 定義域を少数の不等区間へ高速に分割でき, 実用的であると言える.

6.2　数値計算回路のメモリ量の比較

本節では, 本数値計算回路のコンパクトさを示すために, 既存手法により生成された数値計算回路との比較を行う. 表 3 は, [18] の線形近似法に基づく数値計算回路と比較している. こ

表 2　様々な区間分割法における区間数の比較

関数 $f(x)$	定義域	許容近似誤差: 2^{-17}				許容近似誤差: 2^{-25}			
		線形近似	二次近似		計算時間	線形近似	二次近似		計算時間
		不等区間	等区間	不等区間	[msec]	不等区間	等区間	不等区間	[msec]
2^x	[0, 1]	128	9	7	0.1	2048	65	44	70
$1/x$	[1, 2)	124	16	11	0.1	1982	128	64	60
$\sqrt{x}$	[1/32, 2)	193	252	24	10	3082	2016	138	150
$1/\sqrt{x}$	[1, 2)	46	16	8	0.1	1024	128	46	50
$\log_2(x)$	[1, 2)	128	16	10	10	2048	128	56	70
$\ln(x)$	[1, 2)	89	16	9	10	1437	128	50	50
$\sin(\pi x)$	[0, 1/2)	127	17	12	10	2027	129	74	90
$\cos(\pi x)$	[0, 1/2)	127	17	12	10	2027	129	74	90
$\tan(\pi x)$	[0, 1/4)	112	33	12	10	1787	129	73	110
$\sqrt{-\ln(x)}$	[1/32, 1)	354	31744	52	70	5933	8126464	331	720
$\tan^2(\pi x)+1$	[0, 1/4)	256	33	17	20	4096	257	101	170
Entropy	[1/256, 255/256]	520	509	40	30	8320	4065	234	300
Sigmoid	[0, 1]	127	33	13	20	2020	129	76	160
Gaussian	[0, 1/2]	32	5	4	0.1	512	33	18	30
平均		170	2337	17	20	2739	580995	99	100

実験環境

CPU: Pentium4 Xeon 2.8GHz　　メモリ: 4GB　　OS: Redhat (Linux 7.3)　　C コンパイラ: gcc -O2

表 4　等区間 5 次近似法とのメモリ量の比較

関数 $f(x)$	定義域	確度	メモリ量 [bit]		比率 [%]
			5 次近似 等区間	二次近似 不等区間	
$\sin(\pi x)$	[0, 1/4]	2^{-23}	70528	18048	26
$\exp(x)$	[0, 1]	2^{-24}	82432	43136	52
2^x-1	[0, 1]	2^{-24}	89600	19968	22

5 次近似: Taylor 5 次近似法 [3].

二次近似: Chebyshev 二次近似法.

表 5　等区間二次近似法とのメモリ量の比較

関数 $f(x)$	定義域	確度	メモリ量 [bit]		比率 [%]
			Minimax 等区間	Chebyshev 不等区間	
$\sin(\pi x/4)$	[0, 1)	2^{-24}	16288	19200	118
2^x-1	[0, 1)	2^{-16}	2208	2512	114

Minimax: Minimax 二次近似法 [4].

Chebyshev: Chebyshev 二次近似法.

の手法は, 本手法同様, 定義域の不等区間分割を用いている. 表が示しているように, 本数値計算回路は, 線形近似法の回路に比べメモリ量を大幅に削減できる. 特に, 24 ビット精度回路においては, 線形近似法の 4% 程度のメモリ量で回路を実現できる. 表が示している精度とメモリ量の関係から, 精度を上げるほどメモリ量の差が大きくなることがわかる.

表 4 は, [3] で示された 5 次の Taylor 展開に基づく数値計算回路と, 表 5 は, [4] で示された二次の Minimax 近似 (Remez アルゴリズム) に基づく数値計算回路とそれぞれ比較している. [3], [4] ともに, 定義域の等区間分割を用いている. [18] や表 2 で示されているように, 三角関数や指数関数などのように与えられた定義域で穏やかに変化する関数は, 同じ近似式の下で, 等区間分割と不等区間分割で区間数に大きな差が生じない. そのため, そのような関数では, 区間指定回路を必要としない等区間分割法が, 不等区間分割法に比べコンパクトな数値計算回路を生成できる. 本数値計算回路は, 区間指定回路をもち, なおかつ両手法より近似誤差の大きい近似式を用いているにも関わらず, [3] の 22% から 52% のメモリ量で, そして [4] に匹敵するメモリ量で回路を実現できる. [3] および [4] では, 関数 $\sqrt{x}$ や $\sqrt{-\ln(x)}$ に対しての実験結果が示されていないが, 表 2 より, メモリ量が過度に大きくなることが, 容易に推測できる. 一方, 本数値計算回路は, 様々な関数を少ないメモリ量で実現できる.

6.3　FPGA 実装結果

表 6 は, 線形近似と二次近似に基づく数値計算回路を FPGA 実装した結果を比較している. 線形数値計算回路のアーキテクチャは, 二次数値計算回路より単純なので, より高速で, より少ない論理素子および DSP を使って FPGA 実装できる. しかし, 線形近似は, 表 2 と表 3 で示したように, 二次近似より多くの区間数とメモリ量を必要とする. 表 6 で, 24 ビット精度の線形数値計算回路は, 論理素子と DSP が十分余っているにも関わらず, *Gaussian* を除いたすべての関数をメモリ不足のため, FPGA (Stratix ファミリで最小のデバイス) に実装できなかった. FPGA 実装では, これらのハードウェア資源を効率よく利用することが重要である. 24 ビット精度の線形数値計算回路は, メモリ不足のため, より大きな (高価な)FPGA を必要とするが, その FPGA では, さらに多くの論理素子と DSP が余ってしまい実装効率が悪くなる. 一方, 二次数値計算回路は, メモリ量が線形数値計算回路より大幅に少なく, 使用する論理素子と DSP も比較的少ないため, より小さな (安価な)FPGA で実装できる. 実際に, 24 ビット精度の二次数値計算回路は, より小さく安価な FPGA (Cyclone II) で実装できる.

7.　結論とコメント

本稿は, 三角関数, 対数関数, 平方根演算, 逆数演算などの関数を計算する数値計算回路の構成とその自動合成法を提案した. LUT (Look-Up Table) カスケードは, 定義域の任意の不等区間分割を高速でコンパクトに実現できるため, 多様な関数を効率良く二次近似でき, 高速でコンパクトな数値計算回路の自動合成を可能にする. 実験により, 本合成法は, 多様な関数を少ない区間数で近似でき, メモリ量の少ない数値計算回路を生成することを示した. 24 ビット精度において, 本数値計算回路は, 線形近似に基づく数値計算回路の 4% のメモリ量で, 5 次近似に基づく数値計算回路の 22% のメモリ量で実装できる. 線形近似に基づ

表 6　線形近似と二次近似に基づく数値計算回路の FPGA 実装

FPGA デバイス:	Altera Stratix (EP1S10F484C5: 論理素子数 10570 個, DSP ユニット数 48 個)											
論理合成ツール:	Altera QuartusII 5.0											
合成オプション:	速度最適化, タイミング制約: 200MHz											
関数	16 ビット精度						24 ビット精度					
$f(x)$	論理素子数		DSP 数		動作周波数 [MHz]		論理素子数		DSP 数		動作周波数 [MHz]	
	線形	二次	線形	二次	線形	二次	線形	二次	線形	二次	線形	二次
2^x	167	482	2	4	195	185	604	758	2	10	–	131
$1/x$	204	376	2	4	234	186	636	859	2	10	–	134
$\sqrt{x}$	270	496	2	4	237	179	1211	822	2	16	–	124
$1/\sqrt{x}$	186	475	2	4	237	186	402	753	2	10	–	131
$\log_2(x)$	163	381	2	4	194	186	597	757	2	10	–	131
$\ln(x)$	170	379	2	4	197	185	416	863	2	10	–	131
$\sin(\pi x)$	154	424	2	4	197	192	480	646	8	10	–	134
$\cos(\pi x)$	172	354	2	4	237	179	412	647	8	10	–	131
$\tan(\pi x)$	234	382	2	4	237	178	655	604	2	10	–	131
$\sqrt{-\ln(x)}$	304	623	2	10	215	135	854	942	8	16	–	130
$\tan^2(\pi x)+1$	132	282	2	4	194	215	991	720	2	10	–	135
Entropy	141	403	2	4	235	206	1370	914	2	16	–	128
Sigmoid	167	430	2	4	194	191	627	706	2	10	–	131
Gaussian	181	419	2	4	237	186	303	747	2	10	216	129
平均	189	422	2	4	217	185	683	767	3	11	–	131

線形: 線形近似法 [18].　　二次: Chebyshev 二次近似法.
表中の "–" は FPGA 上の組込み RAM が不足し実装できなかったことを示す.

く数値計算回路は, 二次近似に基づく数値計算回路より高速であるが, 高精度数値計算回路を FPGA 実装するためには, 多くのメモリを備えた大規模 FPGA が必要になる. 一方, 二次近似に基づく数値計算回路は, 小規模で安価な FPGA で高精度数値計算回路を実装できる.

謝　　辞

本研究は一部, 文部科学省, 知的クラスタ創成事業, 日本学術振興会, 科学研究費による.

文　　献

[1] R. Andraka, "A survey of CORDIC algorithms for FPGA based computers," *Proc. of the 1998 ACM/SIGDA Sixth Inter. Symp. on Field Programmable Gate Array*, pp. 191–200, Monterey, CA, Feb. 1998.

[2] J. Cao, B. W. Y. Wei, and J. Cheng, "High-performance architectures for elementary function generation," *Proc. of the 15th IEEE Symp. on Computer Arithmetic (ARITH'01)*, Vail, Co, pp. 136–144, June 2001.

[3] D. Defour, F. de Dinechin, and J.-M. Muller, "A new scheme for table-based evaluation of functions," *36th Asilomar Conference on Signals, Systems, and Computers,*, Pacific Grove, California, pp. 1608–1613, Nov. 2002.

[4] J. Detrey and F. de Dinechin, "Second order function approximation using a single multiplication on FPGAs," *Proc. Inter. Conf. on Field Programmable Logic and Applications (FPL'04)*, pp. 221–230, 2004.

[5] N. Doi, T. Horiyama, M. Nakanishi, and S. Kimura, "Minimization of fractional wordlength on fixed-point conversion for high-level synthesis," *Proc. of Asia and South Pacific Design Automation Conference (ASPDAC'04)*, pp. 80–85, 2004.

[6] H. Hassler and N. Takagi, "Function evaluation by table look-up and addition," *Proc. of the 12th IEEE Symp. on Computer Arithmetic (ARITH'95)*, Bath, England, pp. 10–16, July 1995.

[7] T. Ibaraki and M. Fukushima, *FORTRAN 77 Optimization Programming*, Iwanami, 1991 (in Japanese).

[8] Y. Iguchi, T. Sasao, and M. Matsuura, "Realization of multiple-output functions by reconfigurable cascades," *International Conference on Computer Design: VLSI in Computers and Processors (ICCD'01)*, Austin, TX, pp. 388–393, Sept. 23–26, 2001.

[9] D.-U. Lee, W. Luk, J. Villasenor, and P. Y.K. Cheung, "Non-uniform segmentation for hardware function evaluation," *Proc. Inter. Conf. on Field Programmable Logic and Applications*, pp. 796–807, Lisbon, Portugal, Sept. 2003.

[10] D.-U. Lee, W. Luk, J. Villasenor, and P. Y.K. Cheung, "A hardware Gaussian noise generator for channel code evaluation," *Proc. of the 11th Annual IEEE Symp. on Field-Programmable Custom Computing Machines (FCCM'03)*, Napa, CA, pp. 69–78, April 2003.

[11] J. H. Mathews, *Numerical Methods for Computer Science, Engineering and Methematics*, Prentice-Hall, Inc., Englewood Cliffs, NJ, 1987.

[12] J.-M. Muller, *Elementary Function: Algorithms and Implementation*, Birkhauser Boston, Inc., Secaucus, NJ, 1997.

[13] S. Muroga, *VLSI system design: when and how to use very-large-scale integrated circuits*, John Wiley & Sons, New York, 1982.

[14] S. Nagayama and T. Sasao, "Compact representations of logic functions using heterogeneous MDDs," *IEICE Trans. on fundamentals*, Vol. E86-A, No. 12, pp. 3168–3175, Dec. 2003.

[15] S. Nagayama, T. Sasao, and J. T. Butler, "Error analysis for programmable numerical function generators based on quadratic approximation," http://www.lsi-cad.com/Error-QNFG/.

[16] T. Sasao, M. Matsuura, and Y. Iguchi, "A cascade realization of multiple-output function for reconfigurable hardware," *Inter. Workshop on Logic Synthesis (IWLS'01)*, Lake Tahoe, CA, pp. 225–230, June 12–15, 2001.

[17] T. Sasao, J. T. Butler, and M. D. Riedel, "Application of LUT cascades to numerical function generators," *Proc. the 12th workshop on Synthesis And System Integration of Mixed Information technologies (SASIMI'04)*, Kanazawa, Japan, pp. 422–429, Oct. 2004.

[18] T. Sasao, S. Nagayama, and J. T. Butler, "Programmable numerical function generators: architectures and synthesis method," *Proc. Inter. Conf. on Field Programmable Logic and Applications (FPL'05)*, Tampare, Finland, pp. 118–123, Aug. 2005.

[19] Scilab 3.0, INRIA-ENPC, France, http://scilabsoft.inria.fr/

[20] M. J. Schulte and J. E. Stine, "Approximating elementary functions with symmetric bipartite tables," *IEEE Trans. on Comp.*, Vol. 48, No. 8, pp. 842–847, Aug. 1999.

[21] J. E. Stine and M. J. Schulte, "The symmetric table addition method for accurate function approximation," *Jour. of VLSI Signal Processing*, Vol. 21, No. 2, pp. 167–177, June 1999.

[22] J. E. Volder, "The CORDIC trigonometric computing technique," *IRE Trans. Electronic Comput.*, Vol. EC-820, No. 3, pp. 330–334, Sept. 1959.