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.

プログラマブル数値計算回路のアーキテクチャとその合成法

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アーキテクチャ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, 数値関数を各区間ごとに線形多項式で近似する. このため, 従来のアーキテクチャでは実現が困難な, 変化の激しい多様な関数に対しても, 本アーキテクチャは, 高速でコンパクトな数値計算回路を合成できる. 本数値計算回路は, MATLAB 等で記述された仕様から自動合成でき, 本稿では, 自動合成された回路を FPGA (Field Programmable Gate Array) で実装し, 他の数値計算回路との比較を行なう. 種々の関数を用いた実験により, 本アーキテクチャおよびその合成法の有用性を示す.

キーワード　LUT カスケード, 数値計算回路, パイプライン処理, 自動合成, FPGA.

Programmable Numerical Function Generators: Architectures 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) of trigonometric functions, logarithm functions, square root, reciprocal, etc. Our architecture partitions a given domain of function into non-uniform segments using an LUT (Look-Up Table) cascade, and approximates the given function by a linear polynomail for each segment. Thus, our architecture can implement fast and compact NFGs for a wide range of functions. We have developed a synthesis system for NFGs that converts MATLAB-like specification into HDL code. We show and compare three architectures implemented as a FPGA (Field-Programmable Gate Array). Experimental results show the efficiency of our architecture and synthesis system.

Key words　LUT cascades, numerical function generators, pipeline processing, 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. これらの関数は, ソフトウェアでの実装が, 手軽で一般的であるが, 高速な計算が要求される分野では, ハードウェア実装による高速化が求められる. 低精度で関数を計算する場合 (入力のビット数が小さい場合) には, 関数表を単一メモリ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], [20] 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

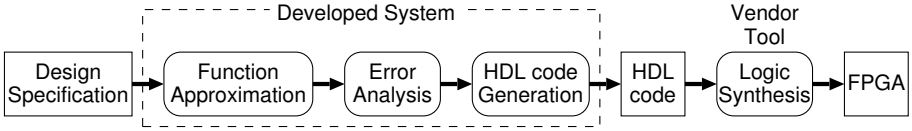


図 1　数値計算回路の合成フロー

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 [10] 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 [8], [17], [19]. 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 [7], [12] を用いた不等区間生成, 線形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. 本アーキテクチャは, 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, 様々な関数を効率的に線形近似する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, 各区間を線形近似する. 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, 関数の線形近似アルゴリズムを示し, 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 を用いた実現法について述べ, 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, [14] を和訳したものである.

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.

(注1)：現在の合成システムでは, フリーソフト Scilab [16] で仕様を記述できる.

［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, $n\_int + n\_frac = n$ を意味する. 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, $n\_frac = m$ を意味する. 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. 線形近似アルゴリズム

関数 $f(x)$ を効率良く線形近似するために, 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, $f(x)$ を線形関数 $g(x) = c_1 x + c_0$ 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_1, c_0 の値の二つに依存する.

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], [5], [17]. 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. 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, [8] 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.

3.1 区間分割アルゴリズム

本稿で提案する区間分割アルゴリズムを図 2 に示す. このアルゴリズムは, 入力として, 関数 $f(x)$, 定義域 $[a, b]$ および許容近似誤差 c を与えると, 分割により生成された t 個の区間 $[s_0, e_0], [s_1, e_1], \ldots, [s_{t-1}, e_{t-1}]$ と線形関数の補正値 $v_0, v_1, \ldots, v_{t-1}$ を出力する. 生成された全ての区間において, 近似誤差は, c 以下である. 本分割アルゴリズムは, 画像の曲線描

入力　数値関数 $f(x)$, 定義域 $[a,b]$, 許容近似誤差 c.
出力　不等区間 $[s_0,e_0],[s_1,e_1],\ldots,[s_{t-1},e_{t-1}]$, 線形関数の補正値 $v_0,v_1,\ldots,v_{t-1}$.

処理　初期区間を $[a,b]$ とした, 再帰処理により定義域を分割する.
1. 与えられた区間 $[s,e]$ に対して, 二点 $(s,f(s)),(e,f(e))$ を通る線形関数 $g(x)$ を求める.
2. 区間 $[s,e]$ において $f(x)-g(x)$ を最大にする点 $x=p_{max}$ を求め, $\max_{fg}=f(p_{max})-g(p_{max})\geqq 0$ とする.
3. 同様に, $f(x)-g(x)$ を最小にする点 $x=p_{min}$ を求め, $\min_{fg}=f(p_{min})-g(p_{min})\leqq 0$ とする.
4. 区間 $[s,e]$ における近似誤差を $error=|\max_{fg}-\min_{fg}|/2$, 線形関数 $g(x)$ の補正値を $v=(\max_{fg}+\min_{fg})/2$ とする.
5. $error\leqq c$ ならば, この区間 $[s,e]$ における再帰処理を終了する.
6. $|\max_{fg}|>|\min_{fg}|$ ならば $p=p_{max}$, そうでなければ $p=p_{min}$ とする.
7. 区間 $[s,e]$ を二つの区間 $[s,p]$ と $[p,e]$ に分割し, 各区間に対し再帰処理を繰り返す.

図 2　定義域の不等区間への分割アルゴリズム

画に用いる Douglas-Peucker アルゴリズム [4] の応用であり, 区間 $[s,e]$ の両端点 $(s,f(s)),(e,f(e))$ を通る近似線形関数 $g(x)$ と $f(x)$ の誤差が最大になる点を分割点 p とし, 区間を分割してゆく組織的な手法である. 分割点 p は, $[s,e]$ における x の値を全て調べることで見つけられるが, それは多くの計算時間を要する. そこで本稿では, 非線形計画法 [6] を用いて, 効率良く分割点 p を見つける (図 2 の 2. および 3. の処理). 本アルゴリズムは, 補正値 v を用いて線形関数 $g(x)$ を垂直方向へ平行移動することで, $f(x)$ との最大近似誤差を削減する.

3.2　近似値の計算

区間 $[s_i,e_i]$ を seg_i と表記すると, 分割アルゴリズムで得られた各区間は, $seg_0,seg_1,\ldots,seg_{t-1}$ と表記できる. 各 seg_i は, それぞれ異なる線形関数 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 seg_i の線形近似関数

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

で計算する. c_{1i} および c_{0i} は, それぞれ次式で計算できる.

$$c_{1i}=\frac{f(e_i)-f(s_i)}{e_i-s_i} \text{ および } c_{0i}=f(s_i)-c_{1i}s_i+v_i$$

この c_{0i} をもとの線形関数 $g_i(x)$ に代入して整理すると,

$$g_i(x)=c_{1i}(x-s_i)+f(s_i)+v_i \tag{2}$$

が得られる. $h=e_i-s_i$ とし, h を 0 に近づけると, $c_{1i}=f'(s_i)$ となり, これを $g_i(x)$ に代入すると

$$g_i(x)=f'(s_i)(x-s_i)+f(s_i)+v_i$$

が得られる. この式は, s_i における補正項付きの一次のテイラー展開であり, 本近似法が, 区間の数を増やすことで, 理論的には任意の誤差で関数を近似できることを示している.

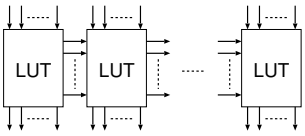
4.　数値計算回路のアーキテクチャ

4.1　全体の回路構成

3.2 節の (1) および (2) は, 同じ値を表現するが, 実現する数値計算回路の構成は異なる. (1) は, 図 3(a) 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, c_{1i} および c_{0i} の係数表, 乗算器, そして加算器の 4 つの構成要素で実現できる. 一方, (2) は, $-s_i$ の係数表および $x+(-s_i)$ を計算する回路が加わり図 3(b) に示す 5 つの構成要素で実現できる. 特に, (2) において, $s_i=(x$ の上位 $n-k$ ビット$)\times 2^k$ としたとき (等区間分割), 区間番号 i は, x の上位 $n-k$ ビット, $x-s_i$ は, x の下位 k ビットにそれぞれ等しくなるため, 図 3(c) に示すように, c_{1i} および c_{0i} の係数表, 乗算器, そして加算器の 3 つの構成要素で実現できる.

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

本合成システムでは, 高速でコンパクトな数値計算回路のために, 図 3(b) のアーキテクチャを採用する. 6. 節で, 三つのアーキテクチャの比較実験を行ない, 本アーキテクチャの性能および効率を示す.

4.2　区間指定回路 (Segment Index Encoder)

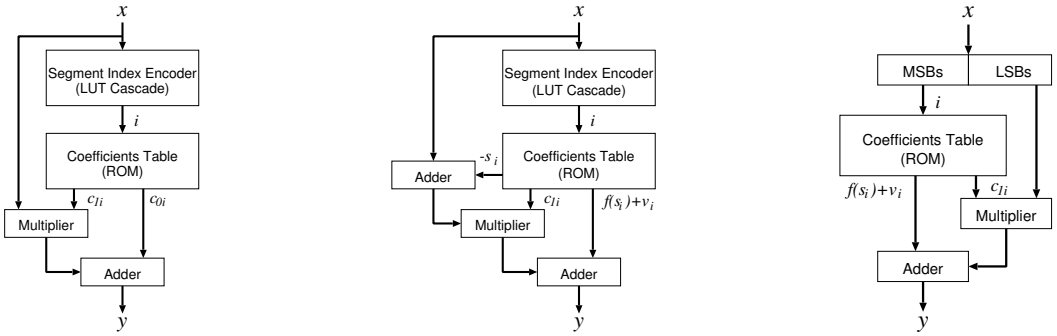
区間指定回路は, x を入力として, x を含む区間 seg_i の区間番号 i を出力する. x を n ビット精度とすると, 区間指定回路は, 関数 $seg\_func(x):B^n\rightarrow\{0,1,\ldots,t-1\}$ を表現する. ただし, $B=\{0,1\}$ であり, t は区間数を表す. 本稿では, 図 4(a) の関数 $seg\_func(x)$ を図 4(b) に示されている LUT カスケード [7], [12], [13] で実現する. 関数 $seg\_func(x)$ を BDD (Binary Decision Diagram) で表現し, 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, [7], [12] 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, [13] 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］　[13] 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) [11] 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



(a) $y = c_{1i}x + c_{0i}$ の実現　(b) $y = c_{1i}(x - s_i) + f(s_i) + v_i$ の実現　(c) 等区間分割の実現

図 3　種々のアーキテクチャ

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, 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_{1i} のビット数に依存するため, 本節では, c_{1i} のビット数を削減する手法を述べる.

まず, c_{1i} の絶対値が大きい場合を考える. c_{1i} の絶対値が大きいとき, c_{1i} に必要なビット数も大きくなる. 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, [8] で示されたスケーリング手法を用いる. $|c_{1i}|$ の値が大きいとき, c_{1i} を $c_{1i} = c_{1i} \times 2^{-l_i} \times 2^{l_i}$ と表現し, 係数表には, $c_{1i} \times 2^{-l_i}$ の値と l_i の値を格納する. $c_{1i} \times 2^{-l_i}$ の値は, c_{1i} より l_i ビット小さく表現できる. そして, 乗算器で, $c_{1i} \times 2^{-l_i} \times (x - s_i)$ を計算した後, 2^{l_i} をシフト演算することでもとの値 $c_{1i}(x - s_i)$ 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_i の値を算出する. 算出した結果, 全ての区間 seg_i において, l_i の値が 0 ならば, シフト演算器を実装しない.

次に, c_{1i} の値が負の場合を考える. c_{1i} が負の値のとき, c_{1i} が正のときに比べ, 1 ビット大きくなる. そこで, c_{1i} が負の値のとき, 本合成システムでは, $|c_{1i}|$ の値と符号ビットを係数表に格納し, 乗算器で $|c_{1i}| \times (x - s_i)$ を計算した後, 2 の補数化回路によりもとの値を得る. ただし, 全ての区間 seg_i において, c_{1i} の値が正ならば, 2 の補数化回路を実装しない.

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. 図 3(b) 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 +4, 最大で LUT カスケードの段数 +6 になる. 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, 非常に高いスループット (動作周波数) を達成できる.

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: $x - s_i$	1
4.　乗算器: $c_{li}(x - s_i)$	1
5.　シフト演算器 (オプション)	1
6.　2 の補数化回路 (オプション)	1
7.　加算器: $c_{li}(x - s_i) + c_{0i}$	1
合計パイプライン段数	$n\_cas + 4 \sim n\_cas + 6$

n_cas: LUT カスケードの段数

表 3　不等区間分割と等区間分割の区間数の比較

許容近似誤差: 2^{-17}			
関数 $f(x)$	定義域	区間数	
		不等区間分割	等区間分割
$\sin(\pi x)$	$[0, 1/2]$	127	257
$\cos(\pi x)$	$[0, 1/2]$	127	257
$\tan(\pi x)$	$[0, 1/4]$	112	257
$\frac{1}{x}$	$[1/8, 1]$	702	3585
$\frac{1}{\sqrt{x}}$	$[1/32, 1]$	620	7937
$\sqrt{x}$	$[0, 1]$	231	32769
$\sqrt{-\log(x)}$	$(0, 1]$	584	32768

6.　実 験 結 果

6.1　分割アルゴリズムの計算時間

本節では, 本合成システムの効率を示すために, 分割アルゴリズムの計算時間を示す. 表 2 は, [13] の実験で使用した関数に対して, 様々な許容近似誤差で分割アルゴリズムを適用したときの計算時間を示している. 表中の *Sigmoid* と *Gaussian* は, 以下のように定義される関数である.

$$Sigmoid = \frac{1}{1 + e^{-4x}} \qquad Gaussian = \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}}$$

本分割アルゴリズムでは, 区間を再帰的に処理してゆくため, 計算時間は, 区間数に依存する. 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, 2 秒以下だった. 実験結果より, 本分割アルゴリズムは, 高速に定義域を不等区間へ分割でき, 実用的であると言える.

6.2　種々のアーキテクチャの比較

本節では, 図 3 で示した三つのアーキテクチャの比較実験を行

表 2　分割アルゴリズムの計算時間 [msec]

関数 $f(x)$	定義域	許容近似誤差: 2^{-9}		許容近似誤差: 2^{-17}		許容近似誤差: 2^{-25}	
		区間数	計算時間	区間数	計算時間	区間数	計算時間
2^x	$[0,1]$	8	0.1	128	0.1	2048	80
$\frac{1}{x}$	$[1/8,1]$	39	0.1	702	30	11218	280
$\frac{1}{\sqrt{x}}$	$[1/32,1]$	31	0.1	620	20	9946	300
$\sqrt{x}$	$[0,1]$	12	0.1	231	10	3941	110
$\sqrt{-\log(x)}$	$(0,1]$	23	0.1	584	40	12089	1840
$\log_2(x)$	$[1,2)$	8	0.1	128	10	2048	70
$\log(x)$	$[1,2)$	6	0.1	89	0.1	1437	30
$\sin(\pi x)$	$[0,1/2]$	8	0.1	127	10	2027	50
$\cos(\pi x)$	$[0,1/2]$	8	0.1	127	10	2027	50
$\tan(\pi x)$	$[0,1/4]$	7	0.1	112	10	1787	50
Sigmoid	$[0,1]$	8	0.1	127	10	2020	60
Gaussian	$[0,1/2]$	2	0.1	32	10	512	10

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　種々のアーキテクチャの実装結果

回路の精度 (確度): 16 ビット精度 (15 ビット確度)

FPGA デバイス: Altera Stratix (EP1S10F484C5)

論理合成ツール: Altera QuartusII 4.1 (ツールオプション: デフォルト設定)

関数 $f(x)$	構成 A						構成 B						構成 C					
	#LE	メモリ	乗算	Freq.	段数	時間	#LE	メモリ	乗算	Freq.	段数	時間	#LE	メモリ	乗算	Freq.	段数	時間
$\sin(\pi x)$	106	19355	8	124	7	56	107	20061	2	185	8	43	82	14848	2	188	3	16
$\cos(\pi x)$	136	19543	8	126	8	64	116	20169	2	187	9	48	67	15417	2	184	3	22
$\tan(\pi x)$	106	19355	8	125	7	56	116	20039	2	190	9	47	83	29696	2	183	3	16
$\frac{1}{x}$	153	172102	8	125	8	64	172	172119	2	179	9	50	112	278594	2	–	4	–
$\frac{1}{\sqrt{x}}$	182	159826	8	124	9	73	183	160861	2	178	10	56	145	557119	2	–	4	–
$\sqrt{x}$	191	43610	2	182	8	44	175	44359	2	179	9	50	195	1048576	0	–	1	–
$\sqrt{-\log(x)}$	226	164944	8	125	9	72	230	164957	2	176	10	57	206	1114112	0	–	1	–

各関数 $f(x)$ の定義域は, 表 3 と同じである.

"–" は, 実装できないため結果が得られなかったことを意味する.

#LE: 論理素子の個数

メモリ: メモリ量 [ビット]

乗算: 9 ビット ×9 ビット乗算器の個数

Freq.: 動作周波数 [MHz]

段数: パイプライン段数

時間: 入力から出力までの時間 (レイテンシ) [nsec]

なう. 本節では, 便宜のため, 図 3(a) のアーキテクチャを構成 A, 図 3(b) のアーキテクチャを構成 B, そして図 3(c) のアーキテクチャを構成 C と表記する. 比較のために, 私たちは, 様々な関数の数値計算回路を三つのアーキテクチャで FPGA (Altera Stratix EP1S10F484C5) に実装した. 各数値計算回路を 15 ビット確度として実装した. ただし, 近似誤差は, 2^{-17} である. 表 3 は, 不等区間分割と等区間分割の区間数, 表 4 は, 各アーキテクチャに基づく数値計算回路のハードウェア量と性能を比較している.

表 3 より, 表中の全ての関数において, 不等区間数は等区間数の半分以下である. 特に, 三角関数以外の関数では, 区間数の差は大きい. 構成 A および構成 B は, 不等区間への分割を実現できるため, 様々な関数を少ない区間数で実装できる. 一方, 構成 C では, 等区間への分割しか実現できないため, 三角関数などのように等区間分割でも区間数が少ない関数は実現できるが, 区間数が多くなる関数では, 係数表が大きくなりすぎ実装が困難になる. 実際, 構成 C では, 表中の三角関数以外の関数は, 係数表のサイズが FPGA 上のメモリサイズを超えたため, FPGA に実装できなかった.

表 4 より, 三角関数では, 構成 C での実装が最も高速で, コンパクトであることがわかる. 表 3 で示したように, 三角関数においては, 不等区間数と等区間数の差が比較的小さい. そのような関数では, 構成 A, B (不等区間分割) と構成 C (等区間分割) において, 係数表のサイズに大きな差が生じない. 構成 C は, 区間指定回路を持たないため, その分のハードウェア量とパイプライン段数を削減でき, 構成 A や B より高速でコンパクトな実装が得られた. しかし, 平方根演算や逆数演算の関数では, 構成 C は係数表が大きくなりすぎ実装できなかった. 表 4 で, 構成 C のパイプライン段数が 1 となっている関数は, 区間数が大きくなりすぎ, 係数表が関数表と等価になったことを意味している. 以上のことから, 構成 C は, 三角関数に対しては有効な実現法であるが, 平方根演算や逆数演算には不向きであることが言える.

構成 B は, 構成 A より乗算器のサイズを小さくでき, より少ない DSP ユニットで実装できる. これにより, 構成 B は, 構成 A より高速な実装が得られる. 構成 A では, 関数 $\sqrt{x}$ を除いた全ての関数において, 構成中で最も遅延時間の長い演算ユニットは, 乗算器だった. 一方, 構成 B では, 全ての関数において, 係数表が最も遅延時間の長い演算ユニットだった. 関数 $\sqrt{x}$ においては, 構成 A も係数表が最大遅延の演算ユニットだったため, メモリ量がわずかに少ない構成 A が構成 B より高い周波数を得た. 以上のことから, 高速な FPGA 実装を得るためには, 乗算器のサイズ削減が重要であり, 乗算器のサイズを効率良く削減できる構成 B が, 多くの関数に対して有効であることが言える.

6.3　従来法との比較

本節では, 自動合成システムの性能を示すために, [8] で示さ

表 5　従来法との性能比較

FPGA デバイス:	Xilinx Virtex-II (XC2V4000-6)										
論理合成ツール:	Xilinx ISE 6.3 (ツールオプション: デフォルト設定)										
関数	定義域	入力精度		出力精度		本稿の性能			[8] の性能		
$f(x)$		整数	小数	整数	小数	周波数	段数	時間	周波数	段数	時間
$\sqrt{-\log(x)}$	(0,1]	1	32	3	5	123	20	163	133	14	105
$\sin(2\pi x)$	$[0,\frac{1}{4}]$	0	16	1	8	153	10	65	133	14	105
$\cos(2\pi x)$	$[0,\frac{1}{4}]$	0	16	1	8	164	11	67	133	14	105

整数: 整数部のビット数 n_int　　小数: 小数部のビット数 n_frac
周波数: 動作周波数 [MHz]　　段数: パイプライン段数
時間: 入力から出力までの時間 (レイテンシ) [nsec]

れている性能と本合成システムにより生成された回路の性能を比較する. [8] も, 本合成と同じ不等区間分割による線形近似法を用いてるが, 簡単な区間指定回路を生成するため, 人手によるアドホックな手法で定義域を不等区間へ分割している. 表 5 は, 数値計算回路を [8] と同じ精度で実装した結果を比較している. 表 5 より, 本合成システムは, 分割点を制限することなく, 高速な区間指定回路を生成でき, 人手設計された数値計算回路 [8] と同等の性能をもつ数値計算回路を生成できる.

ページ制限のため詳細結果は省略するが, 本合成システムでは, 様々な関数に対する 24 ビット精度の数値計算回路が 125MHz 以上の動作周波数で実装できた.

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アーキテクチャ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線形近似でき, 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, 不等区間分割法は, 等区間分割法では実装できない複雑な関数に対しても, 高速でコンパクトに実装できることを示した. また, 本合成システムは, 人手によって設計された回路と同等以上の性能を持った回路を自動生成できることを示した.

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. 本研究の基礎となった [13] での Marc D. Riedel 博士の協力に感謝致します.

文　　献

[1] R. Andrata, "A survey of CORDIC algorithms for FPGA based computers," *Proc. of the 1998 ACM/SIGDA Sixth Inter. Symp. on Field Programmable Gate Array (FPGA'98)*, 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] N. Doi, T. Horiyama, M. Nakanishi, and S. Kimura, "An optimization method in floating-point to fixed-point conversion using positive and negative error analysis and sharing of operations," *Proc. the 12th workshop on Synthesis And System Integration of Mixed Information technologies (SASIMI'04)*, Kanazawa, Japan, pp. 466–471, Oct. 2004.

[4] D. H. Douglas and T. K. Peucker, "Algorithms for the reduction of the number of points required to represent a line or its caricature," *The Canadian Cartographer*, Vol. 10, No. 2, pp. 112–122, 1973.

[5] 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.

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

[7] 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.

[8] 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.

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

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

[11] 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.

[12] T. Sasao and M. Matsuura, "A method to decompose multiple-output logic functions," *41st Design Automation Conference*, San Diego, CA, pp. 428–433, June 2–6, 2004.

[13] 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.

[14] 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, Aug. 2005 (to be published).

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

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

[17] 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.

[18] M. J. Schulte and E. E. Swartzlander, "A family of variable precision interval arithmetic processors," *IEEE Trans. on Comp.*, Vol. 49, No. 5, pp. 387–397, May 2000.

[19] 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.

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