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.

FPGA 実装に適した多項式近似に基づく数値計算回路

永山　忍† 笹尾　勤†† Jon T. 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, $k+1$ 回微分可能な関数の k 次多項式近似に基づく数値計算回路の構成とその自動合成法を示す. 本数値計算回路は, 多項式の次数 k を増加することによって実装に必要なメモリ量を削減できるが, その一方で, 実装に必要な論理素子や乗算器の数が増加する. このトレードオフを考慮し, FPGA 上で使用可能なハードウェアリソースの量に応じて最も効率の良い数値計算回路を得るために, 本稿では, FPGA 使用率という尺度を導入し, 最適な次数 k を求める. 実験により以下を示す: FPGA 上のすべてのハードウェアリソースを使えるとき, 1) 低精度 (17 ビット精度まで) では, 線形近似が最も効率の良い実装をもたらす. 2) 高精度 (18 ビットから 24 ビット精度まで) では, 二次多項式近似が最も効率の良い実装をもたらす.

キーワード　不等区間分割, LUT カスケード, Chebyshev 近似多項式, 数値計算回路, FPGA 使用率.

Numerical Function Generators Based on Polynomial Approximation Suitable for FPGA Implementation

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 numerical function generators (NFGs) based on a kth-order polynomial approximation of a numerical function that is $(k+1)$-times differentiable. By increasing the polynomial order k, we can reduce the memory size of NFGs for a wide range of functions. On the other hand, larger k requires more logic elements and multipliers. To generate the most efficient NFGs depending on the unused hardware resources in an FPGA, we introduce the *FPGA utilization measure*, and find an optimum polynomial order k. Experimental results show that: when all hardware resources in an FPGA can be used for a single NFG, 1) for low-precision (up to 17 bits), 1st-order polynomial approximation produces the most efficient implementation; and 2) for high-precision (18 to 24 bits), 2nd-order polynomial approximation produces the most efficient implementation.

Key words　Non-uniform segmentation, LUT cascades, Chebyshev approximation polynomial, numerical function generators (NFGs), FPGA utilization measure.

1. はじめに

現在の主な FPGA は, 論理素子 (LE, CLB) の他に, メモリブロックや乗算器などの様々なハードウェアリソースで構成されているため, これらのハードウェアリソースを偏りなく使用する設計が望ましい. あるハードウェアリソースの偏った使用は, 他のリソースが十分余っているにも関わらず, リソース不足のため FPGA 実装できなくなる場合がある. FPGA の技術進歩に伴い, 近年では, 複数の設計が一つの FPGA に実装されるようになっているため, ハードウェアリソースの偏りのない効率的な使用が, さらに難しくなっている. この問題を解決するために, 実装するハードウェアリソースをマッピングレベルで変換する手法が提案されている [14], [27]. しかし本稿では, 数値計算回路の FPGA 実装において, より抽象度の高い設計段階でのリソース変換方法について考慮する. すなわち, 本稿では, 使用可能な (余っている) ハードウェアリソースの量に応じた最も効率の良い数値計算回路の設計について考慮する.

三角関数, 対数関数, 平方根演算, 逆数演算などの初等関数およびその合成関数は, 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, ハードウェア実装による高速化がしばしば要求される. 初等関数をハードウェア実装する手法として, CORDIC (COordinate Rotation DIgital Computer) アルゴリズム [1], [26] に代表される繰り返しアルゴリズムがしばしば用いられる. CORDIC アルゴリズムは, 加減算, シフト演

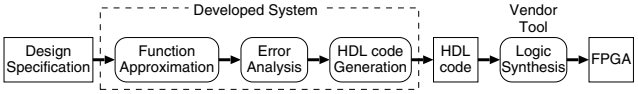


図 1　数値計算回路の合成フロー

算, およびテーブル参照の単純な演算の繰り返しにより, 初等関数を計算できるため, コンパクトな数値計算回路で初等関数を実現できるが, 一般に, 繰り返しアルゴリズムは計算時間が長く, 高速な計算が要求されるアプリケーションに不向きである. また, 複数の初等関数から成る合成関数を計算する場合, 個々の初等関数を CORDIC で順に計算していくと, さらに多くの計算時間を要するため, 合成関数を直接計算できる手法が必要である.

数値関数の高速な実装法として, 関数表を単一メモリで直接実装する方法がある. この手法は, 一回のメモリアクセスだけで関数を計算できるため, 非常に高速である. 近年のメモリの大容量化と低価格化のため, 低精度で関数を実装する場合 (入力のビット数が小さい場合), この手法は有効であるが, 高精度で実装する場合には, メモリ量が大きくなりすぎ, 実装が困難になる.

実装に必要なメモリ量を削減するために, 与えられた関数を多項式 (区分多項式) で近似し, その多項式を実現する手法が提案されている [3], [5]ー[7], [11], [12], [18], [23]ー[25]. 既存手法の多くは, 定義域を等区間に分割し, 多項式近似を行う. $\sin(x)$ や e^x などの単純な初等関数の場合, 等区間分割でも, 高次多項式を用いることで区間数やメモリ量を削減できるが, $\sqrt{-\ln(x)}$ などの複雑な関数では, 等区間分割に基づく手法は, 二次多項式を用いても必ずしも区間数やメモリ量を削減できるとは限らないことが示された [16]. 一方, 本手法は不等区間分割に基づいているので, 多様な関数において, 二次多項式を用いることでメモリ量を削減できる [16]. 高次多項式はメモリ量を削減する一方で, 乗算器や加算器が増加してしまう. このトレードオフを利用し, FPGA 上で使用可能なハードウェアリソースの量に応じて最も効率の良い数値計算回路を得るために, 本稿は, FPGA 使用率という尺度を導入し, 最小の FPGA 使用率をもたらす多項式の次数を求める.

本数値計算回路は, 自動合成できる. 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 $[a, b]$, 設計する数値計算回路の許容誤差, および近似多項式の次数 k 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 k 次多項式で近似する. 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. 本数値計算回路の誤差解析は, [17] と同様であるため割愛する.

2. 数値表現と誤差の定義

［定義 1］ **二進固定小数点**で表現された数値 r を $d_{l-1}\, d_{l-2}\, \ldots\, d_1\, d_0.\, d_{-1}\, d_{-2}\, \ldots\, d_{-m}$ と表記する. ただし, $d_i \in \{0, 1\}$ $(-m \leqq i \leqq l-1)$, l は整数部のビット数, m は小数部のビット数である. このとき r は, $r = -2^{l-1}d_{l-1} + 2^{l-2}d_{l-2} + \ldots + 2^{-m}d_{-m}$ で計算できる. 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, l は符号ビットを含む.

［定義 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 3］ **精度**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, $n = l + m$. 本稿で, n ビット精度の数値計算回路は, n ビット入力でなおかつ, 出力値の誤差が 2^{-m} 以下の回路を意味する. ただし, $m = n - 1$ (つまり $l = 1$) である.

(注1)：現在の合成システムでは, フリーソフト Scilab [22] で仕様を記述できる.

入力	関数 $f(x)$, 定義域 $[a, b]$, 許容近似誤差 ε_a, 多項式の次数 k.
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.	$\varepsilon_k(s_i, p) = \varepsilon_a$ 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　定義域の不等区間分割アルゴリズム

3. 区分多項式近似

関数 $f(x)$ を区分多項式で近似するために, x の定義域 $[a, b]$ をいくつかの区間に分割し, その各区間で, $f(x)$ を多項式近似する. この場合, 近似多項式の係数を記憶するメモリ量を削減するために, できるだけ少ない区間数で関数を近似することが求められる.

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 [3], [5]ー[7], [18], [23]ー[25]. 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. また, そのような関数において, 等区間分割に基づく手法は, 近似多項式の次数を上げたとしても数値計算回路のメモリ量を必ずしも削減できるとは限らない. 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 [11], [12], [21], 近似多項式の次数を上げることによって, 数値計算回路のメモリ量を削減できる [16]. そのため, 本稿では, 不等区間分割により関数を区分多項式近似する.

3.1 不等区間分割アルゴリズム

不等区間数は, 使用した近似多項式に依存する. つまり, 近似多項式が精確であればあるほど, 区間数は少なくなる. 本稿では, k 次の Chebyshev 近似多項式を用いる.

関数 f の区間 $[s, e]$ における k 次 Chebyshev 近似の最大近似誤差 $\varepsilon_k(s, e)$ は, 以下の式で与えられる [13].

$$\varepsilon_k(s, e) = \frac{2(e-s)^{k+1}}{4^{k+1}(k+1)!} \max_{s \leqq x \leqq e} |f^{(k+1)}(x)| \tag{1}$$

ここで $f^{(k+1)}(x)$ は, 元の数値関数の $k+1$ 次導関数を表す. (1) より, $\varepsilon_k(s, e)$ 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 $[a, b]$, 許容近似誤差 ε_a そして多項式の次数 k 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_a 以下である. 図 2 の処理 2. において, $\varepsilon_k(s_i, p) = \varepsilon_a$ 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, $\varepsilon_k(s_i, p') \leqq \varepsilon_a$ を満たす最大の点 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, $\varepsilon_k(s_i, p') \leqq \varepsilon_a$ 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, 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. $\varepsilon_k(s_i, p')$ の計算において, $\max_{s_i \leqq x \leqq p'} |f^{(k+1)}(x)|$ は非線形計画法 [9] で計算する.

3.2 近似値の計算

各区間 $[s_i, e_i]$ 毎に, 関数 $f(x)$ をそれぞれ異なる多項式関数 $g(x, i)$ 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 $[s_i, e_i]$ の多項式関数 $g(x, i) = c_k(i)x^k + c_{k-1}(i)x^{k-1} + \ldots + c_0(i)$ で計算する. ここで係数 $c_k(i), c_{k-1}(i), \ldots, c_0(i)$ は, k 次の Chebyshev 近似多項式から導出する [13]. 乗算器のサイズを削減するために, q_i を任意の値として $x - q_i + q_i$ を x に代入し, $g(x, i)$ を以下のように変形する.

$$\begin{aligned} g(x, i) \;=\; & c_k(i)(x - q_i)^k + c'_{k-1}(i)(x - q_i)^{k-1} \\ & + \ldots + c'_0(i) \end{aligned} \tag{2}$$

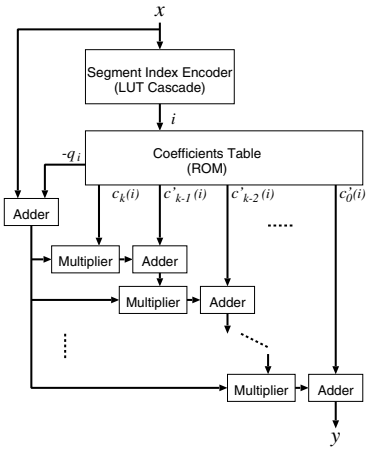
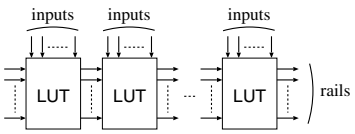


図 3　数値計算回路全体の構成

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

ただし, 各係数 $c'_j(i)$ $(j = 0, 1, \ldots, k-1)$ は, 以下で与えられる.

$$c'_j(i) = \sum_{h=0}^{k-j} \binom{j+h}{j} c_{j+h}(i) q_i^h$$

さらに, 乗算器の数をおよそ $\frac{k^2}{2}$ から k 個へ削減するために, (2) を Horner 法 [15] で計算する. Horner 法により (2) を, 以下のように変形する.

$$g(x,i) = ((c_k(i)(x-q_i) + c'_{k-1}(i))(x-q_i) + \ldots)(x-q_i) + c'_0(i) \quad (3)$$

4. 数値計算回路の構成

(3) の区分多項式は, 図 3 に示すように, x から区間番号 i を計算する区間指定回路, 係数表, 乗算器, 加算器で実現できる.

4.1 区間指定回路

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, 区間指定回路は, 図 4(a) に示されている区間指定関数 $seg\_func(x) : \{0,1\}^n \rightarrow \{0, 1, \ldots, t-1\}$ を実現する回路である. ここで, t は区間数を表す. 等区間分割に基づく数値計算回路は, 区間番号が x の上位ビットに対応しているので, 区間指定回路を必要としない. すなわち, x の上位ビットをアドレスとして, 係数表から対応する係数を読み出せる. 一方, 不等区間分割に基づく数値計算回路は, 係数表のアドレスを計算するために, この追加回路が必要になる. 不等区間分割は, 等区間分割よりも少ない区間数で関数を近似できるため, より小さな係数表で関数を実現できるが, 区間指定回路が必要になるので, この追加回路をコンパクトに実現することが重要である.

区間指定回路をコンパクトに実現するために, [11], [12] は特殊な不等区間分割法を用いた. この手法は, 区間指定回路がコンパクトになる不等区間分割を予め用意しておき, 用意された中から関数に適した分割を選ぶことで, 不等区間分割と区間指定回路を得る. 等区間分割に基づく数値計算回路に比べ, 高速でコンパクトな数値計算回路を生成できるが, 分割の選択基準があいまいであり, その選択法は自動化されていない.

本稿では, 任意の不等区間分割を高速かつコンパクトに実現するために, 関数 $seg\_func(x)$ を図 4(b) に示されている LUT カスケード [10], [19], [20] で実現する. 関数 $seg\_func(x)$ を BDD (Binary Decision Diagram) [2], [4] で表現し, その BDD を用いて関数分解することにより, LUT カスケードを得る. LUT カスケードの詳細な実現法については, [10], [19] を参照されたい. LUT カスケードは, 任意の不等区間分割を実現できるので, 図 2 のアルゴリズムで得た不等区間分割をそのまま実現でき, 分割から回路生成までのすべての処理を自動化できる. 多様な数値関数に対してコンパクトな数値計算回路を生成するためには, 任意の不等区間分割に対して LUT カスケードのサイズを保証することが重要である. [20] で, LUT カスケードは, 任意の不等区間分割をその区間数に依存するサイズで実現できることが示された. そこで, 本稿では, 高次多項式近似を用いて区間数を削減することにより係数表のサイズだけでなく, LUT カスケードのサイズも大幅に削減し, FPGA 上のメモリ量が少ない場合でも実装できるようにする.

4.2 乗算器のサイズ削減

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. 乗算器のサイズを削減するために, 本節では, $(x - q_i)$ のビット数を削減する手法を述べる. $(x - q_i)$ のビット数の削減は, 乗算器のサイズだけでなく, 丸め誤差をも削減する [8]. (2) 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^u ワードの ROM で実現される. ただし t を区間数としたとき, $u = \lceil \log_2 t \rceil$ である. これは, 区間数を 2^u 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 カスケードのサイズも, u に依存するため, 区間数を 2^u 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 [20]. そこで, 区間数が 2^u 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. FPGA 使用率

現在の主な FPGA は, 論理素子, メモリブロック, 乗算器などの様々なハードウェアリソースで構成されているため, これらのハードウェアリソースを効率良く使用する設計が望ましい. 使用可能なハードウェアリソースの量に応じて, 最も効率の良い数値計算回路を生成するために, 本節では, ***FPGA* 使用率**という尺度を導入する.

［定義 4］ ある設計と FPGA 上の使用可能なハードウェアリソースが与えられたとき, ***FPGA* 使用率** U を, 与えられた設計における各ハードウェアリソースの使用率の和で定義する. 本稿では, FPGA 上には, 4 種類のハードウェアリソース: 論理素子 (LE), 乗算器 (DSP), および二種類のメモリブロック (M4K と M512) があると仮定する. すると FPGA 使用率 U は以下の式で計算できる.

$$U = \left(\frac{\text{R\_LEs}}{\text{A\_LEs}} + \frac{\text{R\_DSPs}}{\text{A\_DSPs}} + \frac{\text{R\_M4Ks}}{\text{A\_M4Ks}} + \frac{\text{R\_M512s}}{\text{A\_M512s}} \right) \times 100\%$$

ただし, R_LEs は, 設計に必要な LE 数を表し, A_LEs は, 使用可能な LE 数を表す. DSP, M4K, M512 においても, 同様の表記法を用いている.

この尺度を使って, 最も効率の良い数値計算回路を生成する多項式の次数 k を見つける. すなわち, 最小の FPGA 使用率をもたらす多項式の次数 k を見つける. 設計に必要なリソースの量が使用可能なリソースの量を上回ったとき (例えば, A_LEs < R_LEs のとき), 尺度にペナルティー (例えば, $U = \infty$) を用いることで実装可能な設計だけを見つけることもできるが, 本稿の実験では, そのようなペナルティーを使用しなくても単に最小の FPGA 使用率を探すだけで, 実装可能かつ効率的な設計を見つけることができた.

表 1　$\sqrt{-\ln(x)}$ および $\arcsin(x)$ における等区間数と不等区間数.

関数 $f(x)$	定義域	次数	等区間数	不等区間数
$\sqrt{-\ln(x)}$	(0,1)	1	8,388,607	8,230 (0.0981%)
		2	8,388,607	698 (0.0083%)
		3	8,388,607	213 (0.0025%)
		4	8,388,607	111 (0.0013%)
		5	8,388,607	75 (0.0009%)
$\arcsin(x)$	[0,1)	1	8,388,608	3,067 (0.0366%)
		2	8,388,608	256 (0.0031%)
		3	8,388,608	81 (0.0010%)
		4	8,388,608	45 (0.0005%)
		5	4,194,304	31 (0.0007%)

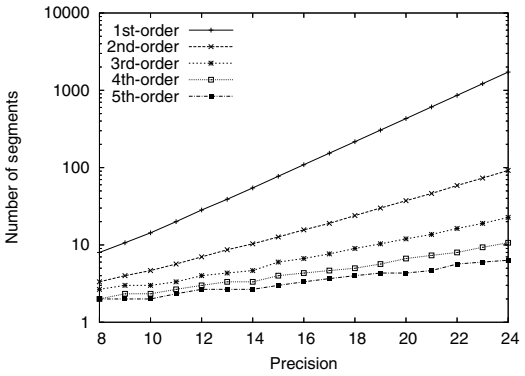


図 5　不等区間数と精度の関係.

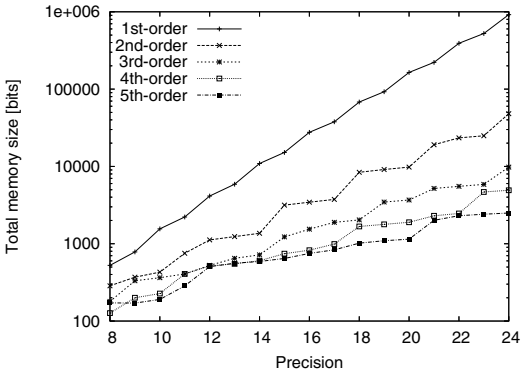


図 6　総メモリ量と精度の関係.

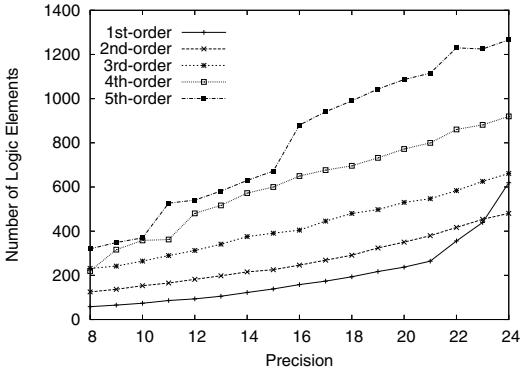


図 7　論理素子数と精度の関係.

6. 実 験 結 果

本節では, 前節で述べた FPGA 使用率を用いて, 与えられた精度における最適な多項式の次数を求める.

6.1 区間数とメモリ量

表 1 は, $\sqrt{-\ln(x)}$ と $\arcsin(x)$ の 24 ビット精度の数値計算回路に必要な等区間数と不等区間数を比較している. この表から, 不等区間数は, 多項式の次数の増加に従って, 大幅に減少するが, 等区間数は, 必ずしも減少しないことがわかる. 区間数は係数表のサイズに直接影響するため, この表から, 等区間分割に基づく多くの既存手法は, これらの関数においては, 多項式の次数を増加してもメモリ量を削減できないことがわかる. 一方, 不等区間分割に基づく本手法は, 多様な関数において, 多項式の次数を増加することでメモリ量を削減できる.

以下の実験では, 標準的な 3 つの数値関数: $\cos(\pi x)$ $(0 \leqq x \leqq 1/2)$, $\sqrt{x}$ $(1/32 \leqq x < 2)$, $1/x$ $(1 \leqq x < 2)$ を使った. 実験結果は, これらの関数の平均値である.

図 5 は, 不等区間数と精度の関係を示している. 図より不等区間数は精度とともに増加し, 多項式の次数を増加することで, 不等区間数を大幅に削減できることがわかる. 例えば, 24 ビット精度において, 5 次多項式は 1 次多項式のわずか 0.37% の不等区間数で関数を近似できる.

図 6 は, 総メモリ量と精度の関係を示している. 本数値計算回路は, 係数表だけでなく, 区間指定回路もメモリで実装しているため, それらの和が本数値計算回路に必要なメモリ量となる. 図よりメモリ量は精度に対し指数関数的に増加することがわかる. また, 図 5 と図 6 から, 本数値計算回路のメモリ量は不等区間数に強く依存していることがわかる. そのため, 区間数同様, 多項式の次数を増加することで, メモリ量を削減できる. 特に, 高精度において多項式の次数の増加は, メモリ量を大幅に削減する. 24 ビット精度において, 5 次多項式は 1 次多項式のわずか 0.27% のメモリ量で関数を実現できる.

しかし, 多項式の次数を上げると係数の数が増加するため, 低精度においては, 区間数 (係数表のワード数) でのわずかな削減が, 係数の数 (係数表のビット幅) の増加でうち消されてしまい, 総メモリ量が必ずしも減少しないことに注意. 実際に, $\sqrt{x}$ の 8 ビット精度の数値計算回路において, 5 次多項式と 4 次多項式は, 共に同じ区間数で関数を近似できるため, 5 次多項式は, 4 次多項式より多くのメモリ量を必要とする.

6.2 FPGA 上のハードウェアリソース

本数値計算回路における, ハードウェアリソースと多項式の次数および精度の関係を調べるために, パイプライン化した数値計算回路をアルテラ Stratix EP1S80F1020C5 FPGA に実装した. 論理合成ツールは, Quartus II ver. 5.0 を使用し, 速度最適化オプションおよびタイミング制約 200MHz で合成した.

図 7 は, 論理素子数と精度の関係を示している. この図から, 論理素子数は, 精度に対しおよそ線形に増加することがわかる. また, 次数を上げると論理素子数も増加する. 8 から 15 ビット精度における 5 次多項式と 8 から 11 ビット精度における 4 次多項式は, 一つの多項式 (区間) で $\cos(\pi x)$ を近似するので, 係数表や区間指定回路を必要としない. そのため, それらのメモリアドレスレジスタが不要になり, 論理素子数が他の精度に比べ極端に少なくなっている. 22 から 24 ビット精度における 1 次多項式は, 不等区間数が多いため区間指定回路 (LUT カスケード) のサイズが大きい. そのため LUT カスケードのパイプライン段数が増え, それに伴い論理素子 (パイプラインレジスタ) 数が極端に増加している.

図 8 は, DSP (9×9 ビット乗算器) 数と精度の関係を示している. この図から, DSP 数も, 精度とともに増加することがわかる. また, 次数を上げると DSP 数も増加する. 24 ビット精度において, 5 次多項式は 1 次多項式の 20 倍もの DSP を必要とする.

図 9 は, メモリブロック数と精度の関係を示している. 実験で使用した FPGA には, M4K と M512 の二種類のメモリブロックがあるので, それらの和を図に示した. ここで, M4K と M512 を同じ 1 ブロックとして数えた. 図よりメモリブロック数は精度に対し指数関数的に増加することがわかる. また, 次数の増加でメモリブロック数を削減できることがわかる. 24 ビット精度において, 5 次多項式は 1 次多項式のわずか 3% のメモリブロック数で関数を実装できる.

以上の結果から, 本数値計算回路は, 多様な関数に対して, 多項式の次数を変更することで, 実装に必要な FPGA 上のハードウェアリソースの量を柔軟に変更できることがわかる. すなわ

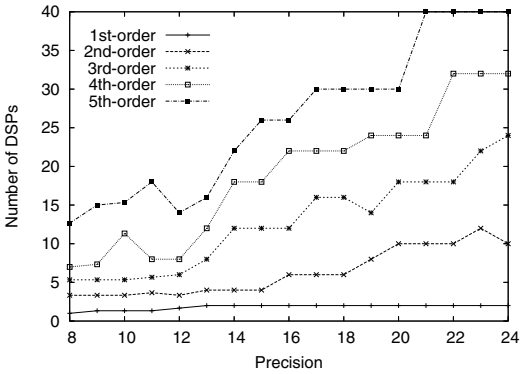


図 8　DSP 数と精度の関係.

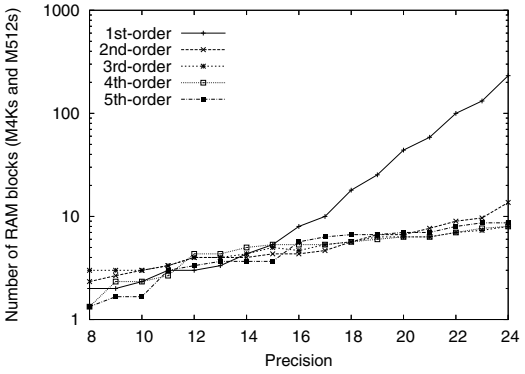


図 9　メモリブロック数と精度の関係.

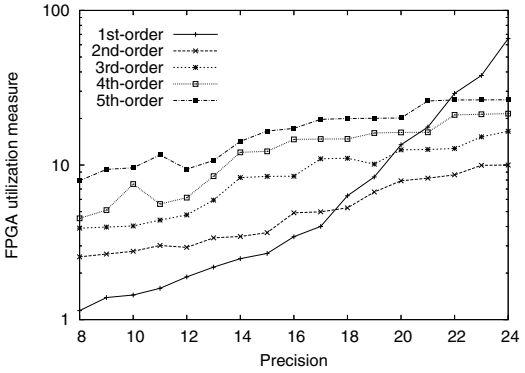


図 10　Stratix EP1S80F1020C5 上のすべてのハードウェアリソースが使用可能な場合の FPGA 使用率と精度の関係.

ち, 本手法は, FPGA の使用状況に応じて多項式の次数を変更することで, 最適な数値計算回路を生成できる.

6.3 FPGA 使用率

本節では, FPGA 使用率を用いて, 与えられた精度と使用可能なハードウェアリソースにおける最適な次数を見つける.

図 10 は, Stratix EP1S80F1020C5 上のすべてのハードウェアリソースが一つの数値計算回路のために使用可能な場合の FPGA 使用率と精度の関係を示している. この FPGA には, 79, 040 個の論理素子, 176 個の DSP, 364 個の M4K, 767 個の M512 がある. 図より, 低精度 (17 ビットまで) では, 1 次多項式が最小の FPGA 使用率をもたらし, 高精度 (18 ビットから 24 ビット) では, 2 次多項式が最小の FPGA 使用率をもたらすことがわかる. 先に示したように高次多項式を用いることで, 不等区間数やメモリ量を大幅に削減できるにも関わらず, 低次多項式が最も効率の良い実装をもたらしたのは, 非常に驚くべき結果である. 実

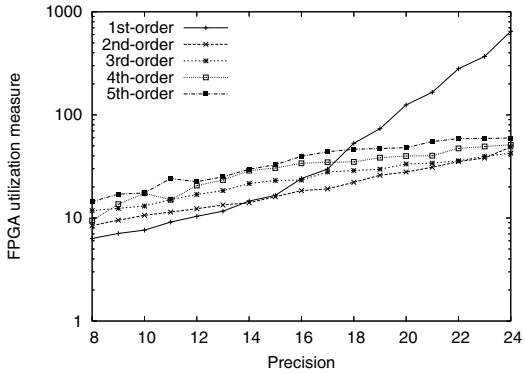


図 11　EP1S80F1020C5 上の 10% の論理素子とメモリブロックおよび 100% の DSP が使用可能な場合の FPGA 使用率と精度の関係.

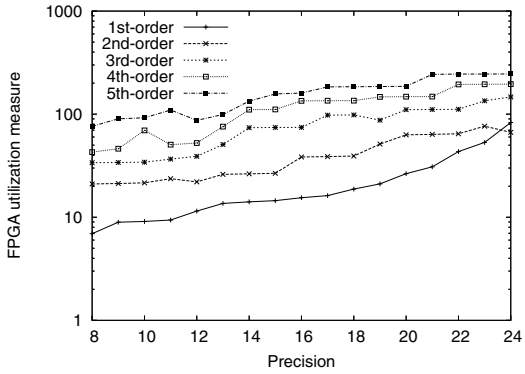


図 12　EP1S80F1020C5 上の 10% の論理素子と DSP および 100% のメモリブロックが使用可能な場合の FPGA 使用率と精度の関係.

験に使用した FPGA には, DSP より多くのメモリブロックがあるため, DSP を多く使用する高次多項式よりもメモリを多く使用する低次多項式が効率の良い実装を生成した.

そこで, FPGA 上の 10% の論理素子とメモリブロックが使用可能な状況を想定した. すなわち, 論理素子とメモリブロックの 9 割が他の設計ですでに使用されており, 残りの 1 割と DSP を使って数値計算回路を実装することを想定した. 図 11 は, この状況での FPGA 使用率を示している. 図より, 13 ビット以下の精度では, 1 次多項式が最小の FPGA 使用率をもたらし, 14 ビットから 23 ビットの精度では, 2 次多項式が最小であり, そして 24 ビット精度では, 3 次多項式が最小の FPGA 使用率をもたらすことがわかる. この FPGA の使用状況では, 20 ビット以上の精度における 1 次多項式は, メモリ不足のため実装できないが, 2 次以上の多項式は, 実装可能である. 図 12 は, FPGA 上の 10% の論理素子と DSP および 100% のメモリブロックが使用可能な状況での FPGA 使用率を示している. 図より, 23 ビット以下の精度では, 1 次多項式が最小の FPGA 使用率をもたらし, 24 ビット精度では, 2 次多項式が最小の FPGA 使用率をもたらすことがわかる. この状況では, 3 次以上の多項式は, DSP 不足のため実装できない.

FPGA 上のすべてのリソースがより少ない状況を想定し, 低コスト FPGA, Cyclon II EP2C5F256C6 を用いて実験した. この FPGA は, Cyclone II ファミリの中で最小のデバイスであり, 4, 608 個の論理素子, 26 個の DSP, 26 個の M4K, 0 個の M512 がある. 図 13 は, この FPGA における実験結果を示している. 高精度において, 1 次多項式は, メモリブロックが不足し, 一方, 4 次と 5 次多項式は, DSP が不足するため実装できない. 図から, 15 ビット精度までは, 1 次多項式が最小の FPGA 使用率をもたらし, 16 ビット精度から 24 ビット精度では, 2 次多項式が最小の FPGA 使用率をもたらすことがわかる.

以上の実験では, 4 次と 5 次多項式の有用性を示せなかったが, 図 6 から, 24 ビットより高い精度で, 4 次と 5 次多項式はメモリ量を削減するために有効だろうと推測できる. 残念ながら, C 言語で開発した合成システムの精度のため, 24 ビット精度を

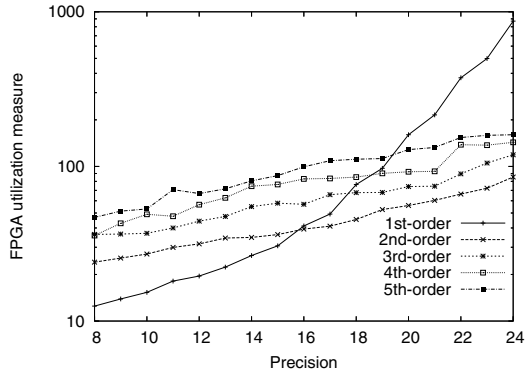


図 13　Cyclone II EP2C5F256C6 における FPGA 使用率と精度の関係.

超える数値計算回路の精確さを検証できず, 24 ビット精度までの結果しか得ていない. 高精度のための合成システムの開発および検証ツールの開発が今後の課題である.

7. 結論とコメント

本稿は, $k+1$ 回微分可能な関数の k 次多項式近似に基づく数値計算回路の構成と合成法を示し, 最も効率の良い数値計算回路を生成するために, FPGA 使用率という尺度を導入した. 実験により以下を示した: 1) 等区間分割に基づく従来手法は, 関数によっては, 高次多項式を用いても必ずしもメモリ量を削減できるとは限らない. 一方, 不等区間分割に基づく本手法は, 多様な関数に対して, 多項式の次数を変更することで, 実装に必要なハードウェアリソースの量を柔軟に変更できる. 2) FPGA 上のすべてのハードウェアリソースが一つの数値計算回路のために使用できるとき, 低精度 (17 ビット精度まで) では, 線形多項式が最も効率の良い FPGA 実装を生成し, 高精度 (18 ビットから 24 ビット精度まで) では, 二次多項式が最も効率の良い FPGA 実装を生成した. 使用可能なハードウェアリソースの量が制限されていても, 精度とリソース制約に適した多項式の次数を見つけることができる.

与えられたハードウェアリソースの量から, 最適な次数を自動的に算出し, その次数の多項式近似に基づく数値計算回路を自動合成するシステムを現在開発中である. この合成システムでは, 数値計算回路に必要なハードウェアリソース量の正確な見積りが重要である. 本稿の実験で, 最適な次数の存在が明らかになり, そのようなシステム開発の目処がついた.

謝辞: 本研究の一部は, 平成 18 年度科学研究費補助金 (若手研究 (B)) 課題番号 18700048 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] R. E. Bryant, "Graph-based algorithms for boolean function manipulation," *IEEE Trans. Comput.*, Vol. C-35, No. 8, pp. 677–691, Aug. 1986.

[3] 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.

[4] E. M. Clarke, K. L. McMillan, X. Zhao, M. Fujita, and J. Yang, "Spectral transforms for large Boolean functions with applications to technology mapping," *Proc. of 30th ACM/IEEE Design Automation Conference*, pp. 54–60, June 1993.

[5] 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.

[6] 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.

[7] J. Detrey and F. de Dinechin, "Table-based polynomials for fast hardware function evaluation," *16th IEEE Inter. Conf. on Application-Specific Systems, Architectures, and Processors (ASAP'05)*, pp. 328–333, 2005.

[8] 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.

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

[10] 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.

[11] 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.

[12] D.-U. Lee, W. Luk, J. Villasenor, and P. Y. K. Cheung, "Hierarchical segmentation schemes for function evaluation," *Proc. of the IEEE Conf. on Field-Programmable Technology*, Tokyo, Japan, pp. 92–99, Dec. 2003.

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

[14] G. W. Morris, G. A. Constantinides, and P. Y. K Cheung, "Using DSP blocks for ROM replacement: a novel synthesis flow," *Proc. Inter. Conf. on Field Programmable Logic and Applications (FPL'05)*, Tampere, Finland, pp. 77–82, Aug. 2005.

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

[16] S. Nagayama, T. Sasao, and J. T. Butler, "Programmable numerical function generators based on quadratic approximation: architecture and synthesis method," *Proc. of Asia and South Pacific Design Automation Conference (ASPDAC'06)*, Yokohama, Japan, pp. 378–383, 2006.

[17] 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/.

[18] J.-A. Piñeiro, S. F. Oberman, J.-M. Muller, and J. D. Bruguera, "High-speed function approximation using a minimax quadratic interpolator," *IEEE Trans. on Comp.*, Vol. 54, No. 3, pp. 304–318, Mar. 2005.

[19] 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.

[20] 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.

[21] 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.

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

[23] M. J. Schulte and J. E. Stine, "Symmetric bipartite tables for accurate function approximation," *13th IEEE Symp. on Comput. Arith.*, Asilomar, CA, Vol. 48, No. 9, pp. 175–183, 1997.

[24] 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.

[25] 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.

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

[27] S. J. E. Wilton, "Heterogeneous technology mapping for area reduction in FPGA's with embedded memory arrays," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, Vol. 19, No. 1, pp. 56–68, Jan. 2000.