

高基数STL法を用いたFPGA向き対数関数計算法

藤原 康史^{1,a)} 高木 一義^{1,b)} 高木 直史^{1,c)}

概要: 本研究では, IEEE754-2008 標準で推奨されている倍精度浮動小数点対数関数の正確丸めを保証する FPGA 向きの計算法を提案する. 本計算法では, 高基数 STL 法とテイラー展開を用いて値を計算する. また正確丸めに必要な制度を低減するため, 一定の精度で丸めを誤る入力を予め特定しておき, それを用いて修正を行う. 提案する計算法を FPGA 搭載の DSP の乗算器のサイズに最適化することで, 高基数 STL 法で用いるテーブルのサイズの低減を図る手法についても述べる.

Calculation method of logarithmic function on FPGAs using high-radix STL method

YASUFUMI FUJIWARA^{1,a)} KAZUYOSHI TAKAGI^{1,b)} NAOFUMI TAKAGI^{1,c)}

Abstract: We propose a calculation method of double precision floating point logarithm function for FPGA with correct rounding recommended in the IEEE 754-2008 standard. In this calculation method, values are calculated by using high-radix STL method and Taylor expansion. And required computational accuracy for correct rounding is reduced by specifying in advance the inputs incorrectly rounded and using it to fix. Moreover, We describe how to reduce the size of the table used in the high radix STL method by optimizing the proposed calculation method to the size of the multiplier of the DSP mounted on the FPGA.

1. はじめに

1985 年に策定された浮動小数点演算の標準規格 IEEE-754 が 2008 年に改定され, 36 の関数の計算を正確に行うことを推奨している.[1] 正確な計算とは, 計算結果を丸めモードに応じて正確に丸めた値を返すことを示す. この改定に合わせ, ソフトウェアでは正確な丸めを高速に行う研究が行われ [2][3], 正確丸めを行う単精度および倍精度の科学技術計算ライブラリが整備されつつある [4][5][6].

科学技術計算では, 高速に大規模な演算が求められるため, CPU に加えアクセラレータが用いられる. アクセラレータの選択肢として FPGA が存在する.

FPGA (field-programmable gate array) では, 回路の再構成を行うことで同じ FPGA 上に異なる論理を実装することができるため, 計算前に用いる計算に特化した計算回

路を構成することで性能の向上が期待できる. よって, アクセラレータ用の 2 進倍精度浮動小数点関数の FPGA ライブラリが科学技術計算に有用であると考えられる.

本稿では, IEEE754-2008 標準で推奨される 2 進倍精度対数関数の FPGA 向き計算法の提案を行う.

対象とする FPGA は, DSP ブロックが搭載されているものであり, DSP ブロックでは一定精度の整数乗算, 及び近年の FPGA では加えて浮動小数点乗算を高速に行うことが可能である.

DSP ブロックの乗算器を並列に用いて高速に動作可能な短冊形乗算を構築可能であり, FPGA により搭載されているリソース量が異なる. そのため短冊形乗算器とテーブル参照を用いた計算法が FPGA 向きであるといえる.

本計算法では, 高基数の STL 法とテイラー展開を用いて対数関数を計算する. 高基数 STL 法はテーブルと短冊形乗算器を用いて計算される. 高基数 STL 法とテイラー展開の構成によりテーブルの使用量と乗算器の使用量をバランスすることが可能である. また IEEE754-2008 標準では, 演算において計算結果を丸めモードに応じて正確に丸

¹ 京都大学
Kyoto University

a) fujiwara@lab3.kuis.kyoto-u.ac.jp

b) ktakagi@lab3.kuis.kyoto-u.ac.jp

c) takagi@i.kyoto-u.ac.jp

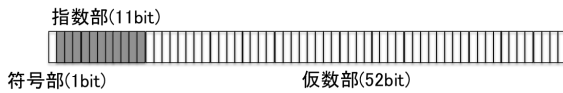


図 1 倍精度浮動小数点形式

めた値を返すことが要求される。正確な丸めを行うためには真値と丸め境界値を正確に比較する必要がある、この値が近い場合には高い精度が必要になる。

必要精度を低減するために、ある定めた精度で丸めを誤る入力を予め特定しその入力に対してのみ丸め前に修正を行う。

高基数 STL 法に用いる基数と回数およびテイラー展開の次数は必要精度になるように定めるが、この値の定め方によって性能とリソース使用量が変化する。とくに乗算器の使用量とテーブルの使用量が多いため、この値に注目しパラメータを決定する。

また基数を FPGA の乗算器サイズに応じて定めることでリソースの使用量を低減することが可能である。これを用いて FPGA デバイスに応じた最適化を行い、更にリソースの使用量を削減する手法についても述べる。

この計算法を用いて、Zynq-7000 All Programmable SoC XC7Z020-CLG484-1 のプログラマブルロジックを対象にパラメータの値を決定しと実装を行った。最大動作周波数は 118MHz、リソース使用量で最大のものは 13.54%となった。

2. 準備

本研究では IEEE754-2008 標準で推奨されている、2 進倍精度浮動小数点の正確丸め対数関数の計算を対象とする。本節では IEEE754-2008 標準、計算法で用いる対数関数の性質について述べる。また、対象とする FPGA について述べる。

2.1 IEEE 754-2008 浮動小数点演算標準

IEEE754-2008 標準では、浮動小数点数の表現形式が定められている。また丸めモードと演算が定義されており、算術演算では丸めモードに応じた正確な結果を返すことが要求される。

2.1.1 表現形式

2 進倍精度浮動小数点標準形式について説明する。表現形式は図 1 のように 3 部分に別れ、指数部、仮数部の値に応じて表現する値が異なる。以下では指数部を e 、仮数部を f 、符号を s で示す。

表 1 に表現する値を示す。

2.1.2 丸めモード

IEEE754-2008 標準では 2 進の浮動小数点数に対して、次の 4 種類の丸めモードが定義されている。

指数部	仮数部	種類	値
0	0	正負 0	$(-1)^s * 0$
0	非 0	非正規化数	$(-1)^s * 0.f * 2^{-1022}$
2047	0	正負無限大	$(-1)^s * \infty$
2047	非 0	非数	NaN
その他		正規化数	$(-1)^s * 1.f * 2^{e-1023}$

表 1 浮動小数点の表現と値

- 最近接丸め (偶数)
- 方向丸め (0)
- 方向丸め ($+\infty$)
- 方向丸め ($-\infty$)

最近接丸めは最も近い表現可能な数に丸めるモードであり、値がちょうど中間の場合は最下位ビットが 0 になる方向に丸める。方向丸めは記載された方向に丸めるモードである。

2.1.3 対数関数

IEEE754-2008 標準では表示形式の数に対し 36 個の関数計算が要求されるが、その際、丸めモードに応じた正確な値を返すことが推奨される。対数関数は次の 6 つである。

- $\log$ $\log(x)$
- $\log_2$ $\log_2(x)$
- $\log_{10}$ $\log_{10}(x)$
- $\logp1$ $\log(1+x)$
- $\log2p1$ $\log_2(1+x)$
- $\log10p1$ $\log_{10}(1+x)$

以下では $\log$ は底がネイピア数の対数とする。

2.2 テーブルメーカーズジレンマ

丸め境界値と真値が近い場合、計算誤差による丸め誤りを除外するために非常に高い精度が必要になる。すべての入力に対して真値の丸め結果と関数値の丸め結果が一致させる問題をテーブルメーカーズジレンマという。

対数関数の場合すべての入力に対して正確に丸めるためには 153bit の精度が必要になる。

2.3 対数関数の計算法

以下では浮動小数点数を $M * 2^E$ で表す。ただし $1 \leq M < 2$ であり、 E は整数とする。

$$\log(M * 2^E) = \log(M) + E \log(2) \quad (1)$$

E は整数及び $\log(2)$ は定数より $E \log(2)$ は容易に求まる。よって $\log(M)$ を求めればよい。以下では $\log(M)$ の求め方について述べる。

2.3.1 高基数 STL 法

対数関数について以下が成り立つ。

$$\log(P) + Q = \log(P * A) + (Q - \log(A)) \quad (2)$$

よって $P_0 = M$, $Q_0 = 0$ として

$$P_{i+1} = P_i * A_i \quad (3)$$

$$Q_{i+1} = Q_i - \log A_i \quad (4)$$

とする.

$$P_i = M * \prod_{j=1}^{i-1} A_j \quad (5)$$

$$Q_i = -\sum_{j=1}^{i-1} \log(A_j) \quad (6)$$

よって, 任意の i に対して以下が成り立つ.

$$\log(P_i) + Q_i = \log(P_0) + Q_0 = \log(M) \quad (7)$$

A_i は P_i の上位数ビットからテーブル参照等を用いて求める P_i の逆数の近似値とすると, $P_i \rightarrow 1$ となるので, $Q_i \rightarrow \log(M)$ となる. $\log(A_i)$ もテーブル参照等により求める.

2.3.2 テイラー展開を用いた計算法

対数関数をテイラー展開すると以下ようになる.

$$\log(1+y) = \sum_{i=1}^{\infty} (-1)^{i-1} \frac{y^i}{i} = y - \frac{y^2}{2} + \frac{y^3}{3} \dots \quad (8)$$

展開式を用いて多項式により計算する.

2.4 FPGA

FPGA での高速な計算では並列性, 使用リソースの量とバランスが重要になる. 本稿では乗算器を搭載した汎用 DSP ブロックを持つ FPGA を対象とする. DSP ブロックを持つ FPGA では小さな乗算及び桁上げ処理を高速に行えるため, 短冊型乗算を用いた計算が適する.

3. 対数関数の FPGA 向き計算法

3.1 計算フロー

計算フローは図 2 のようになる. 無限大, 非正, NaN に対しては適切な出力および例外判定を行う. 出力および例外は表 2 のようになる.

入力	出力, 例外
NaN	NaN
負数	invalid input
$+\infty$	$+\infty$
0	divide by zero

表 2 特殊ケースの出力, 例外

非正規化数に対しては, 正規化を行い, $M \cdot 2^E$ ($1 \leq M < 2$) の形に変形する. 次に正規化された数及び正規化数に対して, $\log(2^E \cdot M) = E \log(2) + \log(M)$ を用いて, 指数と仮数を分解する. $\log(2^E) = E \log(2)$ を用いて指数部の対数の値を求める. 指数が -1 かつ仮数が 2 に近い場合桁落ちが発生するため, $M > 1.5$ である場合には $M' = \frac{M}{2}, E' = E + 1$ とすることにより, 桁落ちを防ぐ. そ

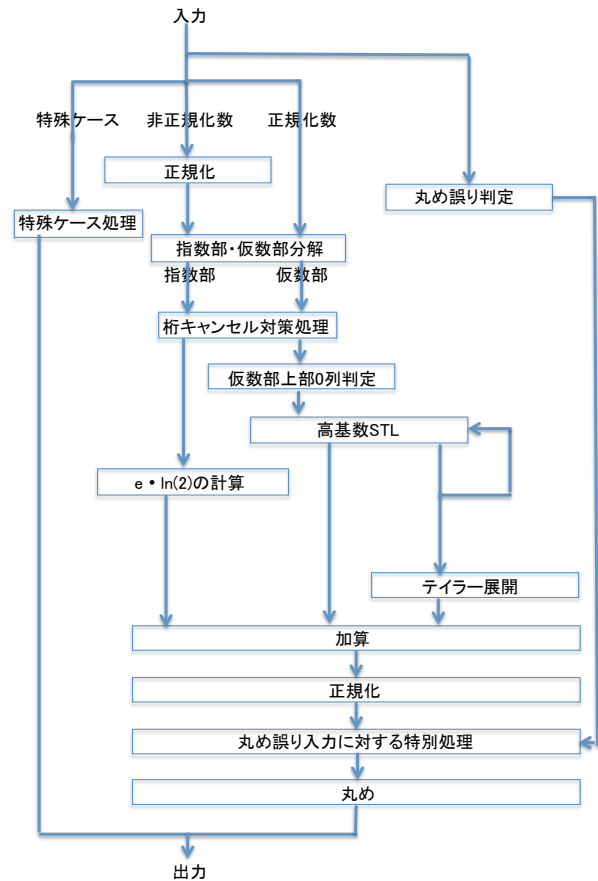


図 2 計算フロー

うでない場合は $M' = M, E' = E$ とする.

仮数に対して高基数 STL 法を用いて計算を行い, その後テイラー展開による計算を行う. この際, 仮数小数部の上位が 0 列であるかによって場合分けを行うことで, 仮数部上位が 0 である場合に相対精度が低下する問題に対処し, 高基数 STL 法の必要回数を低減する. その後, 指数と仮数の対数の値を加算し, 正規化し, 丸めを行う.

3.2 計算精度と丸め

テーブルメーカーズジレンマより, すべての入力に対して正しく丸めるためには計算に非常に高い精度が必要になる. しかし, 丸めが困難な入力少数である. そのため, ある程度の精度 (N ビット) で計算を行い, その精度で丸めを誤る入力を予め求めておき, 丸めを誤る場合には補正を行うようにすることで正確丸めに必要な精度を低減する.

丸めを誤る入力に関しては最近接丸めの場合は丸めビットを反転させる. また, 方向丸めの場合は丸める方向に応じて, 仮数部の最下位ビットと丸めビットの演算を行い, 仮数部の最下位ビットを修正する.

丸めを誤る入力は最大相対誤差におおよそ比例して増加する. 倍精度浮動小数点演算の丸めに関する研究 [7][8][9] のデータをもとに丸めを誤る入力の特定制を行う.

表 3 に真値と丸め境界の距離が真値に対して一定以下で

あるような入力の個数を示す. これは最大誤差がこの値である場合に, 丸めを誤る可能性のある入力の個数を示している. $\exp$ は真値を浮動小数点で表したときの指数を指す.

これらの中から, 丸め困難な入力から丸めを誤る入力を特定する.

また $E = 0$ かつ $M \approx 1$ である場合, 相対的に精度が低下するため, $M \approx 1$ であるかどうかで場合分けを行い, 必要な精度を確保する.

3.3 $\log(M')$ の計算

$E' = 0$ かつ $M' \approx 1$ の時, 相対的に精度が低下する. そのため, 仮数 M' の小数部の上位が 0 列である場合とない場合に分けて処理を行う.

3.3.1 $M' \approx 1$ でない場合

高基数 STL 法を適用し, テイラー展開を行う.

高基数 STL 法の 1 回目は A_0 は $1 < P_1 < 1 + 2^{-k}$ となるように, M' の上位 $k_1 + 1$ ビットを用いてテーブル参照を行う. また $\log(A_i)$ もテーブル参照で求める. テーブルは予め計算して用意しておく.

テーブル使用量は $2^{k+1} * (N + k + 4)$ bit であり, 必要な乗算サイズは $52\text{bit} * (k + 2)$ bit である

2 回目以降は $P_i = 1 + F_i$ ($F_i < 2^{-\sum k_i}$) である時, $A_i = 1 - F_i + C$ とすれば良い. ただし C は定数であり, $C = 2^{-\sum k_i}$ もしくは $C = 2^{1-\sum k_i}$ である. これにより逆数を求める際にテーブル参照が不要になる.

また $\log(A_i)$ も仮数上位が F_i に一致するため, テーブル使用量を削減可能である. 同様に回数が進めば上位が F_i , $\frac{F_i^2}{2}$, $\frac{F_i^3}{3}$ が順に桁として重ならなくなるため, F_i , $\frac{F_i^2}{2}$ 及び $\frac{F_i^3}{3}$ に上位桁が一致する. これを用いることでテーブル参照が必要な桁数が減少し, テーブル使用量を削減可能である.

テーブル使用量は $(N + 2 - \alpha) * 2^{k+1}$, 必要な乗算サイズは $(N + 2)\text{bit} * (k + 2)\text{bit}$ である.

高基数 STL 法により $\log(M) = \log(P_n) + Q_n$ (ただし

$P_n < 1 + 2^{-\sum k_i}$) としたのち, $\log(P_n)$ をテイラー展開により計算する. ここで用いるテイラー展開の次数を D , M の仮数上位にある 0 列の長さを L とすると, $\log(M)$ の精度は $K * D - \lceil \log_2(K + 1) \rceil - L$ となる.

また, 大きな基数で行う高基数 STL 法では対数を以下の式を用いて近似式と複数のテーブルから計算することでテーブルの使用量を低減する事が可能である.

$$\log(1 + z) = \log(1 + z_H + z_L) \quad (9)$$

$$\begin{aligned} &\sim z - \frac{z^2}{2} + \frac{z^3}{3} \\ &= z - \frac{z_H^2 + z_L^2}{2} + \frac{z_H^3 + z_L^3}{3} + z_H z_L (z - 1) \end{aligned}$$

$\frac{z_H^3}{3}$ 及び $\frac{z_L^3}{3}$ をテーブルを用いて, 残りを直接計算することで対数の値を求める.

3.3.2 $M' \approx 1$ の場合

仮数部小数部が 0 列からなり, さらに指数部が 0 である場合, 高基数 STL 法を適用した際に確保される相対精度が低下する. しかし仮数部小数部が 1 個の 0 列からなる場合, 高基数 STL 法を 1 桁以下の任意の部分から適用することが可能である. そのため場合分けを行い, 仮数上位が 0 列の場合は, 途中の部分から同じ回数の高基数 STL 法を適用することで高い絶対精度が確保可能になる. これにより, テイラー展開の次数を D とすると, 場合分けしない場合と比較して最悪精度が $D * l$ 桁改善する.

4. 実現と評価

4.1 評価環境

Zynq-7000 All Programmable SoC XC7Z020-CLG484-1[10] のプログラマブルロジック (Xilinx Artix-7 FPGA ファミリー相当) を対象にパラメータ推定, レイテンシを高める方針で回路設計を行い性能を評価する. プロセッサコアは演算に用いない. 搭載されているリソース量は次のとおりである.

- LUT:53200
- FF:106400
- 25bit*18bit の 2 の補数乗算器が搭載された DSP:220
- 36Kb サイズの BRAM:140 (4.9Mb)

4.2 概要

乗算器は DSP に含まれる乗算器を用いて構成されるとし, テーブルは 2 ポート ROM で構成する. 前処理を 1 ステージ, 高基数 STL 法を 1 回の近似ごとに 1 ステージ, テイラー展開 1 ステージ, 後処理 1 ステージの計 7 ステージのパイプラインとして実装する.

仮数部小数部 0 列判定は上位 16bit に対して行う. 仮数部小数部上位が 0 でない場合は基数 2^8 , 2^8 , 2^9 , 2^{15} の STL 法を適用する. 0 の場合は基数 2^9 , 2^{15} , 2^{15} の STL 法を適

真値と境界の距離	最近接丸め	方向丸め
$2^{\exp - 102}$	998	1040
$2^{\exp - 103}$	876	896
$2^{\exp - 104}$	772	751
$2^{\exp - 105}$	626	630
$2^{\exp - 106}$	496	517
$2^{\exp - 107}$	376	364
$2^{\exp - 108}$	240	244
$2^{\exp - 109}$	137	124
$2^{\exp - 110}$	69	74
$2^{\exp - 111}$	18	20
$2^{\exp - 112}$	7	13
$2^{\exp - 113}$	1	5
$2^{\exp - 114}$	0	3
$2^{\exp - 115}$	0	0

表 3 真値が丸め境界値に近い入力の数

用する。この計算における最大誤差は演算結果に対する相対誤差で、上位列が0列の場合は 2^{-107} 、上位列が0列でない場合は 2^{-102} となる。ただし誤差が大きくなる場合は指数部が-1もしくは0の場合に限られるため、丸め誤り数は大きくならない。また、 2^{16} を基数として行う高基数STL法では対数を式(10)を用いて計算する。

高基数STL法はループ展開したパイプラインで設計する。

テイラー展開は2次で行う。

4.3 評価

実装した結果リソースの使用量は以下ようになった。

使用リソース	使用量	使用割合
LUT	7504	13.54%
LUTRAM	48	0.27%
FF	3207	3.04%
BRAM	13	9.29%
DSP	27	11.25%

表 4 使用リソース量

動作最大周波数は118MHzであった。スループットは14クロックの7段パイプラインで動作する。

4.4 比較

5. その他の対数関数の計算法

前述以外の対数関数の計算法について述べる。

5.1 $\log_2$ 及び $\log_{10}$

$\log$ と同様に計算したのち、乗算による底の変換を行う。また丸め誤り入力底によって異なるため、それぞれ用意する必要がある。

5.2 $\log_2 p1$, $\log_{10} p1$ 及び $\log_{10} p1$

$\log_2 p1$ は入力により場合分けをおこない計算する。 x が十分に小さい場合は多項式による近似で丸めを誤ることなく正確に計算できる。逆に x が十分大きな場合は結果が $\log$ に一致する。どちらでもない場合については正規化を行った後計算する。 x が十分大きな場合は丸め誤りケースも多くが一致するので、同時に実装する場合テーブルを共有することで、丸め誤り修正に必要なリソース量を低減することが可能である。

6. おわりに

本稿では、IEEE754-2008標準で推奨されている正確丸め2進倍精度浮動小数点対数関数のFPGA向き計算法を提案した。提案した計算法では高基数STL法とテイラー展開を用いて計算を行う。また、丸めを誤る場合に対処す

ることによって必要な精度を抑えつつ正確な丸めを保証する。高基数STL法の回数と基数及びテイラー展開の次数の選択により、使用するDSPブロックとRAM使用量のバランスが可能である。

参考文献

- [1] IEEE Standard for Floating-Point Arithmetic, *IEEE Std 754-2008*, pp. 1–70 (online), DOI: 10.1109/IEEESTD.2008.4610935 (2008).
- [2] de Dinechin, F., Lauter, C. and Muller, J.-M.: Fast and correctly rounded logarithms in double-precision, *RAIRO - Theoretical Informatics and Applications*, Vol. 41, No. 1, pp. 85–102 (2007).
- [3] Maire, J. L., Brunie, N., de Dinechin, F. and Muller, J. M.: Computing floating-point logarithms with fixed-point operations, *2016 IEEE 23rd Symposium on Computer Arithmetic (ARITH)*, pp. 156–163 (online), DOI: 10.1109/ARITH.2016.24 (2016).
- [4] Alachiotis, N. and Stamatakis, A.: Efficient floating-point logarithm unit for FPGAs, *2010 IEEE International Symposium on Parallel Distributed Processing, Workshops and Phd Forum (IPDPSW)*, pp. 1–8 (online), DOI: 10.1109/IPDPSW.2010.5470752 (2010).
- [5] Detrey, J. and de Dinechin, F.: A parameterizable floating-point logarithm operator for FPGAs, *Conference Record of the Thirty-Ninth Asilomar Conference on Signals, Systems and Computers, 2005.*, pp. 1186–1190 (online), DOI: 10.1109/ACSSC.2005.1599948 (2005).
- [6] Muller, J.-M. and Muller, J.-M.: *Elementary functions*, Springer (2006).
- [7] Muller, J.-M., Brisebarre, N., de Dinechin, F., Jeanerod, C.-P., Lefevre, V., Melquiond, G., Revol, N., Stehle, D. and Torres, S.: *Handbook of Floating-Point Arithmetic*, Birkhauser Boston, first edition (2010).
- [8] Stehlé, D., Lefevre, V. and Zimmermann, P.: Searching Worst Cases of a One-Variable Function Using Lattice Reduction, *IEEE Trans. Comput.*, Vol. 54, No. 3, pp. 340–346 (online), DOI: 10.1109/TC.2005.55 (2005).
- [9] Hanrot, G., Lefevre, V., Stehlé, D. and Zimmermann, P.: Worst Cases of a Periodic Function for Large Arguments, *18th IEEE Symposium in Computer Arithmetic* (Kornerup, P. and Muller, J.-M., eds.), Montpellier, France, IEEE, pp. 133–140 (online), DOI: 10.1109/ARITH.2007.37 (2007).
- [10] Xilinx 社 : Zynq-7000 All Programmable SoC, <http://japan.xilinx.com/content/xilinx/ja/products/silicon-devices/soc/zynq-7000.html>.