要因別誤差追跡に基づく安全な数値演算論理の自動設計

シグナル・プロセス・ロジック株式会社
瀬尾雄三

要旨

FPGA を用いたパイプライン数値演算論理は、高速処理が可能であるが、多数の演算器を用いるために個々の演算器の最適設計が課題となる．信号を浮動小数点表現して有効桁のみを扱うことで最小ビット幅の数値演算論理を構成する手法は既に報告したが、誤差のキャンセルが発生する場合には無効桁にも有用な情報が含まれるため、単純な無効桁の切り捨てには有用な情報を失う危険があった．誤差のキャンセルは演算過程での誤差の変化を要因別に追跡することで検出することができる．本報では、このための要因別誤差追跡アルゴリズムと、これを用いた数値演算論理の安全な自動設計手法について報告する．

Design automation of safe numeric operation logic based on individual error tracking

Signal Process Logic Inc.
Yuzo Seo

Abstract

The pipeline numeric logic circuit using FPGA can be composed as high through-put. However, since the system uses a lot of calculation units, optimum design of each unit is required. We reported the method to design the numeric operation units of minimum bit width by expressing signal in floating-point format and treating its effective bits. However the method based on simple rounding of error-bits has a risk to lose useful information when the calculating process includes cancellation of errors. In the thesis we treat the issue and report the solution that detects the error cancellation by tracking individual error factors. Adopting the method, the pipeline numeric operation logic circuit can be automatically designed as safe and optimum one.

1. はじめに

数値演算を FPGA で行う場合、演算器を多数用いたパイプライン構成とすることにより、高いスループットが得られる. パイプライン演算装置は CPU 処理とは異なり、個々の演算を専用の演算器で行うため、演算器毎に数値型を最適化することができる. パイプライン演算装置は多数の演算器を使用するために論理規模が大きくなることが短所であるが、各演算器を必要最小限のビット幅で構成することにより、この問題はある程度緩和される．

我々は演算器を必要最小のビット幅で構成

するための機械的手法について検討を行い、有効な桁のみを後段に伝達するよう演算論理を構成することにより、複雑な数値演算過程の各部で扱う数値型を最適化する方法を提案している[1]. しかしながら演算過程で誤差のキャンセルが生じる場合には、無効桁にも有益な情報が含まれるため、この手法をそのまま用いたのでは本来得られる精度の結果を得ることができないという問題があった.

誤差のキャンセルは CPU を用いる一般の科学技術計算でも生じているが、有効桁よりもはるかに広いビット幅を持つ倍精度浮動小数点表現が一般的であるため、誤差のキャンセルは大きな問題とはなっていない. 一方、パイプライン演算処理に際して各演算器のビット幅を有効桁に限定した場合には、誤差のキャンセルが直ちに精度悪化を招く. この問題を回避するため、誤差キャンセルを機械的検出して対処する手法について検討した.

2. 個別誤差要因追跡

2.1. 誤差キャンセルによる問題

誤差のキャンセルが生じるよく知られた例の一つに二次方程式の解の公式がある. たとえば、以下に示す(1)式の二つの解 x_1 および x_2 は(2)～(4)式で与えられるが、b の符号に応じて x_1 もしくは x_2 のいずれか一方の算出に際して b に含まれる誤差がキャンセルされる.

(1) $x^2 - 2\,b\,x + c = 0$
(2) $d = \sqrt{b^2 - c}$
(3) $x_1 = b + d$
(4) $x_2 = b - d$

b に含まれる誤差が c に含まれる誤差に比べて大きな絶対値を持つ場合、(2)式の演算において、b の誤差が d の有効桁を決定する. しかし、b の誤差は(3)式または(4)式でキャンセルされるため、d として有効桁のみを伝達したのでは、c に含まれる有用な情報が失われる.

```
public ref class Real
{
public:
    double r;
    array<double>^ e;
    Real^ operator+(Real^ src);
    Real^ operator– (Real^ src);
    Real^ operator*(Real^ src);
    Real^ operator/(Real^ src);
    Real^ sqrt(Real^ src);
};
```

図 1. class Real

2.2. 要因別の誤差情報

(4)式で b の誤差がキャンセルされることを検出するためには、変数 d に含まれる b 起因の誤差の大きさを知る必要がある. d の誤差要因には、変数 b と変数 c があり、これを識別可能とするためには、変数にはその値と共に要因別の誤差情報を持つ必要がある. そこで、要因毎の誤差の大きさを表す誤差ベクトルを値と共に扱うデータ構造を用いることとする.

個別誤差情報を含む変数のクラス定義を図 1 に示す. クラス Real には、メンバー変数に値 r と誤差ベクトル e が含まれており、値と誤差を同時に演算する演算子や基本関数が準備されている. C++プログラムで変数を定義する際には、double に代えて Real として変数定義を行うだけで、値と誤差を同時に取り扱うことが可能となる.

Real 型の変数 a、b と、誤差のない定数 c に関する各種演算式は以下の通りである.

(5) $r = a + b : \boldsymbol{e}_r = \boldsymbol{e}_a + \boldsymbol{e}_b$
(6) $r = a - b : \boldsymbol{e}_r = \boldsymbol{e}_a - \boldsymbol{e}_b$
(7) $r = c * b : \boldsymbol{e}_r = c\,\boldsymbol{e}_a$
(8) $r = a * b : \boldsymbol{e}_r = b\,\boldsymbol{e}_a + a\,\boldsymbol{e}_b + \boldsymbol{\delta}$
$\boldsymbol{\delta} = (e_{a1}\,e_{b1},\ e_{a2}\,e_{b2}\ , \ldots)$
(9) $r = a / b : \boldsymbol{e}_r = \boldsymbol{e}_a / b - a\,\boldsymbol{e}_b / b^2$
(10) $r = \sqrt{a} \quad : \boldsymbol{e}_r = \boldsymbol{e}_a / (2\sqrt{a})$
(11) $r = a^c \quad : \boldsymbol{e}_r = c\,a^{c-1}\boldsymbol{e}_a$

前報[2]では、これらの式を誤差の高次項を無視して導いたが、ゼロどうしの乗算に際して高次項を無視できないため、(8)式に補正項 $\boldsymbol{\delta}$ を追加している.

2.3. 誤差の定義

本手法で扱う誤差は、主に量子化誤差を意味する．デジタル化された値には、仮数部 LSB の 1/2 に相当する量子化誤差が必然的に含まれる．誤差のキャンセルを検出するため、誤差は符号付の値として扱われる．入力信号の誤差の符号は任意に選べばよく、通常は正とする．

一般に誤差という用語は、扱う値と真値との差を意味し、たとえば AD 変換結果を入力とする場合には、量子化誤差のほかに、AD 変換器自体が持つ各種の誤差やノイズを含む．

しかしながら、これらの誤差は信号処理装置からみれば有用な情報であり、これらの補正が信号処理の目的である場合も多い．そこで、本手法ではこれらを誤差とはみなさない．

2.4. 誤差分散と有効桁

誤差要因が一つであれば、その値自体が誤差となる．複数の誤差要因がゼロでない値を持つ場合は、それぞれの誤差要因の独立性により、これらの自乗和（これを誤差分散と呼ぶ）の平方根が値に含まれる誤差を与える．値に含まれる誤差よりも小さな重みを持つ桁は無効桁であり、要求された数の無効桁のみを後段に伝達するように論理を形成することで、必要最小の論理規模でパイプライン演算器が構成される．

2.5. 浮動小数点表現と有効桁の制御

値を仮数部と指数部を表す二つの整数 M と X とを用いて $M\ 2^X$ で表すことを考える．M に含まれる誤差を 1 未満 1/2 以上とするよう指数部 X を定めれば、無効桁数は 0 となる．

無効桁数を S bit とするには、M に含まれる誤差が 2^S 未満 2^{S-1} 以上となるように指数部 X を定めればよい．これは、誤差が 2^{X+S-1}~2^{X+S} の範囲に収まるように指数部 X を定めればよいことを意味する．実際の計算は、平方根演算の手間を省くため、誤差分散が 4^{X+S-1}~4^{X+S} の範囲となるよう、指数部 X を定める．

2.6. 誤差キャンセルの検出

誤差分散は、演算に伴って増加する場合が多い．しかしながら、演算に供される値が同じ誤差要因を持ち、しかもこの誤差要因が支配的である場合には、演算結果の誤差分散が減少し、演算器入力に桁不足が発生することもありえる．これが誤差キャンセルに伴って生じる問題であり、桁不足の発生を検出することで誤差のキャンセルを機械的に検出することができる．

入力の桁不足の検出は、演算の種類に応じて異なるアルゴリズムで行われる．

浮動小数点数の加減算では、加減算に先立って各項の指数部と演算結果に要求される指数部との差に相当するビット数だけ各項の仮数部をシフトする論理を形成することになるが、仮数部を左シフトさせたのでは、下位の桁に意味を持たないゼロが入力され、新たな誤差を生じることとなる．一般に、演算結果の誤差が減少し、入力の仮数部 LSB よりも下位の桁が出力に影響を与える場合には、誤差キャンセルによる入力仮数部桁数不足を意味する．

2.7. 誤差キャンセルへの対応

入力信号の仮数部桁数が不足した場合は、その入力信号に要求される無効桁数を増加させることで対処する．

(5)~(11)式でも明らかなように、誤差は入力された値にも依存する．誤差キャンセルの検出は、さまざまな入力値の組み合わせに対して、入力側から順次変数の値とこれに含まれる誤差の値を算出することで行われる．誤差のキャンセルが検出された場合には、前段が無効桁を多く出力すればよいが、前段が無効桁を出力するためには更にその入力に無効桁が含まれる必要がある場合も多い．このため、無効桁出力要求を、後段から前段に遡る形で行う．

外部入力の型は前提条件として与えられており、これに対して無効桁を要求することはで

きない．このような変数に対しては、可能な全ての情報が含まれていることを表すフラグ full を立てることで、無効桁要求を拒否するようにする．演算結果である変数に対しても、全ての入力の full フラグが立っており、かつその演算に際して丸めが行われていない場合には、演算結果にこれ以上の無効桁を含めることは不可能であるため、演算結果を格納する変数の full フラグを立ててこれを後段に伝達する．

3. シミュレーション

3.1. アルゴリズム

誤差キャンセルを検出して必要な無効桁を各信号線に設定する処理の概要を図 2 に示す．図の左側には全体のフローを、右側には各部の値を評価する関数 eval のフローを示している．

演算過程は演算器とその入出力間を接続する信号線によって定義され、処理に先立って各演算器に対応する演算器クラスの実体をリストに登録する．また、外部入出力に接続される信号線もそれぞれのリストに登録しておく．

演算器クラスのメンバーには、入力および出力に接続された信号線のリストと、評価関数 eval へのポインタ、および演算論理形成関数 coder へのポインタを含む．また、done フラグは、その演算器に対する処理が完了したことを示す．信号線クラスのメンバーには、それぞれの値と誤差ベクトルの他、仮数部と指数部の属性、要求されている無効桁数、信号源となる演算器、信号の行き先となる演算器のリストが含まれている．

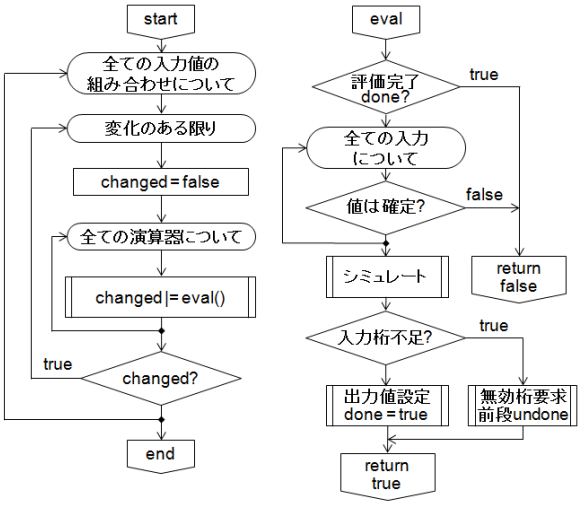

図2. 誤差キャンセル検出のフロー

誤差のキャンセルは特定の入力値の組み合わせにおいて発生するため、全ての入力値の組み合わせに対して演算装置各部の値と誤差をチェックする必要がある．それぞれの入力値の組み合わせに対する処理は、まずフラグ類を初期化し、ついで変化のある限り各演算器に対する評価を繰り返す．

演算器の評価は、それぞれの演算器に対して準備されている関数 eval によって行う．関数 eval は、その演算器の評価が完了しておらず、かつ全ての入力信号線の値が確定している場合に評価を実施する一方、これらの条件が調っていない場合は評価は行わずに false を返す．

演算器の評価を行う際は、値と誤差の各成分を計算し、誤差のキャンセルに伴う入力の桁不足の有無をチェックする．入力の桁に不足がない場合は、出力信号線の値を設定して done フラグを立てる．入力の桁が不足しかつ full でない場合は、これに接続された信号線の無効桁要求を増加し、その信号元である演算器に対して関数 undone を呼び出し、評価完了状態を取り消す．関数 undone は、演算器が done である場合にこれを false とし、その演算器出力に接続された全ての信号線の行き先となっている演算器に対しても undone を行う．

3.2. 演算過程の一例

図 3 に二次方程式一つの解を演算する過程での値と誤差の推移を示す．図中、mul は乗算器、sub は減算器、sqrt は平方根演算器であり、入力 b、c の各組み合わせに対して(6)、(8)、(10)式により値 r_k と誤差ベクトル $\boldsymbol{e}_k$ を算出する．

b が正の場合、減算器 sub2 で誤差のキャンセルが発生する．とくに $X_b \geqq X_c$ の場合には、

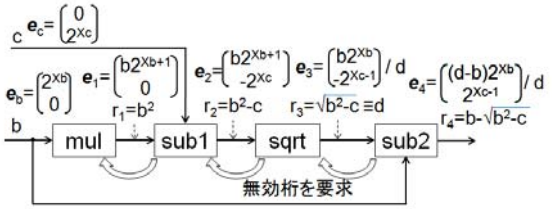

図3.　二次方程式の解を求めるフロー

出力の誤差分散 e_4^2 を管理範囲に収めるために入力に無効桁が必要となる．この場合には、前段に必要な数の無効桁を追加して出力するよう要求を送る．前段が無効桁を出力するためには前段の入力にも無効桁が必要であり、データフローを遡る形で各演算器出力に必要な無効桁を順次設定することとなる．

誤差のキャンセルはアルゴリズムを工夫することで避けられる場合も多い．上の例では、b が正の場合は式 $x=c / (b+\sqrt{b^2-c})$ を用いることで高精度解が得られることが知られている．しかしながら、アルゴリズム最適化の手法は、一般の計算問題において必ずしも自明ではない．そこで、誤差のキャンセルが検出された際に警告メッセージを出力して、論理設計者によるアルゴリズムの見直しを促すようにする．

4.　数値演算論理形成への応用

4.1.　数値演算論理の形成

演算器リストから数値演算論理を形成する手法は前報[1]と同様であるが、各信号線の型は、誤差のキャンセル検出の時点で得られているので、これを利用すればよい．

コードの形成に際しては、数値演算論理の動作時点で誤差情報を利用する場合には、誤差を計算する論理も合わせて実装する．演算論理の動作時点で誤差情報が不要である場合は、誤差の演算を行う論理を実装することなく、仮数部の幅を必要な最大幅に設定することで誤差キャンセルによる精度悪化を回避する．

4.2.　誤差演算論理の実装

各部の誤差の大きさは入力された値により異なるため、演算論理の動作時点で誤差情報を利用するためには、数値演算論理で扱う信号にも誤差情報を含め、値と誤差を同時に演算する論理を形成する必要がある．

このために使用される数値型を図 4 に示す．この型は、数値の仮数部 M と指数部 X を表す二つの整数値に加え、異常状態を表す三つのフラグ O、U、N および n 個の誤差要因に対応した誤差の混入量を表す E_1~E_n からなる．それぞれの要素に対応する信号線は必要な場合にのみ論理回路として実装される．無効桁数が S bit に管理されるため、誤差の大きさは仮数部 LSB の 2^S 倍以下の符号付の値とに限定され、仮数部は小数部 F bit を加えて 1 + S + F bit で表すことができる．また、誤差の指数部は数値の指数部 X が用いられる．

誤差の演算のため、数値を演算する論理に加えて、(5)~(11)式により誤差を演算するための論理を実装する．

誤差は小さな桁数の仮数部のみが実装され、これに対するシフト演算は値と同じシフト数となることなど、誤差を演算する個々の論理は値の演算論理に比較して小規模となる．しかしそれでも演算論理が複雑になることは避けられないことから、誤差を演算する論理は実行中の誤差情報が必要な場合にのみ実装する．具体的な応用分野としては、高信頼性が要求される分野、精度に係わる情報が必要な科学技術演算などが考えられる．

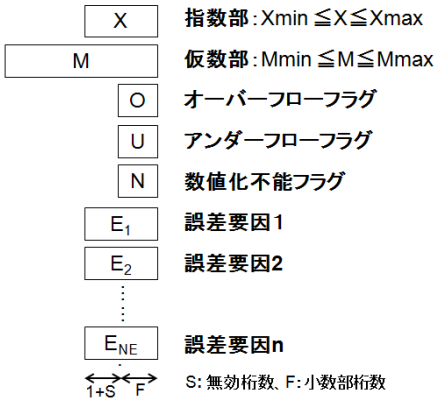

図4.誤差情報を含む浮動小数点表現

4.3. 最大伝送モード

演算実行中に誤差情報を利用しないのであれば、誤差を演算する論理の実装は省略することができる. この場合は、誤差のキャンセルに配慮した仮数部ビット幅で回路を構成し、演算結果の仮数部がこの幅を越す場合にのみ仮数部下位をシフトアウトする論理を形成する.

5. まとめと今後の方向

有効桁に着目した最小規模の数値演算論理形成手法に関しては既に報告済みであるが[1]、その際、二つの課題が残されていた. 第一の課題は演算仕様を記述するための言語の必要性であり、これに関しては既に報告済みである[3]. 本報では第二の課題である誤差キャンセルの問題に対して解決策を提案した. これにより有効桁に着目した最小規模の数値演算論理の自動形成手法の問題点は解消されたこととなる. 今後は、本手法に基づく FPGA 向けの数値演算論理設計支援ツールの実用化に注力したい.

パイプライン演算器は CPU に比較してきわめて高いスループットが得られることから、計測・制御・画像処理などの高速性が要求される分野に向いている. 本手法はこれら分野で用いられる数値演算論理設計の手助けとなろう. 従来は CPU が主に利用されているシミュレーションなどの科学技術計算の分野でも、FPGA を用いたパイプライン演算装置を利用する可能性もあり、この分野でも本手法は有益ではなかろうかと期待している.

科学技術計算の多くは、十進 6 桁程度の有効桁を持つ数値を扱っている. 初期の数値演算では同程度の分解能をもつ単精度浮動小数点数が幅広く利用されていたが、単精度浮動小数点数を用いた数値計算では桁落ちに伴う問題が無視できない頻度で発生したため、今日の科学技術計算には十進 15 桁の分解能をもつ倍精度浮動小数点数が多用されている.

今日行われている科学技術計算の多くは分解能 6 桁で十分であるにもかかわらず、大量の計算が倍精度浮動小数点数を用いて行われており、計算機資源や消費電力の面で多くの無駄が発生しているものと思われる.

有効桁数が 6 桁の計算に 15 桁の分解能を必要とする一つの理由は誤差キャンセルによると考えられている[4]. 本報で提案した手法を応用すれば、誤差キャンセルを検出して、これを解消するようアルゴリズムを改良し、あるいは必要な部分でのみ高い分解能での演算を行うようにすることが可能と考えられ、CPU を用いる演算分野でも多くの改善が期待される.

とはいえ、この手法が真価を発揮するのは個々に専用の演算器を用いるパイプライン方式の数値演算装置であり、特にビット幅を 1 bit 単位で設定することができる FPGA を用いた数値演算装置であろう. 本技術の研究に際しては、当面は組み込み分野での応用にフォーカスして開発を進めるが、数値計算全般にパイプライン演算器を応用する可能性も念頭に検討を進めたい.

引用文献

[1] 瀬尾雄三: 浮動小数点処理を含む論理設計支援システム, 情報処理学会シンポジウムシリーズ, Vol.2010, No.7, pp.3-8, 2010

[2] 瀬尾雄三: 誤差情報を含む浮動小数点表現とこれを用いた数値演算論理, 第76回情報処理学会全国大会講演論文集(1), pp.5-6. 20141

[3] 瀬尾雄三 : パイプライン処理のための演算仕様記述言語mhdlとその処理系, 情報処理学会シンポジウムシリーズ, Vol. 2012, No. 5, pp.115-120, 2012

[4] 佐々木建昭, 加古富志雄 : 悪条件性を推定する浮動小数グレブナー基底の計算法, 数理解析研究所講究録, Vol. 1652, pp. 33-43, 2009