システムＬＳＩ設計技術　108－13
(2003. 1. 29)

SD 数演算を用いた２進浮動小数点加算回路の構成

細川　純一†　　魏　書剛†

† 群馬大学工学部 情報工学科
〒 376-3515 群馬県桐生市天神町 1-5-1
E-mail: †{hosokawa,wei}@ja4.cs.gunma-u.ac.jp

あらまし　従来の２進数演算に基づく浮動小数点演算システムでは、２の補数表現を用いて加減算を行っている。加減算の演算数の仮数部は絶対値と符号からなるため、演算数の順番の入れ換えや加算における桁上げ伝搬は演算の速度を制限してしまう。本論文では、２進 SD(Signed-Digit) 数演算を導入することにより、従来の方法における演算数の入れ換え及び桁上げによる演算制限を軽減させ、高速な２進浮動小数点加算回路を実現するのを目的とする。提案する浮動小数点加算回路の内部は SD 数加算回路を中心とした構成になるが、外部の入出力数表現は従来の２進小数点数表現を用いる。そのために、効率的な丸め及び SD 数 - ２進数変換回路を提案する。VHDL による回路設計及びシミュレーションを行い、従来の浮動小数点加算回路と比較することにより回路性能を考察する。
キーワード　SD 数，浮動小数点数表現，加算回路，端数処理，VHDL

Architecture of Binary Floating-point Adder Using Signed-Digit Number Arithmetic

Junichi HOSOKAWA† and Shugang WEI†

† Department of Computer Science, Gunma University
Tenjin1-5-1,Kiryu-shi,376-3515 Japan
E-mail: †{hosokawa,wei}@ja4.cs.gunma-u.ac.jp

Abstract　In a floating-point number arithmetic system based on the conventional binary number arithmetic, two's complement number representation is used to perform addition/subtraction in the floating-point addition circuit. Since the significands for addition/subtraction are expressed in a signed-magnitude number representation, the swapping operation of the two operands is required. Moreover, the carry propagation in the addition will also limit the arithmetic speed. In this paper, we introduce a radix-two signed-digit(SD) number arithmetic to the floating-point number arithmetic system. Then the swapping operation is not required and the carry propagation becomes free for the inner addition. We present an addition circuit architecture using the SD arithmetic with the input/output data in a normal binary floating-point number representation. Efficient SD number rounding and SD-to-binary conversion circuits are also proposed.
Key words　Signed-Digit Number , floating-point number representation , addition circuit , rounding , VHDL

1.　はじめに

現在数多くのコンピュータやディジタル信号処理プロセッサのような数値演算システムは浮動小数点数表現を用いている。浮動小数点数の演算回路は加減乗除の四則演算回路を基本としているが、そのうちの加算はもっとも重要であり、演算の大部分を占める。

従来の２進数演算に基づく浮動小数点演算システムでは、その入出力では絶対値符号の数表現により仮数部が表されているが、内部では２の補数表現を用いて加減算を行っている [1] [2] [3]。この時に発生する桁上げ伝搬や演算数の順番の入れ換えは回路に長い遅延時間を発生させ、演算の速度を制限してしまうことになる [4]。

一方、２進数の符号化技法を用いて高速な演算回路アルゴリズムを研究するという関心は高く、SD(Signed Digit) 数系もその中の１つである [5]。SD 数系では、演算の各桁は下位桁の中間結果により加減算を行うことで各桁を並列に計算し、桁上げによる遅延時間を発生させなくなるという性質を持ってお

り、様々なVLSI演算が提案されている[6][7]。しかし、SD数演算を用いた浮動小数点数演算回路は報告されていない。従来の浮動小数点加算回路で、演算数の順番の入れ換えは一般にswapperと呼ばれ、指数部の大小によって仮数の演算数順番を決定するが、同じ指数をもつ場合、仮数部の比較を行うため、演算の遅延に主な原因の1つである。冗長な数表現であるSD数演算の導入は、仮数部の演算数順番を決定する必要がなくなるのでswapperそのものを省くことができる。

また、浮動小数点加減算回路内において仮数部の加減算や演算の順番入れ換えのみならず、桁揃えのためのビット・シフト、入出力の正規化、端数処理（丸め）等の処理も行う必要があり、様々な箇所で遅延時間発生の原因が存在する。これには、バレル・シフタやシフト量を予測するアルゴリズム等を用いた高速なシフト動作や、効率の良い様々な端数処理方法なども提案されている[10]。著者らはSD数を入出力データとした浮動小数点加算回路を提案した[8]。しかしながら、数値計算の応用には従来の2進数に基づくデータフォーマット規格が使用されている。本稿では、2進数データを外部入出力とし、内部でSD数演算を行う浮動小数点加算回路の構成を提案する。

SD数演算の導入は、2進数の演算数の順番入れ替えや加算による桁上げ伝搬がなくなるメリットがある反面、冗長数表現を持つため、内部で2進数-SD数及びSD数-2進数の変換回路を用意する必要がある。多ビット化によって従来の2進数でのシフトや丸め込みが適用できないので、特にSD数-2進数の変換回路により、遅延時間が増大する可能性もある。そのために、効率的な丸め及びSD数-2進数変換回路を提案する。VHDLによる回路設計及びシミュレーションを行い、従来の浮動小数点加算回路と比較することにより回路性能を考察する。

2. 2進浮動小数点数加減算

2.1 浮動小数点数表現

実数の表現方法には、固定小数点数表現, 浮動小数点数表現, 対数表現などがある。その中で浮動小数点数表現は4つの要素による以下の式で表される[2][3]。

$$x = (-1)^n s \times b^e$$

$(-1)^n$: sign (符号)　　s : significand (仮数)
b : exponent base (基数)　　e : exponent (指数)

データ配列は、先頭より 符号・指数・仮数各部分に指定ビット長に分割して解釈する。基数には、多くの場合2あるいは2のべき乗数が用いられる。浮動小数点数表現は、固定小数点数表現を使用した場合と比べ同じデータのビット長で非常に広い範囲の値を扱う長所が挙げられる。

図1はIEEE標準に定められた表現規格を示し、公式には“ANSI/IEEE std 754-1985”という名で知られている[1]。

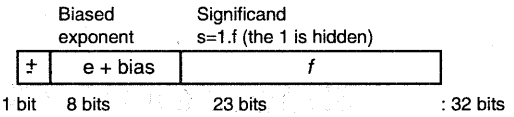

図1　IEEE標準規格に定められた浮動小数点表現

配列データ長は2進法で32ビットであり、うち指数部長は8ビット、仮数部長は23ビットである。IEEE754規格では、指数eには2^8-1の”bias”(偏り)が存在する。biasとは、指数部がeビット長の時、指数の値を2^{e-1}分だけ加えた値として解釈するものである。また、仮数sの範囲は[1,2)(1以上2未満)と定められている。この時sの先頭ビットは常に「1」(整数部の1)となるため普段は省略され、実際の仮数部ビット列には残りの小数部分の値のみが格納されている。

2.2 従来の浮動小数点の加減算回路の構成

2進の浮動小数点演算回路を設計するに当たり、基本となる加減算式は以下の通りである:

$$(\pm s_1 \times b^{e_1}) \pm (\pm s_2 \times b^{e_2}) = (\pm s_1 \pm \frac{\pm s_2}{b^{e_1-e_2}}) \times b^{e_1}$$
$$= \pm s \times b^e \qquad (e_1 > e_2)$$

この中で、中間和$(\pm s_1 \pm \frac{\pm s_2}{b^{e_1-e_2}}) \times b^{e_1}$について$\times b^{e_1}$を加算結果の指数部、残りを仮数部として値を返す。この加減算式より回路を設計する場合、数式の複雑さを比較して分かるように、回路決定の大部分を占めるのは仮数部分ということになる。よって、仮数部分の高速化を実現するということは、回路全体の遅延時間を大幅に改良できることである。

次に示す図は2進浮動小数加減算器のブロック構成である[4],[11]。

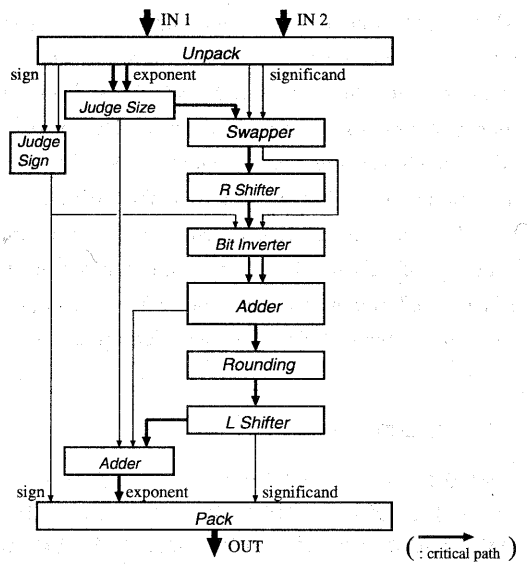

図2　2進浮動小数加減算回路

演算のアルゴリズムは、まず指数部の大小判定により、2入力データの大小を定める。これは、次の仮数部のSwapperで

行い、結果により2入力の入れ換えが必要である。入れ換えを行った後、小さい指数値を持つオペランドの仮数はAlignment Shifterで2入力データの指数差分に応じて仮数部を右にシフトする。データの値が負であった場合や減算処理の場合はBit inverterで符号反転により2の補数で表現される。仮数の加算はAdderで行われる。特に仮数は多ビットの加算となるため、2進数加算で特有の"桁上げ伝搬"の発生により、遅延時間が長くなるという問題が従来の浮動小数点演算回路では起こっていた。Adder以降は、Roundingでの丸め込み処理を行う。出力を正規化(IEEE754等)させる場合は、更にAlignment Shifter等を用いて仮数の値を規格に合うよう揃える必要がある。

指数部は、2入力のうち大きい値を基にするが、丸め込みによる桁上げや正規化による仮数のシフトによりその値は増減する。よってこの加算動作もAdderであるが、指数部は通常、仮数部と比べビット数は少なく、構成される指数部のAdderは仮数部よりも小さいものになる。

3. SD数演算に基づく浮動小数点数加減算

3.1 SD数演算

2進SD数は冗長数系であり、各桁を$1, 0, \bar{1}(\bar{1}=-1)$の3値で表現し、

$$x=\sum_{i=0}^{n-1} x_i 2^i \quad , \quad x_i \in \{\bar{1}, 0, 1\} \tag{1}$$

と表す。例えば$5=(0101)_{(2)}=(10\bar{1}\bar{1})_{SD}$などの、同値表現が複数存在することがある。

SD数の加算$s=x+y$は、まず各桁で並列に中間和u_iと桁上げc_iを求める。

$$u_i+2c_i = x_i+y_i \tag{2}$$

そして

$$s_i = u_i+c_{i-1} \tag{3}$$

で最終和を求める。式(2)で中間和u_iと桁上げc_iは、表1の演算規則に従い、入力桁の値(x_i, y_i)と1桁下位の値(x_{i-1}, y_{i-1})を参照し、式(3)に於けるu_iとc_{i-1}が同符号を持たないような値を生成する[6]。

表1 SD数系の演算表

x_i, y_i	00	01	0$\bar{1}$	0$\bar{1}$	0$\bar{1}$	11	$\bar{1}\bar{1}$
x_{i-1}, y_{i-1}	−	neither is $\bar{1}$	at least one is $\bar{1}$	neither is 1	at least one is 1	−	−
c_i	0	1	0	$\bar{1}$	0	1	$\bar{1}$
u_i	0	$\bar{1}$	1	1	$\bar{1}$	0	0

演算の桁上げ伝搬を無くすことによって、全ての桁を独立して並列に計算することが可能である。

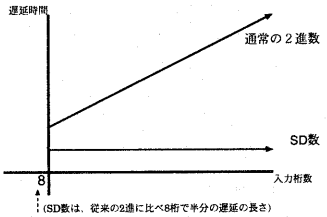

図3 実行時間の比較

従来の2進数の加算では、各桁がそれぞれ下位ビットからの桁上げを順次待つため、桁数に比例して遅延時間が長くなるのが避けられなかった。SD数系では、上記の方法により各桁が並列での処理が可能となるので、全ての桁の加算を(中間和の生成)→(最終和の生成)2回の動作で同時に終わらせることが可能となる。

入出力の桁数と、それに伴って発生する回路の遅延時間のグラフは次の図3の通りである。

SD数系での設計による加算回路の実行時間は入出力の桁数に依存しないので、桁数が増えれば増えるほど従来の2進数と比べ遅延時間の優位性が顕著になる。

SD数導入による演算のアルゴリズムは、まず指数部の大小を行う。小さい指数値を持つオペランドの仮数はAlignment Shifterで2つの入力データの指数差分に応じて仮数部を右にシフトする。データの値が負であった場合や減算処理の場合はBit inverterで符号反転により1を$\bar{1}$に変換する。加算動作にはSD数で設計された回路SD Adderを用い、ここで従来発生する桁上げ伝搬を無くし遅延時間を短く抑える。Adder以降は、Roundingでの丸め込み処理や出力の正規化を行う必要があること、および指数部の演算は従来の2進数での演算と同様である。

以上より、回路の高速化のため仮数部にSD数系を導入することを提案する。しかし、SD数系はその冗長性により2進数へ変換するとき新たな桁上げ問題を引き起こすため、高速な丸め込みや正規化処理の方法が必要である。

3.2 SD数の2値符号化

SD数は冗長数系であり、$1, 0, \bar{1}$の3値表現であるが、これは2値を扱うディジタル回路上では1桁を1ビットで表せないことになる。よって、SD数で演算を行うときは1桁を2進2ビットで表す方法をとる必要があり、これを2値符号化という。

各桁の値x_iに対し、2値符号化させた2ビットの値$x_i(1), x_i(0)$は表2通りとなる[9]。

2値符号化により、回路内部で2進p桁のSD数の仮数は$2p$ビットの2値符号列で表すこととなる。入力をANSI/IEEE標準規格で考えると、SD数変換後は図4のような指数8ビット、仮数46ビットの計54ビットとなる("hidden 1"を考慮

表2　SD 数の2値符号化

x_i	$x_i(1)$	$x_i(0)$
1	0	1
0	0	0
$\bar{1}$	1	0

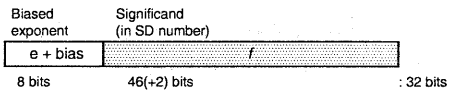

図4　SD 数による浮動小数点表現

すると、2値符号化により更に +2 ビット）。

2値符号列は、その中に sign(符号) 情報も含むため、浮動小数点演算中の符号は仮数が受け持つ。また、このことは演算順番の入れ換えを行う必要がない (Adder 結果が負になっても構わない) ので swapper ブロックが不要になる利点も生まれる。

3.3　SD 数 - 2進数の変換

本研究では、以前はデータの入出力も SD 数を想定して回路を設計していた [8]。しかし、これは一般的な既存の浮動小数点演算回路の応用に対し扱いづらいものとなる。よって入出力は従来の2進数を用い、途中で仮数を SD 数表現へ変換し、その後で SD 数による加算を行い、再び2進数に戻して出力する、という方法をとる。

2進数 -SD 数の変換回路は単純に表1のように各桁を2値符号化により展開し、その回路を SD Coder と言う。2進数表現は SD 数の各桁が同じ符号を持つ特別な数表現と扱われるため、その変換は単純な回路で実現できる。これに対し、SD 数を2進数に変換するときは、桁上げが従来の加算と同じように生じる。

SD 数 - 2進数への変換 (変換回路を SD Decoder という。) は2値符号化された SD 数を、通常の2進に戻す動作を行う。変換手順は、まず、Adder 結果では冗長表現のため 1と$\bar{1}$ が混在している。これを、ビット全体を 1か$\bar{1}$ のどちらか一意に揃える。揃えた値より符号を判断し、2値符号化の逆により通常の2進数に戻す。

例1: SD Decoder の動作

SD 数表現である

$$-5 = (0, 0, \bar{1}, 0, 1, 1)_{(SD)} \qquad (4)$$

は、2値符号で表記すると

$-5 = (00, 00, 10, 00, 01, 01)$

である。冗長性を無くすため、すべての桁を 10 あるいは 00 にする。すなわち、

$-5 = (00, 00, 00, 10, 00, 10)$

となる。結果は

$$5 = (0, 0, 0, 1, 0, 1)_{(2)}, \quad sign = 1 \qquad (5)$$

となり、通常の2進数として出力する。上記の変換には2進数加算と同様に桁上げ伝搬が生じる。

3.4　正規化とシフトの高速化

2進数に戻した仮数は、その出力を IEEE754 に合わせるため正規化させる必要がある。仮数の数表現範囲は [1..2) に制約され、このために仮数に左へ多ビットシフトを行うこととなる。上の例1の場合、例えば先頭桁が 2^0 とすると、左へ3ビットのシフトを行うが、SD 数は冗長性があるため'0' の個数は一意には定まらない。このためには先頭からのシフト量を Lead Zero Circuit 等を用いてカウントし、決定する回路 shiftamount を用意する必要がある。

例2:

$$3 = (0, 0, 1, \bar{1}, \bar{1}, 1)_{(SD)}, exp = 0 \qquad (6)$$

のような加算結果を正規化するため、先頭桁が '0' 以外となるまで左へ多ビットのシフトを行う。通常このような場合、上位から'0' の個数をカウントし (Lead Zero Circuit)、その分だけを左へシフトする。

$12 = (1, \bar{1}, \bar{1}, 1, 0, 0), exp = -2$

しかし、2値符号を用いて2進数に変換した場合、

$12 = (01, 10, 10, 01, 00, 00)$

$= (00, 00, 01, 01, 00, 00)$

$= (0, 0, 1, 1, 0, 0)_{(2)}$

となり、更に左への多ビットシフトが必要となってしまう。

よって、シフタは Decoder の後にまとめて行うことになる。しかし、設計する回路の中で、Decoder や Shiftamount は、比較的規模の大きなブロックであるため、このままの設計では次の図のような構成となり、遅延時間が増大しってしまう。

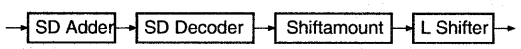

図5　Adder 後のブロック

ここで、先の例では、結局必要なシフト量は『先頭の'0' ビット列』+『数値の最大桁の符号とは逆の値が2桁目以降続く場合、そのビット列』である。このことは、シフト量は Decoder で2進数に変換する前に、Adder の結果より求められる。このことに着目すると、実際のシフト動作の前にこれらシフト量を求める回路 Shiftamount を用い、Decoder と並列動作させ先読みさせる。

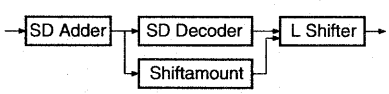

図6　Decoder,Shiftamount の並列化

Shiftamount と Decoder の設計は SD 数導入のデメリットではあるが、並列処理により遅延時間を短く抑えることは可能と言える。

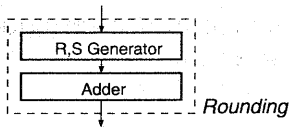

図7 Rounding の内部処理

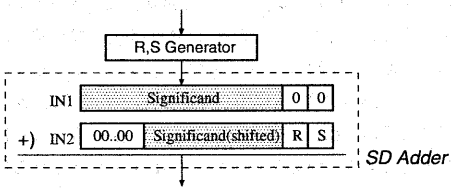

図8 R,S の Adder への追加

3.5 端数処理の高速化

コンピュータ上での数値計算では、出力の正規化や、丸め込み結果が十分正確な精度長をとるために、ulp(最小位桁) 以下右に拡張ビットを保持する考え方があり、これをガードビットと呼ぶ。通常、2進数での丸め込み処理の場合、

- R (Round bit): ulp 未満第1桁
- S (Sticky bit): 第2桁以降、全ビットを OR でとる

が必要となる。Rが切り上げ／捨ての判定を行い、Sは誤差の有無の存在を示す (rounding-to-nearest-"even"scheme 等別の丸め込み方法等を使う場合この限りではない)。

ここで、Rビットの切り上げ動作は、加算のため桁上げ伝搬が発生する、これは僅か'1' の加算ではあるが、無視できない遅延時間を持つ回路を生成する。よって、Rounding ブロックは、詳細には図7のような『R,S 生成』『切り上げのための加算』の2つの処理によって構成されている。

SD 数系の場合に於いても、通常この桁上げ伝搬は発生する上、更に2値符号化の問題もあり、実際の回路上では桁上げが発生したので単純に結果に +'1'、とはいかない問題もある。

しかし、加算規則により SD 数系は冗長性がありかつ各桁は並列に演算される、この長所を用いて桁上げ伝搬の起こらない端数処理を考察すると、次の場合が考えられる
(S_0: 加算結果の最小位桁 (ulp))。:

- 端数が正方向切上 $\rightarrow S_0 \neq' 1'$
- 端数が負方向切上 $\rightarrow S_0 \neq' \bar{1}'$

この条件が保証されている限り、桁上げは ulp でストップし以降の伝搬は起こらない。

ulp 桁の加減算では、表2の x_{i-1}, y_{i-1} の参照は (存在しないため) デフォルトで双方共に'0' となる。この値は変えても結果に影響は無いので、上記の条件に沿うように参照の値を変更する。ただし、Adder で既に丸め込み結果を知る必要があるので、予め R,S Generator を用いてガードビットを生成しておく (図8)。この結果を x_{i-1}, x_{i-2} に代入すれば ($y_{i-1}, y_{i-2} =''00''$)、結局、仮数の加算で通常は存在しない ('0' である) c_{-1} を丸め込みの結果がその役割を果たすため、これらの動作をまとめることにより SD 数での加減算ではやはり桁上げ伝搬を起こさない。

例3: 2入力の加算 A+B を考える。

$$\begin{cases} A = -2^3 \times 1.10000001 \\ B = 2^1 \times 1.10000111 \end{cases}$$

SD 数での加算のための前処理として、シフトとビット反転、ガードビット生成までの処理を行った仮数部の値は次の通りとなる。

Sig A	$\bar{1}.\bar{1}0000000\bar{1}$	
Sig B aligned	0.01100001	R:1,S:1

右シフトを行うのは小さい方のオペランドのみなので、R,S は Adder の前に求められる。通常、丸め込みは Adder の後で処理するが、生成されたガードビットをここで加算の下2桁に含める。

Sig A	$\bar{1}.\bar{1}0000000\bar{1}$	00
Sig B aligned	0.01100001	11
c_i	$\bar{1}0.11000001$	$1\bar{1}$
u_i	$0.\bar{1}0101100$	$\bar{1}1$
Sig A+B	$\bar{1}1.00\bar{1}00001$	$0\bar{1}$

この演算で出た仮数の Adder 結果 Sig A+B は、SD 数の加算規則と R,S の生成規則により切り上げでも ulp を越えて桁上げは起こらない。このため、Rounding 中の Adder 部分が生成されないためその分の回路面積・遅延時間ともに少なくなる。

SD 数系の特徴を生かした桁上げ伝搬の発生しない端数処理は、通常の2進数での演算と比べて大きなメリットとなるだろう。

4. VHDL による演算回路の設計と回路評価

図9は SD 数を導入した場合の2進浮動小数加減算回路のブロック構成である。

これまでより、まず演算の順番入れ換える必要がないので Swapper ブロックが不要となった。Adder 本体は SD 数による加算とし、また、Rounding は桁上げが起こらない (桁上げによる加算が発生しない) ので、Adder 前のガードビット生成 R,S Generator のみとなった。一方、2進数-SD 数変換のため SD Coder と Decoder を追加した。また、シフト量決定回路 Shiftamount は 3.5 節により Decoder と並列な動作とした。

表3は、浮動小数点演算回路を SD 数を用いた場合と従来の2進数での場合双方で回路設計を行い、回路面積・遅延時間を比較した結果である。回路設計は VHDL を使用し [12]、$1\mu m$ CMOS のゲートアレーにより評価を行った。

SD 数の導入では、Swapper が不要なこと、Adder , Rounding で桁上げ伝搬が起こらないことがメリットであ

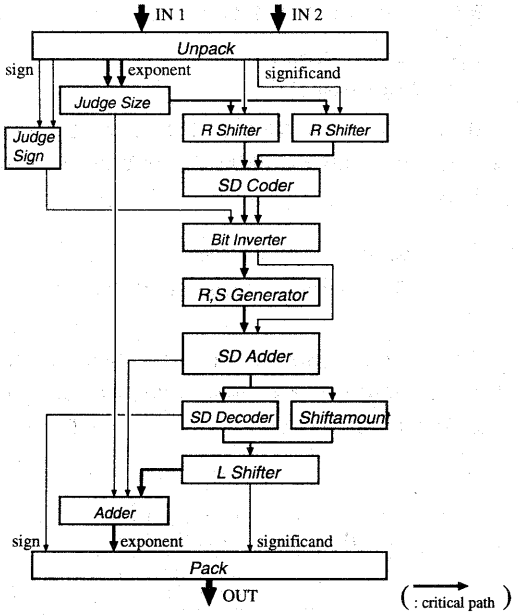

図9　2進SD数浮動小数加減算回路

表3　回路最適化後の結果

回路評価	回路面積		遅延時間 [ns]	
主なブロック	2進	2進SD	2進	2進SD
Swapper	118	–	18.93	–
Adder	228	534	26.90	5.93
Rounding(RS Gene)	244	244	8.34	8.34
Rounding(Adder)	65	–	9.75	–
SD Coder	–	24	–	0.79
SD Decoder	–	208	–	26.91
Shiftamount	–	172	–	(24.39)
Total	945	1449	92.90	71.79

り、これらのブロックでは遅延時間の短縮となった。 Adderでは、各桁の演算が2回のステップを持つことにより回路面積は大きくなっている。

Coder,Decoder,Shiftamount 各ブロックに相当する回路を新たに追加しなければならないのはデメリットだが、前述の並列動作によって、 Decoder と Shiftamount のどちらか片方（遅延時間の長い方）が Critical Path となるため、遅延時間の増加は少なく抑えている。

実際のディジタル回路における演算回路の運用では、加算処理は何千、何万回と繰り返し行われることが多い。 SD 数導入の利点は、 Adder 本体の高速化が主であるため、これは加算を繰り返す度累積される。一方、外部の入出力を従来の2進数と想定した場合においても、加算処理の繰り返しは内部で行えば良いため、2進-SD 2進の変換回路は繰り返しの最初と最後ただ1回ずつ用いれば済む。

以上を考慮すれば、 SD 数を導入した加算回路は従来の2進数の設計と比してより高速になり、回路が大規模になる程有効であることが明らかにされている。

5. まとめ

本稿では、 SD 数演算を従来の2進数浮動小数点加算回路に導入することを提案した。提案の浮動小数点加算回路では、従来の2進数浮動小数点回路の2進数入出力データフォーマットを持ち、内部の加算が SD 数加算器で行われる。したがって、演算数順番の入れ替えが不用となり、2進数加算による桁上げ伝搬を無くすことができる。また、 SD 数を導入した際に起こる SD 数ー2進数の変換など幾つかの問題点とその解決方法について議論した。 VHDL による回路設計および性能評価を行い、従来の方法に比べ高速な浮動小数点加算回路が得られることが明らかにした。

文　献

[1] ANSI/IEEE Standard 754-1985 for Binary Floating-point Arithmetic, IEEE, 1985.

[2] Israel Koren, "COMPUTER ARITHMETIC ALGORITHMS", Prentice-Hall International Editions, 1993.

[3] Behrrooz Parhami, "COMPUTER ARITHMETIC : algorithms and hardware designes", Oxford University Press, 1999.

[4] H.Suzuki,Y.Nakase,H.Makino,H.Morinaka,K.MAshiko and T.sumi, "Leading-zero Anticipatory Logic for High-speed Floating Point Addition", 信学技報 Technical Report of IEICE DSP95-98 ICD95-147, 1995.

[5] A.avizienis, "Signed-digit number representations for fast parallel arithmetic", IRE Trans.Elect.Comput.,EC-10,pp.389-400, 1961.

[6] N.Takagi, H.Yasuura and S.Yajima, "High-speed VLSI multiplication algorithm with a redundant binary addition tree," IEEE Trans.Comput., vol.C-34, no.9, pp.789-796, Sept.1985.

[7] S.Wei and K.Shimizu, "A Novel Residue Arithmetic Hardware Algorithm Using a Signed-Digit Number Representation" IEICE TRANS.INF.&SYST.,Vol.E83-D,No.12,pp.2056-2064, 2000.

[8] 細川 純一、魏 書剛, ＳＤ数演算を用いた浮動小数点算術演算回路, 電子情報通信学会 2002 年総合大会講演論文集, C-12-25, 2002.

[9] H. Makino, Y. Nakase, H. Suzuki, H. morinaka, H. Shinohara and K. Mashiko, "An 8.8-ns 54 × 54-bit multiplier with high-speed redundant binary architecture," IEEE J.Solid-State Circuits, vol.31, pp.773-782,June 1996.

[10] K.C.Chang "Digital Systems Design with VHDL and Synthesis", IEEE Computer Society, pp.132-137, 1999.

[11] Woo-Chan Park, "Design of the Floating-point Adder Supporting the Format Conversion and the Rounding Operations with Simultaneous Rounding Scheme", IEICE Trans.Inf.&Syst.,Vol.E85-D,No.8, 2002.

[12] J.Bhasker, "A VHDL Primer", CQ 出版社, 1995.