3重指数分割に基づく浮動小数点数演算のための指数と仮数の高速分離結合回路の設計と評価

大山光男†

数値演算におけるオーバフローやアンダフローの発生を避けることのできる，可変長指数部を持つ浮動小数点表現が提案されている．3重指数分割に基づく浮動小数点表現は，2重指数分割に基づく浮動小数点表現URRの優れた特徴を継承，指数の桁数が大きい領域ではURRより指数部が短く，表現精度が改善される．しかし，指数部の構成が複雑となるのでURRに比べて演算時間とハードウェア量の増加が予想され，実用化するには高速演算機構の開発が重要となる．そこで本論文では，3重指数分割に基づく浮動小数点数のハードウェアによる高速演算実現におけるキー技術である，指数と仮数の高速分離結合回路を設計，URRとの比較で評価する．まず，処理を簡単化するため，分離後の指数の表現を分離前のデータに等しい長さの2進整数表現とし，指数の範囲をURRと同程度に制限する．そして，データを構成する各部の位置と長さを高速に検出，確定し，組合せ論理回路でビット並列に高速処理を行う．さらに，データ長64ビット，ゲート長0.8 μm CMOSゲートアレイへの実装を想定して処理時間とハードウェア量を見積もり，定量評価を行った．その結果，指数の範囲を上記に制限することにより，指数と仮数の分離時間はURRに比べ41%増加するが，分離のハードウェア量，結合に要する時間，ハードウェア量は13%増以下に抑えられる見込みが得られた．

A Design and Performance Estimation of Circuits for Fast Separating an Exponent and a Fraction from and Combining to a Floating-Point Arithmetic Number Based on Triple Exponential Cut

MITSUO OOYAMA†

New floating-point arithmetic systems those have a variable length exponent part have been proposed and they can overcome overflow and underflow problem on computation. The floating-point arithmetic based on triple exponential cut can gain more bits in the fraction part compared to URR floating-point arithmetic based on double exponential cut because its exponential part is shorter over the range where absolute value of the exponent is large. And it still holds desirable characteristics those of URR. But it requires longer execution time and more hardware cost to calculate because its exponent part is more complicated. So, we must develop a fast calculation technique to use the new floating-point arithmetic in real systems. In this paper, we design circuits for fast separating and combining an exponent and a fraction of the floating-point arithmetic based on triple exponential cut, the key technology to realize fast calculation. To achieve this, we limit the range of an exponent to that of URR by represent the separated exponent in binary integer of the same length as the floating-point data. Then we detect and settle the positions and length of each parts compose the floating-point number, and process fast using combinational logic circuits in bit parallel manner. Furthermore we estimate execution time and hardware cost when data length is 64 bits and implemented in 0.8 μm CMOS gate array LSI. As a result, we conclude it is attainable to separate and combine an exponent and a fraction by increase of 13% or less in execution time and hardware cost compared to URR except separation time that increases 41% under above condition.

1. はじめに

計算機による数値計算において，オーバフローやアンダフローが発生すると，正常に計算を継続するにはスケーリングなどの面倒な処理が必要となり，処理時間も増加する．そこで，この厄介な問題を避けるため，指数部を可変長とすることにより数値の表現範囲を従来の指数部固定長浮動小数点表現に比べて飛躍的に拡大した浮動小数点表現が提案されてい

† 株式会社日立製作所 中央研究所
Central Research Laboratory, Hitachi Ltd.

る[1)~3)]．これらの中で，浜田の提案した2重指数分割に基づくURR（Universal Representation of Real numbers）[3)]は，オーバフローやアンダフローが事実上発生しないことに加えて，フォーマットが比較的簡潔であり，データ長に独立など，好ましい性質を持つことから注目され，表現精度の面からの評価，改良が試みられている[4)~8)]．改良は，指数部をより少ないビット数で表現する点に集中している．これらのうち，中森らは3重指数分割に基づく表現は，数値の絶対値が1から大きく離れた領域でURRに比べて指数部を短かくでき，URRの好ましい性質を継承したまま表現精度が改善されることを示した[4)]．また，横尾は自然数の表現の立場から指数部の表現に検討を加え，さらに表現精度を改善した3重指数分割に基づく表現が存在することを示している[5),6)]．

一方，実用化の立場からは，高速演算方式や，高速演算ハードウェアの開発が欠かせない．しかし従来の研究の多くは，ソフトウェアによるエミュレーションのレベルにとどまっており，URRに関するもの[9)~12)]を除けばきわめて少ない．Azmiらは Morrisの表現[1)]の演算方式を検討しているが，シフトレジスタベースで考察しており，ハードウェア実現には至っていないと思われる[13)]．

そこで，本論文では，3重指数分割に基づく浮動小数点数のハードウェアによる高速演算の実現を目的として，そのキー技術である，指数と仮数の高速分離，結合方式，および回路を設計する．そしてURR演算のための指数と仮数の高速分離，結合回路[11),12)]との比較で処理時間とハードウェア量を評価する．演算を高速化するには，できるだけ簡素な手続きで演算できることが望ましい．しかし，指数部の構成が複雑となるので，指数と仮数の分離と結合に，より時間がかかることが予想される．そこで本論文では処理を高速化し，ハードウェアの増加を最小限に抑えるため，分離後の指数の表現を分離前のデータの長さに等しい2進整数表現に制限する．分離後の指数の表現を前記に制限しても，URRと同様，オーバフローやアンダフローは事実上発生しないといえる．そして，データを構成する各部の位置と長さを高速に検出，確定し，組合せ回路によるビット並列処理で高速化を図る．さらに，定量的評価を行うため，データ長64ビット，ゲート長0.8 μm のCMOSゲートアレイに実装する場合を仮定して処理時間とハードウェア量を見積もる．その結果，指数の範囲を前記に制限することにより，URRと比較して，分離は，処理時間1.41倍，ハードウェア量1.07倍で，結合は，処理時間1.10倍，ハードウェア量1.13倍で実現できる見通しが得られることを示す．

以下，2章で対象とする表現の概要を説明し，3章で演算の手順と分離後の指数と仮数の表現について，4章では高速分離方式とその実行回路を，5章では高速結合方式とその実行回路を設計し，6章で評価する．なお，丸め処理については簡単化のため，本論文では切り捨てを行うものとした．

2. 3重指数分割に基づく浮動小数点数表現の概要

本章では，本論文で対象とする3重指数分割に基づく浮動小数点表現をデータフォーマット上で概説する．特徴は，**図1**に示すように，指数部が L 部，N 部，E 部の3つのフィールドから構成されることである．すなわち L 部は1/0のビット列の長さで N 部の長さを表し，N 部は数値で E 部の長さを表し，E 部は指数の大きさを表す．URRには N 部がなく，L 部が直接 E 部の長さを表すところが異なる．N 部を設けることにより，指数の絶対値が大きい領域ではURRに比べて指数部の長さを短くできるので，表現精度が改善される．

いま，実数 x を

$$x = 2^e \times f$$

ただし，e は整数

f は，

$x \geq 0$ のとき　$1 \leq f < 2$

$x < 0$ のとき　$-2 \leq f < -1$

とする．このとき f は2進数表現（負数は2の補数）で

$$f = \cdots 01.f_1 f_2 \cdots f_n \quad (x \geq 0)$$

$$f = \cdots 10.f_1 f_2 \cdots f_n \quad (x < 0)$$

と表されるので，小数点以下のビット列

$$f_1 f_2 \cdots f_n$$

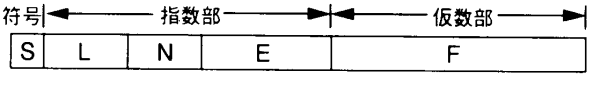
を仮数部Fとする．

次に指数部の表現を説明する．

(1) $x \geq 0$ のとき

(a) $e \geq 0$ で e がちょうど m 桁で表されるとき

$$e = \cdots 01 e_{m-1} \cdots e_2 e_1$$



L: N部の長さを0／1のビット列の長さで表わす．
N: E部の長さを数値で表わす．
E: 指数の値を表わす．
F: 仮数の値を表わす．

図1 3重指数分割に基づく浮動小数点数のデータフォーマット
Fig. 1 Data format of a floating-point arithmetic number based on triple exponential cut.

と表されるので，ビット列 $e_{m-1}\cdots e_2e_1$ を E 部とする．

e の長さは m ビットであるから，m がちょうど k ビットで表されるとき

$$m = 1n_{k-1}\cdots n_2n_1$$

と表されるので，ビット列 $n_{k-1}\cdots n_2n_1$ を E 部の前に付加して N 部とする．

次に L 部は，N 部の長さが $k-1$ ビットであるから $k+1$ 個の1のビット列と区切りの0を N 部の前に付加して L 部とする．

以上により指数部は，

$$\underbrace{1\quad\cdots\quad 10}_{L\text{部}}\ \underbrace{n_{k-1}\cdots\quad n_1}_{N\text{部}}\ \underbrace{e_{m-1}\cdots\quad e_1e_1}_{E\text{部}}$$

（L 部：$k+1$ 個）

と表される．

(b) $e<0$ で e がちょうど m 桁で表されるとき

$$e = \cdots 10e_{m-1}\cdots e_2e_1$$

と表されるので，$e_{m-1}\cdots e_2e_1$ を E 部とする．N 部は $e\geq 0$ のときと同じ手順で求めて1の補数（1と0を反転）をとる．L 部は $k+1$ 個の0のビット列に区切りの1を付加する．以上により指数部は，

$$\underbrace{0\quad\cdots\quad 01}_{L\text{部}}\ \underbrace{\bar{n}_{k-1}\cdots\quad \bar{n}_1}_{N\text{部}}\ \underbrace{e_{m-1}\cdots\quad e_1e_1}_{E\text{部}}$$

（L 部：$k+1$ 個）

となる．

(2) $x<0$ のとき

$x\geq 0$ のときと同様な手順で指数部を求め，1の補数をとって指数部とする．

以上に3重指数分割に基づく浮動小数点数のデータフォーマットの構成を説明したが，ここで具体的な数値により，指数部の構成例を示す．

いま，$x\geq 0$，$e=94$ のとき，e は2進数表現で

$$e = \cdots 01011110$$

と表されるので，E 部は011110の6ビットとなる．次に N 部は値7を表す2進数表現の先頭1ビットを除く2ビット11である．N 部が2ビットであるので L 部は $x\geq 0$ とすれば，4個の1に区切りの0を付加して11110となり，結局，指数部は，

$$\underbrace{1\,1\,1\,1\,0}_{L}\ \underbrace{1\,1}_{N}\ \underbrace{0\,1\,1\,1\,1\,0}_{E}$$

で表される．

3. 演算方式

3.1 演算の手順

図2に演算の一般的手順を示す．3重指数分割に基づく浮動小数点数を演算するには，指数と仮数に分離した後，指数と仮数のそれぞれで演算し，演算結果の指数と仮数を結合して3重指数分割に基づく浮動小数点数を生成する．表現法の性質から比較や符号反転など一部に指数と仮数に分離することなく直接演算可能なものもあるが，通常は前記手順による．指数と仮数に分離した後の演算は従来の指数部固定長浮動小数点数の演算に基本的に同じであるので，高速演算実現は，指数と仮数の分離，および結合の高速化にかかっているといえる．

3.2 分離後の指数と仮数の表現

分離した指数と仮数のデータ長は，それぞれ分離前の3重指数分割に基づく浮動小数点数のデータ長に等しくする．これにより仮数は指数部の長さが最短（2ビット）の場合にも全ビットを保持できる．しかし，分離後の指数をオーバフロー（アンダフロー）に至るまで2進整数表現でハードウェアに保持するのは，もはや困難であり，たとえば浮動小数点数により表現する必要がある．しかし，それでは演算手続きの複雑化による処理速度の大幅低下とハードウェアの増加が避けられない．そこで，これを最小限に抑えるため，ハードウェアでの演算単位との整合性も考慮して，分離後の指数は分離前のデータに等しい長さの2進整数で表す．指数の範囲が制限されるが，URRと同様にオーバフローやアンダフローは事実上発生しないといえる．分離後の指数が上記範囲を超える場合をハード

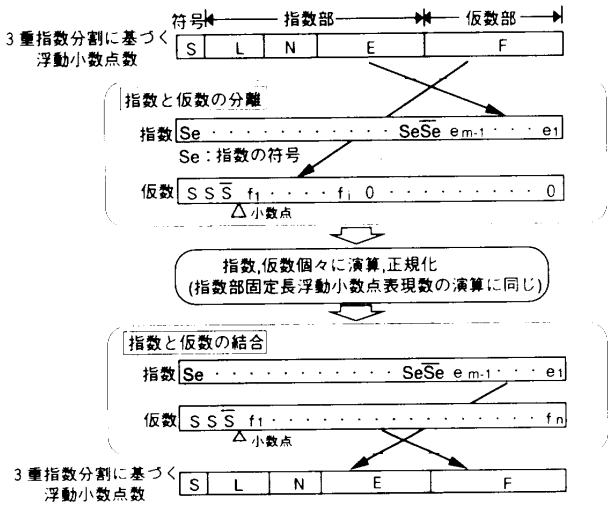


図2 演算の手順

Fig. 2 General scheme of calculation.

ウェアで検出する必要がある場合は，4章で説明する境界コードからただちに判別できる．

4. 指数と仮数の分離

4.1 高速分離方式

図3(a) に指数の分離の，図3(b) に仮数の分離の手順を示し，以下に指数の分離から説明する．なお，データ長は64ビットとする．

(1) L 部と N 部の境界検出，コード化

指数と仮数を分離するには，L, N, E, F の各部の位置と長さを知る必要がある．そこで，まず L 部と N 部の境界を検出，コード化する．仮数の符号 S の右隣りのビット λ を基準に右に順次調べ，最初の反転ビットが見つかれば，それが L 部の右端のビットであるので反転ビットの位置をコード化，先頭に仮数の符号 S と λ を付加して境界コードを作成する．境界コードは，**表1**に示すように，64ビットのデータからは8ビットのコードが生成される．境界コードからは図3における k がわかるので，L 部と N 部の位置と長さ，E 部の左端の位置がわかる．

(2) N 部の分離

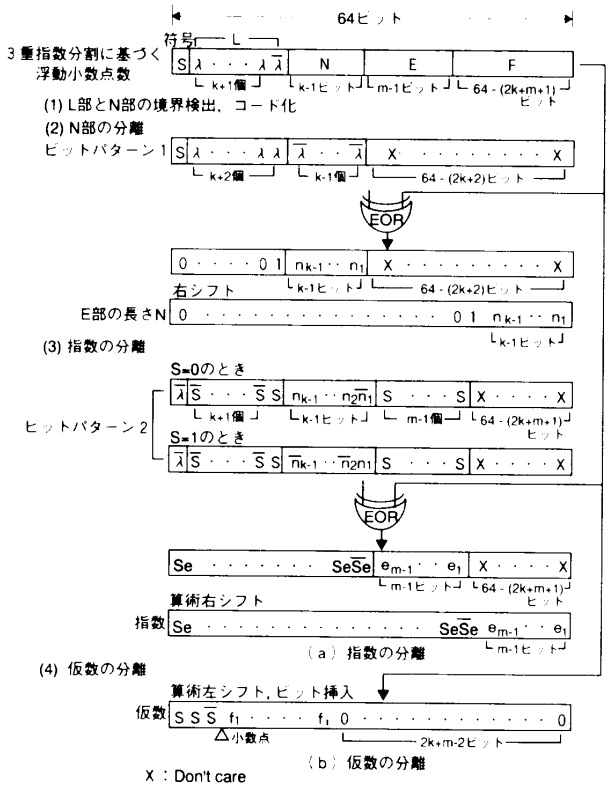


図3 指数仮数の分離の手順
Fig. 3 Separation process of an exponent and a fraction.

N 部を切り出し，節約表現（正規化条件により一意に定まる最上位ビットの表現を省略する）の省略ビットを付加し，N 部の最下位ビットを右端に揃える．これは境界コードをもとに生成したビットパターン1と排他的論理和演算を行うことにより S, L 部をゼロにクリア，L 部の右端ビットを1とした後，右シフトすることにより行う．シフトビット数 $64-(2k+2)$ は境界コードから決まる．分離した N 部の値からは E 部の長さ $m-1$ が決まる．

(3) 指数の分離

仮数の符号 S，および L, N 部を指数の符号 Se に変換する．これは境界コードと分離した N 部をもとに生成したビットパターン2との排他的論理和演算により行う．最後に算術右シフトして E 部の最下位ビットを右端に揃える．このときシフトビット数 $64-(2k+m+1)$ は境界コードと分離した N 部の値から決まる．

(4) 仮数の分離

まず，F 部の最上位ビットが左端から4ビット目に位置するように左シフトする．小数点は左から3ビット目と4ビット目の間にある．このとき，算術シフトを行うことにより左端の仮数の符号 S を保存する．シフトビット数 $2k+m-2$ は，境界コードと N 部の値から決まる．ただし，左端から2ビット目は仮数の符号 S，3ビット目は，その反転ビットを挿入し，節約表現の省略ビットを復元する．

以上に指数部が存在する場合の指数と仮数の分離方式について述べたが，指数の絶対値がきわめて大きい場合は，固定データ長では仮数部が存在しない場合がありうる．たとえば，データ長が64ビットの場合，指数の桁数が51桁以上では仮数部が存在せず，さらに E 部の一部も欠けた状態となる．このような状態は，実際の演算ではまず発生しないと考えられるが，オーバフローやアンダフローではないので，演算機構としては演算を継続できるようにしておきたい．そこで，以下にこのような場合の指数仮数の分離方式について述べる．

図4に仮数部が存在しない場合の指数と仮数の分離の手順を示す．N 部の分離，指数の符号の復元，指数の分離の手順は，仮数部が存在する場合に同じであるが，指数の分離は左シフトとなるところが異なる．また，仮数の分離では，仮数は全ビットが丸められていると解釈し，定数（正のときは +1.0，負のときは −2.0）を置数する．

表1 境界検出とコード化（データ長64ビット）
Table 1 Boundary detection and encoding its position.

3重指数分割に基づく浮動小数点数 #63 … #55 … #30 … #0	コード (HEX)	解釈
0	3 F	0
0 0 1	0 0	N部なし
0 1 X	0 1	E部なし
0 1 X	1 E	E部なし
0 1 X	1 F	仮数部なし
0 1 X	3 6	仮数部なし
0 1 X	3 7	N≧51のとき 仮数部なし N≦50のとき 仮数部あり
0 1 X	3 8	仮数部あり
0 0 1 X	3 D	仮数部あり
0 1 0 X	7 D	仮数部あり
0 1 0 X	7 8	仮数部あり
0 1 0 X	7 7	N≧51のとき 仮数部なし N≦50のとき 仮数部あり
0 1 0 X	7 6	仮数部なし
0 1 0 X	5 F	E部なし
0 1 0 X	5 E	E部なし
0 1 0 X	4 1	N部なし
0 1 0	4 0	N部なし
0 1	7 F	N部なし
1 0	B F	N部なし
1 0 0 1	8 0	N部なし
1 0 0 1 X	8 1	E部なし
1 0 0 1 X	9 E	E部なし
1 0 0 1 X	9 F	仮数部なし
1 0 0 1 X	B 6	仮数部なし
1 0 0 1 X	B 7	N≧51のとき 仮数部なし N≦50のとき 仮数部あり
1 0 0 1 X	B 8	仮数部あり
1 0 1 X	B D	仮数部あり
1 1 0 X	F D	仮数部あり
1 0 0 1 X	F 8	仮数部あり
1 0 0 1 X	F 7	N≧51のとき 仮数部なし N≦50のとき 仮数部あり
1 0 0 1 X	F 6	仮数部なし
1 0 0 1 X	D F	E部なし
1 0 0 1 X	D E	E部なし
1 0 0 1 X	C 1	E部なし
1 0	C 0	N部なし
1	F F	N部なし

注） X は don't care を表す.

4.2 分離回路

以上に説明した高速結合方式の実行回路を構成する．高速処理を実現するため，組合せ回路によりビット並列処理を行う．**図5**にその回路構成を示す．

(1) 境界位置検出コード化回路

図6に回路の内部構成を示す．8ビットの部分ビット列に分割して並列に反転ビットの位置を調べることにより検出の高速化を図っている．

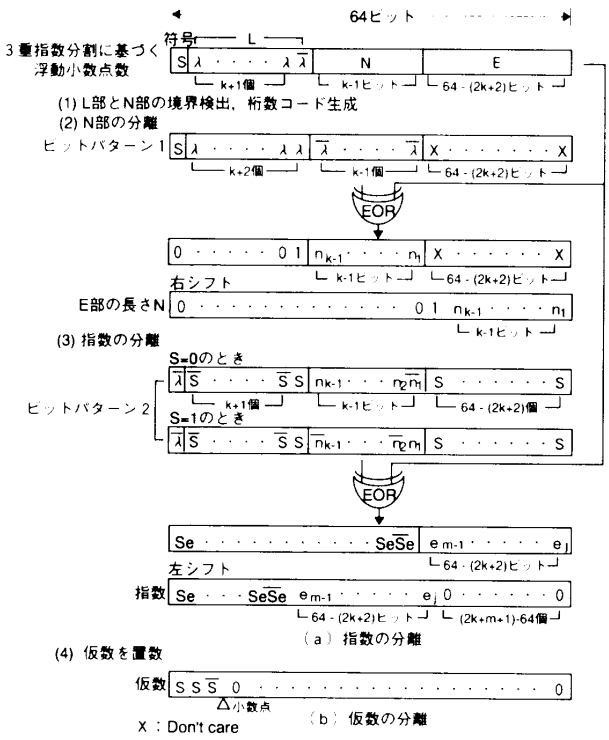


図4 指数仮数の分離の手順（仮数部なし）

Fig. 4 Separation process of an exponent and a fraction (with no fraction part).

非数では反転ビットが存在しない場合があるが，この場合は反転ビット位置を示す6ビットをすべて1として他と区別する．すなわち，部分ビット列が反転ビットを含まなければフラグ1を0とし，フラグ1がすべて0ならフラグ2を1とする．境界コードは表1に示すように，非数を区別し，4つのコードを除いて仮数部の有無も区別するが，4つのコードについては仮数部の有無を知るには，さらに N の値を調べる必要がある．また，分離後の指数が2進64ビット整数（負数は2の補数）で表される条件は，$N \leq 63$ であり，境界コードから判別できる．

(2) ビットパターン発生回路1

N 部を分離するため，境界コードを基に図3，図4におけるビットパターン1を発生する．E 部，F 部に対するビットは必要ないので，分離後の指数の長さを64ビット整数に制限する場合はビットパターンの長さは14ビットとなり，ハードウェア量は少なくて済む．

(3) シフト回路1

境界コードにより制御し，N 部の右端ビットを右端に揃えるために，$64-(2k+2)$ ビット右シフトを行う．分離後の指数の表現を64ビット整数に制限する場合は，シフトビット数の範囲は50～62ビットであり，かつシフトの対象は N 部までなので，ハードウェア量は少なくて済む．

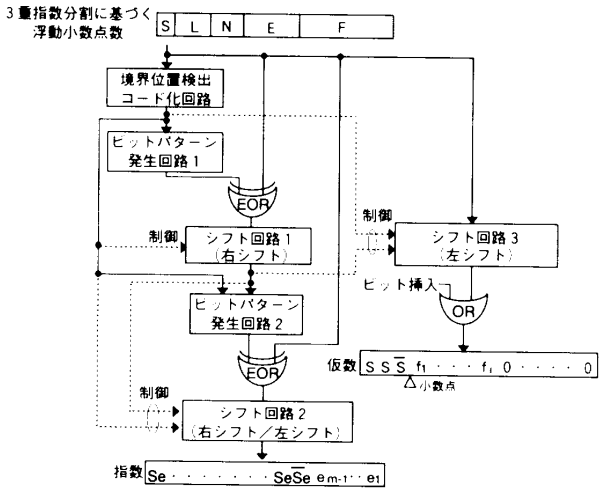


図5 指数仮数の高速分離回路の構成

Fig. 5 A block diagram of a circuit for fast separation of an exponent and a fraction.

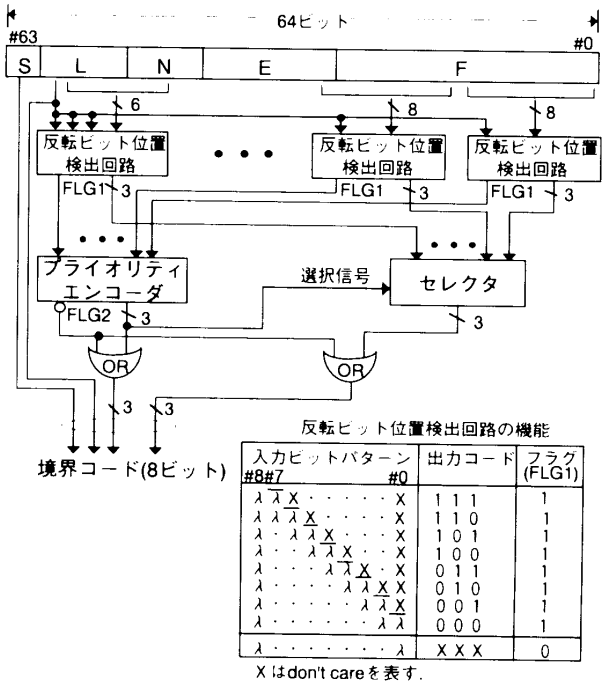


図6 境界検出コード化回路

Fig. 6 A block diagram of a circuit for fast boundary detection and encoding.

(4) ビットパターン発生回路2

仮数の符号 S，L 部，N 部を指数の符号 Se に変換するため，境界コードと分離した N 部の値とから図3，図4におけるビットパターン2を発生する．ただし N 部の右端ビットは節約

表現の省略ビットを復元し，仮数が負の場合は E 部は1の補数とする．

(5) シフト回路2
境界コードと分離した N の値により制御する．指数の符号 Se を復元した結果を $64-(2k+m+1)$ ビット算術右シフトし，指数を分離する．

(6) シフト回路3
境界コードと分離した N の値により制御する．仮数を分離するため，小数点が左端から3ビット目と4ビット目の間に位置するように，$2k+m-2$ ビット算術左シフトを行う．このとき左端から2ビット目を仮数の符号 S で，3ビット目をその反転ビットで置き換える．

以上は仮数部が存在する場合であるが，図4に示した仮数部が存在しない場合の指数と仮数の分離は，シフト回路2を算術左シフトとすることにより同一回路構成で処理できる．

5. 指数仮数の結合

5.1 高速結合方式

図7 に指数と仮数を結合して3重指数分割に基づく浮動小数点数を生成する手順を示し，以下に説明する．ただし，データ長は64ビットとする．

(1) 指数桁数の検出
まず指数の桁数を調べる．指数の桁数が分かれば E 部の長さが決まるので，E 部の長さを表す N 部の値と長さが決まり，L 部の長さ，仮数部Fの長さが順次定まる．すなわち図7における k，m を決定できる．また L 部ビット列は仮数の符号 S と指数の符号 Se とから決まる．そこで指数の左端のビットを基準として右にビット反転を順次調べ，最初の反転ビットの位置に指数の符号を添えて**表2**に示すように指数コードを生成する．

(2) 指数部の生成
指数を左シフトし，E 部の位置に合わせる．シフトビット数 $64-(2k+m+1)$ は指数コードから決まる．次に E 部の位置合わせ結果とビットパターン3との排他的論理和演算により L，N 部を生成し，先頭ビットをゼロにクリアして仮数の符号 S を埋め込む準備を行う．このとき指数の最上位ビットは節約表現により表示されないので，N 部の最下位ビットに変換する．ビットパターン3は指数コードと仮数の符号 S とから図7に示すように決まる．

(3) 仮数部の生成
仮数の上位3ビットを0にクリアし，右シフトして仮数部 F の位置に合わせる．シフトビット数 $2k+m-2$ は指数コードから決まる．

(4) 指数部と仮数部の結合
生成した指数部と仮数部を論理和演算により結合，左端に符号 S を付加して3重指数分割に基づく浮動小数点数を生成する．

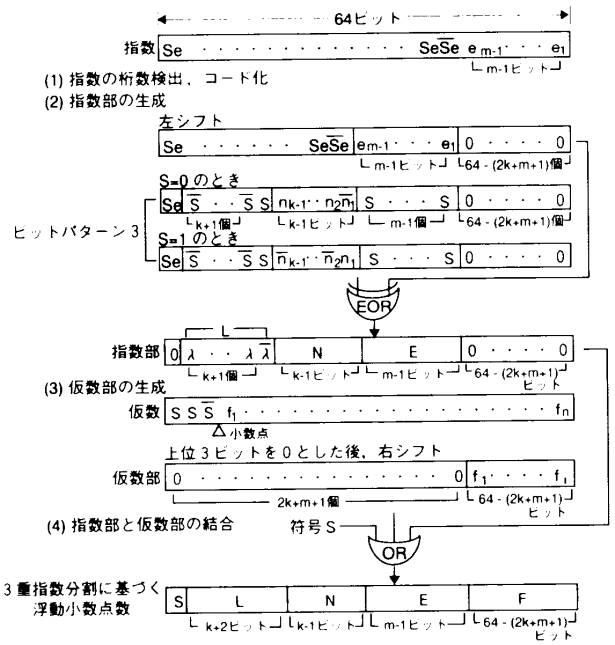


図7 指数仮数の結合手順
Fig. 7 Combining process of an exponent and a fraction.

次に，**図8** に仮数部が存在しない場合の指数と仮数の結合の手順を示す．仮数部が存在する場合との違いは，指数部生成時のシフトが右算術シフトであること，仮数部の生成では全ビットシフトアウトされることである．すなわち仮数部は存在しないが，これは全ビットが丸められたと考えて，正のときは $+1.0$ を，負のときは -2.0 をあて，指数と仮数の分離時に置数する．

5.2 結合回路

以上に説明した指数と仮数の高速結合の実行回路を構成する．組合せ回路によるビット並列処理を行い処理の高速化を図る．**図9** に回路構成を示し，以下に説明する．

(1) 指数桁数検出コード化回路
指数の左端のビットを基準に順次右に調べ，最初の反転ビットの位置をコード化し，左端に指数の符号 Se を付加して表2に示す指数コードを生成する．回路は図6の境界検出回路と同様な構成であり，指数コードは以下のビットパターン発生回路，シフト回路の制御に用いる．

表2 指数のビットパターンとコード化（データ長64ビット）
Table 2 Bit patterns of an exponent and encoding.

指数のビットパターン（64ビット） #63 … #49 … #0	コード（HEX）		意味
0 … 0	3	F	
0 … 0 1	0	0	
0 … 0 1 X	0	1	
0 … 0 1 X … X	3	0	仮数部あり
0 … 0 1 X … X	3	1	
0 … 0 1 X … X	3	2	
0 1 … 1	3	E	
1 0 … 0	7	E	仮数部なし
1 … 1 0 X … X	7	2	
1 … 1 0 X … X	7	1	
1 … 1 0 X … X	7	0	
1 … 1 0 X	4	1	仮数部あり
1 … 1 0	4	0	
1 … 1	7	F	

注） X は don't care を表す．

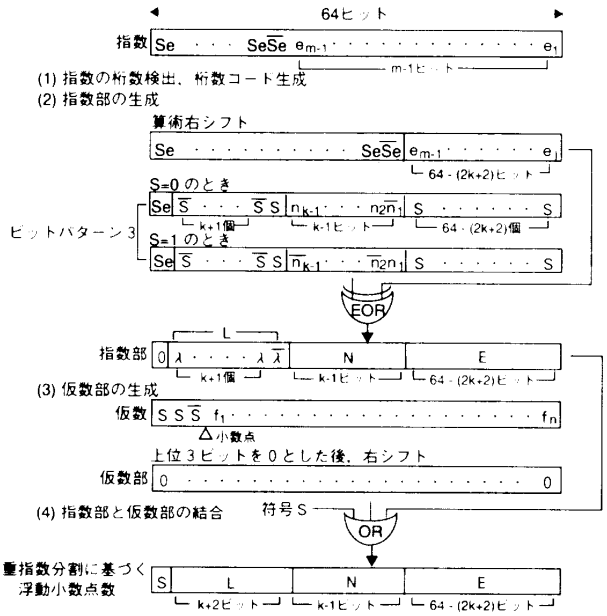


図8 指数仮数の結合手順（仮数部なし）
Fig. 8 Combining process of an exponent and a fraction (with no fraction part).

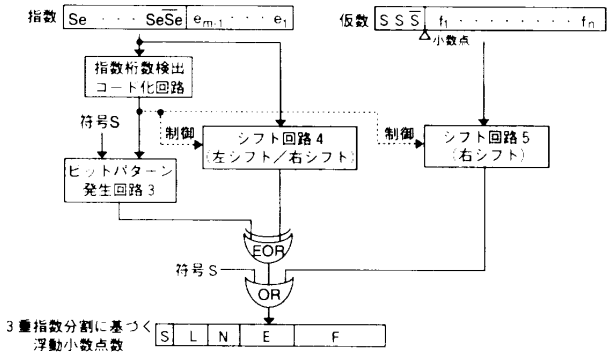


図9 指数仮数の高速結合回路
Fig. 9 A block diagram of a circuit for fast combining an exponent and a fraction.

(2) シフト回路4
指数コードにより制御する．指数部を生成するため，指数を $64-(2k+m+1)$ ビット左シフトして E 部の位置に合わせる．

(3) ビットパターン発生回路3
指数コードと仮数の符号 S とから図7，図8に示すビットパターン3を発生，シフト後の指数の符号 Se のビット列との排他的論理和演算により左端ビットをゼロにクリアし，L 部，N 部を生成する．このとき指数の最上位ビット（Se の反転ビット）は節約表現により表示せずに N 部の最下位ビットに変換，仮数が負の場合は，E 部を含めて1の補数（ビット反転）をとって指数部とする．

(4) シフト回路5
指数コードにより制御する．仮数部を生成するため，仮数の上位3ビットをゼロとした後，$2k+m-2$ ビット右シフトし，仮数部 F の位置に合わせる．

以上に仮数部が存在する場合について述べたが，仮数部が存在しない場合は，シフト回路4を算術右シフトとすることにより，同一回路構成で図8に示す結合手順を実行できる．

表3 評価に用いたゲートアレイLSIの基本仕様
Table 3 Basic specifications of the gate array LSI.

項目	仕様
プロセス	ゲート長0.8 μm CMOS，3層アルミ配線
伝搬遅延時間	2入力NAND（内部ノーマルゲート）： 立ち上がり伝搬遅延時間 (t_{PLH}) = 0.35 ns (typical) 立ち下がり伝搬遅延時間 (t_{PHL}) = 0.56 ns (typical) 2入力NAND（内部パワーゲート）： 立ち上がり伝搬遅延時間 (t_{PLH}) = 0.26 ns (typical) 立ち下がり伝搬遅延時間 (t_{PHL}) = 0.34 ns (typical) ただし， ファンアウト= 2，ファンアウトあたり配線長2 mm 周囲温度 (T_a) = +25 ℃，電源電圧 (V_{cc}) = +5.0 V

6. 評　　価

ハードウェア化で重要な指標となる，指数と仮数の分離，結合に要する処理時間とハードウェア量をURRとの比較で評価する．これらはトレードオフの関係にあり，回路の構成や実装によっても変わるが，ここでは典型的なハードウェア技術で実現する場合の具体的な数値を比較しておきたい．そこでデータ長を64ビットとし，CMOSゲートアレイへの実装を想定して以下の方法で評価した．なお評価に用いるのはゲート長0.8 μm CMOSゲートアレイ[14]で，その基本仕様を**表3**に示す．

6.1 評価の方法

評価の方法と手順を以下に示す．

(1) 論理設計
まず，図5，図9に示す回路の詳細論理を設計する．このとき，処理に必要なビットパターンは，入力データからAND-ORの組合せ論理により発生する．他の方法として，ROMにルックアップテーブルとして格納しておくことも考えられる．コンパイルドROMが利用できるセルベースICでは検討の余地があるかもしれない．

(2) ゲートアレイへの実装設計
次に設計した論理を，ゲートアレイの設計ツールで用意されているセルライブラリ[14]を用いてゲートアレイに実装できる論理回路を構成する．利用したライブラリは，8入力までの(N)AND，(N)OR，およびExclusive ORの基本ゲートである．ただし，シフト回路は，AND-ORを組み合わせた複合ゲートを用い実装効率の改善を図った，また，ファンアウトの数が大きい場合は駆動能力の高いパワーゲートを用いた．

(3) 処理時間の見積もり
処理に要する時間を見積もるため，クリティカルパスの伝搬遅延時間を算出するが，ここで算出する伝搬遅延時間は配置配線前の見積もり値である．すなわち，まずファンアウト1個あたりの配線長を2 mmと仮定した仮想配線容量に，負荷ゲート入力容量を加算して仮想負荷容量を求める．次にゲート単体の遅延時間に仮想負荷容量による遅延時間を加算し，周囲温度が+25 ℃，電源電圧が+5 V時の標準（typical）値を算出する．算出したクリティカルパスの遅延時間を処理に要する時間の見積もり値とする．

(4) ゲート数の見積もり
ゲート数は，ゲートの種類ごとにセルライブラリの換算表[14]に基づいて2入力NANDゲートの数に換算して求める．

以上の条件のもとでの処理のシーケンスと見積もった処理時間を，URRの場合を含めて**図10**に示す．また，ハードウェア量を含めたURRとの比較評価を**表4**に示す．

6.2 結果と考察

(1) 処理時間
URRに比べて分離は1.41倍である．図10から分かるように，増加の主たる要因はN部の分離の手続きが必要となることにある．次に，結合では，N部を含めた指数部は一度に生成でき，手続きの数は同じである．しかし，指数部の構成が複雑なので，指数部の生成に必要なビットパターンの発生に要する時間が増加，URRに比べて1.08倍となった．

(2) ハードウェア量
分離回路では，N部の分離回路が必要であるにもかかわらず，URRに比べて1.07倍に抑えられている．指数の範囲を制限したことにより，N部の分離に要する回路が小さくて済んだことが貢献している．一方，結合回路の構成要素はURRと同じであるが1.13倍のゲート数を

表4 指数仮数の分離，結合回路のURRとの比較評価
Table 4 Comparison of processing time and hardware cost to those of URR.

表現方式	3重指数分割に基づく実数値表現		URR	
評価項目	処理時間	ハードウェア量	処理時間	ハードウェア量
指数仮数の分離	20.2 ns (1.41)	3828 ゲート (1.07)	14.3 ns (1.0)	3563 ゲート (1.0)
指数仮数の結合	15.8 ns (1.10)	4022 ゲート (1.13)	14.3 ns (1.0)	3566 ゲート (1.0)

注) ゲート数は2入力NANDゲート換算

評価条件

(1) データ長は64ビット，分離後の指数は64ビット2進整数表現とする．
(2) ゲート長0.8 μm CMOSゲートアレイに実装．
(3) カッコ内は，URRを1.0とする相対評価．

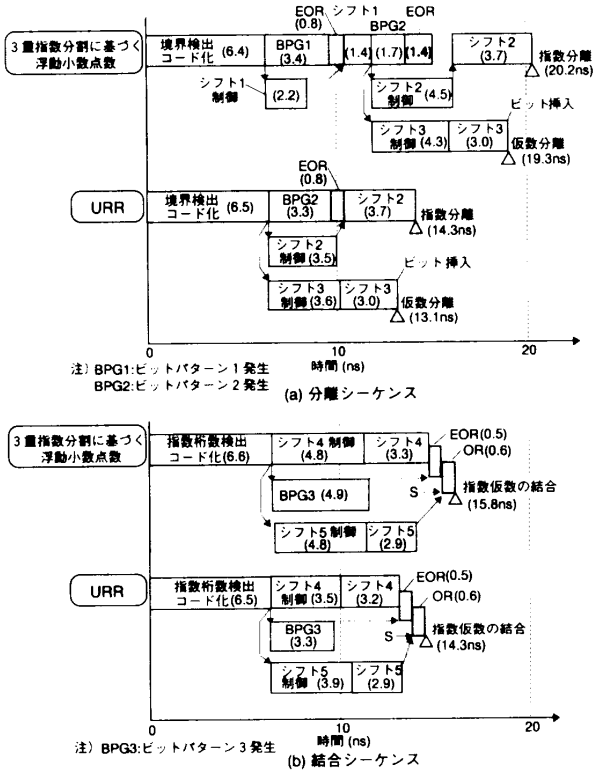


図10 指数仮数の分離，結合シーケンスと処理時間
Fig. 10 A chart of sequence and estimated timing of separating and combining an exponent and a fraction.

要している．これはURRに比べてシフト量の変化の規則性が低下，制御回路を含めてシフト回路のハードウェア量が増加することによる．

(3) 演算器の全体構成との関係に関する考察

高速浮動小数点数演算器は，通常，パイプライン演算器として構成され，指数と仮数の分離，結合はそれぞれ演算の前後で行われる．従来の指数部固定長浮動小数点数の演算では，指数と仮数の分離，結合に要する時間は演算時間に比べて十分短く，演算のステージに含めることができる．しかし，3重指数分割に基づく浮動小数点数の演算では，これを無視することは難しくなる．しかし本論文で述べた指数と仮数の分離，結合回路は，フィードバックパスを持たない組合せ論理回路であり，パイプラインのステージに組み込むことができる．この場合，演算時間(latency)は増加するが，スループットは従来の浮動小数点数演算に比べてさほど低下しないことが期待できる．

一方，他の構成として，浮動小数点演算器に内蔵されるレジスタファイルに指数と仮数を分離して格納することが考えられる．この場合，レジスタファイルに格納されたデータ間の演算では，指数と仮数の分離，結合は行われないので，そのためのオーバヘッド時間は発生しない．また，データがレジスタファイルにとどまる限りは，指数の桁数が大きくなっても表現精度は低下しない．これらの点では3重指数分割に基づく浮動小数点数やURRの演算には有利な構成といえる．ただし，レジスタファイルの容量が2倍必要であるほか，演算器へのデータ入出力時に行う，指数と仮数の分離，結合に要する時間の影響をいかに小さくできるかがポイントとなる．しかし，これらを詳細に議論するにはシステムレベルでの設計と定量評価が必要であり，今後の課題としたい．

7. おわりに

3重指数分割に基づく浮動小数点数のハードウェアによる高速演算の実現を目的として，そのキー技術となる，指数と仮数の高速分離，結合回路を設計した．そして処理時間とハードウェア量を見積もり，URRと比較，評価した．その結果，分離後の指数の表現を64ビット整数に制限することにより，データ長64ビット，0.8 μm CMOSゲートアレイに実装する場合，分離は，処理時間1.41倍，ハードウェア量1.07倍で，

結合は，処理時間1.10倍，ハードウェア量1.13倍で実現の見込みを得た．3重指数分割に基づく浮動小数点表現では，指数のレンジが格段に広くなる．このため高速演算ハードウェアにおいて，定義範囲すべてを扱おうとすれば，処理時間，ハードウェア量の著しい増加を招くことが予想される．URRでは全定義範囲をハードウェアで扱うことが十分可能であるが，3重指数分割に基づく浮動小数点数演算のハードウェア化にあたっては，指数の範囲に制限を設けるのが現実的と考える．また実用化の視点からは，今後さらに，実際にLSIに実装しての評価や，実システムでの試用など，総合的な評価を進めることが必要であろう．

謝　辞

有益なご助言をいただいた電気通信大学情報工学科浜田穂積教授に深謝いたします．

参考文献

1) Morris, R.: Tapered Floating Point: A New Floating-Point Representation, *IEEE Trans. Comput.*, Vol.TC-20, No.6, pp.1578–1579 (1971).
2) 松井正一，伊理正夫：あふれのない浮動小数点表示方式，情報処理学会論文誌，Vol.21, No.4, pp.521–526 (1980).
3) 浜田穂積：二重指数分割に基づくデータ長独立実数値表現法II，情報処理学会論文誌，Vol.24, No.2, pp.149–156 (1983).
4) 中森真理雄，土井　孝：3重指数分割による数値表現方式について，電子情報通信学会論文誌，Vol.J71-A, No.7, pp.1468–1469 (1988).
5) 横尾英俊：自然数の表現に基づく多重指数分割浮動小数点表示方式のクラス，電子情報通信学会論文誌，Vol.J72-A, No.12, pp.1998–2004 (1989).
6) Yokoo, H.:Overflow/Underflow-Free Floating-Point Number Representation with Self-Delimiting Variable-Length Exponent Field, *IEEE Trans. Comput.*, Vol.TC-41, No.8, pp.1033–1039 (1992).
7) 中森真理雄，萩原洋一：多重指数分割による数値表現における多重度の変動化について，情報処理学会数値解析研究報告，Vol.89, No.55 (1989).
8) 中森真理雄：変動式多重指数分割による数値表現方式，電子情報通信学会論文誌，Vol.J72A, No.6, pp.1009–1011 (1989).
9) 中原雅彦，中川正樹，高橋延匡：URR浮動小数点コプロセッサのアーキテクチャの検討と言語C処理系catによる利用形式，第36回情報処理学会全国大会論文集，pp.210–220 (1988).
10) 中原雅彦，早川栄一，岡野裕之，並木美太郎，高橋延匡：複数の浮動小数点表現法を処理するシステム環境の設計と実現，情報処理学会論文誌，Vol.33, No.4, pp.481–490 (1992).
11) 大山光男，浜田穂積：URR演算のための指数・仮数高速分離結合回路方式，電子情報通信学会春期全国大会論文集，分冊6, p.49 (1989).
12) 大山光男，浜田穂積：URR浮動小数点数演算のための指数仮数高速分離・結合回路方式とそのURRプロセッサへの応用，情報処理学会論文誌，Vol.35, No.8, pp.1642–1651 (1994).
13) Azmi, A.M. and Lombardi, F.:On a Tapered Floating Point System, *Proc. IEEE Symposium on Computer Arithmetic*, pp.2–9 (1989).
14) 日立製作所：HG62S/Gシリーズデザインマニュアル，第3版 (1993).

(平成7年7月24日受付)
(平成8年2月 7 日採録)

大山　光男（正会員）

1972年，名古屋工業大学工学部電子工学科卒業．同年（株）日立製作所中央研究所入社．計算機ハードウェア，アレイ型ディスク記憶装置等の研究開発に従事．1996年4月より倉敷芸術科学大学産業科学技術学部教授．電子情報通信学会，ACM，IEEE各会員．