

解 説**4. 各種のハードウェアアルゴリズム****4.1 算術演算のハードウェアアルゴリズム†**高 木 直 史^{††} 矢 島 脩 三^{††}**1. ま え が き**

加減乗除、開平等の算術演算は、計算機をはじめとするさまざまなデジタルシステムにおいて基本演算として広く用いられており、従来、高速アルゴリズムの研究、開発が盛んに行われてきた^{1), 2)}。特に、近年の集積回路技術の進歩に伴い、規則正しい回路構造をもつ高速算術演算回路の研究が盛んになってきている。本稿では、加減算、乗算、除算、開平および指数・対数関数等の初等関数について、従来知られている種々のアルゴリズムを実用的なものを中心に紹介するとともに、新しい研究として、内部計算に冗長2進表現を利用した VLSI 向き的高速ハードウェアアルゴリズムについても述べる。

優れたハードウェアアルゴリズムを設計するには、データ表現やデータ構造を工夫することが有効であると考えられる。本稿で紹介する冗長2進表現を利用した算術演算用ハードウェアアルゴリズムは、数表現の工夫により高速計算と VLSI 向きの規則正しい回路構造を同時に実現したものであり、データ表現の工夫による優れたハードウェアアルゴリズムの好例であるといえる。

本稿では、次章で計算モデルに関する準備を行う。3章では加減算、4章では乗算、5章では除算のハードウェアアルゴリズムについて述べ、6章では開平および初等関数のためのハードウェアアルゴリズムについて述べる。

2. 準 備

本稿では、算術演算回路を組合せ回路として実現した場合の段数や素子数によってアルゴリズムを評価する。組合せ回路は、ある与えられた基本論理素子を用

いて構成される、フィードバックループを持たない論理回路である。本稿では、回路を構成する基本素子の入次数 (fan-in) は、ある定数以下に制限されるものとする。出次数 (fan-out) は制限しない。また、遅延は素子内部にのみ存在するものとし、配線上での遅延は考慮しない。このような仮定の下では、回路での計算時間は回路の段数に比例すると考えられる。組合せ回路の段数とは、回路の入力から出力に至る最長経路、すなわち最も多くの素子をもつ経路、上の素子の個数である。また、組合せ回路の素子数とは、回路を構成する素子の総数である。

本稿では、回路を VLSI 上に実現した場合の面積についても考える³⁾。VLSI 上では、素子はある一定値以上の面積をもち、配線はある一定値以上の幅をもつものとする。また、素子は他の素子や配線と重ならないものとし、任意の点での配線の重なり (配線層数) はある定数以下であるとする。VLSI 上での回路の面積は、回路を含む平面上の最小の凸領域の面積によって定義する。ただし、回路の入出力は凸領域の周上で行うものとする。

3. 加減算のハードウェアアルゴリズム

加減算は最も基本的な算術演算の1つで広く用いられており、多くの加減算器が設計され実用されている。ここでは n ビット2進数の加減算について考える。加減算を高速化するには桁上げの伝搬をいかに高速に行うかが問題である。全加算器 (FA) を単純に一列に接続した順次桁上げ法に基づく加算器では、桁上げが1桁ずつ順次伝搬して行くため、計算時間は $O(n)^*$ になる。素子数と面積ともに $O(n)$ になる。これに対し、桁上げを木状に伝搬させる桁上げ先見法に基づく加算器では、計算時間は $O(\log n)$ と高速で、素子数は $O(n)$ であるが、面積は $O(n \log n)$ になる。

† Hardware Algorithms for Arithmetic Operations by Naofumi TAKAGI and Shuzo YAJIMA (Department of Information Science, Kyoto University).

†† 京都大学工学部情報工学科

* $O(n)$ は n が大きくなった場合漸近的に n に比例することを示す。

4. 乗算のハードウェアアルゴリズム

n ビット2進数の乗算について考える。乗算についてはいろいろなアルゴリズムが提案されているが、ここでは実用的な並列乗算法として、配列型乗算法、並列カウンタ型乗算法および冗長2進加算木を用いた乗算法を紹介する。

4.1 配列型乗算法

配列型乗算法⁴⁾は加算シフト型乗算法を組合せ回路用に展開したもので、 n 個の部分積を順次加えていく。ただし、各加算を桁上げ保存方式で行い、最後の加算でのみ桁上げを伝搬させるようにすることにより、計算を高速化している。配列型乗算器は図-1に示すように全加算器を2次元配列状に並べた構成になる。段数は $O(n)$ 、素子数と面積はともに $O(n^2)$ になる。

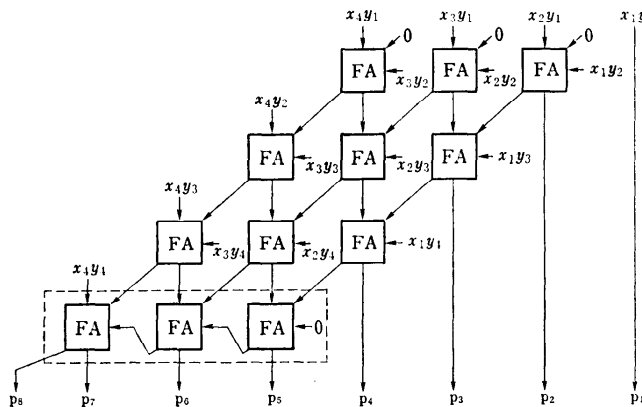


図-1 配列型乗算器

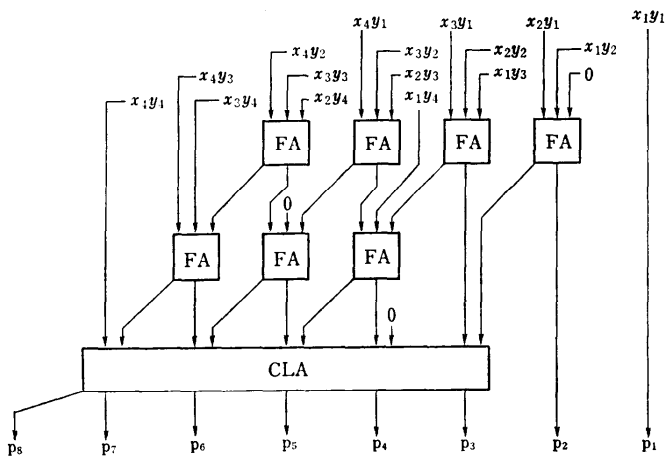


図-2 Wallace 木を用いた乗算器

る。図からわかるように規則正しいセル配列構造でLSI化に適しており、乗算用LSIや信号処理用LSIで用いられている。最後の加算に桁上げ先見加算器を用いることや拡張Boothの方法で乗数をリコード(recode)し部分積の数を減らすこと^{5),6)}等により、計算を高速化できる。

配列型乗算法の改良として、加算系列を2本に並列化することにより配列部の段数をおよそ半分に減らした変形配列型乗算法が提案されており、LSIが試作されている⁷⁾。

4.2 並列カウンタ型乗算法

並列カウンタ型乗算法は、 n 個の部分積を並列カウンタを用いて木状に加えていくもので、Wallace木を用いた乗算法⁵⁾等が知られている。Wallace木を用いた乗算法では、 m 個の全加算器を用いれば m に

無関係な一定時間で3つの m ビット2進数 x, y, z を $x+y+z=u+v$ となる2つの2進数 u, v に変換できることを利用している。この方法で部分積の数を約2/3にする操作を $\lceil \log_3 n \rceil$ 回繰り返して、 n 個の部分積を2数にし、最後にこの2数を桁上げ先見加算器(CLA)で加え合せる。Wallace木を用いた乗算器は図-2に示すような構成になる。基本セルは配列型乗算法と同様、全加算器であるが、配線は複雑になる。段数は $O(\log n)$ と高速で、素子数は $O(n^2)$ であるが、面積は $O(n^2 \log n)$ のレイアウトしか知られていない⁸⁾。

Wallace木では基本セルとして全加算器を用いているが、全加算器は3ビットの入力中の1の個数を計数する並列カウンタと見ることができる。Wallace木と同じ原理で、基本セルとして7-3カウンタや15-4カウンタを用いて乗算器を構成することができ、並列カウンタ型乗算器と呼ばれている。並列カウンタ型乗算器にも拡張Boothの方法が適用でき、段数や素子数を減らすのに有効である。並列カウンタ型乗算器は高速の大型計算機等で用いられている。

4.3 冗長2進加算木を用いた乗算法

冗長2進加算木を用いた乗算法⁹⁾では、内部計算に冗長2進表現を利用する。冗長2進表現はA. Avizienisが提案したSD(Signed-Digit)表現¹⁰⁾の一種である。通

常の2進表現と同様、下位から i 番目の桁は 2^{i-1} の重みをもつが、各桁は0か1か-1(以後 $\bar{1}$ と表わす)である。冗長2進表現には冗長性があり、1つの数をいく通りにか表わすことができる。例えば、'5'は4桁で $[0101]_{SD2}$ の他、 $[011\bar{1}]_{SD2}$ 、 $[10\bar{1}\bar{1}]_{SD2}$ などと表わすことができる。ただし、'0'の表現はひとつである。冗長2進数体系では、その冗長性を利用し、2数の加算において桁上げの伝搬をなくすることができる。桁上げが伝搬しない加算は2つのステップで行う。第1ステップでは、各桁で、被加数 x_i と加数 y_i から、 $x_i + y_i = 2c_i + s_i$ となるように、中間桁上げ $c_i (i \in \{\bar{1}, 0, 1\})$ と中間和 $s_i (i \in \{\bar{1}, 0, 1\})$ を定める。このとき、中間和 s_i と1つ下位からの桁上げ c_{i-1} がともに1あるいはともに $\bar{1}$ となることのないよう

表-1 冗長2進数体系での加算の第1ステップの計算規則

被加数 (x_i)	加数 (y_i)	1つ下位の桁 (x_{i-1}, y_{i-1})	中間桁上げ (c_i)	中間和 (s_i)
1	1	—	1	0
1	0	両方とも非負	1	$\bar{1}$
0	1	少なくとも一方負	0	1
0	0	—	0	0
0	$\bar{1}$	—	0	0
$\bar{1}$	1	—	0	0
0	$\bar{1}$	両方とも非負	0	$\bar{1}$
$\bar{1}$	0	少なくとも一方負	$\bar{1}$	1
$\bar{1}$	$\bar{1}$	—	$\bar{1}$	0

被加数 $[1\bar{1}1\bar{1}00\bar{1}01011]_{SD2}$
 加数 $+ [10\bar{1}00\bar{1}\bar{1}\bar{1}\bar{1}101]_{SD2}$
 中間和 $0101010101\bar{1}0$
 中間桁上げ $+ 1\bar{1}000\bar{1}\bar{1}00111$
 和 $[1\bar{1}10\bar{1}\bar{1}0011000]_{SD2}$

図-3 冗長2進数体系における加算の例

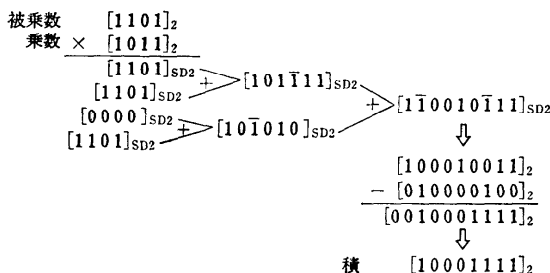


図-4 冗長2進加算木を用いた乗算の例

に、表-1に示すような規則に従い、1つ下位の桁の x_{i-1} と y_{i-1} も調べて c_i と s_i を決定する。第2ステップでは、各桁で、 $s_i + c_{i-1} = z_i$ となるように和 $z_i (i \in \{\bar{1}, 0, 1\})$ を求める。このとき新たな桁上げは生じない。図-3に桁上げが伝搬しない加算の例を示す。 z_i は $x_i, y_i, x_{i-1}, y_{i-1}, x_{i-2}, y_{i-2}$ の6つのみから定まる。したがって、組合せ回路による並列加算が演算数の桁数に関係なく一定の段数で行える。必要な素子数は演算数の桁数に比例する。

冗長2進加算木を用いた乗算法では、まず n 個の部分積を生成し、冗長2進表現で表わす。通常の2進数は各桁が0か1であるから、そのまま冗長2進表現と見なすことができる。次に n 個の部分積を2つずつ2分木状に冗長2進数体系で加え合せ、冗長2進表現で表わされた積を得る。加算木内の同レベルでの加算は並列に行う。冗長2進数体系では2数の並列加算が演算数の桁数に無関係な一定時間で行えるので、加算木の各レベルでの計算は一定時間で行える。加算木の高さは $\lceil \log_2 n \rceil$ であるから、 $O(\log n)$ の計算時間で冗長2進表現で表わされた積が求まる。最後に積を通常の2進表現に変換する。この変換には、冗長2進表現で1である桁だけを1とする2進数から $\bar{1}$ である桁だけを1とする2進数を引く減算が必要であり、桁上げ先見加算器を用いれば $O(\log n)$ の計算時間で行える。したがって、全体として、計算時間 $O(\log n)$ で n ビット乗算が行える。図-4に冗長2進加算木を用いた乗算の例を示す。

冗長2進加算木を用いた乗算器は図-5に示すような構成になる。段数は $O(\log n)$ 、素子数は $O(n^2)$ で、面積は $O(n^2 \log n)$ になる。また、規則正しいセル配列構造でLSI化に適している。冗長2進加算木を用いた乗算器にも拡張Boothの方法が効率よく適用できる。冗長2進加算木を用いた乗算法に基づく16ビット乗算器のLSIが試作されている¹¹⁾。

4.4 乗算用ハードウェアアルゴリズムの比較

表-2に、上記の3つの乗算法に基づく乗算器の比較を示す。また、表-3に、3つのタイプの乗算器を同じ2値論理素子(4入力NOR/OR)を用いて論理設計した場合の段数と素子数の例を示す。2つの表からわかるように、計算時間は、並列カウンタ型と冗長2進加算木を用いたものはほぼ等しく、配列型はこれらに較べ特に n が大きい場合非常に大きくなる。素子数は3つの乗算器で差はない。面積は配列型が $O(n^2)$ であるのに対し、他の2つは

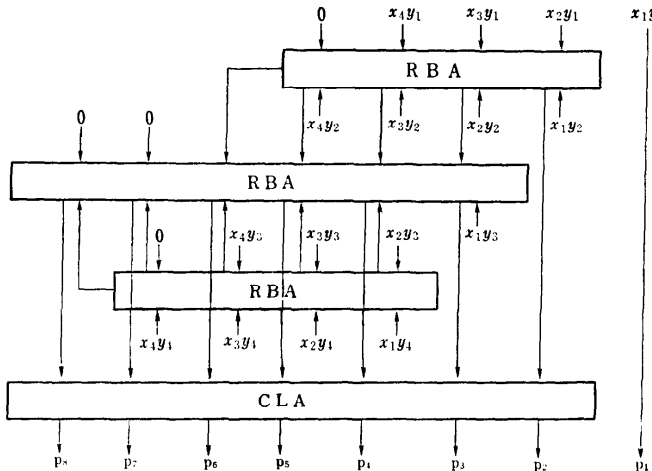


図-5 冗長2進加算木を用いた乗算器 (RBA: 冗長2進加算器)

表-2 乗算器の比較

配列型	段数 (計算時間)	素子数	面積	レイアウト
配列型	$O(n)$	$O(n^2)$	$O(n^2)$	容易
並列カウンタ型	$O(\log n)$	$O(n^2)$	$O(n^2 \log n)$	複雑
冗長2進加算木	$O(\log n)$	$O(n^2)$	$O(n^2 \log n)$	容易

表-3 乗算器の段数と素子数 (段数/素子数)

n	16	32	64	128
配列型*	61/2336	125/9792	253/40064	509/162048
Wallace 木*	24/2939	30/9965	34/37423	40/142335
冗長2進加算木	28/2535	33/9948	39/38842	44/152385

* 文献12)による。

$O(n^2 \log n)$ となる。しかし、冗長2進加算木を用いた乗算器は配列型乗算器と同様の規則正しいセル配列構造で、並列カウンタ型乗算器に較べレイアウトが容易である。

他にもいくつかの乗算用ハードウェアアルゴリズムが提案されている。このうち実用的であると考えられるものとしては、内部計算に各桁が0か1か2である桁上げ保存表現を利用した乗算法がある¹³⁾。この乗算法に基づく乗算器は、冗長2進加算木を用いた乗算器と同様の2分木構造になる。段数や素子数も冗長2進加算木を用いた乗算器と同程度になる。

5. 除算のハードウェアアルゴリズム

n ビット2進小数の除算について考える。被除数、

除数とも正規化された n ビット2進小数で $1/2$ 以上1未満であり、商を小数点以下 n ビットまで求めるものとする。除算についてもいろいろなアルゴリズムが提案されている。理論的には、剰余系演算を利用した $O(\log n)$ 段の除算法が最近提案されたが¹⁴⁾、非常に複雑であり、その実用化に関する検討は今後の課題である。ここでは実用的な除算法として、減算シフト型除算法と乗算型除算法を紹介する。減算シフト型除算法の一種として、内部計算に冗長2進表現を利用した高速除算法も紹介する。

5.1 減算シフト型除算法

減算シフト型除算法は、減算とシフトの繰返しにより商を上位桁から順に求めて行くもので、回復型除算法（引き戻し法）の他、非回復型除算法（引き放し法）やSRT除算法¹⁵⁾等が知られている。減算シフト型除算法に基づく除算器は、減算器（加算器）とシフタからなる順序回路として容易に実現することができ、特にマイクロプログラムの制御により効率よく計算を行えるので、広く用いられてきた。

一般に、2進の減算シフト型除算法は、 $R^{(j+1)} = 2 \times R^{(j)} - q_j \times D$ という漸化式に従う。 D は除数、 q_j は商の小数点以下 j 桁目、 $2 \times R^{(j)}$ は q_j を決定する前の部分被除数、 $R^{(j+1)}$ は q_j を決定した後の部分剰余である。回復型除算法では、 $2 \times R^{(j)} - D$ が非負ならば q_j を1、負ならば0とし、部分剰余 $R^{(j+1)}$ が常に非負になるようにする。非回復型除算法では商の各桁を $\{\bar{1}, 1\}$ の中から定める。部分剰余が負になってもそのまま計算を進め、次のステップで商を $\bar{1}$ として除数を加える。すなわち、 $2 \times R^{(j)}$ が非負ならば q_j を1、負ならば $\bar{1}$ とする。SRT除算法では商の各桁を $\{\bar{1}, 0, 1\}$ の中から定める。 $2 \times R^{(j)} > (1/2)$ ならば $q_j = 1$ 、 $-(1/2) \leq 2 \times R^{(j)} \leq (1/2)$ ならば $q_j = 0$ 、 $2 \times R^{(j)} < -(1/2)$ ならば $q_j = \bar{1}$ とする。 q_j として0を許すことにより加減算の回数を減らし、非回復型除算法を高速化している。

減算シフト型除算器を順序回路として実現する場合は、SRT除算法が高速化に有効であるが、組合せ回路として実現する場合は、単純な回復型除算法が適している。組合せ回路として実現した場合の段数や素子数、面積のオーダは、3つの除算法で違いはない。各加減算にどのような加算器を用いるかにより全体の段

数や素子数、面積が決まる。順次桁上げ加算器を用いると、段数、素子数、面積はともに $O(n^2)$ になり、規則正しいセル配列構造で LSI 化に適している。桁上げ先見加算器を用いると、段数は $O(n \log n)$ 、素子数は $O(n^2)$ となるが、面積は $O(n^2 \log n)$ になり、レイアウトも順次桁上げ加算器を用いる場合より複雑になる。

5.2 冗長2進表現を利用した減算シフト型除算法

従来の減算シフト型除算法では、各加減算における桁上げの伝搬のため大きな計算時間が必要であった。減算シフト型除算法を高速化するための1つの手法として、SD 表現を利用し各加減算において桁上げの伝搬をなくすことが有効であり¹⁶⁾、2進の除算では4.3で述べた冗長2進表現が利用できる。

冗長2進表現を利用した減算シフト型除算法¹⁷⁾は、従来の減算シフト型除算法と同じく、 $R^{(j+1)} = 2 \times R^{(j)} - q_j \times D$ という漸化式に従う。ただし、部分剰余 $R^{(j)}$ を整数部1桁小数部 n 桁の冗長2進表現で表わし、その上位3桁の正負0により q_j を $\{\bar{1}, 0, 1\}$ の中から定める。すなわち、 $R^{(j)}$ の上位3桁が正なら q_j を1、0なら0、負なら $\bar{1}$ とする。 $R^{(j+1)}$ を求める計算は冗長2進数体系で行う。除数 D は各桁が非負の冗長2進数（通常の2進数）であるから、4.3で述べたより計算が簡単である。 q_j は $R^{(j)}$ の上位3桁のみから定まり、また、冗長2進数体系では並列加減算が演算数の桁数に無関係な一定時間で行えるので、漸化式

1回分の計算が n に無関係な一定時間で行える。この計算を n 回繰返し、 n 桁の冗長2進表現で表わされた商を得る。最後に商を通常の2進表現に変換する。したがって、全体として、計算時間は $O(n)$ となる。

図-6に冗長2進表現を利用した減算シフト型除算の例を示す。

冗長2進表現を利用した減算シフト型除算法に基づく除算器を組合せ回路として実現すると、図-7のよ

$$\begin{array}{ll} R = [0.100111]_2 & \text{被除数} \\ D = [0.110001]_2 & \text{除数} \end{array}$$

$$\begin{array}{r} q_0=1 \quad 0.100111 \\ \quad -0.110001 \\ \hline q_1=\bar{1} \quad 0.0\bar{1}0110 \\ \quad +0.110001 \\ \hline q_2=1 \quad 0.100\bar{1}1\bar{1} \\ \quad -0.110001 \\ \hline q_3=0 \quad 0.001001 \\ \hline q_4=1 \quad 0.010010 \\ \quad -0.110001 \\ \hline q_5=\bar{1} \quad 0.\bar{1}101\bar{1}1 \\ \quad +0.110001 \\ \hline q_6=1 \quad 0.10\bar{1}00\bar{1} \end{array}$$

$$[1.\bar{1}101\bar{1}1]_{SD2}$$



$$Q = [0.110011]_2 \text{ 商}$$

図-6 冗長2進表現を利用した減算シフト型除算の例

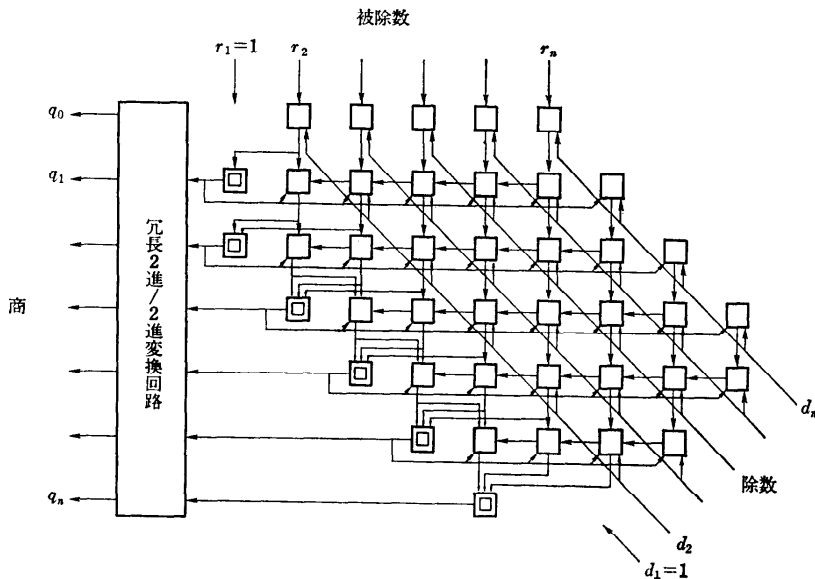


図-7 冗長2進表現を利用した減算シフト型除算器

表-4 除算器の比較

		段数(計算時間)	素子数	面積
減 シ フト 算 型	順次桁上げ加算器	$O(n^2)$	$O(n^2)$	$O(n^2)$
	桁上げ先見加算器	$O(n \log n)$	$O(n^2)$	$O(n^2 \log n)$
	冗長2進加算器	$O(n)$	$O(n^2)$	$O(n^2)$
乗 算 型	配列型乗算器	$O(n \log n)$	$O(n^2 \log n)$	$O(n^2 \log n)$
	並列カウンタ型乗算器 冗長2進加算木を用いた乗算器	$O(\log^2 n)$	$O(n^2 \log n)$	$O(n^2 \log^2 n)$

表-5 除算器の段数と素子数(段数/素子数)

	n	16	32	64	128
減 シ フト 算 型	順次桁上げ加算器	511/2717	2047/11565	8191/47693	32767/193677
	桁上げ先見加算器	128/3103	320/14361	640/64475	1536/284421
	冗長2進加算器	70/2758	136/10939	264/43435	522/172936
乗 算 型	配列型乗算器	244/16352	625/88128	1518/440704	3563/2106624
	Wallace 木を用いた乗算器	96/20573	150/89685	204/411653	280/1850355

うな構成になる。段数は $O(n)$ と高速で、素子数と面積はともに $O(n^2)$ になる。図からわかるように、従来の減算シフト型除算器で順次桁上げ加算器を用いたものと同様の規則正しいセル配列構造で LSI 化に適している。

5.3 乗算型除算法

乗算型除算法は、乗算の繰り返しにより除算を行うもので、高速乗算器をもつ大型計算機等でマイクロプログラムの制御により実現されている¹⁸⁾。乗算型除算法では、 n が 2 のべきで、 $0 < X \leq (1/2)$ であるとき、

$$\frac{1}{1-X} \cong \frac{1-X^{2^n}}{1-X} = (1+X)(1+X^2)(1+X^4) \cdots (1+X^{2^{n-1}})$$

であることを利用している。被除数を R 、除数を D ($(1/2) \leq R, D < 1$)、商を Q とし、 $X = 1 - D$ とすると、

$$Q = \frac{R}{D} = \frac{R}{1-X} \\ \cong R(1+X)(1+X^2)(1+X^4) \cdots (1+X^{2^{n-1}})$$

となり、 $O(\log n)$ 回の乗算で商が求まる。

乗算型除算法に基づく除算器を組合せ回路として実現すると、乗算器として配列型乗算器を用いると、段数は $O(n \log n)$ 、素子数と面積はともに $O(n^2 \log n)$ になり、並列カウンタ型乗算器や冗長2進加算木を用いた乗算器を用いると、段数は $O(\log^2 n)$ 、素子数は $O(n^2 \log n)$ 、面積は $O(n^2 \log^2 n)$ になる。しかし、組合せ回路として実現するには膨大なハードウェアが必要であり、実際的でない。大型計算機等で用いられて

いるように、高速乗算器を用いて順序回路として実現するのが実際的である。

5.4 除算用ハードウェアアルゴリズムの比較

表-4 に、上記の除算法に基づく除算器を組合せ回路として実現した場合の比較を示す。また、表-5 に、これらの除算器を同じ論理素子（4入力 NOR/OR）を用いて論理設計した場合の段数と素子数の例を示す。表からもわかるように、組合せ回路として LSI 化するには冗長2進表現を利用した減算シフト型除算法が最も適している。乗算型除算法は、高速乗算器を用いて順序回路として実現するのが実際的である。

6. 開平および初等関数のハードウェアアルゴリズム

6.1 開平のハードウェアアルゴリズム

理論的には、除算の場合と同じく、計算時間 $O(\log n)$ で n ビットの開平が行える¹⁹⁾。また、計算機向きの実用的な開平法としては、減算シフト型開平法や Newton の収束法等に基づく開平法が知られている。

減算シフト型開平法は、加減算とシフトの繰り返しにより平方根を上位桁から順に求めていくもので、回復型開平法¹⁹⁾や非回復型開平法²⁰⁾が知られている。減算シフト型開平器は、加減算器とシフトからなる順序回路として比較的容易に実現でき、特にマイクロプログラムの制御により効率よく計算を行える。また、組合せ回路として実現すると、加減算に順次桁上げ加算

器を用いると、段数、素子数、面積とも $O(n^2)$ になり、規則正しいセル配列構造で LSI 化に適している。加減算に桁上げ先見加算器を用いると、段数は $O(n \log n)$ 、素子数は $O(n^2)$ であるが、面積は $O(n^2 \log n)$ になり、レイアウトも順次桁上げ加算器を用いる場合より複雑になる。

従来の減算シフト型開平法では、加減算における桁上げの伝搬のため、 n が大きい場合大きな計算時間が必要である。除算の場合と同様、減算シフト型開平法でも内部計算に冗長 2 進表現を利用することにより加減算における桁上げの伝搬をなくし、計算を高速化できる²¹⁾。冗長 2 進表現を利用した減算シフト型開平器を組合せ回路として実現すると、段数は $O(n)$ 、素子数と面積はともに $O(n^2)$ になり、規則正しいセル配列構造で LSI 化に適している。回路構造やハードウェア量は、従来の減算シフト型開平器で順次桁上げ加算器を用いたものと大差ない。

Newton の収束法等に基づく開平法は、ソフトウェアによる開平法として広く用いられてきた。また、アルゴリズムを工夫し、組合せ回路で実現した場合、段数が $O(\log^2 n)$ となるハードウェアアルゴリズムも提案されているが²²⁾、乗算器を多用するのでハードウェア量が膨大になり実際的ではない。

6.2 初等関数のハードウェアアルゴリズム

指数・対数関数、三角関数等の初等関数のための計算機向きのアルゴリズムとしては、Chebyshev 展開²³⁾等の多項式近似による計算法や、STL 法^{24)~26)}、CORDIC 法^{27), 28)}等の収束法が知られている。

多項式近似による計算法は、ソフトウェアによる初等関数計算法として広く用いられてきた。また、組合せ回路として実現すると、段数を $O(\log^2 n)$ にできるが、膨大なハードウェアが必要で実際的ではない。

STL 法や CORDIC 法等の収束法では、加減算とシフトおよび定数の読み出しという単純な操作の繰り返しにより、しだいに求める関数値に収束させていく。例えば、STL 法による指数関数の計算では、任意の $X^{(i)}$, $Y^{(i)}(>0)$, $A^{(i)}(>0)$ に対して、 $X^{(i+1)} + L_n Y^{(i)} = (X^{(i)} - L_n A^{(i)}) + L_n (Y^{(i)} \times A^{(i)})$ が成り立つことを利用している。数列 $\{X^{(i)}\}$, $\{Y^{(i)}\}$ を考え、 $X^{(i+1)} = X^{(i)} - L_n A^{(i)}$, $Y^{(i+1)} = Y^{(i)} \times A^{(i)}$ とすると、 $X^{(1)} + L_n Y^{(1)} = X^{(2)} + L_n Y^{(2)} = \dots = X^{(m)} + L_n Y^{(m)}$ となる。ここで、 $X^{(1)} = X$, $Y^{(1)} = 1$ とし、 $A^{(i)}$ を適当に選んで、 $X^{(m)} \cong 0$ となるようにすると、 $Y^{(m)} \cong \exp X$ となる。

Specker²⁵⁾ や Chen²⁶⁾ によって改良された STL 法では、 $A^{(i)}$ として 1 あるいは $1+2^{-i}$ を選ぶ。 $L_n A^{(i)}$ は定数であり、あらかじめ用意しておく。すると、 $X^{(i+1)}$ の計算は定数の減算、 $Y^{(i+1)}$ の計算は i 桁のシフトと加算で行える。この操作をおよそ n 回繰り返すと、関数値が n ビットまで正確に求まる。対数関数の計算にも同様の原理が利用できる。また、CORDIC 法では、平面座標上でのベクトルの段階的な回転を考えることにより、三角関数の値を計算する。

収束法に基づく初等関数計算回路の実現法としては、加算器とシフタおよび減算器と定数テーブル用のメモリからなる順序回路が考えられており、マイクロプログラムの制御により同一の操作を繰り返すことにより効率よく計算が行える。CORDIC 法については LSI も試作されている。また、組合せ回路として実現すると、加減算器として順次桁上げ加算器を用いると、段数と素子数はともに $O(n^2)$ になり、桁上げ先見加算器を用いると、段数は $O(n \log n)$ 、素子数は $O(n^2)$ になる。

STL 法や CORDIC 法も内部計算に冗長 2 進表現を利用し高速化することが可能で、組合せ回路で実現すると、段数は $O(n)$ 、素子数は $O(n^2)$ になる^{29), 30)}。

7. む す び

算術演算のハードウェアアルゴリズムについて、実用的なものを中心に紹介した。算術演算については古くから研究が行われており、すでにさまざまなアルゴリズムが提案され、実用されている。しかし、VLSI 時代を迎えた現在、ハードウェアによる高速計算の立場から VLSI 向きの優れたハードウェアアルゴリズムを設計することは、ますます重要な課題になってきている。算術演算用のハードウェアアルゴリズムでは、本稿で紹介した冗長 2 進表現等の冗長表現や剰余表現等を利用することが有効であろう。

謝辞 本稿で紹介した冗長 2 進表現を利用した算術演算用ハードウェアアルゴリズムの研究を共同で行っている京都大学工学部安浦寛人博士ならびに矢島研究室の諸氏に感謝致します。

参 考 文 献

- 1) Hwang, K.: Computer Arithmetic/Principles, Architecture, and Design, John Wiley & Sons (1979). 堀越 彌監訳: コンピュータの高速演算方式, 近代科学社, 東京 (1980).
- 2) 萩原 宏: 電子計算機通論 2 演算・制御装置,

- 朝倉書店, 東京 (1971).
- 3) 安浦, 矢島: 論理回路の VLSI 上での面積について, 信学論 (D), Vol. J65-D, No. 8, pp. 1080-1087, (Aug. 1982).
 - 4) Braun, E. L.: Digital Computer Design, Academic Press, New York (1963).
 - 5) Wallace, C. S.: A Suggestion for a Fast Multiplier, IEEE Trans. Elec. Comput., Vol. EC-13, No. 1, pp. 14-17 (Feb. 1964).
 - 6) Waser, S.: High-Speed Monolithic Multipliers for Real-Time Digital Signal Processing, IEEE Computer, Vol. 11, No. 10, pp. 19-29 (Oct. 1978).
 - 7) Iwamura, J., Suganuma, K., Taguchi, S., Kimura, M. and Maeguchi, K.: A 16-Bit CMOS/SOS Multiplier-Accumulator, Proc. IEEE International Conference on Circuits and Computers 1982, 12.3, (Sep. 1982).
 - 8) 安浦, 矢島: 組合せ論理回路の VLSI 上への埋め込み問題, 信学技報, AL 81-97 (Jan. 1982).
 - 9) 高木, 安浦, 矢島: 冗長 2 進加算木を用いた VLSI 向き高速乗算器, 信学論 (D), Vol. J66-D, No. 6, pp. 683-690 (Jun. 1983).
 - 10) Avizienis, A.: Signed-Digit Number Representations for Fast Parallel Arithmetic, IRE Trans. Elec. Comput., Vol. EC-10, No. 3, pp. 389-400 (Sep. 1961).
 - 11) Harata, Y., Nakamura, Y., Nagase, H., Takigawa, M. and Takagi, N.: High Speed Multiplier Using a Redundant Binary Adder Tree, Proc. IEEE International Conference on Computer Design 1984, (Oct. 1984).
 - 12) 宮田, 安浦, 矢島: 拡張 Booth の方法と Wallace Tree を用いた高速乗算器, 信学論 (D), Vol. J65-D, No. 6, pp. 807-808 (Jun. 1982).
 - 13) Vuillemin J. E.: A Very Fast Multiplication Algorithm for VLSI Implementation, Integration, The VLSI Journal, Vol. 1, No. 1, pp. 39-52 (Apr. 1983).
 - 14) Cook, S. A., Beame, P. W. and Hoover, J.: Log Depth Circuits for Division and Related Problems, 25th IEEE Symp. Foundations on Computer Science (Oct. 1984).
 - 15) Robertson, J. E.: A New Class of Digital Division Methods, IRE Trans. Elec. Comput., Vol. EC-7, No. 3, pp. 218-222 (Sep. 1958).
 - 16) Atkins, D. E.: Design of the Arithmetic Units of ILLIAC III: Use of Redundancy and Higher Radix Methods, IEEE Trans. Comput., Vol. C-19, No. 8, pp. 720-733 (Aug. 1970).
 - 17) 高木, 安浦, 矢島: 冗長 2 進表現を利用した VLSI 向き高速除算器, 信学論 (D), Vol. J67-D, No. 4, pp. 450-457 (Apr. 1984).
 - 18) Anderson, S. F. et al.: The IBM System/360 Model 91: Floating-Point Execution Unit, IBM Journal, Jan. 1967, pp. 34-53 (Jan. 1967).
 - 19) Lenaerts, E. H.: Automatic Square Rooting, Electronic Engineering, Vol. 27, No. 329, pp. 287-289 (Jul. 1955).
 - 20) Cowgill, D.: Logic Equations for a Built-In Square Root Method, IEEE Trans. Elec. Comput., Vol. EC-13, No. 2, pp. 156-157 (Apr. 1964).
 - 21) 高木, 安浦, 矢島: 冗長 2 進表現を利用した VLSI 向き高速開平方, 第 27 回情報処理全国大会論文集, 7J-4 (Oct. 1983).
 - 22) 安浦, 矢島: 組合せ回路における平方根の計算時間について, 信学技報, AL 79-29 (Jul. 1979).
 - 23) 一松: 初等関数の数値計算, シリーズ新しい応用の数学, No. 8, 教育出版 (1974).
 - 24) Cantor, D., Estrin, G. and Turn, R.: Logarithmic and Exponential Function Evaluation in a Variable Structure Digital Computer, IRE Trans. Elec. Comput., Vol. EC-11, No. 4, pp. 155-164 (Apr. 1962).
 - 25) Specker, W. H.: A Class of Algorithms for $\ln x$, $\exp x$, $\sin x$, $\cos x$, $\tan^{-1} x$ and $\cot^{-1} x$, IEEE Trans. Elec. Comput., Vol. EC-14, No. 1, pp. 85-86 (Jan. 1965).
 - 26) Chen, T. C.: Automatic Computation of Exponentials, Logarithms, Ratios, and Square Roots, IBM J. Res. Dev., Vol. 16, No. 4, pp. 380-388 (Jul. 1972).
 - 27) Volder, J. E.: The CORDIC Trigonometric Computing Technique, IRE Trans. Elec. Comput., Vol. EC-8, No. 3, pp. 330-334 (Mar. 1959).
 - 28) Walther, J. S.: A Unified Algorithm for Elementary Functions, AFIPS 1971 SJCC, pp. 379-385 (1971).
 - 29) 高木, 矢島: 冗長 2 進表現を利用した指数・対数関数用ハードウェアアルゴリズム, 信学技報, AL 83-70 (Feb. 1984).
 - 30) 高木, 浅田, 矢島: 冗長 2 進表現を利用した CORDIC 法に基づく三角関数用ハードウェアアルゴリズム, 信学技報, AL 84-59 (Jan. 1985).

(昭和 60 年 1 月 10 日受付)