冗長2進数表現による繰返し乗算方式†

外村 元伸††

コンピュータ・グラフィックスの急激な発展によって，ますます，32ないし64ビットの浮動小数点演算が要求されるようになってきた．このように処理桁数が増加してくると，桁上げ伝播による遅延が問題になり，これを解決するために桁上げ伝播のない演算器が注目されている．そして，{−1, 0, +1} で表現する冗長2進加算器の利用は，従来の桁上げ保存加算器などで構成するよりもすぐれた規則性をもち，加減乗除，開平などの高速演算アルゴリズムが導出できることが最近わかってきた．本報告では，特に冗長2進数表現を利用した乗算方式についてわかってきた最近の成果をもとにして多項式計算などのように繰り返して乗算する方式に応用することについて述べる．すなわち，従来の乗算器が主として桁上げ保存加算器を利用していたために，せいぜい2次のBoothアルゴリズムにしか適していなかったが，冗長2進加算器を利用すると，4次のBoothアルゴリズム相当のものが実現でき，Boothアルゴリズムと相性がよいことに注目する．そして，繰返し乗算においても，演算の途中で桁上げ伝播の発生する通常の2進数に変換することなく冗長2進数体系で一貫して高速に演算できるようにするため，新たに冗長2進数のままリコードする冗長2進数リコーダとその論理回路化方式を提案する．性能評価の結果，本提案方式によれば従来よりも 15〜34% 程度改善できる見通しが得られた．

1. はじめに

強力な演算能力を必要とするコンピュータ・グラフィックス分野が急速に成長し，32ビット（単精度）ないし64ビット（倍精度）の浮動小数点演算が，最近のマイクロプロセッサには，ごく普通に要求されるようになってきた[1]．このように処理桁数が増加してくると，すべての算術演算器の基本となるのは加算器であるため，加算の桁上げ伝播による遅延が本質的に高速化を妨げるようになってくる．ここで問題となる遅延は処理桁数 n に依存し，一定の遅延は本質的とは考えられないので，以降，遅延には処理桁数 n に依存するもののみを主として考えることにする．古くから桁上げ伝播の遅延を減らしたり，なくす方法がいくつか考案されている．遅延を減らすには，桁上げ先見加算器があり，n ビットの加算に $\log_2 n$ の時間を要する[2),3)]．また，遅延をなくす方法としては，桁上げを一時的に保存しておいて，それを次の加算時に加える桁上げ保存加算器がよく使われている．さらに，桁上げ伝播が高々1回で加算ができる符号付き2進数 {−1, 0, +1} を使う方法が古くAvizienis[4]によって考案されている．符号付き2進数表現は，表現ビット数が通常の符号なし2進数表現の2倍必要なことから，計算機の命令語体系の中では扱われていない．しかし，命令語を実際に内部処理する方法については何ら拘束性がないため，乗除算などのように連続して部分加算が多数必要な演算において，この符号付き2進数表現を利用することは，演算の途中に桁上げ伝播が生じないために高速化に大いに役立つ．ところが，最後に結果の符号付き2進数を通常の2進数に変換する過程で，通常の加算器，例えば，桁上げ先見加算器を使わなければならず，この意味では依然，加算器の桁上げ伝播の遅延問題は解決されない．そしてこのような最終結果の逆変換も含めた完全な桁上げ伝播なし加算器を論理的に構成することは原理的に不可能である．したがって，できるかぎり内部処理で符号付き2進数表現を長い間保つことができるような命令語を設けることが，高速化のためには必要になってくる．このように，内部処理に−1という冗長な表現を追加して使うという意味で，以後，符号付き2進数表現を今日よく使われている冗長2進数表現という言葉で統一することにする．

扱う数体系は，桁上げ保存加算器が {0, 1, 2}，冗長2進加算器が {−1, 0, 1} であるが，この対称性を含めた数体系の違いがインプリメント上の違いとなって現れてくる．桁上げ保存加算器は乗算器への応用では，3入力を2出力にできる性質を利用したWallace木を用いて構成する方法がよく知られているが，対称性が悪く配線が複雑になる[3)]．一方，冗長2進加算器を使う方法に関して，最近，高木ら[5)〜7)]，原田ら[8),9)]，仲西ら[10),11)]，谷口ら[12)〜15)]の一連の研究で著しい進展があり，乗除算器その他に応用された．特に，谷口らは，冗長2進加算器の入力能力を利用して2次の

† Iterative Multiplication Method by Redundant Binary Representation by MOTONOBU TONOMURA (Hitachi Central Research Laboratory, Hitachi Ltd.).

†† (株)日立製作所中央研究所

Booth リコーダを2組用いて4次の Booth アルゴリズムに相当する乗算器を実現した．また，部分積を加算するために，冗長2進加算規則の符号に関する対称性を利用して回路をコンパクト化した冗長2進加算器を用いた．そのため，桁上げ保存加算器はせいぜい2次の Booth アルゴリズムにしか適さないのに対し，冗長2進加算器を利用すれば，4次の Booth アルゴリズム相当のものが自然に実現でき，Booth アルゴリズムと相性がよいことが示されたことになる．

本報告では，このような最近の成果を利用するとともに，できるかぎり内部処理で冗長2進数表現を長い間保ち，高速化をはかるために，命令を多項式計算にまで拡大する．すなわち，多項式計算の途中においても桁上げ伝播の発生する通常の2進数に変換することなく冗長2進数体系で一貫して高速に演算する繰返し乗算方式を提案する．そのため，新たに冗長2進数のままリコードするリコーダとその論理回路化方式を提案する．

2. 乗算器の構成

2次の Booth アルゴリズムのリコード規則は，**表1**に示すように被乗数 Y を2ビット単位に分割し，さらにそれぞれ右に隣接する1ビットを加えて隣接3ビット $y_1y_0y_{-1}$ を入力とする．そして各分割単位ごとに入力の隣接3ビット $y_1y_0y_{-1}$ を並列リコードし，それぞれ2桁 $y_1'y_0'$ を出力する．ここで $\bar{1}=-1$ と記号定義する．リコード値は冗長2進数表現になっており，しかも非ゼロ値が高々1桁なので，部分積を求めるには乗数 X の高々1ビットの左シフトと冗長2進加算によって行うことができる．次に，**表2**に示すように Booth アルゴリズムを2次から4次へ拡張する．4次のリコード規則では，被乗数 Y を4ビット単位に分割し，それぞれ右に隣接する1ビットを加えた隣接5ビット $y_3y_2y_1y_0y_{-1}$ が入力される．そして4桁のリコード値 $y_3'y_2'y_1'y_0'$ が出力される．4次のリコード値の大部分は，2組の2次のリコード値を並べたものに等しいが，隣接5ビットが 00100, 10100, 01011, 11011 のときのみ両者のリコード値（上側が4次の，下側が2組の2次のリコード値を並べたものである）が異なる．しかし，どちらにしてもリコード値の非ゼロ値は高々2桁であり，同符号の場合が存在する．ところで，例えば，隣接5ビットが 01100 の場合，そのリコード値は $10\bar{1}0$ で，乗数 X の加算値は $+8X-2X$ であるが，この加算の任意の桁において両方とも1のときは打ち消しあいゼロにすることができる．そのため，乗数 X の加算値は，各桁において正と負の値を相殺する回路を構成することによって簡単に実現できる．また，例えば，隣接5ビットが 01010 の場合，そのリ

表1 2次の Booth リコード規則
Table 1 Second-order Booth recode rule.

隣接3ビット $y_1y_0y_{-1}$	リコード値 $y_1'y_0'$	乗数Xの加算値
0 0 0	0 0	0
0 1 0	0 1	+X
1 0 0	$\bar{1}$ 0	-2X
1 1 0	0 $\bar{1}$	-X
0 0 1	0 1	+X
0 1 1	1 0	+2X
1 0 1	0 $\bar{1}$	-X
1 1 1	0 0	0

表2 2次から4次へ拡張するためのリコード規則
Table 2 Recode rules for extending from second-order to fourth-order.

隣接5ビット $y_3y_2y_1y_0y_{-1}$	リコード値 $y_3'y_2'y_1'y_0'$	乗数Xの加算値	隣接5ビット $y_3y_2y_1y_0y_{-1}$	リコード値 $y_3'y_2'y_1'y_0'$	乗数Xの加算値
0 0 0 0 0	0 0 0 0	0	0 0 0 0 1	0 0 0 1	+X
0 0 0 1 0	0 0 0 1	+X	0 0 0 1 1	0 0 1 0	+2X
0 0 1 0 0	0 0 1 0	+2X	0 0 1 0 1	0 1 0 $\bar{1}$	+4X -X
	0 1 $\bar{1}$ 0	+4X-2X	0 0 1 1 1	0 1 0 0	+4X
0 0 1 1 0	0 1 0 $\bar{1}$	+4X -X	0 1 0 0 1	0 1 0 1	+4X +X
0 1 0 0 0	0 1 0 0	+4X	0 1 0 1 1	1 0 $\bar{1}$ 0	+8X-2X
0 1 0 1 0	0 1 0 1	+4X +X		0 1 1 0	+4X+2X
0 1 1 0 0	1 0 $\bar{1}$ 0	+8X-2X	0 1 1 0 1	1 0 0 $\bar{1}$	+8X -X
0 1 1 1 0	1 0 0 $\bar{1}$	+8X -X	0 1 1 1 1	1 0 0 0	+8X
1 0 0 0 0	$\bar{1}$ 0 0 0	-8X	1 0 0 0 1	$\bar{1}$ 0 0 1	-8X +X
1 0 0 1 0	$\bar{1}$ 0 0 1	-8X +X	1 0 0 1 1	$\bar{1}$ 0 1 0	-8X+2X
1 0 1 0 0	$\bar{1}$ 0 1 0	-8X+2X	1 0 1 0 1	0 $\bar{1}$ 0 $\bar{1}$	-4X -X
	0 $\bar{1}$ $\bar{1}$ 0	-4X-2X	1 0 1 1 1	0 $\bar{1}$ 0 0	-4X
1 0 1 1 0	0 $\bar{1}$ 0 $\bar{1}$	-4X -X	1 1 0 0 1	0 $\bar{1}$ 0 1	-4X +X
1 1 0 0 0	0 $\bar{1}$ 0 0	-4X	1 1 0 1 1	0 0 $\bar{1}$ 0	-2X
1 1 0 1 0	0 $\bar{1}$ 0 1	-4X +X		0 $\bar{1}$ 1 0	-4X+2X
1 1 1 0 0	0 0 $\bar{1}$ 0	-2X	1 1 1 0 1	0 0 0 $\bar{1}$	-X
1 1 1 1 0	0 0 0 $\bar{1}$	-X	1 1 1 1 1	0 0 0 0	0

コード値は 0101 で，乗数 X の加算値は $+4X+X$ であるが，$+4X$ に対しての負の2の補数表現をとり，$-(\overline{4X})-1+X$ のようにすれば，先ほどのように異符号同士の加算として扱える．ここで，$(4X)$ の上側につけた横棒線記号は 1，0 値の反転を示す．最下位桁への－1の加算は冗長2進加算の桁上げ入力として扱える．このように同符号の場合，どちらか一方に対して2の補数表現をとれば，異符号同士の加算として扱えるので相殺回路が使える．

次に，**図1**に示すように2組の2次の Booth リコーダを使って4次相当の Booth リコーダが構成できる[12),13)]．2次のリコーダ対は，入力 $y_3y_2y_1y_0y_{-1}$ に対して，冗長2進数で表されるリコーダの出力対 $(y_3'y_2', y_1'y_0')$ を符号部 s ($y'=\bar{1}$ のとき $s=1$) と絶対値部 z ($y'=1$, $\bar{1}$ のとき $z=1$) に分けて2値信号化して対 $(s_2z_3z_2, s_1z_1z_0)$ として出力する．このように冗長2進数を符号と絶対値に分けて出力するリコーダを構成することによってその符号判定が容易になる．また，2次のリコーダ対を使って4次相当のリコーダを構成するほうが，4次のリコーダを直接構成するよりも論理の実現が簡単になる．

以下に，2次のリコーダ対を使って4次の Booth アルゴリズム相当のものを具体的に実現する乗算器の構成方法について説明する．まず，**図2**に示すように乗数 X および被乗数 Y がそれぞれのレジスタに格納される．このとき，リコードのために Y レジスタの右側に1ビットの0値を余分に追加する．そして，Y レジスタの内容を4ビット単位に並列リコードするために複数個の2次のリコーダ対を設ける．各2次のリコーダ対の出力信号に従って，部分積生成部では桁合わせされた X レジスタの内容をそれぞれ高々1ビットシフト制御して入力する．すなわち，**図3**に示すように，部分積の加算に関して各桁ごとに考えると，X レジスタのある位置の隣接4ビット (x_3x_2, x_1x_0) に対して，その桁には z_i が1のとき x_i が入力される．そして，符号信号 s によって入力値 x_i の符号が決定される．

ところで，リコード対の値に関して，例えば，左 s_2 側は非正 ($=\bar{1}$, 0) の表現のみ ($\bar{1}$ 表現部と呼ぶ) が，右 s_1 側は非負 ($=1$, 0) の表現のみ (1表現部と呼ぶ) が許されるように固定してみる．そして，符号信号対 (s_2, s_1) が許されている表現の符号値とは異なる場合には，シフト入力値の2の補数表現をとることにする．そのために入力の全ビット値を反転し，最下位桁については1表現にするときは 1，$\bar{1}$ 表現にするときは $\bar{1}$ を加算する．ゼロ値の符号信号の値は正なので，2の補数表現に関して，もう少し細かく考えると，all 0の反転 +1 はやはり all 0の意味であるから，$s_2=0$ のとき，$\bar{1}$ 表現へ変換してよいことに注意する．また，両方とも2の補数表現をとる場合は，最下位桁に対する加算をゼロに相殺する．

さらに冗長2進数表現のもう1つの能力，すなわち，任意の桁に $\bar{1}$ 表現値と1表現値が同時に入力された場合，お互いに打ち消し合いゼロになるということ

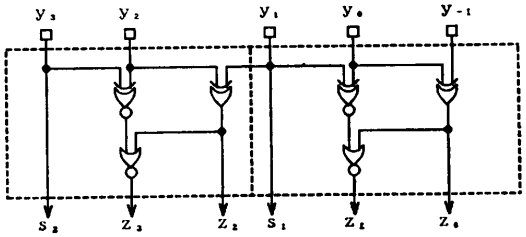

図1 2次の Booth リコーダ×2
Fig. 1 Second order Booth recoder×2.

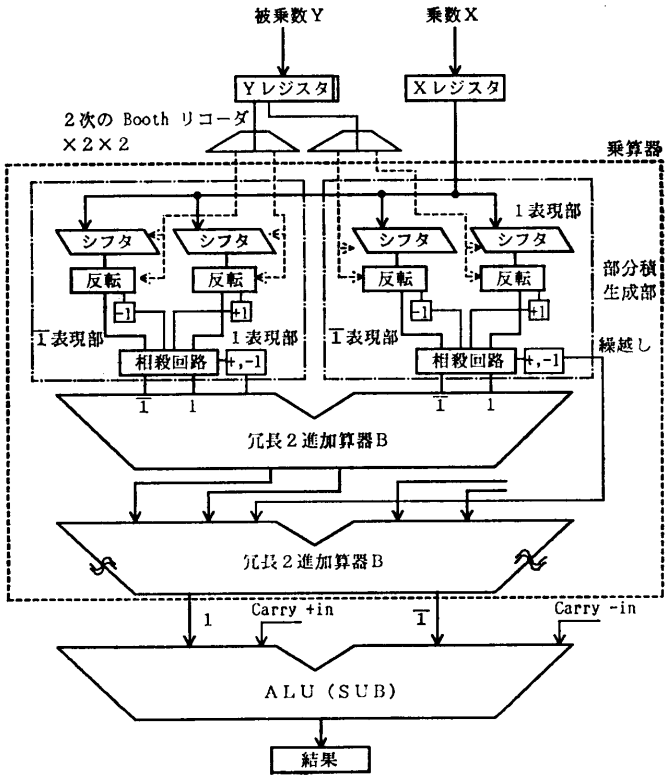

図2 乗算器の構成スキーマ
Fig. 2 Construction schema for multiplier.

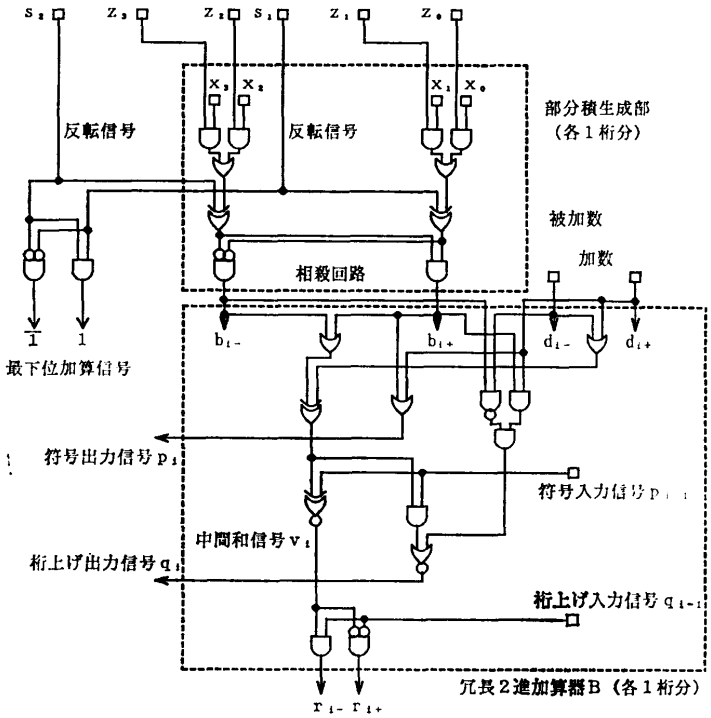

図 3 乗算回路 (部分積生成部+冗長2進加算器B)
Fig. 3 Multiplier (partial product+redundant binary adder B).

に注目して，$\bar{1}$ 表現と1表現の絶対出力値を相殺回路部へ通し，両絶対値がともに1のときはあらかじめ0に相殺する．もちろん最下位桁に対する加算に関しても，$\bar{1}$ 表現部と1表現部の値がともに1のときは0に相殺する．このようにして，まず部分積 $b_i=(b_{i-}, b_{i+})$ が生成される．ここで，添字 i は i 桁目のものであることを，最右添字 $-$ と $+$ は，それぞれ $\bar{1}$ 表現および1表現に関するものであることを示す．この部分積はさらに別のリコーダ対によって制御され求められた部分積 $d_i=(d_{i-}, d_{i+})$ と冗長2進加算される．

このようにして冗長2進加算器Bによって得られた部分積同士の冗長2進加算値 $R=(R-, R+)$ は，さらに別の冗長2進加算器Bによって得られた部分積同士の冗長2進加算値 $R'=(R'-, R'+)$ とさらに別の冗長2進加算器Bによって冗長2進加算される．このように木状に冗長2進加算器Bが配置され冗長2進加算が行われる．2の補数表現をとった場合の最下位桁に対する 1, $\bar{1}$ 加算に関しては，両入力のもののうちの1つは冗長2進加算器の最下位桁への桁上げ入力として扱うことができるので，桁上げ伝播は生じない．もう1つは2個分の冗長2進加算器Bの結果をまとめてさらに冗長2進加算する次段以降の冗長2進加算器Bに繰り越す．最後に，木の根に相当する位置の冗長2進加算器Bの結果 $R=(R-, R+)$ が乗算 $Y\cdot X$ の結果となる．

冗長2進加算の最終段の結果は，1表現部と $\bar{1}$ 表現部に分けて通常の2進数を扱う算術論理ユニット (ALU : Arithmetic Logical Unit) に入力され，減算 (SUB) される．このとき，最後まで繰り越された1個の最下位桁の加算がALUの桁上げ入力として処理される．そして，通常の2進数で表現された乗算 $Y\cdot X$ の結果を得る．

以下に，部分積 b_i と d_i を冗長2進加算する冗長2進加算器Bについて説明する．冗長2進加算器Bの演算規則は，いくつかあるが，例えば，**表3**に示すようなものを採用する．ここでは，従来文献で採用されているものとは異なっているものを採用することによって，いくつかあることを強調し，さらに以下に導く性質はそれらの中で規則表の上下に対称性をもつものに共通していることを明らかにする．演算規則は1桁下位の b_{i-1} と d_{i-1} の値を余分に見ることによって，2桁以上の上位桁へ桁上げ伝播しないように桁上げ c_i の値が決定されている．1桁下位からの桁上げ入力 $c_{i-1}=(c_{i-1+}, c_{i-1-})$ を加えて結果 $r_i=(r_{i-}, r_{i+})$ を得る前の一時的な和 $m_i=(m_{i+}, m_{i-})$ は中間和と呼ばれている．表3の演算規則を論理回路化するために，各桁の結果 $r_i=(r_{i-}, r_{i+})$ が $m_{i-}+c_{i-1+}$ または $m_{i+}+c_{i-1-}$ で表されることに注目する．すなわち，$m_{i-}+c_{i-1+}$ と $m_{i+}+c_{i-1-}$ は互いに排他的であるから，**表4**に示すように桁上げ入力信号 q_{i-1} と中間和信号 v_i の意味を定めれば，

表 3 冗長2進加算器Bの演算規則
Table 3 Computation rule for redundant binary adder B.

被加数 b_i	加数 d_i	被加数 b_{i-1}	加数 d_{i-1}	桁上げ c_i	中間和 m_i
1	1	———		1	0
1	0	下記でない		1	$\bar{1}$
0	1	両方とも非1		0	1
0	0	———		0	0
1	$\bar{1}$				
$\bar{1}$	1				
0	$\bar{1}$	下記でない		0	$\bar{1}$
$\bar{1}$	0	両方とも非1		$\bar{1}$	1
$\bar{1}$	$\bar{1}$	———		$\bar{1}$	0

$m_{i-}+c_{i-1+}$ と $m_{i+}+c_{i-1-}$ を重ねあわせても結果 $r_i=(r_{i-}, r_{i+})$ を得ることができる．桁上げ入力信号 q_{i-1} には，桁上げ入力 $c_{i-1+}=0$ または $c_{i-1-}=\bar{1}$ のときに1の値をのせることにする．また，$c_{i-1+}=1$ または $c_{i-1-}=0$ のときに0の値をのせることにする．中間和信号 v_i には，中間和 $m_{i-}=\bar{1}$ または $m_{i+}=0$ のときに1の値をのせることにする．また，中間和 $m_{i-}=0$ または $m_{i+}=1$ のときに0の値をのせることにする．ここで，中間和 $m_i=(m_{i+}, m_{i-})$ については，まず，その絶対値を求め，1桁下位の被加数 b_{i-1} と加数 d_{i-1} の {{両方とも非1である} ことはない}，すなわち，{どちらかが1である} を符号信号 p_{i-1} として使用することができる．以上のようにして信号の意味の真理値を決めることによって，表4に示すように中間和信号値 v_i と桁上げ入力信号値 q_{i-1} から i 桁目の結果 $r_i=(r_{i-}, r_{i+})=\{m_{i-}+c_{i-1+}, m_{i+}+c_{i-1-}\}$ を一意に求めることができる．そして，桁上げ出力信号 q_i と中間和信号 v_i について，これまで説明してきた信号の意味の真理値に留意して，表3の演算規則表からそのまま論理式を導出することによって，図3に示すように冗長2進加算器Bの1桁分の論理回路を実現することができる．以上説明した性質は，明らかに従来文献で採用されている演算規則についても規則表の上下に対称性をもつので成立する．

表4 $m_{i-}+c_{i-1+}$ と $m_{i+}+c_{i-1-}$ の重ねあわせ
Table 4 Overlap of $m_{i-}+c_{i-1+}$ and $m_{i+}+c_{i-1-}$.

			桁上げ入力信号 q_{i-1}	
			1	0
			$c_{i-1+}=0$ $c_{i-1-}=\bar{1}$	$c_{i-1+}=1$ $c_{i-1-}=0$
中間和信号 v_i	1	$m_{i-}=\bar{1} \wedge p_{i-1}=1$ $m_{i+}=0 \wedge p_{i-1}=0$	$(m_{i-}=\bar{1}, c_{i-1+}=0)$ $(m_{i+}=0, c_{i-1-}=\bar{1})$	$(m_{i-}=\bar{1}, c_{i-1+}=1)$ $(m_{i+}=0, c_{i-1-}=0)$
	0	$m_{i-}=0 \wedge p_{i-1}=1$ $m_{i+}=1 \wedge p_{i-1}=0$	$(m_{i-}=0, c_{i-1+}=0)$ $(m_{i+}=1, c_{i-1-}=\bar{1})$	$(m_{i-}=0, c_{i-1+}=1)$ $(m_{i+}=1, c_{i-1-}=0)$

3. 繰返し乗算方式

3.1 冗長2進数リコーダと繰返し乗算方式

一般に，多項式

$$A_m \cdot X^m + A_{m-1} \cdot X^{m-1} + \cdots + A_1 \cdot X + A_0 \qquad (1)$$

は，

$$((\cdots(A_m \cdot X + A_{m-1}) \cdot X + \cdots) \cdot X + A_1) \cdot X + A_0 \qquad (2)$$

のようにネストするかたちにして計算すると乗算回数が少なくなるので効率がよい．これは Horner の方法としてよく知られている．本章では，このように，最初に $Y_m=A_m$ と置いて，$Y_{i-1}=Y_i \cdot X + A_{i-1}$ のかたちをした乗算と加算を $i=m$ から1まで繰り返して行う繰返し乗算方式について検討する．高木・矢島[16]は，桁上げ保存形で得られた部分積の中間値（中間積）をリコードする方式を提案している．われわれは前章で指摘した冗長2進加算器が4次相当の Booth アルゴリズムを容易に実現できるという理由から，冗長2進数表現のまま乗算を行う繰返し乗算方式を提案する．この場合，被乗数 Y_i と乗数 X のどちらをリコードすると都合がよいかというと，どちらにも問題がある．乗数 X のほうをリコードすると，被乗数 Y_i は冗長2進数表現で得られているので，4次の Booth 相当のリコードが使えず，2次のリコードに縮退させなければならない．そうすると，冗長2進加算器Bの数を倍にしなければならない．逆に被乗数 Y_i のほうはそのままではリコードできない．そこで，被乗数 Y_i を4桁（=2桁×2）単位にリコードするために，任意の隣接する5桁 $y_0y_1y_{-1}y_{-2}y_{-3}$ に対して2桁 y_0y_1 の下位3桁 $y_{-1}y_{-2}y_{-3}$ を**表5**に示すように，あらかじめ等価な $y_{-1}'y_{-2}'y_{-3}'$ へ書換え変換を行う．そして，**表6**に示すように，書換え値 y_{-1}' にもとづいて表1の2次の Booth リコードに類似した方法で2桁単位の冗長2進数リコードを行う．4次相当の冗長2進数リコードは Booth のときと同様に，2桁単位のリコーダを2組設けることで実現できる．表5に示した書換え変換をする理由は，表6の規則にもとづいてリコードしたときに，桁上げ伝播が生じないようにあらかじめ予測しておくためである．そのため，もとの値を変更することはしない．また，必要な出力は y_{-1}' のみであって，$y_{-2}'y_{-3}'$ は利用しないので特に出力する必要はない．例えば，$y_1y_0y_{-1}y_{-2}y_{-3}=10\bar{1}01$ の場合，もし，$y_{-1}y_{-2}y_{-3}$ が書換え変換されないとすると，$y_{-1}=\bar{1}$ は $\bar{1}$ を y_0 の桁へ桁上げする．さらに，$y_1=1$ は1を上位に桁上げし，y_1 の桁は $\bar{1}$ になる．その結果，$y_1'y_0'=\bar{1}\bar{1}$ になり，非ゼロが連続しないようにリコードすることから桁上げ伝播が生じてしまう．そこで，あらかじめ書換え変換を行い，$y_{-1}y_{-2}y_{-3}=\bar{1}01$ は数値上は等価な $y_{-1}'y_{-2}'y_{-3}'=0\bar{1}\bar{1}$ へ変換し，y_0 より上位の桁への桁上げが生じないようにする．このように，表5に示したあらかじめの書換え変換によって，リ

表 5 冗長2進数の3桁 $y_{-1}y_{-2}y_{-3}$ の書換え変換
Table 5 Rewriting of three figures $y_{-1}y_{-2}y_{-3}$ in redundant binary.

入力	出力	入力	出力	入力	出力
$y_{-1}y_{-2}y_{-3}$	$y_{-1}'y_{-2}'y_{-3}'$	$y_{-1}y_{-2}y_{-3}$	$y_{-1}'y_{-2}'y_{-3}'$	$y_{-1}y_{-2}y_{-3}$	$y_{-1}'y_{-2}'y_{-3}'$
$\bar{1}\ \bar{1}\ \bar{1}$	$1\ \bar{1}\ \bar{1}$	$0\ \bar{1}\ \bar{1}$	$0\ 1\ \bar{1}$	$1\ \bar{1}\ \bar{1}$	$0\ 1\ \bar{1}$
$\bar{1}\ \bar{1}\ 0$	$1\ \bar{1}\ 0$	$0\ \bar{1}\ 0$	$0\ 1\ 0$	$1\ \bar{1}\ 0$	$0\ 1\ 0$
$\bar{1}\ \bar{1}\ 1$	$1\ \bar{1}\ 1$	$0\ \bar{1}\ 1$	$0\ 1\ 1$	$1\ \bar{1}\ 1$	$0\ 1\ 1$
$\bar{1}\ 0\ \bar{1}$	$1\ 0\ \bar{1}$	$0\ 0\ \bar{1}$	$0\ 0\ \bar{1}$	$1\ 0\ \bar{1}$	$0\ 1\ 1$
$\bar{1}\ 0\ 0$	$1\ 0\ 0$	$0\ 0\ 0$	$0\ 0\ 0$	$1\ 0\ 0$	$1\ 0\ 0$
$\bar{1}\ 0\ 1$	$0\ \bar{1}\ \bar{1}$	$0\ 0\ 1$	$0\ 0\ 1$	$1\ 0\ 1$	$1\ 0\ 1$
$\bar{1}\ 1\ \bar{1}$	$0\ \bar{1}\ \bar{1}$	$0\ 1\ \bar{1}$	$0\ 1\ \bar{1}$	$1\ 1\ \bar{1}$	$1\ 1\ \bar{1}$
$\bar{1}\ 1\ 0$	$0\ \bar{1}\ 0$	$0\ 1\ 0$	$0\ 1\ 0$	$1\ 1\ 0$	$1\ 1\ 0$
$\bar{1}\ 1\ 1$	$0\ \bar{1}\ 1$	$0\ 1\ 1$	$0\ 1\ 1$	$1\ 1\ 1$	$1\ 1\ 1$

表 6 冗長2進数リコード規則
Table 6 Redundant binary recode rule.

$y_0y_1y_{-1}'$	$y_0'y_1'$	$y_0y_1y_{-1}'$	$y_0'y_1'$	$y_0y_1y_{-1}'$	$y_0'y_1'$
$\bar{1}\ 1\ \bar{1}$	0 0	$0\ \bar{1}\ \bar{1}$	$\bar{1}\ 0$	$1\ 1\ \bar{1}$	0 0
$\bar{1}\ 1\ 0$	0 1	$0\ \bar{1}\ 0$	$0\ \bar{1}$	$1\ \bar{1}\ 0$	0 1
$\bar{1}\ 1\ 1$	1 0	$0\ \bar{1}\ 1$	0 0	$1\ \bar{1}\ 1$	1 0
$\bar{1}\ 0\ \bar{1}$	0 1	$0\ 0\ \bar{1}$	$0\ \bar{1}$	$1\ 0\ \bar{1}$	0 1
$\bar{1}\ 0\ 0$	1 0	$0\ 0\ 0$	0 0	$1\ 0\ 0$	$\bar{1}\ 0$
$\bar{1}\ 0\ 1$	$0\ \bar{1}$	$0\ 0\ 1$	0 1	$1\ 0\ 1$	$0\ \bar{1}$
$\bar{1}\ 1\ \bar{1}$	$\bar{1}\ 0$	$0\ 1\ \bar{1}$	0 0	$1\ 1\ \bar{1}$	$\bar{1}\ 0$
$\bar{1}\ 1\ 0$	$0\ \bar{1}$	$0\ 1\ 0$	0 1	$1\ 1\ 0$	$0\ \bar{1}$
$\bar{1}\ 1\ 1$	0 0	$0\ 1\ 1$	1 0	$1\ 1\ 1$	0 0

コードのときに桁上げ伝播が生じないことを保証できる．そして，表6の冗長2進数リコード規則の論理に従って，$y_1y_0y_{-1}'$ を非ゼロが連続しないようにリコードし，$y_1'y_0'$ を出力する．ただし，Booth のときと同様に出力値 $y_1'y_0'$ は，符号信号 s_1 と絶対値信号 z_1z_0 にして出力信号化する．表6の冗長2進数リコード規則は，1表現部のみを考えれば，表1の2次の Booth リコード規則そのものであるため，ある意味で拡張になっている．そして，その出力値 $s_1z_1z_0$ による制御は，前章で説明した2次の Booth 方式とまったく同じである．冗長2進数リコードの具体的な論理回路化については次節で説明する．

そこで以下に，第1ステップで，$Y \leftarrow A_m$ にセットし，各ステップで，$Y \leftarrow Y \cdot X + A$ のかたちをした演算を冗長2進数体系で一貫して行う繰返し乗算方式の詳細について**図4**を用いて説明する．ここで，A には i ステップで係数値 A_{m-i} をセットする．乗数 X は X レジスタに，被乗数 Y は冗長2進数で表現されているので，Y_+ レジスタと Y_- レジスタに，係数値 A は加数の A レジスタに格納される．冗長2進数リコーダは，(Y_+, Y_-) レジスタの内容各4桁 $(y_3y_2,\ y_1y_0)$ 単位に設けられて，それぞれ出力信号 $(s_2z_3z_2,\ s_1z_1z_0)$ を出す．そして，それらの各出力信号に従って，前章で説明した乗算器がまったく同様の方法で制御される．被乗数 Y の桁数が $2n$ 桁ならば，n 個の冗長2進数リコーダが設けられる．$Y \cdot X$ の乗算結果 $R=(R_-, R_+)$ は係数値 A と冗長2進加算器 A を使って冗長2進加算される．被加数 (R_-, R_+) は冗長2進数であるが，加数である係数 A は通常の2進数なので，前章で説明した冗長2進加算器B規模のものは必要がなく，それよりも簡単な冗長2進加算器Aを採用する．その演算規則を**表7**に示す．表中の演算結果は，c_i が桁上げ値で，m_i が中間値である．例えば，加数 0，被加数1のとき，桁上げ値は1で，中間値は $\bar{1}$ である．また，加数 1，被加数 $\bar{1}$ のとき，桁上げ値は0で，中間値は0である．これを回路化すると，**図5**に示すようなものになる．被加数が $\bar{1}$ でないときに，被加数か加数のどちらかが1ならば，桁上げ c_i が生じるという非常に簡単な規則である．中間和も，被加数の絶対値と加数の排他的論理和が1ならば，$\bar{1}$ になるという非常に簡単な規則である．したがって，中間和値 $\{0, \bar{1}\}$ と桁上げ入力値 $c_{i-1}\{0, 1\}$ の和 $(y_{i-},$

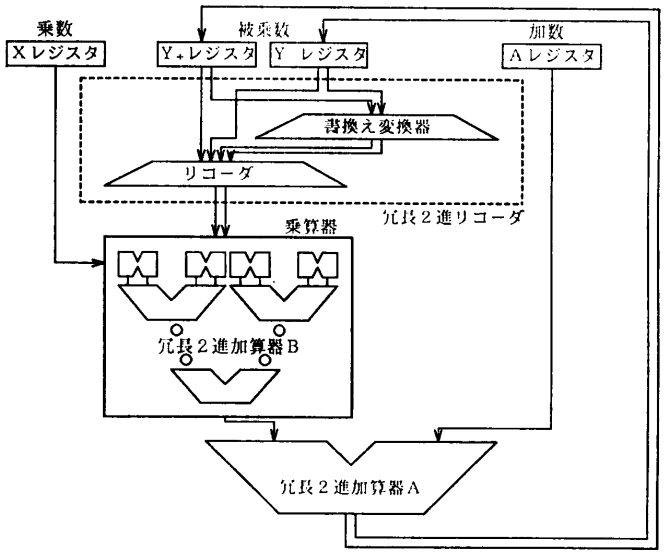

図 4 繰返し乗算方式の構成スキーマ
Fig. 4 Construction schema for iterative multiplication method.

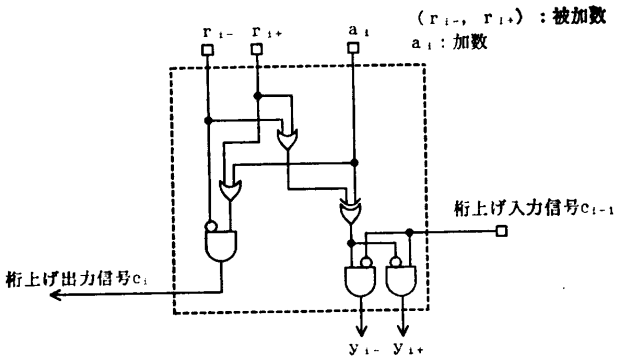

図 5 冗長2進加算回路A
Fig. 5 Redundant binary adder circuit A.

表 7 冗長2進加算器 A の演算規則
Table 7 Computation rule for redundant binary adder A.

		被加数					
		0		1		$\bar{1}$	
		c_i	m_i	c_i	m_i	c_i	m_i
加数	0	0	0	1	$\bar{1}$	0	$\bar{1}$
	1	1	$\bar{1}$	1	0	0	0

y_{i+}) も簡単に求まる．冗長2進加算器Aによって加算された結果の冗長2進数出力値は被乗数の (Y_+, Y_-) レジスタへ送られる．そして，加数のAレジスタには，次の係数値が入力される．このようにして，$Y \leftarrow Y \cdot X + A$ というかたちで m 回の繰返し乗算が冗長2進数表現によって次次と実行されていく．そして，最終結果は通常の ALU (Arithmetic Logic Unit) によって通常の2進数に変換されて，多項式の計算は終了する．

3.2 冗長2進数リコードの論理回路化

冗長2進数リコードを具体的に論理回路化するために，被乗数 Y の任意の隣接する2桁 y_1y_0 とその下位3桁 $y_{-1}y_{-2}y_{-3}$ について，まず，表5の3桁 $y_{-1}y_{-2}y_{-3}$ の書換え変換規則によって y_{-1}' へ変換する論理式を求める．そのために，**図6**に示すように2値のカルノー図を3値に拡張する．ただし，3値にすると囲う部分が必ずしも連続するとは限らないので，そのかわりに同じもの同士をまとめるために網かけ表示を使用することにした．また，3値論理を2値化するときの容易性から判断して，$\bar{1}$ あるいは1でまとめるよりも，網かけで囲った0でまとめることにした．そうすると，

$$y_{-1}'_+ = y_{-1+} \wedge \neg\,(y_{-2-} \vee \neg y_{-2+} \wedge y_{-3-}),$$
$$y_{-1}'_- = y_{-1-} \wedge \neg\,(y_{-2+} \vee \neg y_{-2-} \wedge y_{-3+}) \quad (3)$$

のように2値化できる．ここで，$\wedge$は論理積，$\vee$は論理和，$\neg$は論理否定であることを示す．そして，式(3)を論理回路図化した書換え変換回路を**図7**に示す．

同様の考え方で，表6の冗長2進数リコード規則を**図8**に示すような3値変形カルノー図を作って論理式を求める．ただし，出力を符号と絶対値に信号化したいので，それぞれ別々に考える．まず，出力の絶対値を求めることから，図8に実線の矢印で示すように入力 y_{1+} と y_{1-} について絶対値化することができる．出力 y_1' については，さらに入力 y_0 と y_{-1}' の $\bar{1}$ 表現と1表現がぶつかる場合にはゼロ化する必要がある．このようにして，出力 y_1' と y_0' のそれぞれの絶対値信号 z_1 と z_0 を論理式化すると，

$$z_1 = \neg[\neg\{(y_{1-} \vee y_{1+}) \oplus (y_{0-} \oplus y_{-1}'_+ \vee y_{0+} \oplus y_{-1}'_-)\} \vee (y_{0-} \vee y_{0+}) \oplus (y_{-1}'_- \vee y_{-1}'_+)],$$
$$z_0 = (y_{0-} \vee y_{0+}) \oplus (y_{-1}'_- \vee y_{-1}'_+) \quad (4)$$

ここで，$\oplus$は排他的論理和を示す．次に，出力 y_1' と y_0' の共通の符号信号 s_1 を求めるために，破線で囲った部分を区別に利用して論理式化する．すなわち，

y_{-1}	$y_{-2}y_{-3}$									
	$\bar{1}\bar{1}$	$\bar{1}0$	$\bar{1}1$	$0\bar{1}$	00	01	$1\bar{1}$	10	11	
$\bar{1}$	$\bar{1}$	$\bar{1}$	$\bar{1}$	$\bar{1}$	$\bar{1}$	0	0	0	0	
0	0	0	0	0	0	0	0	0	0	y_{-1}'
1	0	0	0	0	1	1	1	1	1	

図 6 書換え変換論理の3値変形カルノー図
Fig. 6 Three-valued modified Karnaugh map for rewriting logic.

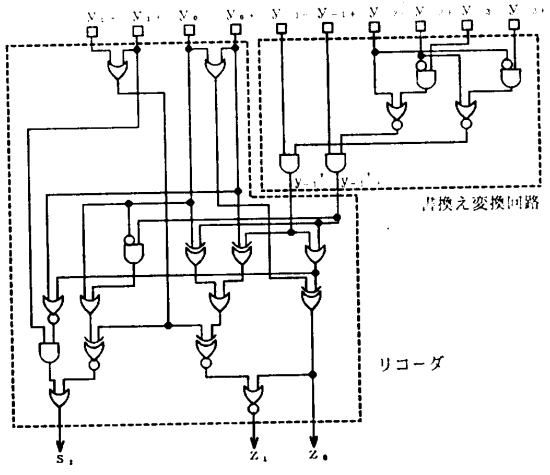

図 7 冗長2進数リコーダ
Fig. 7 Redundant binary recoder.

$$s_1=[y_{1+}\wedge\neg\{(y_{0-}\vee y_{0+})\vee(y_{-1}'-\vee y_{-1}'+)\}]\vee\neg[\{y_{0+}\vee(\neg y_{0-}\wedge y_{-1}'+)\}\oplus\{y_{1-}\vee y_{1+}\}] \quad (5)$$

のようにまとめることができる．以上のようにして，表6のリコード規則を論理式化することができたので，それを論理回路化して図7にリコーダ部として示した．そして，図4において，書換え変換回路部とリコーダ部を合わせて冗長2進数リコーダを構成することができる．この中には，図1に示した Booth リコーダに対応する部分を含んでいることから，冗長2進数リコーダは論理規模と段数が Booth リコーダの2倍を超えて増加していることがわかる．しかし，ゲート遅延時間は，一定に押さえられるために，桁上げ伝播時間が桁数の $\log_2$ に比例する ALU を使って通常2進数に変換して Booth リコーダを使用するよりも，直接冗長2進数リコーダを使用するほうがゲート遅延時間を少なくすることができる．次節でその詳細な評価を行う．

3.3 繰返し乗算方式の評価

繰返し乗算方式について，表8に示すように，本提案方式，高木方式[16]および ALU を使って通常2進数に変換して Booth リコーダを使用する場合のゲート段数（繰返しの1回分）とトランジスタ規模の見積りを処理ビット数32と64について行った．ただし，最近は，様々な回路技術が提案されており，その具体的な評価方法は各回路技術によって微妙に異なるので，ここでは回路技術にあまりこだわらないで2入力の論理積と論理和またはそれに相当するものを段数の基本単位として大まかな評価をするのに止めた．また，トランジスタ数は標準的な CMOS 回路を例にとり，1単位4トランジスタで数えた．本提案方式については，図4の構成図をもとにして，高木方式については，文献16)に示された構成図と規則をもとに図4相当のものをわれわれが設計して評価した．両方式についてそれぞれ，ALU を使って通常2進数に変換して Booth リコーダを使用する場合は，†部の数値が括弧内に示した2次の Booth リコーダと桁上げ先見加算器のものに置き換わる．ただし，トランジスタ数については，桁上げ先見加算器は ALU に含まれているということで，見積りの中には含めず，そのかわりにYレジスタの規模が半減するので，X, Y, A レジスタのトランジスタ数を示すことにした．また，ゲート段数の評価の中には，Yレジスタへのラッチ分を含めた．

表 8 繰返し乗算方式のゲート段数とトランジスタ規模の見積り
Table 8 Gate steps and number of transistors in iterative multiplication methods.

	構成要素 ＼ 処理ビット数	ゲート段数 32	ゲート段数 64	トランジスタ数 32	トランジスタ数 64
本提案方式	乗算器（冗長2進加算器B）	23	29	25,480	108,296
	冗長2進加算器A	4	4	1,792	3,584
	冗長2進数リコーダ	9†	9†	1,644†	3,288†
	（2次のBoothリコーダ）	(3)	(3)	(320)	(640)
	（桁上げ先見加算器）	(12)	(14)	(1,280)	(2,560)
	X, Y, Aレジスタ	2	2	1,920†	3,840†
	合　計	38	44	30,836	119,008
		(44)	(52)	(28,872)	(115,080)
高木方式	乗算器（桁上げ保存加算器）	28	36	25,096	107,528
	桁上げ保存加算器	4	4	1,792	3,584
	桁上げ保存リコーダ	9†	9†	3,584†	7,168†
	（2次のBoothリコーダ）	(3)	(3)	(320)	(640)
	（桁上げ先見加算器）	(12)	(14)	(1,280)	(2,560)
	X, Y, Aレジスタ	2	2	1,920†	3,840†
	合　計	43	51	32,392	122,120
		(49)	(59)	(28,488)	(114,312)

以上の基準による評価によって，トランジスタ規模は冗長2進数リコーダまたは桁上げ保存リコーダ（高木方式のリコーダ）部とYレジスタの倍増分が 3〜14% 程度加わるが，ゲート段数については，本提案方式，高木方式，そして両方式の ALU を使って通常2進数に変換して Booth リコーダを使用する場合の順にすぐれているので，冗長2進数リコーダを設ける本提案方式が性能的に最もすぐれていることがわかる．本提案方式は従来方式に比べて 15〜34% 程度の性能改善を実現している．

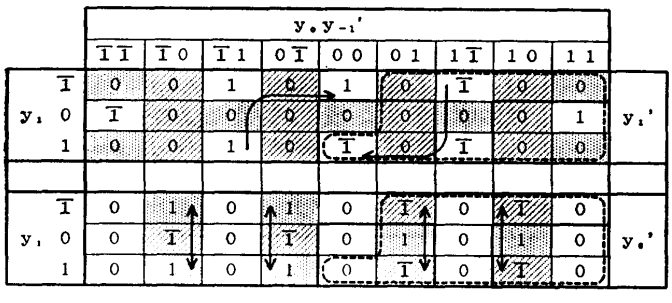

図 8 リコード論理の3値変形カルノー図
Fig. 8 Three-valued modified Karnaugh map for recode logic.

4. おわりに

乗算器の構成において，部分積の加算に冗長2進加算器を使用すれば，4次の Booth アルゴリズム相当のものが自然に効率よく実現できることを説明した．そして，多項式計算などのように繰返し乗算を行う場合に，冗長2進数表現で得られている被乗数を4桁単位にリコードする冗長2進数リコーダを考案し，前述の冗長2進加算器を使用した乗算器を制御する繰返し乗算方式を提案した．従来方式との性能比較により，本提案方式が，15～34% 程度の改善を実現し，性能的に最も優位であることを明らかにした．多項式表現は曲線や曲面の表現に現れるため，本提案方式は，特にグラフィックス分野への応用に有効性を発揮する．

謝辞 日頃の研究にご理解をいただいている当社中央研究所西向井忠彦主任研究員ならびに助言をいただいた同研究所木内淳氏に感謝します．

参考文献

1) Roe, S.: 3D Graphics Gets Boost from Double-Precision Processing, *Electronic Design*, Vol. 36, No. 17, pp. 115-118 (1988).
2) Hwang, K. (堀越 彌監訳): コンピュータの高速演算方式, pp. 129-162, 近代科学社 (1980).
3) 高木直史，矢島脩三：算術演算のハードウェアアルゴリズム，情報処理，Vol. 26, No. 6, pp. 632-639 (1985).
4) Avizienis, A.: Signed Digit Number Representations for Fast Parallel Arithmetic, *IRE Trans. Electronic Comput.*, Vol. EC-10, No. 9, pp. 389-400 (1961).
5) 高木直史，安浦寛人，矢島脩三：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 (1983).
6) Takagi, N. et al.: High-Speed VLSI Multiplication Algorithm with a Redundant Binary Addition Tree, *IEEE Trans. Comput.*, Vol. C-34, No. 9, pp. 789-796 (1985).
7) 高木直史，安浦寛人，矢島脩三：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 (1984).
8) Harata, Y. et al.: High Speed Multiplier Using a Redundant Binary Adder Tree, *Proc. of ICCD '84*, pp. 165-170 (1984).
9) 原田義久，服部佳晋，長瀬 宏，瀧川光治：冗長2進加算セルの回路構成の検討，電子情報通信学会全国大会予稿集，p. 2-82 (1987).
10) Nakanishi, T. et al.: CMOS Radix-2 Signed-Digit Adder by Binary Code Representation, *The Trans. of the IECE of Japan*, Vol. E69, No. 4, pp. 261-263 (1986).
11) 仲西 正：2進冗長 SD コードの2値符号化方式，公開特許公報(A)，昭 62-204332 (1987).
12) Kuninobu, S. et al.: Design of High Speed MOS Multiplier and Divider Using Redundant Binary Representation, *IEEE International Symposium on Computer Arithmetic*, pp. 80-86 (1987).
13) 谷口隆志，枝末壽一，西山 保，國信茂郎：冗長2進表現を用いた高速浮動小数点プロセッサ，信学技報，ICD 87-100, pp. 43-48 (1988).
14) 高木直史，西山 保，國信茂郎：演算処理装置，公開特許公報 (A)，昭 63-25728-25729，昭 63-49835-49836 (1988).
15) Edamatsu, H. et al.: A 33MFlops Floating Point Processor Using Redundant Binary Representation, *Proc. of ISSCC88*, pp. 152-153 (1988).
16) 高木直史，矢島脩三：中間積のリコードによる繰り返し乗算の高速化について，第 36 回情報処理学会全国大会論文集，(I)4C-1, pp. 213-214 (1988).

(平成元年8月24日受付)
(平成2年4月17日採録)

外村 元伸 (正会員)

1952 年生．1979 年名古屋大学大学院工学研究科情報工学専攻博士課程前期課程修了．同年，(株)日立製作所中央研究所入社．制御用ミニコンピュータ，3次元グラフィック・プロセッサ，GMICRO/200 の開発に従事．現在，ULSI プロセッサ，特に演算・制御の論理・回路方式などの研究に従事．IEEE，日本応用数理学会各会員．