システムＬＳＩ設計技術　108－12
(2003. 1. 29)

冗長２進加算における中間和演算を省略した高速冗長２進乗算器の提案

仲里　常智†　　島尻　寛之†　　吉田たけお†

† 琉球大学工学部情報工学科　〒903-0213 沖縄県中頭郡西原町字千原１番地
E-mail: †{tune,shimajiri,tyoshida}@fts.ie.u-ryukyu.ac.jp

あらまし　乗算器の高速化手法として冗長２進表現を用いる手法は，VLSI化に適しているという特徴を持つため，近年注目を集めている．本稿では，冗長２進表現を用いた乗算器(冗長２進乗算器)の高速化手法として，２次のBooth法との組み合わせを考慮した手法について検討する．Booth法を適用した冗長２進乗算器では，乗数２桁毎に１つの部分積を生成し，それらの加算を繰り返す．そこで本稿では，Booth法を適用した冗長２進乗算器を高速化するために，部分積の加算を高速に行える冗長２進表現について検討する．提案手法を適用した配列型冗長２進乗算器の評価を行った結果，通常の冗長２進乗算器に比べ遅延時間が約53%，回路面積が約56%それぞれ削減できることが分かった．
キーワード　冗長２進表現，冗長２進乗算器，Booth法，部分積，中間和演算

A Fast Redundant Binary Multiplier Excluding the Intermediate Sum Operation of Redundant Binary Adders

Tsunetomo NAKAZATO†, Hiroyuki SHIMAJIRI†, and Takeo YOSHIDA†

† Department of Information Engineering, University of the Ryukyus.
E-mail: †{tune,shimajiri,tyoshida}@fts.ie.u-ryukyu.ac.jp

Abstract　Accelerating methods that use a redundant binary representation for multipliers are good for VLSI implementation and have been concentrated lately. We examine an accelerating method for redundant binary multipliers using second order Booth's method. At a redundant binary multiplier which Booth's method is applied to, partial products are generated by every two digit of a multiplier and are accumulated. In this paper, we consider a redundant binary representation that makes additions of partial products fast, in order to accelerate a redundant binary multiplier which Booth's method is applied to. We evaluate delay time and circuit area of proposed redundant binary array multiplier. As a result, delay time and circuit area are reduced about 53% and 56%, respectively.
Key words　Redundant Binary Representation, Redundant Binary Multiplier, Booth's Method, Partial Product, Intermediate Sum Operation

1. はじめに

乗算器は，計算機の基本構成要素の１つであり，その性能は計算機そのものの性能に大きな影響を及ぼす．このため，乗算器の様々な高速化手法が提案されてきた[1]．乗算は，部分積の生成と部分積の加算によって行われるため，乗算器の高速化手法は，生成される部分積の数を削減する手法と部分積の加算を高速化する手法とに大別される．

これまでに提案されてきた乗算器の高速化手法は，部分積の加算の高速化を目的としたものが多く，桁上げ保存加算や冗長２進表現を用いる手法などがある[2]～[5]．特に，冗長２進表現[6]を用いる手法は，レイアウトが規則正しくVLSI化に適しているという特徴を持つため，近年注目を集めている．本稿では，この冗長２進表現を用いた乗算器(冗長２進乗算器)の高速化手法について述べる．

一方，部分積の数を削減する手法としては，Booth法がよく知られている[7],[8]．Booth法は，乗数の表現を変換することによって，生成される部分積の数を削減する手法である．例えば，$i(i \geq 2)$ 次のBooth法を用いた場合，生成される部分積の数を $(1/i)$ 個に削減することができる．さらにBooth法は，先に述べた桁上げ保存加算や冗長２進表現を用いた手法との組み合わせも可能であり，広く用いられている．ただし，Booth法の次数 i を大きくした場合，部分積を生成するための回路が複雑になるため，特に２次のBooth法がよく用いられている．

本稿では，冗長2進乗算器の高速化手法として，2次のBooth法との組み合わせを考慮した手法について検討する．

Booth法を適用した冗長2進乗算は，変換された乗数2桁毎に1つの部分積(シフトされた被乗数)を生成し，それらの加算を繰り返すことによって行われる．本稿では，Booth法との組み合わせを可能とするために，乗数の表現は変更せず，被乗数の表現に着目する．ただし被乗数の表現は，冗長2進表現になっている．そこで本稿では，Booth法を適用した冗長2進乗算器を高速化するために，部分積の加算を高速に行える冗長2進表現について検討する．

本稿の構成は，以下の通りである．まず準備として**2.**で，Booth法と冗長2進表現について簡単に説明する．**3.**では，冗長2進表現の性質について検討し，部分積の加算を高速に行える冗長2進表現を提案する．さらに，提案冗長2進表現への変換法と提案冗長2進表現同士の加算法について述べる．**4.**では，この提案冗長2進表現を用いた冗長2進乗算器の高速化手法を提案する．つづいて**5.**では提案手法とBooth法のハイブリッド手法について説明し，**6.**で提案手法の評価を行う．

2. 準　　備

提案手法を説明するための準備として，Booth法と冗長2進表現について説明する．

2.1 Booth法

ここでは，2次のBooth法について説明する．なお，以下では，2次のBooth法を単にBooth法と呼ぶものとする．Booth法では，乗数を2桁毎に区切り，SD (Signed Digit)表現を用いてどちらか一方の桁が必ず0となるように変換する．これにより，変換後の乗数の表現を用いた乗算では，乗数2桁毎に1つの部分積が生成される．そのため，Booth法を適用しない場合に比べ，部分積の数を約半分にすることができる．なお，この変換はリコードと呼ばれる．表1に，Booth法のリコード規則を示す．2桁毎に区切った乗数を y_{2i+1}, y_{2i} とし，これに1つ下位桁 y_{2i-1} を加えた3桁を用いて，リコード後の表現 y'_{2i+1}, y'_{2i} を得る．またBooth法では，リコードされた表現をもとに，被乗数のシフト操作と正負の反転操作によって部分積を求める．これらの操作を行う回路は，Booth法を適用することによって削減される加算器よりも小さな回路となるため，2次のBooth法を用いることによって，乗算器の遅延時間と回路面積を削減することができる．

2.2 冗長2進表現

一般に，n 桁の数表現の各桁を $a_i \in \{-1, 0, 1\}, i = 0, 1, \cdots, n-1$ とする基数2のSD表現を冗長2進表現という[6]．以下では，-1 を $\bar{1}$ と表す．また，冗長2進表現によって表された数を冗長2進数と呼び，$A = (a_{n-1}a_{n-2}\cdots a_0)_R$ のように表す．ここで，冗長2進数 A の10進数としての値 $V(A)$ は，

$$V(A) = \sum_{i=0}^{n-1} a_i \cdot 2^i \qquad (1)$$

となる．

冗長2進数同士の加算は，加数を $A = (a_{n-1}\ a_{n-2}\cdots a_0)_R$，

表1　Booth法のリコード規則
Table 1　Recode rules of Booth's method.

multiplier			recode digit	
y_{2i+1}	y_{2i}	y_{2i-1}	y'_{2i+1}	y'_{2i}
0	0	0	0	0
0	1	0	0	1
1	0	0	$\bar{1}$	0
1	1	0	0	$\bar{1}$
0	0	1	0	1
0	1	1	1	0
1	0	1	0	$\bar{1}$
1	1	1	0	0

表2　中間桁上げと中間和の計算規則
Table 2　Computation rules of a intermediate carry and a intermediate sum.

a_i	b_i	a_{i-1}, b_{i-1}	c_i	s_i
$\bar{1}$	$\bar{1}$	-	$\bar{1}$	0
0	$\bar{1}$	Neither is Negative	0	$\bar{1}$
$\bar{1}$	0	Otherwise	$\bar{1}$	1
0	0	-	0	0
$\bar{1}$	1			
1	$\bar{1}$			
0	1	Neither is Negative	1	$\bar{1}$
1	0	Otherwise	0	1
1	1	-	1	0

被加数を $B = (b_{n-1}b_{n-2}\cdots b_0)_R$，その和を $Z = (z_n z_{n-1}\cdots z_0)_R$ とした場合，以下の2つのステップにより行われる．

(ステップ1)　$2c_i + s_i = a_i + b_i, i = 0, 1, \cdots, n-1$ を満たすような中間桁上げ $c_i \in \{\bar{1}, 0, 1\}$ および中間和 $s_i \in \{\bar{1}, 0, 1\}$ を決定する．

(ステップ2)　$c_{-1} = s_n = 0$ とおき，ステップ1で求めた中間和 s_j および1つ下位桁からの中間桁上げ c_{j-1} を用いて $z_j = s_j + c_{j-1}, j = 0, 1, \cdots, n$ により，2数 A, B の和 Z を求める．

冗長2進加算では，ステップ1において，中間和 s_i と1つ下位桁からの中間桁上げ c_{i-1} がともに1(または $\bar{1}$)とならないようにするために，a_i と b_i のほかに a_{i-1} と b_{i-1} も調べて s_i と c_i を決定している．これにより，桁上げ伝搬のない加算を実現している．表2に，上記ステップ1の計算規則を示す．

3. 部分積の加算を高速化する冗長2進表現

3.1 冗長2進加算の性質

表2に示す計算規則をもとに，i 桁目の加数 a_i と被加数 b_i に対する冗長2進加算の性質について述べる．

(ケース1)　a_i と b_i の一方のみが0の場合

この場合，桁上げ伝搬を防ぐために a_i, b_i に加え，1つ下位桁の a_{i-1}, b_{i-1} も考慮して c_i と s_i を決定している．4つの値を調べる必要があるため，この操作が冗長2進加算器におけるクリティカルパスとなる．

(ケース2)　ケース1以外の場合

表 3　RC 表現への変換規則

Table 3　Transformation rules for RC representation.

x_{i+1}	x'_i
0	$\bar{1}$
1	1

x_0	c_0
0	$\bar{1}$
1	0

この場合は，1 つ下位桁の a_{i-1}, b_{i-1} とは無関係に a_i, b_i のみによって c_i, s_i を決定している．このとき，必ず $s_i = 0$ となる．したがって，$z_j = c_{j-1}$ となり，加算の結果は 1 つ下位桁からの中間桁上げと等しくなる．

以上より，任意の桁 a_i, b_i が常にケース 2 となる場合，すなわち，加算する値が両方とも非零要素のみの表現となる場合，中間和 s_i を求める必要がないため，冗長 2 進加算器を簡単化でき，部分積の加算を高速に行えると考えられる．そこで以下，各桁に非零要素のみを用いた冗長 2 進表現について検討する．

3.2　非零要素のみを用いた冗長 2 進表現への変換法

まず，符号なし 2 進数 $X = (x_{n-1}\, x_{n-2} \cdots x_0)_2$ から非零要素のみを用いた冗長 2 進表現 X' への変換法について検討する．

いま，$(0 \cdots 01)_2$ のように，0 が 1 つ以上連続しており，これらの連続した 0 の下位桁が 1 である表現について考える．上記の表現が X に含まれている場合には，冗長 2 進表現を用いて $(1\bar{1} \cdots \bar{1})_R$ のように，0 をすべて非零要素のみを用いた表現に変換することができる．しかし，1 つ以上連続した 0 の下位桁が 1 とならない表現が X に含まれているとき，すなわち X の最下位桁 x_0 が 0 の場合，非零要素のみを用いた表現に変換することができない．

そこで $x_0 = 0$ の場合，仮に $x_0 = 1$ とすることにより非零要素のみを用いた表現に変換する．ただし，このままでは X の値が 1 大きくなるため，値の整合性を保持するために補正値 c_0 を用いる．すなわち $x_0 = 0$ の場合，補正値 $c_0 = \bar{1}$ としておき，後でこの補正値を加算することにより値の整合性を保持する．このように，補正値を用いることによって，符号なし 2 進数表現 X を非零要素のみを用いた冗長 2 進表現 X' に変換することができる．

本稿では，補正値を用いた非零要素のみの冗長 2 進表現 X' を RC 表現（Redundant Binary Representation with Correcting Value）と呼び，

$$
\begin{aligned}
&X' = (x'_{n-1}x'_{n-2}\cdots x'_0 c_0)_{RC},\\
&x_i \in \{\bar{1}, 1\}, i = 0, 1, \cdots, n-1,\\
&c_0 \in \{\bar{1}, 0\}
\end{aligned}
\tag{2}
$$

と表す．ここで，RC 表現 X' の 10 進数としての値 $V(X')$ は

$$
V(X') = \sum_{i=0}^{n-1} x'_i \cdot 2^i + c_0 \tag{3}
$$

となる．

表 3 に，符号なし 2 進数 X から RC 表現 X' への変換規則を示す．表 3 からも分かるように，補正値を除いた変換後の各桁は変換前の 1 つ上位桁により決定することができる．なお，

$(38)_{10} + (24)_{10} = (62)_{10}$

$(38)_{10} = (0\,0\,1\,0\,0\,1\,1\,0)_2 \rightarrow (1\,\bar{1}\,\bar{1}\,1\,\bar{1}\,\bar{1}\,1\,\boxed{\bar{1}})_{RC}$

$(24)_{10} = (0\,0\,0\,1\,1\,0\,0\,0)_2 \rightarrow (1\,\bar{1}\,\bar{1}\,\bar{1}\,1\,1\,\bar{1}\,\boxed{\bar{1}})_{RC}$

□ : correcting value

図 1　RC 表現への変換

Fig. 1　A transformation for RC representation.

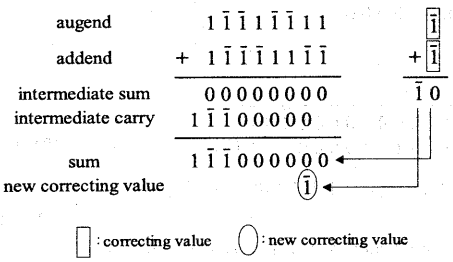

図 2　RC 表現同士の加算

Fig. 2　An Addition of RC representations.

X は正の数であるため，変換後の最上位桁 x'_{n-1} は必ず 1 となるため，便宜上 $x_n = 1$ とする．これにより，表 3 の変換規則を用いて，すべての桁を並列に変換することができる．

3.3　RC 表現同士の加算

ここでは，図 1, 2 の例を用いて，RC 表現同士の加算について説明する．

まず図 1 に示すように，符号なし 2 進数から RC 表現へ変換する．次に，図 2 に示すように，RC 表現における非零要素のみの表現と補正値を別々に加算する．非零要素のみの表現同士の加算では，中間和の値が常に 0 となることから，中間桁上げの値が加算結果となる．また，補正値同士の加算では加算結果が 2 桁となる．ここではその結果を $(c'_1 c'_0)_R$ と表す．補正値 c'_0 については，非零要素のみの表現同士を加算した場合，その最下位桁は常に 0 であるため，この最下位桁に埋めこむことが可能である．一方，c'_1 は c'_0 のように埋めこむことができないため，加算結果に対する補正値として扱う．

RC 表現同士の加算では，最終的に補正値 c'_1 を加算する必要があるため，冗長 2 進加算器の高速化手法として用いることはできない．しかし以下で述べるように，加算を繰り返し行う乗算においては，この補正値の加算時間を隠蔽できる．

4.　RC 表現を用いた冗長 2 進乗算器の高速化

ここでは RC 表現を用いた冗長 2 進乗算において，乗算のアルゴリズムを利用することにより，補正値 c'_1 の加算を隠蔽する方法を示す．なお，乗数と被乗数はそれぞれ n 桁の符号なし 2 進数とする．ただし，符号付き 2 進数への拡張は容易である．

まず比較のために，図 3 に通常の冗長 2 進乗算の例を示す．通常の冗長 2 進乗算では，乗数の桁数が n の場合，生成される部分積の数は n 個となるため，図 3 の例では，8 個の部分積が生成される．

次に，RC 表現を用いた乗算について説明する．RC 表現を

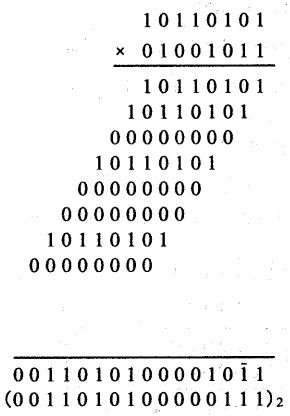

図3　通常の冗長2進乗算の例 (8 ビット)

Fig. 3　An example of Multiplication for normal redundant binary representations (8 bits).

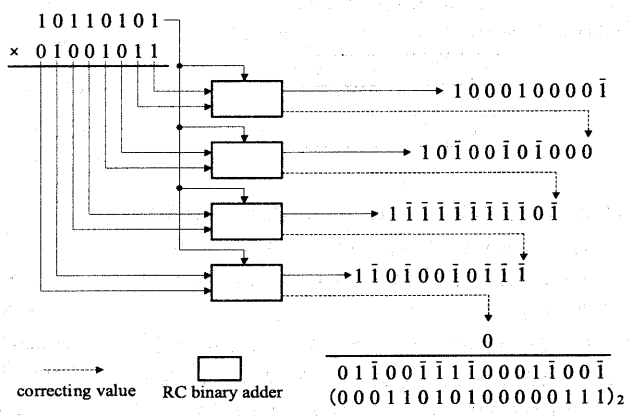

図4　RC 表現を用いた手法の乗算例 (8 ビット)

Fig. 4　An example of Multiplication for RC representations (8 bits).

用いた乗算では，以下の2つのステップによって部分積を生成する．

(ステップ1)　乗数を2桁毎にグループ化する．

(ステップ2)　それぞれのグループ毎に乗数と被乗数をかけた値を RC 表現に変換して加算し，部分積と補正値をそれぞれ1つずつ生成する．なお，このステップを行う回路を RC 加算器と呼ぶものとする．

RC 表現を用いた乗算では，乗数2桁毎に1つの部分積と1つの補正値が生成される．生成された補正値も部分積として扱った場合，部分積の総数は n 個となる．そのため，部分積の数は通常の冗長2進乗算と等しくなり，乗算器の高速化にはならない．ここで，部分積と補正値の桁の位置に注目する．RC 加算器により生成された部分積の最下位桁は，それぞれ $1,3,5,\cdots$ 桁目となる．また補正値は，それぞれ $2,4,6,\cdots$ 桁目となる．そこで図4に示すように，これらの補正値は，1つ上位桁の部分積の最下位桁として埋め込むものとする．これにより，補正値を1つ上位の部分積の一部として扱うことができ，補正値の加算に必要な計算時間を隠蔽することができる．なお，乗数の最上位のグループによって生成された補正値は，どの部分積にも埋めこむことができないため，部分積の1つとして扱う．

ここで，図5に RC 加算器の回路を示す．なお，1桁の冗長2進表現 x を $\{x^-x^+\}$ と2ビットを用いて表し，$\{00\}=(0)_R, \{01\}=(1)_R, \{10\}=(\bar{1})_R$ とする．図5から分かるように，RC 加算器は通常の冗長2進加算器に比べ非常に簡単な構成となる．そのため，Booth 法において部分積を生成する回路と同様に，RC 加算器を乗数2桁毎に1つの部分積を生成する回路として見なすことができる．したがって，RC 表現を用いた冗長2進乗算器では，生成される部分積の数が $\lfloor (n/2) \rfloor + 1$ 個となり，通常の冗長2進乗算器よりも高速に乗算を行うことができると考えられる．ここで，$\lfloor x \rfloor$ は x 以下の最大の整数を表す．

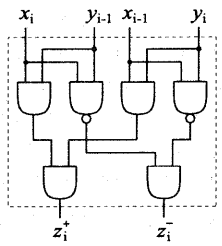

図5　RC 加算器 (1 桁)

Fig. 5　RC adder (1 digit).

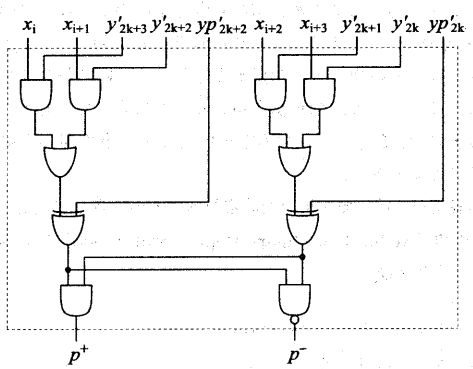

図6　ハイブリッド手法を用いた部分積生成回路 (1 桁)

Fig. 6　A partial product generator for hybrid-method (1 digit).

5. ハイブリッド手法

提案手法は，被乗数の表現に着目した高速化手法であり，乗数の表現は変更していない．このことから提案手法は，Booth 法との組み合わせが可能である．これら2つの手法を組み合わせることによって，さらに冗長2進乗算器を高速化できると考えられる．以下では，RC 表現と Booth 法とのハイブリッド手

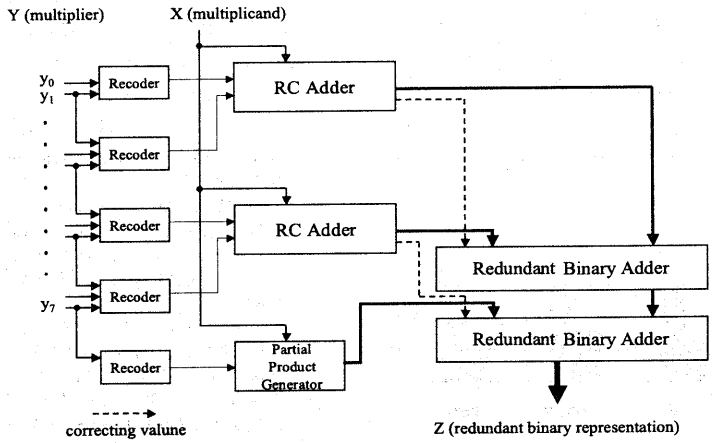

図 7　ハイブリッド手法を用いた配列型冗長 2 進乗算器 (8 ビット)

Fig. 7　A redundant binary array multiplier for hybrid-method (8 bits).

表 4　ハイブリッド手法における乗数のリコード規則

Table 4　Recode rules of multiplier for hybrid-method.

multiplier			recode digit		
y_{2k+1}	y_{2k}	y_{2k-1}	y'_{2k+1}	y'_{2k}	yp'_{2k}
0	0	0	0	0	0
0	1	0	0	1	0
1	0	0	1	0	1
1	1	0	0	1	1
0	0	1	0	1	0
0	1	1	1	0	0
1	0	1	0	1	1
1	1	1	0	0	0

法について説明する．

ハイブリッド手法では，Booth 法によって得られた部分積を RC 表現に変換する．通常の冗長 2 進乗算器に対して Booth 法を適用した場合，乗数 2 桁毎に 1 つの部分積を生成する．これに提案手法を適用することによって，乗数 4 桁毎に 1 つの部分積が生成されるとみなすことができる．

Booth 法を用いた場合，得られる部分積は正の値だけではなく負の値になる場合がある．しかし，RC 表現への変換では，変換前の値を常に正の数としていた．そこで，Booth 法における乗数のリコードを，シフト操作と正負の反転操作に分けて行い，RC 表現に変換する前に正負の反転操作を行わないようにする．これにより，ハイブリッド手法においても表 3 の変換規則をそのまま用いることができる．表 4 に，ハイブリッド手法における乗数のリコード規則を示す．$y'_{2k+1}y'_{2k}, k = 0, 1, \cdots, \lfloor n/2 \rfloor + 1$ はシフト操作に用い，yp'_{2k} は正負の反転に用いる．

ハイブリッド手法では，以下のステップによって部分積を生成する．

(ステップ 1)　表 4 に示すリコード規則を用いて乗数のリコードを行う．

(ステップ 2)　被乗数のシフト操作を行う．

(ステップ 3)　ステップ 2 で求めた値を RC 表現に変換する．

(ステップ 4)　ステップ 3 で求めた RC 表現の正負の反転操作を行う．

(ステップ 5)　ステップ 4 により得られた RC 表現同士の加算を行い，部分積と補正値をそれぞれ 1 つずつ生成する．

ハイブリッド手法においても，乗数の最上位のグループによって生成される補正値以外は，1 つ上位の部分積の中に埋めこむことができる．したがって，ハイブリッド手法により生成される部分積の個数は，乗数の桁数を n とした場合 $\lfloor (n+2)/4 \rfloor + 1$ 個となる．

図 6,7 にそれぞれ，ハイブリッド手法における部分積を生成するための回路とハイブリッド手法を用いた 8 ビットの配列型乗算器の構成を示す．ハイブリッド手法において部分積を生成するためには，図 6 に示す回路と表 4 のリコード規則を実現する回路が必要となる．これらの回路は，冗長 2 進加算器に比べ簡単な回路となっていることから，ハイブリッド手法を用いた冗長 2 進乗算器では，通常の冗長 2 進乗算器よりもさらに高速化することができると考えられる．

6. 評　　価

ここでは，通常の冗長 2 進乗算器，RC 表現を用いた手法，Booth 法，ハイブリッド手法のそれぞれの遅延時間と回路面積の評価を行う．それぞれの手法を適用した配列型乗算器を VHDL を用いて記述し，Synopsys 社の DesignCompiler (Ver.2001.08) を用いて回路の合成を行った．図 8, 9 に，8, 16, 32 ビットの冗長 2 進乗算器の遅延時間と回路面積を示す．なお，RC 表現を用いた手法を RC 手法と略記する．

RC 手法を用いた 32 ビット冗長 2 進乗算器は，通常の冗長 2 進乗算器に比べ，遅延時間が約 44%，回路面積が約 46% 削減できることが分かった．また，Booth 法と比べた場合，回路面積と遅延時間ともにほぼ同程度となっている．ハイブリッド手法を用いた冗長 2 進乗算器は，通常の冗長 2 進乗算器に比べ，32 ビットの場合では，遅延時間が約 53%，回路面積が約 56%

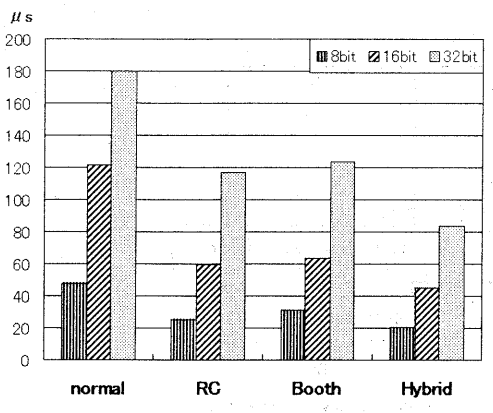

図 8　配列型冗長 2 進乗算器の遅延時間の比較

Fig. 8　Comparison of delay time of redundant binary array multipliers.

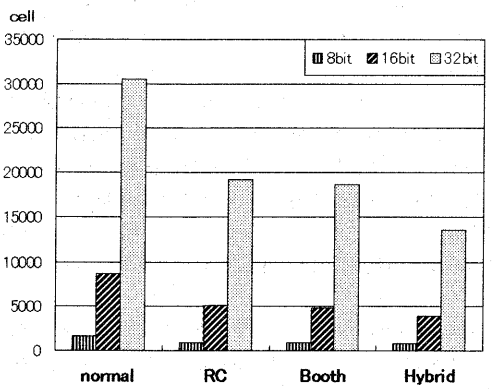

図 9　配列型冗長 2 進乗算器の回路面積の比較

Fig. 9　Comparison of circuit area of redundant binary array multipliers.

削減できることが分かった．さらに，Booth 法に比べ遅延時間が約 33%，回路面積が約 27% 削減でき，RC 手法に比べ遅延時間が約 29%，回路面積が約 29% 削減できることが分かった．

したがって，RC 手法は Booth 法と同等の効果をえることができ，さらに RC 手法と Booth 法を組み合わせることによって，回路面積を削減しつつ冗長 2 進乗算器を高速化できることが分かった．

7. おわりに

今回，Booth 法を適用した冗長 2 進乗算器を高速化するために，部分積の加算を高速に行える冗長 2 進表現の検討を行い，提案冗長 2 進表現を用いた冗長 2 進乗算器の構成を示した．また，提案冗長 2 進表現を用いた手法と Booth 法のハイブリッド手法を適用した冗長 2 進乗算器の構成を示した．

提案手法の評価を行い，冗長 2 進乗算器が高速化できることを示した．また，提案手法と Booth 法との組み合わせにより，さらに冗長 2 進乗算器が高速化できることも示した．

今後の課題として，他の演算器に対して提案冗長 2 進表現を適用することが挙げられる．

文　献

[1] Israel Koren, “Computer Arithmetic Algorithms,” Prantice Hall, 1990.

[2] C. S. WALLACE, “A Suggestion for a Fast Multiplier, ” IEEE Trans. Electronic Computer, pp.14-17, Feb. 1964.

[3] 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, Vol.J66-D, No.6, pp.683-690, 1983.

[4] 恒川 佳隆, 日野杉 充希, 齊藤 正人, 三浦 守, “1 桁 2 ビット/3 ビット混合型高性能冗長 2 進加算器とその乗算器への応用, ” 電学論, Vol.119-C-5, pp.644-653, 1999.

[5] 石井 晋司, 大山 勝一, 山中 喜義, “高速公開鍵暗号プロセッサ, ” 信学論, Vol.J80-D-I, No.8, pp.725-735, 1997.

[6] A.Avizienis, “Signed-digit number representations for fast parallel arithmetic,” IRE Trans. Electronic Computer, pp.389-400, Sept. 1961.

[7] A.D.Booth, “A signed multiplication technique, ” Ouart.J.Mech. and Appl. Math., vol.4, Pt.2, pp.236-240, 1951.

[8] Wen-Chang Yeh and Chein-Wei Jen, “High-Speed Booth Encoded Parallel Multiplier Design, ” IEEE Trans, Computers, vol.49, pp.692-701, Jun. 2000. Ouart.J.Mech. and Appl. Math., vol.4, Pt.2, pp.236-240, 1951.