アルゴリズム 52－2
（1996．7．24）

冗長複素数系に基づく再構成型算術演算回路の構成

天田 博章　　青木 孝文　　樋口 龍雄
(東北大学大学院情報科学研究科)

あらまし — 本稿では複素数演算を高速に実行し，かつ，実数演算器へ実時間で再構成可能な実数／複素数再構成型算術演算回路のハードウェアアルゴリズムについて述べる．提案する再構成型算術演算回路は，(i) 単精度複素数乗算，(ii) 倍精度実数乗算，(iii) 単精度実数 4 入力積和演算 2 並列の 3 つの演算モードを実現する．本稿では，各演算モードで用いる数系の代数的構造に着目して，演算回路の再構成を議論する方法を提案する．また，レイアウトのグリッドモデルを用いた再構成効率を定義し，本手法の有効性を評価する．さらに，実際の回路の構成法についても述べる．

Design of Reconfigurable Arithmetic Circuits Using Redundant Complex Number Systems

Hiroaki AMADA, Takafumi AOKI, and Tatsuo HIGUCHI
(Graduate School of Information Sciences, Tohoku University)

Abstract — This paper presents a hardware algorithm for a real/complex reconfigurable arithmetic unit, which can change it's structure for three arithmetic modes in real time. The three modes realize (i) a single precision complex-number multiplication, (ii) a double precision real-number multiplication, and (iii) a pair of single precision real-number four-input multiply additions, respectively. We discuss the reconfiguration of the hardware structure and its efficiency on the basis of algebraic similarity among number systems used in the three arithmetic modes. This paper also discusses the design of the proposed arithmetic unit using standard binary logic circuits.

1　まえがき

近年，ディジタル信号処理の応用分野の拡大に加えて，とりわけマルチメディア関連産業の出現に伴い，我々に身近な音声，画像，映像（動画像）などの情報表現を媒介にするアプリケーションが重要になってきている．これらの応用分野において要求される演算量は，年々増加の一途をたどっており，このためシグナルプロセッサやマイクロプロセッサへの搭載を前提として，加算器や乗算器に限らず，除算，CORDIC，DCT などを高速に実行するさまざまな算術演算コアが開発されている．その中でも特に，高速フーリエ変換（FFT）に代表される種々の複素直交変換は，音声や画像の認識アルゴリズムの基本操作であり，次世代ディジタル信号処理技術の中で，ますますその重要性が高まるものと考えられる．

これまでに筆者らは，画像のフーリエ変換などにおいて必須になる膨大な複素演算を効率的に行うための数表現として，「冗長複素数系」を提案し，その有効性を実証してきた [1],[2]．本数系では，複素数を実部と虚部に分離することなく，あたかも 1 次元実数データのように扱うことができ，これにより高性能な複素数演算回路を構成することが可能になると考えられる．

本稿では冗長複素数系のこのような優れた特

徴を利用して，複素数演算を高速に行え，かつ，実数演算回路へ実時間で再構成可能な実数／複素数再構成型算術演算回路のハードウェアアルゴリズムについて述べる．

2 冗長複素数系とその加算アルゴリズム

2.1 冗長複素数系 (RCNS)

冗長複素数系 (RCNS : Redundant Complex Number Systems) は基数が $r\boldsymbol{j}$ ($r \geq 2$, $\boldsymbol{j}$: 虚数単位)，各桁が正負対称の冗長性を有した値をとる重み数系である．冗長複素数系の各桁は次の $2\alpha+1$ 個の値をとる．

$$D_\alpha = \{-\alpha, \cdots, -1, 0, 1, \cdots, \alpha\} \quad (1)$$

ここで，α は次のような範囲にある整数である．

$$\left\lceil \frac{r^2-1}{2} \right\rceil \leq \alpha \leq r^2-1 \quad (2)$$

ただし，$\lceil x \rceil$ は x 以上の最小の整数を表す．以下では，2進数との変換が容易である等の理由から，$r=2, \alpha=3$ の場合 (RCNS $2\boldsymbol{j},3$ と表す) について詳しく述べていく．RCNS $2\boldsymbol{j},3$ で表現された複素数 $\boldsymbol{X}=(X_nX_{n-1}\cdots X_0\cdots X_{-k})$ の値は，以下の式で与えられる．

$$\boldsymbol{X} = \sum_{i=-k}^{n} X_i(2\boldsymbol{j})^i \quad (3)$$

上式の右辺は，次式のように実数部と虚数部とに分離することができる．

$$\boldsymbol{X} = \sum_{i=\lceil(-k)/2\rceil}^{\lfloor n/2\rfloor} X_{2i}(-4)^i + 2\boldsymbol{j}\left\{\sum_{i=\lfloor(-k)/2\rfloor}^{\lceil n/2\rceil-1} X_{2i+1}(-4)^i\right\} \quad (4)$$

ここで，$\lfloor x \rfloor$ は x 以下の最大の整数を表す．この式からわかるように，RCNS $2\boldsymbol{j},3$ を実部と虚部とに分離すると，各部は基数 -4 の Signed-Digit 数系 [3] とみなすことができる．このような特徴から，高速かつ規則的な構造をもつ複素数演算器を構成することが可能となる．

	$\boldsymbol{X}$			1	1	3	3	-2	-1	
+)	$\boldsymbol{Y}$			0	1	0	3	3	1	
	$\boldsymbol{Z}$			1	2	3	6	1	0	Step 1
	$\boldsymbol{W}$			1	-2	-1	2	1	0	Step 2
	$\boldsymbol{C}$	0	-1	-1	-1	0	0			
	$\boldsymbol{S}$	0	-1	0	-3	-1	2	1	0	Step 3

$\boldsymbol{X}$ = 3 + 4j　　$\boldsymbol{Y}$ = 5 + 6j　　$\boldsymbol{S}$ = 8 + 10j

図 1: RCNS $2j,3$ の加算

2.2 RCNS $2j,3$ の加算

RCNS $2j,3$ の2数 $\boldsymbol{X}=(X_n\cdots X_i\cdots X_{-k})$ と $\boldsymbol{Y}=(Y_n\cdots Y_i\cdots Y_{-k})$ (ただし，$X_i, Y_i \in \{-3,\cdots,0,\cdots,3\}$) の加算は以下の3ステップで実行される．

$$\text{Step } 1 \quad Z_i = X_i + Y_i \quad (5)$$

$$\text{Step } 2 \quad -4C_i + W_i = Z_i \quad (6)$$

$$\text{Step } 3 \quad S_i = W_i + C_{i-2} \quad (7)$$

ここで，Z_i は線形加算和，W_i は中間和，C_i はキャリーであり，それぞれ次の値をとる．

$$Z_i \in \{-6,\cdots,-1,0,1,\cdots,6\} \quad (8)$$

$$W_i \in \{-2,-1,0,1,2\} \quad (9)$$

$$C_i \in \{-1,0,1\} \quad (10)$$

これらから，最終和 S_i は $\{-3,\cdots,0,\cdots,3\}$ の値をとることがわかる．キャリー C_i は線形加算和 Z_i のみに依存し，他のキャリーによらないので，桁上げ伝搬のない高速な加算が可能となる．図1に加算の例を示す．先程も述べたように，実部と虚部はともに基数 -4 の SD 数とみなすことができるため，キャリーは2桁上に加算され，結果として，実部と虚部が分離して演算されていることになる．

3 数系の変換とハードウェアの再構成

本稿では，各演算モードで使用する数系の代数的構造に着目して，演算器の再構成を議論す

る方法を提案する．

一般に，通常の重み数系は，ディジットセット D，重みベクトル$\boldsymbol{W}$，符号ベクトル $\boldsymbol{\Lambda} = (\lambda_n, \cdots, \lambda_i, \cdots, \lambda_0)$ で表現できる．ここで，λ_i は ± 1 の値をとり，各桁の重みの符号を決定する要素である．また，$\boldsymbol{W}$ の要素は2値論理回路での実現を議論するため，ここでは2のベキとする．これによれば，例えば4ビットの2の補数表現は，次のようにかける．

$$\langle D, \boldsymbol{W}, \boldsymbol{\Lambda} \rangle = \langle \{0,1\}, (2^3, 2^2, 2, 1), (\bar{1}, 1, 1, 1) \rangle \quad (11)$$

ここで，$\bar{1} = -1$ であり，以下ではこの記法を用いる．このままでは複素数を記述できないので，λ_i を $\lambda_i \in \{1, \bar{1}, \boldsymbol{j}, \bar{\boldsymbol{j}}\}$ のように拡張する．

本稿で述べる実数／複素数再構成型演算回路は以下の3つの演算を実現する．

(i) 単精度複素数乗算
(ii) 倍精度実数乗算
(iii) 単精度実数4入力積和演算2並列

まず，(i) では RCNS $2\boldsymbol{j}, 3$ を用いる．この数系は4桁の場合，次のようにかける．

$$\begin{aligned} \langle D_{RCNS} &= \{\bar{3}, \bar{2}, \bar{1}, 0, 1, 2, 3\}, \\ \boldsymbol{W}_{RCNS} &= (2^3, 2^2, 2, 1), \\ \boldsymbol{\Lambda}_{RCNS} &= (\bar{\boldsymbol{j}}, \bar{1}, \boldsymbol{j}, 1) \rangle \end{aligned} \quad (12)$$

実際には2進SD数 (冗長2進数 [4],[5]) に基づく回路で構成するため，この各桁を2進SD数2桁に分解すると，次式となる．

$$\begin{aligned} \langle D_1 &= \{\bar{1}, 0, 1\}, \\ \boldsymbol{W}_1 &= (2^4, 2^3, 2^3, 2^2, 2^2, 2, 2, 1), \\ \boldsymbol{\Lambda}_1 &= (\bar{\boldsymbol{j}}, \bar{\boldsymbol{j}}, \bar{1}, \bar{1}, \boldsymbol{j}, \boldsymbol{j}, 1, 1) \rangle \end{aligned} \quad (13)$$

この数系では，キャリー配線を実部どうし，虚部どうしで2桁飛びに符号反転してつなぐ必要がある．図2(a) に対応する回路構成を示す．

この実部と虚部を同一視し，図2(b) のようにキャリールールを変更して隣へつなぐようにすると (13) 式は

$$\begin{aligned} \langle D_2 &= \{\bar{1}, 0, 1\}, \\ \boldsymbol{W}_2 &= (2^7, 2^6, 2^5, 2^4, 2^3, 2^2, 2, 1), \\ \boldsymbol{\Lambda}_2 &= (\bar{1}, \bar{1}, \bar{1}, \bar{1}, 1, 1, 1, 1) \rangle \end{aligned} \quad (14)$$

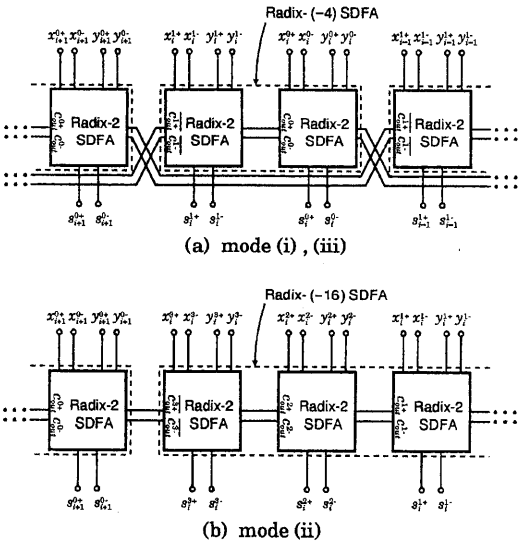

図 2: RCNS $2j, 3$ 加算器

と書き直すことができる．これは -16 進SD数の各桁を2進SD数4桁に分解したものと捉えることもでき，これにより実数演算 (ii) が可能となる．

一方，(13) 式の実部・虚部を別々にみて，それぞれを2つの実数がマージしたものとみなすと，キャリールールを変えずに，

$$\begin{aligned} \langle D_3 &= \{\bar{1}, 0, 1\}, \\ \boldsymbol{W}_3 &= \begin{pmatrix} 2^3, 2^2, & 2, 1 \\ 2^4, 2^3, & 2^2, 2 \end{pmatrix} \quad (15) \\ \boldsymbol{\Lambda}_3 &= \begin{pmatrix} \bar{1}, \bar{1}, & 1, 1 \\ \bar{1}, \bar{1}, & 1, 1 \end{pmatrix} \rangle \quad (16) \end{aligned}$$

と分解できる．この数表現により加算木を2つに分離することができ，(iii) の処理が可能となる．以上から，それぞれの数系は非常に類似性が高いことがわかる．特に，(13) と (15) は $\boldsymbol{W}$ が等しく，$\boldsymbol{\Lambda}$ も符号が一致している．また，(14) は他の2数系と $\boldsymbol{W}$ が異なり，キャリー配線等をつなぎ直す必要があるが，$\boldsymbol{\Lambda}$ の符号が等しいために部分積生成部などはそのまま使え，効率的にハードウェアの再構成が行える．

4 各演算モードにおける演算アルゴリズム

以下，$\langle D, \boldsymbol{W}, \boldsymbol{\Lambda} \rangle$ を用いた各演算モード (i)～(iii) におけるアルゴリズムを述べていく．ここで，入出力は 2 の補数表現を仮定している．

4.1 単精度複素数乗算のアルゴリズム

A. 入力の変換

2 の補数表現の入力を $\langle D_1, \boldsymbol{W}_1, \boldsymbol{\Lambda}_1 \rangle$ の表現形式へ直すアルゴリズムは，被乗数 $\boldsymbol{X}$ と乗数 $\boldsymbol{Y}$ で異なる．被乗数は，実部と虚部を互い違いに配列してから符号調整を施すことにより，$\boldsymbol{W}_1, \boldsymbol{\Lambda}_1$ を用いた表現に変換する．このとき，2 進 SD 数 2 桁で RCNS $2j, 3$ の 1 桁を表していることになる．

乗数の場合はブースのアルゴリズム [6] を用いて，RCNS $2\boldsymbol{j}, 2$ ($= \langle \{\overline{2}, \overline{1}, 0, 1, 2\}, \boldsymbol{W}_{RCNS}, \boldsymbol{\Lambda}_{RCNS} \rangle$) に変換する．このように，ディジットセットの範囲を制限することによって部分積の生成を簡略化することができる．変換の具体例を図 3 に示す．従って，実部・虚部 n ビットの場合，被乗数 $\boldsymbol{X}$ と乗数 $\boldsymbol{Y}$ は次のようにコーディングされる．

$$\boldsymbol{X} = (x^0_{n-1} \cdots x^1_s x^0_s \cdots x^1_0 x^0_0 x^1_{-1}) \quad (17)$$

$$\boldsymbol{Y} = (Y_{n-2} \cdots Y_t \cdots Y_{-1}) \quad (18)$$

ただし，

$$x^1_s, x^0_s \in \{\overline{1}, 0, 1\} \quad (19)$$

$$Y_t \in \{\overline{2}, \overline{1}, 0, 1, 2\} \quad (20)$$

B. 部分積の生成

前述の $\boldsymbol{X}$ と $\boldsymbol{Y}$ により得られる n 個の部分積 $\boldsymbol{P}_{n-2}, \cdots, \boldsymbol{P}_{-1}$ をそれぞれ次のように表す．

$$\begin{aligned} \boldsymbol{P}_t &= \boldsymbol{X} \cdot Y_t \times 2^{2t} \\ &= (p^0_{nt} p^1_{(n-1)t} p^0_{(n-1)t} \cdots p^1_{st} p^0_{st} \\ &\quad \cdots p^1_{0t} p^0_{0t} p^1_{-1t}) \times 2^{2t} \end{aligned} \quad (21)$$

$$p^0_{st}, p^1_{st} \in \{\overline{1}, 0, 1\}, \quad (t = -1, \cdots, n-2)$$

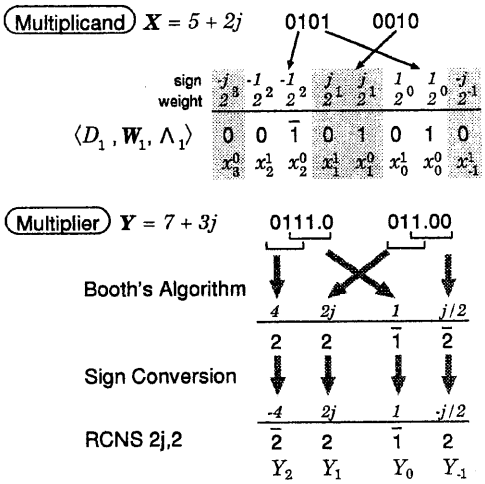

図 3: 2 の補数入力から RCNS $2j, 3$ への変換

このとき，これらの部分積はシフトと符号反転により，次式で生成される．

$$Y_t = 2 \text{ のとき } \quad p^1_{st} = x^0_s, \ p^0_{st} = \overline{x^1_{s-2}} \quad (22)$$

$$Y_t = 1 \text{ のとき } \quad p^1_{st} = x^1_s, \ p^0_{st} = x^0_s \quad (23)$$

$$Y_t = 0 \text{ のとき } \quad p^1_{st} = p^0_{st} = 0 \quad (24)$$

$$Y_t = \overline{1} \text{ のとき } \quad p^1_{st} = \overline{x^1_s}, \ p^0_{st} = \overline{x^0_s} \quad (25)$$

$$Y_t = \overline{2} \text{ のとき } \quad p^1_{st} = \overline{x^0_s}, \ p^0_{st} = x^1_{s-2} \quad (26)$$

このようにして生成された部分積を加算することにより，最終的な乗算結果 $\boldsymbol{Z}$ が得られる．この加算は 2 節で述べたアルゴリズムにより，2 進木構造をとることができる．図 4 に複素乗算の例を示す．

4.2 倍精度実数乗算のアルゴリズム

A. 入力の変換

次に (ii) 倍精度実数乗算モードについて述べる．このモードでは，入力を $\langle D_2, \boldsymbol{W}_2, \boldsymbol{\Lambda}_2 \rangle$ 表現の実数とみなして演算を行う．

まず，被乗数 $\boldsymbol{X}$ は符号調整を行って，$\boldsymbol{W}_2, \boldsymbol{\Lambda}_2$ による表現へ変換する．乗数 $\boldsymbol{Y}$ は先程と同様に，ブースのアルゴリズムと符号調整が施される．複素数乗算の場合と違うのは，並べ換えを

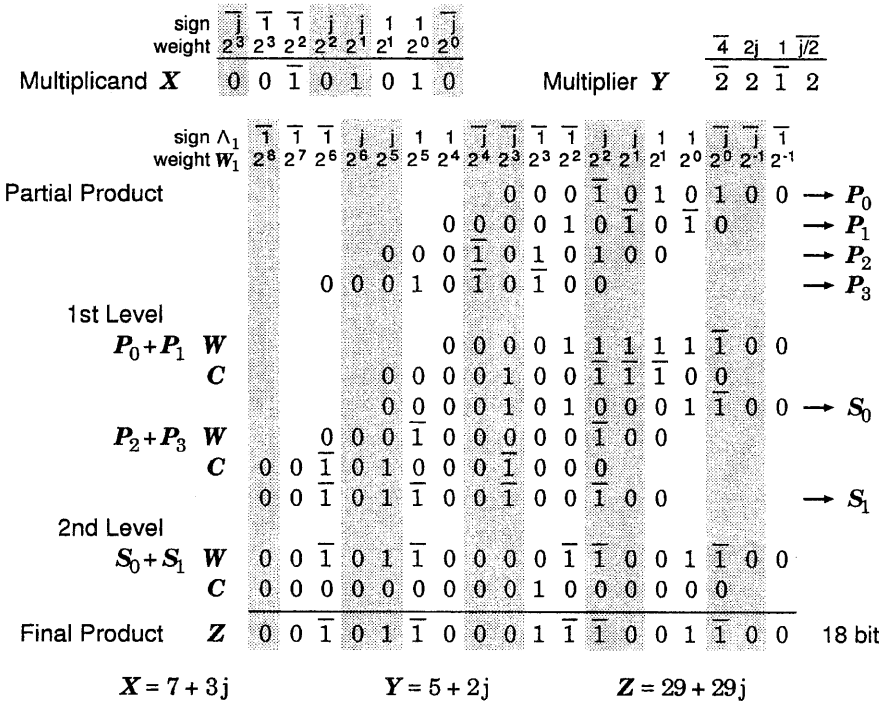

図 4: 複素数乗算の例 (実部・虚部 4 ビット)

行わない点のみである．結局，$2n$ ビットの被乗数 $\boldsymbol{X}$ と乗数 $\boldsymbol{Y}$ は次のように変換される．

$$\boldsymbol{X} = (x_{2n-2}\cdots x_s\cdots x_0 x_{-1}) \quad (27)$$

$$\boldsymbol{Y} = (Y_{n-2}\cdots Y_t\cdots Y_0 Y_{-1}) \quad (28)$$

ただし，

$$x_s \in \{\overline{1}, 0, 1\} \quad (29)$$

$$Y_t \in \{\overline{2}, \overline{1}, 0, 1, 2\} \quad (30)$$

B. 部分積の生成

$\boldsymbol{X}$ と $\boldsymbol{Y}$ により生成される n 個の部分積 $\boldsymbol{P}_{n-2}, \cdots, \boldsymbol{P}_{-1}$ をそれぞれ次のように表す．

$$\boldsymbol{P}_t = \boldsymbol{X}\cdot Y_t \times 2^{2t} = (p_{(2n-1)t}\cdots p_{st}\cdots p_{-1t}) \times 2^{2t} \quad (31)$$

$$p_{st} \in \{\overline{1}, 0, 1\}, \quad (t = -1, \cdots, n-2)$$

このとき，これらの部分積はシフトと符号反転により，次式で生成される．

(i) t : even の場合

$$Y_t = 2 \text{ のとき } p_{st} = \begin{cases} \overline{x_{s-1}} & (s = 4m) \\ x_{s-1} & (\text{その他}) \end{cases} \quad (32)$$

$$Y_t = 1 \text{ のとき } p_{st} = x_s \quad (33)$$

$$Y_t = 0 \text{ のとき } p_{st} = 0 \quad (34)$$

$$Y_t = \overline{1} \text{ のとき } p_{st} = \overline{x_s} \quad (35)$$

$$Y_t = \overline{2} \text{ のとき } p_{st} = \begin{cases} x_{s-1} & (s = 4m) \\ \overline{x_{s-1}} & (\text{その他}) \end{cases} \quad (36)$$

(ii) t : odd の場合

$$Y_t = 2 \text{ のとき } p_{st} = \begin{cases} x_{s-1} & (s = 4m) \\ \overline{x_{s-1}} & (\text{その他}) \end{cases} \quad (37)$$

$$Y_t = 1 \text{ のとき } p_{st} = \begin{cases} x_s & (s = 4m, 4m+1) \\ \overline{x_s} & (\text{その他}) \end{cases} \quad (38)$$

$$Y_t = 0 \text{ のとき } p_{st} = 0 \quad (39)$$

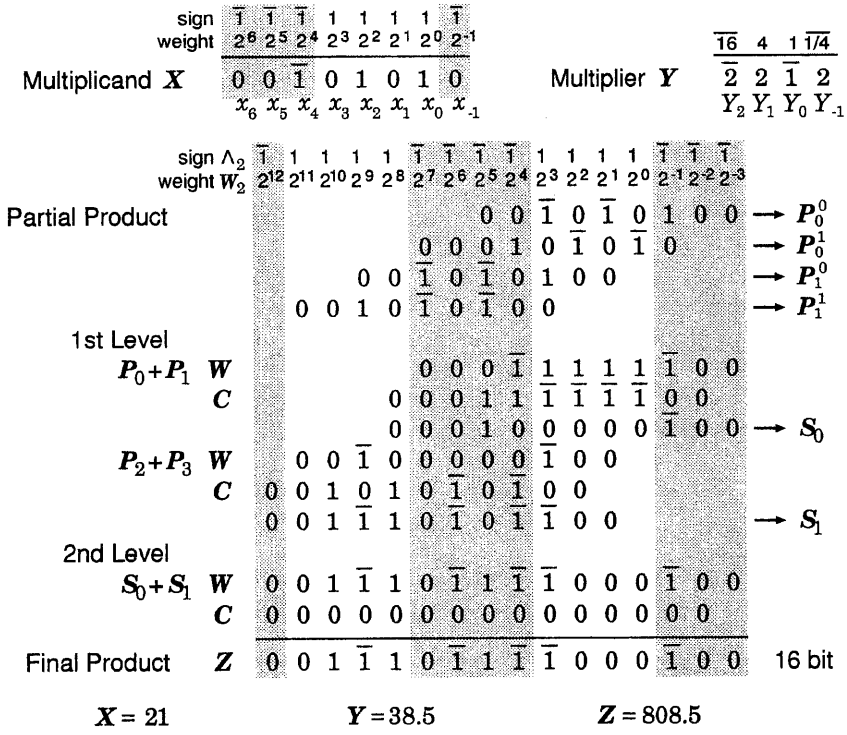

図 5: 実数乗算の例 (8 ビット)

$$Y_t = \overline{1} \text{ のとき } p_{st} = \begin{cases} \overline{x_s} & (s = 4m, 4m+1) \\ x_s & (\text{その他}) \end{cases} \quad (40)$$

$$Y_t = \overline{2} \text{ のとき } p_{st} = \begin{cases} \overline{x_{s-1}} & (s = 4m) \\ x_{s-1} & (\text{その他}) \end{cases} \quad (41)$$

このようにして生成された部分積を加算することにより，最終的な乗算結果 $\boldsymbol{Z}$ が得られる．この加算も 2 進 SD 数で行うため，2 進木構造による高速な加算が可能となる．図 5 に実数乗算の具体例を示す．

先程の複素数乗算の場合と違う点は，キャリーを 2 ビット上にではなく，隣のビットに入力するという点である．この再構成は，SD 全加算器のキャリー出力を 2 ビット先と隣のビットに，それぞれ入力しておき，これらを伝送ゲートで切替えることにより実現可能である．

4.3 単精度実数 4 入力積和演算 2 並列のアルゴリズム

次に，(iii) 単精度実数 4 入力積和演算 2 並列モードについて述べる．本モードでは，入力を $\langle D_3, \boldsymbol{W}_3, \boldsymbol{\Lambda}_3 \rangle$ に変換して 4 入力積和演算を 2 つ並列して行う．ここで，複素乗算は次式のように行われる．

$$(a+bj)(c+dj) = (ac-bd) + (ad+bc)j \quad (42)$$

従って，入力を a, b, c, d とすれば，複素乗算結果の虚部 $ad+bc$ がこれらの積和演算となっている．しかし，虚部を算出するときに動作するのは，全回路のうちの半分である．残りの半分を利用するため，実部を算出する部分には次式のような別のデータ $e, f, g, -h$ を入力する．

$$(e+fj)(g-hj) = (eg+fh) + (fg-eh)j \quad (43)$$

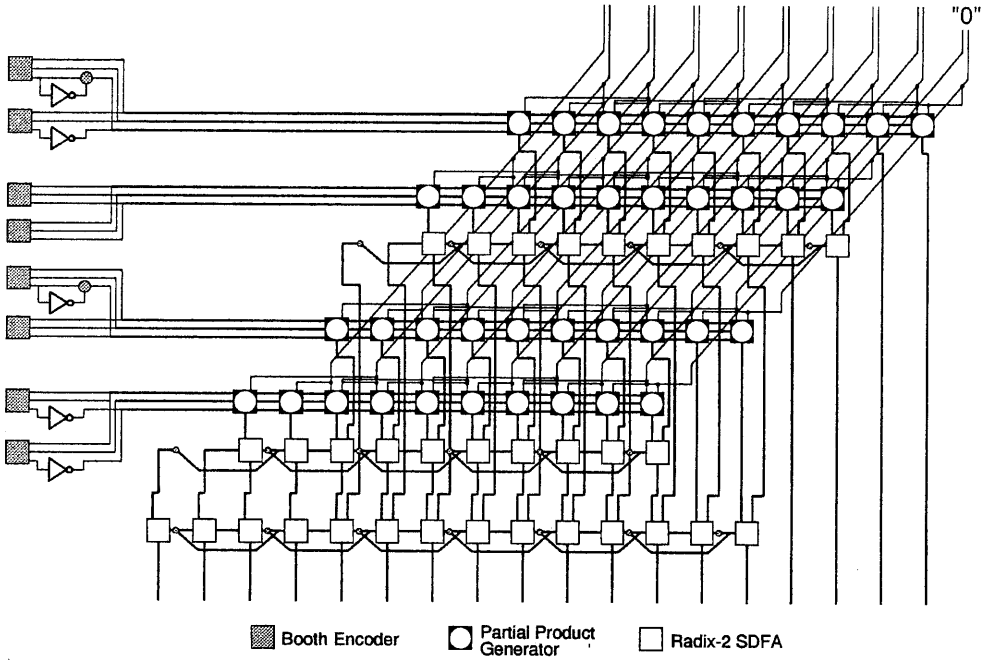

図 6: 実数／複素数再構成型算術演算回路 (実部・虚部 4 ビット)

このように乗数の符号を変えることにより，実部 $eg + fh$ がこれらの積和演算となる．このモードでは，部分積生成部への入力を選択しさえすれば，演算器内部での再構成は不要である．

5 再構成とその効率

ある回路を別の回路に再構成するということは，ハードウェア量に制限がなければ，どのような回路であっても理論的には可能である．しかしながら，一般に演算回路の構造の規則性を無視して設計すると，チップ面積を抑えるどころか増加させる結果となる．

以下では，グラフを用いて再構成の効率を定量的に評価する方法を提案する．まず，前節で述べた 3 つの演算モードを実行する回路は，図 6 のように構成される．この図の各演算モジュールと配線をそれぞれグラフの点と辺に対応させて，各モジュールを等間隔のグリッド上に配置して描き，そのグラフの辺の長さ (隣合うグリッド間の長さを単位長とする) から再構成の効率を次のように定義する．

(再構成効率)

$$= \frac{(\text{不変辺長})}{(\text{不変辺長}) + (\text{削除辺長}) + (\text{追加辺長})} \tag{44}$$

また，各演算モードで使用する配線の全配線長に対する割合を，配線の利用率として次式のように定義する．

(配線利用率)

$$= \frac{(\text{そのモードで使用する辺長})}{(\text{不変辺長}) + (\text{削除辺長}) + (\text{追加辺長})} \tag{45}$$

従って，再構成の効率とは各演算モード間の相対的な値であり，配線の利用率はその演算モードでの絶対的な値である．図 7 に本提案の回路と実数乗算器 4 つで構成した回路との比較を示した．矢印が再構成効率，丸の中が配線利用率を示している．

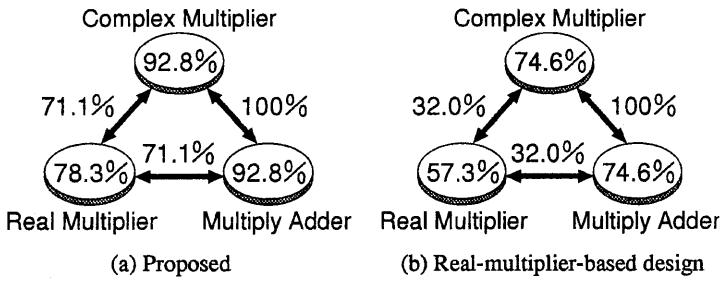

図 7: 再構成効率 (矢印)，配線利用率 (丸の中) の比較

両者で差が顕著であるのは，複素数乗算器と実数乗算器間の再構成効率である．本提案の演算器では，再構成後もその 71.1 %の配線が利用されるのに対し，実数乗算器 4 つで構成された演算器では，32.0 %しか使われていない．つまり，このような再構成の仕方では，半分以上の配線が無駄となってしまうことになる．

また，複素数乗算器と積和演算器間の再構成効率が，ともに 100 %となっているのは，グリッドモデルを描く際に部分積生成部の入力まで考慮していないためであり，実際には演算器への入力線を切替える必要がある．

6 むすび

本稿では実数／複素数再構成型演算回路を提案し，再構成の効率等の評価を通して，その有用性を示した．また，演算器の再構成は，数系の代数的な変換によって議論可能であることを初めて示した．その際には，重みベクトル $\boldsymbol{W}$ と符号ベクトル $\boldsymbol{\Lambda}$ を用いた表現が有用である．本稿で提案した設計手法は，他の再構成型演算回路の設計を議論する際にも有効であると考えられる．

このような再構成型の演算回路は，チップ面積を大幅に削減できるため，実数演算，複素数演算ともに必要とするディジタル信号処理に非常に有用である．

参考文献

[1] Y. Ohi, T. Aoki, and T. Higuchi, "Redundant complex number systems," *Proc. 25th IEEE Int'l Symp. Multiple-Valued Logic*, pp. 14-19, May 1995.

[2] T. Aoki, Y. Ohi, and T. Higuchi, "Redundant complex number arithmetic for high-speed signal processing," *Proceeding of the 1995 IEEE Workshop on VLSI Signal Processing*, October 1995.

[3] A. Avizienis, "Signed-digit number representations for fast parallel arithmetic," *IRE Trans. Electronic Computers*, Vol. EC-10, pp. 389–400, September 1961.

[4] Y. Harata *et al.*, "A high-speed multiplier using a redundant binary adder tree," *IEEE J. Solid-State Circuits*, Vol. SC-22, pp. 28–34, 1987.

[5] H. Makino, *et al.*, "An 8.8-ns 54×54-bit multiplier with high speed redundant binary architecture," *IEEE J. Solid-State Circuits*, Vol. 31, No. 6, pp. 773–783, June 1996.

[6] A. D. Booth, "A signed binary multiplication technique," *Quart. J. Mech. Appl. Math.*, Vol. 4, pp. 236–240, August 1951.