社団法人　情報処理学会　研究報告
IPSJ SIG Technical Reports

2005－SLDM－118（27）
2005／1／26

SD数演算を用いた剰余数系一重み数系変換回路

小川　由真†　　陳　土爽清†　　魏　書剛††

†† 群馬大学工学研究科 情報工学専攻
〒376-3515 群馬県桐生市天神町 1-5-1
E-mail: †{ogawa,chen,wei}@ja4.cs.gunma-u.ac.jp

あらまし　SD（Signed-Digit）数演算を剰余数演算に導入することにより、剰余数系における算術演算が高速に行われる。特に、$2^p \pm 1$ と 2^p を法とした場合、剰余数加算回路は、SD 数加算回路に設計できる。本論文では、非重み数系の剰余数系の SD 数表現を重み数表現へ変換するため、SD 数剰余演算を用いる。設計した SD 数演算の剰余数系一重み数系の変換回路の高速性を、従来の 2 進数演算による変換回路との比較により明らかにする。

キーワード　剰余数系，重み数系，混合基数，Signed-Digit(SD) 数

Residue-to-Weighted Converter Using Signed-Digit Number Arithmetic

Yumi OGAWA†, Shuangching CHEN†, and Shugang WEI††

†† Department of Computer Science, Gunma University
Tenjin1-5-1,Kiryu-shi,376-3515 Japan
E-mail: †{ogawa,chen,wei}@ja4.cs.gunma-u.ac.jp

Abstract　By introducing a signed-digit(SD) number arithmetic into a residue number system (RNS), arithmetic operations can be performed efficiently. In the cases of modulu $m = 2^p - 1$, $2^p + 1$ and 2^p, the modulo m addition can be implemented with an end-around-carry SD adder. In this study, we use the SD number arithmetic circuits to construct a residue-to-weighted converter. Compared with the binary number system by simulation, the proposed residue-to-weighted converter with the SD number representation has hign performance for large moduli.

Key words　residue number system(RNS), weighted number system , mixed-radix system, signed-digit(SD) number

1.　はじめに

従来の 2 進数系は、加算間の桁上げ伝播が生じるため、演算速度が制限される。これに対し、剰余数系は、お互いに素である法を選び、各剰余桁において、独立に並列処理を行うことができ，高速な算術演算システムが期待されている [1]。

また、法を $2^p, 2^p \pm 1$ とした場合、剰余加算は p ビットの 2 進加算器を用いてメモリなしで算術演算を行うことができる [2], [3]。これらの法を用いた剰余乗算器が提案されている [3], [4]。しかし、これらの $2^p, 2^p \pm 1$ を法とした加算器、乗算器は 2 進数系を使用しているため、剰余桁内部での桁上げが生じ、演算速度に影響を及ぼす原因となる。

一方、Singed-Digit(SD) 数系は、加算の際に桁上げ伝播が生じないという特徴をもつ数系である [5]。そこで、剰余演算に SD 数を導入すると、剰余桁内部でも桁上げのない高速な算術演算を行うことができる [6], [7]。

以上の高速処理が行うことができる剰余演算で得られる結果は剰余数系であり、非重み数系である。剰余数系以外のシステムに出力する場合、非重み数系から重み数系に変換する必要がある。非重み数系から重み数系に変換する方法として、中国人剰余定理と、混合基数変換の 2 通りの方法が良く知られている。

中国人剰余定理は、原理は簡単であるが、演算に M ($M = \prod_{i=1}^{n} m_i$) を必要とする。一方、剰余計算機は m_i を法とする演算で構成されているので、中国人剰余定理を用いて非重み数系から重み数系に変換するのに適していない。

混合基数数系は、重み数系のため、大きさの比較が容易であること、剰余計算機では、混合基数変換が比較的速いことの 2 つの理由から、混合基数変換の方法を用いる。

本研究では、混合基数変換を SD 剰余演算を用いることにより、高速の剰余数系一重み数系の変換回路を加算のみで実現し、その高速性を、通常の 2 進数演算と比較することにより、明らかにする。

表 1　SD 加算の中間和と桁上げの生成規則

	$abs(x_i) = abs(y_i)$	$abs(x_i) \neq abs(y_i)$	
		$(x_i + y_i) \times (x_{i-1} + y_{i-1}) \leq 0$	$(x_i + y_i) \times (x_{i-1} + y_{i-1}) > 0$
w_i	0	$x_i + y_i$	$-(x_i + y_i)$
c_i	$(x_i + y_i)/2$	0	$x_i + y_i$

2. 冗長な対称剰余数数系と重み数系

2.1 剰余数数系

$\{m_n, \cdots, m_2, m_1\}$ をお互いに素である法の集合とし、正の整数 A に対して、A 剰余表現は $(a_n, \cdots, a_2, a_1)$ に定義されている。ここで $m_i > 1$ であり、$a_i = |A|_{m_i}$ である。また A の剰余表現は唯一である。a_i は次の式で求める。

$$a_i = |A|_{m_i} = A \ mod \ m_i \tag{1}$$

ここで、$a_i \in L_{m_i} = \{0, 1, \cdots, m_i - 1\}$ である。A は $0 \leqq A \leqq (M-1)$ を満たす $(M = \prod_{i=1}^{n} m_i)$。

算術演算が簡単に行えるため、次の L_{m_i} を定義する。p 桁の SD 数の数範囲は次の L_{m_i} で表される。

$$\begin{aligned} L_{m_i} &= \{-(2^p - 1), \cdots, -(m_i - 1)/2, \\ &= \cdots, (m_i - 1)/2, \cdots, (2^p - 1)\} \end{aligned} \tag{2}$$

整数 X の SD 剰余数 $x_i = \langle X \rangle_{m_i}$ は

$$x_i = \langle X \rangle_{m_i} = \left| X \right|_{m_i} \tag{3}$$

あるいは

$$x_i = \langle X \rangle_{m_i} = \left| X \right|_{m_i} - m_i \tag{4}$$

で表される。

例えば、29 mod 17 は式 (3) より、$\langle 29 \rangle_{17} = |29|_{17} = 12$ になり、式 (4) により $\langle 29 \rangle_{17} = -5$ になる。

L_{m_i} の定義により、二つの冗長な値をもつ。それに SD 数を導入することにより、L_{m_i} 中のすべての値は冗長性をもつ SD 数表現で表せ、その 2 重の冗長性を利用して高速化を実現する。

A と B の剰余表現は $(a_n, \cdots, a_2, a_1)$ と $(b_n, \cdots, b_2, b_1)$ とすると、剰余算術演算は $\langle A \otimes B \rangle_{m_i} = (c_n, \cdots, c_2, c_1)$ で表される。ここで $c_i = \langle a_i \otimes b_i \rangle_{m_i}$ であり、$\otimes$ は加算、減算と乗算という意味を表される。

2.2 混合基数数系

混合基数数系は、各桁に重みが加えられた数系なので、大きさの比較が一見して可能である。混合基数数系において、整数 X は次のように表す。

$$X = a_n \prod_{i=1}^{n-1} m_i + ... + a_3 m_2 m_1 + a_2 m_1 + a_1 \tag{5}$$

ここで $0 \leqq a_i < m_i$ また $m_i > 1$。m_i は基数であり、a_i は係数である。

3. SD 演算を用いた剰余演算

従来の剰余演算では、2 進数表現を用いて演算を行う。しかし、2 進数加算により桁上げ伝播が生じ、演算速度が制限されてしまう。ここで、剰余演算を SD 数表現を用いて演算を行うと、桁上げ伝播が無いので、演算速度が桁数により左右されないという利点がある。

3.1 SD 数表現

SD(signed digit) 表現は、通常の 2 進表現と同様、ある整数 X を、次式で表す。

$$X = x_{p-1} 2^{p-1} + x_{p-2} 2^{p-2} + ... + x_1 2^1 + x_0 \tag{6}$$

ここで、$x_i \in \{-1, 0, 1\}$。SD 数表現は冗長性をもつ。つまり、ある値に対して複数の表記法ができる。例えば、4 桁で 10 進数の 5 を表すと $(0101)_{2SD}$、$(1\bar{1}1\bar{1})_{2SD}$、$(10\bar{1}\bar{1})_{2SD}$ などの表記ができる。

3.2 剰余 SD 数加算

剰余数数系の加算は、加算演算後に剰余をとるため、演算後の桁数と入力桁数に変化が無い。SD 数数系加算は、一部の桁の情報のみから加算演算が可能で、桁上げが生じない。SD 数加算は、2 段階に分けて処理することにより、桁上げ伝播を防ぐ。第 1 段階では、計算規則を記した表をもとに、桁上げ c_i と、中間和 w_i を求め、第 2 段階では、求めた桁上げ c_{i-1} と、中間和 w_i を加算し、最終和を求める。

Algorithm 1

(1)　次式を満たすように、計算規則を決め、各桁で c_i, w_i を求める。ここで、c_i は桁上げ、w_i は中間和である。

$$ADD1: \quad 2 \times c_i + w_i = x_i + y_i \tag{7}$$

(2)　求めた桁上げ c_{i-1} と、中間和 w_i を加算し、最終和 s_i を各桁で求める。

$$ADD2: \qquad s_i = c_{i-1} + w_i \tag{8}$$

中間和と桁上げの生成ルールは表 1 で表される。$abs(x_i)$ は x_i の絶対値で表される。

法を $m = 2^p + \mu$ とすると、次式を満たす。

$$\langle 2^p \rangle_m = -\mu \tag{9}$$

$\mu \in \{-1, 0, 1\}$ の場合、次式が成り立つ。

$$\langle c_{p-1} 2^p \rangle_m = -c_{p-1} \times \mu = c_{-1} \tag{10}$$

上式から c, x, y のそれぞれの最下位桁よりさらに 1 つ下位桁を求めることができ、-1 桁と 0 桁との加算演算は、剰余を求めることと同値である。

$$c_{-1} = -\mu \times c_{p-1} \tag{11}$$

$$x_{-1} = -\mu \times x_{p-1} \tag{12}$$

$$y_{-1} = -\mu \times y_{p-1} \tag{13}$$

法を $2^p, 2^p \pm 1$ とした場合、求めた和の最上位桁に法の係数 {-1,0,1} を乗算し、最下位桁と加算することにより、剰余演算を求めることができる。この加算演算に SD 加算演算を用いることにより、剰余 SD 数加算は、全て SD 加算演算を用いて実行することとなり、全ての桁において、桁上げ伝播の無い、高速な加算演算ができる。

3.3 剰余 SD 数乗算

剰余 SD 乗算は、通常の 2 進乗算と同じく、剰余部分積と剰余加算を求めるため、シフト演算と加算演算から構成される。通常の 2 進乗算と異なる部分は、SD 数として扱うため、シフト演算は符号を考えずに簡単に求まること、また加算演算は SD 数加算を用いるため、桁上げ伝播が無く、高速演算が期待できる。

剰余部分積は SD 数表現であるので、乗数が 0 の時、剰余部分積は 0、乗数が 1 の時、剰余部分積は被乗数そのまま、乗数が-1 の時、剰余部分積は被乗数の各桁の符号ビットを反転させたものとなる。

剰余部分積の生成はその後の加算に比べ、圧倒的に短い時間で得られるので、乗算器の速度は加算器の演算速度に左右される。仮に X は被乗数、Y は乗数、剰余乗算は次式で表せる。

$$\langle X \times Y \rangle_m = \left\langle \sum_{i=0}^{p-1} \langle y_i 2^i \times (X) \rangle_m \right\rangle_m \tag{14}$$

$$= \left\langle \sum_{i=0}^{p-1} \langle y_i SX_i \rangle_m \right\rangle_m \tag{15}$$

$$= \left\langle \sum_{i=0}^{p-1} pp_i \right\rangle_m \tag{16}$$

ここで、$SX_i = \langle 2^i \times (X) \rangle_m$ は剰余シフト、pp_i は剰余部分積で表される。

剰余 SD 数乗算は次の 3 つのステップから実行することができる。(9) から、剰余シフト SX_i について次のような式変形ができる。

Algorithm 2

(1) X をシフトさせ, 剰余を求める

$$\begin{aligned} SX_i &= \langle 2^i \times X \rangle_m \\ &= \left\langle 2^i \times (x_{p-1}2^{p-1} + x_{p-2}2^{p-2} + \ldots + x_0) \right\rangle_m \\ &= \left\langle \left\langle 2^p (x_{p-1}2^{i-1} + x_{p-2}2^{i-2} + \ldots + x_{p-i}) \right\rangle_m \right. \\ &\quad \left. + x_{p-i-1}2^{p-1} + x_{p-i-2}2^{p-2} + \ldots + x_0 2^i \right\rangle_m \\ &= \left\langle x_{p-i-1}2^{p-1} + x_{p-i-2}2^{p-2} + \ldots + x_0 2^i \right. \\ &\quad \left. - \mu(x_{p-1}2^{i-1} + \ldots + x_{p-i+1}2 + x_{p-i}) \right\rangle_m \end{aligned} \tag{17}$$

(2) 乗数 y_i と剰余シフト SX_i との剰余部分積 pp_i を求める

$$pp_i = y_i \times SX_i \tag{18}$$

(3) 剰余 SD 数加算を用いて pp_0から pp_{p-1}まで の加算を実行

$$\langle X \times Y \rangle_m = \langle pp_0 + pp_1 + \ldots + pp_{p-1} \rangle_m \tag{19}$$

p 個の剰余部分積 pp_i を剰余 SD 加算で求める。この加算は 2 分木構造となるため、X, Y が p 桁の場合、$log_2 p$ 段の剰余加算演算で処理できることになる。

4. 混合基数への変換

4.1 2 進数表現を用いた混合基数変換

剰余数数系の法と、混合基数数系の基数を同一のものとすると、混合基数数系は、剰余数系と同じレンジを持ち、混合基数数系で、ある正整数 X は $X = (a_3, a_2, a_1)$ と表記され、次のように表される。

$$X = a_3(m_2 m_1) + a_2 m_1 + a_1 \tag{20}$$

m_i は基数で、a_i は係数で、$0 \leq a_i < m_i$ である。剰余数数系の法と、混合基数数系の基数が同じレンジを持つため、上の式はどちらの数系でも成り立つので、剰余数数系から、混合基数数系に変換することができる。混合基数数系は、各桁に重みのある数系により、変換後は、大きさの比較が一見して可能となる。

仮に X の剰余数表現は $(x3, x2, x1)_{(m_3, m_2, m_1)}$ とする。混合基数数系の係数 a_1 を求めるには、法 m_1 とした X の剰余を求めればよい。a_2, a_3 も上式からそれぞれ次のように求めることができる。

$$a_1 = |X|_{m_1} = x_1 \tag{21}$$

$$a_2 = \left| \frac{x_2 - a_1}{m_1} \right|_{m_2} = \left| |\frac{1}{m_1}|_{m_2} \times |x_2 - a_1|_{m_2} \right|_{m_2} \tag{22}$$

$$\begin{aligned} a_3 &= \left| \frac{|\frac{x_3 - a_1}{m_1}|_{m_3} - a_2}{m_2} \right|_{m_3} \\ &= \left| |\frac{1}{m_2}|_{m_3} \times \left((|\frac{1}{m_1}|_{m3} \times |x_3 - a_1|_{m_3} - a_2) \right) \right|_{m_3} \end{aligned} \tag{23}$$

ここで、$|\frac{1}{m_1}|_{m_2}$ は m_2 に対して m_1 の乗算逆元を表される。剰余除算は剰余乗算で計算できる。つまり、係数 a_1, a_2, a_3 は剰余減算と剰余乗算によって求められる。

混合基数変換では、a_3 を求める手順が一番演算処理が多く、a_2 が次に続く。そのため、剰余演算処理が最も多い a_3 は桁あふれ切り捨てで剰余を求められる法 2^p にするべきである。次に剰余演算処理が多い a_2 は加算で剰余を求める法 $2^p - 1$, 最後の a_1 は剰余演算が無いので減算で剰余を求める法 $2^p + 1$ を選択する。

例 1: $p = 3$ とし、法を 7,8,9 とし、27 の剰余数表現 $((011),(110),(0000))_{8,7,9}$ から混合基数に変換する手順を示す。よって $(a_3,a_2,a_1) = ((000),(011),(0000))_{63,9,1}$。$a3\times63+a2\times$

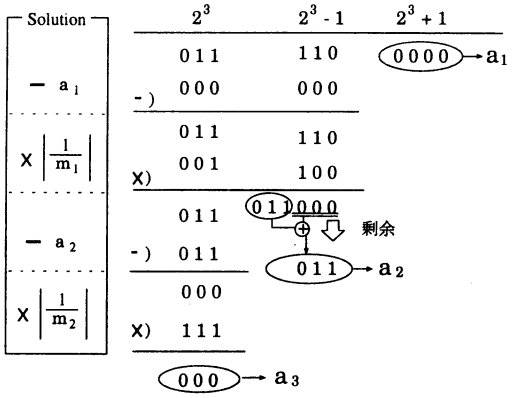

図 1　剰余数一混合基数変換 (2 進数表現)

$9+a1=27$ より、正しい結果を得られていることを分かった。

混合基数変換は、剰余減算と剰余乗算で構成されている。剰余数系から混合基数数系に変換する際、演算処理時間に最も影響を及ぼすのは剰余乗算演算である。乗算は加算で構成されているため、剰余加算演算の高速化が、剰余乗算演算の高速化につながり、結果混合基数変換の高速化につながる。

4.2 SD 数表現を用いた混合基数変換

通常の混合基数変換と同じ手順であるが、一番異なる点は、SD 数表現を用いるため、負の数を扱えるという点である。また、剰余加算演算において、桁上げ伝播が無いため、混合基数変換の高速化が期待できる。

SD 数表現を用いているため、混合基数変換後の混合基数 a_1,a_2,a_3 においても、負数が存在する。SD 数表現を用いたことによる利点と、混合基数変換演算の簡単化において、次の 5 つを挙げることができる。

（1） 剰余加算演算の高速化

剰余数に SD 数表現を用いることにより、剰余加算演算に SD 数加算を用いることができる。SD 数加算を用いることにより、剰余加算演算における桁上げ伝播を防ぐことができ、桁数によらず、一定時間で処理することができる。このことから、剰余加算演算の高速化が見込まれ、剰余 SD 数加算演算を用いることで、混合基数変換においても、高速化が期待できる。

（2） 符号反転の簡単化

混合基数変換には、減算演算がある。加算器を用いる場合、符号を反転し、加算演算をする操作がある。通常の 2 進表現を用いた場合、減算処理または、2 の補数を求めるための処理操作の後加算演算処理が必要である。SD 数表現を用いた場合は、要素に負の数を含んでいるため、符号反転において特別な処理を必要とせず、加算演算処理に移ることができる。

（3） 剰余演算の簡単化

混合基数変換には演算毎に剰余を求める操作がある。法を $2^p,2^p\pm1$ とすることにより、剰余演算は容易に求めることができるが、2 進数表現を用いた場合、テーブルを参照し、剰余を求めるという複雑な操作が必要となる。一方、法を $m=2^p,2^p\pm1$ とした場合、1 個の SD 加算器で剰余を求めることができる [4]。

（4） 法の逆元の簡単化

混合基数変換には、法の逆元との乗算演算部分が存在する。このため、法の逆元を求めるために、複雑な演算が必要となる。しかし、本研究では、法を $2^p,2^p\pm1$ とすることにより、次のような式変形が可能となり、法の逆元を簡単な形で求めることができる。

$$\left\langle\frac{1}{2^k-1}\right\rangle_{2^k+1} = 2^{k-1} \tag{24}$$

$$\left\langle\frac{1}{2^k-1}\right\rangle_{2^k} = 2^k-1=-1 \tag{25}$$

$$\left\langle\frac{1}{2^k}\right\rangle_{2^k+1} = 2^k \tag{26}$$

$$\left\langle\frac{1}{2^k}\right\rangle_{2^k-1} = 1 \tag{27}$$

$$\left\langle\frac{1}{2^k+1}\right\rangle_{2^k} = 1 \tag{28}$$

$$\left\langle\frac{1}{2^k+1}\right\rangle_{2^k-1} = 2^{k-1} \tag{29}$$

以上から、混合基数変換の乗算演算を、シフト移動、符号反転、そのまま出力、の 3 通りに置き換えることができる。これを代入することにより、乗算演算の式は簡単化され、処理時間を大幅に減らすことができる。

（5） 法の順番

(4) から、混合基数変換の乗算演算を、シフト移動、符号反転、そのまま出力、の 3 通りに置き換えることができる。剰余乗算演算で示した式 (17) から、シフト移動は、$\mu=-1$ の場合、符号反転なしの桁連結のみで表すことができる。$\mu=-1$ の場合のシフト移動は最上位桁の符号反転しながら最下位に桁連結を行う。よって m_1 は 2^p+1 にするべき。演算処理が一番多い a_3 は $m_3=2^p$ にすると m_1 と m_2 に対する乗算逆元はそれぞれ 1(式 27) と-1(式 24) となり、剰余乗算は簡単に求める。以上から $m_1=2^p+1, m_2=2^p-1, m_3=2^p$ を決定すると一番理想的な順番である。

これを混合基数の基数とすることにより、混合基数変換における乗算演算の式は簡単化され、処理時間を大幅に減らすことができる。

(x_3,x_2,x_1) は X を法 $(2^p,2^p-1,2^p+1)$ に対する剰余表現とし、$Z1,Z2,Z3$ は減算演算結果、$D1,D2$ は乗算演算結果、各 a_i は変換後の混合基数の係数で表される。SD 数を用いた混合基数変換アルゴリズムを次の Algorithm 3 で示す。

Algorithm 3

(1)　a_1 を求める。

$a_1 = x_1$;

(2)　$a2$ を求める。

(2A) $Z1 = \langle x_2 - a_1\rangle_{m_2}$;

(2B) $a_2 = \langle 2^{p-1}Z1\rangle_{m_2}$;

(3) a_3 を求める。

(3A) $Z2 = \langle x_3 - a_1 \rangle_{m_3}$;

(3B) $D1 = \langle Z2 \times 1 \rangle_{m_3}$

(3C) $Z3 = \langle D1 - a_2 \rangle_{m_3}$;

(3D) $D2 = \langle Z3 \times (-1) \rangle_{m_3}$;

(3E) 正の数を選択し、出力する。$D2 \geqq 0$ の時 $a_3 = D2$、その以外の場合、$a_3 = m_3 + D2$。

例 2: $p = 3$ とし、法を 7,8,9 とし、27 の剰余 SD 数表現 $((011),(00-1),(000))_{7,8,9}$ から混合基数に変換する手順を示す。

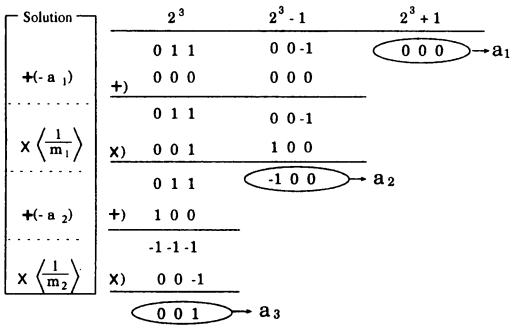

図 2 SD 剰余数ー SD 混合基数変換 (SD 数表現)

よって $(a_3, a_2, a_1) = ((001),(-100),(000))_{63,9,1}$。$a3 \times 63 + a2 \times 9 + a1 = 27$ により、正しい結果を得られていることを分かった。

5. 回路設計

図 3 は剰余 SD 数加算器を示す。図 3 にあるように、p 桁の SD 数加算器 (SDA) は、p 個の SD 全加算器 (SDFA) で構成される。それぞれの SDFA は、ADD1 と、ADD2 のブロックから構成され、アルゴリズム 1 の (1),(2) の順に実現する。最下位の SDFA は、剰余を求めているので、法を $2^p, 2^p \pm 1$ とした時の剰余加算器 (MSDA) は、SDA のみで実現できる。

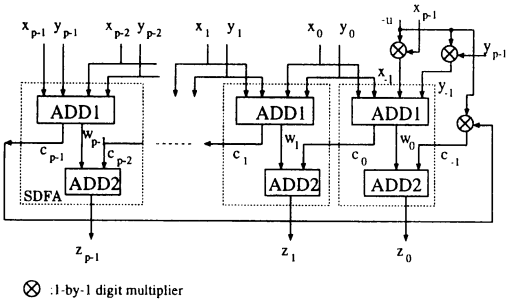

図 3 剰余 SD 数加算器

図 4 は 8 桁の剰余 SD 数乗算器を表され、3 段の剰余 SD 数加算器を用いて 2 分木構造で実現できる。アルゴリズム 2 の (1)(2) の後に (3) が MSDA の 2 分木で実現する。この方法で求める法 m の剰余乗算処理時間は $O(log_2 p)$ である。

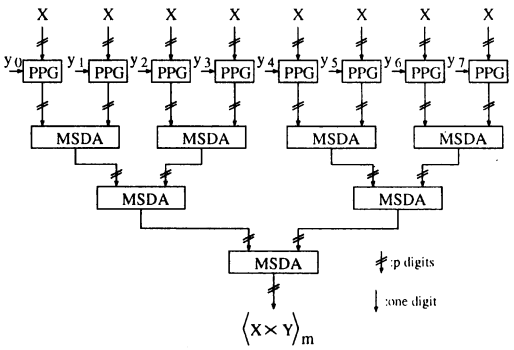

図 4 剰余乗算器

図 5 に示すように、SD 数演算を用いた混合基数変換器は 3 つの MSDA と、シフタ、decide で構成され、アルゴリズム 3 を実現する。混合基数の基数 a_1 は SD 剰余数そのままの出力、a_2 は、アルゴリズム 3 の (2A) の減算を MSDA で実現、(2B) の乗算をシフタのみで実現できる。a_3 は 2 つの MSDA で (3A) と (3C) の減算を実現、(3C) の剰余加算は、そのままの出力と、(3D) 符号反転で実現する。また、入力を正に限定しているため、a_3 のみ、deside で正の数を選択し、出力する。

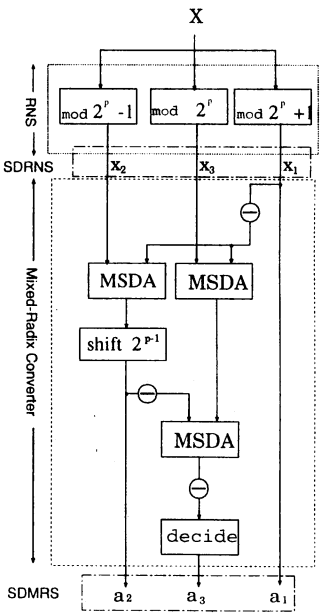

図 5 SD 数表現を用いた混合基数変換器

6. 回路評価

表 2 は 2 進の 2 ビットで 1 桁の SD 数を示す。$a_i(1)$ と $a_i(0)$ はそれぞれ a_i の符号と絶対値を表す。

表 2　SD 数の 2 進数表現

a_i	$a_i(1)$	$a_i(0)$
-1	1	1
0	0	0
1	0	1

上記の 2 値符号を利用して、p 桁の SD 数を $2p$ ビットの 2 値符号列で表す。例えば、$13 = (1,0,0,-1,-1)_{SD}$ は [01 00 00 11 11] で表現される。回路評価は $1\mu m$CMOS ゲートアレーを用いたものである。SD 数表現と 2 進表現を用いた混合基数変換の回路評価は表 3 に示す。通常の 2 進数表現を用いた場合と、SD 数表現を用いた場合の混合基数変換回路を比較する。2 進数表現と比べ、SD 数表現を用いた場合、4 桁、8 桁共に、回路面積は約 2 倍弱となるが、遅延時間は、4 桁の場合、2 進数表現と比べ、SD 数表現を用いた場合、遅延時間は約 16.2%減となる。8 桁の場合、SD 数表現を用いた場合、遅延時間が約 50%減となる。桁数が増大するほど、遅延時間の違いが顕著になり、SD 数表現を用いた場合の高速性が明らかとなる。

表 3　SD 数表現と 2 進表現を用いた混合基数変換の回路評価

語長	面積 (ゲート)		遅延時間 (ns)	
	2 進数	SD 数	2 進数	SD 数
4 桁	147	238	18.73	15.70
8 桁	329	563	38.81	20.97

7. ま と め

従来の 2 進数表現を用いた剰余演算は桁上げ伝播により、演算速度の高速化に限界がある。これに対し、SD 数表現を用いた剰余演算は剰余桁内部での桁上げが無いため、高速な演算処理が可能となる。本研究では、この SD 数表現を用いた剰余演算を使用し、SD 剰余数から重み数系である SD 混合基数へ高速に変換をする方法を提案した。

混合基数変換には、剰余加算演算と剰余乗算演算が必要であるので、剰余 SD 数加算、剰余 SD 数乗算について述べた。整数 $m = 2^p, 2^p \pm 1$ を選択し、m を法とした剰余数系の演算を SD 表現を用い、剰余桁内部においても桁上げの無い、高速な加算器を作成した。また、m を法とした SD 剰余乗算器は、SD 剰余加算器の 2 分木で構成され、$log_2 p$ 段の加算処理でできる高速な乗算器である。

さらに、本研究では、混合基数変換における乗算演算を簡単にすることにより、演算速度の高速化を図り、混合基数変換器を剰余 SD 加算器 3 個で実現することができた。

実験結果により、SD 数を用いた混合基数変換回路は回路面積はやや大きくなるが、遅延時間は約半分とすることができた。

SD 数を用いるため、変換後の混合基数にも負数が存在する。これにより、重み数系である混合基数数系の大小比較に演算が必要となる。高速な大小比較する方法を今後の課題として研究していきたい。

文　献

[1] N.S.Szabo and R.I.Tanaka, "Residue Arithmetic and Its Applications to Computer Technolorgy", New York:McGraw-Hill, 1967.

[2] D.P. Agrawal and T.R.N.Rao,"Modulo $(2^n + 1)$ arithmetic logic,"IEE J. Electronic Circuits and Systems, Vol.2, pp. 186-188, Nov. 1978.

[3] F.J.Taylor,"A VLSI residue arithmetic multiplier," IEEE Trans. Comput., Vol.C-31, pp.540-546,June 1982.

[4] A. Hiasat, "New memoryless, mod $(2^n \pm 1)$ residue multiplier," *Electron. Lett.*, vol.28, no.3, pp.314-315, Jan. 1992.

[5] A.Avizienis,"Signed-digit number representations for fast parallel arithmetic,"IRE Trans. Elect. Comput., EC-10,pp.389-400, Sept. 1961.

[6] S.Wei and K.Shimizu, "Modulo $2^p - 1$ arithmetic hardware algorithm using signed-digit number representation," Trans. IEICE, Vol.E79-D, No.3, pp.242-246, March 1996.

[7] S.Wei and K.Shimizu, "A Novel Residue Arithmetic Hardware Algirithm Using a Signed-Digit Number Representation," IEICETRANS.INF.&SYST., Vol.E83-D, No.12, pp.2056-2064, Dec. 2000.