

# 基数変換回路の構成法 (4)

## RNS の場合

井口 幸洋<sup>†</sup> 笹尾 勤<sup>††</sup>

<sup>†</sup> 明治大学 理工学部 情報科学科

<sup>††</sup> 九州工業大学 情報工学部 電子情報工学科

あらまし デジタル信号処理を高速化するために RNS (residue number system: 剰余数系) を用いることがある。このような場合、RNS と 2 進数との間の基数変換回路が必要である。本稿では、剰余数から 2 進数への高速な変換回路の構成法を示す。基数を指定することにより、RNS 二進変換回路を自動的に合成するシステムを試作し、多数の回路を FPGA 上に実現した結果を示す。

キーワード 基数変換, 剰余数系, 多値論理, FPGA.

# Design Method of Radix Converters (4)

## RNS to binary Converters

Yukihiro IGUCHI<sup>†</sup> and Tsutomu SASAO<sup>††</sup>

<sup>†</sup> Department of Computer Science, Meiji University

<sup>††</sup> Department of Computer Science and Electronics, Kyushu Institute of Technology

**Abstract** In digital signal processing, RNS (residue number system) is often used for high-speed computation. In such cases, radix converters are necessary. This paper presents design methods of radix converters from residue numbers into binary numbers. We have developed an automatic design system for RNS to binary converters. Various radix converters were designed to compare their performance.

**Key words** Radix converter, residue number system, multiple valued logic, FPGA.

## 1. はじめに

デジタル信号処理では、通常、2 進数系が使用される [9]。しかし、高速のデジタル信号処理では、剰余数系 (**Residue Number System, RNS**) をよく使う [3] ~ [5], [8], [9], [14]。また、情報セキュリティへの応用としてサイド・チャンネル・アタック対策としても RNS が用いられている [1]。

高速信号処理への応用では、AD(Analog Digital: アナログ・デジタル) 変換程度のビット数の精度が扱われることが多い。一方、情報セキュリティへの応用では、128 ビット以上の大きなビット数の RNS が用いられる。いずれの応用でも、剰余数と 2 進数間の基数変換が必要である [2], [4], [12]。しかし、これらの応用ではビット長が大きく異なるので、適した変換回路の構成法は応用に依存する。本稿では、高速信号処理への応用を目的とする。従って、2 進数 10 ~ 15 ビット程度相当の数を高速に変換する回路を検討する。

剰余数を 2 進数に変換するには、中国人剰余定理 (**Chinese remainder theorem, CRT**) [13] を用いればよい [2], [12]。全ての入力組合せに対してこの変換を予め計算しておき、メモリなどに基数変換表として記憶し、表引きにより変換を実行すると高速に実行できる。しかし、入力数が大きくなると、必要なメモリ量が増加するので実用的ではなくなる。

そこで CRT を実現する上で、加算器は、Carry Save Adder (CSA) をツリー上に並べて高速化したり、剰余数を 2 進数に変換する上で高コストな剰余演算を工夫することで高速化を行っている [2], [12], [14]。

本稿では、基数変換表の二段 AND-OR 論理式を直接、論理合成するという単純な方法で基数変換回路を作る。これは、率直な

方法であるが、従来その方法を実験した報告が無い。CRT を率直に実現する場合に問題となる剰余加算を高速に求める構成法も併せて報告する。基数を指定することにより、RNS 二進変換回路を自動的に合成するシステムを試作し、多数の回路を FPGA 上に実現し、性能やハードウェア量を比較した結果を示す。

## 2. 剰余数系 (Residue Number System, RNS)

### 2.1 剰余数系とは [14]

剰余数系 (以後簡単のために RNS と表記) では、整数  $X$  を互いに素な  $n$  個の組  $(m_1, m_2, \dots, m_n)$  で除算し、剰余の組  $(r_1, r_2, \dots, r_n)$  で表現する。

[定義 2.1] 整数  $X$  を整数  $m_i$  で割ったときの商を  $q_i$ 、剰余を  $r_i$  ( $i = 1, 2, \dots, n$ ) とする。この時、 $X = q_i \cdot m_i + r_i$  が成り立つ。ここで  $r_i$  は、 $[0, m_i - 1]$  の範囲の整数値を持つ。 $r_i = X_{(\text{mod } m_i)}$  と表記する。

[定義 2.2] 整数  $X$  を互いに素な  $n$  個 ( $n \geq 2$ ) の整数の組  $(m_1, m_2, \dots, m_n)$  の各要素で除算し、その剰余の組を  $(r_1, r_2, \dots, r_n) = (X_{(\text{mod } m_1)}, X_{(\text{mod } m_2)}, \dots, X_{(\text{mod } m_n)})$  とする。これを整数  $X$  の RNS  $(m_1, m_2, \dots, m_n)$  表現と呼び、 $X_{rns(m_1, m_2, \dots, m_n)} = (\hat{x}_1, \hat{x}_2, \dots, \hat{x}_n)$  と書く。

[例 2.1] 10 進数表現 12 の RNS(3, 5, 7) による表現は、 $12_{rns(3, 5, 7)} = (12_{(\text{mod } 3)}, 12_{(\text{mod } 5)}, 12_{(\text{mod } 7)}) = (0, 2, 5)$  である。(例終り)

[定義 2.3]  $M = \prod_{i=1}^n m_i$  をダイナミックレンジと呼ぶ。

RNS では  $M$  個の連続する数値を表現できる。

[例 2.2] RNS(3, 5, 7) による RNS 表現では、 $M = 3 \times 5 \times 7 = 105$  となる。例えば、 $[0, 104]$  の範囲の整数値を表現できる。

(例終り)

## 2.2 RNS における演算

RNS における演算の特徴と高速信号処理に使用する理由を本節で述べる。

### 2.2.1 剰余数系における加減算と乗算

2進数での加算を考える。桁数が大きくなればなるほど、下位桁からの桁上がりの上位桁への伝搬時間が大きくなる。一方、RNS での加減算、及び、乗算は次のように行える。

$$(X \pm Y)_{(\text{mod } m)} = (X_{(\text{mod } m)} \pm Y_{(\text{mod } m)})_{(\text{mod } m)} \quad (2.1)$$

$$(X \cdot Y)_{(\text{mod } m)} = (X_{(\text{mod } m)} \cdot Y_{(\text{mod } m)})_{(\text{mod } m)} \quad (2.2)$$

[例 2.3]  $X = 12, Y = 5$  とする。RNS(3, 5, 7) による表現では、 $12_{rns(3,5,7)} = (0, 2, 5), 5_{rns(3,5,7)} = (2, 0, 5)$ 。

(1) 加算  $X + Y$ :

$(X + Y)_{rns(3,5,7)} = (12 + 5)_{rns(3,5,7)} = (2, 2, 3)$  となる。一方、各要素毎の加算を行うと  $((0 + 2)_{(\text{mod } 3)}, (2 + 0)_{(\text{mod } 5)}, (5 + 5)_{(\text{mod } 7)}) = (2, 2, 3)$  となり、これらの 2 つの値が一致する。

(2) 減算  $X - Y$ :

$(X - Y)_{rns(3,5,7)} = (12 - 5)_{rns(3,5,7)} = (1, 2, 0)$  となる。一方、各要素毎の減算を行うと  $((0 - 2)_{(\text{mod } 3)}, (2 - 0)_{(\text{mod } 5)}, (5 - 5)_{(\text{mod } 7)}) = (1, 2, 0)$  となり、これら 2 つの値が一致する。ここで  $-2_{(\text{mod } 3)} = 1$  である。

(3) 乗算  $X \cdot Y$ :

$(X \cdot Y)_{rns(3,5,7)} = (12 \cdot 5)_{rns(3,5,7)} = (0, 0, 4)$  となる。一方、各要素毎の乗算を行うと  $((0 \cdot 2)_{(\text{mod } 3)}, (2 \cdot 0)_{(\text{mod } 5)}, (5 \cdot 5)_{(\text{mod } 7)}) = (0, 0, 4)$  となる。

(例終り)

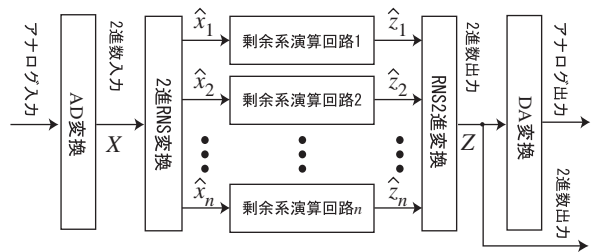
このように加減算と乗算では、各要素ごと(つまり  $X_{(\text{mod } m_i)}, i = 1, \dots, n$  ごと)に演算を行えるために要素間の桁上がりが生じない。つまり、各要素ごとの演算を独立かつ並列に行える。また、RNS の各要素のビット長は、もとの数値を 2 進数で表した場合のビット長よりも小さい。

加減算と乗算では、RNS 表現が有利である。一方、除算、大小比較は RNS ではコストがかかる [9]。従って、RNS は加減算と乗算が主たる演算となる応用に適する。例えば、デジタル・フィルタなどに有効である。

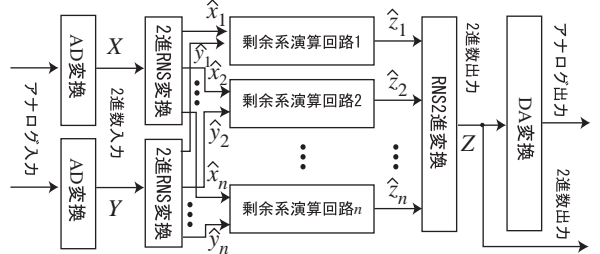
### 2.3 RNS を用いたデジタル信号処理システム

RNS をデジタル信号処理システムに用いた場合の典型的な構成図を図 2.1 に示す。図 2.1(a) では、1 個のアナログ入力をまず AD 変換して 2 進数  $X$  を得る。次に、この 2 進数を RNS に変換し、 $n$  個の値を求める。この  $n$  個の  $X_{(\text{mod } m_i)}$  を表すために必要な各ビット長は、 $X$  のビット長より小さい。これらの値を、並列に RNS 演算回路(もしくは RNS 演算を行う信号処理用プロセッサ)で演算処理を行う。得られた RNS を 2 進数に変換し、2 進数を得る。必要があれば DA(Digital Analog) 変換を行ってアナログ値を得る。例えば、デジタル・フィルタは、典型的な例である。図 2.1(b) では、2 入力の場合を示している。

AD 変換を用いてアナログ値を 2 進数に変換しているため、AD 変換器の精度程度の 2 進数を取り扱う。これを高速に RNS に変換し、RNS で演算を行い、またそれを 2 進数に変換する。AD 変換器が扱う精度は、8-14 ビット程度が多いので、本稿で扱う数



(a) 1入力の場合



(b) 2入力の場合

図 2.1 RNS を用いたデジタル信号処理システム  
値の範囲も上限を 14 ビットとする。

高速信号処理を行う場合、演算部だけでなく 2 進数から RNS への変換回路や RNS から 2 進数への変換回路も高速に実現する必要がある。本稿では、このうち RNS で表現された数値を 2 進数に変換する方法について検討する。

## 3. 基数変換回路

### 3.1 基数変換

[定義 3.1] 10 進数  $X$  を  $RNS(m_1, m_2, \dots, m_n)$  で表現し、 $X_{rns(m_1, m_2, \dots, m_n)} = (\hat{x}_1, \hat{x}_2, \dots, \hat{x}_n)$  で表す。一方、 $X$  は  $p$  桁の 2 進数  $\vec{x} = (x_{p-1}, x_{p-2}, \dots, x_0)_2$  で表される<sup>(注1)</sup>。 $(\hat{x}_1, \hat{x}_2, \dots, \hat{x}_n)$  が与えられたとき、 $\vec{x}$  を求める操作を RNS-2 進変換、その逆を 2 進-RNS 変換と呼ぶ。

[例 3.1] RNS(3,4) から 2 進数への変換を表 3.1 に示す。RNS の入力を、 $(\hat{x}_1, \hat{x}_2)$  と表す。出力である 2 進数の表記は、 $\vec{x} = (x_3, x_2, x_1, x_0)$  と表す。ここで  $(m_1, m_2) = (3, 4)$  であるので、 $\hat{x}_1 = [0, 2], \hat{x}_2 = [0, 3]$  の値をとる。これらは 2 進数で入力されるものとする。ここで  $d$  は、ドントケアを表す。(例終り)

### 3.2 中国人剰余定理を用いた RNS-2 進変換法

[定理 3.1] (中国人剰余定理) [13]

$X_{rns(m_1, m_2, \dots, m_n)} = (\hat{x}_1, \hat{x}_2, \dots, \hat{x}_n)$  とする。 $(\hat{x}_1, \hat{x}_2, \dots, \hat{x}_n), \hat{x}_i \in \{0, 1, \dots, m_i - 1\}$  が与えられたとき  $X$  は次の式で計算できる。

$$X = \left( \sum_{i=1}^n M_i (\alpha_i \hat{x}_i)_{(\text{mod } m_i)} \right)_{(\text{mod } M)}$$

ここで  $M = \prod_{i=1}^n m_i, M_i = M/m_i, \alpha_i = (M_i^{-1})_{(\text{mod } m_i)}$  は、 $m_i$  に対する逆元。

[例 3.2] RNS(3,5,7) を考える。 $M = 3 \cdot 5 \cdot 7 = 105, M_1 = M/3 = 35, M_2 = M/5 = 21, M_3 = M/7 = 15, \alpha_1 = 2,$

(注1):  $x_i$  は 2 進数,  $\hat{x}_i$  は整数を表す。

表 3.1 RNS(3,4)-2 進変換の真理値表

$X_{rns(3,4)}$		Binary		Decimal
$\hat{x}_1$	$\hat{x}_2$	$x$		
0	0	0	0	0
0	0	0	1	9
0	0	1	0	6
0	0	1	1	3
0	1	0	0	4
0	1	0	1	1
0	1	1	0	10
0	1	1	1	7
1	0	0	0	8
1	0	0	1	5
1	0	1	0	2
1	0	1	1	11
1	1	0	0	$d$
1	1	0	1	$d$
1	1	1	0	$d$
1	1	1	1	$d$

$\alpha_2 = \alpha_3 = 1$ . よって,  $X = 2 \cdot 35\hat{x}_1 + 21\hat{x}_2 + 15\hat{x}_3$ .

ここで  $\alpha_1 = 2$  が  $m_1 = 3$  に対する逆元であることは,  $(2 \cdot 35)_{(mod\ 3)} = 70_{(mod\ 3)} = 1$  の計算で確認できる.

RNS-2 進変換は, 中国人剰余定理を用いる方法以外に RNS をまず Mixed RNS に変換し, それから 2 進数に変換する方法もある [14]. 本稿では, 中国人剰余定理を用いる方法のみを検討する.

### 3.2.1 1 個のメモリによる実現

RNS-2 進変換回路の最も単純な実現方法は, 1 個のメモリに RNS-2 進変換の真理値表を直接記憶させる方法である. 本方法では, 予めすべての RNS の入力組合せに対して, 中国人剰余定理 (以後 CRT と省略) を用いて, 対応する 2 進数を計算し, これをメモリに記憶させておく. 入力のビット長が小さいときは単純で高速である. しかし, 入力のビット長が増えると必要なメモリ量が増大するので, コンパクトな回路が必要なときは他の方法を検討する必要がある.

[定理 3.2] RNS-2 進変換を 1 個のメモリで実現した場合, 入力のビット長は  $\sum_{i=1}^n \lceil \log_2 m_i \rceil$  である. 出力のビット長は,  $\lceil \log_2 M \rceil$ , 但し,  $M = \prod_{i=1}^n m_i$ . よって, メモリ量は

$$\lceil \log_2 \prod_{i=1}^n m_i \rceil \cdot 2^{\sum_{i=1}^n \lceil \log_2 m_i \rceil} \text{ [bits]}$$

図 3.1(a) に RNS(3,5,7) から 2 進数への変換回路をメモリを用いて実現する方法を示す. メモリ量は,  $7 \cdot 2^8$  [bits] である.

### 3.2.2 乗算器とモジュロ加算器による実現

乗算器, モジュロ加算器により RNS-2 進変換回路を実現できる. CRT における乗算は定数倍なので, 乗算器を用いずに加算器の組合せで実現することもできる. また, メモリを用いてテーブル・ルックアップにより実現もできる. メモリを用いるときは,  $(mod\ M)$  の演算は予め計算してメモリに格納しておく.

[例 3.3] 図 3.1(b) に RNS(3,5,7)-2 進変換回路を定数倍の乗算とモジュロ加算器とで構成した例を示す. (例終り)

例 3.3 において,  $\hat{x}_1, \hat{x}_2, \hat{x}_3$  の取り得る範囲はそれぞれ  $[0, 2], [0, 3], [0, 6]$  であるので, 乗算器への入力はそれぞれ 2, 3, 3 ビットとなる.

$70\hat{x}_1$  の取り得る値は,  $\{0, 70, 140\}$  のいずれかであるが, CRT において  $Y = (70\hat{x}_1 + 21\hat{x}_2 + 15\hat{x}_3)_{(mod\ M)} =$

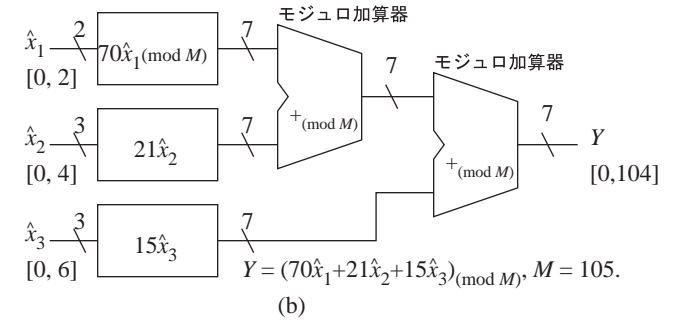
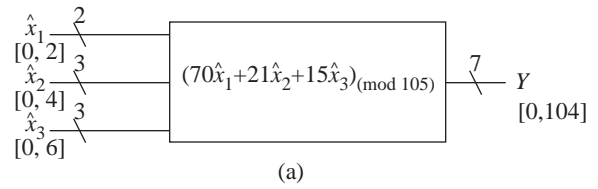


図 3.1 RNS-2 進変換回路

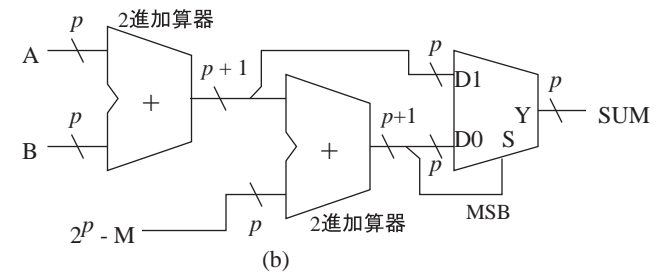
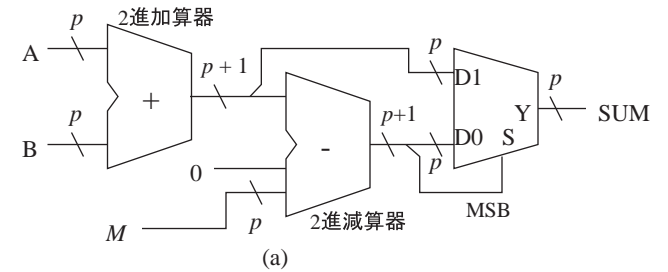


図 3.2 モジュロ M 加算器の構成

$((70\hat{x}_1)_{(mod\ M)} + 21\hat{x}_2)_{(mod\ M)} + 15\hat{x}_3)_{(mod\ M)}$  と変形できるので,  $70\hat{x}_1$  を実現する回路は  $(70\hat{x}_1)_{(mod\ M)}$  とし, 取り得る出力を  $\{0, 35, 70\}$  としても良い.

モジュロ M 加算器は, 2 進数の加算を行ってから M の剰余をとる. 7 ビット・モジュロ加算器の構成例を図 3.2(a) に示す. ただし, この回路では  $A + B < 2M$  という制約がある.  $A + B$  から M を減算し, それが負 (つまり  $A + B < M$ ) の場合  $A + B$  を出力とし, 正 (つまり  $A + B \geq M$ ) の場合  $A + B - M$  をマルチプレクサで切り換えて, モジュロ加算器の出力とする. 図 3.2(a) の最終段の減算器は, 図 3.2(b) の最終段のように  $2^p - M$  を加算することによって, 加算器に置換できる. 定数の減算であるので, 通常の減算器 (加算器) よりも回路規模は小さい.

変換回路を定数倍の乗算器とモジュロ加算器とで構成する場合, モジュロ加算器の遅延が大きいのが問題になる. 遅延が少ない回路の構成法を次節で提案する.

### 3.2.3 比較器が不要な実現法

3.2.2 で紹介した従来法では, 計算結果を M と比較し, 計算結果をそのまま出力するか, M を減算した結果を出力するかをマ

ルチプレクサで切り替えてモジュロ加算器を実現していた。

以下では、CRT の計算は、入力  $\hat{x}_1, \dots, \hat{x}_n$  の値だけで減算する値が判定できることを利用する。一方、減算する値の大きさと種類は、予めわかっているなのでその計算回路を並列に実現し、切り替えることでレイテンシの小さな回路を実現できる。

[例 3.4] 図 3.3 に RNS(3,5,7)-2 進変換回路の構成法を示す。(例終り)

図 3.3(a) は、三つの式  $70\hat{x}_1 + 21\hat{x}_2 + 15\hat{x}_3$ ,  $70\hat{x}_1 + 21\hat{x}_2 + 15\hat{x}_3 - 105$ ,  $70\hat{x}_1 + 21\hat{x}_2 + 15\hat{x}_3 - 210$  を並列に計算する。入力  $\hat{x}_1, \hat{x}_2$ , 及び  $\hat{x}_3$  の値によって切り換える回路構成を示す。選択回路の構成法については後述する。積和演算回路の出力は、すべて 7 ビットであることに注意されたい。

図 3.3(b) には、減算する値の 2 の補数を選択回路によって切り換えて最終段の加算器で減算する構成法を示す。本例では、 $128 - 105 = 23$ ,  $256 - 210 = 46$  と 6 ビットで表現できるのでマルチプレクサの出力は 6 ビットで十分である。また、2 進加算器の出力も下位 7 ビットのみが必要で、桁上げ出力は不要である。

ブロック図の構成の各ブロックをモジュールとして HDL (Hardware Description Language) で記述して回路実現する場合、論理合成システムやフィッター、実現するデバイスにより大きく性能が異なる。ブロックレベルでの遅延の議論だけでは、実際の回路の性能を必ずしも推し量れない場合も多い。よって、ブロック図の階層を様々に記述し、回路実現の選択の幅を増やし、所望の仕様に最も近い性能と面積の回路を選択可能にすることが大切である。

この考えの下では、一つのモジュールを構成するブロックのまとめ方や記述法も重要になる。図 3.3(c) には、図 3.3(b) の破線部を一つにまとめた構成を示す。また、図 3.3(d) には図 3.3(c) 全体を一つのブロックでまとめた構成を示す。図 3.3(a) ではこれをメモリで実現することを述べたが、図 3.3(d) では、これを一つのブロックで記述できることのみを述べている。これらの回路はメモリでも、FPGA 内部の論理部でも、乗算器と論理部との併用でも構成できる。

### 3.2.4 選択回路の構成法

RNS-2 進変換回路でモジュロ演算は、 $M$  の定数倍 ( $q$  倍) を減算 (その 2 の補数を加算) することに対応する。ここでは  $q$  を求める回路を選択回路と呼ぶ。

[定義 3.2] 以下の等式を満たす  $q$  を生成する回路を選択回路と呼ぶ。

$$\left(\sum_{i=1}^n M_i(\alpha_i \hat{x}_i)_{(\text{mod } m_i)}\right)_{(\text{mod } M)} = \sum_{i=1}^n M_i(\alpha_i \hat{x}_i) - qM.$$

選択回路の出力  $q$  について次の性質が成り立つ。

[性質 3.1] 出力  $q$  を 0 以上の整数とみなすと、入力値  $\hat{x}_1, \hat{x}_2, \dots, \hat{x}_n$  に関して単調増加関数となる。つまり、入力変数の値が増加すると出力値は等しいか増加する。

選択回路の構成を考える。CRT の計算でモジュロ演算を除いた部分は、積和演算である。これから  $0, M, 2M, \dots, qM$  のいずれかの値を減算した値を出力すればよい。

この関数を求めるには、全ての入力の組合せに対して CRT のモジュロ演算を取り除いた値を計算する。そして、その結果がどの数値の範囲に入るかを論理式で表せばよい。ここでは、比較器

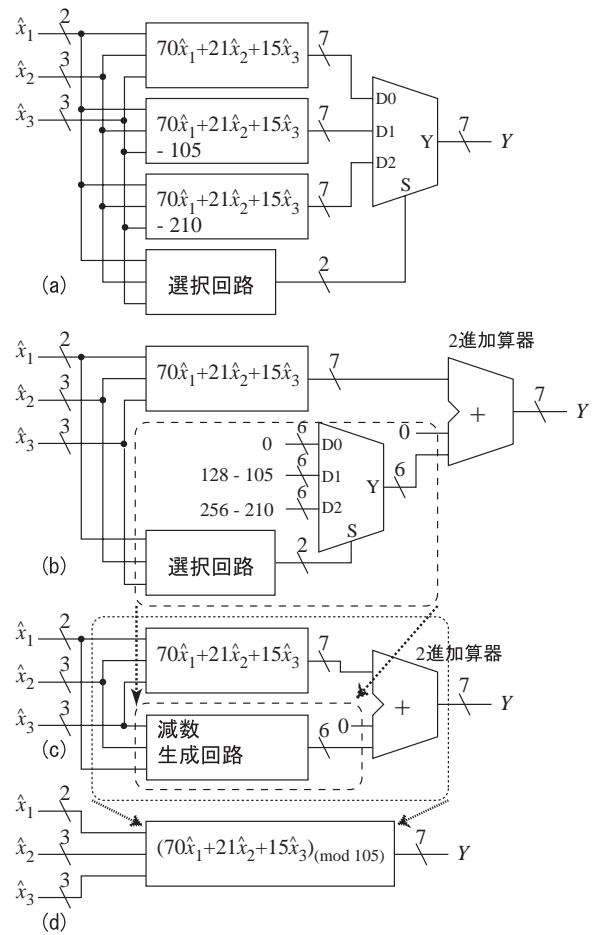


図 3.3 提案する RNS-2 進変換回路の概念図と AND, OR とで論理式を表現する方法を提案する。

[例 3.5] RNS(3,5,7)-2 進変換において、選択回路の出力を  $s_{105} = (s \geq 105)$ ,  $s_{210} = (s \geq 210)$ ,  $s = 70\hat{x}_1 + 21\hat{x}_2 + 15\hat{x}_3$  とするとこれらは以下の式で表現できる。

$$\begin{aligned} s_{105} &= (\hat{x}_1 \geq 2) \cdot (\hat{x}_3 \geq 5) \vee (\hat{x}_2 \geq 3) \cdot (\hat{x}_3 \geq 3) \\ &\quad \vee (\hat{x}_2 \geq 4) \cdot (\hat{x}_3 \geq 2) \vee (\hat{x}_1 \geq 1) \cdot (\hat{x}_3 \geq 3) \\ &\quad \vee (\hat{x}_1 \geq 1) \cdot (\hat{x}_2 \geq 1) \cdot (\hat{x}_3 \geq 1) \\ &\quad \vee (\hat{x}_1 \geq 1) \cdot (\hat{x}_2 \geq 2) \vee (\hat{x}_1 \geq 2), \\ s_{210} &= (\hat{x}_1 \geq 1) \cdot (\hat{x}_2 \geq 3) \cdot (\hat{x}_3 \geq 6) \\ &\quad \vee (\hat{x}_1 \geq 1) \cdot (\hat{x}_2 \geq 4) \cdot (\hat{x}_3 \geq 4) \\ &\quad \vee (\hat{x}_1 \geq 2) \cdot (\hat{x}_3 \geq 5) \vee (\hat{x}_1 \geq 2) \cdot (\hat{x}_2 \geq 1) \cdot (\hat{x}_3 \geq 3) \\ &\quad \vee (\hat{x}_1 \geq 2) \cdot (\hat{x}_2 \geq 2) \cdot (\hat{x}_3 \geq 2) \\ &\quad \vee (\hat{x}_1 \geq 2) \cdot (\hat{x}_2 \geq 3) \vee (\hat{x}_3 \geq 1) \\ &\quad \vee (\hat{x}_1 \geq 2) \cdot (\hat{x}_2 \geq 4), \end{aligned}$$

$$SEL = (s_{210}, \bar{s}_{210} \cdot s_{105}). \quad (\text{例終り})$$

[例 3.6] 例 3.5 の式  $s_{105}$  を求める方法を図 3.4 に示す。変数  $\hat{x}_1, \hat{x}_2, \hat{x}_3$  に値をそれぞれの変数の下限値から上限値までを 1 ずつ変化させて、順に  $(0, 0, 0)$  から  $(2, 4, 6)$  に対する  $70\hat{x}_1 + 21\hat{x}_2 + 15\hat{x}_3 \geq 105$  の値を計算する。  $(\hat{x}_1, \hat{x}_2, \hat{x}_3) = (0, 2, 5)$  の時に  $70\hat{x}_1 + 21\hat{x}_2 + 15\hat{x}_3 \geq 105$  が真 (T: True) となるので、これに対応する項として  $(\hat{x}_1 \geq 0) \cdot (\hat{x}_2 \geq 2) \cdot (\hat{x}_3 \geq 5)$  を生成する。ここで  $\hat{x}_1$  の取り得る値は  $[0, 2]$  であり  $(\hat{x}_1 \geq 0) = 1$

$\hat{x}_1$	$\hat{x}_2$	$\hat{x}_3$	$70\hat{x}_1+21\hat{x}_2+15\hat{x}_3 \geq 105$
0	0	0	F
0	0	1	F
⋮			
0	0	6	F
0	1	0	F
⋮			
0	1	6	F
0	2	0	F
⋮			
0	2	4	F
0	2	5	T
↓			$(\hat{x}_1 \geq 0)(\hat{x}_2 \geq 2)(\hat{x}_3 \geq 5)$ $= (\hat{x}_2 \geq 2)(\hat{x}_3 \geq 5)$
0	3	0	F
⋮			
0	3	2	F
0	3	3	T
↓			$(\hat{x}_2 \geq 3)(\hat{x}_3 \geq 3)$
0	4	0	F
0	4	1	F
0	4	2	T
↓			$(\hat{x}_2 \geq 4)(\hat{x}_3 \geq 2)$
1	0	0	F
⋮			
1	0	2	F
1	0	3	T
↓			$(\hat{x}_1 \geq 1)(\hat{x}_3 \geq 3)$
1	1	0	F
1	1	1	T
↓			$(\hat{x}_1 \geq 1)(\hat{x}_2 \geq 1)(\hat{x}_3 \geq 1)$
1	2	0	T
↓			$(\hat{x}_1 \geq 1)(\hat{x}_2 \geq 2)$
2	0	0	T
↓			$(\hat{x}_1 \geq 2)$

図 3.4 選択回路の論理式の生成法

であるので、 $(\hat{x}_2 \geq 2) \cdot (\hat{x}_3 \geq 5)$  と簡化できる。

$(\hat{x}_1, \hat{x}_2, \hat{x}_3) = (0, 2, 5)$  の次の値は  $(0, 2, 6)$  であるが、 $(0, 2, 5)$  の時に  $70\hat{x}_1 + 21\hat{x}_2 + 15\hat{x}_3 \geq 105$  であるから入力値の各要素の値が増えれば、必ず関係  $70\hat{x}_1 + 21\hat{x}_2 + 15\hat{x}_3 \geq 105$  を満たす。よって、次は  $(0, 3, 0)$  から同様の計算を再開し、 $(\hat{x}_2 \geq 3) \cdot (\hat{x}_3 \geq 3)$ 、 $(\hat{x}_2 \geq 4) \cdot (\hat{x}_3 \geq 2)$  を生成する。ここで、 $(\hat{x}_1, \hat{x}_2, \hat{x}_3) = (0, 4, 2)$  の次の値は、 $(1, 0, 0)$  となることに注意されたい。このようにして得られた項の論理和を求めて例 3.5 の式  $s_{105}$  を得る。(例終り)

[例 3.7] 例 3.5 の式を論理回路で実現した例を図 3.5 に示す。ここでは、PLA (Programmable Array Logic) [11] の入力デコーダ部の代わりに、比較器を用いている。●印は論理積 (AND)、×印は論理和 (OR) を表している。(例終り)

#### 4. FPGA 上への実装実験

FPGA (Field Programmable Gate Array) 上に RNS-2 進変換回路を実現し、その有効性を比較する。基数を入力すると図 3.3 の各方法に対応した Verilog HDL を生成するツールを現在、開発中である。

本稿では、このうち図 3.3(d) に対応する回路を生成するツール (図 4.1 参照) で得た Verilog HDL による実験結果を示す。本ツールでは、RNS-2 進変換回路を 2 段の AND-OR 論理式で表現し、それを Verilog HDL に変換したものをを用いている。RNS-2 進

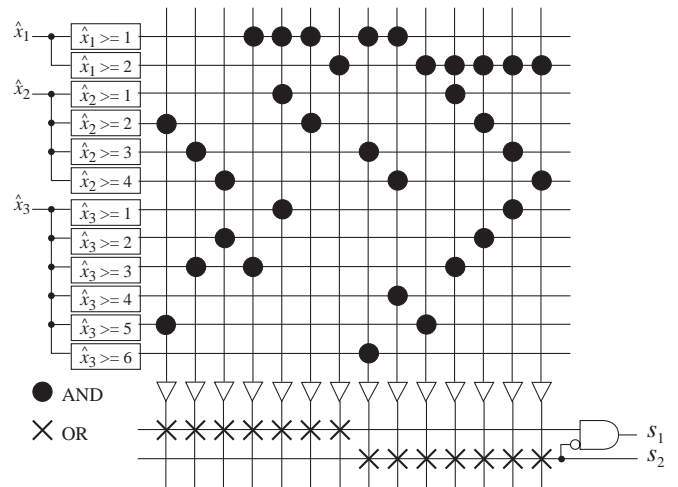


図 3.5 RNS-2 進変換回路の選択回路

変換では、don't care が存在するが、今回は don't care をすべて 0 と指定している。

Altera 社 FPGA (Field Programmable Gate Array) の Cyclone II で実現した実験結果を表 4.1 に示す。但し、一部の回路では Cyclone II には収納できずに Stratix II を利用した。全ての入力値に対し、得られた出力値が正しいことをシミュレーションによって確認している。

表 4.1 の Design 欄の数字は基数を示している。例えば、rms3\_5\_7 は、基数  $m_1 = 3$ 、 $m_2 = 5$ 、 $m_3 = 7$  を表す。Range 欄は  $M = m_1 \times m_2 \times \dots \times m_n$  で求められる。In と Out は入出力のビット数である。rms3\_5\_7 では入力ビットは  $2 + 3 + 3 = 8$  ビットなので  $In = 8$ 、出力の値域は  $[0, 104]$  であるので 7 ビットの 2 進数で表現でき  $Out = 7$  となる。

本実験では、組込みメモリや組込み乗算器は一切利用せず、すべて LE (Logic Element) で実現した。従って、LE 欄が回路の規模を表すと考えられる。Stratix II の場合は ALUT の個数を示す。

Delay 欄には、入力端子から出力端子までの最大遅延を示す。

表 4.1 の Range と LE 数との関係を図 4.2 に示す。このグラフでは Cyclone II を利用した実現のみを示している。Range と LE はほぼ比例すると考えられるが、基数の選び方によって回路規模が大きく異なる場合がある。例えば rms8\_19\_27 と rms7\_19\_31 の Range の値はほぼ同じであるが LE 数は、2639 と 5793 と大きく異なる。また、rms19\_27\_32 の Range は rms16\_17\_31 の Range の 2 倍程度であるが LE 数は 1.3 倍に留まっている。

遅延についても Range にほぼ比例している。

この実験から、基数の選び方が重要であることがわかる。

#### 5. まとめ

本稿では、RNS-2 進変換器の設計法について述べた。読みやすさのために、RNS(3,5,7) を例にとって実現法を説明した。また、提案手法の一部を FPGA 上へ実装する実験結果を示した。この実験から基数の選び方が回路規模や遅延に影響を及ぼすことがわかった。

今後の課題としては、他の方法との比較、基数をどのように選べば良い回路が生成できるかの検討がある。

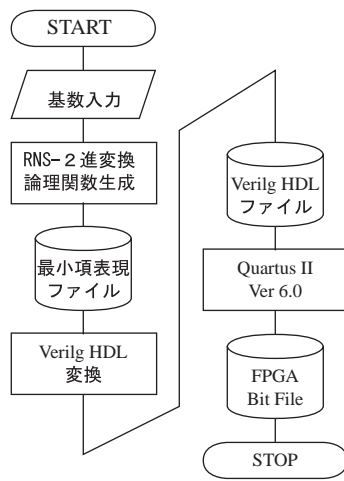


図 4.1 RNS-2 進変換回路の合成システム.

表 4.1 RNS-2 進変換回路を Altera FPGA 上に実装したときのハードウェア量と性能

Design	Range	In	Out	LE(ALUT)	Delay[ns]	FPGA
rns3_5_7	105	8	7	130	13.4	A
rns5_7_8	280	9	9	223	16.3	A
rns5_8_13	520	10	10	361	17.0	A
rns3_11_16	528	10	10	335	18.3	A
rns7_9_10	630	11	10	901	21.3	A
rns5_13_16	1040	11	11	559	18.6	A
rns3_4_7_13	1092	11	11	1288	21.7	A
rns8_11_13	1144	11	11	1249	20.7	A
rns4_5_7_9	1260	12	11	1714	24.1	A
rns11_13_15	2145	12	12	2907	26.6	A
rns3_5_11_13	2145	13	12	3600	31.3	A
rns5_6_7_11	2310	13	12	3044	25.1	A
rns8_19_27	4104	13	13	2639	24.5	A
rns7_19_31	4123	13	13	5793	29.0	A
rns2_11_13_15	4290	13	13	5173	27.8	A
rns3_7_13_16	4368	14	13	4114	32.3	A
rns15_16_19	4560	13	13	5534	28.7	A
rns16_19_27	8208	14	14	4191	28.3	A
rns16_17_31	8432	14	14	9193	32.1	A
rns4_11_13_15	8580	14	14	10212	36.8	A
rns7_8_11_15	9240	14	14	11488	41.5	A
rns19_27_32	16416	15	15	5513	29.1	A
rns17_31_32	16864	15	15	(11314)	48.2	B
rns8_11_13_15	17160	15	15	(15455)	50.5	B

A: Cyclone II (EP2C35F672C6)

B: Stratix II (EP2S180F1508C5)

#### 謝辞

本研究は、一部、文部科学省・科学研究費補助金、文部科学省・知的クラスター創成事業（第2期）の補助金による。FPGAの実装実験は、明治大学基礎理工学専攻の清水大和氏による。九州工業大学情報工学部松浦宗寛氏には、論文作成時に貴重なご意見を頂いた。

#### 文献

[1] M. Ciet, M. Neve, E. Peeter, and J.-T. Quisquater, "Parallel FPGA implementation of RSA with residue number systems, — Can side-

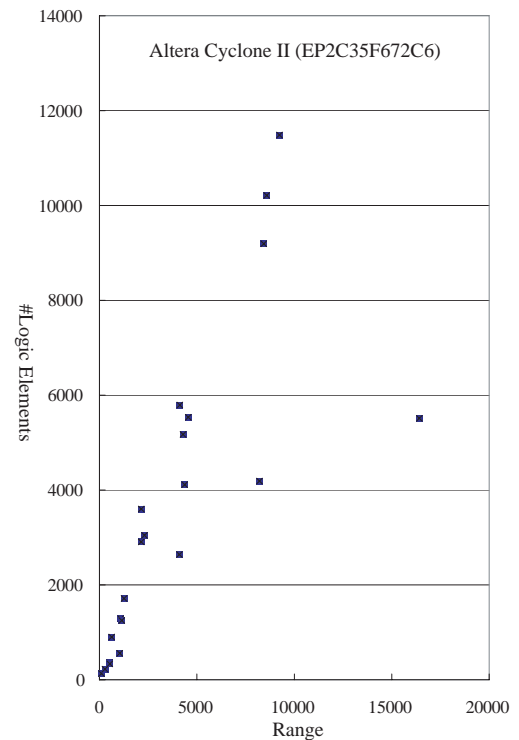


図 4.2 Range に対する Logic Element 数の変化

channel threats be avoided? —," MIDWEST 2003, April 2003.

[2] R. Conway and J. Nelson, "Fast converter for 3 moduli RNS using new property of CRT," *IEEE Trans. Comput.*, Vol. 48, No. 8, Aug. 1999.

[3] P. V. Ananda Mohan, *Residue Number Systems: Algorithms and Architectures*, Kluwer Academic Publ. (2002).

[4] J. P. Deschamps, G. J. A. Bioul, and G. D. Sutter, *Synthesis of Arithmetic Circuits, FPGA, ASIC, and Embedded Systems*, Wiley, March 2006.

[5] M. Honda, M. Kameyama, and T. Higuchi, "Multiple-valued VLSI image processor based on residue arithmetic and its evaluation," *IE-ICE Trans. Electron.*, Vol. E76-C, No. 3, pp. 455-462, March 1993.

[6] C. H. Huang, "A fully parallel mixed-radix conversion algorithm for residue number applications," *IEEE Trans. Comput.*, vol. 32, pp. 398-402, 1983.

[7] K. Ishida, N. Homma, T. Aoki, and T. Higuchi, "Design and verification of parallel multipliers using arithmetic description language: ARITH," *34th International Symposium on Multiple-Valued Logic*, Toronto, Canada, May 2004, pp.334-339.

[8] M.A.Soderstrand, W.K.Jenkins, G.A.Jullien and F.J.Taylor Ed., "A new scaling algorithm in symmetric residue number system based on multiple-valued logic in residue number system arithmetic," in *Modern Applications in Digital Signal Processing*, IEEE Press, 1986.

[9] I. Koren, *Computer Arithmetic Algorithms, 2nd Edition*, A. K. Peters, Natick, MA, 2002.

[10] D. Olson, and K. W. Current, "Hardware implementation of supplementary symmetrical logic circuit structure concepts," *30th IEEE International Symposium on Multiple-Valued Logic* Portland, Oregon, May 23-25, 2000.

[11] T. Sasao, *Switching Theory for Logic Synthesis*, Kluwer Academic Publishers, 1999.

[12] 佐藤清, 豊嶋久道, "中国人の剰余定理を実現する高速ハードウェアアルゴリズム," *信学論* Vol. J78-A, No.9, Sep. 1995.

[13] SunTzu, "孫子算経," Third-century AD.

[14] F. J. Taylor, "Residue arithmetic: A tutorial with examples," *IEEE Computer*, pp. 50-62, Vol. 17, May 1984.