社団法人　電子情報通信学会
THE INSTITUTE OF ELECTRONICS,
INFORMATION AND COMMUNICATION ENGINEERS

信学技報
TECHNICAL REPORT OF IEICE.

算術分解を用いた基数変換回路の構成法

井口　幸洋† 　笹尾　勤†† 　松浦　宗寛††
† 明治大学 理工学部 情報科学科
†† 九州工業大学 情報工学部 電子情報工学科

あらまし　デジタル信号処理用の算術演算回路などでは, 基数が 2 以外の論理回路を用いて, 高速かつコンパクトな回路を実現する場合がある. この場合, 基数変換回路が必要であるが, 一般には複雑な回路となる. 本論文では, p 進数を 2 進数に変換する基数変換回路の構成法を提案する. 本構成法では, 算術分解と呼ぶ新しい方法を用いる. また, FPGA(Field Programmable Gate Array) 上に基数変改回路を実現したときのハードウェア量と性能の比較を行う.

キーワード　基数変換, 多値論理, LUT カスケード.

Design of Radix Converters Using Arithmetic Decomposition

Yukihiro IGUCHI†, Tsutomu SASAO††, and Munehiro MATSUURA††
† Department of Computer Scienece, Meiji University
†† Department of Computer Scienece and Electronics, Kyushu Institute of Technology

Abstract　In arithmetic circuits for digital signal processing, radixes other than two are often used to make circuits faster. In such cases, radix converters are necessary. However, in general, radix converters tend to be complex. This paper considers design methods for p-nary to binary converters. It introduces a new design technique called arithmetic decomposition. It also compares the amount of hardware and performance of radix converters implemented on FPGAs.

Key words　Radix converter, Multiple valued logic, LUT cascades, FPGA.

1. はじめに

デジタルシステムにおける算術演算は, 通常 2 進数を用いる. しかし, デジタル信号処理などでは高速演算処理のため, p 進数 $(p > 2)$ を用いる場合がある [1], [4]. この場合, 問題となるのは 2 進数と p 進数との間の変換操作, 即ち **基数変換**である.

p 進数を q 進数に変換する方法として, 種々の方法が知られているが, いずれも計算量が多い. 特に, 基数変換を組合せ論理回路で実行する場合, 回路は非常に複雑となる [5]. 基数変換表をメモリなどに記憶し, 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, 必要なメモリ容量も指数関数的に増加する. 文献 [8] では, 2 進 3 進, 3 進 2 進, 2 進 10 進, 10 進 2 進変換を LUT カスケード [7] で実現している. また, 文献 [10] では, WS 関数 (Weighted-Sum function) の概念に基づき, LUT カスケードを用いた基数変換回路の構成法が提案されている.

本稿では, LUT カスケード [7] と算術分解 [9] とを用いて $p(> 2)$ 進数から 2 進数に変換する基数変換回路を能率良く実現する方法を述べる. FPGA (Field Programmable Gate Array) 上での実現も示す. また, LUT カスケードによる実現と提案方法との比較を計算機実験に基づき行う.

本稿は, 著者らによる発表 [2] に理論的検討を加筆し, 修正を行ったものである.

2. 基数変換回路

2.1 基数変換

［定義 2.1］　n 桁の p 進数を $\vec{x} = (x_{n-1}, x_{n-2}, \ldots, x_0)_p$, m 桁の q 進数を $\vec{y} = (y_{m-1}, y_{m-2}, \ldots, y_0)_q$ とする. $\vec{x}$ が与えられたとき, 関係

$$\sum_{i=0}^{n-1} x_i p^i = \sum_{j=0}^{m-1} y_j q^j \tag{2.1}$$

を満たす $\vec{y}$ を求める操作を**基数変換**という. ここで, $x_i \in P, y_j \in Q, P = \{0, 1, \ldots, p-1\}, Q = \{0, 1, \ldots, q-1\}$ である.

表 2.1　3 進 2 進変換の真理値表

Binary-coded-ternary				Ternary		Binary				Decimal
z_3	z_2	z_1	z_0	x_1	x_0	y_3	y_2	y_1	y_0	
0	0	0	0	0	0	0	0	0	0	0
0	0	0	1	0	1	0	0	0	1	1
0	0	1	0	0	2	0	0	1	0	2
0	1	0	0	1	0	0	0	1	1	3
0	1	0	1	1	1	0	1	0	0	4
0	1	1	0	1	2	0	1	0	1	5
1	0	0	0	2	0	0	1	1	0	6
1	0	0	1	2	1	0	1	1	1	7
1	0	1	0	2	2	1	0	0	0	8

$\vec{y} = (y_{m-1}, y_{m-2}, \ldots, y_0)_q$, $y_i \in \{0, 1\}$ を p 進数から 2 進数への基数変換（以後, 簡単のために p 進 2 進変換と略記）の出力関数とする. p が素数の場合は, y_i は, 入力のすべての変数 $x_i (i = 0, 1, \ldots, n-1)$ に依存する. また, p が 2 のべき乗でないとき, 出力関数は不完全定義の関数となる. 従って, 未使用の入力組合せが存在するが, 通常は, 未使用入力に対する出力には 0 を割り当てる.

［例 2.1］　3 進 2 進変換を考える. 3 進数を表現するために 2 進化 3 進数を使うこととする. つまり, 0 は (00), 1 は (01), 2 は (10) で表す. (11) は未使用コードであることに注意されたい. 表 2.1 は, 2 桁の 3 進数から 2 進数への変換のための真理値表である. (11) は未定義入力であり, 対応する出力はドント・ケアとなる. また, 2 進化 3 進数で表された入力は, $\vec{z} = (z_3, z_2, z_1, z_0)$ と表記する. 3 進数の表記は, $\vec{x} = (x_1, x_0)$ とする. 出力である 2 進数は, $\vec{y} = (y_3, y_2, y_1, y_0)$ である.　(例終り)

2.2 直接的な実現法

基数変換を実装する直接的な方法は, 数式 (2.1) から y_i を求め, それを実現すれば良い.

2.2.1 ランダムロジックによる実現

［例 2.2］　8 桁の 3 進 2 進変換 (**8ter2bin**) を考えてみる. この場合, 以下の数式を実現すればよい.

$$3^7 x_7 + 3^6 x_6 + 3^5 x_5 + 3^4 x_4 + 3^3 x_3 + 3^2 x_2 + 3^1 x_1 + 3^0 x_0. \tag{2.2}$$

図 2.1 に, 論理合成プログラムによって生成した **8ter2bin** の回

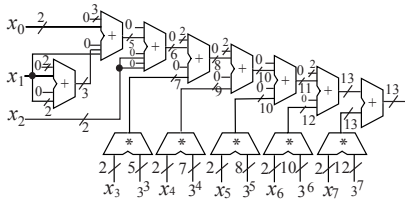

図 2.1　8 桁 3 進 2 進変換回路: 直接的な実現法

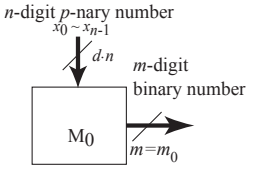

図 2.2　n 桁 p 進 2 進変換: 1 個のメモリによる実現

路図を示す. 3^2x_2 は, $8x_2 + x_2$ で, そして, 3^1x_1 は, $2x_1 + x_1$ で実現しているが, 3^ix_i $(2 < i \leq 7)$ は乗算器で実現している. 直列接続された加算器で加算を行っている. 乗算の一方の係数は定数であるので, この乗算は加算器で置換することもできる. (例終り)

2.2.2　1 個のメモリによる実現

最も単純な実現法は, 1 個のメモリに基数変換の表を記憶させることである.

n 桁の p 進数が表現する数値は, 0 から $p^n - 1$ である. これを 2 進数では, $m = \lceil \log_2(p^n - 1) \rceil$ ビットで表現できる. 基数変換回路の入力の各桁が, 2 進化 p 進数で表されているときの各桁のビット数を d とする. この時, $d = \lceil \log_2 p \rceil$ である. 入力の各桁が 2 進化 p 進数で表された n 桁の p 進数を m 桁の 2 進数に表現する回路を考える. この回路を 1 個のメモリで実現するのに必要なメモリ量 (bit 数) を $SIZE(n, p)$ で表す. このとき, $SIZE(n, p) = 2^{\lceil \log_2 p \rceil \cdot n} \cdot \lceil \log_2(p^n - 1) \rceil$ が成立する.

n 桁の p 進数を 2 進化 p 進数で入力し, 2 進数で出力する基数変換回路を図 2.2 に示す.

[例 2.3]　8 桁の 3 進 2 進変換回路が表現する値の範囲は, $[0, 3^8 - 1] = [0, 6560]$ である. 変換後の 2 進数は, $m = \lceil \log_2 6560 \rceil = 13$ 桁となる. これを 1 個のメモリで実現すると $SIZE(8, 3) = 2^{2 \cdot 8} \cdot 13 = 851,968$ となる. (例終り)

入力の桁数が増えると指数関数的にメモリ量が増大するので, 多桁の基数変換を 1 個のメモリで実現するのは実用的ではない.

3.　LUT カスケード実現と WS 関数

p 進 2 進変換の直接的な実現法では, 入力桁の増加に伴う信号遅延やハードウェア量の増大が問題となる. LUT を直列多段に接続する方法（LUT カスケード）で基数変換回路を構成する方法が提案されている [8]. これは, 出力の桁を複数のグループに分割した後, 関数分解の方法を用いて回路を構成している. まずは, 出力を分割しないで構成する方法を検討する.

3.1　LUT カスケードを用いた基数変換回路の実現

メモリ 1 個で基数変換回路を実現するのは単純だが, メモリ量が大きくなりすぎる. そこで, メモリを複数個カスケード接続（直列接続）することで総メモリ量を削減することを考える. まず, 図 3.1 に示すように 2 個のメモリ M_0, M_1 のカスケードの構成法を検討する. 図 3.1 では, 2 進化 p 進数の下位 k 桁が, メモリ M_0 の入力に接続されている. M_0 の出力を, 次段のメモリ M_1 の入力に, 残りの入力に上位 $n-k$ 桁の 2 進化 p 進数を入力する. M_1 の出力には, 変換済みの 2 進数を出力することを考える. 関数分解 [6] を用いれば, このような実現が可能かどうかを判定できる.

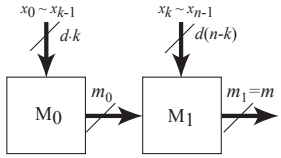

図 3.1　n 桁 p 進 2 進変換: LUT カスケード法による実現

表 3.1　4 桁の 3 進数 2 進数変換の基本分解表.

		0	0	0	1	1	1	2	2	2	X_1
X_3	X_2	0	1	2	0	1	2	0	1	2	X_0
0	0	0	1	2	3	4	5	6	7	8	
0	1	9	10	11	12	13	14	15	16	17	
0	2	18	19	20	21	22	23	24	25	26	
1	0	27	28	29	30	31	32	33	34	35	
1	1	36	37	38	39	40	41	42	43	44	
1	2	45	46	47	48	49	50	51	52	53	
2	0	54	55	56	57	58	59	60	61	62	
2	1	63	64	65	66	67	68	69	70	71	
2	2	72	73	74	75	76	77	78	79	80	

[定義 3.1]　関数 $f(\vec{X}) : X^n \to Y$, $X = \{0, 1, \ldots, p-1\}$, $Y = \{0, 1, \ldots, r\}$ が与えれられたとき, $(\vec{X}_H, \vec{X}_L)$, $\vec{X}_H = (X_0, X_1, \ldots, X_{k-1})$, $\vec{X}_L = (X_k, X_{k+1}, \ldots, X_{n-1})$ を $\vec{X}$ の分割とし, X の元の $\vec{X}_L$ への割当ての全ての組合せを列のラベルとしてもち, X の元の $\vec{X}_H$ への割当ての全ての組合せを行のラベルとしてもち, 対応する値が $f(\vec{X}_H, \vec{X}_L)$ の値であるような表を f の**分解表**という. f の分解表でラベルが左から右に, 上から下に移るに従い, $\vec{X}$ を p 進数としてみたときの値が増加するように並べたものを**基本分解表**という. 分解表の異なる列パターンの個数を**列複雑度**という.

通常の分解表では, 列ラベルや行ラベルに $\vec{X}$ の変数を任意の順に割当てれるが, 基本分解表では行のラベルに $\vec{X}_H = (X_0, X_1, \ldots, X_{k-1})$, 列のラベルに $\vec{X}_L = (X_k, X_{k+1}, \ldots, X_n)$ を順に割当てるという制限が付いている [8].

[補題 3.1]　p 進 2 進変換を表現する関数 $f(\vec{X})$ の基本分解表の列複雑度は p^k である, ここで, $(\vec{X}_H, \vec{X}_L)$ は $\vec{X}$ の分解であり, $\vec{X}_H$ の変数の個数を k とする.

[例 3.1]　4 桁の 3 進数を 7 桁の 2 進数に変換する場合の基本分解表を表 3.1 に示す. (例終り)

3.2　WS 関数

WS 関数 (weighted sum function) は, 基数変換回路, ビット計数回路, 畳み込み演算の数学的モデルである [9], [10]. 本節では, WS 関数の性質のいくつかを明らかにして, それを用いて基数変換回路の構成法を検討する.

[定義 3.2]　n 入力の WS 関数は, 次のように定義される [10]:

$$WS(\vec{x}) = \sum_{i=0}^{n-1} w_i x_i, \tag{3.1}$$

ここで $\vec{x} = (x_{n-1}, x_{n-2}, \ldots, x_0)$ を**入力ベクトル**, $\vec{w} = (w_{n-1}, w_{n-2}, \ldots, w_0)$ を**重みベクトル**と呼ぶ. 各要素は整数である.

本稿では, 2 進化 p 進数で入力された p 進数から 2 進数への基数変換の表現に WS 関数を用いる. これ以後特に断らない限り, w_i は 1 以上の整数, x_i は 0 以上の整数とする.

[定義 3.3]　WS 関数 $WS(\vec{x}) = \sum_{i=0}^{n-1} w_i x_i$, $w_i \geq 0$, $x_i \in \{0, 1, \ldots, p-1\}, p \geq 2$ が取り得る最小値を MIN_n, 最大値を MAX_n とする.

WS 関数は, 全ての入力が $x_i = 0$ の時, 最小値 $MIN_n = WS(0, 0, \ldots, 0) = 0$ をとる. また, 全ての入力が $x_i = p - 1$ の時, 最大値 $MAX_n = WS(p-1, p-1, \ldots, p-1) = \sum_{i=0}^{n-1} \{w_i \cdot (p-1)\} = (p-1) \sum_{i=0}^{n-1} w_i$ をとる.

[定義 3.4]　整数 $i, j, (i < j)$ に対し, 整数の集合 $\{i, i+1, \ldots, j\}$ を $[i, j]$ と表記する.

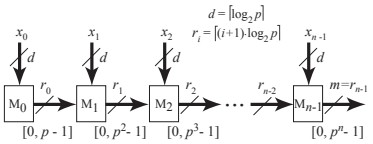

図 3.2　入力毎に 1 つの LUT を割り当てて実現した基数変換回路

次に WS 関数の値域を検討する.

［定義 3.5］ 関数 $f(x)$ の値域を $Range(f(x))$ と記す.

［例 3.2］ $WS_1(\vec{x})= x_0 + 3x_1$, $x_i \in \{0,1,2\}$ とする. $Range(x_0)= \{0,1,2\}$, $Range(3x_1) = \{0,3,6\}$ である. 従って, $Range(WS_1(\vec{x}))= \{0,1,2,3,4,5,6,7,8\} = [0,8]$. 一方, $WS_2(\vec{x})= x_0 + 4x_1$, $x_i \in \{0,1,2\}$ の時, $Range(x_0)= \{0,1,2\}$, $Range(4x_1) = \{0,4,8\}$ である. 従って, $Range(WS_2(\vec{x}))= \{0,1,2,4,5,6,8,9,10\} \neq [0,10]$. また, $WS_3(\vec{x})= x_0 + 2x_1$, $x_i \in \{0,1,2\}$ とする. $Range(x_0)= \{0,1,2\}$, $Range(2x_1)= \{0,2,4\}$ である. 従って, $Range(WS_3(\vec{x}))= \{0,1,2,3,4,5,6\}= [0,6]$. (例終り)

例 3.2 から明らかなように, w_i の値の組合せによって $Range(WS(\vec{x})) = [0, MAX_n]$ が成立する場合と成立しない場合とがある. $WS_2(\vec{x}) = x_0 + 4x_1$ の場合, $MIN = 0$, $MAX = 2\cdot(1+4) = 10$ である. ところで, x_i の取り得る値は, 3 通りであるから, $WS_2(\vec{x})$ の取り得る値は高々$3^2 = 9$ 通りである. 従って, $Range(WS_2(\vec{x})) \neq [0,10]$ である.

［補題 3.2］ WS 関数, $WS(\vec{x}) = \sum_{i=0}^{n-1} w_i x_i$, $w_i \geqq 0$, $x_i \in \{0,1,\ldots,p-1\}$,$p \geqq 2$ が取り得る値の種類は, 高々p^n 通りである.

次に, $Range(WS(\vec{x})) = [0, MAX_n]$ であるための必要十分条件について検討する.

［定理 3.1］ WS 関数 $WS(\vec{x}) = \sum_{i=0}^{n-1} w_i x_i$, $w_i \geqq 0$, $x_i \in \{0,1,\ldots,p-1\}$,$p \geqq 2$ において, $Range(WS(\vec{x})) = [0, MAX_n]$ が成立するための必要十分条件は, $w_0 = 1$ かつ $w_i \leqq MAX_{i-1} + 1$ である.

［系 3.1］ $WS(\vec{x}) = \sum_{i=0}^{n-1} p^i x_i$ は, $[0, p^n - 1]$ の全ての値を取り得る.

次に, 総メモリ量が最小な LUT カスケードで基数変換回路を実現する方法を考える. 図 3.2 に各入力 x_i に 1 つの LUT を割り当てて実現した n 桁 p 進数から q 進数への変換回路を示す. この場合、i 番目 , $(0 \leqq i \leqq n-1)$ の LUT の出力値は, $[0, p^{i+1} - 1]$ であり, $r_i = \lceil (i+1)\log_2 p \rceil$ 本の出力線が必要かつ十分である. LUT と LUT との間は, $n-1$ 個あるので隣接する LUT を併合して一つの LUT にするかどうかの場合の数は, 2^{n-1} 通り存在する. 入力変数 x_i は, $d = \lceil log_2 p \rceil$ 本の 2 値入力線として LUT に接続されているが, これを 1 つの LUT の入力とするという制約の元では, この 2^{n-1} 通りの組合せが全ての LUT カスケードの実現法を示している.

［定理 3.2］ 変数順序を固定した場合, n 桁の基数変換回路の最小 LUT カスケード実現を求めるには, 高々2^{n-1} 個の組合せを考慮すれば十分である.

［例 3.3］ 図 3.3 に 4 入力 3 進 2 進基数変換回路の全ての LUT カスケード実現を示す. 全部で (0) ～ (7) の $2^{4-1} = 8$ 通りの実現がある. 各 LUT 内部に記載した数値はその LUT のメモリ量である. (0) は, 4 個の LUT からなる実現であり, 総メモリ量は 1288 である. (0) の左 2 つの LUT を 1 つの LUT で実現したものが (1) の実現である. (7) は全ての LUT を併合し, 1 つの LUT で実現したものである. 8 通りの実現の内, (3) の実現が, 総メモリ量が最小の実現となる. (例終り)

次に基数変換を実現する LUT カスケードで, 併合可能な LUT を検出するための補題を示す.

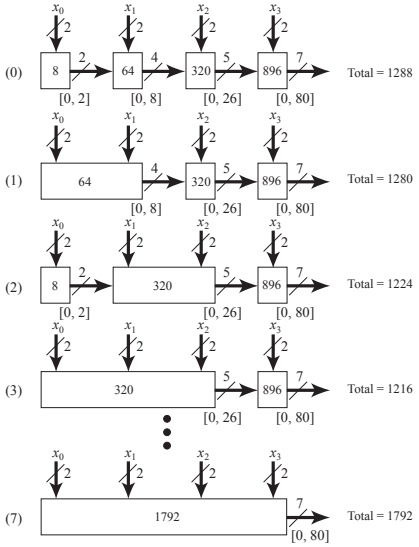

図 3.3　4ter2bin の全ての LUT カスケード実現

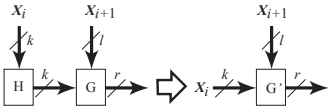

図 3.4　併合可能な LUT

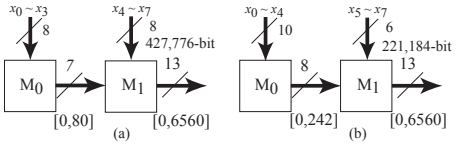

図 3.5　8 桁 3 進 2 進変換の LUT カスケード実現

［補題 3.3］ 図 3.4 の LUT カスケードで H は k 入力 k 出力である. この場合, 最小性を失わずに, LUT H と G とを併合し G' とすることが可能である.

補題 3.3 を用いると LUT カスケードにおいて, 併合可能な LUT を検出でき, 考慮すべき LUT カスケード数を削減できる. 例 3.3 の場合, (0) の LUT カスケードにおいて, まず左 2 つの LUT は併合可能であり, (1) の LUT カスケードが求まる. 次に (1) の左 2 つの LUT を併合し, (3) の LUT カスケードを得る. LUT 数は 2 個であり, LUT と LUT との間は 1 個のみであるので, これを併合するか否かの二つの場合みを検討すればよい.

以下に基数変換回路を実現する総メモリ量最小の LUT カスケードを合成するアルゴリズムを示す.

［アルゴリズム 3.1］

（1） 入力 x_i, $(0 \leqq i \leqq n-1)$ に対して 1 つの LUT を対応させた LUT カスケードで基数変換回路を構成する.

（2） 補題 3.3 を適用し, 入力数と出力数が同じ LUT は後に続く LUT と併合し, LUT のデータを書き換える. 本ステップを適用可能な限り繰り返す. 適用可能でなくなったら次へ

（3） 得られた LUT カスケードの LUT 数を s 個とする. 隣り合う LUT を併合するかしないかの組合せは, 2^{s-1} 通りなので, この全ての組合せの中から総メモリ量最小のものを選ぶ.

［例 3.4］ 例 2.3 の 8 桁 3 進 2 進変換回路を LUT カスケードで実現した例を図 3.5(a)(b) に示す. 1 個のメモリで実現した場合は, 851, 968 ビット必要だったメモリ量が, (a) の実現では, 427, 776 ビット, (b) では 221, 184 ビットのメモリ量で実現できる. (例終り)

出力を複数のグループに分割し, 複数の LUT カスケードで基数変換回路を実現する方法も提案されている [8]. この方法では, 二分決定グラフを用いた関数分解法を利用している. 以下に, 出

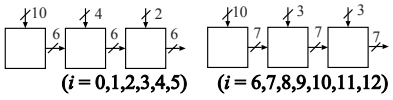

図 3.6　出力を 2 グループに分割して LUT カスケードで実現した 8 桁 3 進 2 進変換回路

力分割に基づいた構成例を示す.

［例 3.5］　[8] 8 桁の 3 進 2 進変換回路 **(8ter2bin)** を出力の分割に基づく方法で実現する. **don't care** 入力に対する出力値を 0 として求める. 列複雑度は $3^8 = 6551$ であるので 1 本のカスケードで実現すると, $\lceil \log_2 6551 \rceil + 1 = 14$ 入力の LUT を用いなければならず, これはかなり大きい. そこで, メモリ量を削減するために出力を 2 つのグループに分割する.

10 入力の LUT を用いることにすると, 図 3.6 で示される基数変換回路を得られる. この実現では, 上位 7 ビットと下位 8 ビットを別々の LUT カスケードで実現している. メモリの総量は, $2^{10}(7 \cdot 3 + 6 \cdot 2) + 2^8 \cdot 6 = 35,328$ ビットである.　　(例終り)

[8] の方法では, 多桁入力の基数変換回路の最適解を得るためには, 大量の計算を必要とするのに対し, アルゴリズム 3.1 の手法では複雑な計算を必要としないのが特長である. 次に, 算術分解 [9] を用いて, 更にコンパクトな基数変換回路を構成する方法を提案する.

4.　算術分解に基づいた基数変換回路の構成法

4.1　WS 関数の算術分解

前節で LUT カスケードを用いて基数変換回路を実現する方法について述べた. ここでは算術分解 [9] を用いた基数変換回路の構成法を提案する.

［定理 4.1］　WS 関数は, 以下のように 2 個の WS 関数の和に分解可能である:

$$WS(\vec{x}) = \sum_{i=0}^{n-1} w_i x_i = \alpha WS_A(\vec{x}) + WS_B(\vec{x}), \qquad (4.1)$$

ここで $WS_A(\vec{x}) = \sum_{i=0}^{n-1} a_i x_i$, $WS_B(\vec{x}) = \sum_{i=0}^{n-1} b_i x_i$, α は整数. (4.1) を $WS(\vec{x})$ の**算術分解**といい, α を **分解係数**と呼ぶ. α には, $2 \leqq \alpha < MAX_n$ の任意の整数を選ぶことができる. また, $WS_B(\vec{X})$ の取り得る最大値を MAX_B とする. 分解係数 α は, 必ずしも $MAX_B < \alpha$ を満足する必要は無いが, 本稿では $MAX_B < \alpha$ である時のみを検討する.

次に基数変換を表現する WS 関数を算術分解して得られる WS 関数の性質を考える.

［定理 4.2］　基数変換を表現する $WS(\vec{x})$ の算術分解 $WS(\vec{X}) = \sum_{i=0}^{n-1} w_i x_i = \alpha WS_A(\vec{x}) + WS_B(\vec{x})$ を考える. ここで, α は, $2 \leqq \alpha < p^n - 1$ を満たす整数, $WS_A(\vec{x}) = \sum_{i=0}^{n-1} w_i^A x_i$, $WS_B(\vec{x}) = \sum_{i=0}^{n-1} w_i^B x_i$ とする. この時, $Range(WS_B(\vec{X})) = [0, (p-1)\sum_{i=0}^{n-1} w_i^B]$ となる.

4.2　異なる分解係数による算術分解

基数変換は, WS 関数として表現できる. よって, 様々な分解係数 α を用いて算術分解を行えば, 様々な基数変換回路を実現できる. 本節では, 分解係数が 2^k の場合と p^k の場合の二つの場合を検討する.

分解係数が 2^k の場合: 基数変換回路は, 次のように実現できる:

$$WS(\vec{x}) = 2^k WS_A(\vec{x}) + WS_B(\vec{x}).$$

2^k の乗算は, シフト操作 (信号線の接続) で実現できるので, この回路は, コンパクトに実現可能である. しかし, WS_B はすべての入力変数に依存するので, 回路全体のハードウェア量は, 比較的大きいと予想される.

分解係数が p^k の場合: 基数変換回路は, 次のように実現できる:

表 4.1　3 のべき数の算術分解係数

		Decomposition Coefficients	
i	3^i	$\alpha = 2^6 = 64$	$\alpha = 3^4 = 81$
0	1	$64 \times 0 + 1$	$81 \times 0 + 1$
1	3	$64 \times 0 + 3$	$81 \times 0 + 3$
2	9	$64 \times 0 + 9$	$81 \times 0 + 9$
3	27	$64 \times 0 + 27$	$81 \times 0 + 27$
4	81	$64 \times 1 + 17$	$81 \times 1 + 0$
5	243	$64 \times 3 + 51$	$81 \times 3 + 0$
6	729	$64 \times 11 + 25$	$81 \times 9 + 0$
7	2187	$64 \times 34 + 11$	$81 \times 27 + 0$

$$WS(\vec{x}) = p^k WS_A(\vec{x}) + WS_B(\vec{x}).$$

p^k との乗算により $p^k \cdot WS_A$ の出力ビット数が増える. しかし, WS_A と WS_B の入力数が, もとの回路の入力数の半分にまで削減可能なので, 回路全体のハードウェア量は小さく抑えられる可能性がある.

［例 4.1］　8 桁の 3 進数から 2 進数への基数変換 **(8ter2bin)** を行う回路を考える. ここでは, 分解係数を α=$2^6 = 64$ と α=$3^4 = 81$ の 2 つの場合を考える.

3 進数は 2 進化 3 進数で表現されている. 表 4.1 に, 3 のべき数 3^i, $(i = 0, 1, 2, \ldots, 7)$ のそれぞれを分解係数 $\alpha = 2^6 = 64$, および, $\alpha = 3^4 = 81$ で割った時の商と余りで表現した数式を示す. これらの商と余りは, $WS_A(\vec{x})$ と $WS_B(\vec{x})$ の重み w_i に等しいことに注意されたい. ここでは, 11 入力のセルがカスケード実現で使用可能と仮定する. 表 4.1 から, **8ter2bin** の二つの実現方法が求まる.

分解係数を 2^6 とした場合:

- $WS(\vec{x}) = 2^6 \cdot WS_A(\vec{x}) + WS_B(\vec{x})$.
- $WS_A(\vec{x}) = 34x_7 + 11x_6 + 3x_5 + x_4$.
- $WS_B(\vec{x}) = 11x_7 + 25x_6 + 51x_5 + 17x_4 + 27x_3 + 9x_2 + 3x_1 + x_0$.
- $WS_A(\vec{x})$ は, 入力 $x_4 \sim x_7$ のみに依存するので, 入力数は, 8 となる. 出力値の最大値は, $2(1 + 3 + 11 + 34) = 98$ となり, これを表現するためには, 7 ビット必要である. $WS_A(\vec{x})$ は, 8 入力 7 出力であるので, 1 個のセルで実現できる.
- $WS_B(\vec{x})$ は, すべての入力 $x_0 \sim x_7$ に依存するので, 入力数は 16 となる. $WS_B(\vec{x})$ の重みベクトルは, 定理 3.1 の条件を満たしているので, 出力値の範囲は, $[0, 2(1 + 3 + 9 + 27 + 17 + 51 + 25 + 11)] = [0, 288]$ となり, この表現に出力は, 9 ビット必要である. 従って, $WS_B(\vec{x})$ は, 16 入力 9 出力である. これを, 2.3 節で述べた LUT カスケード実現で 11 入力のセルを使って実現すると 3 段のカスケードとなる.
- 2^6 の乗算は, 6 ビット左シフトすることで実現できる. WS_B の上位 3 ビットと WS_A とを 7 ビット加算器で加算すればよい. 図 4.1 に, 45824 ビットメモリと 7 ビット加算器で実現した例を示す.

分解係数を 3^4 とした場合:

- $WS(\vec{x}) = 3^4 WS_A(\vec{x}) + WS_B(\vec{x})$.
- $WS_A(\vec{x}) = 3^3 x_7 + 3^2 x_6 + 3^1 x_5 + 3^0 x_4$.
- $WS_B(\vec{x}) = 3^3 x_3 + 3^2 x_2 + 3^1 x_1 + 3^0 x_0$.
- $WS_A(\vec{x})$ は, $x_4 \sim x_7$ に依存するので, 入力数は 8 となる. 定理 3.1 より出力値の最大値は, $2(1 + 3 + 9 + 27) = 80$ となるので, これを表現するために出力には, 7 ビットあれば十分である. 従って, $WS_A(\vec{x})$ は, 11 入力のセル 1 個で実現できる.
- $WS_B(\vec{x})$ は, $x_0 \sim x_3$ に依存するので, 入力数は 8 となる. 定理 3.1 より出力値の範囲は, $[0, 2(1+3+9+27)] = [0, 80]$ となる.
- $3^4 WS_A$ の最大値は, 6480 となる. 従って, この出力を表現するために 13 ビット必要となる. ここでは, 1 個のセルで直接 $3^4 WS_A$ を実現している. $3^4 WS_A$ と WS_B とを加算するために 13 ビット 加算器を用いる. 図 4.2 に 5120 ビットメモリと 13 ビット加算器とで実現した例を示す.　　(例終り)

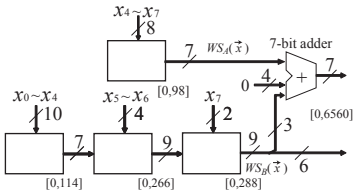

図 4.1　分解係数 2^6 で算術分解して得た 8 桁 3 進 2 進変換回路

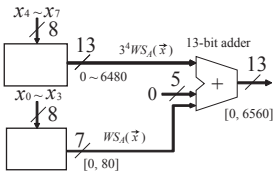

図 4.2　分解係数 3^4 で算術分解して得た 8 桁 3 進 2 進変換回路

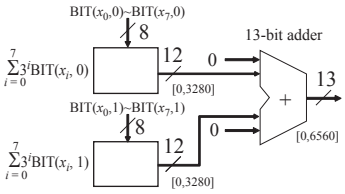

図 4.3　入力の 2 進数表現に注目した算術分解で得た 8 桁 3 進 2 進変換回路

4.2.1　入力の 2 進数表現に注目した算術分解

本節では, 入力の 2 進数表現に注目した算術分解を提案する.

［定義 4.1］　i を整数とするとき, $BIT(i,j)$ は i の 2 進数表現の j 番目のビットを表現する. ここで, 最下位ビット（LSB) は 0 番目のビットとする.

［例 4.2］　$BIT(2,1) = 1, BIT(2,0) = 0, BIT(1,1) = 0, BIT(1,0) = 1.$　(例終り)

整数 i は, $\lceil \log_2 i \rceil$ ビットで表現できる. ゆえに次の関係を満たし, これにより次の定理 4.3 が得られる.

$$i = \sum_{j=0}^{\lceil \log_2 i \rceil - 1} 2^j BIT(i,j).$$

［定理 4.3］　p 進 2 進変換は次のように表現できる:

$$WS(\vec{x}) = \sum_{j=0}^{\lceil \log_2 p \rceil - 1} 2^j \sum_{i=0}^{n-1} p^i BIT(x_i, j).$$

［例 4.3］　8 桁の 3 進 2 進変換回路 **(8ter2bin)** を構成する. 定理 4.3 より, $WS(\vec{x})$ は次のように表現可能である:

$$WS(\vec{x}) = 2\sum_{i=0}^{7} 3^i BIT(x_i, 1) + \sum_{i=0}^{7} 3^i BIT(x_i, 0). \quad (4.2)$$

図 4.3 は, 上記の分解に対応した回路である. 各モジュールは 8 入力である. $\sum_{i=0}^{7} 3^i = 3280$ であるので, 各モジュールの出力数は 12 となる. 2 倍は, 1 ビット左シフトすることで実現できるので, 信号の接続だけで実現できる. 回路は, 6K ビットメモリ 2 個と 13 ビット加算器とで構成できる. さらに，(4.2) の各項を, 定理 4.1 を用いて, 分解係数 $3^4 = 81$ で算術分解すると,

$$WS(\vec{x}) = 2 \cdot [3^4 \cdot \sum_{i=4}^{7} 3^{i-4} BIT(x_i, 1) + \sum_{i=0}^{3} 3^i BIT(x_i, 1)]$$
$$+1 \cdot [3^4 \cdot \sum_{i=4}^{7} 3^{i-4} BIT(x_i, 0) + \sum_{i=0}^{3} 3^i BIT(x_i, 0)] \quad (4.3)$$

が得られる.

式 (4.3) に基づき，構成した回路を図 4.4 に示す．各セルは,

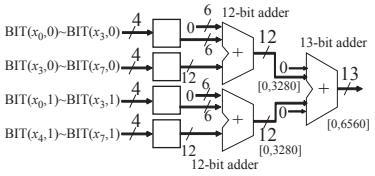

図 4.4　図 4.3 を 3^4 で算術分解して得た 8 桁 3 進 2 進変換回路

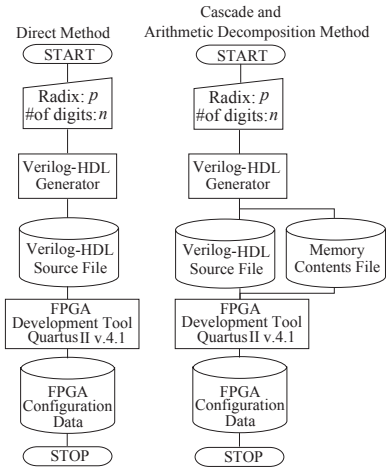

図 5.1　基数変換回路開発システム

4 入力であり. メモリ総量は 576 ビットである. 但し, 2 個の 12 ビット加算器と 1 個の 13 ビット加算器が必要である. (例終り)

5.　FPGA での実現

提案方法の有効性を示すために, 3 進 2 進基数変換回路を FPGA 上に実現し, ハードウェア量と動作速度についての各実現方法の優位性を検討した.

5.1　FPGA と開発システム

FPGA デバイスには, Altera 社の Cyclone II (EP2C5T144C7) を用いた．このデバイスは, 13 個の組込み乗算器 (Embedded Multipler: EM), 26 個の 4K ビット組込みメモリ (M4K), そして, 4608 個のロジック・エレメント (Logic Element: LE) を持っている. FPGA 開発システムには, Altera 社の Quartus II V. 4.1 を用いた.

開発した基数変換の合成システムを図 5.1 に示す. このシステムは, Verilog-HDL 形式のコードと, 組込みメモリである M4K に格納するデータとして前節で述べた設計を生成する. FPGA では, LUT(セル) は, M4K を用いて実現し, 加算器はロジック・エレメント (LE) を用いて実現した.

5.2　8 桁の 3 進 2 進変換回路

表 5.1 に, 8 桁の 3 進 2 進変換回路 **(8ter2bin)** を 7 種類の異なる設計法で実現した結果を示す.

直接実現法 (Direct Method: DM)

仕様として基数 p と桁数 n を与えると合成システムは, Verilog-HDL コードを生成する. なお, メモリ 1 個で実現する方法は, 組込みメモリの容量を大幅に超えるのでこの FPGA では実現不能である.

- DM1 は, 数式 $\sum_{i=0}^{7} 3^i x_i$ を直接実現したものである. 図 2.1 は, Quartus II が生成した回路である. マッピング後, Quartus II は, 7 個の EM で乗算を, 66 個の LE で加算器を実現している. M4K は未使用である.
- DM2 も, 図 2.1 に対応した回路であるが, この場合は, EM を LE で置換している. そのため, LE だけで構成しており, EM も M4K も使用していない. 195 個の LE を使用している. つまり, 129 個の LE で 7 個の EM を置き換えたことになる. LE で

表 5.1　8 桁 3 進 2 進変換回路の Cyclone II FPGA での実現結果

Design Method			LE	M4K	EM	Delay [nsec]
DM1	With EM	Fig.2.1	66	0	7	26.7
DM2	W/O EM	Fig.2.1	195	0	0	23.7
AD1	2^6	Fig.4.1	8	13	0	30.0
AD2	3^4	Fig.4.2	13	2	0	14.8
AD3	BIT	Fig.4.3	12	2	0	14.3
AD4	BIT+3^4	Fig.4.4	36	4	0	16.8
PAR	M4K only	Fig.3.6	0	11	0	22.2

表 5.2　12 桁 3 進 2 進変換回路の Cyclone II FPGA での実現結果

Design Method			LE	M4K	EM	Delay [nsec]
DM1	With EM	Fig.2.1	139	0	15	35.8
DM2	W/O EM	Fig.2.1	457	0	0	32.0
AD2	3^6	Fig.4.2	20	30†	0	17.3
AD3	BIT	Fig.4.3	19	38‡	0	16.6
AD4	BIT+3^6	Fig.4.4	57	4	0	17.6

†:EP2C8T144C7, ‡:EP2C20F256C7 使用

構成した定数倍の乗算器が EM より高速に定数倍の乗算を実行できるので, DM2 は, DM1 より高速に動作する.

算術分解に基づいた方法 (Arithmetic Decomposition Method: AD)

システムは, Verilog-HDL コードと M4K に格納するデータを生成する.

- AD1 は, 図 4.1 に対応するもので, 2^6 を分解係数として得られた回路である. Quartus II は, 4 個の LUT を 13 個の M4K で, 加算器を 8 個の LE で実現している.
- AD2 は, 図 4.2 に対応するもので, 3^4 を分解係数として得られた回路である. Quartus II は, 2 個の LUT を 2 個の M4K で, 加算器を 13 個の LE で実現している.
- AD3 は, 図 4.3 に対応するもので, 入力の 2 進表現に基づく算術分解で得られた回路である. Quartus II は, 2 個の LUT を 2 個の M4K で, 加算器を 12 個の LE で実現している.
- AD4 は, 図 4.4 に対応するもので, 入力の 2 進表現に基づく算術分解で得られた回路 (AD3) を更に 3^4 で算術分解して得られた回路である. Quartus II は, 4 個の LUT を 4 個の M4K で, 加算器を 36 個の LE で実現している. 加算器がより複雑なので AD3 よりも低速である.

出力分割に基づいた方法 (Partition of Outputs Method: PAR) [8]

システムは, Verilog-HDL コードと M4K に格納するデータを生成する.

- PAR は, 図 3.6 に対応し, 6 個の LUT で構成されている. Quartus II は, 6 個の LUT を 11 個の M4K で実現している.

8ter2bin の場合, AD3 は, 最も高速でハードウェア量が最小である.

5.3　12 桁の 3 進 2 進変換回路

表 5.2 に, 12 桁の 3 進 2 進変換回路 (**12ter2bin**) を実現した結果を示す.

直接実現法 (Direct Method: DM)

- DM1 は, 15 個の組込み乗算器 (EM) を用いている.
- DM2 は, **8ter2bin** の場合と同様に DM1 より高速に動作する.

算術分解に基づいた方法（Arithmetic Decomposition Method: AD)

- AD2 は, 図 4.2 の 12 桁版に対応するもので, 3^6 を分解係数として求めた回路である. 1 個の 12 入力 20 出力の LUT, 1 個の 12 入力 10 出力の LUT, そして 20 ビット加算器を必要とする. Quartus II は, これらの LUT を 30 個の M4K で, 加算器を 20 個の LE で実現した. このため, 36 個の M4K を持つより規模の大きい FPGA(EP2C8T144C7) を必要とした.
- AD3 は, 図 4.3 の 12 桁版に対応する. 2 個の 12 入力 19 出力 LUT, 20 ビット加算器を 1 個必要とする. Quartus II は, これらの LUT を 38 個の M4K で実現するので, AD2 で用いたものより更に大規模な 52 個の M4K を持つ FPGA(EP2C20F256C7) を必要とした.
- AD4 は, 図 4.4 の 12 桁版に対応する. AD3 を更に 3^6 で算術分解して得た回路である. 4 個の 6 入力 LUT が必要で, Quartus II は, これらの LUT を 4 個の M4K で, 加算器を 57 個の LE で実現した.

AD1 は, EM を使いすぎるために実現不能であり, PAR は論理合成に時間がかかり **12ter2bin** の実現には使用できなかった. **8ter2bin** の場合, AD2 と AD3 が高速である. 一方, **12ter2bin** の場合は, AD2 や AD3 は, より規模の大きい FPGA を必要とするため不適であり, AD4 が, 最小のハードウェア量を使用するので最良の構成法であるといえる.

規模が大きくなると組込みメモリが少なくてすむ AD4 の方法が FPGA での実現に適している.

6.　まとめと考察

本稿では, p 進数から 2 進数への変換回路を設計するための算術分解法を提案した. 3 進数から 2 進数への変換を例に用いて, その考え方を示した. 方法の有効性を確認するために, FPGA で実際に基数変換回路を実現するための合成システムを構築し, 実現を行い性能評価も行った.

図 4.1 は, 非分離的分解であるが, 一方, 図 4.2, 図 4.3, 図 4.4 は, 分離的分解であることに注意されたい. 図 4.2 の分離的分解は, 式 (2.2) や図 2.1 から容易に推察できる. 一方, 図 4.3 の分離的分解は, 簡単には気がつかない. また, 図 4.2 の分解は, 異なる部分回路を必要だが, 図 4.3 の分解は, 同一の部分回路を 2 つ持つことがわかる. 図 4.3 の回路は, 図 4.2 の回路より高速である.

これらの方法は, 3 進以外の基数変換や他の算術回路 [3] にも応用可能である. 今回は, 説明を割愛したが, メモリの読み出し時間と加算器 1 段分の遅延は, ほぼ同じだったのでパイプライン動作をさせてスループットを向上させることも可能である.

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・北九州地域知的クラスター創成事業の補助金, 明治大学特別研究による.

文　　献

[1] T. Hanyu and M. Kameyama, “A 200 MHz pipelined multiplier using 1.5 V-supply multiplevalued MOS current-mode circuits with dual-rail source-coupled logic,” *IEEE Journal of Solid-State Circuits 30, 11,* (1995), 1239-1245.

[2] Y. Iguchi, T. Sasao, and M. Matsuura, “On design of radix converters using arithmetic decompositions,” *36th International Symposium on Multiple-Valued Logic*, Singapore, May 17-20, 2006, p. 3

[3] 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.

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

[5] S. Muroga, *VLSI System Design*, John Wiley & Sons, 1982, pp. 293-306

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

[7] T. Sasao, M. Matsuura, and Y. Iguchi, “A cascade realization of multiple-output function for reconfigurable hardware,” *International Workshop on Logic and Synthesis(IWLS01)*, Lake Tahoe, CA, June 12-15, 2001, pp. 225-230.

[8] T. Sasao, “Radix converters: Complexity and implementation by LUT cascades,” *35th International Symposium on Multiple-Valued Logic*, Calgary, Canada, May 19-21, 2005, pp.256-263.

[9] T. Sasao, Y. Iguchi, and T. Suzuki, “On LUT cascade realizations of FIR filters,” *DSD2005, 8th Euromicro Conference on Digital System Design: Architectures, Methods and Tools*, Porto, Portugal, Aug. 30 - Sept. 3, 2005, pp.467-474.

[10] T. Sasao, “Analysis and synthesis of weighted-sum functions,” *IEEE Trans. on CAD*, Vol. 25, No. 5, May 2006, pp. 789-796..