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 (2)

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　デジタル信号処理では, 高速演算のために 2 以外の基数がよく用いられる. また, 金融計算では，10 進数が 2 進数の代わりに用いられる. このような場合, 基数変換回路が必要である. 本論文では, 2 進数を q 進数に変換する新しい基数変換回路の構成法を提案する．これは, weighted-wum (WS) 関数の概念に基づく新しい方法である. 各桁の WS 関数を LUT カスケードと 2 進加算器とで計算し, それらを q 進加算器で足し合わせることで基数変換器を構成する．16 ビットの 2 進 10 進変換器を例にその構成法を説明する.

キーワード　基数変換, 多値論理, LUT カスケード.

Design Method of Radix Converters Using Arithmetic Decompositions (2)

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 digital signal processing, radixes other than two are often used for high-speed computation. In the computation for finance, decimal numbers are used instead of binary numbers. In such cases, radix converters are necessary. This paper considers design methods for binary to q-nary converters. It introduces a new design technique based on weighted-sum (WS) functions. We compute a WS function for each digit by an LUT cascade and a binary adder, then add adjacent digits with q-nary adders. A 16-bit binary to decimal converter is designed to show the method.

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

1. はじめに

デジタル信号処理では, 通常, 2 進数が使用される [8]. しかし, 高速のデジタル信号処理では, p 進数 ($p > 2$) をよく使う [1], [5]. また, 金融計算では，10 進数を 2 進数の代わりに用いる．このような場合, 2 進数と p 進数間の**基数変換**が必要である [2], [7].

p 進数を q 進数 ($p, q \geq 2$) に変換する種々の方法が知られている. しかし, それらの多くが, 多大な計算量を必要とする. 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, 必要なメモリ量が増加するので実用的ではなくなる.

文献 [6] では, ROM と加算器とを組合せて 2 進 10 進変換を構成する方法を述べている. 文献 [10] 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 [9] で実現している. 文献 [13] 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を提案している. 文献 [3] では, LUT カスケード [9] と算術分解 [12] とを用いて p 進数 ($p > 2$) から 2 進数への基数変換回路を従来法より少ないメモリ量で実現する方法を提案している.

本稿では, p 進数から q 進数 ($q > 2$) への変換に LUT カスケードと算術分解とを組合せた構成法を提案する. 本稿では 16 桁 2 進 10 進変換回路の設計を通じて, 設計法の考え方を説明するが, 提案法は, 任意の p, q に容易に変更可能である.

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　2 進 10 進変換の真理値表

Binary				Decimal		Binary Coded Decimal							
x_3	x_2	x_1	x_0	y_1	y_0								
0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	1	0	1	0	0	0	0	0	0	0	1
0	0	1	0	0	2	0	0	0	0	0	0	1	0
0	0	1	1	0	3	0	0	0	0	0	0	1	1
0	1	0	0	0	4	0	0	0	0	0	1	0	0
0	1	0	1	0	5	0	0	0	0	0	1	0	1
0	1	1	0	0	6	0	0	0	0	0	1	1	0
0	1	1	1	0	7	0	0	0	0	0	1	1	1
1	0	0	0	0	8	0	0	0	0	1	0	0	0
1	0	0	1	0	9	0	0	0	0	1	0	0	1
1	0	1	0	1	0	0	0	0	1	0	0	0	0
1	0	1	1	1	1	0	0	0	1	0	0	0	1
1	1	0	0	1	2	0	0	0	1	0	0	1	0
1	1	0	1	1	3	0	0	0	1	0	0	1	1
1	1	1	0	1	4	0	0	0	1	0	1	0	0
1	1	1	1	1	5	0	0	0	1	0	1	0	1

p 進数 (q 進数) が 2 進数でない時, その各桁は, 2 進化 p (q) 進数で表されているとする.

［定義 2.2］　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 2.1］　$BIT(6, 2) = 1$, $BIT(6, 1) = 1$, $BIT(6, 0) = 0$. (例終り)

［例 2.2］　2 進 10 進変換を考える. 10 進数を表現するために 2 進化 10 進数 (binary coded decimal: BCD) を使う. つまり, 0 は (0000), 1 は (0001), ..., 8 は (1000), 9 は (1001) で表す. ここで, (1010), (1011),..., (1111) は未使用コードである.

表 2.1 は, 4 桁の 2 進 10 進変換のための真理値表である. 2 進数の入力を, $\vec{x} = (x_3, x_2, x_1, x_0)$ と表す. 出力である 10 進数の表記は, $\vec{y} = (y_1, y_0)$ と表す. BCD での表現は, $\vec{y}$= (y_1, y_0) = $(BIT(y_1, 3), BIT(y_1, 2), BIT(y_1, 1), BIT(y_1, 0), BIT(y_0, 3), BIT(y_0, 2), BIT(y_0, 1), BIT(y_0, 0))$ と表せる.　(例終り)

2.2 従来の実現法

2.2.1 ランダム・ロジックによる実現

比較器, 減算器, 及び, マルチプレクサを用いて, 基数変換回路

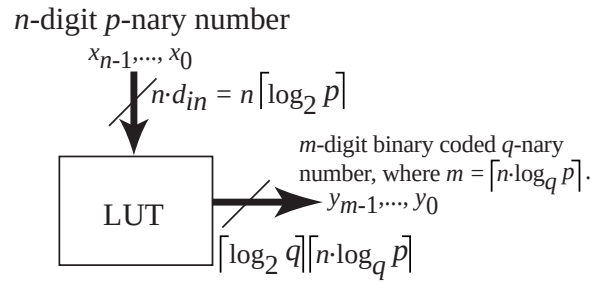


図 2.2　n 桁 p 進 q 進変換: 1 個のメモリによる実現

を構成できる.

［例 2.3］　図 2.1 に示した 8 桁 2 進 10 進変換 (**8bin2dec**) を考える. これは, 8 桁の 2 進数 $x[7:0]$ を 3 桁の BCD に変換する回路である. 出力値の範囲は, 0 から 255 である.

まず, 減算器 S_1 が, $x-200$ を計算する. 比較器 C_1 は, 入力 x と 200 とを比較する. $x \geqq 200$ ($x < 200$) のとき, $o2[1] = 1(0)$ とし, マルチプレクサ M_1 は, $t_1 = x - 200$ ($t_1 = x$) を次段に伝える. 次に, 減算器 S_2 が, $t_1 - 100$ を計算する. 比較器 C_2 は, t_1 と 100 とを比較する. $t_1 \geqq 100$ ($t_1 < 100$) のとき, $o2[0] = 1(0)$ とし, マルチプレクサ M_2 は, $t_2 = t_1 - 100$ ($t_2 = t_1$) を次段に伝える. 以後同様に, 回路は 2 進数を 10 進数に変換する. (例終り)

2.2.2　1 個のメモリによる実現

最も単純な方法は, 1 個のメモリに基数変換の真理値表をそのまま記憶させる方法である.

2 進化 p 進数で表された n 桁の数を m 桁の q 進数に 1 個のメモリを用いて変換する回路を図 2.2 に示す.

入力の各桁のビット数を d_{in} とする. この時, $d_{in} = \lceil \log_2 p \rceil$ である. 入力は, n 桁あるので, 入力の総ビット数は nd_{in} である. この n 桁の p 進数が表現する数値は, 0 以上の整数とすると, 0 から $p^n - 1$ の区間となる.

基数変換の出力は, m 桁の 2 進化 q 進数である. 出力の各桁のビット数を d_{out} とすると, $d_{out} = \lceil \log_2 q \rceil$ である.　一方, 回路の出力は, $m = \lceil n \log_q p \rceil$ 桁である. ただし, 最上位桁は必ずしも d_{out} ビット必要ではなく, $d_{out_{m-1}} = \lceil \log_2([p^{n-1}/q^{m-1}]) \rceil$ ビットでよい. それゆえ, メモリサイズは, $2^{nd_{in}}((m-1)\cdot d_{out} + d_{out_{m-1}})$ ビットとなる.

本方式は単純で高速だが, 入力の桁数が増えるとメモリ量が非常に大きくなり実用的ではない.

［例 2.4］　16 桁 2 進 10 進変換回路 (以後, 簡単のために 16bin2dec と省略) が表現する値の範囲は, $[0, 2^{16}-1] = [0, 65535]$ である. 変換後の 10 進数は, $m = \lceil \log_{10} 65535 \rceil = 5$ 桁となる. $d_{out} = 4$, $d_{out_{m-1}} = 3$ これを 1 個のメモリで実現すると $2^{16} \cdot ((5-1)4+3) = 1,245,184$ ビットとなる.　(例終り)

2.2.3　LUT カスケードを用いた実現

メモリ 1 個で基数変換回路を実現するのは単純だが, 入力桁数が増えると指数関数的にメモリ量が増大する.

そこで, LUT カスケード実現を用いてハードウェア量を削減する方法が提案されている [10]. この方法では, 出力をグループに分割することで効果を高めている. 関数分解 [8] を用いて, LUT カスケード法で論理関数を実現できるかどうか判定できる. 関数分解には, データ構造として特性関数の二分決定グラフによる表現 (Binary Decision Diagrams for Characteristic Functions: BDD for CF) を用いる.

図 2.3 に 16 桁 2 進 10 進変換回路を示す [10]. この例では, LUT3 段からなる単純な構成で全ての出力を実現している. LUT は, メモリで実現する. メモリの総量は, 73, 728 ビットである. 本方法は, メモリだけで構成でき, 配線は隣接メモリ間のみであり,

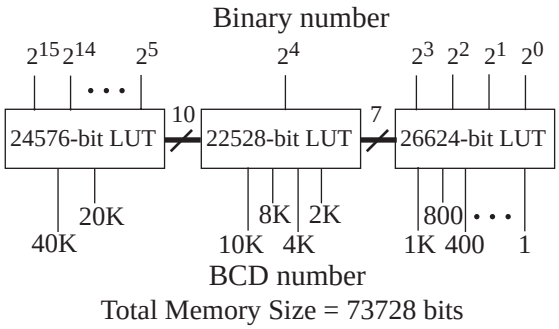


図 2.3　16 桁 2 進 10 進変換: LUT カスケード法による実現

単純な構成であることが特徴である. しかし, 論理合成に BDD for CF を用いるので, 入力の桁数が大きくなると構成を求める計算時間が増えすぎて実用的ではない.

文献 [3] では, BDD for CF を用いずに LUT カスケードを構成する方法を提案している. これに算術分解を組合せて加算器とメモリを用いることで, 文献 [10] の構成よりもメモリ量を削減している. しかし, 残念なことに, この方法は入力が 2 進化 p ($p > 2$) 進数で表されていて, それを 2 進数に変換する場合のみに適用可能であり, 本稿で取り扱う 2 進 q 進変換 ($q > 2$) には適用できない.

2.2.4　メモリと q 進加算器とを組合せた実現

2 進 10 進基数変換器をメモリと加算器を組合せて実現する方法が提案されている [6]. 図 2.4 に 16 桁 2 進 10 進変換回路を示す [6]. この回路の特徴を以下に示す.

（1） 2 進 10 進変換では, 入力 2^0 の桁と出力の最下位ビットつまり 1 の桁とは一致するので直結する.

（2） 残りの 15 入力を 2 分割する: 上位 9 ビットと下位 6 ビットとに分割する. 全ての入力パターンに対して対応する BCD の値を各 LUT に記憶させておく.

（3） 最上位桁 (40K, 20K, and 10K) は, 3 入力 LUT (原図ではゲートで実現されている) と 2 進加算器とで実現される.

（4） 総メモリ量は, 8, 216 ビットである.

（5） 加算器は 2 進加算器を用いる. 最下位の LUT には, 予め桁ごとに 6 を加算した **excess-6** コードで値を格納しておく.

（6） BCD 加算は, 2 進加算器と特別な減算器とで実現されている. 桁上がりがあるときは, 補正のために当該の桁から 6 を減算する.

なお, 原図 [6] では, 1-of-16 デコーダが用いられている. これは, 16 個の小規模メモリに対する選択信号を出力するためである. 現在では, より大きなメモリを使用できるため, 16 個のメモリを 1 個の LUT で置換した.

以上のような巧妙な方法を用いて, メモリと q 進加算器 ($q = 10$) とで 2 進 10 進変換器を構成している. この方法は, 入力を 2 分割することで, 総メモリ量を削減している.

3.　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 [12], [13]. 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 3.1］　n 入力の WS 関数は, 次のように定義される [13]:

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

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

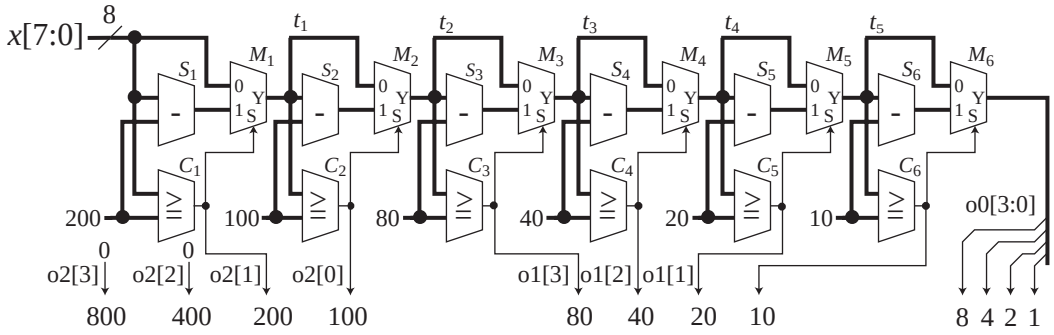


図 2.1　8 桁 2 進 10 進変換回路 (ランダムロジックによる実現)

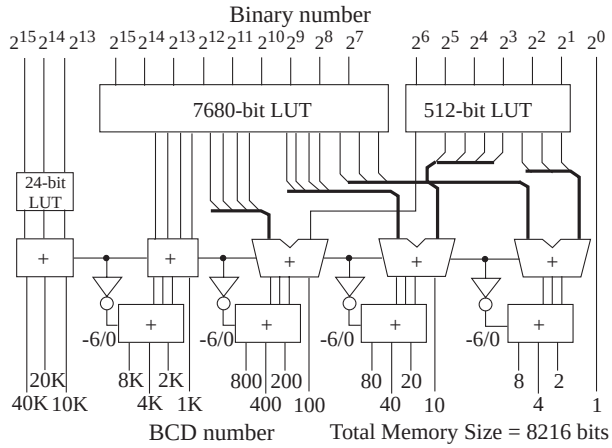


図 2.4　16 桁 2 進 10 進変換: メモリと BCD 加算器とを用いた実現

である.

本稿では, p 進数から q 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 と x_i は, 非負の整数とする.

［定義 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 3.3］ 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.

次に WS 関数の値域を検討する.

［定義 3.4］ 関数 $f(x)$ の値域を $Range(f(x))$ と記す.

［例 3.1］ $WS_1(\vec{x}) = x_0 + 3x_1$, $x_i \in \{0, 1, 2\}$ 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 3.1 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 3.1］ WS 関数, $WS(\vec{x}) = \sum_{i=0}^{n-1} w_i x_i$, $w_i \geqq 0$, $x_i \in \{0, 1, \ldots, p-1\}$,$p \geqq 2$ が取り得る値の種類は, 高々 p^n 通りである.

(証明略)

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.

(証明略)

［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.

(証明略)

4. LUT カスケードと算術分解とを用いた実現

本節では, LUT カスケードと算術分解とを組合せて基数変換回路を設計する方法を提案する.

従来法 [6] の図 2.4 を機能ブロックで書き直し, 図 4.1(a) に示す. 本回路は, 3 個の LUT と 5 個の 10 進加算器だけで実現されている. 図 4.1(a) では, 中央の LUT の出力は, 全ての q 進加算器の入力に接続されている.

一方, 本稿で提案する方法を図 4.1(b) に示す. 出力桁の 1 桁もしくは 1 桁とその桁上がりの値を入力に対して予め計算しておき, LUT に格納しておく. 桁毎に独立して計算するので, 各 LUT からの出力は, 対応する 1 個の q 進加算器と隣接する上位の q 進加算器に入力される. しかし, 本方法は m 個の LUT を必要とする. さらに, 下位桁に対する LUT の入力数は大きくなってしまう. ここからは, WS 関数の概念を用いて LUT カスケードに対する総メモリ量を削減する方法を考えてみる.

4.1 WS 関数の LUT カスケード実現

ここからは, 図 4.1(b) の各 LUT によって実現される論理関数を考える. 表 4.1 は, 2^i $(i = 15, 14, \ldots, 0)$ に対する 10 進数を示している. 各行は, それぞれ 10^4, 10^3, 10^2, 10^1, 10^0 の桁を示している. 図 4.1(b) 内の LUT が表す関数を表 4.1 から得られる. 例えば, $10000 = 10^4$ の位の値は, x_{14}, x_{15}, および, 下位桁からの桁上がりで求められる.

各 LUT が実現する式を以下に示す:

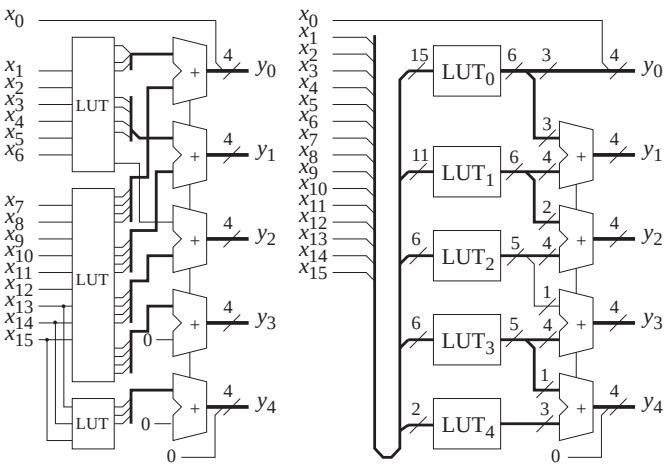
$$z_4 = x_{14} + 3x_{15}, \tag{4.1}$$

$$z_3 = x_{10} + 2(x_{11} + x_{15}) + 4x_{12} + 6x_{14} + 8x_{13}, \tag{4.2}$$

$$z_2 = (x_7 + x_{13}) + 2x_8 + 3x_{14} + 5x_9 + 7x_{15}, \tag{4.3}$$

$$z_1 = (x_4 + x_9) + 2(x_7 + x_{10}) + 3x_5 + 4x_{11} + 5x_8 + 6(x_6 + x_15) + 8x_14 + 9(x_{12} + x_{13}), \tag{4.4}$$

$$z_0 = x_0 + 2(x_1 + x_5 + x_9 + x_{13}) +$$



(a) Conventional Method (b) Proposed Method

図 4.1 メモリと q 進加算器とを使った基数変換器の構成法

表 4.1 2 のべき乗の 10 進数表現

x	2^{15}	2^{14}	2^{13}	2^{12}	2^{11}	2^{10}	2^9	2^8	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
10^4	3	1														
10^3	2	6	8	4	2	1										
10^2	7	3	1	0	0	0	5	2	1							
10^1	6	8	9	9	4	2	1	5	2	6	3	1				
10^0	8	4	2	6	8	4	2	6	8	4	2	6	8	4	2	1

$$4(x_2 + x_6 + x_{10} + x_{14}) + 6(x_4 + x_8 + x_{12}) + 8(x_3 + x_7 + x_{11} + x_{15}), \quad (4.5)$$

$$y = 10^4 z_4 + 10^3 z_3 + 10^2 z_2 + 10 z_1 + z_0, \quad (4.6)$$

ここで, $z_i (i = 0, 1, 2, 3, 4)$ は, LUT$_i$ の論理関数を表している.

式 (4.1) – (4.5) は, WS 関数を表しているので, 総メモリ量が小さい LUT カスケードで関数を実現するために 3 節で述べた性質を利用する. LUT カスケードの段数も削減する.

図 4.2(a) は, 式 (4.5) を実現している. 各 LUT は, 唯一の外部入力 x_i を持っている. 表 4.1 に示されているように, 重み係数は非負の整数である. 式 (4.5) の重み係数は全て非ゼロである. 一方, 式 (4.3) は, 10^2 の桁を表しているが, $w_{12}, w_{11}, w_{10}, w_6, \ldots, w_0$ は 0 であり, これは $x_{12}, x_{11}, x_{10}, x_6, \ldots, x_0$ と乗算される.

回路を実現する場合, ゼロの係数は省略できる. さらに, 入力を係数が昇順になるように入力変数を並び替えることにする.

総メモリ量最小のカスケードを見つけるために, 隣接する LUT を併合するかしないかを検討する. 異なる組合せの数は, LUT の個数を k 個とすると 2^{k-1} である. この場合, 入力変数 x_i は, LUT_i に入力されていて, 各 LUT は, $d = \lceil \log_2 p \rceil$ 個の 2 値入力を持つ. このような制約の下では, LUT カスケードの全ての実現は 2^{k-1} 個の異なる組合せとなる.

［定理 4.1］ 図 4.3 に示す LUT カスケードを考える, ここで, 各セルへの変数の順序と割り当て方は固定する. この場合, 最小のメモリサイズを持つ LUT カスケードは, 2^{s-1} 個の異なる組合せの中から見つけられる.

(証明略)

［例 4.1］ 図 4.4(a) に 3 セルの LUT カスケードを示す. 隣接する LUT を併合するかしないかの組合せは, $2^{3-1} = 4$ 通りである. これを (a)–(d) に示す. (例終り)

隣接する LUT を併合可能かどうかを検出するための補題を示す.

［補題 4.1］ 図 4.5 に示す LUT カスケードを考える, ここで, LUT H は, k 入力 k 出力である. この場合, メモリ量を増加させることなく 2 つの LUT を 1 個に併合可能である.

(証明略)

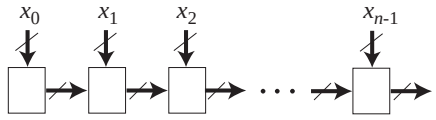


図 4.3 LUT カスケード.

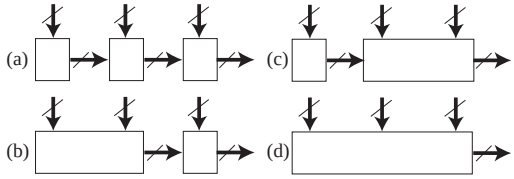


図 4.4 3 セルの LUT カスケードの全ての併合の組合せ.

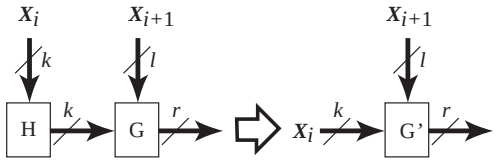


図 4.5 併合により総メモリ量を削減可能な LUT.

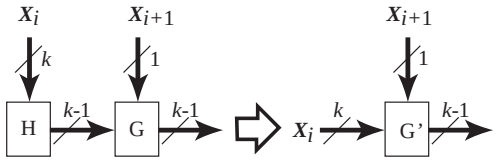


図 4.6 総メモリ量を変化させない LUT の併合.

補題 4.1 を用いると, 併合可能な LUT を見つけられる. 図 4.5 では, LUT H のメモリ量を削減でき, 段数も 1 段削減できる.

［補題 4.2］ 図 4.6 に示すような LUT カスケードを考える, ここで, LUT H は, k 入力 $(k-1)$ 出力, LUT G は, k 入力 $(k-1)$ 出力である. この場合, メモリ量を増加させることなく 2 つの LUT を 1 つに併合可能である.

(証明略)

補題 4.2 を用いることで, 総メモリ量を増加させることなく段数を 1 段削減できる. 補題 4.1, 4.2 以外の併合を行うと LUT カスケードの総メモリ量は増加する.

［アルゴリズム 4.1］ (LUT の併合)

（1） 各 LUT_i には, 1 個の外部入力 $x'_i, (i = 0, 1, \ldots, k-1)$ が入力されている LUT カスケードで WS 関数を実現する.

（2） 補題 4.1 を LUT カスケードにできる限り適用する.

（3） 補題 4.2 を LUT カスケードにできる限り適用する.

［定理 4.2］ アルゴリズム 4.1 は, 最小のメモリサイズの LUT カスケードを生成する.

(証明略)

総メモリ量が増加しても, より段数の少ない LUT カスケードが必要な場合がよくある. これは以下のように実現できる:

s をアルゴリズム 4.1 で得られた LUT カスケードの LUT の個数とする. 隣接した LUT を併合するかしないかの組合せの個数は, 2^{s-1} 個である. これらの全ての LUT カスケードの中から, 仕様を満たす LUT カスケードを選択すればよい.

［例 4.2］ アルゴリズム 4.1 を用いて, 式 (4.5) が表す WS 関数を LUT カスケードで実現する設計過程を図 4.2 に示す. (e) と (f) の LUT 内に書いた数値はメモリ量を表す.

- 図 4.2(a) は, 式 (4.5) に対応する LUT カスケードである. ここで, 各 LUT は唯一の外部入力 x_i を持っている. 総メモリ量は 6, 778 ビットであり, 段数は 16 である.
- 2 進数から 10 進数の変換においては, 入力 x_0 は, 出力 $BIT(y_0, 0)$ を直接表現している [6]. よって, x_0 の項を式 (4.5)

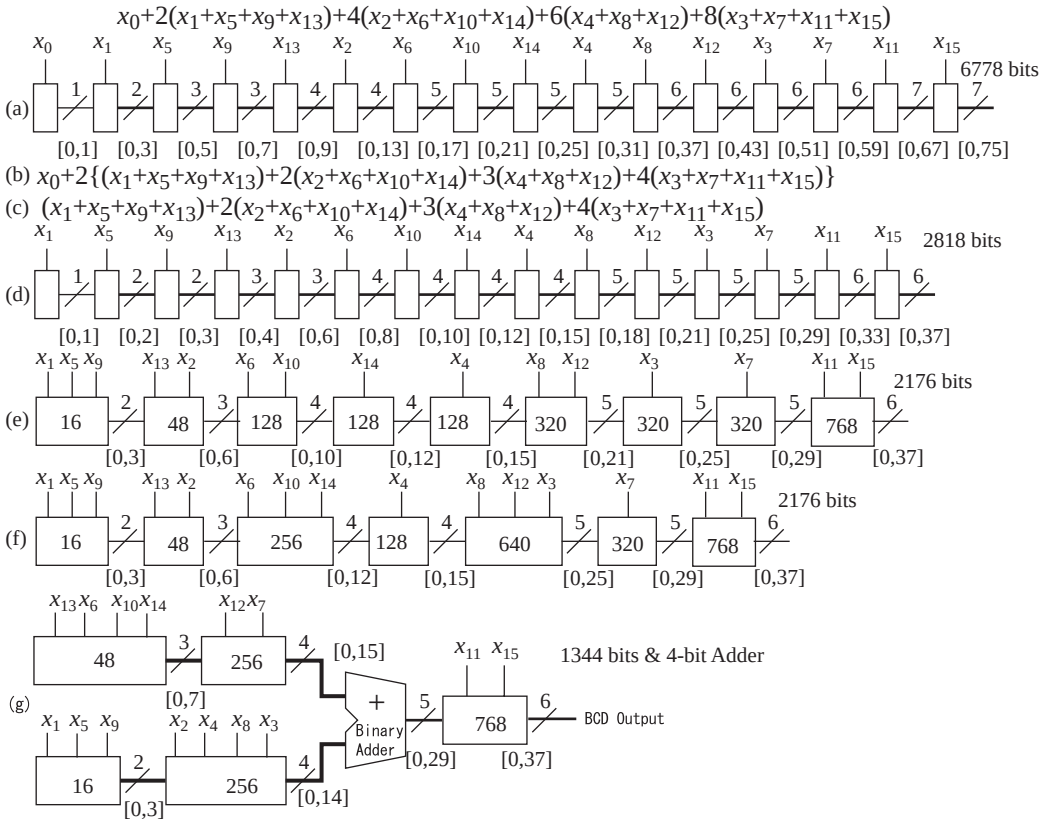


図 4.2　z_0 を実現する LUT カスケード.

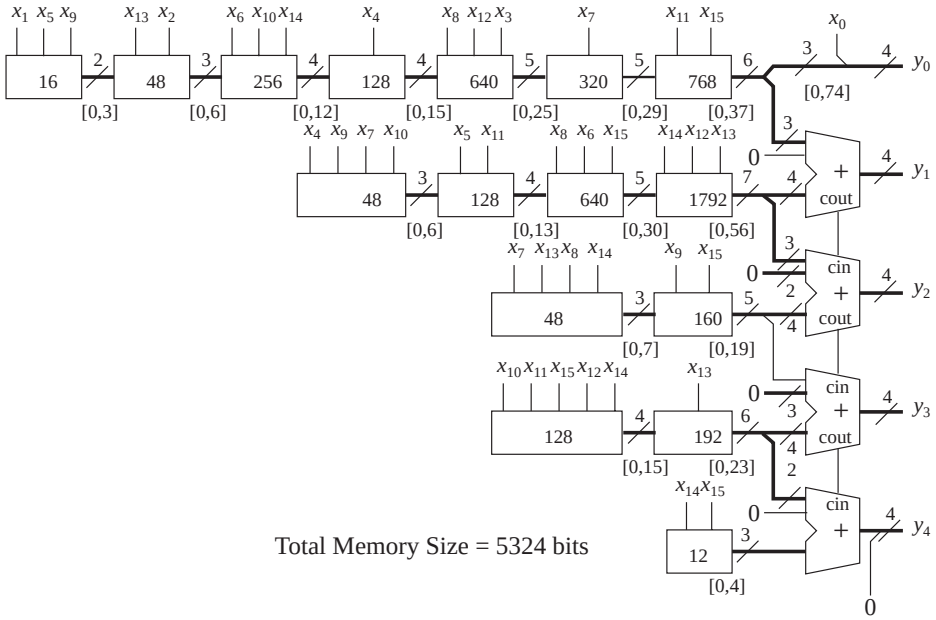


図 4.7　16 桁 2 進 10 進変換回路を LUT カスケードと BCD 加算器とで実現した例.

から除去する. 次に, 2 で括り, (b) に示す式を得る.

- (d) は, (c) に対応する LUT カスケードである. 総メモリ量は 2, 818 ビットであり, 段数は 15 である.
- 補題 4.1 を (d) に繰り返し適用し, (e) に示す LUT カスケードを得る. 総メモリ量は 2, 176 ビットであり, 段数は 9 である.
- 補題 4.2 を (e) に繰り返し適用し, (f) に示す LUT カスケードを得る. 総メモリ量は 2, 176 ビットであり, 段数は 7 である. (例終り)

最終段の LUT の出力は, BCD 加算器への入力となるので, BCD コードで出力することに注意されたい.

［例 4.3］ アルゴリズム 4.1 を式 (4.1) – (4.5) に適用し, LUT カスケードを得る. 次に, これらを BCD 加算器で加算する. 図 4.7 に 16 桁 2 進 10 進変換回路を示す. 総メモリ量は 5, 324 であり, 4 個の 10 進加算器を使用する. 一方, 図 2.4 に示した従来法での総メモリ量は, 8, 216 ビットであり, 5 個の BCD 加算器を使用している. (例終り)

4.2 WS 関数の算術分解を用いた実現

［定理 4.3］ 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:

$$WS(\vec{x}) = \sum_{i=0}^{n-1} w_i x_i = \alpha WS_A(\vec{x}) + WS_B(\vec{x}), \qquad (4.7)$$

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. 式 (4.7) を $WS(\vec{x})$ の**算術分解**といい, α を **分解係数**と呼ぶ. α には, $1 \leqq \alpha < \sum_{i=0}^{n-1} w_i x_i$ の任意の整数を選ぶことができる. ここでは $\alpha = 1$, $X_A \cap X_B = \Phi$, $X_A \cup X_B = X$ となるような分解のみを考える.

算術分解を用いて LUT カスケードの総メモリ量の削減を考える. LUT カスケードは算術分解により分解できる. 2 進 10 進変換回路を LUT カスケードと BCD 加算器とで実現するとき, 最終段の LUT は BCD コードで出力されなければならない. つまり, 最終段の LUT は, 符号化の機能も担当している. それゆえ, 最終段の LUT を取り除いた残りの LUT カスケードを 2 進加算器で算術分解することにする.

2 進加算器の 2 つの入力のビット数は, 平衡しているとより小さな加算器を用いることができる. そこで, 2 つの入力変数を重み係数の和が等しくなるような分割を試みる.

［アルゴリズム 4.2］ (算術分解のための入力の 2 分割法)

（1） $X_A \leftarrow \phi$; $X_B \leftarrow \phi$; $IN \leftarrow \{w_0, w_1, w_2, \ldots, w_{n-1}\}$;

（2） If $IN = \phi$ then goto step (7);

（3） w_i を集合 IN 内の最小の要素とする;

（4） $IN \leftarrow IN - \{w_i\}$;

（5） If ($|X_A| \leqq |X_B|$), then $X_A \leftarrow X_A + \{x_i\}$, else $X_B \leftarrow X_B + \{x_i\}$;

（6） step (2) へ;

（7） 集合 X_A と X_B は, 入力の分割を示している.

［例 4.4］ 図 4.2(f) の LUT カスケードを算術分解して実現した例を図 4.2(g) に示す. 総メモリ量は 1, 344 ビットに削減されている. 但し, 4 ビット 2 進加算器が, 1 個必要になる.

［例 4.5］ アルゴリズム 4.2 を図 4.7 に適用して図 4.8 の 16bin2dec が得られる. 総メモリ量は 3, 788 ビットであり, 3 個の 2 進加算器と 4 個の BCD 進加算器を使用している. (例終り)

アルゴリズム 4.2 は, 平衡のとれた分割を生成する.

5. ま と め

本稿では, p 進 q 進変換器の設計法を提案した. 読みやすさのために, $p = 2$, $q = 10$, つまり 2 進 10 進変換器を例にその方法を説明した. しかし, 表 4.1 内の一番上の行の 2^i を p^i に, 一番左の列の 10^j を q^j で置換することで, 本方法は他の基数にも容易に変更できる. この場合, p 進 q 進変換器は, LUT カスケード, 2 進加算器, q 進加算器とで構成できる.

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

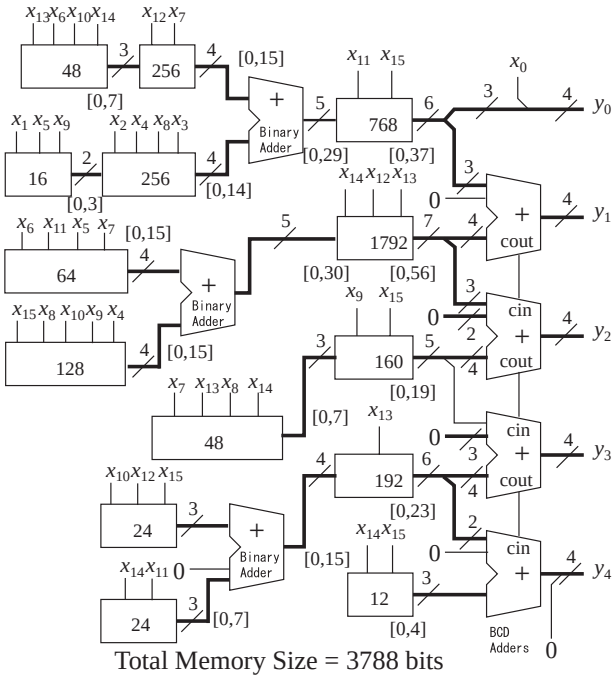


図 4.8 16 桁 2 進 10 進変換回路: LUT カスケード, 2 進加算器, BCD 加算器を用いた実現.

[3] 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

[4] 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.

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

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

[7] 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.

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

[9] 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.

[10] 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.

[11] T. Sasao, “Analysis and synthesis of weighted-sum functions,” *International Workshop on Logic and Synthesis*, Lake Arrowhead, CA, USA, June 8-10, 2005, pp.455-462.

[12] 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.

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