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

2004－SLDM－117 (38)
2004／12／2

LUTカスケードにおけるレール出力の符号化法について

永安 伸也† 笹尾 勤†† 松浦 宗寛††

† 九州工業大学大学院 情報創成工学専攻
〒820–8502 福岡県飯塚市大字川津 680–4
†† 九州工業大学 情報工学部
〒820–8502 福岡県飯塚市大字川津 680–4

あらまし 中間出力を有するLUTカスケードでレール出力の符号化法を工夫することにより，セルの出力数を削減する方法を示す．セルの出力数を削減することでLUTの出力数とカスケードの段数を削減できる．LUTの出力数を削減すればメモリ量を削減でき，段数を削減すれば評価時間を削減できる．LUTカスケードの設計には，多出力関数の特性関数を表すBDD(BDD_for_CF)を用いる．本手法を用いて，多数のベンチマーク関数においてLUTの出力数を10%程度削減できた．また，LUTの出力数と同時にカスケードの段数も削減できた場合もあった．

キーワード 論理合成，BDD_for_CF，関数分解，LUTカスケード，符号化問題，Non-Strict encoding

An Encoding Method for Rail Outputs in LUT cascades

Shinya NAGAYASU†, Tsutomu SASAO††, and Munehiro MATSUURA††

† Program of Creation Informatics, Kyushu Institute of Technology
kawazu 680–4, Iizuka, Fukuoka, 820–8502 Japan
†† Department of Comoputer Science and Eloctronics, Kyushu Institute of Technology
kawazu 680–4, Iizuka, Fukuoka, 820–8502 Japan

Abstract This paper shows a method to reduce the number of cell outputs for an LUT cascade with intermediate outputs by considering encoding. Reduction of the number of rail outputs reduces the number of outputs of LUTs and, the number of levels. Reduction of the number of outputs of LUTs reduces the amount of memory, while the reduction of the number of levels reduces the evaluation time. We use a BDD for characteristic function (BDD_for_CF) in the design. Experimental results show that our approach reduces the number of outputs of LUTs by 10%, and sometimes reduces the number of levels as well.

Key words Logic synthesis, BDD_for_CF, Functional decomposition, LUT cascade, Encoding problem, Non-Strict encoding

1. はじめに

LSIの集積度の増大に伴い，その開発期間や設計コストが増大している[1]．この問題を解決する方法の一つとして再構成可能なアーキテクチャを採用することが挙げられる．本論文では，再構成可能なアーキテクチャとして次のものを考える．

- RAM
- PLA
- FPGA
- 汎用マイクロプロセッサ

このうち，RAMとPLAは任意の論理関数を実現可能である．しかし，単一素子では回路の大きさが2^nに比例して増加するため実用的でない．従って，一般的にはFPGAを用いる．但し，FPGAは大規模になると配線部分の領域が大きくなってしまう．また，配線部分が不規則なので遅延時間の予測が困難である．演算速度が低くても良い場合には汎用マイクロプロセッサを利用できる．しかしこれは，消費電力が大きい割には低速である．そこで，本研究ではマイクロプロセッサとFPGAとの中間の性能を持ったLUTカスケードを用いる[2]．LUTカスケードは，論理関数をメモリのカスケードで表現する．LUTカスケードは規則的回路構造を有しているので次のような特徴を持つ．

- 使用環境や，製造ばらつきに対して安定．
- 再プログラムが容易．
- 冗長性の付加と誤り検出・訂正技術使用により，耐故障化が可能．

LUTカスケードでは，LUT間の配線は隣同士のLUT間の

表 1 分 解 表

$X_2=(x_3,x_4)$ \ $X_1=(x_1,x_2)$	0 0	0 1	1 0	1 1
0 0	1	0	0	1
0 1	0	0	0	0
1 0	0	0	0	0
1 1	1	0	0	1
	Φ_1	Φ_2	Φ_2	Φ_1

みに限定されている. そのため, 配線領域の問題が解決できる. 従って, 後は面積と評価時間の削減が問題となる. LUT カスケードでは段数が大きくなれば評価時間も大きくなり, LUT の出力数が大きくなれば必要とする面積も大きくなる. よって段数と LUT の出力数の削減が問題となる. 本研究では, LUT カスケードのレール出力の符号化を工夫することでセルの出力数を削減する. これにより LUT の出力数と段数を削減できる.

2. 論理関数の関数分解 [3]

本章では, 論理関数を複数個に分解し別々に実現する方法について述べる.

[定義 1] 入力変数を $X = (x_1, x_2, \ldots, x_n)$ とする. X の変数の集合を $\{X\}$ で表す. $\{X_1\} \cup \{X_2\} = \{X\}$ かつ $\{X_1\} \cap \{X_2\} = \phi$ のとき, (X_1, X_2) を X の**分割** (partition) という. X の変数の個数を $|X|$ で表す.

[定義 2] 論理関数 $f(X)$ に対して, X の分割を (X_1, X_2) とする. 2^{n_1} 列 2^{n_2} 行の表で, 各行, 各列に二進符号のラベルを持ち, その要素が f の対応する真理値表であるような表を, f の**分解表** (decomposition chart) といい, $M(f : X_1, X_2)$ で表す. ここで, $n_1 = |X_1|, n_2 = |X_2|$ であり, 列, 行はそれぞれ n_1, n_2 ビットの全てのパターンを有する. X_1 を**束縛集合** (bound set), X_2 を**自由集合** (free set) という.

[定義 3] 分解表の異なる列パターンの個数を**列複雑度** (column multiplicity) と呼び μ で表す. また, 各列パターンが表す論理関数を**列関数**といい $\Phi_i(X_2)$ $(i = 1, \ldots, \mu)$ で表す.

論理関数 $f(X)$ の列複雑度は, 入力変数の分割 (X_1, X_2) に依存する.

[例 1] $f(X)$ を 4 変数関数, (X_1, X_2) を X の分割, 但し $X_1 = (x_1, x_2)$, $X_2 = (x_3, x_4)$ とする. 表 1 は分解表の例である. ここで, 列複雑度は $\mu = 2$ である. また, 列関数は $\Phi_1(X_2) = x_3x_4 \vee \bar{x}_3\bar{x}_4$, $\Phi_2(X_2) = 0$ である. (例終り)

[定理 1] X の分割 (X_1, X_2) において, 関数 f の分解表 $M(f : X_1, X_2)$ の列複雑度が μ のとき, f は

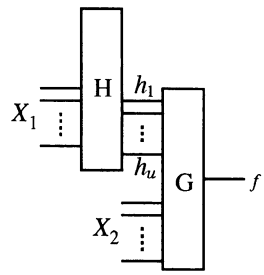

図 1 関 数 分 解

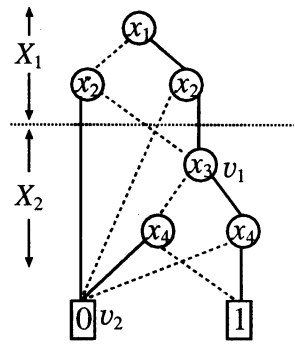

図 2 カット集合

$$f(X) = g(h_1(X_1), h_2(X_1), \ldots, h_u(X_1), X_2)$$

と表現でき, 図 1 の回路構造で実現可能である. ここで, 回路 H と G の間の接続線数は $u = \lceil \log_2 \mu \rceil$ である.

[定義 4] (X_1, X_2) を X の分割とする. 関数 f を表現する BDD において, その変数順序を (X_1, X_2) とする. この BDD において X_1 の節点から直接接続される X_1 以外の節点の集合を**カット集合** (cut set) という. また, カット集合の要素数を**カットの大きさ**という.

[定理 2] X の分割を (X_1, X_2) とし, 分解表 $M(f : X_1, X_2)$ の列複雑度を μ, 各列関数を $\Phi_i(X_2)(i = 1, \ldots, \mu)$ とする. カット集合 *cut_set* の各要素を根とする BDD は, 列関数 Φ_i を表現する. また, これより明らかに $|cut\_set| = \mu$ である.

[例 2] 例 1 で用いた論理関数 f を表現する BDD を図 2 に示す. 破線は 0 枝を, 実線は 1 枝を表す. ここで, 束縛集合は $X_1 = (x_1, x_2)$, 自由集合は $X_2 = (x_3, x_4)$ である. このとき, カット集合は $cut\_set = \{v_1, v_2\}$ である. また, v_1 を根とする BDD は列関数 Φ_1 を, v_2 を根とする BDD は列関数 Φ_2 を表現する. (例終り)

3. 多出力関数の表現 [4]

前章で述べた関数分解は単一出力関数のみを対象にしており, 多出力関数にはそのままでは適用できない. そこで本論文では,

表 2　特性関数の分解表

$Z_2=(x_3,y_2)$	$Z_1=(x_1,x_2,y_1)$							
	0 0 0	0	1	0	0	0	0	1
	0 0 1	0	0	1	0	1	0	0
	0 1 0	0	0	1	0	1	0	1
	0 1 1	0	1	0	0	0	0	0
		Φ_1	Φ_2		Φ_2			Φ_3

特性関数 (characteristic function, CF) を用いた多出力関数の表現法を用いる. 特性関数を用いることで, n 入力 m 出力の多出力関数を $(n+m)$ 入力 1 出力の論理関数として扱うことができる.

[定義 5]　入力変数を $X = (x_1, x_2, \ldots, x_n)$, 多出力関数を $F = (f_1(X), f_2(X), \ldots, f_m(X))$ とする. このとき,

$$\chi(X,Y) = \bigwedge_{i=1}^{m} (y_i \equiv f_i(X))$$

を**多出力関数の特性関数**という. ここで y_i は出力を表す変数である.

多出力関数の特性関数は入力と出力の組合せのうち, 許されているものを表す. よって, 許されない入力と出力の組合せの場合は値が 0 となる.

[定義 6]　多出力関数の特性関数を $\chi(Z)$ とする. ここで, Z は入力変数または出力変数を表す. $\chi(Z)$ の分解表で, 異なる列パターンの個数を**列複雑度**という. 但し列複雑度を数えるときに, 全ての値が 0 である列パターンは無視する.

[例 3]　多出力関数の特性関数 $\chi(Z)$ において, Z の分割を (Z_1, Z_2) としたときの分解表の例を表 2 に示す. 但し, $Z_1 = (x_1, x_2, y_1)$, $Z_2 = (x_3, y_2)$ である. 定義 6 で述べたように, 特性関数の分解表では列複雑度を数えるときに, 全ての値が 0 である列パターンは無視する. よって表 2 の場合, 列複雑度 μ は 3 である.　(例終り)

[定理 3]　多出力関数の特性関数 $\chi(Z)$ において, Z の分割を (Z_1, Z_2) とし, 分解表 $M(f : Z_1, Z_2)$ の列複雑度を μ とする. ここで, $\{Z_1\} = \{X_1\} \cup \{Y_1\}$, $\{Z_2\} = \{X_2\} \cup \{Y_2\}$ とし, $\{X_1\}, \{X_2\}$ は入力変数の集合を表し $\{Y_1\}, \{Y_2\}$ は出力変数の集合を表す. このときこの多出力関数は, 関数分解により図 3 のような中間出力 Y_1 を有する回路で実現できる. また, 回路 H と G の間の接続線数は $u = \lceil \log_2 \mu \rceil$ である.

[定義 7]　関数 f が変数 x に依存するとき x は f の**サポート変数**という.

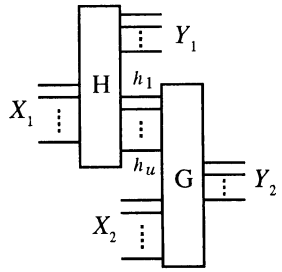

図 3　中間出力を有する関数分解

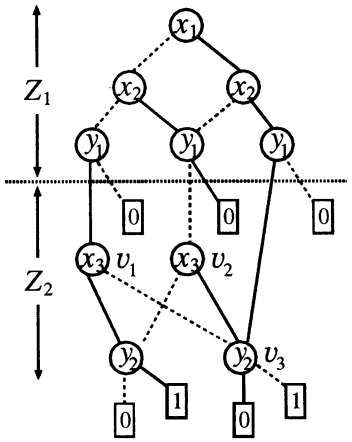

図 4　BDD_for_CF

[定義 8]　多出力関数 $F = (f_1(X), f_2(X), \ldots, f_m(X))$ の特性関数 χ を表現する BDD を, 多出力関数 F の **BDD_for_CF** という. 但し BDD の変数は, 根節点を最上位としたとき, 出力を表す変数 y_i は f_i のサポート変数の下に置く.

[定理 4]　多出力関数の特性関数 $\chi(Z)$ において, Z の分割を (Z_1, Z_2) とし, 分解表 $M(f : Z_1, Z_2)$ の列複雑度を μ, 各列関数を $\Phi_i(Z_2)(i = 1, \ldots, \mu)$ とする. カット集合 *cut_set* の各要素を根とする BDD は, 列関数 Φ_i を表現する. 但しカット集合を求めるときに, 出力を表現する変数から定数 0 に向かう枝は無視する.

[例 4]　表 2 の分解表で表現した特性関数の BDD_for_CF を図 4 に示す. 破線は 0 枝を, 実線は 1 枝を表す. ここで, 束縛集合は $Z_1 = (x_1, x_2, y_1)$, 自由集合は $Z_2 = (x_3, y_2)$ である. また, カット集合 *cut_set* は, 出力変数から定数 0 に向かう枝は無視するので *cut_set* $= \{v_1, v_2, v_3\}$ となる. また, v_1 を根とする BDD は列関数 Φ_1 を, v_2 を根とする BDD は列関数 Φ_2 を, v_3 を根とする BDD は列関数 Φ_3 を表現する.　(例終り)

BDD_for_CF に対して繰り返し関数分解を行うことで図 5 に示すような LUT のカスケード回路を得る. この図において, 各

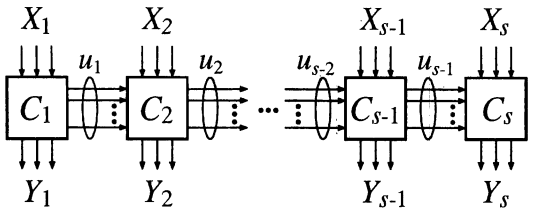

図 5　LUT カスケード

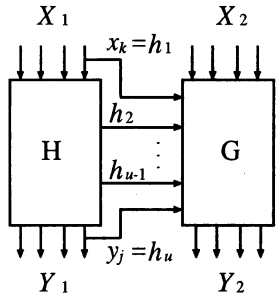

図 6　2 本のセル出力を削減した回路

段の回路 C_i を**セル**といい, X_i を**外部入力**, Y_i を**外部出力**という. また, C_i から C_{i+1} への出力線を**レール出力線**といい, レール出力線の本数 u_i を**レール数**という. さらに, 外部出力とレール出力とを合わせたセルの出力を**セル出力**といい, その本数を**セル出力数**という.

k 入力 LUT を用いて LUT カスケードを実現する際, 各段のレール数は $k-1$ 以下とする必要がある. これを**レール数制限**という. セルの最大出力数が r のとき, $u_i + |Y_i| \leq r$ とする必要がある. これを**セル出力数制限** という.

4. セル出力数の削減法

本章では, LUT カスケードにおいてセル出力数を削減するためのレール出力の符号化法を考案する. この符号化法では LUT カスケードのある段において, レール出力 h_i をその段の外部出力 y_j または外部入力 x_k と等しくする. これによりレール出力 h_i は不要となりセル出力数を削減できる. 図 6 に $h_1 = x_k$, $h_u = y_j$ とした回路を示す. 但し, $x_k \in \{X_1\}$, $y_j \in \{Y_1\}$ である.

[定義 9]　分解表の 1 つの列関数に 1 つの二進符号を割り当てる符号化法を **Strict encoding** という. また 1 つの列関数に複数の二進符号を割り当てる符号化法を **Non-Strict encoding** という.

本研究では, 関数分解を行う際に Non-Strict encoding を行うことで, レール出力を外部出力関数または入力変数と等しくする. 次に, 本章で用いる用語について述べる.

[定義 10]　多出力関数の特性関数 $\chi(Z)$ において, Z の分割を (Z_1, Z_2) とし, $\{Z_A\} \subseteq \{Z_1\}$ とする. Z_A の変数に $z_{i_1} = t_1, z_{i_2} = t_2, \ldots, z_{i_k} = t_k$ と 2 値の定数を割り当てた際の列パターンが表す論理関数を $(z_{i_1} = t_1, z_{i_2} = t_2, \ldots, z_{i_k} = t_k)$ **に対する列関数**という. また, そのときの列パターンの個数を $(z_{i_1} = t_1, z_{i_2} = t_2, \ldots, z_{i_k} = t_k)$ **に対する列複雑度**という.

[例 5]　表 2 において, $(x_1 = 0)$ に対する列関数は Φ_1 と Φ_2 であり, $(x_1 = 1, x_2 = 1)$ に対する列関数は Φ_3, $(x_1 = 0, x_2 = 0, y_1 = 1)$ に対する列関数は Φ_1 である. 同様に, $(x_1 = 0)$ に対する列複雑度は 2 であり, $(x_1 = 1, x_2 = 1)$ に対する列複雑度は 1, $(x_1 = 0, x_2 = 0, y_1 = 1)$ に対する列複雑度も 1 である.
(例終り)

4.1　1 本のセル出力の削減法

まず, 1 本のレール出力を外部出力と等しくする符号化を考える.

[定理 5]　多出力関数の特性関数 $\chi(Z)$ において, Z の分割を (Z_1, Z_2), 分解表の列複雑度を μ, レール数を $u = \lceil \log_2 \mu \rceil$ とし, 各列関数を $\Phi_i(Z_2)$ $(i = 1, 2, \ldots, \mu)$ とする. $(y_j = 0)$ に対する列複雑度が 2^{u-1} 以下でかつ $(y_j = 1)$ に対する列複雑度が 2^{u-1} 以下のとき, かつその時のみ, 符号化を工夫することによりレール出力 h と外部出力 y_j を等しくできる. ここで $y_j \in \{Z_1\}$ である.

[アルゴリズム 1]　$(h_1 = y_j$ とする符号化)

（1）　y_j が定理 5 の条件を満足するとき, $(y_j = 0)$ に対する列関数に, 0 から順に二進符号 v $(0 \leq v < 2^{u-1})$ を割り当てる.

（2）　$(y_j = 1)$ に対する列関数について, (1) において既に符号 v が割り当てられている場合, その列関数に二進符号 $v + 2^{u-1}$ を割り当てる. (1) で符号を割り当てられていない列関数には, 未使用の二進符号 t $(2^{u-1} \leq t < 2^u)$ を割り当てる.

[例 6]　図 7 はある関数を表現する BDD_for_CF の上部を表したものである. ここで出力変数から定数 0 に向かう枝は省略してある. また, 破線は 0 枝を, 実線は 1 枝を表す. 束縛集合は $Z_1 = (x_1, y_1, x_2, x_3)$ であり, カットの大きさは 5 である. $(y_1 = 0)$, $(y_1 = 1)$ に対する列複雑度は共に 3 である. $u = \lceil \log_2 5 \rceil = 3$, $2^{u-1} = 4$ よりこの分解は定理 5 の条件を満足するため, $h_1 = y_1$ と符号化できる. まず $(y_1 = 0)$ に対する列関数 Φ_3,Φ_4,Φ_5 にそれぞれ符号 (000), (001), (010) を割り当てる. 次に $(y_1 = 1)$ に対する列関数 Φ_1,Φ_2,Φ_3 に符号を割り当てる. Φ_3 には既に符号 (000) が割り当てられているため, 既に割り当てられている符号 (000) の最上位ビットを 1 にした符号 (100) を更に割り当てる. Φ_1,Φ_2 にはそれぞれ未使用の符号 (101),(110) を割り当てる. このように符号化を行えば $h_1 = y_1$ となる.
(例終り)

上と同様の操作で, レール出力と外部入力変数 x_i とを等しく

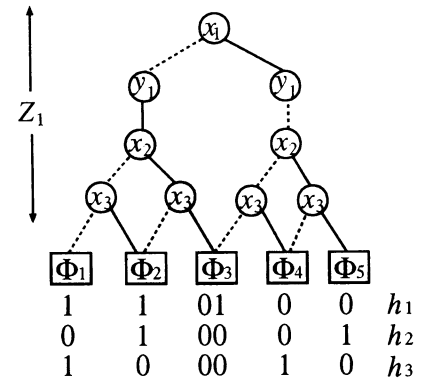

図 7　$h_1 = y_1$ となる符号化

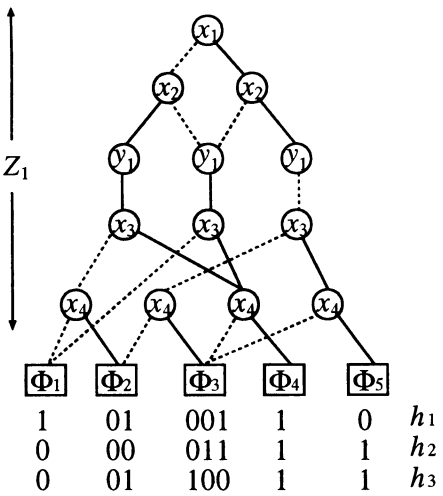

図 8　$h_1 = y_1, h_2 = x_3$ となる符号化

することも可能である [5].

4.2　複数のセル出力の削減法

定理 5 を一般化することにより複数のレール出力を外部入力や外部出力と等しくできる. そのための条件は以下のようになる.

[定理 6]　多出力関数の特性関数 $\chi(Z)$ において Z の分割を (Z_1, Z_2), 分解表の列複雑度を μ, レール数を $u = \lceil \log_2 \mu \rceil$ とし, 各列関数を $\Phi_i(Z_2)(i = 1, 2, \ldots, \mu)$ とする. $\{Z_A\} \subseteqq \{Z_1\}$, $z_{j_r} \in \{Z_A\}$ としたとき, Z_A への割り当て

$$(z_{j_1} = 0, z_{j_2} = 0, \ldots, z_{j_{p-1}} = 0, z_{j_p} = 0)$$
$$(z_{j_1} = 0, z_{j_2} = 0, \ldots, z_{j_{p-1}} = 0, z_{j_p} = 1)$$
$$\vdots$$
$$(z_{j_1} = 1, z_{j_2} = 1, \ldots, z_{j_{p-1}} = 1, z_{j_p} = 0)$$
$$(z_{j_1} = 1, z_{j_2} = 1, \ldots, z_{j_{p-1}} = 1, z_{j_p} = 1)$$

に対する列複雑度がいずれも 2^{u-p} 以下であるとき, かつそのときのみ, 符号化を工夫することによりレール出力 h_r と z_{j_r} を等しくできる. 即ち

$$h_r = z_{j_r} (r = 1, 2, \ldots, p)$$

とできる.

定理 6 を用いた, レール出力を外部入力や外部出力と等しくするアルゴリズムを示す. これにより複数のセル出力を削減できる.

[アルゴリズム 2]　(複数のセル出力の削減)

（1）　定理 6 を満足しかつ p の値を最大とする変数集合 $Z_A = \{z_{j_1}, z_{j_2}, \ldots, z_{j_{p-1}}, z_{j_p}\}$ を求める.

（2）　1 で $p \geqq 2$ の場合, Z_A の割り当て

$$(z_{j_1} = 0, z_{j_2} = 0, \ldots, z_{j_{p-1}} = 0, z_{j_p} = 0)$$
$$(z_{j_1} = 0, z_{j_2} = 0, \ldots, z_{j_{p-1}} = 0, z_{j_p} = 1)$$
$$\vdots$$
$$(z_{j_1} = 1, z_{j_2} = 1, \ldots, z_{j_{p-1}} = 1, z_{j_p} = 0)$$
$$(z_{j_1} = 1, z_{j_2} = 1, \ldots, z_{j_{p-1}} = 1, z_{j_p} = 1)$$

のそれぞれに対する列関数を調べる.

（3）　$(z_{j_1} = 0, z_{j_2} = 0, \ldots, z_{j_{p-1}} = 0, z_{j_p} = 0)$ に対する列関数から順に, 上位 p ビットが $(z_{j_1}, z_{j_2}, \ldots, z_{j_{p-1}}, z_{j_p})$ である未使用の二進符号を割り当てる.

[例 7]　図 8 はある関数を表現する BDD_for_CF の上部を表したものである. ここで出力変数から定数 0 に向かう枝は省略してある. また, 破線は 0 枝を, 実線は 1 枝を表す. 束縛集合は $Z_1 = (x_1, x_2, y_1, x_3, x_4)$ であり, カットの大きさは 5 である. $(y_1 = 0, x_3 = 0), (y_1 = 0, x_3 = 1), (y_1 = 1, x_3 = 0), (y_1 = 1, x_3 = 1)$ に対する列複雑度はそれぞれ 2 である. $u = \lceil \log_2 5 \rceil = 3$, $2^{u-2} = 2$ より, この分解は定理 6 の条件を満足するため, $h_1 = y_1, h_2 = x_3$ と符号化できる. $(y_1 = 0, x_3 = 0)$ に対する列関数 Φ_2, Φ_3 に上位 2 ビットが 00 である符号を割り当てる. ここでは, Φ_2 に符号 (000) を, Φ_3 に符号 (001) を割り当てる. 同様に $(y_1 = 0, x_3 = 1)$ に対する列関数 Φ_3, Φ_5 にそれぞれ符号 (010), (011) を割り当てる. さらに, $(y_1 = 1, x_3 = 0)$ に対する列関数 Φ_1, Φ_2 にそれぞれ符号 (100), (101) を割り当てる. 最後に,$(y_1 = 1, x_3 = 1)$ に対する列関数 Φ_3, Φ_4 にそれぞれ符号 (110), (111) を割り当てる. このように符号化を行えば $h_1 = y_1, h_2 = x_3$ となる.　(例終り)

5.　実験結果

各ベンチマーク関数に対し, LUT カスケードを設計する際に本手法を適用した場合の結果を表 3 に示す. このとき LUT の最大入力数 k を 13, セルの最大出力数を 8 とした.

- PC: IBM PC/AT 互換機
- CPU: Pentium4 xeon 2.8GHz
- memory: 4GB
- OS: Linux 2.4.2

表 3 の Strict は Strict encoding のみを用いた場合の結果を表

表 3 Strict encoding と本手法の比較

Name	In	Out	Strict			Non-Strict		
			Cas	Lvl	LUTs	Cas	Lvl	LUTs
5xp1	7	10	1	2	13	1	2	11
C1908	33	25	5	5	140	5	5	134
apex2	39	3	1	5	20	1	5	19
apex3	54	50	2	26	234	2	25	215
apex7	49	37	2	16	139	2	11	96
b9	41	21	1	7	45	1	6	32
bw	5	28	1	7	55	1	5	37
c8	28	18	2	6	49	2	6	47
cc	21	20	1	4	31	1	4	27
cm150a	21	1	1	2	5	1	2	3
decode	5	16	1	3	23	1	3	21
duke2	22	29	1	9	69	1	7	55
frg2	143	139	6	19	414	6	16	333
lal	26	19	1	4	29	1	4	27
ldd	9	19	1	4	30	1	4	23
misex2	25	18	1	5	31	1	4	28
misex3	14	14	2	3	29	2	3	25
mux	21	1	1	2	5	1	2	3
pcler8	27	17	2	6	46	2	4	32
rot	135	107	12	16	610	12	12	526
seq	41	35	1	14	108	1	11	83
term1	34	10	1	5	25	1	5	23
too_large	38	3	1	5	20	1	5	19
ttt2	24	21	2	7	58	2	7	57
vda	17	39	1	13	98	1	9	68
vg2	25	8	1	3	14	1	3	11
unreg	36	16	1	4	25	1	4	25
x1	51	35	4	6	121	4	6	106
x3	135	99	5	17	307	5	17	297
x4	94	71	3	18	194	3	9	147

Strict : Strict encoding のみを行った場合
Non-Strict : 本手法を適用した場合

し, Non-Strict は Non-Strict encoding を用いてセル出力数の削減を行った場合の結果を表す. また, Cas はカスケードの本数, Lvl は最大段数, LUTs は LUT の出力数を表す.

本手法により, ほとんどの場合で LUT の出力数を削減できた. 特に apex7, bw, pcler8 は LUT の出力数を 30% 以上削減できた. また, LUT の出力数と最大段数を同時に削減できる場合があることも分かった. カスケードの段数を削減するには, 関数分解を行う際に束縛集合 X_B の要素数 $|X_B|$ をできるだけ大きくする必要がある. しかし $|X_B|$ を大きくするとセル出力数が大きくなり過ぎ, セル出力数制限に収まらず関数分解ができないときがある. このような場合にも, 本手法を用いてセル出力数を削減することで関数分解が可能になることがある. よって, 本手法を用いることでカスケードの段数を削減できる場合がある.

6. 結　　論

本論文では, LUT カスケードのレール出力の符号化を工夫することで, セル出力数を削減する手法を示した. これにより LUT の出力数を削減できた. また, LUT の出力数とカスケードの段数を同時に削減できる場合があることも示した.

LUT カスケードを実現する際, 使用できるメモリ量には上限がある. だが本手法を用いれば LUT の出力数を削減できるので, LUT の入力数をより大きくしても回路を実現できる場合がある. 一般に, LUT の入力数を大きくすればより少ない段数でカスケードを作ることができる. よって, メモリ量削減よりも高速化を図りたい場合にも本手法は有効であるといえる.

謝　　辞

本研究は一部, 日本学術振興会, 科学研究費補助金及び文部科学省・北九州地域・知的クラスタ創成事業補助金による. 明治大学の井口幸洋助教授には, 熱心にご討論頂いた.

文　　献

[1] R. K. Brayton, "The future of logic synthesis and verification," *in S. Hassoun and T. Sasao (e.d.), Logic Synthesis and Verification,* Kluwer Academic Publishers, Oct. 2001.

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

[3] 笹尾 勤 著『論理設計 スイッチング回路理論』, 近代科学社, 1995.

[4] T. Sasao and M. Matsuura, "A method to decompose multi-output logic functions," 41st Design Automation Conference, San Diego, CA, USA, June 2-6, 2004, pp.428-433.

[5] 郷司隼人, 笹尾勤, 松浦宗寛, "LUT カスケードにおける LUT 数削減法", 電子情報通信学会 VLSI 技術研究会, VLD2001-99, 北九州 (2001-11).

[6] H. Sawada, T. Suyama and A. Nagoya "Logic synthesis for look-up table based FPGAs using functional decomposition and boolean resubstitution," IEICE Trans. Information and Systems, Vol. E80-D, No.10, pp.1017-1023, Oct. 1997.

[7] ミシュチェンコ, 笹尾, "レイル数に制限のある LUT カスケードの論理合成法," 電子情報通信学会 VLSI 技術研究会, VLD2002-99, 琵琶湖 (2002-11).

[8] R. Murgai, F. Hirose, and M. Fujita, "Logic synthesis for a single large look-up table," *Proc. International Conference on Computer Design,* pp.415-424, Oct. 1995.

[9] Y-T. Lai, M. Pedram and S. B. K. Vrudhula, "BDD based decomposition of logic functions with application to FPGA synthesis," 30th ACM/IEEE Design Automation Conference, June 1993.