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

2005－SLDM－122 (31)
2005／12／2

LUTカスケード・エミュレータ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 LUTカスケード・エミュレータにおいてレール出力の符号化法を工夫することにより, セルの総出力数を削減する方法を示す. LUTカスケード・エミュレータでは, 各セルの入力数を減らしてメモリ量を削減するとセル数が増加し評価時間が大きくなる. しかし本手法によりレール出力の符号化を工夫してセルの総出力数を削減することで, セル数を増加させることなく必要メモリ量を削減できる. 本手法では多出力関数を表現するために, 特性関数を表すBDD(BDD_for_CF) を用いる. 実験により多くのベンチマーク関数において, 必要メモリ量を平均約25%削減することができた.

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

An Encoding Method for Rail Outputs in LUT Cascade Emulators

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 numbers of outputs of cells for an LUT cascade emulator with intermediate outputs by considering encoding. The amount of memory decreases with the reduction of the number of inputs for each cell. However, it increases the number of cells in LUT cascade emulators, and increases the evaluation time. However, our approach reduces the amount of memory by reducing outputs of cells without increasing the number of cells. We use a BDD for characteristic function (BDD_for_CF) for representing multiple-output logic functions. Experimental results show that our approach reduces the amount of memory ,on the average, by 25%.

Key words Logic synthesis, BDD_for_CF, Functional decomposition, LUT cascade emulator, 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. 再構成可能なアーキテクチャの最も簡単な例としてRAMがある. これは任意の論理関数を実現可能であるが, 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. しかし, 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.

これらの問題点を解決するアーキテクチャとしてLUTカスケードが開発されている[2]. LUTカスケードは図1のような構造をしており, 多出力論理関数をLUT(Look-Up Table) のカスケードで表現する. 各段のLUTのことを**セル**といい, あるセルから次のセルへ向かう出力を**レール出力**という. LUTカスケードは配線を隣同士のセル間のみに限定しているので配線領域の問題を解決できる. しかし, LUTカスケードではセルの最大入力数・最大出力数を一旦決定すると, 実現できる関数が限られる. そこで, 本研究ではLUTカスケード・エミュレータ (図2) を用いる[2]. これはLUTカスケードを模擬するものであり, LUTカスケードに比べより広い範囲の関数を実現できる.

LUTカスケード・エミュレータでは総セル数が大きくなれば評価時間も大きくなり, セルの総出力数が大きくなればメモリ

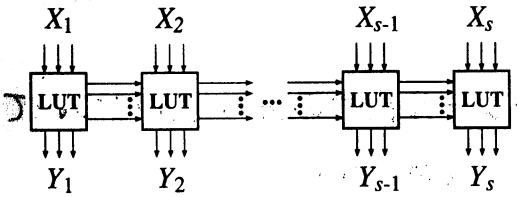

図1 LUT カスケード

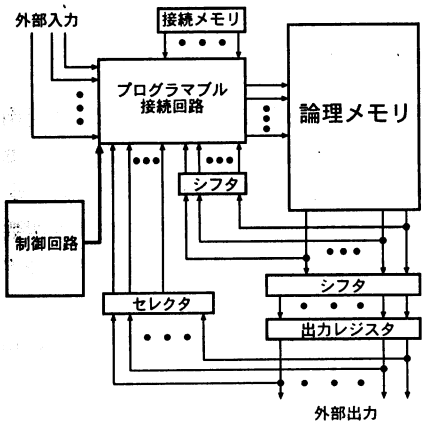

図2 LUT カスケード・エミュレータ

量も大きくなる. 手法[3]ではレール出力の符号化を工夫することで, レール出力を同一段の外部入力と等しくした. これによりセルの総出力数を10%程度削減できた. 本論文では, この符号化法を一般化することで同一段の入出力のみでなく前段以前の入出力・レール出力と等しくする. これによりLUTカスケード・エミュレータの必要メモリ量を削減可能である.

2. 論理関数の関数分解[4]

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.

表1 分解表

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

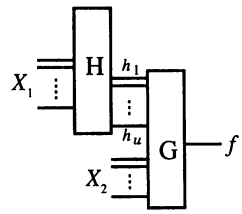

図3 関数分解

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

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

と表現でき, 図3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) といい, 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 $X_1=(x_1,x_2)$, 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)

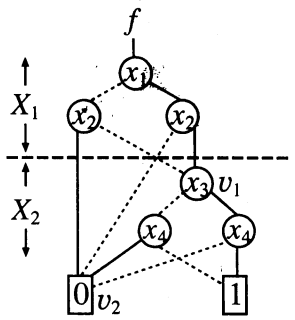

図4　カット集合

3. 多出力関数の表現 [5]

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，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では，入力と出力の組合せが許されている場合には値は 1 となり，入力と出力の組合せが許されていない場合には値が 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 5 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

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 1 1 1
		0 0 1 1 0 0 1 1
		0 1 0 1 0 1 0 1
0 0		0 1 0 0 0 0 0 1
0 1		0 0 1 0 1 0 0 0
1 0		0 0 1 0 1 0 0 1
1 1		0 1 0 0 0 0 0 0
		Φ_1 Φ_2 Φ_2 Φ_3

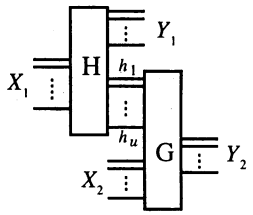

図5　中間出力を有する関数分解

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．

[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 6 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 LUT のカスケード回路を得ることができる．図 1 は関数分解を $s-1$ 回繰り返して得られる回路である．

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

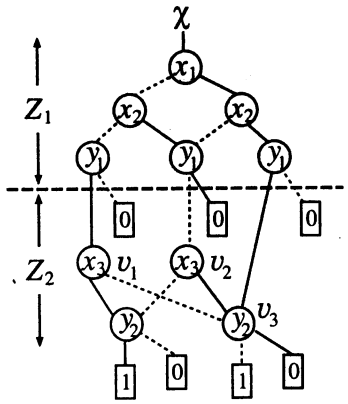

図6 BDD_for_CF

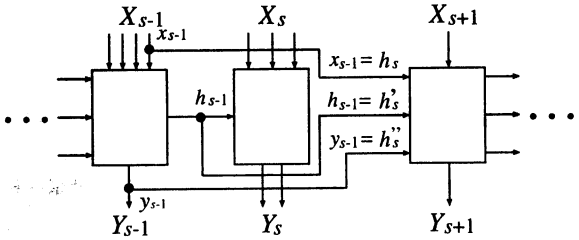

図7 レール出力を前段の外部入出力・レール出力と等しくした回路

カスケードのある段において, レール出力 h_s をその段の外部入力 x_s または外部出力 y_s もしくはその段以前の外部入力 x_{s-i} や外部出力 y_{s-i}, レール出力 h_{s-i} と等しくする. これによりレール出力 h_s は不要となりセル出力数を削減できる. 図7に $h_s = x_{s-1}, h'_s = y_{s-1}, h''_s = h_{s-1}$ とした回路を示す.

[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, レール出力を外部入出力またはレール出力と等しくする. 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)

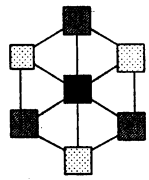

図8 3色で彩色可能なグラフ

[定義11] 無向グラフ G が与えられたとき, G の各点を k 色で塗り, しかもどの隣接する2点も異なる色で塗分けられるとき, G は k 色で**彩色可能**であるという.

[例6] 図8は, 7点からなる無向グラフである. これは図8に示すように3色で彩色可能である. (例終り)

次に, セル出力数の削減法について述べる.

[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. このとき節点を $\Phi_i(Z_2)(i = 1, 2, \ldots, \mu)$ とする無向グラフを考える. まず,

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

に対する列関数間を全て枝で結ぶ. 同様にして,

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

に対する列関数間を全て枝で結ぶ. 同様の操作を

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

のそれぞれに対しても行なう.

このようにして得られたグラフが 2^{u-p} 色以下で彩色可能であるとき, 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5を用いた, レール出力を同一段の外部入出力また前段以前の外部入出力・レール出力と等しくするアルゴリズムを示す.

[アルゴリズム1] (セル出力数の削減アルゴリズム)

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.

(1) 定理5を満足しかつ p の値を最大とする変数集合

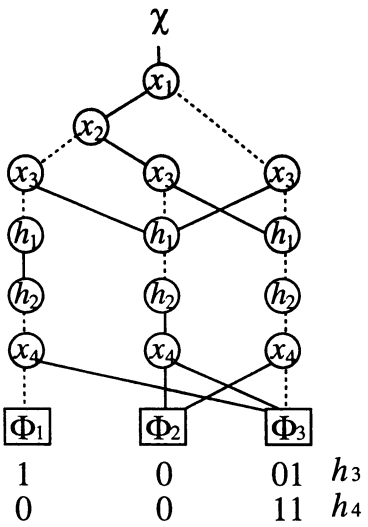

図 9 Non-Strict encoding

$Z_A = \{z_{j_1}, z_{j_2}, \ldots, z_{j_{p-1}}, z_{j_p}\}$ を求める.

（2） 定理 5 の要領で列関数 $\Phi_i(Z_2)(i = 1, 2, \ldots, \mu)$ を節点としたグラフを作り, 彩色を行なう.

（3） 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に対し, 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. このとき下位 $u - p$ ビットは, (2) で同色に彩色した列関数は等しくなるようにする.

本手法ではレール出力を同一段の外部入出力のみでなく前段以前の外部入出力・レール出力とも等しくする. そのため符号を割り当てるときには前段以前の外部入出力・レール出力を表す変数を含む BDD_for_CF を使用する. 以下にレール出力を前段のレール出力と等しくする符号化の例を示す.

[例 7] 図 9 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. x_1, x_2, x_3 は 1 段目の外部入力, h_1, h_2 は 1 段目のレール出力, x_4 は 2 段目の外部入力である. 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. カットの大きさは 3 であり, Φ_i で表した節点は列関数に対応する. まず, 列関数 Φ_1, Φ_2, Φ_3 を節点とするグラフを作る. 次に $(h_1 = 0)$ に対する列関数 Φ_2, Φ_3 間を枝で結び, $(h_1 = 1)$ に対する列関数 Φ_1, Φ_3 間を枝で結ぶ. すると図 10 のような 2 色で彩色可能なグラフを得る.

$h_1 = 0$ に対する列関数 Φ_2, Φ_3 に, 上位 1 ビットが 0 である符号を割り当てる. ここでは Φ_2 には符号 00 を, Φ_3 には符号 01 を割り当てた. 次に, $h_1 = 1$ に対する列関数 Φ_1, Φ_3 に, 上位 1 ビットが 1 である符号を割り当てる. ここで, 図 10 において Φ_1 と Φ_2 は同色なので Φ_1 と Φ_2 の下位ビットは等しくなるようしなければならない. よって Φ_1 には符号 10 を割り当てる. また, 同一の列関数に割り当てられる符号も下位ビットは等しくしなければならないので, Φ_3 に符号 11 を割り当てる. このように符号化を行えば図 11 のように $h_1 = h_3$ となる. （例終り）

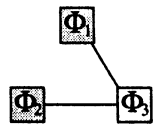

図 10 符号の下位ビット決定のための彩色グラフ

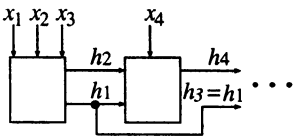

図 11 $h_1 = h_3$ とした回路

5. LUT カスケード・エミュレータ

LUT カスケード・エミュレータは図 2 の構造を持つ. 全てのセルのデータを格納する大きなメモリである**論理メモリ**, セル間の接続情報を保持する**接続メモリ**, セルの入力を生成する**プログラマブル接続回路**, 外部出力を蓄積する**出力レジスタ**などから構成されている.

LUT カスケード・エミュレータにおいてある段のレール出力 h_s を前段以前のレール出力 h_{s-i} と等しくする際, h_{s-i} の保存場所が問題となる. 図 2 の出力レジスタは, 外部出力を蓄積するもので, 最終段の評価が行なわれるまで最終段の外部出力数のビットは未使用である. 従って, この未使用部に h_s の値を格納しておくことで h_s と h_{s-i} を等しくすることが可能である.

6. 実験結果

MCNC89 ベンチマーク関数を LUT カスケード・エミュレータにマッピングする際に, 既存手法, また本手法を適用したときのセルの総出力数, 必要メモリ量を表 3 に示す. Name は関数名, In は入力数, Out は出力数を表す. Cell's out はセルの総出力数を表し, Mem はメモリ量で単位は [Kbit] である. また, Str_Enc はレール出力の削減を行なわなかった場合を, 既存手法は文献 [3] の手法に基づきレール出力を同一段の外部入出力のみと等しくした場合を, 本手法はレール出力を同一段の外部入出力やその段以前の外部入出力・レール出力と等しくした場合を示す. 実験では, マッピングする際に各セルの最大入力数を 8 とし, 各セルの最大出力数を 16 とした. また, メモリパッキング [7] を適用した. 但し, 本手法はセル数が 2 以下のカスケードには適用できないのでセル数が 2 以下で実現できた関数は省いた. 計算機の環境は次の通りである.

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

表 3　Strict encoding・既存手法・本手法の比較

Name	In	Out	Str_Enc		既存手法		本手法	
			Cell's out	Mem	Cell's out	Mem	Cell's out	Mem
C432	36	7	163	64.00	148	64.00	129	64.00
alu4	14	8	50	16.00	44	16.00	43	16.00
apex5	117	88	368	128.00	351	128.00	239	64.00
apex6	135	99	291	128.00	252	128.00	169	64.00
cht	47	36	64	32.00	60	16.00	50	16.00
comp	32	3	11	2.75	11	2.75	11	2.75
cu	14	11	17	8.00	15	3.75	15	3.75
des	256	245	1638	512.00	1510	512.00	842	256.00
duke2	22	29	60	16.00	57	16.00	51	15.00
example2	85	66	317	128.00	311	128.00	116	32.00
frg2	143	139	487	128.00	449	128.00	291	128.00
lal	26	19	30	8.00	29	8.00	27	8.00
misex3	14	14	60	16.00	56	16.00	51	16.00
mux	21	1	12	3.00	8	2.00	7	1.75
pair	173	137	1114	512.00	1015	256.00	802	256.00
pcler8	27	17	47	16.00	43	16.00	37	16.00
seq	41	35	173	64.00	163	64.00	105	32.00
tcon	17	16	20	6.00	17	6.00	16	4.00
term1	34	10	80	32.00	73	32.00	51	16.00
vg2	25	8	21	8.00	17	7.50	17	7.50
x1	51	35	165	64.00	147	64.00	130	32.00
x4	28	71	271	128.00	257	128.00	198	64.00

Str_Enc : レール出力の削減を行なわなかった場合
既存手法 : レール出力を同一段の外部入出力のみと等しくした場合
本手法 : レール出力を同一段の外部入出力, また前段以前の外部入出力・レール出力と等しくした場合

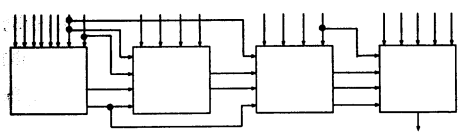

図 12　mux の LUT カスケード回路

• OS: Linux 2.4.2

表 3 から明らかなように, Str_Enc, 既存手法, 本手法の順に Cell's out が減少している. 本手法を用いることで, セル数が 3 以上となる関数のほとんどでセルの総出力数を削減できた. セルの総出力数を削減することで, 多くの関数において必要メモリ量を削減できた. 特に example2 では必要メモリ量を 75% 削減できた. 例としてベンチマーク関数 mux を本手法を用いて設計した際の回路を図 12 に示す. 本手法により同一段の入力だけでなく前段の入力・レール出力とも等しくできていることがわかる.

表 3 中の関数で最も計算時間の大きかったものは des であり, レール出力を削減しなかった場合は 7.4[sec], 既存手法を用いた場合は 7.5[sec], 本手法を用いた場合は 414.5[sec] だった. 設計のための計算時間であるが, 一般にレール出力と等しくしようとする対象がそのレール出力のセルから離れているほど計算時間が長くなる. 変数順序が $(z_1, z_2, \ldots, z_k)$ である BDD に対し, 自由集合を $Z_F = \{z_j, z_{j+1}, \ldots, z_k\}$ とする関数分解を考える. レール出力を $z_i (i < j)$ と等しくできるか判定する為に必要な計算時間は, $z_i, z_{i+1}, \ldots, z_j$ に対応する節点数に比例する. そのため, 計算時間を小さくしたいときにはレール出力と等しくする対象を制限することが必要となる.

7. 結　　論

本論文では, レール出力の符号化法を工夫することでセルの総出力数を削減する手法を示した. 従来手法 [3] では同一段の入出力のみレール出力と等しくしたが, 本手法では, 同一段の外部入出力のみでなく前段以前の外部入出力・レール出力と等しくできる. これにより, LUT カスケードエミュレータの必要メモリ量を平均で約 25%削減できた.

LUT カスケードでは, レール出力を前段の外部入出力やレール出力と等しくすると図 7 のように配線が複雑になってしまう. しかし LUT カスケード・エミュレータでは, 実際にセル間を配線するのではなく接続メモリの配線情報をもとにプログラマブル接続回路でセルの入力を生成するので, 配線は複雑にならない. よって本手法は LUT カスケード・エミュレータに向いている.

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.

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, Y. Iguchi, and M. Matsuura, "LUT Cascades and Emulators for Realization of Logic Functions," RM2005, Tokyo, Japan, Sept.5 - Sept.6, 2005, pp.63-70.

[3] 永安伸也, 笹尾勤, 松浦宗寛, "LUT カスケードにおけるレール出力の符号化法について", 電子情報通信学会 VLSI 技術研究会, VLD2004-86, 北九州 (2004-12).

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

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

[6] 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).

[7] T. Sasao, M. Kusano, and M. Matsuura, "Optimization methods in look-up table rings," International Workshop on Logic and Synthesis, (IWLS-2004), June 2-4, 2004, Temecula, California, U.S.A., pp.431-437.

[8] 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).

[9] 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.

[10] 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.

[11] J.A.Bondy, U.S.R.Murty 著 立花 俊一, 奈良 知恵, 田澤 新成 共訳『グラフ理論への入門』, 共立出版株式会社, 1991.