LUT カスケードにおける LUT 数削減法

郷司 隼人 [1]　笹尾 勤 [1,2]　松浦 宗寛 [1]

[1] 九州工業大学 情報工学部
[2] 九州工業大学 マイクロ化総合技術センター

あらまし: RAM とシーケンサを用いた多出力論理関数の実現する方法を示す. まず, 多出力関数を ECFN(encoded characteristic function for non-zeros) で表現し, それを LUT(look-up table) カスケードで実現する. LUT カスケードにおいて, 符号化を工夫することにより LUT の個数を削減することができる. LUT の個数を削減すれば, 論理関数を実現するのに必要なメモリの量を削減できる. 本論文では中間変数を 1 変数関数に変換する符号化法について述べる. 実験により本手法が多くのベンチマーク関数において LUT の個数を 10%程度削減できることを示す.

キーワード : BDD, 関数分解, カスケード実現, 符号化問題, non-strict encoding.

On a Method to Reduce the Number of LUTs in LUT cascades

Hayato GOUJI[1], Tsutomu SASAO[1,2], and Munehiro MATSUURA[1]

[1]Department of Computer Science and Electronics, Kyushu Institute of Technology
[2]Center for Microelectronic Systems, Kyushu Institute of Technology

Abstract: A realization of multiple-output logic function using a RAM and a sequencer is presented. First, a multiple-output function is represented by an encoded characteristic function for non-zeros (ECFN), then it is impremented by a cascade of look-up tables (LUTs). In a cascade of LUTs, we can reduce the number of LUTs by considering encoding. So, the amount of memory that is necessary to imprement logic function can be reduced. This paper shows an encoding method that transforms intermediate variables into one-variable functions. Experimental results show that our approach can reduce the number of LUTs about 10%.

Key words: BDD, Functional decomposition, Cascade realization, Encoding problem, Non-strict encoding.

1 はじめに

LSIの微細化がこのまま進むと, 従来の論理合成法は, 規則的構造にしか適用できないと予想されている [1]. 従来の論理合成手法では, 使用環境や製造時のばらつきのため, カスタム設計した場合, 回路動作の保証が困難となる. また, クロストーク雑音, 電磁誘導効果, 遅延予想が極めて複雑になり, カスタム設計が可能としても, 極めて高価になる. 従って, それほど, 設計コストをかけられない製品には, 規則的回路構造を採用せざるを得ない. 規則的回路とは, 繰り返し構造をもつ回路であり, 次の特長をもつ.

- 回路は大域的に見た際, 均一であり, 絶縁物の厚みなど製造時のばらつきを削減でき, 遅延予想が容易である.
- 回路は, 繰り返し部分を1度設計すればよいので, 人手設計可能であり, DSM問題も回避できる.
- 再プログラム化も容易なので, 一つの回路が種々の応用に利用できる.
- 拡張が容易.
- 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.

メモリの集積度が, 年々, 指数関数的に上がっているのは, それが規則的構造を有するからである. また, 集積度をぎりぎりまで上げれるのは, 基本セルを人手設計できるからである.

ここでは, 簡単のために, 単一の n 変数論理関数の実現問題を考える. また, ハードウエアのモデルとして次のものを考える.

1. ROM/RAM
2. PLA
3. FPGA
4. ROM/RAM+汎用マイクロプロセッサ

このうち, ROM/RAMでは, 任意の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 に比例して増え, 実用的ではない. PLAも基本的に同じである. FPGAでは, 論理部と配線部が変更可能であり, 現実的回路が実現可能であり, 極めて有望である. ただし, a) 大規模回路の場合, 配線部分の領域が大きくなる. b) 遅延の予想が困難, という技術上の問題あり, また, c)FPGAの基本特許が, Xilinx社とAltera社に抑えられている, という知的所有権上の問題もある. FPGAでは, LUTの入力数に関わらず, 回路規模の増大とともに, 配線の問題が出てくる. 配線部分は, 不規則であり, 遅延時間の予測が困難, クロストークの発生などカスタム設計と同様な本質的問題をはらむ. 4は, 通常のマイクロプロセッサのモデルであり, ブランチング・プログラムを実行することにより, 論理関数を実現できる [11]. この場合, 遅延の予測は容易であり, 回路はそれ程大きくならない. しかし, n 変数論理関数の計算のために n ステップ必要となり, 消費電力が多い割には, 低速である. ここでは, 1と4の中間の方法(LUTカスケード法)を用いる [7, 12]. LUTカスケード法では,

表 1.1: 種々の手法の比較

	性能	面積	設計コスト
カスタム論理回路	大大	小小	大大
ROM/RAM	大中	大大	小小
FPGA	大中	中	中
LUTカスケード	中小	小	小
ROM+マイクロプロセッサ	小	中小	小小

1. 論理関数をRAM/ROMのカスケードで表現する.
2. 配線部分は, 最小にするが, それも, 直接実現せず, 仮想的(ソフトウエア的) に行う.
3. 制御部は, 専用ハードウエアを用いる.

これにより, 配線の問題が解決できる. 従って, 論理設計では, 回路の段数の削減と, ROM/RAM容量の削減が問題となる. LUT数が少なければ, 必要なハードウェア量を削減できる. 本研究では, ROM/RAM容量をできるだけ削減するために, なるべくLUTの個数を少なくした回路で論理関数をカスケード実現する手法を示す. 予備実験 [3] によると, LUTカスケード法は, 通常のブランチング・プログラムに比べて, 5〜10倍高速という結果が出ている. これらを纏めると, 表1のようになる. 本手法の応用分野として, カスタム論理回路ほど高性能・低電力を狙わないが, マイクロプロセッサを用いてソフトウエア的に実現した回路よりも高速であるもの, また低消費電力が要求されるものを考えている. LUTカスケード法では

- 回路構造が規則的なので, 配線部の実現が容易.
- メモリを時分割で, LUTとして使用.
- メモリでは多ビット同時に読み出せることを利用し, 並列性を上げる.
- 「ビットパッキング」という手法で, メモリに, 多数のLUTを詰め込む.
- ECFN(Encoded Characteristic Function for Non-zero) を用いて, 多出力関数を単一出力関数に変換し, 論理圧縮を行う
- 利用できるメモリ量に応じて性能(段数)を調整する.

等がポイントである. LUTカスケードは, 数学的には万能である. しかし, 全てをLUTカスケードで対応

表 2.1: 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	0	1	1	0
0 1	1	1	1	1
1 0	0	1	1	0
1 1	0	0	0	0

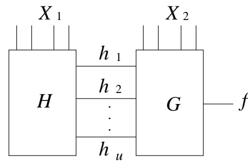

図 2.1: 関数分解

するのは能率が悪い．従って，実際の応用では他の手法（ROM，RAM, FPGA，マイクロプロセッサ等）との併用が現実的である．

論理ゲートを ROM/RAM で実現するというアイデアは，1995 年に, Murgai-Hirose-Fujita が発表している [6]．彼らの論理設計方法では，通常の FPGA の設計法を流用しているので，回路は，ランダム論理となり，配線が複雑になる．また，イベント・ドリブン方式で出力値を評価するので，評価時間は，LUT 数に比例する．一方，LUT cascasde 法では，規則的な回路構造（カスケード）を利用するので，配線は規則的である．また，ROM/RAM の多ビットの出力を同時に利用するので，高速にできる. 評価時間は, カスケードの段数に等しい. また，Murgai-Hirose-Fujita の方法よりも，少ない LUT で実現できる．

2　論理関数のカスケード実現

本章では, 論理関数を LUT カスケードで実現する手法を紹介する [7, 12].

定義 2.1 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, $X = (X_1, X_2)$ を X の分割 *(partition)* という. X の変数の個数を $|X|$ で表す.

定理 2.1 論理関数 $f(X)$ に対して, X の分割を $X = (X_1, X_2)$ とする. 2^{n_1} 列 2^{n_2} 行の表で, 各行, 各列に *2* 進符号のラベルをもち, 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 を束縛変数 *(bound variable)*, X_2 を自由変数 *(free variable)* という.

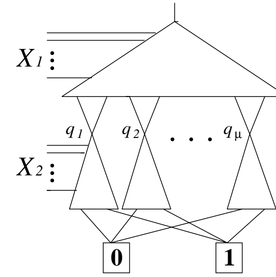

図 2.2: BDD を用いた関数分解

例 2.1 $f(X)$ を *4* 変数関数, $X = (X_1, X_2)$ を X の分割, 但し, $X = (x_1, x_2)$, $X = (x_3, x_4)$ とする. 表 *2.1*は分解表の例である. *(例題終)*

定義 2.2 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.

列複雑度は, 分解表に対して定義され, 入力変数の分割 $X = (X_1, X_2)$ に依存する.

補題 2.1 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

$$f(X) = g(h_1(X_1), h_2(X_1,), \ldots, h_u(X_1), X_2) \quad (2.1)$$

と表現でき, 図 *2.1*の回路構造で実現可能である. ここで, $u = \lceil \log_2 \mu \rceil$ である.

補題 2.2 *[10]* (X_1, X_2) を X の分割とし, 関数 f の *BDD* が, 図 *2.2*に示すように二つのブロックに分割できるとする. 下位ブロックの k 個の節点が, 上位ブロックと隣接しているとする. また, 関数 f の分解表 $M(f : X_1, X_2)$ の列複雑度を μ とする. このとき, $\mu = k$ である.

定義 2.3 論理関数 f を表現する *BDD* の x_k と x_{k+1} の間を交差する枝の数を第 k 段目での *BDD* の幅という. ここで同じ節点を指している枝は *1* と計数する. *(*図 *2.2* の *BDD* の幅は μ*)*

定理 2.2 n 変数関数 f を実現する *BDD* の幅の最大値を μ_{max} とする. $u = \lceil \log_2 \mu_{max} \rceil \leq k - 1$ ならば, f は図 *2*で示すような k 入力*-LUT* のカスケード回路網で実現可能である. 図 *2*は, 関数分解を $s - 1$ 回繰り返して得られる回路である.

3　多出力関数の表現

前章で述べた, 関数分解による回路合成法は, 単一出力関数には適用できるが, 多出力関数では, そのままでは適用できない. m 値出力関数の場合, MTBDD では 2^m の終端節点が必要な場合もあり, BDD を構築するには大きすぎることが多い. また特性関数 (characteristic

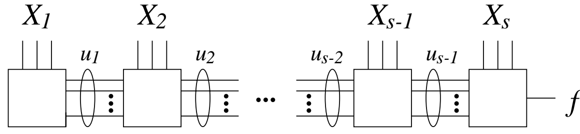

図 2.3: 論理関数のカスケード実現

function,CF) を用いた多出力関数の表現法も開発されているが, CF の BDD も大きくなり過ぎてしまう場合が多い. 従って, 本論文では以下の手法を用いて多出力関数を表現する.

定義 3.1 *[8]* m 出力関数を f_j $(j=0,1,\ldots,m-1)$ とする. 非ゼロ出力の符号化された特性関数 *(The encoded characteristic function for non-zeros, ECFN)* を

$$ECFN = \bigvee_{j=0}^{w-1} z_{w-1}^{b_{w-1}} z_{w-2}^{b_{w-2}} \cdots z_0^{b_0} f_j,$$

と定義する. ここで $\boldsymbol{b}=(b_{w-1},b_{w-2},\ldots,b_0)$ は整数 j の *2* 進表現であり, $w=\lceil \log_2 m \rceil$ である. $z_0,z_1,\ldots,z_{w-1}$ は出力を表現するための補助変数である.

例 3.1 $m=8$ の場合を考える. 出力の集合を表す補助変数を z_0,z_1,z_2 とする. このとき *8* 出力関数 $(f_0,f_1,\ldots,f_7)$ の *ECFN* は, 次のように表現できる.

$$\begin{aligned} ECFN &= \bar{z}_2\bar{z}_1\bar{z}_0 f_0 \vee \bar{z}_2\bar{z}_1 z_0 f_1 \vee \bar{z}_2 z_1 \bar{z}_0 f_2 \vee \bar{z}_2 z_1 z_0 f_3 \vee \\ & \quad z_2\bar{z}_1\bar{z}_0 f_4 \vee z_2\bar{z}_1 z_0 f_5 \vee z_2 z_1 \bar{z}_0 f_6 \vee z_2 z_1 z_0 f_7 \end{aligned}$$

(例題終)

ECFN は時分割 (time domain multiplexing) により n 入力 m 出力関数を表す $(n+w)$ 入力 1 出力関数である. ECFN の BDD を作成した場合, 補助変数と入力変数を混合させることにより, BDD を小さくできる [8].

4 中間変数の簡単化

本章では, LUT カスケードにおいて LUT 数を削減するために中間変数 h_1 が $h_1=x_j$ となるような符号化法を考案する. これは与えられた関数 f を,

$$f(X_1,X_2)=g(h_2(X_1),h_3(X_1),\ldots,h_u(X_1),x_j,X_2) \tag{4.1}$$

という形で表す符号化である. ここで, $x_j \in X_1$ である. これは非分離的関数分解と考えることもできる. (図 4.1)

定義 4.1 分解表の一つの列パターンに一つの符号を割り当てる符号化法を *strict encoding* という. また *1* つの列パターンに複数の符号を割り当てる符号化法を *non-strict encording* という.

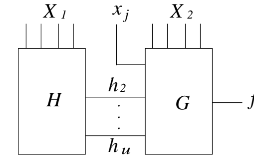

図 4.1: h_1 を 1 変数関数に変換した回路

本研究では, 関数分解を行う際 non-strict encoding を行い, 中間変数を 1 変数関数に変換する.

定理 4.1 *[5]* 分解表の列複雑度を μ, $u=\lceil \log_2 \mu \rceil$ とする. 分解 $f(X_1,X_2)=g(h(X_1),X_2)$ における 2^{n_1} 個の部分関数の同値類が生成する関数を $\Psi_i(X_1)$ $(i=0,1,\ldots,\mu-1)$ とする. $\Psi_i(|x_j=0)$ $x_j \in \{X_1\}$ の異なる関数の個数が 2^{u-1} 以下 かつ $\Psi_i(|x_j=1)$ の異なる関数の個数が 2^{u-1} 以下のとき, そのときのみ関数 f は、式 *4.1* の形で分解可能である. このとき, h_1 は *1* 変数関数となり, 回路は不要である.

アルゴリズム 4.1 *(1* つの中間変数の簡単化*)*

1. Ψ_i ここで $(i=0,1,\ldots,\mu-1)$, が定理 *4.1* の条件を満足するとき, $\Psi_i(|x_j=0)$ の関数に *0* から順に符号 v $(0 \le v < 2^{u-1})$ を割り当てる.
2. $\Psi_i(|x_j=1)$ の関数について, *1* において既に符号 v が割り当てられている場合, その関数に符号 $v+2^{u-1}$ を割り当てる.
3. 符号をまだ割り当てていない $\Psi_i(|x_j=1)$ に未使用の符号 t $(2^{u-1} \le t < 2^u)$ を割り当てる.

例 4.1 図 *4.2* は *BDD* の上部を表したものである. ここで破線は *0* 枝, 実線は *1* 枝を示す. このとき第 *4* 段目での *BDD* の幅は *5* である. $\Psi_i(|x_1=0)$, $\Psi_i(|x_1=1)$ が生成する異なる関数の個数は共に *4* である. $u=\lceil \log_2 5 \rceil=3$, $2^{u-1}=4$ より, Ψ_i は定理 *4.1* の条件を満足するため, $h_1=x_1$ と変換できる. $\Psi_i(|x_1=0)$ が生成する関数 $\Psi_0,\Psi_1,\Psi_2,\Psi_3$ に符号を (000) から順に割り当てる. 次に $\Psi_i(|x_1=1)$ が生成する関数 $\Psi_1,\Psi_2,\Psi_3,\Psi_4$ に符号を割り当てる. Ψ_1,Ψ_2,Ψ_3 には既に符号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の最上位ビットを *1* にしたものを更に割り当てる. Ψ_4 には未使用の符号 (100) を割り当てる. この場合 $h_1=x_1$ となる. *(例題終)*

定理 4.1 を一般化することにより p 個の中間変数を 1 変数関数に簡単化できる. そのための条件は以下のようになる.

定理 4.2 定理 *4.1* において,

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

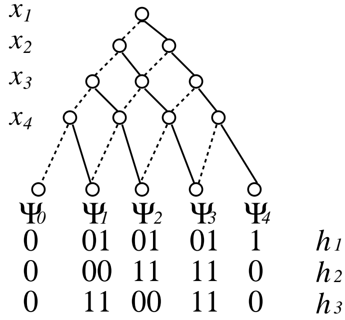

図 4.2: $h_1 = x_1$ となる符号化

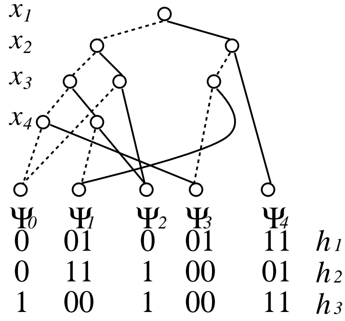

図 4.3: $h_1 = x_1, h_2 = x_3$ となる符号化

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

の 2^p 個の部分関数の異なる関数の個数がいずれも 2^{u-p} 以下であるとき f は

$$f(X_1, X_2) = g(h_{u-p}(X_1), h_{u-p+1}(X_1), \ldots, h_u(X_1), x_{j_1}, x_{j_2}, \ldots, x_{j_p}, X_2)$$

の形で分解可能である． ここで，$x_{j_r} \in \{X_1\}$ $(r = 1, 2, \ldots, p)$，$j_1 < j_2 < \cdots < j_p$，$1 \le p \le k$ である．

中間変数を 2 つ以上簡単化する場合, アルゴリズム 4.1を単純に拡張したものでは良い結果を生成しない．ここでは, 次の手法を用いる．

アルゴリズム 4.2 *(複数個の中間変数の簡単化)*

1. $\binom{k}{p}$個 $(p = 1, 2, \ldots, k-1)$，$(k$ は束縛変数の個数$)$ の場合について, 定理 *4.2*を満足する p の最大値と中間変数を簡単化する変数を求める．候補となる変数の組が複数存在するときは, 部分関数の個数の総和が最も小さい組を選択する．
2. *1* で $p \ge 2$ の場合, それぞれの Ψ_i が

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

   の 2^p 個の部分関数のうち, どの部分関数から構成されているか調べる．各 Ψ_i を構成する部分関数の個数を q_i とする．*(詳細は例 4.2参照)*
3. q_i が大きい Ψ_i から順に, 上位 p ビットが $(x_{j_1}, x_{j_2}, \ldots, x_{j_p})$ である未使用の符号を *0* から順に割り当てる．

例 4.2 図 *4.3*について考える．このとき第 *4* 段目での *BDD* の幅は *5* である．$\Psi_i(|x_1 = 0, x_3 = 0), \Psi_i(|x_1 = 0, x_3 = 1), \Psi_i(|x_1 = 1, x_3 = 0), \Psi_i(|x_1 = 1, x_3 = 1)$ が生成する異なる関数の個数は全て *2* である． $u = \lceil \log_2 5 \rceil = 3$，$2^{u-2} = 2$ より，Ψ_i は定理 *4.2*の条件を満足し，$h_1 = x_1, h_2 = x_3$ と変換できる． Ψ_0 は $\Psi_0(|x_1 = 0, x_3 = 0)$ のみから構成されているので $q_0 = 1$, Ψ_1 は $\Psi_1(|x_1 = 0, x_3 = 1)$, $\Psi_1(|x_1 = 1, x_3 = 1)$, の *2* つの部分関数から構成されているので $q_1 = 2$ である．同様に $q_2 = 1, q_3 = 2, q_4 = 2$ である．Ψ_1, Ψ_3, Ψ_4 の順に, 符号を割り当てる．Ψ_1 には上位 *2* ビットが *01* と *11* の符号 (010) と (110) を割り当てる．同様に Ψ_3 には (000) と (100) を, Ψ_4 には (101) と (111) を割り当てる．次に Ψ_0 には未使用の符号 (001) を, Ψ_2 には (011) を割り当てる．この場合 $h_1 = x_1, h_2 = x_3$ となる． *(例題終)*

5 実験結果

各 LUT の入力数 k を 15 に固定してアルゴリズム 4.1～4.2 をベンチマーク関数に適用し, カスケード回路を生成した．1 変数関数に変換できる中間変数が存在しない場合は strict encoding を行った．表 5.1に各ベンチマーク関数のカスケードの段数と LUT の個数を示す．実験に用いた計算機の性能は次の通りである．

- CPU: PentiumIII 1GHz
- 1 次キャッシュ: 32KB
- 2 次キャッシュ: 256KB
- memory: SDRAM 4GB

s はカスケードの段数を表す．*Strict* は strict encoding のみを行った場合の LUT の総数を表す．N_1 は 1 つの段において最大 1 つの中間変数を 1 変数関数に変換したときの LUT の個数を表す．同様に N_2, N_3 はそれぞれ 1 つの段において最大 2 つの中間変数を 1 変数関数に変換した場合, 3 つの中間変数を 1 変数関数に変換した場合の LUT の個数を表す．-は, N_t とするとその関数において最大 t 個変換できる中間変数が見つからなかったことを表す．本手法により, ほとんどの場合で LUT の個数を削減できた．特に C2670, C5315, および des の場合に

表 5.1: カスケード回路の LUT 数

Name	In/Out	s	*Strict*	N_1	N_2	N_3	N_4	N_5
C432	36/7	4	20	17	-	-	-	-
C499	41/32	7	60	55	54	53	52	-
C880	60/26	8	51	46	45	44	-	-
C1908	33/25	5	35	33	32	-	-	-
C2670	233/140	28	180	155	140	131	-	-
C3540	50/22	11	107	97	94	94	90	-
C5315	178/123	23	151	132	118	109	104	103
C7552	207/108	25	155	138	129	125	-	-
apex2	39/3	3	5	4	-	-	-	-
apex3	54/50	6	28	24	22	19	-	-
apex7	49/37	5	24	21	20	-	-	-
b9	41/21	4	14	11	10	9	-	-
cordic	23/2	2	4	3	-	-	-	-
des	256/245	34	241	211	185	165	155	-
duke2	22/29	3	10	9	-	-	-	-
e64	65/65	7	27	24	22	-	-	-
k2	45/45	6	30	26	24	24	23	-
misex2	25/18	3	9	8	7	-	-	-
rot	135/107	18	128	115	106	100	97	95
spla	16/46	2	8	7	6	-	-	-
t481	16/1	2	3	-	-	-	-	-
vg2	25/8	3	6	-	-	-	-	-

s : カスケードの段数

Strict : strict encoding のみを行った場合の LUT の個数

N_t : 1 つの段につき最大 t 個の中間変数を 1 変数関数に変換した場合の LUT の個数

は, LUT 数を約 30%削減できた．一般に BDD の形が細長くなるようなベンチマーク関数では, LUT の個数を削減しやすい．次にカスケード回路生成に要する計算量について述べる．アルゴリズム 4.2の 1,2 において最大の p と各部分関数を求めるために $\binom{k}{p}2^p$ 回 BDD の上部 k 段について探索を行う．次に 3 において $\sum_{i=0}^{u-1} q_i$ 個の要素に符号を割り当てる作業を行う．そして BDD を再構成する．この演算をカスケードの段数だけ行う．表 5.1のどの関数も数秒以内でカスケード回路を生成することができた．

6 結論

本論文では, 論理関数を LUT カスケードで実現する手法を紹介した．次に中間変数を簡単化することにより LUT の個数を削減する手法を示した．本手法は回路構造を大きく変化させずに LUT の個数を削減することができる．また本論文では, 各 LUT の入力数 k を固定した場合のみを考えている．しかし一般には k は各段で異なってもよい．これを考慮することにより, 更に少ないメモリで大規模な関数を実現することができる．

謝辞

本研究は一部, 日本学術振興会, 科学研究費補助金による．明治大学の井口幸洋助教授には, 熱心にご討論頂いた．

参考文献

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

[2] G. De Micheli, *Synthesis and Optimization of Digital Circuits*, McGraw-Hill, 1994.

[3] Y. Iguchi, T. Sasao, and M. Matsuura, “ Realization of multiple-output functions by reconfigurable cascades,” *International Conference on Computer Design (ICCD-2001)*, Austin, Texas, Sept. 23-26, 2001, pp. 388-393.

[4] J.-H. R. Jian, J.-Y. Jou, and J.-D. Huang, “Compatible class encoding in hyper-function decomposition for FPGA synthesis,” *Design Automation Conference*, pp. 712-717, June 1998.

[5] C. Legl, B. Wurth, and K. Eckl “Computing support-minimal subfunctions during functional decomposition,” *IEEE Trans. VLSI*, Vol. 6, No. 3, pp. 354-363, Sept. 1998.

[6] 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.

[7] 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.

[8] T. Sasao, M. Matsuura, Y. Iguchi, and S. Nagayama, ”Compact BDD representations for multiple-output functions and their applications to embedded system, ” *IFIP VLSI-SOC 2001,* Dec. 3-5, 2001 (accepted).

[9] T. Sasao, ”Compact SOP representations for multiple-output functions: an encoding method using multiple-valued logic,” *31th International Symposium on Multiple-Valued Logic,* Warsaw, Poland, May 22 - 24, 2001, pp.207-212.

[10] T. Sasao, “FPGA design by generalized functional decomposition,” in *Logic Synthesis and Optimization* (T. Sasao, ed.), pp.233-258, Kluwer Academic Publishers, 1993.

[11] 井口，笹尾，松浦，伊勢野，”決定グラフに基づく論理関数の評価システム”, 電子情報通信学会論文誌 D-I, Vol. J84-D-I, No.6, pp.523-530, 2001-06.

[12] 笹尾勤，井口幸洋, 松浦宗寛, 永山忍 ”多出力関数のカスケード実現と再構成可能ハードウエアによる実現 ”, 電子情報通信学会 FTS 研究会, FTS2001-8, pp. 57-64, 三重大学 (2001-04).