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

2003－ＳＬＤＭ－112　(43)
2003／11／28

順序回路方式LUTカスケードにおけるメモリパッキングについて

草野　将樹†　　笹尾　　勤†　　松浦　宗寛†　　井口　幸洋††

† 九州工業大学情報工学部電子情報工学科
〒820–8502 福岡県飯塚市大字川津 680–4
†† 明治大学理工学部情報科学科
〒214–8571 神奈川県川崎市多摩区東三田 1–1–1
E-mail: †kusano@aries02.cse.kyutech.ac.jp, ††{sasao,matsuura}@cse.kyutech.ac.jp, †††iguchi@cs.meiji.ac.jp

あらまし　順序回路方式LUTカスケードで，各セルの入力数を可変にした場合，各セルのデータを最小のメモリに格納する問題は，ビンパッキング問題と類似の問題となる．本稿では，この問題をダイナミック・プログラミングを用いて解く．本手法を用いて，LUTカスケードの段数を増加させることなく，メモリ量をもとの大きさの40%まで削減できた．
キーワード　BDD, LUTカスケード，メモリ・パッキング

Memory Packing Method in Sequential Look-Up Table Cascades

Masaki KUSANO†, Tsutomu SASAO†, Munehiro MATSUURA†, and Yukihiro IGUCHI††

† Department of Computer Science and Electronics,
Kyushu Institute of Technology
680–4 Kawazu, Iizuka, Fukuoka, 820–8502, Japan
†† Department of Computer Science,
Meiji University
1–1–1, Higashi-mita, Tama-ku, Kawasaki, Kanagawa, 214–8571, Japan
E-mail: †kusano@aries02.cse.kyutech.ac.jp, ††{sasao,matsuura}@cse.kyutech.ac.jp, †††iguchi@cs.meiji.ac.jp

Abstract　A sequential look-up table (LUT) cascade consists of memory and a control circuit, and simulates a combinational LUT cascade. In a sequential cascade, we assume that the number of inputs of each cell can be different. In this case, we can minimize the number of levels and total amount of memory for cascade by using dynamic programming. By using this technique, we could reduce the amount of memory into 40% of original size.
Key words　BDD, LUT cascade, memory packing

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．回路を大域的に見た際，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．また，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．

本研究は，速度，面積，再プログラム可能性の点で有利な規則的論理回路を検討し，その論理合成手法を開発することを目的とする．

規則的回路構造を採用すると

（1）　使用環境や，製造ばらつきに対して安定．

（2）　拡張が容易．

（3）　再プログラムが容易．

（4）　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．などの特長を生かせる．

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

表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

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，遅延時間の予測困難，クロストーク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．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）を提案する[7]．

LUTカスケード法では，

（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，専用の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．

予備実験[6]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，約10倍高速という結果が出ている．これらを纏めると，下表のようになる．

応用分野としては，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り，低消費電力が要求されるものを考えている．

LUTカスケードでは

（1）　回路構造が規則的なので，配線部の実現が容易．

（2）　メモリを時分割で，LUTとして使用．

（3）　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．

（4）　「メモリ・パッキング」という手法で，メモリに，多数のLUTを詰め込む．

（5）　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．

多段論理回路をROM／RAMで実現するというアイデアは，1995年に，Murgai-Hirose-Fujita [3] が発表している．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. 定　　義

本章では，本稿で用いるいくつかの用語について述べる．

2.1 特性関数

入力変数を $X=(x_1,x_2,...,x_n)$．　多出力関数を $F=(f_1(X),f_2(X),...,f_m(X))$ とする．多出力関数の特性関数を

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

とする．ここで y_i は出力を表す変数である．

n 入力 m 出力関数の特性関数は，$(n+m)$ 変数の二値論理関数であり，入力変数 $x_i(i=1,2,...,n)$ の他に，各出力 f_i に対して出力変数 y_i を用いる．$B=\{0,1\}$ とする．$\vec{a}\in B^n$ かつ $F(\vec{a})=(f_0(\vec{a}),f_1(\vec{a}),...,f_{m-1}(\vec{a}))\in B^m$ とする．いま，$\vec{b}\in B^m$ とすると，

$$\chi(\vec{a},\vec{b}) = 1 \quad (\vec{b}=F(\vec{a}))$$
$$= 0 \quad (\text{上以外の時})$$

となる．

多出力関数 $F=(f_1,f_2,...,f_m)$ の BDD for CF(Binary Decision Diagram for Characteristic Function) とは，F の特性関数 χ を表現する BDD である．ただし，BDD の変数は，根節点を最上位としたとき，変数 y_i は，f_i が依存する変数の下に置く．

2.2 分　　割

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**という．X の変数の個数を $|X|$ で表す．

論理関数 $f(X)$，X の分割 (X_1,X_2) に対して，$2^{|X_1|}$ 列 $2^{|X_2|}$ 行の表で，各列，各行に二進符号のラベルをもち，その要素が f の対応する真理値である表を**分解表**といい，分解表の異なる列パターン数を**列複雑度** μ という．図1に4変数関数の分解表の例を示す．

$X_2=(x_3,x_4)$ \ $X_1=(x_1,x_2)$	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 1　4 変数関数の分解表

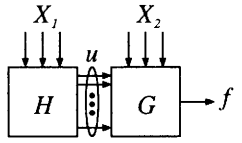

図 2　関数分解による関数 f の実現

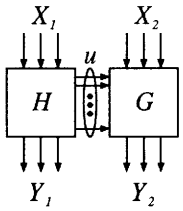

図 3　中間出力のある関数分解

2.3 関数分解

X の分割 (X_1,X_2) において, 論理関数 $f(X)$ の分解表の列複雑度が μ のとき, 関数 $f(X)$ 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 2 の回路構造で実現可能である. このとき, $u=\lceil\log_2\mu\rceil$ である. この操作を $s-1$ 回繰り返すと, 図 4 に示すような組み合せ回路方式 LUT カスケードが構成できる. ここで, 第 i 段目のセルの入力数を k_i とすると, $k_i=|X_i|+u_{i-1}$ である. また, X_1,X_2 をそれぞれ, Bound set, Free set という. BDD で論理関数を表現をする場合, Bound set から Free set への直接の後続点の個数が列複雑度になる.

ここで, BDD for CF の場合の関数分解を考える. (X_1,X_2) を入力変数の集合, (Y_1,Y_2) を出力変数の集合とする. BDD for CF の変数順序を (X_1,Y_1,X_2,Y_2) とするとき, BDD for CF の (X_1,Y_1) における列複雑度を μ とする. ここで, μ を数える際, 出力を表現する変数から, 定数 0 に向かう枝は無視する. 多出力関数を図 3 の回路で実現する場合, 二つの回路 H と G の間の必要かつ十分な接続線数は $\lceil\log_2\mu\rceil$ である. 回路 G の入力となる回路 H の出力を中間変数という.

3. 順序回路方式 LUT カスケード

組み合せ回路方式 LUT カスケードとは, 図 4 に示すように複数の LUT から成るセルを直列に接続して, 論理回路を実現したものである. **順序回路方式 LUT カスケード・アーキテクチャ** を図 5 に示す. これは, 組み合せ回路方式 LUT カスケードを模擬する. ここで, 使用する LUT をまとめて一つの大きなメモリに格納し, これに外部入力の値を保持するレジスタ, メモリ番地を保持する MAR(Memory Address Register), 出力値を保持する MBR(Memory Buffer Register), 外部出力を保持するレジスタとシーケンサを備えている. 本方式では, メモリ出力を MAR にフィードバックさせたり, 必要な信号値を所定のレジスタに代入する操作などの制御はシーケンサが担当する. カスケードのうち LUT 部分は Memory for Logic に, 配線部分はシーケンサのメモリ (Memory for connection) に格納する.

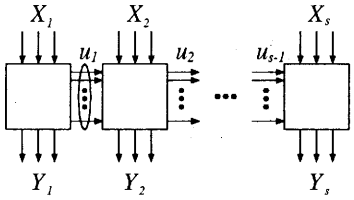

図 4　組み合せ回路方式 LUT カスケード

k 入力 LUT のメモリ量は, 2^kbit である. LUT カスケードでは一般に多出力のセルを用いており, i 番目のセルの入力変数の個数を k_i, 中間変数の個数を u_i, 外部出力変数の個数を $|Y_i|$ で表す. カスケードの段数を s とすると, 与えられたある変数分割 $(X_1,Y_1,X_2,Y_2,\ldots,X_s,Y_s)$ において, LUT カスケードを実現するために必要なメモリ量は

$$L(X_1,Y_1,X_2,Y_2,\ldots,X_s,Y_s)=\sum_{i=1}^{s}2^{k_i}\cdot(u_{i-1}+|Y_i|) \qquad (2)$$

となる. これより, 順序回路方式 LUT カスケードの設計問題を次のように定式化可能である.

問題 3.1 : 多出力関数 $F=(f_1(X),f_2(X),\ldots,f_m(X))$ を表現する F の特性関数 χ が与えられたとき, メモリ量が L_0 以下で, カスケードの段数が最小, かつ, メモリ量が最小となる分割 $(X_1,Y_1,X_2,Y_2,\ldots,X_s,Y_s)$ を求めよ. ただし, F の特性関数 χ を表す BDD for CF の変数順序は与えられているものとする.

各セルの内容を格納しているアドレスの開始番地は, 下位ビットがすべて 0 である必要がある. 例えば, 10 入力の LUT を使用する場合, 開始番地は xxxxx0000000000 の形をしている. このため, 実際に必要となるメモリ量は, 式 (2) で求めた値よりも大きくなる. ただし, 後述するメモリ・パッキングを行うことで, 必要なメモリ量を削減できる.

3.1 セルの入力数が可変な場合のメモリ量の下界探索アルゴリズム

順序回路方式 LUT カスケードにおいて, 各セルの入力数 k が等しいとき, 各 LUT のメモリ量は等しい. 各セルの入力数は固定で, 前段のセルからの中間変数と入力変数の個数の和が k となるように分割される. ただし, 最終段のセルの入力数のみ k 以下である.

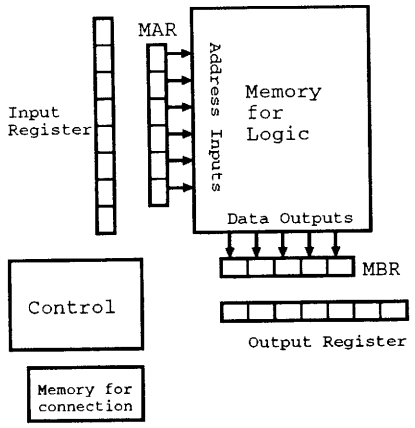

図 5　順序回路方式 LUT カスケード・アーキテクチャ

しかし, セルの入力数をセル毎に変えることも可能である. 例えば, 最終段の一つ前の段のセルの入力数を 1 つ減らし, 最終段のセルの入力数を 1 つ増やすと, メモリ量が変化する. BDD では, i 番目の変数 x_i の列複雑度 μ_{x_i} と i+1 番目の変数 x_{i+1} の列複雑度 $\mu_{x_{i+1}}$ の関係は, $\mu_{x_i}/2 \leqq \mu_{x_{i+1}} \leqq 2 \cdot \mu_{x_i}$ となる. したがって, 中間変数の個数 u_{x_i} と $u_{x_{i+1}}$ の関係は, $u_{x_i} - 1 \leqq u_{x_{i+1}} \leqq u_{x_i} + 1$ となる.

ここで, 変数 X の分割を $(X_1, X_2, ..., X_s)$ とし, i 段目のセルの入力数を k_i, 出力数を u_i, X_i を $X_i = (x_j, x_{j+1}, ..., x_l)$ とする. $i+1$ 段目のセルの入力数 k_{i+1} と X_{i+1} の関係は,

$$k_{i+1} = |X_{i+1}| + u_i \tag{3}$$

となる. ここで, X_i の代わりに X_i から $\{x_l\}$ を取り除いた $X_{i'} = (x_j, x_{j+1}, ..., x_{l-1})$ を用いる場合について考える. 出力数を $u_{i'}$ とすると, $(i+1)$ 段目のセルの入力数 k_{i+1} と $X_{i'+1}$ の関係は,

$$k_{i+1} = |X_{i'+1}| + u_{i'} \tag{4}$$

となる. もし, $u_i = u_{i'} + 1$ ならば, 式 (3.2) と式 (3.3) から $|X_{i'+1}| = |X_{i+1}| + 1$ となる. $X_{i'} = X_i - \{x_l\}$ なので, $X_{i'+1} = X_{i+1} \cup \{x_l\}$ となる. 仮定より, $(i+1)$ 段目のセルの入力数は, いずれの分割の場合も同じであるため, メモリ量は等しい. 一方, i 段目のセルは, もとの分割ではメモリ量は, $2^{k_i} \cdot u_{i-1}$ ビット, 後の分割では, $2^{k_i - 1} \cdot (u_{i-1} - 1)$ となる. よって, X_i を用いた場合よりも, $X_{i'}$ を用いた場合のほうがメモリ量は小さくなる.

以上のことを考慮して, メモリ量や段数が最小となる分割をダイナミック・プログラミングを用いて求める. ダイナミック・プログラミングとは, 問題を段階的に解く手法で, ある段階 (部分問題) で得られた最適解をもとに, 次の段階 (より大きな部分問題) の最適解を求めるという操作を繰り返し行なう.

以下に, 問題 3.1 を解くためのアルゴリズムを示す.

3.1.1　アルゴリズム 1 (メモリ量の下界の探索)

n 入力論理関数の BDD が与えられているとする. BDD の高

```
1   dyna_cascade(BDD,n,k) {
2     for(i = 1; i <= n; i++){
3       μ_{x_i} を計算する.
4       u_{x_i} = ⌈log2 μ_{x_i}⌉(中間変数の個数の計算)
5     }
6     for(i = 1; i < n; i++){
7       for(j = i + 1; j <= n; j++){
8         p = ((x_i, x_{i+1}, .., x_j) の入力変数の個数)
9         if(p + u_{x_{i-1}} <= k){
10          l(j), s(j) を計算する.
11          if(s(j) < S(j)){ (段数の判定)
12            S(j) = s(j);
13            L(j) = l(j);
14          }
15          else if(s(j) == S(j)&&l(j) < L(j)){(メモリ量の判定)
16            L(j) = l(j);
17          }
18        }
19      }
20    }
21  return L(n), S(n)
22  }
```

図 6　アルゴリズム 1(メモリ量の下界の探索) の疑似コード

さ i の変数を x_i として, 列複雑度を μ_{x_i}, 中間変数の個数を u_{x_i} とする. 変数 x_i まで LUT カスケードを作成した場合のメモリ量を $l(x_i)$, 段数を $s(x_i)$ とし, その最適解をそれぞれ, $L(x_i)$, $S(x_i)$ とする. セルの入力数の上限を k とする. 図 6 に本アルゴリズムの疑似コードを示す.

6-7 行目で, 分割の生成をする. 8 行目は, 6-7 行目で選んだ分割の入力変数の個数を数えている. 9 行目は 8 行目で調べた入力変数の個数がセルの入力数に収まるかの判定をする. 11 行目以降は, 最適な分割かどうかの判定を行なう. このアルゴリズムは, LUT カスケードを 1 段目から順に 2 段目, 3 段目と作成していき, 最終的に段数最小, かつ, メモリ量最小な分割を見つける.

4.　メモリ・パッキング

順序回路方式 LUT カスケードでは, すべての LUT 情報を一つのメモリに格納する. この際, メモリ・パッキングを行なうことによりメモリ量を削減できる.

例題 4.1 11 入力 3 出力関数 $F(f_0(X), f_1(X), f_2(X))$ を実現する 5 入力 LUT カスケードを図 7 に, また, LUT データのメモリへのマッピングを図 8 に示す. メモリのアドレスは, 6 ビット, 1 ワードが 4 ビットとする. 図 8 の $(A_0, A_1, .., A_5)$ はデータアドレスを示し, 上位 2 ビット (A_0, A_1) はメモリのページ番号を示す. D_0, D_1, D_2, D_3 はメモリの出力線を表す. ハッチング部分は未使用部分を表す. この実現では, 一つのページ内に一つの LUT のデータのみを格納している. そのため, 図 8 では, メモリの半分が未使用領域となっている. page3 の LUT データを page1 の D_0

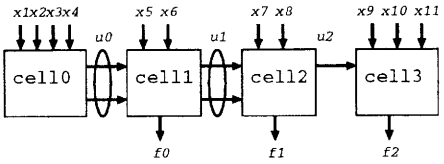

図 7 LUT カスケードの例

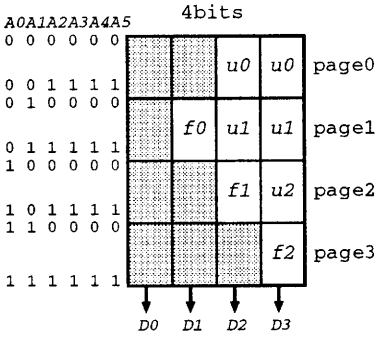

図 8 LUT データのメモリへのマッピング

部分に, page2 の LUT データを page0 の D_0,D_1 部分に埋め込むことにより, 必要なメモリ量を節約できる. (例終り)

一般に, カスケードの同一段の LUT データは, 同時に読み出す必要があり, メモリの同一ページ内に格納する必要がある. しかし, 空きがあれば, 例題 4.1 のように同一ページ内に異なる段のセルを格納しても良い. これを**メモリ・パッキング**と呼ぶ. メモリ・パッキングを行なうためには, MBR と MAR の間にビットシフトを行なうシフタが必要である. 3.1.1 のアルゴリズムを適用して分割を求めると, 各セルの入力数は一般に異なる. セルの入力数が一つ減ると, データアドレス線を一本減らして実現できる. 図 9 のように, 入力数が異なるセルを同一ページ内に格納する場合, MAR に定数を供給する回路が必要となる.

以下にメモリ・パッキングにおいて, メモリ量を削減するための発見的アルゴリズムを示す.

4.1 アルゴリズム 2 (メモリ・パッキング)

カスケードの第 i 段目 (i = 1,2,...,s) のセルの出力数を w_i とする. $w = max_i\{w_i\}$ ビット同時に読みだせるメモリが与えられているものとする.

(1) 各セルを入力数の大きい順に並び替える. さらに, 入力数が同じセルでは出力数の大きい順に並び替え, その出力数を $v_1, v_2, ..., vs$ とする.

(2) $i \leftarrow 0$.

(3) $i \leftarrow i + 1$, $j \leftarrow 0$ とする.

(4) セル i の v_i 個の出力を第 j ページに割り当て可能か否かを調べる. 可能な場合, 割り当てを実行し, 3 に戻る. 不可能な場合, $j \leftarrow j + 1$ とし, 4 に戻る.

(5) i の値を s 回を変更したら, 停止する.

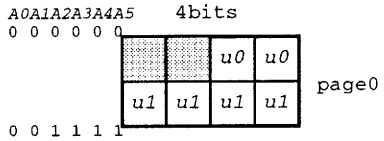

図 9 可変時のパッキングの例

5. 実験結果

アルゴリズム 1 を C 言語で実装し, MCNC ベンチマーク関数に適用した. 各セルの入力数を固定として分割を求めた場合 (Case.1) と, アルゴリズム 1 を用いて各セルの入力数を可変にして分割を求めた場合 (Case.2) と, セルの入力数を可変にして段数を最小にした場合 (Case.3) で, 1Mbit(64k ワード× 16 ビット) のメモリに収まるように回路を作成した. また, それぞれの場合について, アルゴリズム 2 を用いてメモリ・パッキングをした場合と, しない場合を比較した結果を表 2 に示す. 表中の Name は関数名, In は入力数, Out は出力数, k は LUT の入力数の最大値, s はカスケードの段数, Memory はメモリ量で単位は Mbit を表す. また, unpack はメモリ・パッキングを行なわない場合を, pack はメモリ・パッキングを行なった場合を示す. 実行環境は, CPU:Pentium4 Xeon 2.8GHz, L1 Cache:32KB, L2 Cache:512KB, Memory:4GB, OS:redhad(Linux 7.3) 上で gcc を用いてコンパイルした. LUT カスケードは, 各セルの入力数 k が大きくなると, セル一つ当りのメモリ量が増える. セルの入力数が増えると回路全体の LUT 数は減るが, 全体のメモリ量は増える. Case.1 の k の値は, 全体が 1Mbit に収まり, 最小段数で回路を実現できる値を用いた. Case.2 では, Case.1 の k を用いて, メモリ量と段数が最小になる分割を求めた. 表 2 に示すように, ほとんどの関数で Case.1 よりも Case.2 のほうが小さなメモリ量で回路を実現できた.

ベンチマーク関数 C432 を $k = 15$ で本手法を用いて実現した際の回路を図 10 に示す. 段数は 4 段で, メモリ・パッキング後のメモリ量は 0.75Mbit, 1〜3 段目のセルの入力数が 14 で, 4 段目のセルの入力数が 15 である. セルの入力数を可変にすると固定にした場合よりも, より小さなメモリ量で LUT カスケードを実現できた.

次に, 1Mbit(64k ワード× 16 ビット) を上限として, セルの入力数を可変にして段数を最小にした Case.3 と, Case.1 を比較する. 入力数を可変にすると, 入力数を固定にした場合よりも小さなメモリ量で回路を実現することができる. 1Mbit を上限としたとき, いくつかのベンチマーク関数では Case.1 の k より大きな値を用いることができ, 段数の削減ができた.

ベンチマーク関数 misex2 を $k = 14, 16$ とし, 入力数を可変で実現したときの, 回路を図 11 に示す. $k = 14$ のときは 4 段で, $k = 16$ のときは 3 段で回路を実現できた. 一方, 入力数を固定にした場合には, $k = 16$ の場合にも 1Mbit 以下のメモリで実現不可能であった.

表 2　ベンチマーク関数を表現するためのメモリ量 (k 固定と可変の場合の比較)

Name	In	Out	Case.1 入力数固定				Case.2 入力数可変				Case.3 段数最小			
					Memory				Memory				Memory	
			k	s	unpack	pack	k	s	unpack	pack	k	s	unpack	pack
C432	36	7	15	4	2.000	1.000	15	4	1.250	0.750	15	4	1.250	0.750
apex1	45	45	13	10	1.250	1.000	13	10	0.721	0.596	13	10	0.721	0.596
apex2	39	3	15	3	1.000	0.500	15	3	1.000	0.500	15	3	1.000	0.500
apex3	54	50	12	16	0.938	0.625	12	16	0.666	0.416	13	13	1.135	0.760
comp	32	3	12	3	0.051	0.016	12	3	0.051	0.016	12	3	0.051	0.016
duke2	22	29	14	3	1.000	0.750	14	3	0.376	0.376	14	3	0.376	0.376
e64	65	65	13	6	0.750	0.750	13	6	0.313	0.313	13	6	0.313	0.313
k2	45	45	13	10	1.125	1.000	13	10	0.721	0.596	13	10	0.721	0.596
misex2	25	18	14	3	0.750	0.500	14	3	0.078	0.068	16	2	0.750	0.750
seq	41	35	13	8	1.000	0.625	13	8	0.626	0.376	15	7	1.113	0.751
vg2	25	8	13	3	0.375	0.250	13	3	0.250	0.125	16	2	1.500	1.000
x6dn	39	5	13	5	0.625	0.250	13	5	0.375	0.188	16	4	1.125	0.625
ratio					1	0.669			0.592	0.398			0.926	0.647

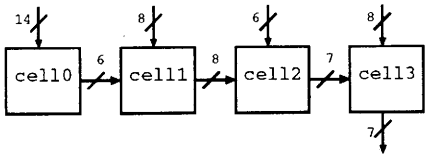

図 10　C432 の LUT カスケード回路

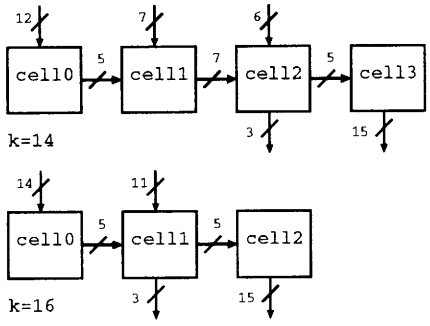

図 11　misex2 の LUT カスケード回路

6. まとめと今後の課題

本論文は, 順序回路方式 LUT カスケードの段数とメモリ量の最適構成手法, および, LUT の入力数を可変にした場合の, メモリ・パッキング法を考案した. 本手法は, LUT の入力数を一定にした場合に比べ, メモリ・パッキング後のメモリ量を平均で約 40%削減ができた. 本手法を用いて順序回路方式 LUT カスケードを作成する場合, シーケンサ部分に, セルの接続情報を記憶するためのメモリが必要になる. このメモリには, セルを参照するためのページ番号や中間出力を記憶する. ただし, このメモリ量は, 図 5 の Memory for Logic に比べて非常に小さいので無視できる. したがって, 本手法はメモリ・段数の削減について, 有効な手法であるといえる. 本手法は BDD の幅から分割を求めているため, BDD の変数順序により分割が変わる. 本手法で最適となるような変数順序が求まれば, よりメモリ量, 段数を削減できる. 今後の課題として, 本手法に適した変数順序最適化アルゴリズムの開発が必要である.

7. 謝　　辞

本研究は一部, 日本学術振興会・科学研究費, 文部科学省・北九州地域・知的クラスタ創成事業補助金, および, 武田計測先端知財団補助金による.

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, 2001.*

[2] 井口幸洋, 笹尾勤, "LUT カスケード・アーキテクチャについて," 電子情報通信学会, リコンフィギャラブルシステム研究会, *2003* 年 *9* 月 *18–19* 日.

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

[4] Qin Hui, 笹尾勤, 松浦宗寛, 永山忍, 中村和之, 井口幸洋 "順序回路方式 LUT カスケードについて," 電子情報通信学会, 第 *2* 種研究会第 *7* 回システム *LSI* ワークショップ, *2003* 年 *11* 月 *25–27* 日 *(*発表予定*)*

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

[6] 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).*

[7] T. Sasao, M. Matsuura, and Y. Iguchi, "A cascade realization of multiple-output function for reconfigurable hardware," *International Workshop on Logic Synthesis (IWLS-2001), Lake Tahoe, CA, JUNE 12-15,2001, pp.225-300.*

[8] 笹尾勤, 松浦宗寛, "BDD を用いた多出力論理関数分解の一手法について," 電子情報通信学会, *VLSI* 設計技術研究会, *2003* 年 *11* 月 *(*発表予定*)*