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

2004－SLDM－117（37）
2004／12／2

LUTリングを用いた順序回路の合成アルゴリズムについて

中原　啓貴†　笹尾　勤††　松浦　宗寛††

† 九州工業大学大学院 情報創成工学専攻
†† 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　順序回路をLUTリング上に実現するための手法を示す. 本手法は次の二つのステップから成る: まず, 与えられた多出力論理関数の出力を分割する. 次に, 分割した論理関数の各部分集合をカスケード実現し, LUTリングのメモリ上に格納する. また, 本論文では, メモリを用いた他の順序回路の実現法との比較を行い, 本手法が有効であることを示す. 本手法ではメモリ量制限を与えるだけで自動でLUTリングを合成することができ, 従来手法と比較して, 高速に合成できるという特徴がある.

キーワード　BDD_for_CF, 関数分解, 再構成可能アーキテクチャ, LUTカスケード

A Design Algorithm for Sequential Circuit Synthesis using LUT Ring

Hiroki NAKAHARA†, Tsutomu SASAO††, and Munehiro MATSUURA††

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

Abstract　This paper shows a design method for a sequential circuit by using a Look-Up Table(LUT) ring. The method consists of two steps: The first step partitions the outputs into groups. The second step realizes them by LUT cascades, and stores the content of cells in the cascades into the memory. We also compare the method with other methods to realize sequential circuits. With the presented algorithm, we can easily design sequential circuits satisfying given specifications.

Key words　BDD_for_CF, Functional decomposition, Reconfigurable architecture, LUT cascade

1. はじめに

LSIの集積度の向上に伴い設計コストの増大が問題になっている. 特に, 配線遅延の見積りが困難, 電磁誘導現象, クロストークノイズ, といったDSM問題が生じている. また, 製品寿命もますます短くなっている. これらの問題を解決する1つの方法は, 規則的構造を持ち再構成可能なアーキテクチャを採用することである. 規則的構造アーキテクチャは, 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問題が回避できる[2]. また, 再構成可能なアーキテクチャは再プログラム化が可能なため, 一つの回路が種々の応用に利用でき, 設計コストを下げることができる.

規則的構造を持ち再構成可能なアーキテクチャとしてメモリが挙げられる. 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.

メモリなどの規則的構造を用いて順序回路を実現する方法は様々あるが, 最も単純なものとして, メモリやPLA(Programmable Logic Array)とフリップ・フロップを組み合わせたものが知られている. メモリやPLAを用いて順序回路を率直に実現する場合, 入力変数の個数 n が大きい時, 必要なハードウェア量が大きくなり過ぎる. 文献[3]〜[5]はメモリと付加的なハードウェアを用いて順序回路を実現する方法に関して述べている. 特に, [5]はマルチプレクサを用いて, 必要メモリ量を減らす方法について述べている. 順序回路では状態遷移は一部の変数に依存することが多い. よって, マルチプレクサを用いて依存変数のみを選択しメモリの入力にすることにより, メモリの入力数を減らすことができ, 必要なハードウェア量を削減できる. しかしながら, 状態変数の個数が多く, 状態遷移の依存変数が多い場合にはメモリの入力数をほとんど削減できず, この手法は役に立たない.

出力値の評価に時間がかかってもよい場合には, 汎用マイクロプロセッサ上の論理シミュレータとメモリを用いて順序回路を模擬できる. 論理回路が与えられた場合, 各ゲートにプログラムコードを割り当てメモリに格納する. このプログラムコードを汎用マイクロプロセッサで評価する. 評価時間はゲート数に比例する. しかしながら, 汎用プロセッサによる実現は, 専用回路に比べ2桁から3桁遅くなり, 消費電力も大きい.

本論文では, LUT リング [6]〜[10] を用いて順序回路を実現する方法を提案する. LUT リングは制御部とメモリとレジスタを持ち, 書き換え可能な接続回路でメモリと各レジスタを接続し, メモリに格納した論理回路を評価する. 本論文では以下の手順で順序回路を実現する.

1. 組合せ回路部を表現する論理関数を BDD(Binary Decision Diagram) で表現する.
2. 表現した BDD を LUT カスケードに変換する.
3. 変換した LUT カスケードを LUT リングのメモリに格納し, 順序回路を模擬する.

本手法は汎用マイクロプロセッサを用いる手法より高速であり, 必要メモリ量も比較的少ない. LUT リングの設計に関しては [6], [8]〜[10] で述べられているが, 従来の設計法では, 設計者が多くのパラメータを人手で与えなければならない. また, LUT リング上のメモリを効率良く使用するには, パラメータを変化させて何度も合成する必要がある. 本論文で述べる設計法は, 人手で与えるパラメータは総メモリ量, セルの出力数にとどめ, 他の最適化はすべて自動で行う. また, LUT リング上の利用可能なメモリを最大限に利用し高速な順序回路を実現する.

また, 本論文では, 順序回路をメモリを用いて実現する他の手法と比較を行い, 本手法の有効性を示す. 本手法は, [6] の手法を改良したものである.

2. 論理関数のカスケード実現

多出力関数を表現する方法として, MTBDD(multi-terminal BDD), BDD_for_ECFN(BDD for encoded characteristic function for non-zero outputs), ならびに BDD_for_CF(BDD for characteristic function) が提案されている. 文献 [7] は各手法で必要なメモリ量と評価時間を比較している. これらの手法は論理関数を表現し LUT カスケードを合成する際, 以下の特徴を有する.

- MTBDD では BDD の幅が大きくなり, カスケードが合成不可能な場合が多い
- BDD_for_ECFN では出力数が大きい場合, 評価時間が大きい
- BDD_for_CF は MTBDD ほど BDD の幅が大きくなく, 同時に多数の論理関数を評価できる

よって本研究では, 多出力関数を BDD_for_CF で表現する.

2.1 BDD_for_CF を用いた関数分解

[定義 2.1] 入力変数を $X = (x_1, x_2, \ldots, x_n)$ とし, 多出力関数を $\vec{f} = (f_1(X), f_2(X), \ldots, f_m(X))$ とする. **多出力関数の特性関数**を $\chi(X,Y) = \bigwedge_{i=1}^{m}(y_i \equiv f_i(X))$ とする.

n 入力 m 出力関数の特性関数は, $(n+m)$ 変数の 2 値論理関数であり, 入力変数 $x_i (i = 1, 2, \ldots, n)$ の他に, 各出力 f_i に対して出力変数 y_i を用いる. $\vec{a} \in B^n$ かつ $F = (f_1(\vec{a}), f_2(\vec{a}), \ldots, f_m(\vec{a})) \in B^m$, $B = \{0, 1\}$ とする. ここで, $\vec{b} \in B^m$ のとき,

$$\chi(\vec{a}, \vec{b}) = \begin{cases} 1 & (\vec{b} = F(\vec{a})) \\ 0 & (otherwise) \end{cases}$$

とする.

[定義 2.2] 関数 f が変数 x に依存するとき, x を f の**サポート変数**という.

[定義 2.3] 多出力関数 $\vec{f} = (f_1, f_2, \ldots, f_m)$ の **BDD_for_CF** とは, $\vec{f}$ 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 のサポート変数の下に置く.

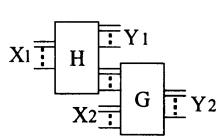


図 1 中間出力のある関数分解

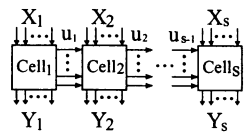


図 2 LUT カスケード

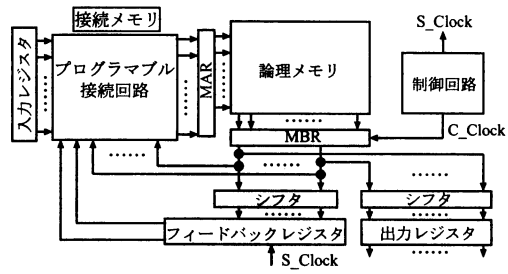


図 3 順序回路用 LUT リング

[定義 2.4] 多出力関数 $\vec{f}$ を表現する BDD_for_CF 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_for_CF 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. また, 幅を計数する際, 出力を表現する変数から, 定数 0 に向かう枝は無視する.

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. 多出力関数 $\vec{f} = (f_1, f_2, \ldots, f_m)$ を表現する 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) における幅を W とする. 多出力関数 $\vec{f}$ を**関数分解**することにより図 1 の回路構造で実現可能である. ここで, 2 つの回路 H と G の間の接続線数は $t = \lceil \log_2 W \rceil$ である [9].

[定理 2.1] [7] n 変数多出力関数 $\vec{f}$ を実現する BDD_for_CF の幅の最大値を μ_{max} とする. $u = \lceil \log_2 \mu_{max} \rceil \leqq k - 1$ ならば, $\vec{f}$ は図 2 で示すような k 入力-LUT のカスケード回路で実現可能である. 図 2 は関数分解を $s - 1$ 回繰り返して得られた回路である.

3. LUT カスケードと LUT リング

3.1 LUT カスケード

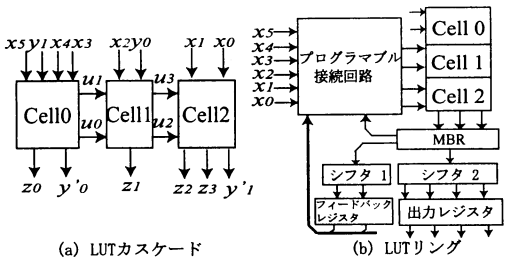
図 2 に **LUT カスケード**の概念図を示す. LUT カスケードは複数の多出力 LUT(セル) を直列に接続した構造を持つ. 各セル間はレールと呼ばれる信号線で接続される. セル間の配線は隣同士のセル間のみに限定するため, 配線遅延が小さい. また, 図 (1) の回路構造を実現するため, 各セルは外部入力の他にレール出力とは独立して, 外部出力を持つ. 本論文では, $|X_i|$ をセル i の外部入力数, u_i をセル i のレール出力数, $|Y_i|$ をセル i の外部出力数, s を LUT カスケードの段数, r を LUT カスケードの本数とする.

3.2 順序回路用 LUT リング [10]

LUT カスケードは前述のように優れた性質をもつ論理素子ではあるが, 入力数, セル数, レール数などのパラメータを一旦決めてしまうと, 限られた組み合わせ論理関数しか実現できない.

本論文では 1 つのアーキテクチャで広範囲の機能を実現可能な順序回路用 LUT リング (図 3) を提案する.

順序回路用 LUT リングは, 組合せ回路を実現する LUT カス



(a) LUTカスケード　　(b) LUTリング

図4　LUT カスケード (a) を実現する LUT リング

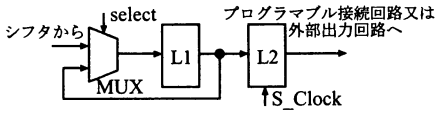


図5　ダブルランク・フリップ・フロップ

ケードの模擬を行い, 状態出力をフィードバックし, 外部出力値を出力することによって順序回路を模擬する. 利用できるメモリ量に余裕がある場合, 複数の LUT カスケードを格納し模擬することによって, さまざまな関数を実現できる. また, 利用できるメモリ量に応じて性能 (セル数) を調節できる.

順序回路用 LUT リングは, LUT カスケードの論理データを格納する**論理メモリ**と呼ばれる 1 つの大きなメモリ, 外部入力の値を保持する**入力レジスタ**, メモリ番地を保持する **MAR**(Memory Address Register), メモリの出力値を保持する **MBR**(Memory Buffer Register), **プログラマブル接続回路**, **接続メモリ**, データのシフトを行う**シフタ**, 状態変数を記憶する**フィードバック・レジスタ**, 外部出力を記憶する**出力レジスタ**, **制御回路**から構成されている. 図 4(b) は図 4(a) に示されたセル数が 3 の LUT カスケードを格納した例である.

本方式では, 論理メモリの出力値を MAR にフィードバックさせたり, 必要な信号を所定のレジスタに代入する操作の制御は制御回路が担当する. 配線部分の情報は専用のメモリ (接続メモリ) に格納する. 各カスケードを評価する際, 外部出力となる出力は出力レジスタに, 状態出力となる出力はフィードバック・レジスタに格納する. 全カスケードの評価が終了すれば, フィードバック・レジスタの値を接続回路 (プログラマブル接続回路) にフィードバックし, 出力レジスタの値を外部出力とする.

図 5 にフィードバック・レジスタや出力レジスタを構成する**ダブルランク・フリップ・フロップ**を示す. L1, L2 はそれぞれ D ラッチである. カスケードの評価後に select 信号を ON にし, L1 に状態変数の値を格納する. 全カスケードの評価終了後, S_Clock にパルスを加え, 状態変数の値を L2 に転送することにより, 状態遷移が終了する. カスケードの評価を行なう際に使用する C_Clock と, ダブルランク・フリップ・フロップを動作させる S_Clock, の 2 種類のクロックが必要になる. これらのクロックは**制御回路**で制御する. 論理メモリと接続メモリを書き変え可能なメモリで実現することにより, 再構成可能なアーキテクチャとなる. また, メモリ・パッキング [8] を行なって必要メモリ量を削減できる. ただし, メモリ・パッキングを行なうには, プログラマブル接続回路にマッピング情報を記憶した**マッピング・メモリ**が必要となるが, その面積は論理を実現するメモリに比べて小さいので無視できる [10].

3.3 LUT リングを用いた順序回路の実現と模擬

LUT リングを用いて順序回路を以下のように実現し, 模擬する.

(1) 順序回路の組合せ回路部を実現する出力関数を分割し, それぞれを LUT カスケードで実現する.

(2) (1) で実現した LUT カスケードを LUT リングで模擬する.

(3) 出力値の評価を行う際, 外部出力となる出力値は出力レジスタに格納する. また, 状態出力となる出力値はフィードバック・レジスタに格納する.

(4) 全ての LUT カスケードを評価後, フィードバック・レジスタの値をプログラマブル接続回路にフィードバックし, 出力レジスタの値を外部出力として出力する.

[例 3.1] 図 6 に順序回路の模擬を示す. ただし, 図 6 において, x_i は入力変数を, y_i は状態変数を, z_i は外部出力を, u_i はレール変数を示す. また, 順序回路の組合せ部は図 4(a) の LUT カスケードで実現されているものとする.

$time = 1$ でセル 0 の評価を行う. ページ 0 を参照するため, MAR の上位 2 ビットを 00 にセットし, プログラマブル接続回路を介してセル 0 の入力変数 (x_5, y_1, x_4, x_3) をセットする. ページ 0 を読み, 出力値を MBR にセットする. 外部出力 (z_0) を出力レジスタにセットし, 状態出力 (y_0') をフィードバック・レジスタにセットする (図 6(a)).

$time = 2$ でセル 1 の評価を行う. ページ 1 を参照するため, MAR の上位 2 ビットを 01 にセットし, プログラマブル接続回路を介してフィードバックされたセル 0 のレール出力 (u_1, u_0) とセル 1 の入力変数 (x_2, y_0) をセットする. ページ 1 を読み, 出力値を MBR にセット後, 各出力をそれぞれのレジスタにセットする (図 6(b)).

$time = 3$ でセル 2 の評価を行う. ページ 2 を参照するため, MAR の上位ビットを 10 にセットし, プログラマブル接続回路を介して入力変数をセットする. ページ 2 を読み, 出力値を MBR にセット後, 各出力をそれぞれのレジスタにセットする.

すべての LUT カスケードが評価されたので, 制御回路は S_Clock をフィードバック・レジスタと出力レジスタに送り, フィードバック・レジスタの値 (y_0', y_1') をプログラマブル接続回路にフィードバックする. また, 出力レジスタの値 (z_0, z_1, z_2, z_3) を外部出力として出力する (図 6(b)). (例終り)

4. LUT リングの設計アルゴリズム

4.1 出力集合の分割

一般に, 多出力関数を単一の BDD_for_CF で表現した場合, 節点数が増大し, コンピューターのメモリ上に格納できないことが多い. また, BDD_for_CF の幅も増大し, LUT カスケードの実現ができない. この場合, 出力関数を複数のグループに分割し, 各出力集合を個別の LUT カスケードで実現する. LUT リングの評価時間は総セル数に比例する. 評価時間を最小にするために, LUT リングの設計問題を以下のように定式化する.

[問題 4.1] メモリ量制限 L_0 の下で総セル数が最小となる出力の分割, およびそれらの LUT カスケード実現を求めよ.

従来手法 [8] では出力関数を 1 つずつグループに加えて BDD_for_CF を構成し, 変数順序最適化を行うという操作を, LUT カスケード実現できなくなるまで繰り返し行う. しかし, 出力関数をグループに加えるたびに変数順序最適化を行うため, 計算時間が増大し実用的ではない. また, この手法は可能な限り

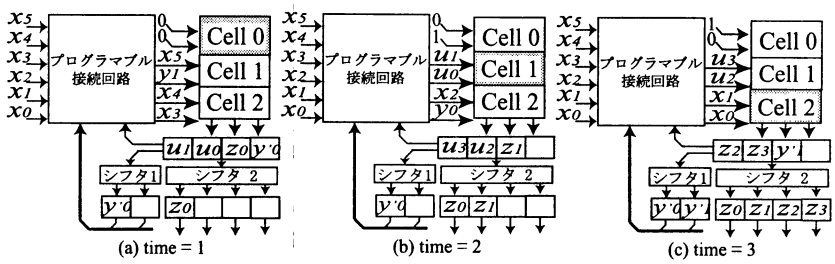


図6 LUT リングを用いた順序回路の模擬

1 つのグループの出力数を増やすので, BDD_for_CF の幅が増大し, レール数が増え, 1 本当りの LUT カスケードの段数が大幅に大きくなる.

そこで, 本論文では, 各グループの出力数を増やしながら, 分割毎に総セル数予測関数関数を計算し, 全カスケードの総セル数を最小にする出力関数の分割を求める方法を提案する. 一般に, BDD_for_CF で表現する出力数が増えると BDD_for_CF の幅や節点数は増える. したがって, 各 BDD_for_CF で表現する出力数がなるべく小さくなるように均等に分割したほうがよい. ここで, 各グループの出力数を決定する問題が生ずる. すべての可能性を試みると計算量が増大してしまい実用的ではない. そこで, 本論文では発見的手法で最適な各グループの出力数を求める. BDD_for_CF で表現する出力数が小さいほど, BDD_for_CF を構成し変数順序最適化を行う計算量は小さいので, 各グループの出力数の初期値を 1 として, 各グループの出力数を同時に増やしながら最適な分割を探索する.

次に, 全カスケードの総セル数予測関数を定義する. この値によって併合を続けるべきか否かを決定する.

[定義 4.1] (総セル数予測関数)
出力の集合が r 個のグループに分割され, 各グループがそれぞれ BDD_for_CF で表現されているものとする. LUT リングのメモリ出力数を w とするとき, i 番目の BDD_for_CF の入力数を n_i, 各レベルでの幅の総和を Sum_u_i, 全カスケードの**総セル数予測関数**を

$$EC = \frac{\sum_{i=1}^{r} n_i - \bar{u} \cdot r}{(w+1) - \bar{u}}, \tag{1}$$

と定義する. ここで $\bar{u} = \dfrac{\sum_{i=1}^{r} \lceil \log_2 Sum\_u_i \rceil}{\sum_{i=1}^{r} n_i}$ は各レベルでの幅の対数の平均値を表す.

[定理 4.1] 式 (1) の総セル数予測関数 EC は与えられた多出力論理関数を実現する LUT カスケードの総セル数の下界を近似する.

(証明)
出力を r 個のグループに分割しそれぞれを BDD_for_CF で表現する. セルの入力数の最大値を k, i 番目の BDD_for_CF の入力数を n_i, セル数を s_i, レール数の平均値を $\hat{u}_i$ とする. 多出力関数をカスケード実現した場合,

$$n_i + \hat{u}_i(s_i - 1) \leqq s_i k \ (i = 1, 2, \ldots, r)$$

が成立する. ただし, $\hat{u}_i$ は直接求まらないので, 全ての BDD_for_CF の各レベルでの幅の対数値の平均値 $\bar{u}$ で近似を行う. これより関係

$$\sum_{i=1}^{r} n_i + \bar{u} \sum_{i=1}^{r} (s_i - 1) \leqq \sum_{i=1}^{r} s_i \cdot k \tag{2}$$

を得る. 与えられた論理関数が LUT カスケードで構成可能な条件は

(BDD の幅の対数値) ≦ (セルの入力数 − 1) かつ
(BDD の幅の対数値) ≦ メモリ出力数

であるから, 式 (2) に対して $k = w + 1$, $Cs = \sum_{i=1}^{r} s_i$ とおくと $EC \leqq Cs$ を得る. (証明終)

出力関数をグループ化する際, なるべく小さな BDD_for_CF を構成するためにサポート変数が増えないように出力関数を分割する.

[定義 4.2] 論理関数の集合を $F = \{f_1, f_2, \ldots, f_m\}$ とし, $G \subseteqq F$ とし, $f_i \in F - G$ とする. G に**最も類似した出力** f_i とは

$$Similarity(i, G, F) = |Sup(f_i) \cap Sup(G)|, \tag{3}$$

の値を最大にする f_i のことである. ここで, $Sup(F)$ は F のサポート変数の集合を示す.

次に, 与えられた多出力論理関数の出力を分割するアルゴリズムを示す.

[アルゴリズム 4.1] (出力の分割と BDD_for_CF の構成)
論理関数の集合を $F = \{f_1, f_2, \ldots, f_m\}$, 分割後の出力関数の部分集合の集合を $\mathcal{Z} = \{G_1, G_2, \ldots, G_r\}$, 各グループの出力数の最大値を g, 分割の個数を r とする.

(1) $g \leftarrow 0, thEC \leftarrow \infty, H \leftarrow F$.
(2) $g \leftarrow g + 1, EC \leftarrow 0, \mathcal{Z} \leftarrow \phi, r \leftarrow 1$.
(3) $G_r \leftarrow \phi$.
(4) F の中で G_r に最も類似した出力 f_i を選ぶ. $G_r = \phi$ ならば, 依存変数が最も多い f_i を選ぶ.
(5) $F \leftarrow F - \{f_i\}, G_r \leftarrow G_r \cup \{f_i\}$.
(6) $|G_r| \leqq g$ かつ $F \neq \phi$ ならば 4 へ戻る.
(7) G_r を表現する BDD_for_CF を構成し, $\mathcal{Z} \leftarrow \mathcal{Z} \cup \{G_r\}$.
(8) $F \neq \phi$ ならば $r \leftarrow r + 1$ として 3 へ戻る.
(9) 現在の分割 $\mathcal{Z}$ に対する式 (1) を計算し EC とする.
(10) $thEC < EC$ または $g = |F|$ なら停止. そうでなければ, $thEC \leftarrow EC, F \leftarrow H$ とし 2 へ戻る.

4.2 カスケード実現とメモリパッキング

アルゴリズム 4.1 により, 与えられた論理関数の出力が複数のグループに分割され, 各グループが BDD_for_CF で表現されているものとする.

4.2.1 メモリ量の下界とセルの入力数の関係

[定理 4.2] n 変数論理関数を図 3 に示す LUT リングで実現する場合, 最大入力を有するセルの入力数を k, 使用可能な論理メモリ量を L[bit], 論理メモリの出力数を w, BDD の幅の最大値を μ とすると $n \geqq k$ のとき,

$$\frac{2^k}{k-1} \leqq \frac{L}{w(n-1)} \tag{4}$$

が成立する.

式 (4) よりセルの入力数の上界が得られる.

[例 4.1] ベンチマーク関数 s208 を $L = 2^{20}(= 1Mbit), w = 16$ のメモリ上に実現する場合, $n = 18$ であるから, $k \leqq 16$ の k に関して調べれば十分である.

LUT のデータを LUT リングのメモリ上に格納する際, メモリ・パッキングを行なって必要メモリ量を削減できる. LUT カスケードの総セル数を s とすると, メモリパッキングには $O(s)$ の計算時間が必要である. 総セル数 s は比較的小さいため, BDD の計算に比べるとメモリパッキングの計算時間は無視できる.

本手法は各カスケードのセルの入力数の最大値を変化させながら, メモリパッキングを行って LUT リングのメモリに格納できるか否かをダイナミック・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り, 部分問題で得られた最適解をもとに次の部分問題の最適解を順次求めていく.

以下に, メモリ量制限下で総セル数が最小となるカスケード実現を求めるアルゴリズムを示す.

[アルゴリズム 4.2] (メモリ量制限下で総セル数が最小となるカスケード実現)

アルゴリズム 4.1 により論理関数の集合 $F = \{f_1, f_2, \ldots, f_m\}$ が r 個の BDD-for-CF($bddcf_1, bddcf_2, \ldots, bddcf_r$) で表現されているものとする. また, メモリ量制限 L が与えられているものとする. 式 (4) で得られるセルの入力数の上限を k_{max} とすると, $S_i(1 \leq i \leq r)$ はセルの入力数を $k_j \in \{1, 2, \ldots, k_{max}\}$ としてカスケード実現を行って得られるセル数 s_j を要素とする集合である. ここで, 総セルの最大値を $l_{max} = \sum_{i=1}^{r} \max_{s \in S_i} s$ とし, 計算を終えた LUT カスケードの総セル数を l とする. 計算途中の最小メモリ量を要素とする配列を $mem\_opt(l)(1 \leq l \leq l_{max})$ とし, セルの入力数を要素とする配列を $cellin(l)(1 \leq l \leq l_{max})$ とする. また, 計算を終えた $bddcf_i$ のセルの入力数 k_i を要素とする集合を $cellin\_opt_l(1 \leq l \leq l_{max})$ とする.

1: 各 $cellin\_opt_i(1 \leq i \leq l_{max})$ に対して $cellin\_opt_i \leftarrow \phi$ とする.

2: セルの入力数 $k_j \in \{1, 2, \ldots, k_{max}\}$ を変化させ, $bddcf_1$ のカスケード実現を行って得られるメモリ量を mem, セル数を s とし, $mem\_opt(s) \leftarrow mem$, $cellin\_opt_s \leftarrow \{k_j\}$ とする.

3: for($c \leftarrow 2$; $c \leqq r$; $c++$){

4: 各 $mem\_opt(i), cellin(i)(1 \leq i \leq l_{max})$ に対して, $mem\_opt(i) \leftarrow \infty, cellin(i) \leftarrow \phi$ とする.

5: for($l \leftarrow 1$; $l \leq l_{max}$; $l++$){

6: if($|cellin\_opt_l| = c - 1$){

7: $S' \leftarrow S_c$;

8: while($S' \neq \phi$){

9: $s \leftarrow S' - \{s\}, k \leftarrow (bddcf_c$ をカスケード実現したときにセル数が s となるセルの入力数).

10: $C \leftarrow cellin\_opt_l \cup \{k\}$.

11: 各 $bddcf_j(1 \leq j \leq c)$ のセルの入力数を $k_j \in C$ としてカスケード実現とメモリパッキングを行い, 必要メモリ量を mem とする.

表 1 ISCAS'89 の順序回路ベンチマーク関数を実現するための各パラメータ

Name	In	Out	FF	r	1Mbit (64k×16bit)		8Mbit(256k (256k×32bit)		48Mbit (1M×48bit)	
					s	Mem [Mbit]	s	Mem [Mbit]	s	Mem [Mbit]
s208	10	1	8	2	3	0.625	3	0.062	3	0.1875
s344	9	11	15	3	4	0.156	4	0.156	4	0.156
s386	7	7	6	2	3	0.063	2	0.063	2	0.063
s420	18	1	16	1	4	0.500	3	2.000	2	48.000
s510	19	7	6	2	3	0.251	3	0.251	3	0.750
s641	36	23	19	3	10	1.000	7	4.750	6	12.000
s713	36	23	19	3	9	0.755	7	4.000	7	6.000
s820	18	19	5	2	3	0.251	3	0.500	3	0.750
s838	34	1	32	3	11	1.000	10	2.000	7	48.000
s1196	13	13	19	3	7	0.438	5	3.000	4	24.000
s1423	17	5	74	9	52	1.000	34	7.125	28	48.000
s5378	35	49	164	22	—	—	83	7.625	65	48.000
s9234	36	39	211	62	—	—	153	7.968	98	45.750
s13207	62	152	638	79	—	—	283	7.945	208	47.683

12: if($mem\_opt(l+s) > mem$){

13: $mem\_opt(l+s) \leftarrow mem$;

14: $cellin(l+s) \leftarrow \{k\}$;

15: }

16: }

17: }

18: 各 $cellin\_opt_i(1 \leq i \leq l_{max})$ に対して, $cellin\_opt_i \leftarrow cellin\_opt_i \cup \{cellin(i)\}$ とする.

19: }

20: $mem\_opt(Level) \leq L$ を満たす最小の $Level$ を求め停止.

アルゴリズム 4.1 で求めた BDD_for_CF に対して 1 つずつカスケード実現を行いながら必要メモリ量を求め (行 10), メモリ量最小となるか判定を行い (行 11), 最適なセルの入力数の最大値及び, 最小メモリ量を求める.

5. 実験結果

第 4 章のアルゴリズムを C 言語で実装し, ISCAS'89 順序回路のベンチマーク関数 [12] を LUT リングにマッピングした. 制限メモリ量を 1Mbit(64k ワード× 16 ビット) にした場合 (**1Mbit**), 1Mbyte(256k ワード× 32 ビット) にした場合 (**8Mbit**), 6Mbyte(1M ワード× 48 ビット) にした場合 (**48Mbit**) を比較した結果を表 1 に示す. 表中の Name は関数名, In は入力数, Out は出力数, FF は状態変数の個数, r は LUT カスケード数, s は総セル数, Memory はメモリ量で単位は [Mbit] を表す. 実行環境は, IBM PC/AT 互換機, Pentium4 Xeon 2.8GHz, L1 Chache:32KB, L2 Chache:512KB, Memory:4GB, OS:Redhad(Linux 7.3) 上で gcc を用いてコンパイルした. なお, 全ての実験でセルの入力数を可変 [8] とした.

表中の (—) は, 必要メモリ量が制限メモリ量を越えてしまったため合成できなかったことを示す. 本手法はメモリ量に空きがある限り, 段数の小さい LUT カスケードを構成する. 従って, 制限メモリ量を増やすことで高速な LUT リングを得ることができる. 表 1 に示すように, ほとんどの関数で (1Mbit), (8Mbit), (48Mbit) の順に総セル数が減少し, より高速な順序回路を実現できた.

次に, メモリを用いて順序回路を実現する他の方法と, メモリ量, 評価時間について比較を行った結果を表 2 に示す. 表 2 中の Name, In, Out, FF は表 1 と同じものを示す.

表2 LUTリングと他の手法との比較

Name	In	Out	FF	Direct [Mbit]	Mem+ Mux [Mbit]	LCC Mem [Mbit]	LCC time [ns]	LUT Ring Mem [Mbit]	LUT Ring time [ns]
s344	9	11	15	416.000	208.000	0.015	1180	0.006	46.4
s382	3	6	21	432.000	216.000	0.015	1530	0.008	41.9
s386	7	7	6	0.102	0.025	0.016	1630	0.016	19.4
s400	3	6	21	432.000	216.000	0.015	1540	0.008	41.9
s820	18	19	5	192.000	0.188	0.019	4080	0.015	32.9
s1494	8	19	6	0.391	0.049	0.025	7830	0.025	32.9

Direct は順序回路の組合せ回路の部分を率直にメモリで実現した場合の必要メモリ量である. n を入力数, m を出力数, 状態変数の個数を s とすると, 必要メモリ量は $(m+s)\cdot 2^{n+s}$bit であり, 入力数や状態変数が大きい場合, 必要メモリ量が増大し実現不可能となる. 評価時間は1クロックで評価できるため省略した.

Mem+Mux は文献[5]の方法を用いて順序回路を実現した場合の必要メモリ量である. 多くの順序回路の状態遷移関数は状態変数と入力変数の一部に依存する. 状態遷移関数の入力変数は依存している変数の集合 Q に置き換えることができる. $|Q|=q$ とし, m を出力数, s を状態変数の個数とすると, 必要メモリ量は $(m+s)\cdot 2^{q+s}$ となる. 明らかに $q \leqq n$ が成立し, 率直にメモリ上に実現するよりも少ないメモリ量で実現できる. ただし, 各状態で依存している入力変数を選択するために, 2^s : 1-マルチプレクサが q 個必要である.

LCC[1]は汎用CPUを用いる論理シミュレーションの一種である. LCCでは, 論理回路の各ゲートにプログラムコードを割り当て, 入力から出力に対してトポロジカルな順序に評価を行なう. 関数評価は汎用CPU上で行なうため, 安価に実現できるが専用回路に比べて低速である. 評価時間はゲート数に比例する. 本論文では性能比較を行うために, まず, ISCAS'89 ベンチマーク関数から LCC C-code を生成し, gcc でオプション-O2 を与えてコンパイルを行なった. 次に, IBM PC/AT 互換機 (Pentium III 800MHz, メモリ:256MB) 上でテストベクトルを 100 万個評価し, 1 テストベクトル当りの評価時間を測定した. この結果を表2に示す. 評価時間の単位は [ns] である. また, gcc で生成した実行コードの大きさを必要メモリ量とした. 単位は [Mbit] である.

LUT Ring の欄は本手法を用いて ISCAS'89 ベンチマーク関数を LUT リング上に実現した場合の必要メモリ量と評価時間を示す. LUT リングの評価時間は文献[11]を参考に, 以下の式で見積もった.

LUT リングの評価時間 = 4.5× 総セル数 + 5.9 (ns).

また, LUT リングのメモリ量制限は, LCC を用いてベンチマーク関数を実現した場合に必要なメモリ量とした. したがって, LUT リングに必要なメモリ量は LCC で実現した場合に必要となるメモリ量以下になる.

表2に示すように, メモリとマルチプレクサを用いる方法 (Mem+Mux) はメモリ上に直接実現する方法 (Direct) に比べて, 状態変数の個数が少ない場合 (s386,s1494) や, 状態遷移の依存変数が少ない場合 (s820) に, 少ないメモリ量で実現できた. しかし, 状態変数の個数が多い場合や状態遷移がほとんどの入力変数に依存している場合, 必要メモリ量は増大した. LCC と LUT リングはいずれの関数も少ないメモリ量で実現できた. また, LUT リングは LCC と比較して同量のメモリ量で 25〜237 倍高速な回路を合成できた.

6. 結　論

本論文では, LUT リングを用いて順序回路を合成する方法について述べた. 本手法では, メモリ量とセルの入出力数の最大値をパラメータとして与えればよく, 設計はすべて自動的に行われる. さらに, LUT リング上のメモリを有効に利用し高速な順序回路を実現する方法を述べた. また, メモリを用いた順序回路を実現する他の手法と比較を行い, 本手法がメモリ量, 評価時間において有効であることを示した.

7. 謝　辞

本研究は, 一部, 文部科学省・科学研究費補助金, 日本学術振興会・科学研究費補助金, および, 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による. 熱心にご訂論頂いた明治大学井口助教授に深謝する.

文　献

[1] M. Abramovici, M. A. Breuer, and A. D. Friedman, *Digital Systems Testing and Testable Design,* Wiley-IEEE Press; Rev. Print edition, Sept. 1994.

[2] 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, 2001.*

[3] C. H. Clare, *Designing Logic Systems Using Sate Machines,* McGraw-Hill, New York, 1973.

[4] M. Davio, J. -P Deschamps, and A. Thayse, *Digital Systems with Algorithm Implementation,* John Wiley & Sons, New York, 1983.

[5] D. Green, *Modern Logic Design,* Addison-Wesley Publishing Company, 1986.

[6] H. Nakahara, T. Sasao, and M. Matsuura, "A design algorithm for sequential circuits using LUT Rings," SASIMI2004, Oct.2004, pp.430-437.

[7] T. Sasao, M. Matsuura, and Y. Iguchi, "A cascade realization of multiple-output function for reconfigurable hardware," *IWLS-2001,* Lake Tahoe , CA, June 12-15,2001, pp.225-300.

[8] 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, USA, pp.431-437.

[9] 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.

[10] T. Sasao, H. Nakahara, M. Matsuura and Y. Iguchi, "Realization of sequential circuits by look-up table ring," *MWSCAS2004,* Hiroshima, July 25-28, 2004, I517-I520.

[11] H. Qin, T. Sasao, M. Matsuura, S. Nagayama, K. Nakamura, Y. Iguchi, "Realization of multiple-output functions by a sequential look-up table cascade," *IEICE, Transactions on Foundamentals of Electronics, vol.E87-A,*(accepted).

[12] http://www.cbl.ncsu.edu/pub/Benchmark_dirs/ISCAS89/