LUT カスケード・アーキテクチャについて

井口 幸洋 [1]　　笹尾 勤 [2,3]

[1] 明治大学 理工学部 情報科学科
[2] 九州工業大学 情報工学部 電子情報工学科
[3] 九州工業大学 マイクロ化総合技術センター

On Look-Up Table Cascade Architecture

Yukihiro Iguchi[1]　　Tsutomu Sasao[2,3]

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

抄録:

複数のルックアップテーブル (LUT) の接続形態を直列多段のみに限定し, 論理回路を実現する **LUT カスケード・アーキテクチャ**を提案する.

LUT の接続段数を減らすために数 10k-bit 程度の中規模メモリを LUT として用いる. LUT 間の配線は隣同士の LUT 間のみに限定するため配線遅延は小さい. LUT 間を接続する信号線数 (レイル数) は, 実現する論理関数及び分解法に依存する. レイル数を固定すると, 実現可能な関数の個数が少なくなったり, メモリの使用効率が低下する. そこで, アーキテクチャの適用範囲を増やすために LUT 間に接続回路を設ける. この回路を用いると, 途中の LUT に追加の外部入力を入れたり, 途中の LUT から出力を引き出したりすることが可能となる. メモリの使用効率を上げるための構造上の工夫についても述べる. すべての LUT と, 接続を規定する接続メモリを書換え可能にすれば再構成可能なアーキテクチャとして使用可能である.

LUT カスケード・アーキテクチャについて

井口 幸洋 [1]　　笹尾 勤 [2,3]

[1] 明治大学 理工学部 情報科学科
[2] 九州工業大学 情報工学部 電子情報工学科
[3] 九州工業大学 マイクロ化総合技術センター

On Look-Up Table Cascade Architecture

Yukihiro Iguchi[1]　　Tsutomu Sasao[2,3]

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

概要: 複数のルックアップテーブル (LUT) の接続形態を直列多段のみに限定し, 論理回路を実現する **LUT カスケード・アーキテクチャ**を提案する. LUT の接続段数を減らすために数 10k-bit 程度の中規模メモリを LUT として用いる. LUT 間の配線は隣同士の LUT 間のみに限定するため配線遅延は小さい. LUT 間を接続する信号線数 (レイル数) は, 実現する論理関数及び分解法に依存する. レイル数を固定すると, 実現可能な関数の個数が少なくなったり, メモリの使用効率が低下する. そこで, アーキテクチャの適用範囲を増やすために LUT 間に接続回路を設ける. この回路を用いると, 途中の LUT に追加の外部入力を入れたり, 途中の LUT から出力を引き出したりすることが可能となる. メモリの使用効率を上げるための構造上の工夫についても述べる. すべての LUT と, 接続を規定する接続メモリを書換え可能にすれば再構成可能なアーキテクチャとして使用可能である.

キーワード: 再構成可能アーキテクチャ, 多出力論理関数, カスケード実現, FPGA.
Key words: Reconfigurable architecture, Multiple-output logic function, Cascade realization, FPGA.

1 はじめに

システム LSI の集積度が増大するにつれて, その開発期間や, 開発経費が増大している [1]. この問題を解決する一つの方法は, 再構成可能なアーキテクチャを採用することである. アーキテクチャを再構成可能なものとして, 一つのシステム LSI を種々の用途に利用できるようにすれば, ハードウエアのコストは, 飛躍的に削減できる.

本稿では, 簡単のために, 多出力論理関数を実現する組合せ回路の実現を考える. 再構成可能なアーキテクチャで最も単純なものとして, RAM(Random Access Memory) や PLA(Programmable Logic Array) が知られている. RAM や PLA を用いて組合せ回路を直接実現する場合, 変数の個数 n が大きい時, 必要なハードウエア量が大きくなり過ぎる. 従って, 一般には, FPGA(Field Programmable Gate Array)[3] を用いるが, FPGA では, 論理設計の他に, 配置配線設計等が必要になる. また, チップ上に FPGA を組み込む場合, 論理を実現する論理領域よりもむしろ配線領域が大きくなり, 必要なチップ面積も大きくなりがちである. また, 構造が複雑なのでチップも高価であるという問題もある. 演算速度が低くてもよい場合には, 汎用マイクロプロセッサを利用できる. しかし, マイクロプロセッサによる実現は, 専用組合せ論理回路に比べ 2 桁から 3 桁遅くなる.

2 順序回路方式LUTカスケード・アーキテクチャ

我々はマイクロプロセッサと FPGA との中間の性能を持った実現方法として, LUT カスケード・アーキテクチャを提案している. LUT カスケードは, 図 1 に示すように LUT(look-up table) を直列多段に接続した構造で論理関数を実現する方法である.

隣接する LUT 間を結ぶ信号線を**レイル**と呼ぶ. レイルの本数 (**レイル数**) は, 実現すべき論理関数や論理合成法に依存する. レイル数が固定である場合, 様々な論理関数の実現に対応するには, レイル数の大きなものが必要である. この場合はメモリの使用効率が低下する. そこで, 図 2 に示すように, 複数の LUT をまとめて一つのメモリで実現し, これに入力レジスタ, メモリ番地を保持する MAR(memroy address register), 出力レジスタとシーケンサを備えた方式を提案した [9]. この方式を**順序回路方式 LUT カスケード・アーキテクチャ**と呼ぶ. 以下では, 例題で順序回路方式 LUT カスケード・アーキテクチャを概説する.

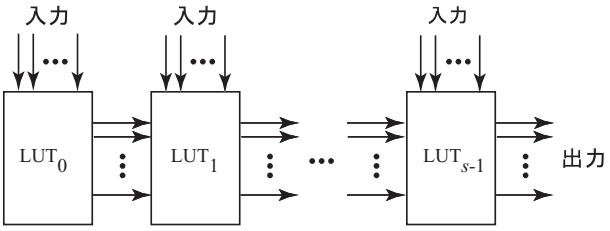

図 1: LUT カスケード

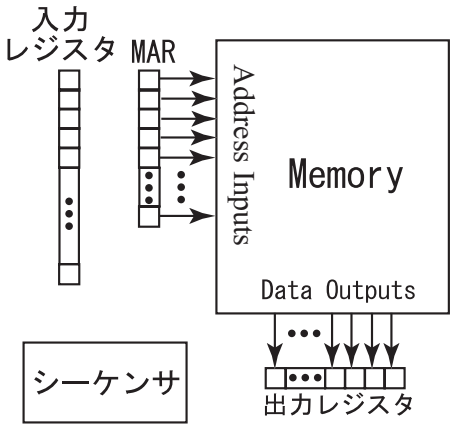

図 2: 順序回路方式 LUT カスケード・アーキテクチャ

例 1 . 図 *3*に, *2*つの *4* ビットの *2*進数 $A = (a_3, a_2, a_1, a_0)$, $B = (b_3, b_2, b_1, b_0)$ の大小を比較し, $A > B$ なら $F = (0, 0)$, $A = B$ なら $F = (1, 1)$, $A < B$ なら $F = (0, 1)$ の *2* ビットを出力する比較器の LUT カスケード実現を考える.

一つのメモリだけで単純に実現する場合は, *8* 入力 *2* 出力つまり $2^8 \cdot 2 = 512$-*bit* の容量のメモリが必要になる. これに対し, *4* 入力 *2* 出力の $LUT(2^4 \cdot 2 = 32$-*bit* メモリ*)* を *3* 個, つまり *96-bit* の容量で LUT カスケード実現した例を図 *3(a)* に示す.

順序回路方式 LUT カスケード・アーキテクチャの実行の様子を図 *3(b)(c)(d)* に示す.

1. 時刻 *0:* 図 *3(a)* の第 *0* 段目の部分の計算に必要な信号 a_3, b_3, a_2, 及び, b_2 の値を入力レジスタから MAR に移動する. LUT_0 が実現する真理値表はメモリのページ *0* に入れてあるので, メモリのアドレス A_5, A_4 には, ページ *0* を参照するためにそれぞれ *0* を代入する. メモリの読み出しを行うと, 指定アドレスの内容が出力される. この値を y_1, y_0 とする.

2. 時刻 *1:* 図 *3(a)* の第 *1* 段目の部分の計算に必要な信号 a_1, b_1 の値を入力レジスタから MAR に移動する. 時刻 *0* で求めた y_1, y_0 も MAR に移動する.

LUT_1 の真理値表はメモリのページ *1* に入れてあるので, $A_5 = 0$, $A_4 = 1$ を代入する. メモリの読み出しを行うと, 指定アドレスの内容が出力される. この値を y_3, y_2 とする.

3. 時刻 *2:* 図 *3(a)* の第 *2* 段目の部分の計算に必要な信号 a_0, b_0 の値を入力レジスタから *MAR* に移動する. 時刻 *1* で求めた y_3, y_2 も *MAR* に移動する. LUT_2 の真理値表はメモリのページ *2* に入れてあるので, $A_5 = 1$, $A_4 = 0$ を代入する. メモリの読み出しを行うと, 指定アドレスの内容が出力される. この時, 値 y_5, y_4 が求める出力となる.

(例終り)

本方式では, 出力をメモリの入力につながる 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. つまり, 論理はハードウェアであるメモリで, 配線はシーケンサのソフトウェアで実現している. 一方, FPGA は論理と配線ともハードウェアで実現している. 複数の LUT を一つのメモリで実現し, 制御をシーケンサで実現したことで, LUT カスケードの段数, レイル数の変化に柔軟に対応できるのが特長である. つまり,

- 専用回路を個別に設計しなければならないほどは高速性は要求されないが, ソフトウェア実現では低速で困る場合.
- PLA, RAM では直接実現できないほどの複雑さのもの. 例えば, 入力変数, 出力変数の個数が大きい場合.
- 設計時間が短く, 設計変更に柔軟に対応でき, 定期的に, バージョンアップが必要なシステムなど.

に向いている.

しかし, カスケードの段数分だけメモリをアクセスする必要があり, シーケンサを用いているので, 高速化が難しいという問題点がある. 高速動作が必要な場合のアーキテクチャを次章で提案する

3 組合せ回路方式 LUT カスケード・アーキテクチャ

図 1 のように LUT を実際にカスケード (直列) 接続したアーキテクチャのプログラマブル・デバイスがあったとする. これを利用して, 所望の論理関数を実現する場合の問題点は以下の通りである.

1. LUT の段数が論理関数と論理合成方式に依存する.

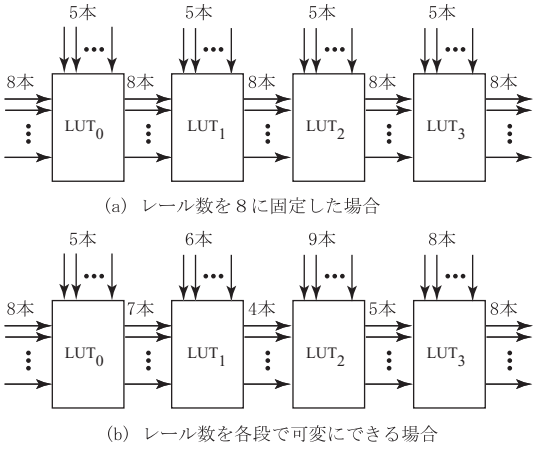

図 4: レイル数が実現規模に与える影響

2. LUT 間をつなぐ信号線数 (レイル数) が論理関数と論理合成方式に依存する.

実際のデバイスでは, LUT の入力数や出力数は固定されており, 段数も固定されている. 以後, 話を簡単にするために 13 入力 8 出力の 8KByte の LUT が 4 段カスケード接続された構造例で考える.

3.1 段数の自由度について

段数に関しては, 必要な段数だけデバイスをカスケード接続すれば解決できる. 例えば, 5 段の LUT カスケード接続が必要な場合, 2 個のデバイスを使用すれば良い. ただし, 8 段全部を信号が通過するとなると残り 3 段分の遅延と LUT が無駄になる. この問題の解決を考える必要があるが, これは後述する LUT 間のレイル数の柔軟化で解決できる.

3.2 レイル数の柔軟化

例 2 図 *4* に *13* 入力 *8* 出力の *8KByte LUT* を *4* 段カスケード接続した *2* つの例を示す. *(a)* は, レイル数が *8* に固定されている. 外部入力数は $8+5+5+5+5 = 28$ である. 一方, *(b)* は *(a)* と同じ容量の *LUT* を使用しているが, レイル数が *7* 本, *4* 本, *5* 本であるので, LUT_1 への外部入力は *5* 本から *6* 本に増やせる. 外部入力数は $8+5+6+9+8 = 36$ に増えている.

このように, レイル数に柔軟性があると同容量のデバイスでも実現できる論理関数の範囲は大きくなる. (例終り)

FPGA では, 様々な LUT 間を自由に配線できるような構造が必要である. 一方, LUT カスケードでは隣接す

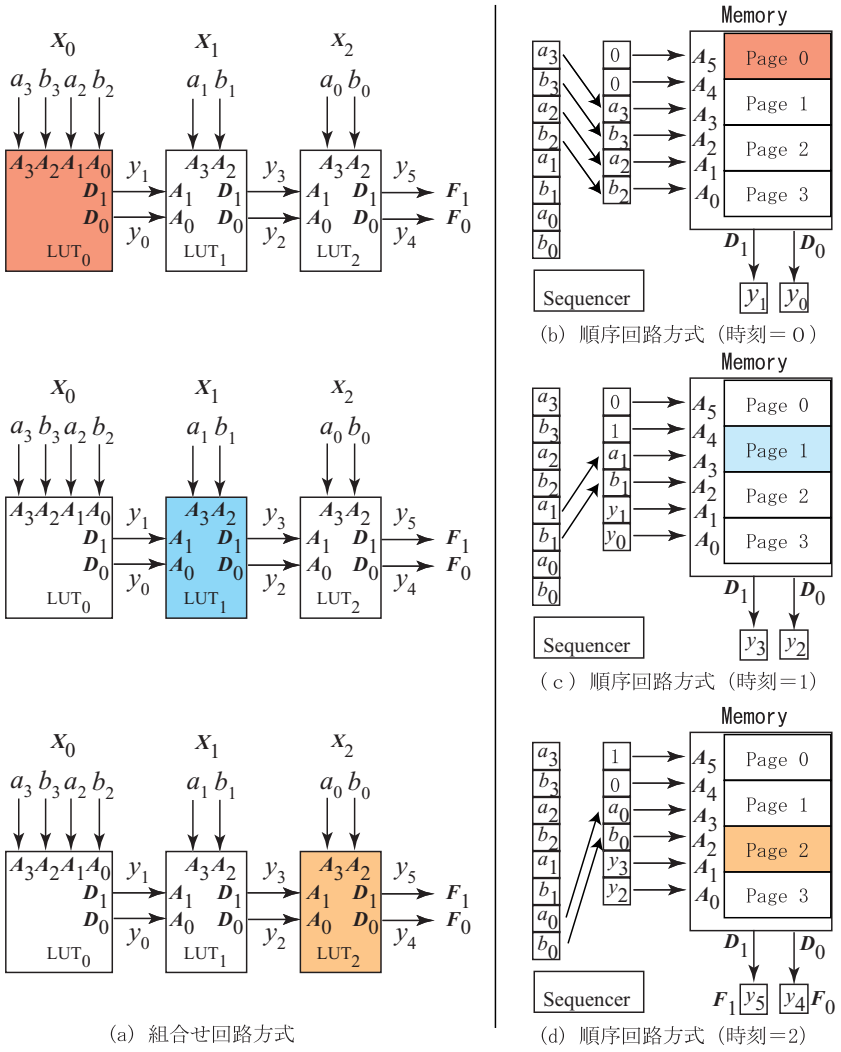

図 3: 順序回路方式 LUT カスケード・アーキテクチャの動作

る LUT 間のみで, ある程度の自由度を持って接続できればよい. そこで, 図 5 に示す例のように最大レイル数を 8 として, LUT_k と LUT_{k+1} の間のレイル数が 7 で十分な場合は, 後段の外部入力を 1 本増やす. 一方, 前段の出力 1 本は未使用になる. そこで, それを外部出力に引き出す機構を付加し, これを外部出力として利用する. 同様にレイル数が 6 で十分な場合は, LUT_k から 2 本を外部出力に引き出し, LUT_{k+1} への外部入力を 2 本増やして 7 本にする. 外部出力への引き出しは, 未使用出力を活用できる利点の他に, 演算の高速化の点からも望ましい.

LUT 間に接続回路を入れることによって, レイル数の柔軟化と外部出力への途中の LUT からの出力の引き出しの両方を実現できる. 接続回路としては種々のものが利用できる. 例えば, 図 6 ではバレルシフタを用いている. レイル数を最大の 8 にしたいときは, シフト量を 0 にする. レイル数が 7 の時は, シフト量を 1 とし外部から入力を後段の LUT に, 前段の LUT から 1 本を外部に出力できる. 以後, 同様に $2, 3, \ldots, 8$ とシフト量を増やせる. シフト量を 8 にすると, LUT_k の出力を全て出力し, それ以後の LUT_{k+1} 以後は, また別の出力を求めるために使用できる.

接続回路はシフタの他に, クロスバースイッチなどでも実現できる.

3.3 出力選択

最終段の LUT の出力数は 8 である. 必要な出力数が 4 であれば, メモリが無駄となる. そこで, 図 7 に示すよう

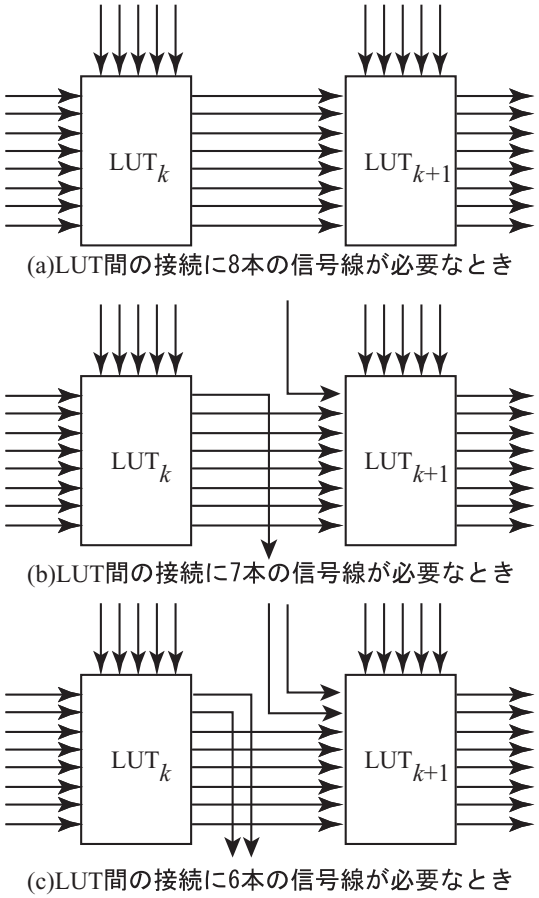

図 5: レイル数柔軟化と LUT 出力の途中からの引出し

に出力の外にマルチプレクサを付加し, マルチプレクサの選択信号を外部入力とすることで外部入力数を増やせる. 例えば, マルチプレクサを 1 段挿入することにより, 13 入力 8 出力のメモリを, 14 入力 4 出力のメモリとして使用できる. 同様に, 2 段挿入すると, 15 入力 2 出力のメモリ, 3 段挿入すると, 16 入力 1 出力のメモリとして利用できる. ただし, これらの出力すべて $8+4+2+1=15$ 本を外部に出力するのは端子の無駄なので, これを 8 本にまとめる回路を挿入する.

3.4 入力ピンの自由度の増加法

プログラマブル・デバイスにおいて, 特定の入力端子に特定の入力変数を割り当てることを強制するのは, 基板設計などの点や仕様変更などの点から不便である.

その場合, 任意の入力端子から各 LUT に自由に信号を分配できる構造が必要になる. 例えば, クロスバースイッチを使えば自由に配線可能となる. また, シフタを使えば自由度は固定の場合よりは増える.

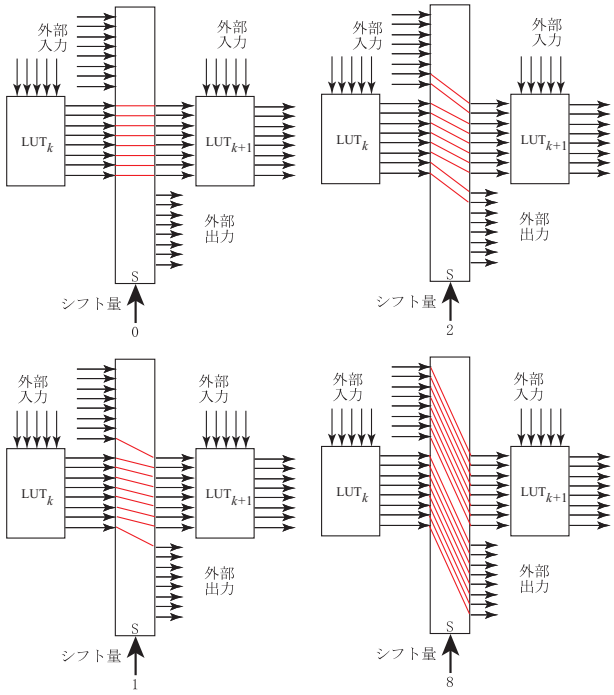

図 6: レイル数柔軟化と LUT 出力の途中からの引出しのための接続回路

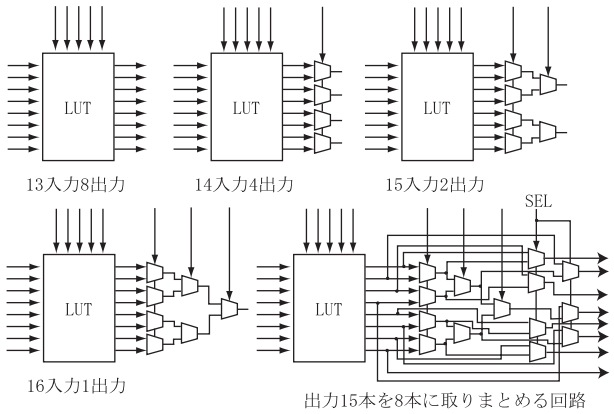

図 7: 最終段 LUT の出力選択回路

図 1 のカスケードに, 本章で述べた 3.2~3.4 の機能を実現する回路を挿入したものを, 図 8 に示す. ここで, 接続メモリとは接続回路の接続情報を格納するための記憶装置であり, 小規模メモリあるいは, レジスタファイルで実現できる. 例えば, 接続回路をシフタで実現する場合には, この接続メモリ内にはシフト量を書き込む. 同様に, 入力選択メモリは, 各 LUT と接続回路にどの入力を選択して供給するかの情報を格納する. 出力選択メモリには, 出力を選択する情報を書き込む.

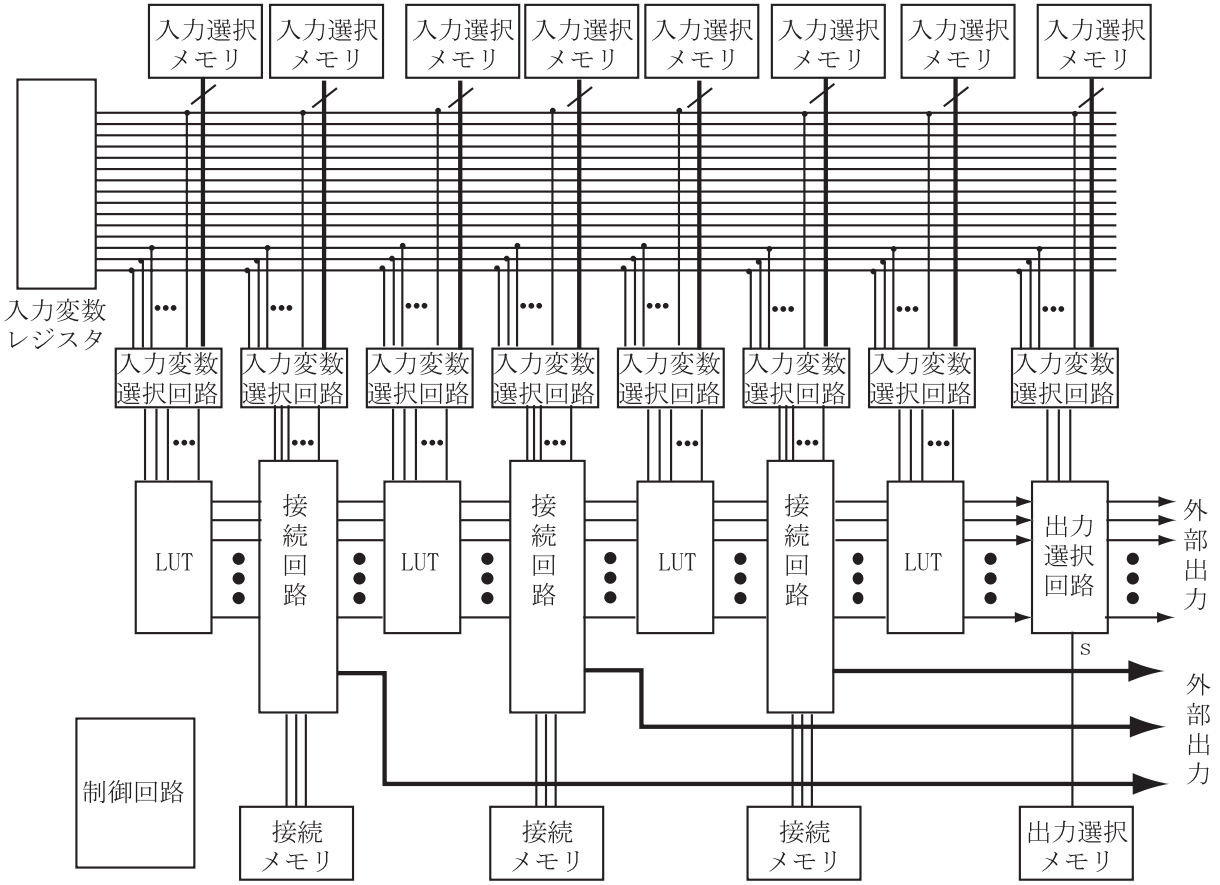

図 8: 組合せ回路方式 LUT カスケード・アーキテクチャの構造

3.5 再構成可能デバイスへの応用

本方式では, LUT を基本素子に使っているおり, LUT 内の情報, 接続メモリ, 入力選択メモリ, 及び, 出力選択メモリ内の情報を書き換えれば回路を再構成できる.

同時に使用することのない, 異なる複数の機能を本方式で実現できる. 例えば 4 種類の異なる機能の場合, LUT は各々の真理値表を, そして接続メモリ, 入力選択メモリ, 及び, 出力選択メモリ内の情報を 4 個用意しておく. これを切り替えれば, メモリと信号線の切り替え時間とそれに伴う信号遅延の時間だけで, 機能を切り替え可能である.

4 まとめ

本稿では, メモリを活用した再構成可能アーキテクチャとして, LUT カスケードを提案した. 従来の FPGA などの構造に比べ, 構造が簡単であり, チップ面積のほとんどがメモリで占められる. このアーキテクチャは, 配線部分が少ないという特長を持っている. これに基づいたチップ試作については稿を改めて報告する予定である. また, 紙面の関係で論理合成手法については述べなかったが, 本アーキテクチャに使うことのできる論理合成ツール [2] が開発されている.

謝辞

本研究は, 一部, 文部科学省・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. また, LUT カスケードチップの試作を共同で行っている, 九州工業大学・マイクロ化総合技術センターの中村和之助教授に深謝する.

参考文献

[1] R. K. Brayton, “The future of logic synthesis and verification,” in Hassoun and Sasao (eds.), *Logic Synthesis and Verification*, Kluwer Academic Publishers, Norwell, MA, USA, 2002.

[2] A. Mishchenko and T. Sasao, “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 設計技術研究会, VLD2002-99, 琵琶湖, (2002-11).

[3] R. Murgai, R. K. Brayton, and A. Sangiovanni-Vincentelli, *Logic Synthesis for Field-Programmable Gate Arrays*, Kluwer Academic Publishers, Norwell, MA, USA, 1995.

[4] T. Sasao, “FPGA design by generalized functional decomposition,” *(Sasao ed.) Logic Synthesis and Optimization*, Kluwer Academic Publishers, 1993.

[5] T. Sasao and M. Fujita (ed.), *Representations of Discrete Functions*, Kluwer Academic Publishers 1996.

[6] T. Sasao, *Switching Theory for Logic Synthesis*, Kluwer Academic Publishers, 1999.

[7] http:\\www.altera.com

[8] http:\\www.aritec.com

[9] T. Sasao, M. Matsuura, and Y. Iguchi, “A cascade realization of multiple-output function for reconfigurable hardware,” *International Workshop on Logic and Synthesis(IWLS01),* Lake Tahoe, CA, June 12-15, 2001. pp.225–230.