TK-2-2

メモリ構造をしたプログラム可能論理素子とその応用

MEMORY-BASED PROGRAMMABLE LOGIC ELEMENTS and THEIR APPLICATIONS

笹尾　勤 [1]
Tsutomu Sasao

九州工業大学　情報工学部 [1]
Department of Computer Science, Kyushu Institute of Technololgy

1　まえがき

本稿では, 知的クラスタ創成事業補助金を受けて当研究室で 2002 年 4 月から 2007 年 3 月までに行ったメモリ構造をした再構成可能論理素子について概説する. LUT カスケード [7] は図 1 のようにメモリを直列接続したものであり, 幾つかの有用なクラスの論理関数を効率よく実現する.

2　関数分解と LUT カスケード

任意の論理関数は, 単一のメモリで実現できるが 入力変数の増加とともに, 必要なメモリの大きさが指数関数的に増加する.

定義 1 多出力論理関数 $\vec{F}$ の**分解表** *[6]* のうちで, 分解表の列を左から右に, また, 行を上から下に移動した場合, 列のラベルの値および行のラベルの値が増加するような分解表を**基本分解表**という. 分解表の異なる列パターンの個数を**分解表の列複雑度**という. 論理関数 $f(x_1, x_2, \ldots, x_n)$ の **C 尺度**とは, $f(x_1, x_2, \ldots, x_n)$ の n 個の基本分解表の列複雑度の最大値である.

定理 1 与えられた関数 f に対して, $\vec{X}_L$ を列を表わす変数とし, $\vec{X}_H$ を行を表わす変数とする. また, μ を分解表の列複雑度とする. このとき, 関数 f は, 図 *2* の回路で実現できる. この場合, 二つのブロック H と G の間を連結する信号線数は $\lceil \log_2 \mu \rceil$ である.

二つのブロック間を連結する信号線数が $\vec{X}_L$ の入力変数の個数よりも少ない場合, 図 2 の実現により, この関数を実現するための総メモリ量を削減できることが多い. この技法を **関数分解**という. 与えられた関数に関数分解を繰り返し適用することにより, 図 1 に示す **LUT カスケード** [7] を得る. LUT カスケードは, **セル**から構成され, 隣接するセル間を連結する信号線を**レイル**という. C 尺度が小さな論理関数は, コンパクトな LUT カスケードで実現できる.

定理 2 *[7]* 関数 f の C 尺度を μ とするとき, f は入力数が高々$\lceil \log_2 \mu \rceil + 1$, 出力数が $\lceil \log_2 \mu \rceil$ のセルを用いた LUT カスケードで実現できる.

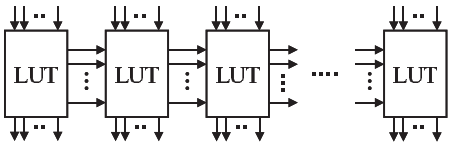

図 1　LUT カスケード.

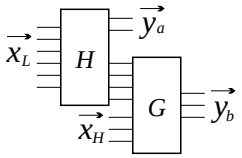

図 2　論理関数の分解による実現.

3　C 尺度の小さな論理関数

数学的には, 殆どすべての関数において, C 尺度は, 入力変数の個数 n の増加とともに指数関数的に増加する. しかし, 多くの実用的な関数では C 尺度は比較的小さい. 以下に示す幾つかの実用的な関数では, C 尺度が小さく LUT カスケードで効率的に実現できる.

定義 2 アドレス表は, k 個の異なる二値ベクトルを 1 番から k 番までのアドレスに格納している. アドレス表に対応する**アドレス生成関数**とは, 以下の条件を満たす論理関数 $\vec{f}: B^n \rightarrow B^m, m = \lceil \log(k+1) \rceil$ である. アドレス表中に $\vec{x}$ と完全に一致したベクトルが格納されている場合には $\vec{f}(\vec{x})$ は $\vec{x}$ に対応したアドレスを生成する. 一致したベクトルが格納されていない場合には, 0^m を出力とする.

論理関数 $\vec{f}(\vec{x})$ において, $\vec{f}(\vec{a}) \neq 0$ となる入力 $\vec{a}$ の個数を $\vec{f}$ の**非零出力数**という.

定理 3 *[9]*. k を関数の非零出力数とする. このとき, 関数の C 尺度は高々$k+1$ である.

上の定理は, 非零要素数の小さな関数は LUT カスケードで効率的に実現できることを示す.

関数の非零出力数が, 入力の組み合わせの総数 2^n に比べて, 十分に小さな関数を**疎な関数**という. アドレス生成関数 [9] は, 疎である場合が多い. 従って, これらの関数は, LUT カスケードで効率的に実現できる.

定義 3 *[12]* **区間指定関数**とは, $SIE: I \rightarrow I$, なる関数である. ここで, I は整数の集合であり, $a \geq b$ のとき $SIE(a) \geq SIE(b)$ が成立する. **区間指定論理関数**とは, 区間指定関数を二値変数で表現した関数である.

定理 4 *[12]* 区間数 λ の SIE 関数の C 尺度は高々λ である.

定義 4 LPM テーブルは, $VEC_1 \cdot VEC_2$ の形をしたベクトルを格納する. ここで, VEC_1 は *0* または *1* のストリングから構成され, VEC_2 は, *から構成されたストリングである. 最長プレフィックスをもったアドレスを見つけるために, LPM テーブルでは, プレフィックスの長い順にベクトルを格納する. 対応する **LPM 関数** は次

2008/3/18 ～ 21　北九州市

のような論理関数 $\vec{f}: B^n \to B^m$ である．ここで，$\vec{f}(\vec{x})$ はドント・ケア以外の部分が $\vec{x}$ と一致する場合，対応する最小のアドレスである．そのようなベクトルが格納されていない場合には，$\vec{f}(\vec{x}) = 0^m$ とする．最長プレフィックス一致（*LPM: longest prefix match)* 関数とは，入力に対して *LPM* テーブルに格納されたベクトルのうちで最長のプレフィックスに一致したベクトルのアドレスを生成する関数である．

LPM 関数 [10] はインターネット用ルータのホップ・アドレスの決定 [5] 等で使用する．

定理 5 *[10]* ベクトル数 k の *LPM* 関数の C 尺度は高々 $k+1$ である．

定義 5 n 入力 **WS 関数** $F(\vec{X})$ は，$WS(\vec{X}) = \sum_{i=1}^{n} w_i \cdot x_i$ の値を計算する．ここで $\vec{X} = (x_1, x_2, \ldots, x_n)$ は**入力ベクトル**，$\vec{W} = (w_1, w_2, \ldots, w_n)$ は**重みベクトル**，w_i $(i = 1, 2, \ldots, n)$ は整数である．$\vec{F} = (f_{q-1}, f_{q-2}, \ldots, f_0)$ を *WS* 関数の二進表現とするとき，関係 $WS(\vec{X}) = \sum_{i=0}^{q-1} f_i(\vec{X}) \cdot 2^i$ が成立する．

WS 関数 [8] は，**基数変換回路** [1]，分散演算回路等の設計に応用できる．

定理 6 *[8]* 重みベクトルが $\vec{W} = (w_1, w_2, \ldots, w_n)$ の WS 関数の C 尺度は高々 $1 + \sum_{j=1}^{n} |w_j|$ である．

WS 関数の重みの総和が大きい場合，その関数を単一の LUT カスケードで実現するとカスケードは大きくなる．これは，WS 関数の C 尺度が n とともに指数関数的に増加するからである．その場合，出力を幾つかに分割し，各グループ毎に独立なカスケードで関数を実現し，最後に加算器で結合する．これを WS 関数の**算術分解**という．

4　数値関数生成器への応用

プログラマブル数値関数生成器 (NFG) のアーキテクチャとその合成法を開発した [12]．与えられた関数の定義域を幾つかの区間に分割し，各区間において関数を一次関数で近似する．定理 4 に示したように，区間生成回路（SIE 関数）は LUT カスケードで効率的に実現できる．三角関数，対数関数，平方根，逆数などの初等関数の広い定義域に対応した数値関数生成器を実現できる．

5　パターン・マッチング回路への応用

厳密マッチングでは，検索パターンがアドレス表中の登録ベクトルと一致するか調べ，一致する場合にはそのアドレスを，一致しない場合には 0 を生成する．これは，アドレス生成関数に対応する．定理 3 より，アドレス生成関数は LUT カスケードで効率的に実現できる [9, 10, 11]．

6　サイクルベース論理シミュレータへの応用

与えられた回路を LUT カスケードに変換し，次に，その LUT のデータを LUT カスケード・エミュレータのメモリに格納する．さらに，LUT カスケード・エミュレータの制御回路を表現する C 言語を生成し，最後に，C 言語を実行モジュールに変換し，PC 上で実行する．本方法は LCC 法に比べ 12-64 倍高速である [3]．

7　LUT カスケード用チップ

0.35μ CMOS プロセスを用いてチップを試作した [4]．8 個の 64 K ビット同期式 SRAM からなる．

謝辞

本研究は一部，文部科学省および日本学術振興会科学研究費補助金による．本プロジェクトには多数の人が貢献した．井口幸洋，松浦宗寛，中村和之，中原啓貴，永山忍，鈴木隆広，Alan Mishchenko, Jon T. Butler, Bogdan Falkowski, Marek Perkowski, Hui Qin, Marc Riedel．ヤマハ　臼井章氏，千葉雅之氏には，応用に関して貴重な討論を頂いた．知的クラスタ本部の影山隆雄氏，大田俊彦氏には大変お世話になった．

参考文献

[1] Y. Iguchi, T. Sasao, and M. Matsuura, "Design methods of radix converters using arithmetic decompositions," *IEICE Trans. on Information and Systems*, Vol.E90-D, No.6, June 2007, pp.905-914.

[2] M. Matsuura and T. Sasao, "BDD representation for incompletely specifiled multiple-output logic functions and its application to the design of LUT cascades," *IEICE Transaction on Fundamentals of Electronics, Communications and Computer Sciences*, Vol.E90-A, No.12, Dec. 2007, pp.2770-2777.

[3] H. Nakahara and T. Sasao, "A PC-based logic simulator using a look-up table cascade emulator," *IEICE*, Vol. E89-A, No.12, Dec. 2006, pp.3471-3481.

[4] K. Nakamura, T. Sasao, et.al, "A memory-based programmable logic device using look-up table cascade with synchronous static random access memories," *Japanese Journal of Applied Physics*, Vol. 45, No. 4B, 2006, April, 2006, pp. 3295-3300.

[5] H. Qin, T. Sasao, and J. T. Butler, "On the design of LPM address generators using multiple LUT Cascades on FPGAs," *International Journal of Electronics*, Vol. 94, Issue 5, May 2007, pp.451-467.

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

[7] 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.

[8] T. Sasao, "Analysis and synthesis of weighted-sum functions," *IEEE TCAD*, Vol. 25, No. 5, May 2006, pp. 789 - 796.

[9] T. Sasao, "Design methods for multiple-valued input address generators," *ISMVL-2006 (invited paper)*, Singapore, May 17-20, 2006.

[10] T. Sasao and J. T. Butler, "Implementation of multiple-valued CAM functions by LUT cascades," *ISMVL-2006*, Singapore, May 17-20, 2006.

[11] T. Sasao and M. Matsuura, "An implementation of an address generator using hash memories," *DSD 2007* Aug.27-31, 2007, Lubeck, Germany, pp.69-76.

[12] T. Sasao, S. Nagayama and J. T. Butler, "Numerical function generators using LUT cascades," *IEEE Transactions on Computers*, Vol.56, No.6, June 2007, pp.826-838.