パターンマッチング用プログラマブル論理回路とその設計法

A Programmable Logic Circuit for Pattern Matching and Its Design Methods

笹尾　勤

1. はじめに

本稿は，通信用パターンマッチング回路とその設計に関する最近の研究結果を示す．現在，インターネットは日常生活に必須のものとなっている．そこでは，高速のパターンマッチングが必要とされる．IPアドレス表のルックアップ(用語)，パケットフィルタ(用語)，端末アクセス制御などがその例である．これらの処理はソフトウェアで実現すると時間がかかるので，通常，ハードウェアで行う．この処理を行う専用ハードウェアとして，連想メモリ(用語)（CAM：Content Addressable Memory）が従来から使用されているが，この素子は，高価であり消費電力も大きい．筆者らは，文部科学省地域イノベーションクラスター事業の補助を受け，この分野の研究を行ったが[1]，その過程で，「インデックス生成関数」という新たな概念を構築し，汎用メモリを用いてパターンマッチング回路を構成する方法を開発した．この理論は，IPアドレス処理のほかに，コンピュータウイルス検出回路，メモリの故障マップの圧縮，電子辞書，パスワードリスト，符号変換回路の設計にも流用できる．

2. 通信用パターンマッチング回路

ここでは，通信用パターンマッチング回路を説明するために，ローカルエリアネットワーク（LAN）用の端末アクセス制御回路（TAC）を用いる．TACは，LANに接続した端末が，LANの外部に対してE-mail，FTP，あるいはTelnetなどの操作を行うことが許可されているか否かを判断する．図1に，TACの例を示す．ここでは，3台の端末がTACに接続している．各端末は，48 bitで表現された固有のMACアドレスを持つ．第1の端末のMACアドレスは53:03:74:59:03:32であり，この端末は全ての操作，つまり，LAN外へのWeb，E-mail，FTP，及びTelnetが許可されている．第2の端末は，WebとE-mailが許可されている．第3の端末は，Webのみ許可されている．TACを，単一メモリで実現する場合，入力数が48なので$2^{48}\approx256$テラワードの巨大なメモリが必要となる．このため，TACをインデックスを生成するインデックス生成器と，各端末のアクセス権の有無を記憶するメモリの二つの部分に分解する．メモリには，端末の詳細情報を格納する．TACは，頻繁に更新する必要がある．

笹尾　勤　正員　九州工業大学大学院情報工学研究院電子情報研究系
E-mail　sasao@ieee.org
Tsutomu SASAO, Member (Department of Computer Science and Electronics, Kyushu Institute of Technology, Iizuka-shi, 820-8502 Japan).
電子情報通信学会誌　Vol.96 No.2 pp.100-104 2013年2月

3. インデックス生成関数

TAC用インデックス生成器を一般化したものが，インデックス生成関数である．

［定義1］　k個の異なるn bit 2値ベクトルの集合を考える．これらのベクトルを登録ベクトルという．各登録ベクトルに対して，1からkまでの異なる整数を割り当てる．登録ベクトル表は，各登録ベクトルに対するインデックスを示す．インデックス生成関数は，入力が登録ベクトルにマッチするとき，それに対応するインデックスを生成する．また，マッチするベクトルが登録ベクトル中に存在しない場合，0を生成する．kをインデックス生成関数の重みという．インデックス生成関数は，写像$B^n\rightarrow\{0,1,2,\cdots,k\}$を表現する．インデックス生成関数を実現する回路をインデックス生成器という．

［例1］　表1は，k=4個のベクトルからなる登録ベクトル表を示す．これは重みk=4のインデックス生成関数を表す．

インデックス生成関数は，連想メモリ（CAM）で直

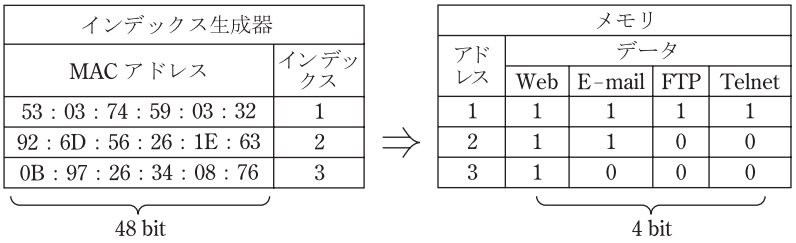
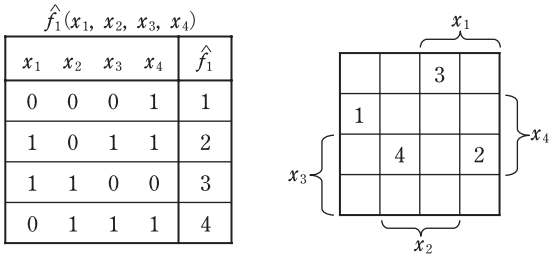


図1　端末アクセス制御回路（TAC）用のインデックス生成器とメモリ

表1　登録ベクトル表の例

ベクトル				インデックス
x_1	x_2	x_3	x_4	
0	0	1	0	1
0	1	1	1	2
1	1	0	0	3
1	1	1	1	4



(a) 登録ベクトル表　　(b) 分解表

図2　不完全定義インデックス生成関数での変数削減

接実現できるが，CAM の消費電力は大きい．したがって，本研究では，通常のメモリを用いて実現する．

4. 不完全定義インデックス生成関数

インデックス生成関数を実現する場合，不完全定義関数を用いると，変数の個数を削減できる場合が多い．

[定義 2]　$B=\{0,1\}$ とし，B^n 中の k 個の異なるベクトルの集合を $D=\{\boldsymbol{a}_1, \boldsymbol{a}_2, \cdots, \boldsymbol{a}_k\}$ とする．このとき，関数 $\hat{f}:B^n \rightarrow \{1, 2, \cdots, k, d\}$ を重み k の不完全定義インデックス生成関数という．ここで

$$\hat{f}(\boldsymbol{a}_i)=i, (\boldsymbol{a}_i \in D \text{ のとき}),$$
$$\hat{f}(\boldsymbol{b})=d, (\boldsymbol{b} \in B^n - D \text{ のとき}),$$

である．また，d はドントケア（未定義）を表す．

[例 2]　図 2(a)に示す登録ベクトル表を考える．これは，4 変数の不完全定義インデックス生成関数 $\hat{f}_1(X)$ を表す．$\hat{f}_1(X)$ を表現する分解表（カルノー図）を図 2(b)に示す．ここで，空白のセルはドントケアを表す．この分解表では各列で定義された要素がちょうど 1 個なので，ドントケアの値を列の定義された値と等しくすることにより，この関数は，次に示すように列変数のみで表現できる．

$$f_1=1\cdot\bar{x}_1\bar{x}_2 \vee 2\cdot x_1\bar{x}_2 \vee 3\cdot x_1x_2 \vee 4\cdot\bar{x}_1x_2.$$

この性質は，図 2(a)において，(x_1,x_2) 列の値が全て異なることから検出できる．

5. 変数の最小化

[例 3]　図 3 に示す 7 セグメントディスプレイは，七つのセグメント：a, b, c, d, e, f, g を用いて 10 進数の 1 桁を表示する．表 2 は，7 セグメントデータと対応する 2 進化 10 進符号（BCD）の関係を示す．これは，重み 10 の 7 変数インデックス生成関数と考えることもできる．率直な設計では，7 個の入力が必要である．しかし，10 進数を識別するには，図 4 に示すように 5 個のセグメントがあれば十分である．つまり，この関数は 5 変数で表現できる．

用　語　解　説

ルックアップ　検索データが表中に存在するか否かを高速に判定すること．ソフトウェアで実行する場合は，二分探索法を用いる．

パケットフィルタ　送られてきたパケットを通過させるか否かを判断する機能．ルータやファイアウォールが持っている機能の一つ．

連想メモリ　データワードを入力すると，全記憶内容を検索し，そのワードの見つかったアドレスを返すメモリ．コンピュータネットワーク機器のほか，CPU のキャッシュ制御部で使用する．

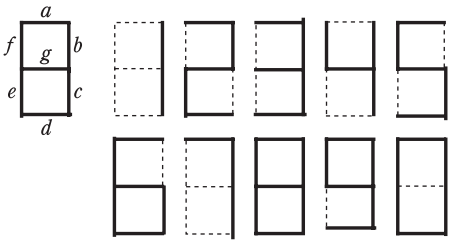


図3　7セグメントディスプレイ

表2　7セグメントBCD変換器

7セグメント							BCD符号				インデックス
a	b	c	d	e	f	g	8	4	2	1	
0	1	1	0	0	0	0	0	0	0	1	1
1	1	0	1	1	0	1	0	0	1	0	2
1	1	1	1	0	0	1	0	0	1	1	3
0	1	1	0	0	1	1	0	1	0	0	4
1	0	1	1	0	1	1	0	1	0	1	5
1	0	1	1	1	1	1	0	1	1	0	6
1	1	1	0	0	0	0	0	1	1	1	7
1	1	1	1	1	1	1	1	0	0	0	8
1	1	1	1	0	1	1	1	0	0	1	9
1	1	1	1	1	1	0	1	0	1	0	10

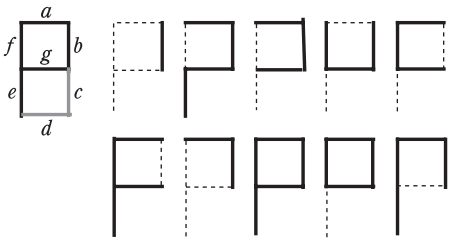


図4　5セグメントディスプレイ

6. ランダムなインデックス生成関数の性質

不完全定義インデックス生成関数の表現には，必ずしも全ての変数は必要ではなく，一部の変数のみで十分であることが多い．例3の7セグメントBCD変換器の場合，関数の表現に変数が5個あれば十分である．それでは，一般に，重み10の7変数インデックス生成関数が与えられた場合，関数の表現には変数が幾つ必要であろうか？　重み10の7変数インデックス生成関数は，全部で$\prod_{i=0}^{9}(128-i)=8.23\times10^{23}$個存在する．全ての関数に対して必要な変数の個数を網羅的に求めるのは不可能なので，ランダムに1,000個の関数を生成し統計的結果を求めた．1,000個の関数のうち，関数の表現に4変数必要な関数は519個，5変数必要な関数は459個，6変数必要な関数は22個であった．つまり，ほとんどの関数は，5変数以下で表現できた．したがって，例3の関数は，重み10の7変数インデックス生成関数としては，標準的なものと考えられる．重みkのインデックス生成関数を表現するために必要な変数の個数の下界は次の定理から容易に求まる．

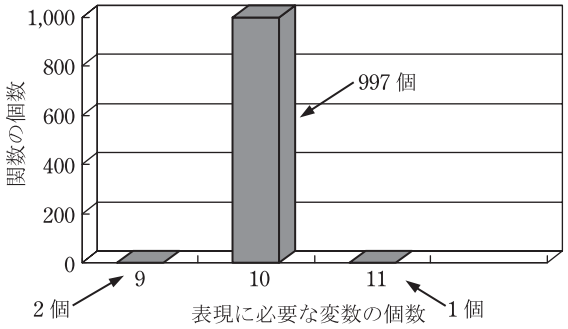


図5　重み127の20変数インデックス生成関数の表現に必要な変数の個数

［定理1］　重みkの不完全定義インデックス生成関数fを表現するためには，少なくとも$q=\lceil\log_2 k\rceil$個の変数が必要である．

次に，必要な変数の上界を考えてみよう．図5は，重み$k=127$の$n=20$変数不完全定義インデックス生成関数の統計を示す．1,000個のランダムな関数のうち，変数を9個必要とする関数が2個，変数を10個必要とする関数が997個，変数を11個必要とする関数は僅か1個である．分散は非常に小さい．ほとんどの関数に対しては，10個必要となっている．nとkの値が与えられたとき，必要な変数の個数が予想できれば，プログラマブル回路の設計に都合が良い．広範囲の実験を行った結果，ほかの組合せの場合にも，必要な変数の個数の分散は非常に小さいことが分かった．数式との格闘の結果[(2)]，次の美しい結果を得た．

［推論1］　均一に分布した重み$k\geqq7$の2値入力n変数不完全定義インデックス生成関数の集合において，$p=2\lceil\log_2(k+1)\rceil-3$変数で表現できる関数の割合は，$n$の増加につれて1.0に近づく．

不完全定義インデックス生成関数を表現するために必要な変数の個数は，関係$k\ll2^n$が成立するとき，登録ベクトルの個数kのみに依存し，ほとんどの場合，もとの関数の変数の個数nには，無関係である．

7. 線形変換を用いた変数削減法

推論1で述べたように，重みkの不完全定義インデックス生成関数のほとんどは，変数を高々$p=2\lceil\log_2(k+1)\rceil-3$個用いて表現できる．しかし，次のように例外も存在する．

表3　登録ベクトル表

登録ベクトル							インデックス	変換後		
x_7	x_6	x_5	x_4	x_3	x_2	x_1		y_3	y_2	y_1
0	0	0	0	0	0	1	1	0	0	1
0	0	0	0	0	1	0	2	0	1	0
0	0	0	0	1	0	0	3	0	1	1
0	0	0	1	0	0	0	4	1	0	0
0	0	1	0	0	0	0	5	1	0	1
0	1	0	0	0	0	0	6	1	1	0
1	0	0	0	0	0	0	7	1	1	1

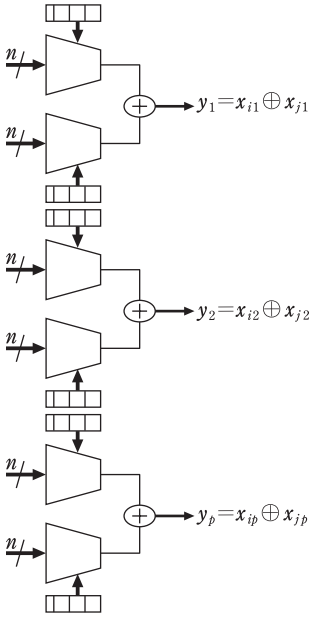


図6　二複合変数生成回路

［例4］　表3に示す登録ベクトル表を考える．これは，重み $k=7$ のインデックス生成関数を示す．この関数の表現に必要な変数の個数は6個まで削減できる．しかし，変数の個数を5個以下には削減できない．　■

例4の関数に対して，次に示す線形変換を適用することにより，変数の個数を削減できる．線形変換を用いると，ほとんど全ての関数は，推論1で与えられる個数の変数を用いて表現可能である．

［定義3］　関数 $f(x_1, x_2, \cdots, x_n)$ を考える．

$$y=c_1x_1\oplus c_2x_2\oplus\cdots\oplus c_nx_n,$$

の形をした変数を複合変数という．ここで $c_i\in\{0,1\}$ である．$\sum_{i=1}^{n} c_i$ の値を変数 y の複合度という．複合度1の変数を原始変数，複合度 t の変数を t 複合変数という．

図6に，二複合変数を生成する回路を示す．この回路は，線形変換 $y_i=x_i\oplus x_j$ あるいは $y_i=x_i$ を行う．ここで $i\neq j$ である．この回路では，各変数 y_i に対して，一対のマルチプレクサを用いる．上部のマルチプレクサには入力 $x_1, x_2, \cdots, x_n$ が接続されており，$\lceil\log_2 n\rceil$ bit のレジスタが，選択すべき変数を指定する．下部のマルチプレクサには x_i 以外の入力 $x_1, x_2, \cdots, x_n$ が接続されている．i 番目の入力には，x_i の代わりに定数0が接続されている．$y_i=x_i\oplus 0$ と設定することにより，原始変数 $y_i=x_i$ も生成できる．同様にして，三複合変数生成回路等も構成できる．

［定義4］　不完全定義インデックス生成関数が与えられたとき，最適な線形変換とは，その関数を表現するために必要な変数の個数を最小にする変換である．

変数の個数を $q=\lceil\log_2 k\rceil$ 個まで削減するような線形変換は，定理1から，最適である．

［例5］　表3の関数を表現するために，次の線形変換を考える．

$$y_1=x_1\oplus x_3\oplus x_5\oplus x_7,$$
$$y_2=x_2\oplus x_3\oplus x_6\oplus x_7,$$
$$y_3=x_4\oplus x_5\oplus x_6\oplus x_7.$$

このとき，表3の右端に示す変換した登録ベクトルを得る．この場合，パターンが全て異なるので，7個のベクトルは三つの複合変数 (y_3, y_2, y_1) で識別できる．$\lceil\log_2 k\rceil=3$ なので，この変換は最適である．　■

8. プログラマブル回路とその応用

図7にインデックス生成ユニット（IGU）を示す．線形回路は n 入力 p 出力（$p<n$）であり，複合変数を実現する．これにより，主メモリの変数の個数を削減する．X の変数を $X_1=(x_1, x_2, \cdots, x_p)$ と $X_2=(x_{p+1}, x_{p+2}, \cdots, x_n)$ に分割する．主メモリの入力数は p で，出力数は $q=\lceil\log_2(k+1)\rceil$ である．主メモリは，登録ベクトルに対しては，正しいインデックスを生成する．しかし，入力変数の個数が削減されているので，未登録ベクトルに対しては，誤ったインデックスを生成する場合がある．インデックス生成関数において，入力ベクトルが未登録の場合，出力は00…00とすべきである．主メモリが生成するインデックスが正しいか否かを検査するために補助メモリ（AUXメモリ）を用いる．補助メモリの入力数は $q=\lceil\log_2(k+1)\rceil$ で，出力数は $n-p$ である．各インデックスに対して，登録ベクトルの X_2 の部分を格納する．比較器は，入力ベクトルの X_2 の部分が登録ベクトルの X_2 の部分と同じか否かを判定する．二つのベクトルが

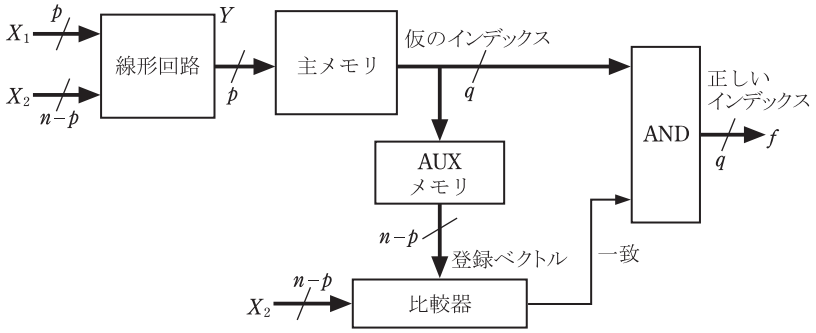


図7　インデックス生成ユニット

等しいとき，主メモリは正しいインデックスを生成している．異なる場合は，主メモリは誤ったインデックスを生成しており，入力ベクトルは，未登録である．この場合，出力の AND ゲートは，入力ベクトルが未登録であることを示す 00…00 を出力する．主メモリは，登録ベクトルに対してのみ正しいインデックスを生成する．このようにして，図 7 に示す IGU は完全定義関数を実現する．IGU を複数個用いることにより，インデックス生成関数を能率良く実現できる．本手法を用いてコンピュータウイルスを 120 万パターン以上格納したウイルス検出システムを FPGA ボード上に開発した[3]．

9. お　わ　り　に

論理合成の研究は，二段論理合成，多段論理合成，FPGA 設計と変化し，現在では，成熟期に入っており，大きな進展は困難となっている．しかし，応用分野を限定することにより，新たな研究の展開ができる．例えば，通信用のパターンマッチングでは，動作中に回路を書き換える必要があるため，時間のかかる従来の論理合成法や，配線の変更が必要な FPGA の手法は使えない．動作中に再構成可能にするため，メモリを基本とした新しいアーキテクチャが必要となる．

プログラマブルなアーキテクチャは，現在までに多数提案されているが，自動設計システムを整備できなかったものは消滅している．現在，主流となっている FPGA は，アーキテクチャも筋が良く，合成アルゴリズムも十分開発されている．大学教育にも使用されており，長い歴史の中で既に大きな「文化」を形成している．汎用回路で，このアーキテクチャを凌駕するものを新たに開発するのは困難であろう．しかし，アプリケーションを限定すると，まだ可能性は残っている[2],[4]～[6]．

謝辞　本研究は，知的クラスタ創成事業（第 1～2 期），及び科学研究費補助金の助成による．共同研究では，北九州産業学術推進機構，福岡県産業・科学技術振興財団，日立超 LSI システムズ，ヤマハ，ルネサスエレクトロニクスの皆様にお世話になった．最後に，永年苦労を共にしてくれた J.T. Butler 氏，井口幸洋氏，永山忍氏，中原啓貴氏，松浦宗寛氏に深謝する．

文　　献

（1） T. Sasao, “Survey of research projects conducted by Sasao’s group (FY2004-FY2011),” http://www.lsi-cad.com/sasao/Papers/pub2012.html
（2） T. Sasao, Memory-Based Logic Synthesis, Springer, March 2011.
（3） H. Nakahara, T. Sasao, and M. Matsuura, “A low-cost and high-performance virus scanning engine using a binary CAM emulator and an MPU,” Reconfigurable computing : architectures, tools and applications : 8th International Symposium, ARC 2012, Hong-Kong, March 2012 (Lect. Notes Comput. Sci., vol. 7199, pp. 202-214, March 2012.)
（4） T. Sasao, “Analysis and synthesis of weighted-sum functions,” IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst., vol. 25, no. 5, pp. 789-796, May 2006.
（5） T. Sasao, S. Nagayama, and J.T. Butler, “Numerical function generators using LUT cascades,” IEEE Trans. Comput., vol. 56, no. 6, pp. 826-838, June 2007.
（6） T. Sasao, “Row-shift decompositions for index generation functions,” Design, Automation & Test in Europe (DATE-2012), pp. 1585-1590, Dresden, Germany, March 2012.

（平成 24 年 5 月 24 日受付　平成 24 年 8 月 29 日最終受付）

笹尾　勤（ささお　つとむ）（正員）
1972 阪大・工・電子卒．1977 同大学院博士課程了．同年同大学・工・助手．1988 九工大・情報・助教授．1993 同教授，現在に至る．著書「論理設計：スイッチング回路理論」（近代科学社，2005），「Switching Thoery for Logic Synthesis」（Kluwer，1999），「Memory-Based Logic Synthesis」（Springer，2011），ほか 8 冊．IEEE フェロー．