FPGAを用いて事例データを直接回路化したパターン認識用集積回路

安永 守利　　高見 知親

筑波大学　電子・情報工学系
〒305-8573 茨城県つくば市天王台 1-1-1
TEL: 0298-53-5323
e-mail: yasunaga@is.tsukuba.ac.jp

あらまし　事例データ（既知パターンデータ）を真理値表に変換し，これを基本ゲートに展開することでパターン認識用集積回路を設計する手法を提案する．遺伝的アルゴリズムを適用することで，真理値表が未知パターンとも正しく一致するように適応変化（進化）させる．近年急速に普及しつつある高集積 FPGA を用いることで，実応用レベルの問題の個別事例データを直接集積回路化することが可能になると考える．英単語の発音記号推論チップの開発に本手法を適用し，プロトタイプを試作した．この結果，推論速度 500ns／音素，回路数 139K ゲート，認識率 71.5%を得た．

キーワード　FPGA，真理値表，パターン認識，事例データ，遺伝的アルゴリズム，発音記号

Pattern Recognition LSI's Designed by Direct Data Implementation Technique Using FPGA's

Moritoshi Yasunaga and Tomochika Takami

Institute of Information Sciences and Electronics
University of Tsukuba, Tsukuba, Ibaraki 305-3573, Japan
TEL: +81-298-53-5323
e-mail: yasunaga@is.tsukuba.ac.jp

Abstract　We proposed a design methodology to implement case data directly onto FPGA chips. Case data (pattern data) are converted into truth tables, and the tables are evolved to match unknown pattern data by using genetic algorithm. The evolved truth tables are simply implemented onto FPGA chips. By using the current high density FPGA chips, specialized pattern recognition chips for individual practical tasks can be rapidly developed at user-sites. We applied this methodology to develop an English pronunciation reasoning chip. The chip was designed with 139K gates and its prototype showed a reasoning speed of 500 ns/phoneme and a reasoning accuracy rate of 71.5%.

key words　FPGA, Truth table, Pattern recognition, Case data, Genetic Algorithm, Phoneme

1．はじめに

従来のパターン認識は，既知パターンデータと未知パターンデータの距離計算を基本とする[1]．両者の距離を逐次計算し，その統計処理結果から未知パターンの属するカテゴリを推論する．このため，パターン数やカテゴリ数が増加すると多大な計算時間が必要となるので，複雑なパターンの実時間認識や高速認識が不可能となる．この問題は，画像パターンや時系列パターンデータに限らず，知識処理における多くの推論問題にも当てはまる[2]-[5]．

その高速化には，距離計算や統計計算を高速化するためのハードウェア（アクセラレータ）が必要である．具体的には，信号処理プロセッサ等の積和演算器を中心としたプロセッサを複数個用いて並列処理を行う必要がある．しかし，このようなプロセッサを基本としたハードウェア(Processor-Based Architecture)はハードウェア量が多く，わずかな並列化（４～１６個のプロセッサ）でも１チップ化が不可能である．このため，実時間処理や高速処理の実現のためには，専用設計を行ったとしても大規模なデスクサイド筐体が必要なることが多い．

本稿では，このような従来手法による Processor-Based Architecture とは全く異なるハードウェアアーキテクチャとその設計手法を提案する．具体的には，既知データ（過去の事例データ）を基に作成された真理値表から，認識用の回路を直接生成する．ここで，真理値表そのものは未知データを認識することはできない．そこで，真理値表に遺伝的アルゴリズム[6]を適用し，未知データにも反応するように真理値表を変化（進化）させる．このようにして，それぞれの既知データから専用の認識回路を直接生成する．この手法を LoDETT (Logic Design with Evolved Truth Table) と呼ぶ（図１）．遺伝的アルゴリズムの適用方法，進化の手法の詳細は次節以降で述べる．

このような，各事例データの直接回路化（専用集積回路化）手法は，従来の再構成不可能な集積回路や，再構成可能であっても集積度の低い集積回路には適用できなかった．しかし今日，デバイスの微細化に伴って実応用問題のデータを直接回路化できるだけの集積規模を FPGA が提供できるようになった（図２）．高集積 FPGA を用いることによって，各問題に最適化された（少ない回路数で最も高い並列性が実現できる）専用集積回路を，各ユーザサイトで自由に実現できると考える．

本稿では，LoDETT の基本的手法を述べる．次に，LoDETT を用いた開発事例とし「英単語の発音記号推論チップ」を示す．英単語の発音記号推論問題は文字列のパターン認識の一つであり，高認識率を得ることと高速処理が困難な実応用問題の一つである．従来より，並列人工知能やニューラルネットワークによる解決が試みられてきた[2][7]．

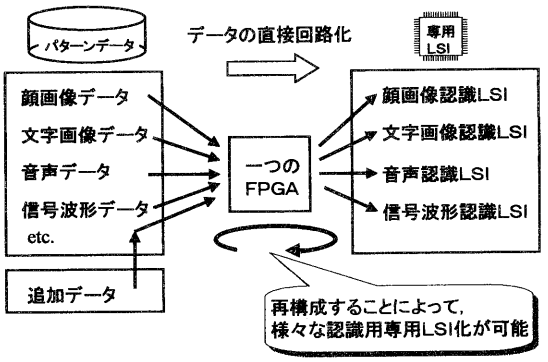

図1 LoDETTの基本的な考え方

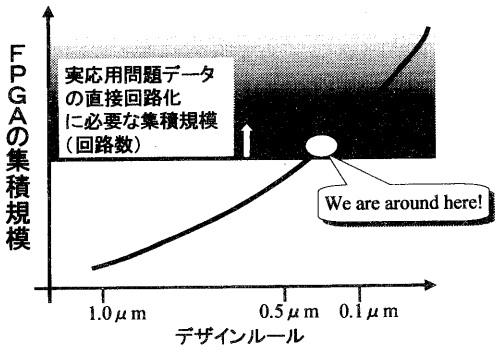

図2 データの直接回路化と FPGAの集積規模の予測

2．真理値表の進化手法

図３に，我々が提案する LoDETT の基本的なフローを示す．パターンデータ（カテゴリがわかっている事例データ）を各カテゴリの１パターン

ごとにバイナリーコードによって符号化し，真理値表を作成する．作成した真理値表には，そのままでは未知データと一致する行は一つもない．そこで，真理値表の作成に使用しなかった既知データ（学習データ）を用いて，これらの学習データも真理値表のいくつかの行と一致するように don't care (*) との置換を行う．ここで，より多くの学習データが正しく一致する（同一のカテゴリの行と一致する）ように，don't care の位置を遺伝的アルゴリズムによって決定する．ここで，学習データを用いる進化過程の中で，真理値表は未知データに対しても正しく一致するように進化する．すなわち，真理値表は，進化の過程で未知データを認識する汎化能力を獲得する．進化後の真理値表は，基本ゲート展開によって容易に回路化すること（専用集積回路化）が可能である．

図４，図５に真理値表の進化手法を用いたパターン認識用集積回路の基本的な考え方を示す．本手法では，はじめに以下の処理を行う：

1）カテゴリ（'風邪' や '虫歯'）が既にわかっているパターンデータを真理値表にコーディングする．
2）真理値表の各行に対応するＡＮＤゲートを作成し，各カテゴリ毎にＯＲゲートに接続する（積和標準形によるＡＮＤ－ＯＲゲート展開）．

この段階（図４）で，カテゴリがわからない未知パターン（問題）をＡＮＤゲートに入力した場合，未知パターンは，基となる真理値表のどの行とも一致していないため，どのカテゴリのＯＲゲートの出力も '０'（偽）である．すなわち，認識できなかったこととなる．

ここで，基となる真理値表中の値の一部が don't care になった場合（図５中の＊），未知パターンと一致する行が現れることがある．この場合，対応するＡＮＤゲートは活性化され，対応するカテゴリのＯＲゲートの出力は '１'（真）となり，未知パターンは，認識されたことになる（ここで，未知パターンは「認識された」だけであって，その結果が正しいとはかぎらないことに注意されたい）．

真理値表の適切なフィールドを don't care に置換することによって，より多くの未知パターンを正しく認識するようになる．本稿では，この置換処理に遺伝的アルゴリズムを用いる．すなわち，

3）遺伝的アルゴリズムによって真理値表を進化させる．

遺伝的アルゴリズムは，最適化問題解法アルゴリズムの一つであり，生体の遺伝子が環境適応の中で最適解をみつける過程（進化過程）をモデル化している[6]．

遺伝的アルゴリズムは，１）セレクション，２）交叉，３）突然変異の３つの基本処理からなる．図６に，LoDETT における基本的な染色体構造を示す．各カテゴリ（例えば，'風邪'）ごとに N 個の染色体集団（人口）を構成する．各染色体は，

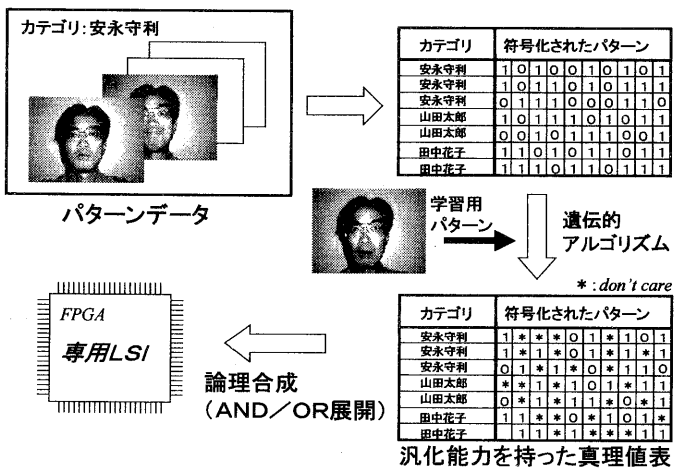

図3 LoDETTの基本的なフロー

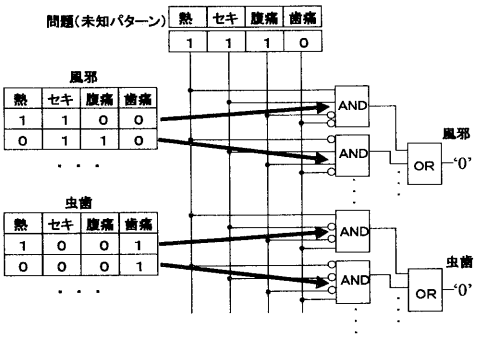

図4 パターンデータ真理値表の回路化

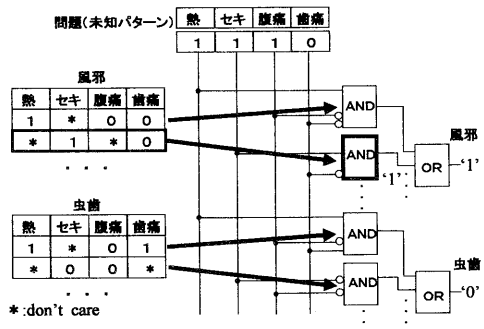

図5 進化後のパターンデータ真理値表の回路化

真理値表の各カテゴリの行数と同数の空行から構成される．空行のいくつかのフィールドには，初期値として don't care を配置する．各フィールドが空か don't care かが染色体の遺伝情報となる．その後，各染色体間でフィールド間の遺伝情報の交換（交叉）を行う．また，ある割合で，遺伝情報をフリップ（空なら don't care，don't care なら空への置換）を行う（突然変異）．交叉処理と突然変異処理の後，各染色体の fitness（適合率）を評価し，N 個の染色体の中で fitness の低い方から M 個の染色体を上位 M 個の染色体と交換する（セレクション）．この交叉，突然変異，セレクションを 1 世代として，1,000 ～ 10,000 世代の進化処理を行う．なお，fitness は，各問題毎に定義する必要がある．具体的な例は，後述の開発事例の中で示す．

3．開発事例（英単語発音記号推論チップ）

英単語の発音記号推論チップの開発事例を示す．英単語の発音記号（音素）の推論問題は，文字と

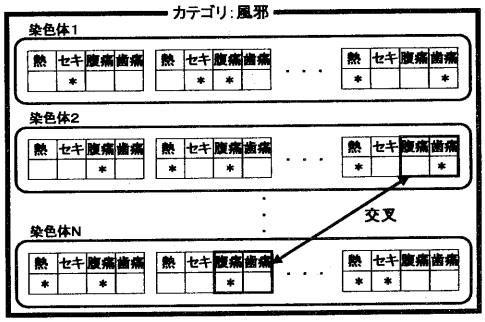

図6 真理値表を進化させるための染色体構造

音素の対応関係規則の発見が困難なため，従来のＡＩ的なアプローチ（プロダクションルール等）が適応できない．このため文字列パターンの認識問題として，ニューラルネットワークや並列人工知能によるアプローチが試みられてきた．

ニューラルネットワークのバックプロパゲーションを用いた英単語発音記号推論は，NETTalk と呼ばれる [7]．NETTalk では，ニューラルネットワークが学習によって規則を自動抽出する．NETTalk の学習能力の高さは 1980 年代に多くの注目を集め，ニューラルネットワーク研究･開発勃興の一つの原因となった．一方，NETTalk の問題点は，その学習に非常に多くの計算時間を必要とすることである．当時のミニコンで，１～２週間の計算時間を要した．それ以来，学習時間を高速化する目的で多くのニューロコンピュータが研究･開発されており，NETTalk は事実上のベンチマークとして用いられている [8]．

並列人工知能の分野では，超並列計算機を用いた推論アルゴリズムとして記憶ベース推論（Memory-Based Reasoning）が開発されている[2]．MBR による英単語の発音記号推論は MBRTalk と呼ばれる．MBR は，既知データと未知データの統計的な重み付け距離から推論を行うシステムである．MBRTalk は高い性能を示すものの，このような複雑な距離計算を各単語の各文字ごとに行う必要がある．このため，ニューラルネットワークの学習と同様に計算量が非常に多く，高速推論（実時間処理）には，超並列計算機のような大規模システムが必要となる（MBRTalk の開発には，超並列計算機 CM2 [9]が使用された）．

上述のように，英単語の発音記号推論問題は実応用問題であり，大規模ハードウェアを必要とす

る．このような理由から，我々はこの問題をLoDETT の評価に用いた．開発した集積回路をGATalk チップと呼ぶ．

GATalk チップの開発にあたり，辞書から図７に示すレコードの切り出しを行った．例えば，pizza[pIt!@] という英単語とその発音記号から，シフト処理によって７文字のレコードを切り出す．発音記号は７文字の中心文字に対応する．この例では，５つのレコードが生成される．各レコードの属するカテゴリは，中心文字の発音記号（音素）となる．なお，このレコードの切り出しとその分類方法は，NETTalk と MBRTalk で用いられた．LoDETT でも，これと同様なレコードを用い，１レコードが１事例となる．すなわち，１レコードが真理値の１行となる．本開発で用いた辞書は，NETTalk の開発者である Sejnwski によって提供されている辞書を用いた．

pizza [pIt!@]

レコード	音素（発音記号）
- - - p i z z	p
- - p i z z a	I
- p i z z a -	t
p i z z a - -	!
i z z a - - -	@

図7 英単語レコードの切り出し

切り出されたレコードの符号化を図８に示す．１文字を 27 フィールド（26 文字+スペース）の１つで表すことで符号化した．従って，１レコード（７文字）は 189 フィールドとなる．この 189 フィールドが真理値表の １ 行である．このようにして切り出したレコードから真理値表への変換を，各カテゴリ（発音記号（音素））毎に行う（図９）．なお，カテゴリの総数（音素総数）は 52 である．

使用した辞書は 20,008 語から構成されている．この中から真理値表作成用，学習用，そして認識率（推論正解率）評価用にそれぞれ 1,000 語をランダムに選んだ．真理値作成用の 1,000 語から生成されたレコードは 7,431 個であった．従って，真理値表の総行数も 7,431 である．

各カテゴリ毎に前述した染色体集団を作成し，遺伝的アルゴリズムを適用した．染色体数 N は，

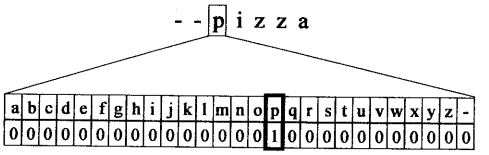

図8 英単語レコードの各文字の符号化

各カテゴリ毎に 20 とした．交叉確率と突然変異確率は各遺伝子毎に 0.1%とした．また，fitness には，

$$f(i)=\frac{n_i}{N_i}\times\left(1+\frac{m_i}{M_i}\right)\times\frac{1}{2}$$

を用いた．ここで，N_i は，カテゴリ i に属するパターンの総数であり，n_i はその中で正しく認識された（一致した）パターンの総数である．また，M_i は i に属さないパターンの総数であり，m_i はその中で正しく認識された（一致しなかった）パターン数である．このような環境の基で 10,000 世代まで進化を繰り返した．進化処理後，各真理値表毎に論理合成を行った．

GATalk チップ全体のブロック図を図１０に示す．各カテゴリ毎の回路ブロックは並列に配置され，未知パターンは，各カテゴリブロックにブロードキャストされる．ここで，図４と図５で示したように，ＡＮＤゲートの出力はＯＲゲートによってまとめることで基本的には認識が可能である．しかし，OR ゲートでまとめる構造では，一つでも

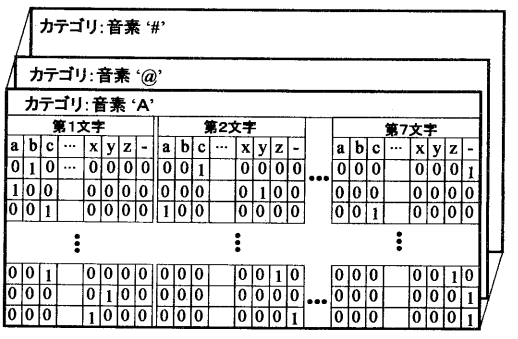

図9 音素予測(認識)のための真理値表の構造

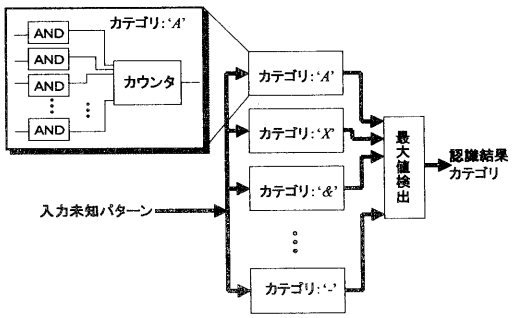

図10 GATalkチップのブロック図

ＡＮＤゲートが活性化された場合，ＯＲゲートの出力は‘真’となる．GATalk やその他の実応用問題の場合，レコード数が非常に多く，また，カテゴリ数も多い．このため，誤ったカテゴリの中のＡＮＤゲートも活性化される確率が高くなる．このため，複数個のＯＲゲートが活性化されるケースが多くなり，図４，図５のままでは「認識不可能」となる割合が高い．

そこで，実際は図１０に示すとおり，各カテゴリ毎に活性化されたＡＮＤゲートの数をカウントするカウンタ回路を設けた．カウンタ回路のカウント値の最も大きいカテゴリを最大値検出回路によって検出し，そのカテゴリを認識結果とする．これは，未知パターンとの一致パターン数が最も多いカテゴリを最もた確からしいとする k-NN 法によるパターン認識と同様である[1].

真理値表を FPGA に実装するまでに，実際には２つの市販ＣＡＤを用いた（図１１）．進化後の真理値表の論理合成には，(株) 図研の Vps を用いた．Vps によって，Xilinx 社の FPGA である XC4010 のセルライブラリを用いたネットファイルを作成する（テクノロジマッピングまでを行う）．作成したネットファイルを Xilinx 社の XACT に入力し，配置配線とコンフィギュレーションデータ作成を行った．

４．評価結果

論理合成後のゲート数は，２入力ＮＡＮＤゲート換算で 138,194 ゲートであった．これは，現在の高集積 FPGA を用いれば，1 チップに十分実装可能な回路規模である．本開発では，コンフィギュレーションデータを図１２に示す評価ボード（Aptix 社製 MP3）に実装した．本評価ボードには，Xilinx 社製の FPGA XC4010 が 12 個と XC4005 が 2 個搭載されている．さらに，各 FPGA 間の配線を再構成するための FPGA である FPIC チップが 3 個搭載されている．これにより，ボード全体が 12 万ゲート相当の FPGA チップと同様に再構成可能である．今回行った 1,000 単語を基とした GATalk チップの総ゲート数は上述のように約 14 万ゲートであるため，本評価ボードに全てを実装することはできない．このため全体論理を 3 分割して実装し，それぞれの分割出力結果をオフライン（PC 上）で評価し，その論理動作を確認した．

GATalk チップの性能評価結果を表１に示す．GATalkチップの１音素当りの認識速度500 nsは，前述した評価ボード上での実行速度である．従って，1 チップ化できた場合はさらに高速化が可能であると考えられる． いずれにせよ，ns オーダでの認識（推論）が可能であり，これは英文テキストの会話速度での音素推論に要求される速度（ms オーダ）よりはるかに高速である．

比較評価のために，NETTalk, MBRTalk の認識率の結果も示す（表１）．NETTalk と MBRTalk は，パーソナルコンピュータ上で動作させた．評

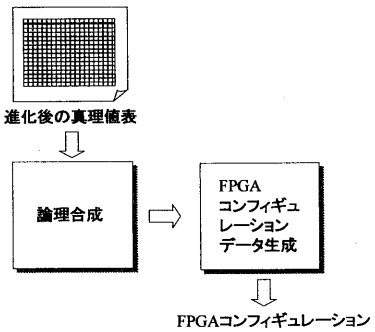

図11 FPGA実装の手順

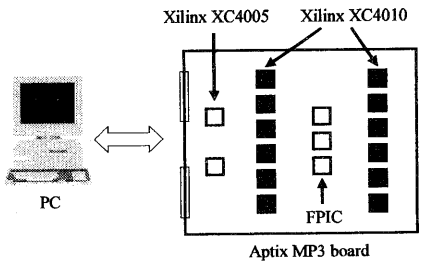

図12 GATalkチップのプロトタイプ

価には，全て同じデータを使用した．認識正解率は，GATalk チップが 7%～9%低い値となった．ネイティブスピーカによる聞き取り実験によれば，MBRTalk の結果 (78.6%)でも十分自然な音声合成が可能であるとの結果が報告されている[2]．GATalk チップは，従来技術に比べて認識率が 7%～9%低いものの，20 万ゲート程度の FPGA チップであれば十分 1 チップ実装が可能である．この観点から，GATalk チップは実用可能な性能（総合性能）を有していると考えられる．

表1 GATalkチップの性能

平均認識(推論)速度: 500 ns／音素
ゲート数: 138,194 ゲート (辞書単語数1,000)
認識(予測)正解率: 71.5% ニューラルネットワーク (NETTalk): 80.5 % 並列AI (MBRTalk): 78.6 %

5・ まとめ

過去の事例データ（既知データ）を直接回路化することで専用パターン認識チップを設計する手法を提案した．既知データから未知データを認識できる回路を生成するために，遺伝的アルゴリズムを用いる．英単語の発音記号（音素）推論問題を対象に本手法を適用した．その結果，推論速度 500ns，回路数約 14 万ゲート，認識率 71.5%のチップを設計することができた．これは，実用可能な性能であると考えられる．

謝辞

本研究の一部は平成１０年度文部省科学研究補助金による．関係各位に感謝いたします．

参考文献

[1] K. Fukunaga, "Statistical Pattern Recognition," Accademic Press, 1990.

[2] C.Stanfill and D.Waltz,"Toward Memory-Based Reasoning," Communications of the ACM, Vol.29, No.12, pp.1213-1228,1986.

[3] X.Zhang, D.Waltz, and J.Mesirov,"Protein Structure Prediction by Memory-based Reasoning," Thinking Machine Corporation, 1988.

[4] R.Creecy, B.Masand, S.Smith, and D.Waltz, "Trading MIPS and Memory for Knowledge Engineering: Automatic Classification of Census Returns on a Massively Parallel Supercomputer," Thinking Machine Corporation, 1990.

[5] H.Kitano and T.Higuchi, "Massively Parallel Memory-Based Parsing," Proc. 11th Int. Joint Conf. Artificial Intelligence, 1991.

[6] D.E.Goldberg,"Genetic Algorithms in Search, Optimization, and Machine Learning," Addison-Wesley Publishing Co., 1989.

[7] T.J.Sejnowski and C.R.Rosenberg, "Parallel Networks That Learn to Pronounce English Text, Complex Systems, Vol. 1, pp.145-168,1987.

[8] S.B.Nikola, "Simulating Artificial Neural Networks on Parallel Architectures, "IEEE COMPUTER, Vol.29, No.3, pp.56-63, 1996.

[9] D. H. Hillis, "The Connection Machine", MIT Press, 1985.