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

2007－SLDM－130（5）
2007／5／10

大語彙連続音声認識用出力確率計算回路アーキテクチャの一検討

橋本 丈† 才辻 誠†† 神戸 尚志†

† 近畿大学 理工学部 電気電子工学科
†† 近畿大学 大学院 総合理工学研究科
〒 577-8502 東大阪市小若江 3-4-1
E-mail: ††tkambe@ele.kindai.ac.jp

あらまし 大語彙連続音声認識エンジンをモバイル端末に搭載することを目的とし、C 言語を拡張したハードウェア記述言語である Bach-C を用い、高速化アーキテクチャの検討を行った。連続音声認識では、入力音声を一定幅のフレーム単位で切り出し、HMM という時系列信号の確率モデルを用いて出力確率計算が行われ、処理時間全体の約 40％を占める。出力確率計算を高速に行うため、パイプライン処理、並列処理、辞書メモリの分割やメモリアクセス用バッファの利用、ループ処理展開などを組みあわせたアーキテクチャを設計した。その内容と性能比較について報告する。

キーワード C 言語設計, Bach システム, 音声認識, 並列処理, パイプライン処理, 出力確率計算

An Architecture Design and its Evaluation for Speech Recognition System

Joe HASHIMOTO†, Makoto SAITSUJI††, and Takashi KAMBE†

† Department of Electrical and Electric Engineering, Kinki University
†† Graduate School of Science and Technology, Kinki University
3-4-1, Kowakae, Higashi-Osaka city, Osaka 577-8502, JAPAN
E-mail: ††tkambe@ele.kindai.ac.jp

Abstract Speech recognition is becoming a popular technology for the implementation of human interfaces. However, conventional approaches to large vocabulary continuous speech recognition require a high performance CPU. In this paper, we describe a speech-recognition system designed using a C-based design methodology and compare several hardware implementations for the computationally intensive parts. Pipelining, parallel processing, buffer memory solution, and for loop unrolling to compute the Hidden Markov Model (HMM) output probability at high speed were implemented and their performances evaluated. It is shown that designers can rapidly explore a wide range of complex circuits using this methodology and that real time speech recognition in small portable systems is possible.

Key words C based design, Bach system, Speech recognition, parallel processing, Pipelining, Output probability computation

1. はじめに

音声認識技術は、キーボードやマウスより便利で自然な形での操作が可能な入力インターフェイスとして期待されている [1], [4]。近年、半導体技術の進歩により音声認識技術が実用化され、携帯電話やカーナビゲーションのボイスサーチ・コマンドや、コールセンター等の自動応答受付などに音声認識が使われ始めている。これらは、主に決められた単語に対して認識を行っており、文章による入力には対応していない。文章による音声認識を特に連続音声認識、または大語彙連続音声認識と呼ばれ、現在、パソコン上で動作するソフトウェアとして実現されている。連続音声認識では、HMM（隠れマルコフモデル）による認識が、基幹技術として使われている。HMM を用いた音声認識は、認識性能が高い反面、出力確率を求める計算量が多い。そのため、ソフトウェアでリアルタイムの音声認識を行うには高い処理能力を持つ CPU が必要となる。

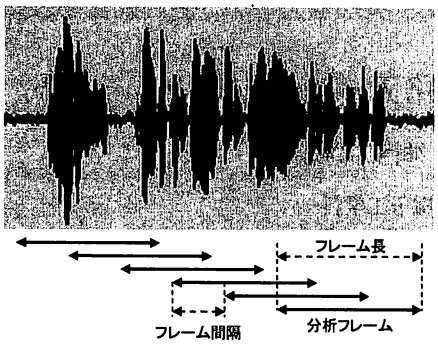

図 1　フレーム分析
Fig. 1　Frame analysis

音声認識のハードウエア化についての研究は、これまで幾つか行われている [2], [3] が、認識する語や文を制限することで実現している。これに対し、我々は、大語彙連続音声認識、すなわち数万語の語彙を対照とした連続する文章の自動認識を対象としている。

本研究は、大語彙連続音声認識エンジン「Julius」[5] をもとにハードウェアとソフトウェアで構成するシステムを設計し、システムの効率化、小規模化を目指す。具体的には、多くの計算を要する出力確率計算部分をハードウェア化することでシステム全体を高速化する手法を提案する。ハードウェア化には、従来の HDL によるハードウェア設計より抽象度の高い設計が行なうことが出来る Bach システム [8] を用いて設計した。Bach システムは、ハードウェアの並列動作等を陽に記述出来るよう、ANSI-C を拡張した Bach-C 言語を入力とし、動作合成により RT レベル HDL 記述を自動生成する。出力確率計算部分をハードウェア化するにあたり、並列処理、パイプライン処理、プラグマ unroll、バッファの利用、メモリ分割などを組み込んだ回路を設計し、RTL シミュレーションで性能評価を行った。

本論文の構成は、2. 章で音声認識技術、3. 章で出力確率計算部分のハードウェア設計、4. 章で出力確率計算部分の高速化手法について、6. 章でまとめと今後の課題について述べる。

2.　音声認識技術

2.1　音響分析

音声は、時間的に特徴量の変化が激しいので、図 1 に示すように一定の時間幅 (フレーム長) で切り出し、定常確率過程に従うと仮定し、スペクトル解析を行い音響特徴量算出する。その際に、切り出されたフレームの両端において不連続が起こりやすいため、フレーム区間の半分程度の長さで重ね合わせながらフレームをシフトさせながら解析する。

Julius は、フレーム長 25ms、フレーム間隔 10ms としてフレーム化処理を行っている。フレーム化された入力音声から抽出された音響特徴量の時系列を生成するモデルとして HMM (隠れマルコフモデル) を用いて出力確率計算を行う。

2.2　HMM の出力確率計算

フレーム単位の音響特徴量は、p 次元ベクトルで表現でき、音響特徴量は複数のガウス分布を組み合わせた混合ガウス分布で表現される。混合ガウス分布による出力確率は以下の計算で求める。p 次元、t 番目フレームの入力ベクトル値を o_{tp} とし、i 状態 m 混合目のガウス分布計算 b_{im} は以下の式で与えられる。

$$\log b_{im}(o_i) = \omega_i - \frac{1}{2}\sum_{p=1}^{P}\frac{1}{\sigma_{imp}^2}(o_{tp} - \mu_{imp})^2 \qquad (1)$$

ここで、混合重み値を ω_i、平均ベクトル値を μ_{imp}、分散ベクトル値を σ_{imp}^2 とする。これらの値は、HMM の学習時に計算により求められる。M 混合ガウス分布の出力確率計算値 b_i は、以下の式で与えられる。

$$\log b_i(o_i) = \log\sum_{m=1}^{M}\exp(b_m) \qquad (2)$$

ソフトウェアに組み込む場合、このままでは計算量が多くなることや、オーバーフローが生じやすいため、b_{im},b_{im+1} を x,y(x > y) とする時、以下の近似式で表される。

$$\log(\exp(x) + \exp(y)) \approx x + \log(1 + \exp(y - x)) \qquad (3)$$

2.3　大語彙連続音声認識

大語彙連続音声認識技術は、数万語の語彙を対照とした文章認識であるため、その探索空間は膨大となり、音響モデル、木構造化辞書と言語モデルを組み合わせた認識方法を用いる。音声認識エンジンである Julius の探索アルゴリズムでは、図 2 に示すように、第 1 パスと第 2 パスのマルチパス探索を行う。第 1 パスでは、音素 HMM による出力確率値とバイグラムモデルから、単語、文の候補を絞り込み、第 2 パスでは、第 1 パスで得た候補を、トライグラムモデルを用いて再検索・再評価を行う。本研究では、95 %の認識精度を誇る Julius の高精度版を用いてハードウエア化の検討を行う。

3.　ハードウエア設計

3.1　音声認識システムの分析

一般に、各種システムにおいて、ハードウェア化は高速化、低消費電力化のために行われ、ソフトウェアは柔軟性を持たせて処理を行いたい時に用いられる。ソフトウェアをハードウェア化するには、構造によって適切な部分が異なる。ソフトウェア処理の中で、処理が独立している部分をハードウェア化することにより、全体の処理を大きく変更することなくハードウェアとソフトウェアを組み合わせてシステムを実現することが出来る。Julius のプログラム構造の解析、処理時間の分析結果 (図 3) より、第 1 パス中で全体の処理時間の約 64 %を占めている出力確率計算に着目しハードウェア化を行う。また、Julius では、フレーム間隔が 10ms のため、リアルタイム認識には 1 フレームの処理を 10ms 以内に終わらせる必要がある。

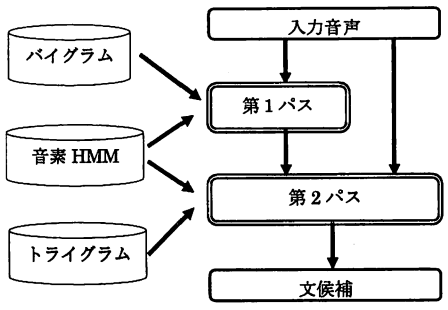

図 2　マルチパス検索
Fig. 2　Multi pass search

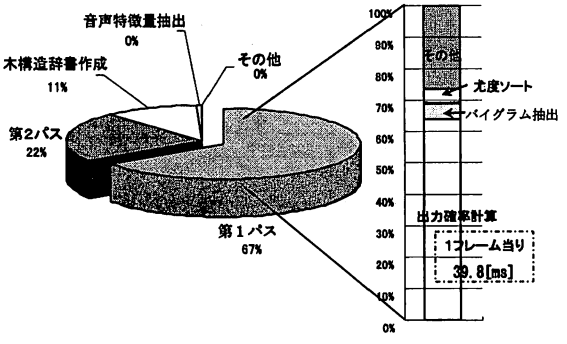

図 3　Julius の処理時間分析
Fig. 3　Processing time analysis

3.2　Bach を用いた設計工程

ハードウェアの設計手法は、主にレジスタ転送レベルのハードウェア設計言語の VHDL、Verilog-HDL を用いた設計法が用いられている。半導体技術の進歩により、1 つの LSI に数千万ゲートの回路が搭載可能となったため、HDL よりもさらに抽象度の高い記述による回路設計が注目されている。これは、動作合成システムと言われ、C 言語などによるアルゴリズム記述からのハードウェアの設計を行う。本研究で使用した Bach もこの動作合成を実現したシステムの 1 つである。

図 4 に示すように、従来の HDL によるハードウェア設計は、C 言語によるアルゴリズムの作成を行い、その後ソフトウェアを使った検証、HDL での回路設計、RTL 論理合成、シミュレーションといった検証手順で行われる。そのため、HDL で回路設計の不具合が見つかった場合、修正に大きな手間が必要となる。Bach を用いた設計では、アルゴリズムの設計段階から抽象度の高い Bach を用い、機能設計・検証を行うことで、HDL による機能検証が削減されるだけでなく、Bach-C 記述から RTL の VHDL が自動生成され、各種アーキテクチャを短期間に探索できる。

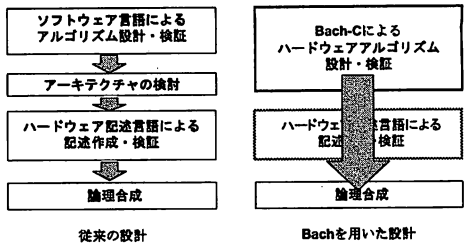

図 4　Bach の設計工程
Fig. 4　Design flow

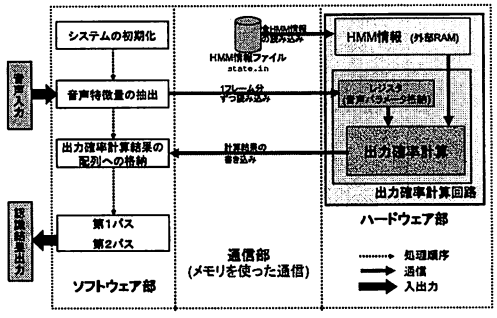

図 5　システムの概要
Fig. 5　Block diagram

3.3　出力確率計算部のハードウェア化

出力確率計算は、第 1 パス、第 2 パスから音響尤度計算を行う度にコールされる為、通信が多い。また、1 回の処理時間は大きくないが総計としては大きな割合を占める。これをハードウェア化し、効率的に動作させるためには、通信を最小限とし、独立して動作させることが必要である。出力確率計算は、最大実行回数が「状態数」×「フレーム数」と決まっている。これに着目し、第 1 パス、第 2 パスに先駆けて、全状態分の出力確率を計算し、その結果を配列に格納する。第 1 パス、第 2 パスからの音響尤度計算の要求に対し、計算結果を配列から参照するように構成の変更を行った。そのため、最大実行回数に対して約 46 %（= (最大実行回数−実行回数)/最大実行回数）が余分な計算となるが、全出力確率を配列に格納することでソフトーハード間の通信量を削減できる（図 5）。

ある状態において、1 つの混合に対する HMM 情報は、ガウス分布の平均と分散（各 25 次元）、重み付け値 2 つの計 52 個のパラメータを持つ。本研究では、4 混合 1012 状態の HMM を使用する。

3.4　固定小数点演算のビット数の決定

HMM の出力確率計算は、ソフトウェアでは浮動小数点演算で行われていたが、ハードウェアで浮動小数点演算を実装すると回路規模が大きくなるため、固定小数点演算を実装した。固定小数点演算の整数ビットと小数ビットを認識率が保証できる最小のビット数にするため、実験的に様々なビット数で音声を認識させてソフトウェアの認識率と比較した。表 1、表 2 より、

表 1 整数ビットと単語認識率
Table 1 Number of integer bit

整数ビット数	単語認識率（%）
浮動小数点	94.72
11bit 固定小数点	4.36
12bit 固定小数点	13.54
13bit 固定小数点	18.51
14bit 固定小数点	94.72
15bit 固定小数点	94.72
16bit 固定小数点	94.72

表 2 小数ビットと単語認識率
Table 2 Number of decimal bit

小数ビット数	単語認識率（%）
浮動小数点	94.72
15bit 固定小数点	0.55
16bit 固定小数点	0.55
17bit 固定小数点	13.42
18bit 固定小数点	94.72
19bit 固定小数点	94.72
20bit 固定小数点	94.72

整数 14bit、小数 18bit の固定小数点演算にすると、ソフトウェアで浮動小数点演算を行ったときと同等の認率を維持できることが確認できた。

4. 高速化手法

著者等が行ってきた幾つかの応用に対する C 言語設計の結果 [11]〜[15] から、C 言語設計におけるアーキテクチャ最適化法としてデータアクセス法、並列化、専用演算回路設計、モジュール構成の変更がある。この 4 種類の方法における性能と回路規模はトレードオフの関係にある。本文では、要求仕様の性能を満足しつつ、回路規模の増大を抑えることを目的とし、並列化、データアクセス法、専用演算回路について考察する。

4.1 並 列 化

2 種類の並列化手法、データ分割による並列処理とガウス分布計算と指数対数計算のパイプライン処理を提案する。

4.1.1 データ分割による並列化

HMM の異なる状態における処理も独立しているため、複数の状態を並列に計算させる。図 6 は、偶数状態を計算する回路と奇数状態を計算する回路を並列動作させ、2 倍の高速化を図るものである。同じように 4 つの状態を並列にした 4 並列回路も作成する。

4.1.2 パイプライン処理

ガウス分布計算回路と指数対数計算回路をパイプライン処理する回路を混合数用意することで並列化を行う。4 混合の場合を図 7 に示す。ガウス分布計算回路はレジスタから音響特徴量を、外部 RAM から HMM 情報を読み込んで 25 次元ガウス分布を計算し、その結果を指数対数計算回路に送信する。その後、次のガウス分布計算を始める。指数対数計算回路では、前段のガウス分布の計算結果と初期値、または上段の指数対数計算を

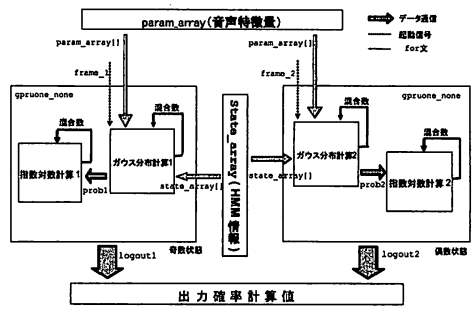

図 6 2 状態分の並列処理回路
Fig. 6 Parallel processing

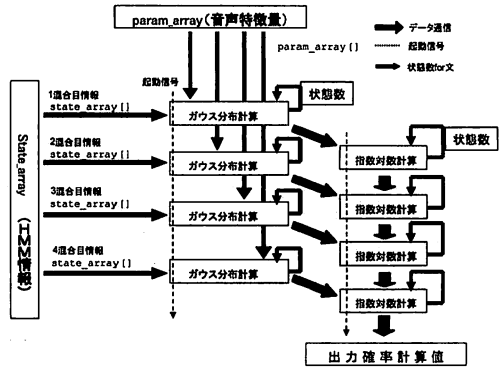

図 7 パイプライン処理
Fig. 7 Pipeline processing

受信し、計算を行う。最終段の指数対数計算関数の計算結果が出力確率値となる。この順番で入力されるフレームに対しパイプライン処理を行う。

4.2 データアクセス法

HMM 情報が格納されている外部 RAM のアクセス時間を改善するために、1 混合分の HMM 情報 (52 個のパラメータ) を格納するバッファを出力確率計算回路内部に 2 個用意する (図 8)。buffer1(2) をガウス分布計算回路がアクセスしている間に、buffer2(1) に次のガウス分布計算に必要な HMM 情報を先読みさせる。ガウス分布計算回路では、25 次元ガウス分布計算を並列に行うように変更し、ガウス分布計算回路と指数対数計算回路をパイプライン動作させるような設計を行う。

もう一つのデータアクセス法として、メモリを分割し、メモリアクセス競合を削減する設計を行なう。

4.3 専用演算回路

専用演算回路を用いる方法として、プラグマ unroll を用いたガウス分布計算専用回路と指数対数計算専用回路を設計する。

4.3.1 プラグマ unroll

ガウス分布計算回路では 25 次元ガウス分布計算を 25 回ループさせることで計算している。Bach システムでは、プラグマ unroll [6] を用いて、繰り返し回数が一定のループを展開する

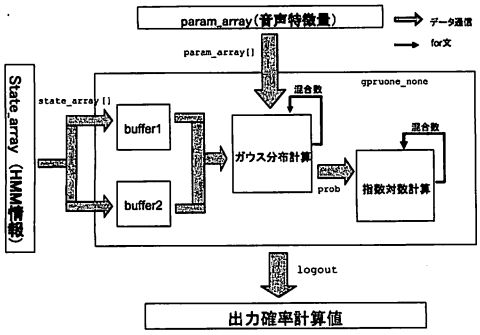

図 8 バッファ搭載回路
Fig. 8 Data bufferring

表 3 並列及びパイプライン回路の設計結果
Table 3 Design results

回路構成	回路規模 gates	処理時間 ms
シーケンシャル処理	17,580	25.77
並列処理 2 状態 RAM 分割なし	28,540	14.07
並列処理 2 状態 RAM 分割あり	28,781	11.41
並列処理 4 状態 RAM 分割なし	50,709	6.13
並列処理 4 状態 RAM 分割あり	49,418	5.77
パイプライン処理 RAM 分割なし	50,418	5.90
パイプライン処理 RAM 分割あり	49,105	5.40

ことができる。各繰り返しは、実行可能なときは並列に実行されるように回路が自動合成されるため、実行ステップが減少する。ただし、ループを展開する場合、ループ内の計算をできるだけ並列に処理するために多くの演算器を用いることになるため、回路規模が増大する。

4.3.2 指数対数計算の実装

出力確率計算には、指数対数計算が多く使われている。本研究では、Erecegovac's radix-16 algorithms, Faster Shift and Add algorithms [9]（FSA 法）を用いた指数対数計算の専用回路を設計する。

5. 設計結果

出力確率計算回路について、パイプライン処理と 2 及び 4 状態並列処理を行う 2 種類の並列化法、メモリ分割とバッファ搭載を行う 2 種類のデータアクセス法、プラグラマ unroll を用いた専用演算回路などを設計に適用し、各々の効果を測定した。指数対数計算専用回路の有効性は明らかなので、すべての回路に搭載した。動作周波数を 100MHz とした。シーケンシャル回路とは、Julius ソフトウエアの処理順序どおりに設計した回路を言う。

表 3 は、シーケンシャル回路、2 及び 4 状態の並列回路、とパイプラン回路について、メモリアクセスの競合を回避するための RAM 分割の適用を行った設計結果である。2 及び 4 並列処理は、処理時間、回路規模共に分割数に比例した結果を得た。パイプライン処理回路においても、ほぼ同様な結果であった。

表 4 データバッファ搭載
Table 4 Data buffering

回路構成	回路規模 gates	処理時間 ms
シーケンシャル処理	46,868	20.71
並列処理 2 状態 RAM 分割なし	88,220	10.32
並列処理 2 状態 RAM 分割あり	87,600	10.37
並列処理 4 状態 RAM 分割なし	168,830	5.17
並列処理 4 状態 RAM 分割あり	167,982	5.19
パイプライン処理 RAM 分割なし	167,982	4.90
パイプライン処理 RAM 分割あり	166,846	4.89

表 5 プラグマ unroll 利用
Table 5 Pragma unroll

回路構成	回路規模 gates	処理時間 ms
シーケンシャル処理	41,867	4.03
並列処理 2 状態 RAM 分割なし	79,778	5.63
並列処理 2 状態 RAM 分割あり	79,643	3.61
並列処理 4 状態 RAM 分割なし	150,603	4.82
並列処理 4 状態 RAM 分割あり	149,679	1.60
パイプライン処理 RAM 分割なし	138,058	5.13
パイプライン処理 RAM 分割あり	142,609	1.26

表 6 データバッファ搭載とプラグマ unroll
Table 6 Data buffering and pragma unroll

回路構成	回路規模 gates	処理時間 ms
シーケンシャル処理	73,184	6.47
並列処理 2 状態 RAM 分割なし	140,829	4.35
並列処理 2 状態 RAM 分割あり	142,249	2.64
並列処理 4 状態 RAM 分割なし	273,128	4.27
並列処理 4 状態 RAM 分割あり	277,346	1.47
パイプライン処理 RAM 分割なし	269,778	4.36
パイプライン処理 RAM 分割あり	268,703	1.12

表 4 は、表 3 の回路に対し、メモリアクセスの競合を回避する方法の一つとしてデータバッファを用いた回路の結果である。表 3 との比較から明らかなように、バッファ搭載は RAM 分割とほぼ同等の処理時間を得るが、回路規模はバッファ分だけ増大してしまう。

表 5 は、表 3 の回路に対し、プラグマ unroll によるガウス分布計算を専用演算回路化した設計結果である。適用したすべてのアーキテクチャに対し、表 3 や表 4 より処理時間が短縮された。unroll により、メモリアクセス、ガウス分布計算が効率的に自動並列化されたためと考えられる。

表 6 は、表 3 の回路に対し、バッファ搭載によるデータアクセス法とプラグマ unroll による専用演算回路化を共に適用した結果である。シーケンシャル回路は、バッファがあるため、プラグマ unroll が効果的に動作せず、かえって処理時間を増大させた。しかし、そのほかの回路は、バッファ搭載により表 5 より 9 %～13 %処理時間が短縮でき、最も高速な回路となった。回路規模はバッファと unroll による並列回路の分だけ増大に留

まった。

6. まとめと今後の課題

本研究では、音声認識システムの計算時間を多く要している部分をハードウェア化することにより、システム全体を効率化する手法を提案した。パイプライン処理回路や並列処理に対し、バッファ搭載、メモリ分割、プラグマ unroll を適用した回路を、音声認識に必要となる HMM の出力確率計算に用いることで処理速度の向上が見られた。リアルタイム化のためには、第1パスを構成する単語バイグラム抽出、尤度ソートといった処理各々も 10ms 以内にする必要がある。本研究で設計した出力確率計算以外の部分のハードウェア化による高速化を検討し、回路設計を今後行う予定である。

謝辞

Bach を用いたハードウェア設計を実現するに当たり、多大なる御指導を頂いたシャープ株式会社 電子デバイス開発本部 先端技術開発研究所 山田晃久様をはじめ、BACH 開発グループの皆様に心から御礼申し上げます。音声認識技術を使用した研究を行うにあたり、オープンソースとして大語彙音声認識エンジン「Julius」を公開し、ソフトウェアの解析にあたってご指導いただいた名古屋工業大学大学院情報工学専攻 李晃伸助教授に深くお礼申し上げます。本研究は、東京大学大規模集積システム設計教育研究センターを通し、シノプシス株式会社の協力で行われたものである。

文　献

[1] 平成 15 年特許庁総務部技術調査課, 音声認識技術に関する特許出願技術動向調査報告, *http : //www.jpo.go.jp/shiryou/pdf/gidouhoukoku/voicerecognition.pdf*.

[2] 吉沢真吾、宮永喜一、吉田則信, 一括並列処理による HMM 高速化手法及びその VLSI 設計, 電子情報通信学会論文誌, Vol.J85-A, 1440～1450, 2002.

[3] S. Nedevsschi, R. K. Patra, and Eric A. Brewer: "Hardware Speech Recognition for User Interfaces in Low Cost Low Power Devices," pp 684-689, proc. of 42nd DAC,(2005)

[4] 中村哲, 音声ヒューマンインターフェース, 専門講習会講演論文集ヒューマンインターフェース技術の最新動向, 26～40, 2003.

[5] 鹿野清宏、伊藤克亘、河原達也、武田一哉、山本幹雄, 音声認識システム, オーム社, 東京, 2002.

[6] Bach システムマニュアル, シャープ株式会社提供, 2004.

[7] Sergiu Nedevsschi, Rabin K. Patra, Eric A. Brewer, Hardware Speech Recognition for User Interfaces in Low Cost,Low Power Devices, proc. of 42th DAC, 2005.

[8] K.Okada, A. Yamada, T. Kambe, Hardware Algorithm Optimization Using Bach C, IEICE Trans. Fundamentals vol.E85-A, No.4, pp835-841, 2002.

[9] Muller,J.M., Elementary Functions, Birkhauser, Boston, 1997. 技術評論社, 東京, 1991.

[10] 松野 裕之, 近畿大学大学院総合理工学研究科平成 17 年度修士論文, 2006.

[11] T. Kambe, et al., "A C-based synthesis system, Bach, and its application," Proc. of ASP-DAC 2001

[12] T.Kambe, et al., "C-based Design of a Real Time Speech Recognition System," ISCAS2006, 2006 年 5 月

[13] K.Jyoko, et al., "C-based Design of a Particle Tracking System," SASIMI2006, 2006 年 4 月

[14] 小田島賢和, 他 "C 言語設計によるリードソロモン符号復号化回路の最適化," 情報処理学会システム LSI 設計技術研究会, 2006 年 10 月

[15] 大窪啓太, 他, "特定用途向け低ビット複合演算回路設計,"電子情