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

2005－ＳＬＤＭ－119（14）
2005／3／17

音声認識システムのハードウェア化の一手法
－HMM出力確率計算のハードウェア化－

松野 裕之* 友利 記昌* 宮崎 崇** 西村 英樹*** 神戸尚志*

*近畿大学大学院総合理工学研究科 エレクトロニクス系工学専攻 システム設計工学研究室
**㈱トヨタテクノサービス

***㈱サニー技研

E-mail: *04870422d@msa.kindai.ac.jp, *tkambe@ele.kindai.ac.jp

あらまし 本研究では、音声認識ソフトウェアにおいて計算時間を多く要している部分をハードウェア化し、システム全体の効率化を図る手法を提案する。HMMを用いた連続音声認識は認識性能が高いが、出力確率計算に処理時間を多く要するため、リアルタイムで実行するには高速なCPUが必要となる。ソフトウェア処理全体の約40%を占める出力確率計算部分をハードウェア化することで、CPUの負荷を少なくし、システム全体を効率化することが出来た。出力確率計算回路では、並列処理、パイプライン処理、キャッシュメモリを搭載する等の高速化手法を用いて設計した。その結果、出力確率計算の部分をリアルタイム処理するために十分な性能が得られた。

キーワード 音声認識，HMM，並列処理，パイプライン処理、C言語設計、BACHシステム

A Hardware Design for Voice recognition
－HMM Output Probabilities calculation circuit－

Hiroyuki MATSUNO* Norimasa TOMORI* Takashi MIYAZAKI** Hideki NISHIMURA*** Takashi KAMBE*

*System Design Methodology Laboratory, Kinki University, 3-4-1 Kowakae, Higashi-Osaka City 577-8502 Japan
** TOYOTA Techno Service Corp. 1-21 Imae Hanamoto-cho, Toyota-shi, Aichi, 470-0334 Japan

*** Sunny Giken Inc. 3-1-9 Nishi-dai, Itami-shi, Hyogo-ken, 664-0858 Japan

E-mail: *04870422d@msa.kindai.ac.jp, *tkambe@ele.kindai.ac.jp

Abstract Voice recognition is becoming a popular technology for human interface. In this paper, the hardware design to calculate output probabilities is proposed to improve system performance. Conventional approach for high level voice recognition needs high speed processor. Our approach can reduce the load of CPU by designing the hardware engine for calculation of output probabilities which is 40% of total processing time. And we adapt parallel and pipelining processing with cash memories for real time processing.

Keyword Voice recognition, HMM, parallel and pipelining processing, C based design, BACH system

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。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、文による入力には対応していない。文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言い、現在は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。本研究では、この出力確率計算部をハードウェア化することでシステム全体を大幅に効率化する手法を提案する。そして、連続音声認識システムを携帯端末等でリアルタイムに動作させる事を目的とする。

出力確率計算部の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える Bach-C[1]を用いて設計した。Bach-Cはハードウェ

アの並列動作等を記述出来るように ANSI-C を拡張したハードウェア記述言語であり、抽象度の高い設計が可能となる。本研究では出力確率計算部をハードウェア化し、高速化のため並列処理、パイプライン処理、キャッシュメモリ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した。

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のハードウェア化手法、4 章で出力確率計算部の高速化について、5 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

本研究では、大語彙連続音声認識エンジン「Julius」[2]を用いて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、低消費電力化、システムの小規模化を目指す。連続音声認識は、入力された音声を、音素→語→単語→文へと推定していく。音声は時間的に特徴量の変化が激しいため、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てスペクトル解析を行い、音響特徴量を抽出する。

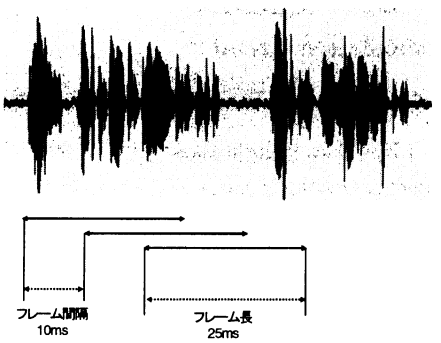


図 1　フレーム分析

Fig.1 Frame analysis

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と 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を行う。

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、音響特徴量の分布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。

$$\log b_{im}(o_t) = \omega_i - \frac{1}{2}\sum_{p=1}^{P}\frac{1}{\sigma^2{}_{jmp}}(o_{tp} - \mu_{imp})^2 \quad (1)$$

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 $\sigma^2{}_{imp}$ 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。

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

ソフトウェアに組み込む場合、このままでは計算量が大きいため、b_{im}, b_{im+1} を $x, y(x > y)$ とするとき、以下の式で近似を行う。

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

大語彙連続音声認識

大語彙連続音声認識技術とは単語認識に留まらず、日常の友人との会話を認識することを目指すものである。その探索空間は膨大なため、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システムを用いる。

大語彙音声認識エンジン Julius は、その最大の特徴として可搬性を持たせて作られており、組み込むモデルファイルを変更することで幅広い用途に対応している。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。

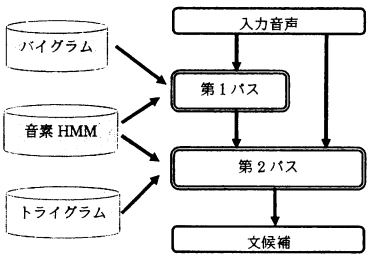


図 2 マルチパス検索

Fig.2 Multi path search

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 パスで得た候補について、高精度な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 の高精度版を用いた高速化を検討している。

3. ハードウェアの設計

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効果的である。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。

本研究では、Julius を連続音声認識エンジンとして利用し、

- 計算量が多い
- 他とは独立し、並列に計算することで効率が上がる

の2点を考慮してハードウェア化する部分を決定し、ソフトウェアとの通信を最小限とし、全体の効率向上を図る。

Julius 高精度版のプログラム構造の解析、処理時間の分析結果より、音響尤度計算部に着目し、その中の出力確率計算部がソフト全体の処理時間の約 40%を占めていることが分かった。したがってこの部分をハードウェア化することに決める。Julius はフレームシフトが 10ms であるので、リアルタイムで音声認識を行うには 1 フレームの処理を 10ms 以内に終わらせなければならない。ハードウェアは並列動作が可能なのでソフトウェアより高速な計算が期待出来る。そこで、Julius からこの部分を独立させて図 3 に示すシステムを構成した。

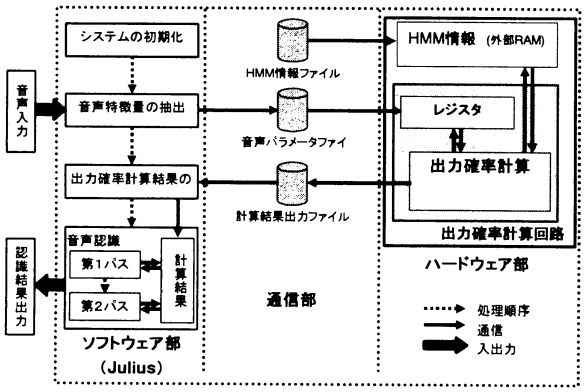


図 3 システムの概要

Fig.3 Block diagram of target system

Bach を用いた設計工程

ハードウェアの設計手法は、ゲートレベルの設計や、レジスタ転送レベルのハードウェア設計言語の VHDL、verilog-HDL を用いた設計が挙げられる。半導体技術の進歩により、1 つの LSI に数百万、数千万ゲートの回路が搭載可能となったため、現在の主流は VHDL など HDL 記述による設計である。近年、HDL よりもさらに抽象度の高い記述での回路設計が行われている。これらは、動作合成システムと言われ、C 言語やそれに近い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-C もこの動作合成を実現したシステムの 1 つである。

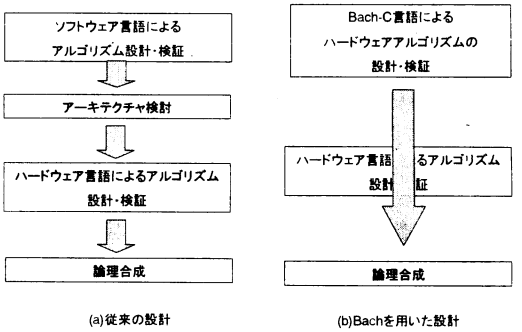


図 4 Bach の設計工程

Fig.4 Design process using Bach system

図 4 に示すように、従来の HDL によるハードウェア設計は、トップダウン式に行われてはいるが、アルゴリズムの作成に C 言語等を用いて行い、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、シミュレーションによる検証という手順で行われる。HDL で回路設計を行った後に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-C を用いた設計では、トップレベルの設計から抽象度の高い Bach-C を用い、機能設計・検証を行うことで、ハードウェア作成のために 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、ハードウェアの設計が大幅に効率化される。

出力確率計算部のハードウェア化

音響尤度計算部は第 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計算をし、その結果を配列に格納する。Julius は、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

を変えた。この方法は、従来 Julius が最大実行回数に対して 53.97%程の計算を行っていたのに対してすべての場合の計算を行う為、約 46%の余分な計算をしている。この点は、今後ソフトウェアとの比較を行うにあたって留意が必要である。

本研究ではフレーム単位で計算を行う出力確率計算回路を設計した。1 フレームの情報を一時的にレジスタに格納し、外部 RAM に格納されている HMM 情報を読み込んで全状態の計算を行う。

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。重み付け値とは、使用する HMM によって混合ごとに決まる定数で、出力確率計算において、ガウス分布の 25 次元の計算をした後加えることで計算量を減らしている。本研究では 4 混合 1012 状態の HMM を使用する。

固定小数点演算のビット数を決定

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少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。その結果、整数 16 ビット、少数 16 ビット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様の認識率を維持できることが分かった。

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。本研究では連分数展開のアルゴリズムを用いた指数対数計算回路を設計した。

また、Erecegovac's radix-16 algorithms, Faster Shift and Add algorithms[4]を用いた指数対数計算回路を組み込んだ回路設計も行い、両アルゴリズムの比較をする。

4. ハードウェアの高速化

高速に出力確率計算をするために、並列処理、パイプライン処理、キャッシュメモリを搭載した回路を設計した。ここでは、その高速化手法について述べる。

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混合数用意することでパイプライン処理を行う。4 混合の場合についてのパイプライン処理を図 5 に示す。

ガウス分布計算回路、指数対数計算回路をそれぞれ 4 つずつ用意する。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。そして、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受けて、計算を行う。最終段の指数対数計算関数での計算結果を出力確率値とする。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。

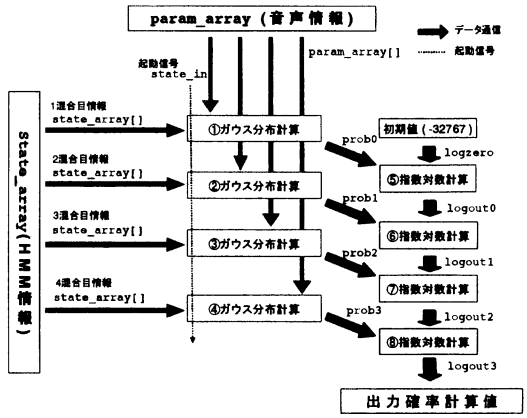


図 5 パイプライン処理

Fig.5 Block diagram of pipeline processing

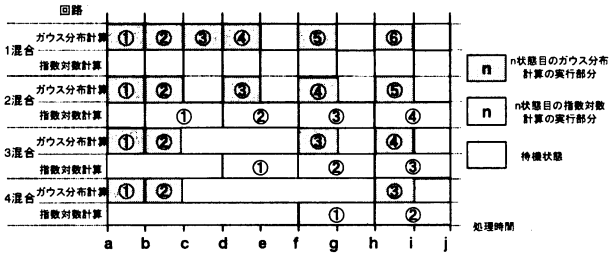


図 6 パイプラインの処理動作

Fig.6 Pipeline processing

ガウス分布計算と指数対数計算のパイプライン処理動作を図 6 に示す。数字と灰色のブロックはガウス分布計算、数字と白色のブロックは指数対数計算、数字のないブロックは待機状態を表している。また、数字 n は n 状態目を表している。

1 混合目の指数対数計算は、入力値が閾値以下のため行われない。①状態目の出力確率計算は図 6 の処理時間 h のところで完了する。以降、②状態目から最終状態目までの出力確率が連続して出力される。

並列処理

ガウス分布計算と指数対数計算はほぼ独立しているため、並列に動作させることができる。また、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動作させる。2 つの状態を並列処理し

た場合について図7に示す。

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。

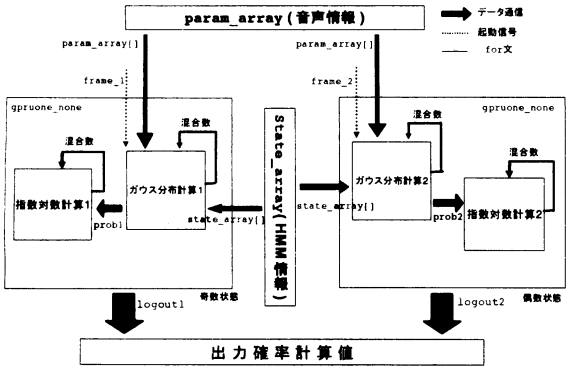


図7 2状態分を並列に処理した回路

Fig.7 two states' parallel processing states

RTLシミュレーションの結果

表1に出力確率計算回路をシーケンシャル処理、並列処理、パイプライン処理で設計し、RTLシミュレーションを行った結果を示す。処理時間は1フレームの出力確率計算にかかる時間とする。

- 回路の動作周波数 200MHz
- 指数対数計算回路(連分数展開アルゴリズム)
- 4混合 1012状態の HMM を使用

表1.処理時間と回路規模

Table 1 processing time and area of parallel processing

		処理時間(ms)	回路規模(ゲート数)
シーケンシャル処理		25.13	75,221
並列処理	2状態	10.84	274,312
	4状態	5.42	505,330
パイプライン処理		5.41	500,161

表1より、回路規模と処理時間は反比例することがわかる。リアルタイム処理のためには1フレーム 10ms 以内に処理する必要があるので、出力確率計算以外の処理を考えた場合、4状態並列処理、またはパイプライン処理の回路がリアルタイム処理の可能性があると言える。

指数対数関数近似アルゴリズム

指数対数計算回路に Erecegovac's radix-16 algorithms, Faster Shift and Add algorithms を用いた出力確率計算回路と連分数展開アルゴリズムを用いた出力確率計算回路の比較を示す。

- 回路の動作周波数 200MHz
- 4混合 1012状態の HMM を使用

表2.両アルゴリズムを用いたときの処理時間

Table 2 comparison of two approximation methods

		処理時間(ms)	
		連分数展開	Erecegovac's radix-16 algorithms, Faster Shift and Add algorithms
シーケンシャル処理		25.13	22.07
並列処理	2状態	10.84	10.84
	4状態	5.42	5.42
パイプライン処理		5.41	5.40

表2より、シーケンシャル処理では連分数展開アルゴリズムより Erecegovac's radix-16 algorithms, Faster Shift and Add algorithms を用いた回路が約14%高速である。しかし、並列処理、パイプライン処理では処理時間の差が現れない。この原因は、ガウス分布計算では外部 RAM へのアクセスが多く、ガウス分布計算がボトルネックとなり、並列処理、パイプライン処理では連分数展開との差が現れない事にあった。

そこでキャッシュメモリを回路に内蔵することでメモリアクセスを高速にした回路を設計した。

キャッシュメモリ内蔵の出力確率計算回路

HMM情報が格納されている外部RAMとガウス分布計算回路と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)を格納するキャッシュメモリを出力確率計算回路内部に 2 個用意した(図8)。キャッシュメモリ 1(2)をガウス分布計算回路がアクセスしている間に、キャッシュメモリ 2(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方法をとった。ガウス分布計算では決まった順番で逐次的に HMM 情報を読むため、先読みすることは簡単である。そして、ガウス分布計算回路では 25 次元ガウス分布計算を一度に並列に行うように変更した。また、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ている。

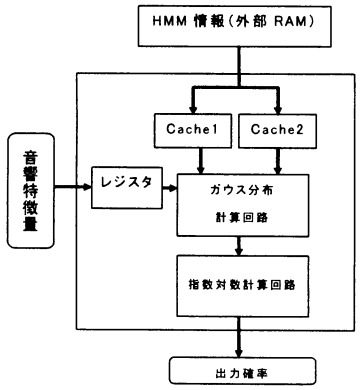


図8 キャッシュ搭載出力確率計算回路

Fig.8 Output Probabilities calculation circuit with cache memory

RTL シミュレーションの結果

表 3 にキャッシュメモリ搭載出力確率計算回路の RTL シミュレーション結果を示す。

- 回路の動作周波数 200MHz
- 4 混合 1012 状態の HMM を使用

表 3.両アルゴリズムを用いたときの処理時間
Table 3 the speed of output probabilities calculation circuit with cache

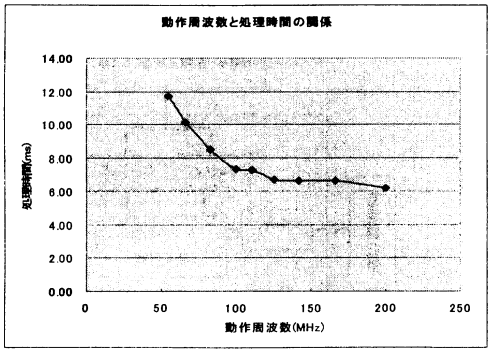
キャッシュ搭載出力確率計算回路		
指数対数計算アルゴリズム	連分数展開	Erecegovac's radix-16 algorithms, Faster Shift and Add algorithms
処理時間(ms)	6.23	5.45

キャッシュメモリを搭載した出力確率計算回路では指数対数計算回路に Erecegovac's radix-16 algorithms, Faster Shift and Add algorithms を用いたほうが連分数展開アルゴリズムを用いたときより約 14%高速になった。このことより、ガウス分布と外部 RAM のアクセス時間の問題を改善することができ、キャッシュメモリの有効性を確認できた。

回路規模は動作周波数 200MHz の回路で約 42 万ゲートとなった。処理時間は表 2 のパイプライン処理と同等で、回路規模は約 8 万ゲート削減できた。

グラフ 1 に、Erecegovac's radix-16 algorithms, Faster Shift and Add algorithms を用いたキャッシュ搭載出力確率計算回路の動作周波数を 55MHz から 200MHz まで変化させた時の処理時間と回路規模の関係を示す。

グラフ 1 動作周波数と処理時間の関係
Graph 1 speed vs. processing time



グラフ 1 より、このキャッシュ搭載回路では動作周波数を 100MHz 程度に下げても、出力確率計算をリアルタイム処理が可能である。動作周波数を下げることにより、消費電力と回路規模の両方を削減できる。

5. 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一手法を提案した。音声認識で用いられる HMM の出力確率計算をフレーム単位で高速に計算するハードウェアを設計し、ソフトウェアとハードウェアを協調設計することで小規模なシステム上で高精度なリアルタイム音声認識を動作させることが出来る可能性を示した。

今後の課題としては、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を検討する。本研究では音声認識システム全体の約 40%をハードウェア化したので、ソフトウェア側に 60%程度の処理が残っている。この中からハードウェアに適している処理をハードウェア化することにより、さらに CPU の負荷を小さく出来る。また、ソフトウェア・ハードウェア協調検証を行い、システム全体の評価を行う。その評価に基づいてソフトウェア・ハードウェアの分割の最適化を検討する。

謝辞

Bach を用いたハードウェア設計を実現するに当たり、多大なるご指導を頂いたシャープ株式会社 IC 事業本部要素技術開発センタ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なうにあたり、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奈良先端科学技術大学院大学情報科学研究科李晃伸助手に深くお礼申し上げます。

文　献

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

[2] 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)

[3] 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)

[4] Muller,J.M.: "Elementary Functions", Birkhauser, Boston,(1997)