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

2004－SLDM－116 (26)
2004／10／22

IPベース設計におけるバスアーキテクチャ最適化手法の提案

上田　恭子† 坂主　圭史† 米岡　昇† 武内　良典† 今井　正治†

† 大阪大学 大学院情報科学研究科　〒565–0871 大阪府吹田市山田丘 1–5
E-mail: †{k-ueda,sakanusi,yoneoka,takeuchi,imai}@ist.osaka-u.ac.jp

あらまし　IPベース設計では，設計対象システムの性能はバスアーキテクチャによって大きく異なるため，最適なバスアーキテクチャの選択は非常に重要である．本稿では，高速性能見積り手法を用いたバスアーキテクチャ最適化手法を提案する．本研究では，バストポロジ，バス動作周波数，バスビット幅，バッファ数によってバスアーキテクチャをパラメタ化する．そして，バスアーキテクチャ最適化問題を定式化し，バスアーキテクチャ探索手法を提案する．提案手法では，システムレベルのプロファイリング結果を用いた高速見積り手法によって性能を見積り，様々なバスアーキテクチャの性能を評価する．評価実験により，提案手法が最適バスアーキテクチャの探索時間を大幅に削減できることが知られた．

キーワード　IPベース設計，バスアーキテクチャ，性能見積り，分枝限定法

Bus architecture optimization method for IP-based design

Kyoko UEDA†, Keishi SAKANUSHI†, Noboru YONEOKA†,

Yoshinori TAKEUCHI†, and Masaharu IMAI†

† Graduate School of Information Science and Technology, Osaka University
1–5 Yamadaoka, Suita, Osaka, 565–0871 Japan
E-mail: †{k-ueda,sakanusi,yoneoka,takeuchi,imai}@ist.osaka-u.ac.jp

Abstract　In IP-based design, to find the optimal bus architecture is very important problem because bus architecture strongly affects the performance of the target system. This paper proposes a bus architecture optimization method using fast performance estimation. The optimization problem of bus architecture that is parameterized by bus topology, bus data transfer rate, bus bit width, and the number of buffers, is formalized and its exploration method is proposed. Proposed method explores all possible bus parameter sets estimating the performance and hardware area with profiling information. Experimental results show that proposed method can determine the optimal bus architecture and its time is drastically reduced compared to the conventional method.

Key words　IP-based design, bus architecture, performance estimation, branch-and-bound method

1. はじめに

近年，多機能化が求められる組み込みシステムは複雑化の一途をたどり，設計コストが増大するという問題が生じている．そこで，既設計の機能ブロックを再利用することで設計時間を短縮する，IPベース設計が注目されている[1]．IPベース設計では，設計者はIPデータベースから機能ブロックを選択し，それらのブロックを接続するバスのアーキテクチャを決定する．面積や処理時間などの設計制約を見たす最適なアーキテクチャを見つけるためには，様々なアーキテクチャでの設計品質を評価する必要がある．したがって，様々なアーキテクチャを対象とした，高速な設計品質見積もり手法，および，効率的なアーキテクチャ探索手法が求められる．

我々は[2]において，システムレベルのプロファイリングに基づく，アーキテクチャレベルでの高速な性能見積り手法を提案した．本稿では，その高速性能見積り手法を用いたバスアーキテクチャ最適化手法を提案する．バスアーキテクチャをバストポロジ，バス動作周波数，バスビット幅，およびバッファ数でパラメタ化し，バスアーキテクチャ最適化問題を定式化する．提案手法はバスパラメタ探索木を用いて，考えうるすべてのパラメタの組合せを探索する．また，探索時間を削減するために，バスパラメタの探索順序について検討した．

本稿の構成は以下のとおりである．第2節でバスアーキテクチャ探索の関連研究について述べる．第3節で高速性能見積り

手法の概要を説明する．第4節でバスアーキテクチャ最適化問題の定式化および探索アルゴリズムについて述べる．第5節で実験結果について述べ，第6節でまとめる．

2. 関連研究

従来，システムの性能はアーキテクチャレベルのシミュレーション[3],[4]によって評価されていたが，アーキテクチャレベルのシミュレーションには時間がかかるという問題があった．そこで，様々なバスアーキテクチャに適応可能な性能評価手法が提案された[5],[6]．これらの手法は，ある一つの評価モデルを使用して様々なバスアーキテクチャの性能を見積もることができるが，実装する機能ブロックが異なるアーキテクチャを評価するためには，評価モデルを再構築する必要がある．したがって，実装する機能ブロックが異なる様々なアーキテクチャを短時間で評価することは困難である．

バスアーキテクチャの初期解を決定し，初期解を改良することで準最適なバスアーキテクチャを生成する，バスアーキテクチャ最適化手法が提案されている[7]．この手法では，設計者の指定したバスアーキテクチャ・テンプレートへのデータ転送のマッピングのみを考えるため，探索空間が制限されており，様々なバストポロジを比較する場合はテンプレートを変えて探索する必要がある．

我々はこれまでに高速性能見積り手法を提案している[2]．本見積り手法では，まず，システムレベルのプロファイリングによって得られた結果から，データ処理とデータ転送の実行順序を表すシステムレベル実行順序グラフを構築する．そして，システムレベル実行順序グラフを変更，解析することで，対象アーキテクチャでの性能を見積もる．本稿では，システムレベルプロファイリングに基づく高速性能見積り手法を用いた，バスアーキテクチャ最適化手法を提案する．

3. 高速アーキテクチャレベル性能見積り手法

本節では，高速アーキテクチャレベル処理時間見積り手法[2]の概要について説明する．

性能見積り手法の見積りフローを図1に示す．見積り手法は，プロファイリングフェーズと見積りフェーズの2フェーズからなる．

（1） **プロファイリングフェーズ**:
プロファイリングフェーズでは，まず，アーキテクチャに依らない，データ処理とデータ転送の間の実行順序関係とデータ転送量を，SystemC[8]記述を用いたプロファイリングによって取得し，システムレベル実行順序グラフ（SL-EOG：System-level Execution Order Graph）を構築する．プロファイリング結果とSL-EOGはアーキテクチャに依存しないため，このフェーズは一度だけ行えばよい．

（2） **見積りフェーズ**:
見積りフェーズでは，SL-EOGおよびアーキテクチャレベルモデルから，アーキテクチャレベルの依存関係を表すアーキテクチャレベル実行依存グラフ（AL-EDG：Architecture-level Execution Dependency Graph）を構築し，データ処理時間とデータ転送時間を計算し，AL-EDGの解析により対象アーキテクチャでの処理時間を見積もる．このフェーズは，対象アーキテクチャごとに繰り返す．

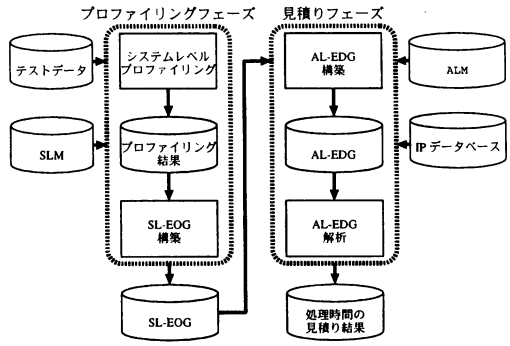


図1 性能見積りフロー
Fig. 1 Performance estimation flow.

[2]では，実行依存グラフの構築を2段階に分けているため，プロファイリングフェーズを対象アーキテクチャごとに繰り返す必要はない．したがって，設計者はプロファイリングフェーズに戻ることなく，様々なアーキテクチャの性能を同じプロファイリング結果から見積もることができる．

次節では，システムレベルモデルおよびアーキテクチャレベルモデルについて説明する．

3.1 システムレベルモデル

システムレベルモデル（SLM）はデータ処理を表すプロセスとプロセス間のデータ転送を表すチャネルで構成される．プロセスは，入力チャネルから受信したデータを処理し，出力チャネルから送信する．システムレベルモデルでは，プロセスは処理時間の情報は持たない．

システムレベルモデルの例を図2に示す．P_iがプロセス，C_jがチャネルとデータ転送の方向を表す．

3.2 アーキテクチャレベルモデル

アーキテクチャレベルモデル（ALM）は機能ブロック群構成とバスアーキテクチャから構成される．機能ブロック群構成は，対象アーキテクチャに実装される各機能ブロックにマッピングされたプロセスのリストを含む．バスアーキテクチャは，チャネルのマッピングを表すバストポロジ，各バスの動作周波数およびビット幅，各チャネルの入力および出力バッファ数から構成される．機能ブロックとバスは次の特徴を持つ．

- 機能ブロックは，その機能ブロックにマッピングされたプロセスのデータ処理を行う．
- バスは，そのバスにマッピングされたチャネルのデータ転送を行う．
- すべての機能ブロックおよびバスは並列に動作することができる．
- 機能ブロックの各ポートにはバッファが存在する．入力バッファのデータは，機能ブロックが処理を完了するまで次のデータによって上書きすることはできない．また，出力バッファにあるデータ転送前のデータを，機能ブロックが上書きするこ

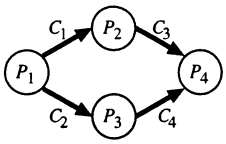
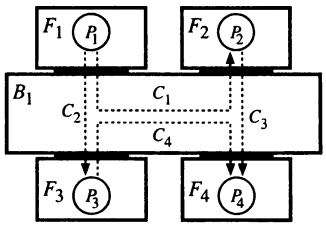
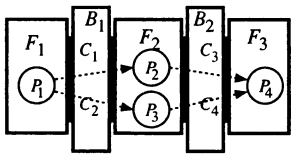


図2　システムレベルモデルの例
Fig. 2　An example of an SLM.



(a) アーキテクチャ $Arch_1$



(b) アーキテクチャ $Arch_2$

図3　アーキテクチャレベルモデルの例
Fig. 3　Examples of ALMs.

とはできない．

- 機能ブロックのハードウェア面積および各プロセスの処理時間は，IPデータベースに登録されている．

アーキテクチャレベルモデルの例を図3に示す．F_i と B_i はそれぞれ機能ブロック，バスを表す．どちらも図2のシステムのアーキテクチャであるが，図3 (a) は機能ブロック4個，バス1本のアーキテクチャ，図3 (b) は機能ブロック3個，バス2本のアーキテクチャを表す．

4. バスアーキテクチャ最適化手法

本節では，バスアーキテクチャ最適化手法について述べる．本稿では，ハードウェア面積制約下での性能最適化を対象とする．

高速性能見積り手法[2]を用いたバスアーキテクチャ最適化フローを図4に示す．バスアーキテクチャ最適化手法の入力は，対象システムのシステムレベルモデル（SLM），プロファイリング用テストデータ，機能ブロック群構成情報，ハードウェア面積制約，バス動作周波数候補，バスビット幅候補，および最大バッファ数である．提案手法は，バストポロジ，各バスの動作周波数およびビット幅，各チャネルの入力および出力バッファ数を最適化する．

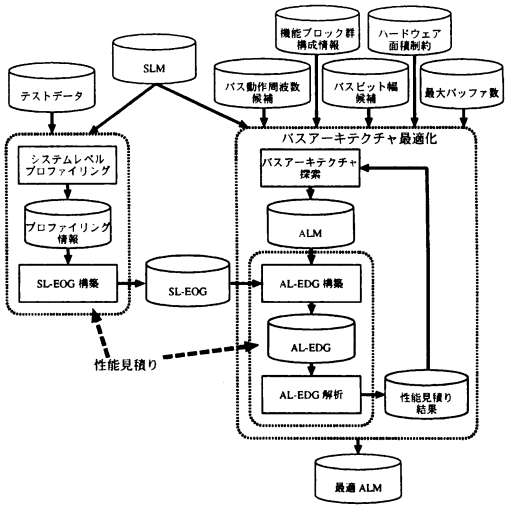


図4　バスアーキテクチャ最適化フロー
Fig. 4　Bus architecture optimization flow.

第2節で説明した高速性能見積り手法は，様々なアーキテクチャの性能を，同じプロファイリング情報を用いて見積もることができるため，提案手法はバスアーキテクチャ探索の前に一度だけ，システムの実行順序関係をプロファイリングにより取得する．そして，バスパラメタ探索木を構築，トレースして評価対象のアーキテクチャを決定し，対象アーキテクチャのハードウェア面積および処理時間を評価する．

4.1 問題の定式化

バスアーキテクチャ最適化問題を次のとおり定式化する．

- 入力
  - システムレベルモデル：$\boldsymbol{M}_{sl} = (\boldsymbol{P}, \boldsymbol{C})$，ここで，$\boldsymbol{P}$ および $\boldsymbol{C}$ はそれぞれプロセスの集合，チャネルの集合を表す．S_i および N_{max_i} はそれぞれ，チャネル $C_i \in \boldsymbol{C}$ のデータサイズと最大転送データ数を表す．
  - システムレベル実行順序グラフ：$\boldsymbol{G}_{sl} = (\boldsymbol{V}, \boldsymbol{E})$．ここで，$\boldsymbol{V}$ および $\boldsymbol{E}$ はそれぞれ点の集合，辺の集合を表す．
  - 機能ブロック群構成情報：$\boldsymbol{Arch}_{fb}$．A_i は機能ブロック $F_i \in \boldsymbol{Arch}_{fb}$ のハードウェア面積を表す．
  - バス動作周波数候補の集合：$\boldsymbol{R}$
  - バスビット幅候補の集合：$\boldsymbol{W}$
  - 最大バッファ数：N_{buf}
  - ハードウェア面積制約：A_{const}
- 出力
  - 最適バスアーキテクチャ：$\boldsymbol{Arch}_{bus} = (\boldsymbol{B}, \boldsymbol{N}_{buf})$，ここで，$\boldsymbol{B}$ および $\boldsymbol{N}_{buf}$ はそれぞれ，バス b_i の集合，各チャネルの入力および出力バッファ数の集合を表す．$r_i \in \boldsymbol{R}$ および $w_i \in \boldsymbol{W}$ はそれぞれ，バス b_i の動作周波数，ビット幅を表す．$n_{in_i} \in \boldsymbol{N}_{buf}$ および $n_{out_i} \in \boldsymbol{N}_{buf}$ はそれぞれ，チャネル $c_i \in \boldsymbol{C}$ の入力および出力バッファ数を表す．
- 目的関数：
  処理時間 $time(\boldsymbol{M}_{sl}, \boldsymbol{G}_{sl}, \boldsymbol{Arch}_{fb}, \boldsymbol{Arch}_{bus})$ の最小化

- 制約：

  ハードウェア面積 $area(\boldsymbol{Arch}_{fb}, \boldsymbol{Arch}_{bus}) \leqq A_{const}$

4.2 設計品質見積り

提案手法では，探索対象のアーキテクチャのハードウェア面積および処理時間を評価し，最適化する．本節では，設計品質の見積りについて説明する．

4.2.1 性能見積り

対象アーキテクチャの処理時間は，第2節で述べた高速性能見積り手法[2]により得る．バスアーキテクチャ最適化の前に，システムの実行順序関係をプロファイリングにより取得し，同じプロファイリング情報を用いて評価対象アーキテクチャの処理時間を見積もる．

4.2.2 ハードウェア面積見積り

対象アーキテクチャのハードウェア面積は，機能ブロックの面積，バッファの面積，およびバスの面積の和である．提案手法では，機能ブロック群構成 $\boldsymbol{Arch}_{fb}$，バスアーキテクチャ $\boldsymbol{Arch}_{bus}$ を持つアーキテクチャのハードウェア面積 $area(\boldsymbol{Arch}_{fb}, \boldsymbol{Arch}_{bus})$ を，式(1)で見積もる．

$$area(\boldsymbol{Arch}_{fb}, \boldsymbol{Arch}_{bus}) = \sum_i A_i + \sum_i (buf\_area(S_i \times N_{max_i} \times n_{in_i}) + buf\_area(S_i \times N_{max_i} \times n_{out_i})) \quad (1)$$

$buf\_area(s)$ は，サイズが s のバッファのハードウェア面積を表し，本稿では，ゲート換算値およびサイズ s のレジスタの面積の積とした．レジスタの面積は，Synopsys 社[9]の Design Compiler の論理合成結果を用いた．

4.3 最適化アルゴリズム

本節では，バストポロジ，バス動作周波数，バスビット幅，およびバッファ数の探索木を用いた，バスアーキテクチャ最適化手法を提案する．

提案手法では，バスアーキテクチャ最適化問題を，ハードウェア面積および処理時間の下限を用いた分枝限定法によって解を求める．

4.3.1 バスパラメタ探索木の構造

図5に，バスパラメタ探索木の構造を示す．以下で，各パラメタの探索木について説明する．

a) バストポロジ

バストポロジは，チャネルのバスへのマッピングを表す．図5において，$C_i \rightarrow B_j$ はチャネル C_i がバス B_j にマッピングされることを表す．図5の点 a は，チャネル C_1 がバス B_1 にマッピングされるが，チャネル C_2 のマッピング先が未定義であることを表し，点 b は，チャネル C_1，C_2 がそれぞれバス B_1，B_2 にマッピングされたバストポロジを表す．バストポロジ探索木の深さは $|C| - 1$ となる．

b) バス動作周波数

バス動作周波数探索木で定義された動作周波数で，各バスは駆動される．図5において，$r_i = R_j$ はバス B_i の動作周波数に $R_j \in R$ が割り当てられたことを表す．図5の点 c は，バス B_1 の動作周波数に R_1 を割り当て，バス B_2 の動作周波数が未定義

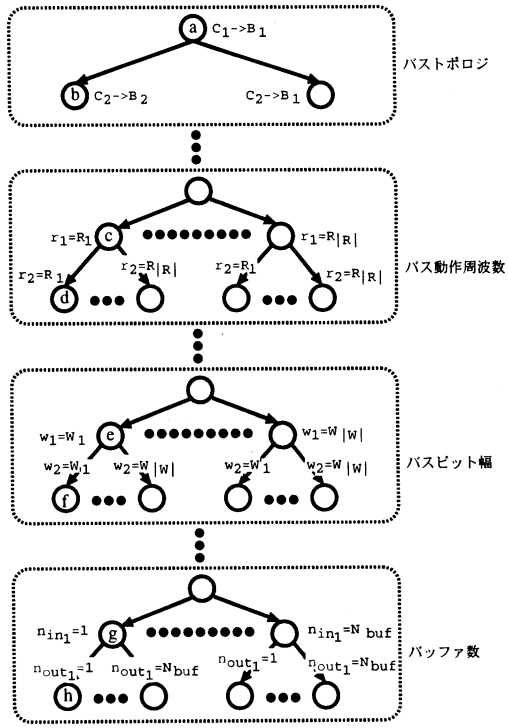


図5 バスパラメタ探索木の例

Fig. 5 An example of bus parameter set search tree.

であることを表し，点 d は，バス B_1，B_2 の動作周波数に R_1 が割り当てられたことを表す．バス動作周波数の探索木の深さは $|B|$ となる．

c) バスビット幅

各バスは，バスビット幅探索木で定義されたビット幅を持つ．図5において，$w_i = W_j$ はバス B_i のビット幅に $W_j \in W$ が割り当てられたことを表す．図5の点 e は，バス B_1 のビット幅に W_1 を割り当て，バス B_2 のビット幅が未定義であることを表し，点 f は，バス B_1，B_2 のビット幅に W_1 が割り当てられたことを表す．バスビット幅探索木の深さは $|B|$ となる．

d) バッファ数

各チャネルの入力および出力バッファ数は，バッファ数探索木で定義された数となる．図5において，$n_{in_i} = j$ はチャネル C_i の入力バッファ数に $j \leqq N_{buf}$ が割り当てられたことを表し，$n_{out_x} = y$ はチャネル C_x の出力バッファ数に $y \leqq N_{buf}$ が割り当てられたことを表す．図5の点 g は，チャネル C_1 の入力バッファ数に1を割り当て，出力バッファ数が未定義であることを表し，点 h は，チャネル C_1 の入力および出力バッファ数に1が割り当てられたことを表す．バッファ数探索木の深さは $2 \times |C|$ となる．

4.3.2 バスパラメタ探索木の枝刈り

提案手法では，ハードウェア面積および処理時間の下限によりバスパラメタ探索木の枝刈りを行う．以下で，2種類の枝刈りについて述べる．

a） ハードウェア面積の下限による枝刈り

バス動作周波数，バスビット幅，およびバッファ数が少ないほど，ハードウェア面積は小さくなるため，子孫のうち最小のハードウェア面積を持つ葉は，探索中の点で未定義のバスパラメタに対して最小の値を割り当てたバスアーキテクチャを表す葉となる．

したがって，ハードウェア面積の下限は式 (2) で表される．ここで，$\boldsymbol{Arch}_{crnt}$ および $\boldsymbol{Arch}_{small}$ はそれぞれ，探索中の点が表す，いくつかのパラメタが未定義のバスアーキテクチャ，探索中の点で未定義のパラメタに対して，最小値を割り当てたバスアーキテクチャを表す．

$$lower\_area(\boldsymbol{Arch}_{fb}, \boldsymbol{Arch}_{crnt}) = area(\boldsymbol{Arch}_{fb}, \boldsymbol{Arch}_{small}) \quad (2)$$

探索中の点の子孫のうち，ハードウェア面積が最小の葉が明らかであり，その葉のハードウェア面積がハードウェア面積制約よりも大きい場合，子孫にはハードウェア面積制約を満たすバスアーキテクチャは含まれない．そこで，式 (3) を満たす点の子孫を枝刈りする．

$$lower\_area(\boldsymbol{Arch}_{fb}, \boldsymbol{Arch}_{crnt}) > A_{const} \quad (3)$$

b） 処理時間の下限による枝刈り

バス動作周波数，バスビット幅，およびバッファ数が大きいほど，処理時間は短くなる傾向にあるため，子孫のうち最短の処理時間を持つ葉は，探索中の点で未定義のバスパラメタに対して最大の値を割り当てた葉となる．

したがって，処理時間の下限は式 (4) で表される．ここで，$\boldsymbol{Arch}_{crnt}$ および $\boldsymbol{Arch}_{large}$ はそれぞれ，探索中の点が表す，いくつかのパラメタが未定義のバスアーキテクチャ，探索中の点で未定義のパラメタに対して最大値を割り当てたバスアーキテクチャを表す．

$$lower\_time(\boldsymbol{Arch}_{fb}, \boldsymbol{Arch}_{crnt}) = time(\boldsymbol{M}_{sl}, \boldsymbol{G}_{sl}, \boldsymbol{Arch}_{fb}, \boldsymbol{Arch}_{large}) \quad (4)$$

探索木の子孫のうち，処理時間が最短の葉が明らかであり，その葉の処理時間が探索済みの葉の最短処理時間よりも長い場合，子孫には探索済みの葉の最小処理時間よりも短い処理時間を持つバスアーキテクチャは含まれない．そこで，式 (5) を満たす点の子孫を枝刈りする．ここで，$\boldsymbol{Arch}_{fast}$ は，探索済みの葉のうち，最短処理時間を持つ葉が表すアーキテクチャである．

$$lower\_time(\boldsymbol{Arch}_{fb}, \boldsymbol{Arch}_{crnt}) > time(\boldsymbol{M}_{sl}, \boldsymbol{G}_{sl}, \boldsymbol{Arch}_{fb}, \boldsymbol{Arch}_{fast}) \quad (5)$$

4.3.3 バスパラメタの探索順序

バストポロジが決定するまではバスの数が不明であるため，バス動作周波数およびバスビット幅は，バストポロジ探索後に探索する必要がある．

処理時間の下限による枝刈りを最大限に利用するためには，処理時間が短い傾向にあるパラメタから探索すべきである．そこで，提案手法では，処理時間に与える影響が少ない傾向にある，バッファ数の探索を最後に行う．

また，バス動作周波数とバスビット幅が処理時間に与える影響は，入力として与えられるそれぞれの候補に依存するため，どちらを先に探索すべきかは一意に決定できない．提案手法では，バス動作周波数を先に探索することとした．

以上より，提案手法では，（1）バストポロジ，（2）バス動作周波数，（3）バスビット幅，（4）バッファ数，の順序でバスパラメタを探索する．

4.3.4 各パラメタの探索順序

各パラメタでの探索順序も探索時間に大きな影響を与える．

バスでのデータ転送量が多いほどデータ転送時間は長くなるため，システムの処理時間が長くなる傾向にある．そこで，提案手法では，見積りのためのプロファイリングによって取得された値を用いて，データ転送量の多いチャネルおよびバスから順にパラメタの割り当てを行う．

また，バスの数，バス動作周波数，バスビット幅，およびバッファ数が多いほど，処理時間は短くなる傾向にあるため，提案手法では，それらの値が大きいものから順に割り当てる．

値が大きいものから順に探索することで，バス動作周波数およびバスビット幅探索において式 (5) を満たす場合，探索前の兄弟は，探索中の点よりも小さい動作周波数およびビット幅を持つため，最適解を含まない．したがって，探索前の兄弟も枝刈りする．

5. 実　　験

提案するバスアーキテクチャ最適化手法の有効性を示すため，評価実験を行った．全探索と分枝限定法における探索時間と探索点数を比較した．

本実験では，提案手法を音声映像圧縮システムに適用した．対象システムのシステムレベルモデルを図 6 に示す．対象システムは，7 個のプロセスと 6 本のチャネルを持つ．また，対象とした機能ブロック群構成を図 7 に示す．対象機能ブロック群構成は 7 個の機能ブロックを持ち，プロセスは 1 対 1 に機能ブロックにマッピングされる．バス動作周波数およびバスビット幅の候補はともに 2 種類，最大バッファ数は 2 とした．

面積制約を 10 mm^2 とした場合の実験結果を表 1 に示す．「従来」および「提案」はそれぞれ，アーキテクチャレベルシミュレーションによる見積り，[2] で提案した高速性能見積り手法を用いた見積りを表す．ここで，アーキテクチャレベルシミュレーションの見積り時間は，実験結果から取得した平均値である 20 秒を用いた．「探索バスアーキテクチャ数」，「探索点数」，および「探索時間」はそれぞれ，バスパラメタ探索時にトレースした葉の数，トレースした点の総数，探索に要した時間を表す．本実験は，Pentium 4（2.4 GHz）プロセッサ，768 MB メモリの PC 上で行い，プログラムのコンパイルには GNU コンパイラ gcc–2.96 を用いた．最適バスアーキテクチャのハードウェア面積と処理時間の見積り結果はそれぞれ，9.09 mm^2，18.4 ミリ秒であった．

実験結果より，全探索と比較して，分枝限定法は探索時間を

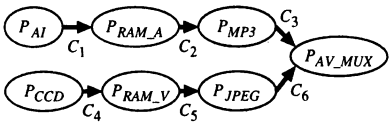


図6 実験対象のシステムレベルモデル

Fig. 6 System-level model of the experimental target system.

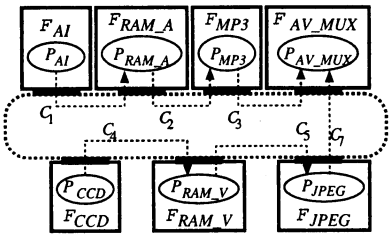


図7 実験対象の機能ブロック群構成

Fig. 7 Functional block set of the experimental target system.

大幅に減少させることが知られた．しかし，分枝限定法を用いたとしても，従来の性能見積り手法を用いた場合は探索時間が非常に長くなってしまうことがわかる．以上より，高速見積り手法を用いたバスアーキテクチャ最適化手法は，最適なバスアーキテクチャを短時間で発見できると言える．

表1 探索時間と探索バスアーキテクチャ数の比較

Table 1 Comparison of exploration time and the number of explored bus architecture.

探索方法	見積り方法	探索バスアーキテクチャ数	探索点数	探索時間
全探索	従来	2,710,784	347,022,302	> 627 日
	提案	2,710,784	347,022,302	> 3 日
分子限定法	従来	540,672	1,083,242	> 250 日
	提案	540,672	1,083,242	14.42 分

6. おわりに

本稿では，システムレベルのプロファイリング結果を用いたバスアーキテクチャ最適化手法を提案した．バスアーキテクチャ最適化問題を定式化し，バスパラメタの最適化手法を提案した．実験結果より，提案手法は最適バスアーキテクチャを短時間で探索可能であることが知られた．

今後の課題は，バスのハードウェア面積の見積り，消費電力見積り，および，各種設計品質見積りを用いたアーキテクチャ最適化手法の開発である．

謝　辞

本研究を進めるにあたり貴重なコメントをいただいた，鶴岡工業高等専門学校の佐藤淳助教授，大阪大学今井研究室の諸氏に感謝する．本研究の一部は，(株) 半導体理工学研究センターとの共同研究による（研究番号 202）．

文　献

[1] D.D. Gajski, "IP-based methodology," Proc. 36th Design Automation Conference (DAC1999), pp.43, June 1999.

[2] K. Ueda, K. Sakanushi, Y. Takeuchi, and M. Imai, "Architecture-level performance estimation for IP-based embedded systems," Proc. Design Automation and Test in Europe 2004 (DATE 2004), pp.1002–1007, February 2004.

[3] F. Balarin, L. Lavagno, C. Passerone, A.Sangiovanni-Vincentelli, Y. Watanabe, and G. Yang, "Concurrent execution semantics and sequential simulation algorithms for the Metropolis meta-model," Proc. of the 10th International Symposium on Hardware/Software Codesign (CODES2002), pp.13–18, May 2002.

[4] E.A. Lee, "Overview of the Ptolemy project," Technical Memorandum UCB/ERL M01/11, March 2001.

[5] M. Takahashi, H. Miyajima, and M. Fukui, "An efficient power and performance evaluation method with reconfigurable bus architecture Model," Proc. 11th Workshop on Synthesis and System Integration of Mixed Information Technologies (SASIMI2003), pp.345–350, April 2003.

[6] S. Yoo, G. Nicolescu, L. Gauthier, and A.A. Jerraya, "Automation generation of fast timed simulation models for operating systems in SoC design," Proc. Design Automation and Test in Europe 2002 (DATE 2002), pp.620–627, March 2002.

[7] K. Lahiri, A. Raghunathan, and S. Dey, "Design space exploration for optimizing on-Chip communication architectures," IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems, vol.23, no.6, pp.952–961, June 2004.

[8] SystemC, http://www.systemc.org/

[9] Synopsys, http://www.synopsys.com/