Vol. 45 No. 5 情報処理学会論文誌 May 2004

コンフィギュラブル VLIW プロセッサの HDL 記述生成手法

小林悠記[†] 小林真輔[†] 坂主圭史[†]
武内良典[†] 今井正治[†]

本稿では，コンフィギュラブルな VLIW プロセッサのアーキテクチャモデルに基づいたプロセッサ仕様記述から，合成可能な HDL 記述を生成する手法を提案する．本生成手法では，パイプラインステージの数やスロットの数，発行された命令を適切なリソースに割り付けるディスパッチ処理のルールなどを変更することが可能である．これらの変更パラメータや各命令の動作は，プロセッサ仕様記述中に表現されており，VLIW プロセッサのパイプライン制御を含む制御部，命令デコーダ部，データパスはこのプロセッサ仕様記述から生成される．提案手法で用いるプロセッサ仕様記述の記述量は小さく，変更量もまた小さいため，提案手法を用いることで，設計者は効果的に設計空間探索をすることができる．評価実験では，約 8 時間で 36 種類のプロセッサ仕様を記述でき，その記述量は生成される HDL 記述の記述量と比較して約 82%削減できることを確認した．

HDL Generation Method for Configurable VLIW Processor

Yuki Kobayashi,[†] Shinsuke Kobayashi,[†] Keishi Sakanushi,[†]
Yoshinori Takeuchi[†] and Masaharu Imai[†]

This paper proposes a synthesizable HDL code generation method using a processor specification description based on a configurable VLIW processor model. The proposed approach can change the number of pipeline stages, the number of slots, and a dispatching rule that manages issued operations assigned to resources, and these parameters and each instruction behavior are represented in the processor specification description. The control logic including the pipeline controller, the decode logic, and the data path for VLIW processor are generated from the processor specification. Designers can explore ASIP design space using the proposed approach effectively, because the amount of description is small and the modification cost is also small. Using this approach, it took about eight hours to design 36 VLIW processors. Moreover, this approach provides a 82% reduction on the average compared to the description of the HDL code.

1. はじめに

組み込みシステムの開発においては，ハードウェア面積，処理性能，消費電力などの厳しい設計制約を満たしたシステムを，短期間で設計する必要がある．また，機能の追加や削除をともなう仕様の変更にも柔軟に対応できることも望まれる．一般に，組み込みシステムは，ASIC（Application Specific Integrated Circuit）や，汎用プロセッサ，または ASIP（Application Specific Instruction-set Processor）を用いて実装される．ASIC を用いる実装では，高いコスト・パフォーマンスを実現できるが，設計完了後の仕様変更に柔軟に対応することはできない．汎用プロセッサを用いる実装では，開発コストを抑えることができるが，目標とする設計制約を満たすことが難しいことが多い．ASIP を用いる実装では，ASIC ほどのコスト・パフォーマンスは得られないが，ソフトウェアを変更することで，機能の追加・削減などの仕様変更にも柔軟に対応できる．すなわち，消費電力の制約や仕様変更の要求がある組み込みシステムの実装には，ASIP が適している．

命令セットプロセッサの性能の向上を図るアーキテクチャとしては，命令レベルの並列性に着目した，スーパースカラ・アーキテクチャ[1)]，VLIW アーキテクチャ[2)]が知られている．スーパースカラ・アーキテクチャでは，並列処理可能な命令をプロセッサでの実行時に動的に抽出，スケジューリングするのに対し，VLIW アーキテクチャでは，並列処理可能な命令の抽出およびスケジューリングはコンパイラによって

† 大阪大学大学院情報科学研究科情報システム工学専攻
Department of Information Systems Engineering, Graduate School of Information Science and Technology, Osaka University

行われる．VLIW アーキテクチャでは，アーキテクチャごとに並列処理可能な命令の組合せが異なるため，VLIW アーキテクチャ間の実行オブジェクトコードレベルでの互換性は低いが，動的に命令の抽出やスケジューリングを行う回路が必要ないため，スーパースカラ・アーキテクチャよりも簡単なハードウェアで実現することができる．すなわち，ハードウェア面積が制限されていることが多い組み込みシステムにおいては，VLIW アーキテクチャが適している．

VLIW アーキテクチャでは，実行オブジェクトコードの肥大化や，配線の複雑化という構造的な問題が指摘されている．これらの問題を解決するために，処理性能を維持しながら，命令発行スロットを必要最低限の数に抑える手法が知られている．しかし，適用アプリケーションによって命令レベルの並列性などが異なるために，スロットの数やハードウェアリソースの数や種類などのパラメータは，適用アプリケーションの実行サイクル数などに大きく影響を与える．また，パイプラインステージの構成はプロセッサの最大動作可能周波数やハードウェア面積に直接影響する．したがって，適用アプリケーションに適した，高性能かつ低消費電力のプロセッサを設計するためには，広大な設計空間を探索する必要がある．さらに，VLIW プロセッサの設計においては，ディスパッチルールの決定は特に重要な要素である．これは，適用アプリケーションでは使用されないオペレーションの組合せを実行する機能などは，回路を複雑にするだけだからである．このように，一定期間内により多くの VLIW アーキテクチャを生成，評価する手法が強く求められている．

プロセッサの自動生成に関しては，多くの研究がある．Yang らが提案する MetaCore は，DSP 用 ASIP の開発環境であり，基本命令と追加命令，およびユーザ定義命令が使用できる[3]．設計者は追加命令からの選択や，新命令の定義を行うことができるが，パイプライン段数の変更はできない．また，VLIW アーキテクチャもサポートしていない．Tensilica の Xtensa は，カスタマイズ可能な基本プロセッサコアを備え，設計者は，専用の言語を用いて命令を追加することができる[4]．しかし，パイプライン段数の変更や，実行ステージ以外の構成変更はできない．大槻らの提案する手法は，基本となるプロセッサコアを用いて，VLIW プロセッサおよびコンパイラを生成するものである[5]．しかし，プロセッサコアのパラメータを変化させてカスタマイズする手法のため，適用できるアーキテクチャ範囲が制限されている．Hadjiyiannis らの ISDL は，VLIW プロセッサを視野に入れて設計されている命令セット仕様記述言語であり，コンパイラやアセンブラ，逆アセンブラ，命令セットシミュレータなどのソフトウェアツールを生成できる[6]．しかし，抽象度の高い記述からハードウェア構造の詳細を推定して HDL 記述を生成するため，設計者の意図どおりのパイプライン構成を生成することは難しい．Halambi らの提案している EXPRESSION は，仕様の詳細まで記述することができ，VLIW プロセッサも表現可能である[7),8]．また，シミュレータやコンパイラを生成することができ，最新のメモリアーキテクチャに対応した設計空間探索を行うことが可能である．しかし，リソース共有を含めた，VLIW プロセッサの合成可能な HDL 記述を生成する具体的な手法については報告されていない．Hoffmann らの提案する LISA は，パイプライン構造を考慮した記述が可能であり，VLIW アーキテクチャやスーパースカラ・アーキテクチャも設計できる[9)~11]．しかし，パイプラインストールなどの機能を実現するためには，パイプラインレジスタの制御を明示的に記述する必要がある．また，合成可能な HDL 記述を生成することは，制御部については可能であるが，データパスを完全に生成することについては報告されていない．ASIP の設計においては，パイプライン構成やステージ内の動作を自由にカスタマイズでき，さらに，人手の作業では誤りが含まれやすい制御回路を自動的に生成できる手法が望ましい．しかしながら，柔軟かつ効率的に VLIW プロセッサを生成する手法は存在していなかった．

そこで本稿では，高位のプロセッサ仕様記述から VLIW プロセッサを自動生成する，コンフィギュラブル VLIW プロセッサ生成手法を提案する．提案手法では，パイプライン段数を変更でき，パイプライン制御論理も自動的に生成される．データパスは，命令のマイクロ動作記述[12]から生成されるので，設計者の意図したプロセッサを生成することが可能である．パイプラインプロセッサの設計において，パイプライン制御論理の設計は煩雑かつバグの出やすい部分であるが，提案手法ではその部分を自動生成するので，設計者はプロセッサのカスタマイズに専念することができる．したがって，提案手法を用いることで，効果的な設計空間探索を行うことができ，用途に特化した組み込みシステムを短期間で開発することが可能となる．さらに，提案手法ではパイプラインステージとディスパッチルールが変更可能である．ASIP の設計においては，実行オブジェクトコードのサイズやハードウェア面積，処理性能に影響する，パイプライン構成やディスパッチルールを，自由にカスタマイズしたいという要求が

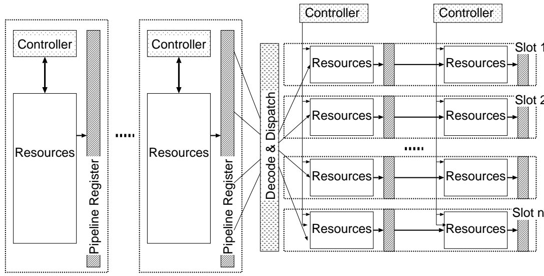
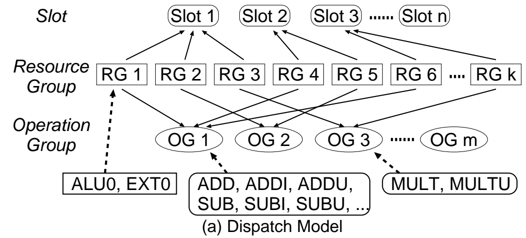


図 1　VLIW プロセッサモデル
Fig. 1　VLIW processor model.




```
slot_opegroup {
  { Slot1: OG1, Slot2: OG1, Slot3: OG3, ... },
  { Slot1: OG1, Slot2: OG2, Slot3: OG1, ... },
  ...
};
opegroup_resgroup {
  OG1: RG1, RG4, RG6;
  OG2: RG2, RG5;
  ...
};
slot_resgroup {
  Slot1: RG1, RG2, RG3;
  Slot2: RG4, RG5;
  ...
};
```

(b) Dispatch Rule Description

図 2　ディスパッチ・モデルの例
Fig. 2　An example of the dispatching model.

ある．提案手法では，これらの要素を効果的にカスタマイズすることが可能であり，しかも変更する記述量は少ない．

以下，本稿では，2 章で採用した VLIW プロセッサのモデルを説明し，3 章で提案する VLIW プロセッサの生成手法を示す．4 章で提案手法の有効性を確認するために行った実験とその結果，および考察を示し，5 章でまとめと今後の課題について述べる．

2. VLIW プロセッサモデル

本章では，Okuda らが提案した VLIW プロセッサのモデル[13)]について述べる．本稿で提案する HDL 記述生成手法ではこのプロセッサモデルを採用している．

VLIW プロセッサがフェッチする **VLIW 命令**は，同時に実行可能な複数個の**オペレーション**からなる．VLIW プロセッサは複数個のスロットを持ち，オペレーションはスロットを経て発行される．また，**ディスパッチ**処理とは，発行されたオペレーションを適切なハードウェア・リソースに割り付ける処理のことである．文献 13) のモデルを用いることにより，広い範囲のアーキテクチャの VLIW プロセッサを表現することができる．本章ではまず VLIW プロセッサのハードウェアモデルについて説明する．次にディスパッチのモデルについて説明し，最後に VLIW プロセッサ生成手法のベースとなったスカラ・プロセッサ生成手法について説明する．

図 1 に生成対象とする VLIW プロセッサのハードウェアモデルを示す．VLIW 命令のフェッチ，オペレーションのデコードなどディスパッチ前の処理は VLIW 命令全体に対するデータパスを用い，ディスパッチ後はそれぞれのスロットごとのデータパスを用いて処理を行う．各パイプラインステージ内のデータパスは，主に組合せ回路から構成されるリソース部と，次のパイプラインステージにデータを送るためのパイプラインレジスタ部から構成される．パイプラインインタロックが発生していない通常動作時にはリソース部は前段のパイプラインレジスタから入力データを受け取り，出力結果を次段へのパイプラインレジスタへと格納する．

図 2 の (a) に文献 13) で提案された VLIW プロセッサ中でのディスパッチのモデルを示す．このモデルでは，複雑なディスパッチのルールを簡潔に記述するために，**オペレーショングループ**と**リソースグループ**の概念が導入されている．オペレーショングループは，オペレーションの集合であり，オペレーションの処理にあたり同じリソースを割り当てられるなど，ディスパッチに関して同じ性質を持つ．図 2 において，オペレーショングループ OG1 のメンバは，オペレーション ADD，ADDI，ADDU，... である．リソースグループは，あるスロットから発行されるオペレーショングループ中の，オペレーションを処理できるリソースの集合である．図 2 において，リソースグループ RG1 のメンバはリソース ALU0，EXT0 である．リソースグループは，1 つのスロットと，1 つのオペレーショングループに対応づけられている．図 2 において，リソースグループ RG1 は，スロット Slot1 とオペレーショングループ OG1 に対応づけられている．リソースは複数のリソースグループに属することができる．すなわち，複数のスロットで共用されるリソースは，複数のリソースグループに属することで表現される．ディスパッチルールは，スロットとオペレーショングループの関係，スロットとリソースグループの関係，オペレーショングループとリソースグループの関係の

```
micro_operation ADD {
 wire [31:0] src0;
 wire [31:0] src1;
 wire [31:0] res;
 stage 2 {
  src0 = GPR.read0(rs0);
  src1 = GPR.read1(rs1);
 };
 stage 3 {
  wire [3:0] flag;
  <res,flag> = ALU.add(src0,src1);
 };
 stage 5 {
  null = GPR.write0(rd,res);
 };
};
```

(a)

Stage 2 GPR
Stage 3 ALU
Stage 5 GPR

(b)

図 3 マイクロ動作記述およびマイクロ動作記述から生成される DFG の例
Fig. 3 An example of micro operation description and DFG generated from the description.

3 種類を用いて記述する．上記のディスパッチモデルは，図 2 (b) のように記述される．

提案手法では，伊藤らが提案したスカラ・プロセッサ生成手法[12),14)]を拡張した．伊藤らの手法では，VLIW プロセッサではなくスカラ・プロセッサを生成対象としており，オペレーションのパイプラインステージごとの動作を記述したマイクロ動作記述から各オペレーションの DFG（Data Flow Graph）を作成し，それらの DFG を統合することで，プロセッサ全体のデータパスを作成する．**図 3** の (a) にマイクロ動作記述の例を示す．図 3 の (a) では，パイプラインステージ 2, 3，5 それぞれについて，ステージ内での処理を記述している．最初にキーワード wire を用いて src0，src1，res という 32 ビットの変数を 3 つ宣言し，ステージ 2 では汎用レジスタファイル GPR からオペランドの値を src0，src1 に読み出している．ステージ 3 では src0，src1 を ALU を用いて加算して結果を変数 res に代入し，ステージ 5 で GPR に書き戻している．図 3 の (a) のマイクロ動作記述よりリソース間接続情報を抽出し，図 3 の (b) に示す DFG を生成する．**図 4** に，DFG の統合の例を示す．図 4 の (a) と (b) はそれぞれ加算とシフトのオペレーションを表す DFG であり，これらの DFG を統合して (c) の DFG が生成される．また，提案手法は文献 14) の手法を拡張しているので，パイプラインハザードにも対応している．

文献 13) の VLIW プロセッサモデルは，オペレーショングループとリソースグループとスロットを用いて，複雑なディスパッチルールを簡潔に記述する．オペレーションはオペレーショングループへ直感的に分類され，オペレーショングループはスロットへ設計者が望むとおりに割り当てられる．リソースグループは，スロットから発行されたオペレーションを実行するのに必要なリソースの集合である．マイクロ動作記述は，

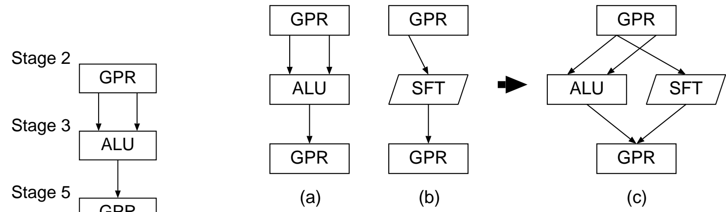


図 4 DFG 統合の例
Fig. 4 An example of merging DFGs.

リソースグループとオペレーションの組に対して記述され，文献 13) のモデルでは，幅広い範囲のディスパッチルールを記述することができる．

3. VLIW プロセッサ生成手法

本章では，まず，提案する VLIW プロセッサ生成手法の入力について説明し，次に，生成する VLIW プロセッサの概要を述べ，続いて，2 章で示したモデルに基づいた VLIW プロセッサの動作のモデルについて述べ，最後に，2 章で示したモデルに基づいた VLIW プロセッサを生成する手法について述べる．VLIW プロセッサ生成手法ではまず，オペレーショングループの組合せで表される **VLIW 命令パターン**に対してディスパッチされるリソースを決定し，次に，データパスの制御信号を生成する．ここで，VLIW 命令パターンは，プロセッサが同じデータパスを用いて処理を行う VLIW 命令の集合である．

3.1 提案手法の入力の形式的定義

本節では，提案する VLIW プロセッサ生成手法の入力について説明する．まず，ディスパッチルールについて説明し，次に入力となるプロセッサ仕様記述全体について説明する．

スロットの集合を $Slot$，リソースグループの集合を RG，オペレーショングループの集合を OG とすると，オペレーショングループとリソースグループの間の関係 $OpegResg$，スロットとリソースグループの間の関係 $SlotResg$，VLIW 命令パターン $VLIW_ptrn$ はそれぞれ

$$og \in OG, OpegResg(og) \subseteq RG, \quad OpegResg(og) \neq \emptyset, \tag{1}$$

$$s \in Slot, SlotResg(s) \subseteq RG, \quad SlotResg(s) \neq \emptyset, \tag{2}$$

$$s \in Slot, VLIW_ptrn(s) \in OG, \tag{3}$$

と表される．また，スロットとオペレーショングループの間の関係 $SlotOpeg$ は，VLIW 命令パターンの集合であり，これらを用いてディスパッチルールは

$$DispatchRule = \{SlotOpeg, OpegResg, SlotResg\} \quad (4)$$

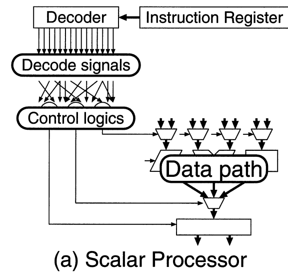
と表すことができる．

また，$Resource$ をリソースの集合，$Operation$ をオペレーションの集合，IO を入出力ポートの集合，Mod をマイクロ動作記述の集合とすると，提案する VLIW プロセッサ生成手法の入力であるプロセッサ仕様記述 $Spec$ は以下のように表すことができる．

$$Spec = \{Slot, Resource, RG, Operation, OG, IO, DispatchRule, Mod\} \quad (5)$$

3.2　VLIW プロセッサ概要

図 5 に，スカラ・プロセッサと VLIW プロセッサの制御の方法を示す．図 5 (a) に示すように，スカラ・プロセッサでは命令レジスタ中の命令をデコードすることで生成されるデコード信号を用いて，データパスを制御する．対して VLIW プロセッサでは，図 5 (b) に示すように，命令レジスタの値から**リソースグループとオペレーションの組に対するデコード信号** $Dec_{rg,ope}$ が生成され，その信号の組合せから**スロットとオペレーショングループの組に対するデコード信号** $Dec_{slot,opeg}$ が生成される．$Dec_{slot,opeg}$ の組合せから **VLIW 命令パターン検出信号** $InstPattern_n$ が生成され，VLIW 命令パターン検出信号を組み合わせて**リソースグループアクティブ信号** $Actv_{rg}$ が生成される．データパスは $Actv_{rg}$ と $Dec_{rg,ope}$ によって制御される．リソースグループとオペレーションの組に対するデコード信号は，命令レジスタ中のオペレーションを識別する信号である．前述のように，マイクロ動作記述はリソースグループとオペレーションの組に対して書かれているので，この信号はマイクロ動作記述と 1 対 1 に対応する．たとえばリソースグループ $RG1$ で処理するオペレーション ADD が存在した場合，そのマイクロ動作記述に対応する信号は $Dec_{RG1,ADD}$ である．スロットとオペレーショングループの組に対するデコード信号は，各スロットから発行するべきオペレーションがどのオペレーショングループに属するかを識別する信号である．スロット $Slot1$ からオペレーショングループ ALU に属するオペレーションが発行されることを表す信号は $Dec_{Slot1,ALU}$ である．VLIW 命令パターン検出信号は命令レジスタ中の VLIW 命令のパターンを識別する信号であり，たとえば 3 番目のパターンを表す信号は $InstPattern_3$ となる．リソースグループアクティブ信号は，あるリソースグループが，検出された VLIW 命令パターンに割り付けられていることを識別する信号であり，リソースグループ $RG1$ に対するアクティブ信号は $Actv_{RG1}$ となる．



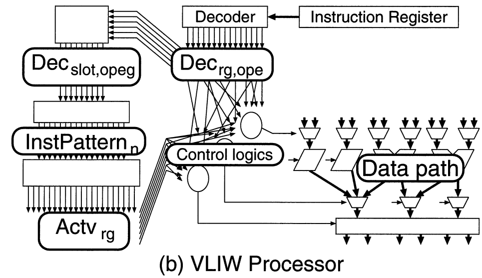


図 5　スカラ・プロセッサと VLIW プロセッサの制御の方法
Fig. 5　Control paths of Scalar processor and VLIW processor.

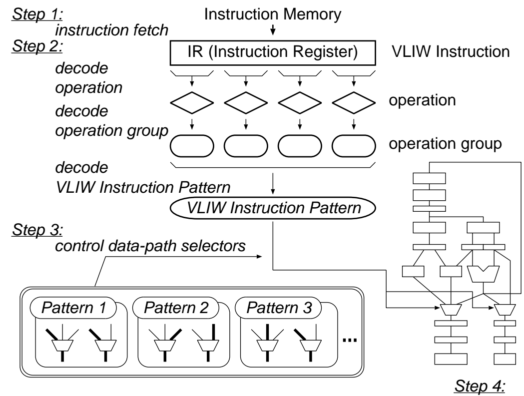


図 6　VLIW プロセッサの動作モデル
Fig. 6　Execution model of VLIW processor.

3.3　VLIW プロセッサ動作モデル

図 6 に，2 章で示したモデルに基づいた VLIW プロセッサでのオペレーションの実行処理の流れを示す．この動作モデルでは，VLIW プロセッサは以下のステップを繰り返すことによって，オペレーションをパイプライン的に実行する．

(1)　VLIW 命令のフェッチ
命令メモリから VLIW 命令をフェッチし，命令レジスタへ格納する．

(2)　オペレーションデコード
命令レジスタ中のオペレーションをデコードし，各オペレーションのオペレーショングループを

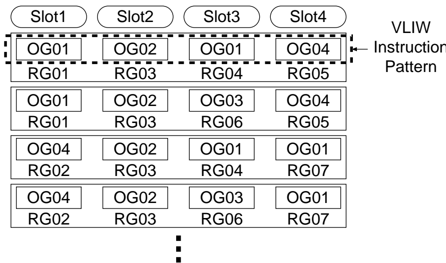


図 7　$T_{\mathcal{IDP}}$ の例
Fig. 7　An example of $T_{\mathcal{IDP}}$.

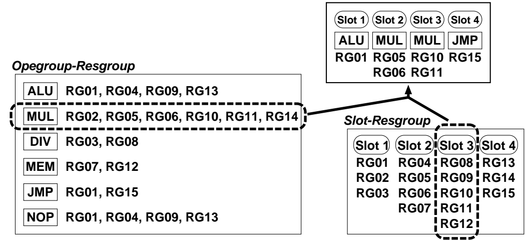


図 8　スロットごとの使用可能なリソースグループの列挙の例
Fig. 8　An example of enumeration of resource groups that are available in each slot.

取得することにより，オペレーショングループの組合せで表される VLIW 命令パターンを検出する．

(3)　データパス実現
検出された VLIW 命令パターンに対し，データパスセレクタを切り替えて，あらかじめ決めておいたデータパスを実現する．

(4)　オペレーション実行
ステップ 3 で実現されたデータパスで演算を処理する．

3.4　命令ディスパッチパターン

提案手法では，各 VLIW 命令パターンに対して，実際に使用するリソースグループに関する情報を，HDL 記述を生成する前にあらかじめ決定しておく．この情報を**命令ディスパッチパターン**と呼ぶ．命令ディスパッチパターン IDP は

$$s \in Slot, og \in OG, rg \in RG,$$
$$IDP(s) = (og, rg) \qquad (6)$$

と表され，すべての命令ディスパッチパターンをまとめた表 $T_{\mathcal{IDP}}$ は IDP の集合として表される．ここで，$Slot$ はスロットの集合，OG はオペレーショングループの集合，RG はリソースグループの集合である．**図 7** に $T_{\mathcal{IDP}}$ の例を示す．図 7 の一番上のエントリは，{OG01, OG02, OG01, OG04} という VLIW 命令パターンに対しては，リソースグループ RG01，RG03，RG04，RG05 が使用されることを示している．

VLIW 命令パターンに対するリソースグループの割付手順は次のとおりである．(1) スロットごとの使用可能なリソースグループの列挙，(2) VLIW 命令パターンに対するリソースグループ割付けの決定．本アルゴリズムは，式 (4) で表されるスロットとオペレーショングループ間の関係（VLIW 命令パターン），スロットとリソースグループ間の関係，オペレーショングループとリソースグループ間の関係を入力とし，$T_{\mathcal{IDP}}$ を作成する．

3.4.1　スロットごとの使用可能なリソースグループの列挙

VLIW 命令パターン中の各スロットに割り当てられたオペレーショングループと処理できるリソースグループを列挙する．まず，オペレーショングループとリソースグループ間の関係より，各オペレーショングループを処理できるリソースグループが算出される．次に，スロットとリソースグループ間の関係より，実際にスロットに属しているリソースグループだけを，スロットごとの使用可能なリソースグループとして，選び出す．**図 8** にリソースグループを列挙した例を示す．スロット 3 でオペレーショングループ MUL を処理する VLIW 命令パターンの場合，オペレーショングループとリソースグループ間の関係から得られたオペレーショングループ MUL を処理できるリソースグループと，スロットとリソースグループ間の関係から得られたスロット Slot3 に属するリソースグループとの共通部を選択することで，Slot3 で MUL を処理するときに使用可能なリソースグループ RG10，RG11 を決定する．

3.4.2　VLIW 命令パターンに対するリソースグループ割付けの決定

前ステップで得られた情報には，スロットに対して使用可能なリソースグループが複数個存在する場合があるため，リソースグループ間でリソースに重複がないように，使用するリソースグループを 1 つ決定する必要がある．**図 9** の場合，リソースグループ RG05 と RG10 は，ともにリソース MUL0 を含んでいるので，RG05 と RG10 を同時に使うことはできず，RG05 と RG11 か，RG06 と RG10 を使用する組合せにする必要がある．リソース重複がなければ複数の候補のうちいずれを選んでもよいものとする．

リソースグループ決定アルゴリズムを**図 10** に示す．`res_set`，`rg_set` を，初期状態が空であるリソースとリソースグループの集合とする．スロット s に含まれるリソースグループの集合を `pattern[s].res_set` と

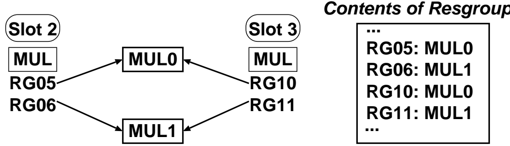


図 9　リソースの競合が発生する場合
Fig. 9　An example of a resource conflict among resource groups of slots.

```
boolean function selectResgroup(
    s: Slot,
    res_set: set of Resource,
    rg_set: set of Resource group  ) {
  each rg in pattern[s].resgroups {
    if( rg.resource not in res_set ){//no conflict
      if( s.next = null ){
        rg_set += rg;
        adopt( rg_set ); // done.
        return true; }
      else if( true = selectResgroup(
                s.next, res_set + rg.resources,
                rg_set + rg ) ) // recursive call.
        { return true; }
    }
  }
  return false; }
```

図 10　リソースグループ決定アルゴリズム
Fig. 10　Resource group decision algorithm.

し，adopt(rgset) はrgset を VLIW 命令パターンに対する割付けリソースグループとして採用する手続きを行う．pattern[s].res_set 中のリソースグループ rg について，rg に含まれるリソースrg.resource と res_set との間にリソース重複がなければ，res_set にrg.resource を，rg_set にrg を追加する．すべてのスロットについて，リソース重複が起こらないリソースグループを見つけたならば，rg_set を結果として採用し，終了する．スロット数を s，リソース数を r，リソースグループ数を g，VLIW 命令パターン数を n とすると，本生成アルゴリズムの最悪時間計算量は $O(nr^2g^s)$ となる．

3.5　ディスパッチ処理関連の信号

本節では，3.2 節で紹介した VLIW プロセッサ内でのディスパッチ処理に係る信号の生成手法について説明する．

3.5.1　スロットとオペレーショングループの組に対するデコード信号

スロット $slot$ に属すリソースグループ RG_{slot} において，オペレーショングループ $opeg$ に属するオペレーション ope を処理できるリソースグループを rg とする．このとき，スロット $slot$ とオペレーショングループ $opeg$ に対するデコード信号 $\mathcal{D}ec_{slot,opeg}$ は，ope と rg に対するデコード信号 $\mathcal{D}ec_{rg,ope}$ の論理和で表され，

$$\mathcal{D}ec_{slot,opeg} = \bigvee_{\substack{ope \in opeg,\\ rg \in \mathcal{RG}_{Slot}}} \mathcal{D}ec_{rg,ope} \wedge Exist(rg, ope) \tag{7}$$

となる．ここで，$Exist(rg, ope)$ は，リソースグループ rg で処理するオペレーション ope が存在する場合真，それ以外の場合に偽をとる関数である．

3.5.2　VLIW 命令パターン検出信号

命令ディスパッチパターン表の n 番目のエントリ $\mathcal{T}_{\mathcal{IDP}_n}$，スロットの集合 $\mathcal{S}lot$，命令ディスパッチパターン $ptrn$ のスロット $slot$ に対応するオペレーショングループ $Opegroup(ptrn_{slot})$ を用いると，n 番目の VLIW 命令パターンを検出する論理 $\mathcal{I}nstPattern_n$ は $slot$ と $opeg$ に対するデコード信号 $\mathcal{D}ec_{slot,\,opeg}$ の論理積で表され，

$$\mathcal{I}nstPattern_n = \bigwedge_{\substack{ptrn = \mathcal{T}_{\mathcal{IDP}_n}\\ slot \in \mathcal{S}lot\\ opeg = Opegroup(ptrn_{slot})}} \mathcal{D}ec_{slot,\,opeg} \tag{8}$$

となる．

3.5.3　リソースグループアクティブ信号

命令ディスパッチパターン表を $\mathcal{T}_{\mathcal{IDP}}$，リソースグループ rg の属するスロットを $\mathcal{S}lot_{rg}$，命令ディスパッチパターン $ptrn$ が検出されたことを表す信号を $InstPattern_{ptrn}$，命令ディスパッチパターン $ptrn$ のスロット $slot$ に対応するリソースグループを $Resgroup(ptrn_{slot})$ とする．このとき，リソースグループ rg に対するアクティブ信号 $\mathcal{A}ctv_{rg}$ は，VLIW 命令パターン検出信号 $InstPattern_{ptrn}$ の論理和で表され，

$$\mathcal{A}ctv_{rg} = \bigvee_{\substack{ptrn \in \mathcal{T}_{\mathcal{IDP}}\\ rg = Resgroup(ptrn_{\mathcal{S}lot_{rg}})}} InstPattern_{ptrn} \tag{9}$$

となる．

3.5.4　データパスセレクタの選択制御信号

前述のように，本モデルでのマイクロ動作記述は，オペレーションとリソースグループの組に対して書かれており，文献 12) の手法と同様に，1 つのマイクロ動作記述に対して 1 つの DFG を作成し，マージすることによりプロセッサ全体のデータパスを生成する．このとき，複数の入力が衝突するリソースの入力ポート部などにデータパスセレクタを挿入して信号衝突を回避する．

データパスセレクタにおいて，各入力ポートが選択される条件を以下に示す．リソースグループ rg で処理するオペレーション ope のマイクロ動作記述を表す

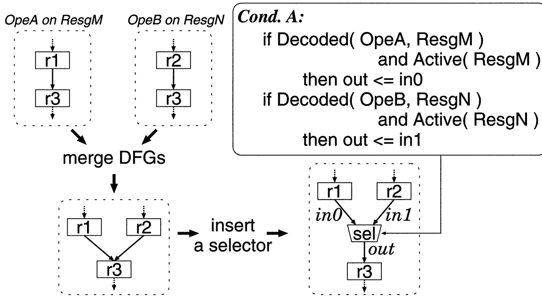


図 11 VLIW プロセッサにおける DFG のマージとセレクタの挿入

Fig. 11 DFGs merge and selector insertion in the VLIW processor generation.

DFG が実際に有効となるのは，命令レジスタに ope を表す値が入っており，かつ，rg を使用することになった場合のみである．これは rg と ope に対するデコード信号と rg のアクティブ信号との論理積で表すことができる．リソースグループ rg で行うオペレーション ope に対するマイクロ動作記述を表す DFG を $\mathcal{DFG}_{ope,rg}$ とする．このとき，$\mathcal{DFG}_{ope,rg}$ に由来する入力が選択される条件 $Cond_{\mathcal{DFG}_{ope,rg}}$ はリソースグループ rg とオペレーション ope に対するデコード信号 $\mathcal{D}ec_{rg,ope}$ と，リソースグループ rg のアクティブ信号 $\mathcal{A}ctv_{rg}$ の積で表され，

$$Cond_{\mathcal{DFG}_{ope,rg}} = \mathcal{D}ec_{rg,ope} \wedge \mathcal{A}ctv_{rg} \quad (10)$$

となる．

図 11 に DFG のマージとセレクタ挿入の様子を示す．リソースグループ ResgM で処理するオペレーション OpeA がデコードされ，かつ ResgM を使用する場合に，ResgM で処理する OpeA を表す DFG を有効にする．この論理を図 11 中 Cond.A に示す．

4. 評価実験と考察

提案手法により，VLIW プロセッサの HDL 記述を正しく生成できること，および複数のアーキテクチャを短期間で設計でき，効率的な設計空間探索を行えることを確認するために，提案手法を実装し，36 種類の VLIW プロセッサを設計した．

評価実験は次の手順で行った．(1) ベースプロセッサと，様々な数のスロットやハードウェアリソース，様々なタイプのディスパッチルールを持った派生プロセッサ計 36 種類のプロセッサ仕様記述の作成，(2) 実装したプロセッサ生成システムによる HDL 記述生成，(3) プロセッサの正常な動作を確認するためのシミュレーション，(4) 論理合成によるハードウェア面積，最大遅延時間の測定，(5) 評価用アプリケーションを用いたゲートレベルシミュレーションによる，ゲートレベルでのタイミング情報の取得，(6) ゲートレベルタイミング情報を用いた消費電力の測定である．

表 1 設計した VLIW プロセッサの変更パラメータ

Table 1 Parameters of designed VLIW processors.

パラメータ	値	パラメータ	値
スロット数	1, 2, 3, 4	ALU の数	1, 2, 3, 4
パイプラインステージ数	3, 4, 5	乗算器の数	1, 2, 3
命令語長	32, 24	除算器の数	0, 1
VLIW 命令パターン数	5 - 938	シフタの数	1, 2, 3, 4

表 2 VHDL ファイル（出力）と提案プロセッサ仕様記述（入力）の記述量

Table 2 Code size of the VHDL files (output) and the proposed specification description (input).

スロット数	VHDL ファイル (Byte)	プロセッサ仕様記述 (Byte)	入力記述量の比率 (仕様記述/VHDL)	スロット数	VHDL ファイル (Byte)	プロセッサ仕様記述 (Byte)	入力記述量の比率 (仕様記述/VHDL)
2	512264	105375	20.5704%	2	620219	123387	19.8941%
2	465844	100865	21.6521%	2	620284	123704	19.9431%
2	462918	103905	22.4457%	2	622288	124477	20.0031%
2	339238	49291	14.5299%	3	1131164	234276	20.7111%
2	273626	42252	15.4415%	2	337087	49650	14.7291%
1	252765	51438	20.3501%	3	541316	85997	15.8867%
1	207595	24783	11.9381%	3	553742	88521	15.9860%
1	183384	22684	12.3697%	3	555351	87876	15.8235%
1	229045	26875	11.7335%	3	538632	85125	15.8039%
2	512328	105694	20.6301%	3	568733	90574	15.9256%
3	857408	186003	21.6936%	3	552016	87824	15.9097%
4	1405075	306750	21.8316%	3	538918	85722	15.9063%
2	513627	106066	20.6504%	2	341613	50106	14.6675%
3	876696	189342	21.5972%	2	339462	50425	14.8544%
4	1469119	320461	21.8131%	2	341070	50797	14.8934%
4	1480101	322023	21.7568%	2	274500	43822	15.9643%
4	1480803	323560	21.8503%	2	279861	43450	15.5256%
4	1570185	344388	21.9330%				
4	1475607	330434	22.3931%		平	均	18.0447%

実験環境は，CPU：Pentium4 2.8 GHz，主記憶：512 MB，OS：RedHat Linux7.3 である．ハードウェア面積，最大遅延時間は，Synopsys 社の論理合成ツール Design Compiler と 0.14 μm CMOS ライブラリを用いて測定した．

36 種類の VLIW プロセッサは，浮動小数点演算命令を省いた DLX プロセッサ[15]の命令セットをベースとして拡張したものであり，**表 1** に示すように，スロット数，パイプラインステージ数，命令語長，演算器数，VLIW 命令パターン数などのパラメータを変化させて設計した．36 種類の VLIW プロセッサのプロセッサ仕様をわずか 8 時間で記述することができた．なお，プロセッサ仕様記述からの HDL 記述生成に要した時間は，試作したシステムでは 1 プロセッサあたり 2 秒から 15 秒程度であった．また，それぞれのプロセッサについて，入力であるプロセッサ仕様記述と，出力である VHDL ファイルの記述量の比較を行った．**表 2** に，比較結果を示す．表 2 に示すとおり，生成された VHDL 記述に対して仕様記述の記述量は 11%から 22%であり，平均すると 18%の記述量であった．記述量の比較結果より，HDL 記述を手作業で記述した場合は，少なくとも提案手法の 5 倍の時

表 3　仕様変更したときの仕様記述と VHDL 記述の変更量
Table 3　The amount of changed lines between the proposed specification description and VHDL description.

	変更前記述量 (行)	変更された記述量 削除 (行)	変更された記述量 追加 (行)	変更後記述量 (行)
仕様記述	4685	10	25	4700
VHDL 記述 (制御部)	5351	178	178	5351
VHDL 記述 (データパス)	2575	263	273	2585

間を要するものと思われる．また，プロセッサ仕様記述の作成を補助する GUI ツールを用いると，さらに効率的に VLIW プロセッサを設計することができる．

表 3 に，2 スロット VLIW プロセッサのシフタ数を 1 から 2 へ変更したときの，仕様記述と VHDL 記述の変更量を比較した結果を示す．表 3 より，仕様記述の変更量はわずか 25 行であったが，生成された VHDL 記述の変更量は，制御部とデータパスそれぞれ約 180 行，約 270 行であった．この結果より，VHDL 記述の大きな変更が必要となる仕様変更であっても，提案手法を用いることによりわずかな仕様記述の変更で対応できることが分かる．

図 12 に，生成されたプロセッサのハードウェア面積と処理性能のトレードオフ関係を示す．x 軸は生成されたプロセッサのハードウェア面積を表し，y 軸は各プロセッサの最大動作可能周波数で FIR フィルタアプリケーションを実行したときの処理時間を表している．この FIR フィルタアプリケーションを高速に処理でき，かつ低ハードウェアコストのアーキテクチャ構成を設計空間探索すると，図 12 においてプロセッサ A から F が解候補となる．この結果より，プロセッサ仕様記述を用いた提案する生成手法を用いることで，大規模な設計空間探索が可能となり，短時間で解候補を列挙できることが分かった．

図 13 に，スロット数と，データパスセレクタの数の関係を，**図 14** に，スロット数と，データパスセレクタの面積の関係を示す．図 13 では，スロット数を x 軸に，データパスセレクタの数を y 軸に，図 14 では，スロット数を x 軸に，y 軸にはデータパスセレクタの面積をとっている．図 13 より，スロット数が増加するにつれてデータパスセレクタ数も増加すること，また，図 14 より，スロット数が増加するに従ってデータパスセレクタの面積も増加することが確認できる．

5. おわりに

本稿では，コンフィギュラブル VLIW プロセッサモデルを用いた VLIW プロセッサの合成可能な HDL 記述生成手法を提案した．提案手法では，様々なディスパッチルールを表現でき，適用アプリケーションに特化した，より低コストなプロセッサを設計することができる．さらに，プロセッサ仕様記述のわずかな変更のみで様々なアーキテクチャ構成を評価できるため，より短期間で，より広範囲の設計空間探索を行うことが可能である．

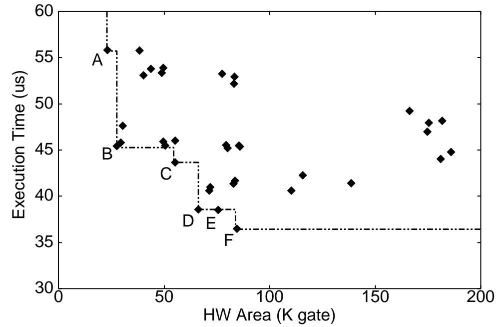


図 12　設計した 36 プロセッサのハードウェア面積と FIR フィルタアプリケーションの実行時間の関係
Fig. 12　Trade-off between HW area and execution time of FIR filter application.

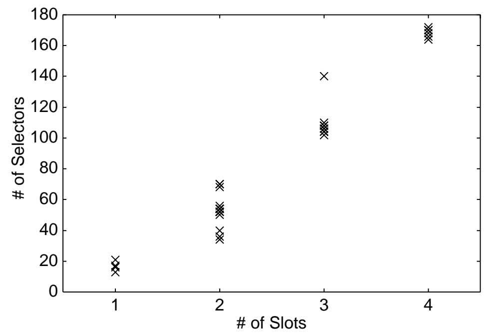


図 13　スロットとデータパスセレクタの数の関係
Fig. 13　Relation between the number of slots and the number of Data-path selectors.

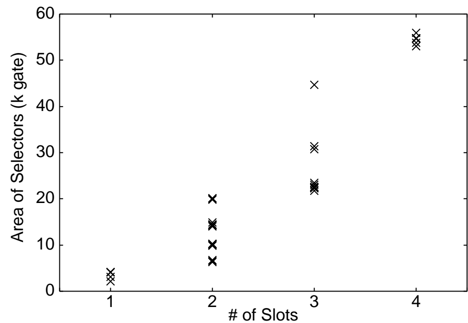


図 14　スロットとデータパスセレクタの面積の関係
Fig. 14　Relation between the number of slots and HW size of Data-path selector.

また，本稿では，提案手法を実装して，VLIWプロセッサを生成し，設計時間と設計品質について評価を行った．その結果として，様々なパラメータを持った36種類のVLIWプロセッサをわずか8時間で設計し，その仕様記述の記述量はHDL記述と比較して82%削減されたことを示した．また，様々な設計品質を持つVLIWプロセッサが生成されることを示すことにより，広大なVLIWアーキテクチャ設計空間を，短時間で効率的に探索可能であることが分かった．以上より，提案手法は，適用アプリケーションに特化した高性能なASIPを開発する際の設計コストを，大幅に削減することが知られた．

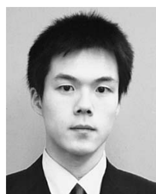
今後の課題としては，コストを考慮した回路の生成などがある．また，今回の実験に用いた実行オブジェクトコードは簡単なスケジューリングを用いたものであるので，今後の課題として適用プロセッサに応じたコンパイラの自動生成がある．

謝辞 本研究に際し，貴重なご意見をもって正しい方向へ進めるようはからっていただいた大阪大学今井研究室の諸氏に感謝する．

参考文献

1) Johnson, M.: *Superscalar Microprocessor Design*, Prentice-Hall, Inc. (1991).
2) Fisher, J.A.: Very Long Instruction Word Architectures and the ELI-512, *Proc. 10th Annual Symposium on Computer Architectures*, pp.140–150, ACM (1983).
3) Yang, J., Kim, B., Nam, S., Kwon, Y., Lee, D., Lee, J., Hwang, C., Lee, Y., Hwang, S., Park, I. and Kyung, C.: MetaCore: An Application-Specific Programmable DSP Development System, *IEEE Trans. Very Large Scale Integration Systems*, Vol.8, No.2, pp.173–183 (2000).
4) Ezer, G.: Xtensa with user defined DSP coprocessor microarchitectures, *Proc. 2000 IEEE International Conference on Computer Design: VLSI in Computers & Processors*, pp.335–342 (2000).
5) 大槻典正，武内良典，今井正治，引地信之：命令発行スロット数を考慮したVLIWプロセッサのアーキテクチャ最適化手法，情報処理学会論文誌，Vol.40, No.4, pp.1507–1516 (1999).
6) Hadjiyiannis, G., Hanono, S. and Devadas, S.: ISDL: An Instruction Set Description Language For Retargetability and Architecture Exploration, *Design Automation for Embedded Systems*, Vol.6, No.1, pp.39–69 (2000).
7) Halambi, A., Grun, P., Ganesh, V., Khare, A., Dutt, N. and Nicolau, A.: EXPRESSION: A Language for Architecture Exploration through Compiler/Simulator Retargetability, *Proc. Design, Automation and Test in Europe*, pp.485–490 (1999).
8) Mishra, P., Kejariwal, A. and Dutt, N.: Rapid Exploration of Pipelined Processors through Automatic Generation of Synthesizable RTL Models, *Proc. 14th IEEE International Workshop on Rapid Systems Prototyping*, pp.226–232 (2003).
9) Hoffmann, A., Kogel, T., Nohl, A., Braun, G., Schliebusch, O., Wahlen, O., Wieferink, A. and Meyr, H.: A Novel Methodology for the Design of Application-Specific Instruction-Set Processors (ASIPs) Using a Machine Description Language, *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, Vol.20, No.11, pp.1338–1354 (2001).
10) Pees, S., Hoffmann, A., Zivojnovic, V. and Meyr, H.: LISA — Machine Description Language for Cycle-Accurate Models of Programmable DSP Architecture, *36th Design Automation Conference*, pp.933–938 (1999).
11) Hoffmann, A., Meyr, H. and Leupers, R.: *Architecture Exploration for Embedded Processors with LISA*, Kluwer Academic Publishers, Boston (2002).
12) Itoh, M., Takeuchi, Y., Imai, M. and Shiomi, A.: Synthesizable HDL Generation for Pipelined Processors from a Micro-Operation Description, *IEICE Trans. Fundamentals of Electronics Communications and Computer Sciences*, Vol.E83-A, No.3, pp.394–400 (2000).
13) Okuda, K., Kobayashi, S., Takeuchi, Y. and Imai, M.: A Simulator Generator Based on Configurable VLIW Model Considering Synthesizable HW Description and SW Tools Generation, *Proc. Workshop on Synthesis And System Integration of Mixed Information Technologies*, pp.152–159 (2003).
14) 伊藤真紀子，塩見彰睦，佐藤　淳，武内良典，今井正治：パイプライン・ハザードを考慮したプロセッサ生成手法の提案，情報処理学会論文誌，Vol.41, No.4, pp.851–862 (2000).
15) Hennessy, J. and Patterson, D.: *Computer Architecture : A quantitative approach*, Morgan Kaufmann Publishers, Inc. (1990).

(平成15年10月22日受付)
(平成16年 3 月 5 日採録)



小林　悠記

平成 15 年大阪大学基礎工学部情報科学科卒業．同年大阪大学大学院情報科学研究科情報システム工学専攻博士前期課程入学．VLSI 設計自動化，特にプロセッサ生成に興味を持つ．IEEE 会員．

小林　真輔

平成 12 年大阪大学大学院基礎工学研究科修士課程修了．平成 15 年同大学院博士課程修了．同年 4 月大阪大学大学院情報科学研究科特任助手．同年 10 月東京大学大学院情報学環助手．工学博士．主としてコンピュータアーキテクチャ，組み込みシステム設計，HW/SW コデザイン，VLSI 設計，VLSI-CAD に関する研究に従事．IEEE，電子情報通信学会各会員．

坂主　圭史

平成 9 年東京工業大学理工学部電気電子工学科卒業．平成 11 年同大学大学院修士課程修了．平成 14 年同大学院博士後期課程修了．平成 14 年 4 月より大阪大学大学院情報科学研究科助手．博士（工学）．VLSI レイアウト設計に関する研究に従事．IEEE，電子情報通信学会各会員．

武内　良典（正会員）

昭和 62 年東京工業大学工学部電気･電子工学科卒業．平成 4 年同大学大学院博士後期課程修了．博士（工学）．平成 8 年大阪大学大学院基礎工学研究科情報数理系専攻講師．現在，同大学大学院情報科学研究科助教授．ディジタル信号処理，VLSI 設計および VLSI CAD の研究に従事．IEEE，電子情報通信学会各会員．

今井　正治（正会員）

昭和 49 年名古屋大学工学部電気工学科卒業．昭和 54 年同大学大学院博士後期課程修了（工学博士）．同年豊橋技術科学大学奉職．平成 6 年同教授．平成 8 年大阪大学大学院基礎工学研究科情報数理系専攻教授．現在，同大学院情報科学研究科教授．その間，昭和 59 年から昭和 60 年にかけて米国サウスカロライナ大学工学部電気計算機工学科客員助教授（文部省在外研究員）．これまで組合せ最適化アルゴリズム，ハードウェア/ソフトウェア協調設計等の研究に従事．平成 3 年より日本電子機械工業会および IEEE/DASC において EDA 標準化作業に従事．現在，情報処理学会設計自動化研究会主査．IEEE，ACM，電子情報通信学会，人工知能学会各会員．