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

伊藤 真紀子[†] 塩見 彰睦[††] 佐藤 淳[†††]
武内 良典[†] 今井 正治[†]

本論文では，マルチサイクル演算，遅延分岐，および外部割込みに対応したプロセッサ生成手法を提案する．本手法では，HW インタロック機構，遅延分岐機構，および外部割込み制御機構を持つパイプライン・プロセッサのモデルを用いて，クロック単位の命令のマイクロ動作記述を含む設計仕様記述からプロセッサのデータパスおよび制御論理を生成する．提案手法では，命令のマイクロ動作記述から得られる情報を基に，(1) データパスの生成，(2) マルチサイクル演算による HW インタロックの制御論理の生成，(3) 分岐の制御論理の生成，および (4) 割込み制御論理の生成を行う．MIPS R3000 のサブセット命令を有するプロセッサについて，生成結果の妥当性およびプロセッサの設計仕様記述の変更容易性を評価し，提案手法の有効性を確認した．

Processor Generation Method for Pipelined Processors in Consideration with Pipeline Hazards

Makiko Itoh,[†] Akichika Shiomi,[††] Jun Sato,[†††]
Yoshinori Takeuchi[†] and Masaharu Imai[†]

In this paper, a synthesizable HDL generation method for pipelined processors which includes multi-cycle operation, delayed branch and external interruption from clock based micro-operation description of instructions. The data path structure and control logic of the processor are generated by applying a processor model which includes hardware interlock, delayed branch and external interrupt control logic to micro-operation description. Then, (1) data path structure, (2) hardware interlock logic for multi-cycle operation, (3) branch control logic and (4) interrupt control logic are generated. Easiness of large design space exploration and effectiveness of the method was evaluated through experiments using a subset of MIPS R3000 instruction set.

1. はじめに

ASIP（Application Specific Integrated Processor，特定用途向き集積化プロセッサ）の開発では，応用分野に応じて適切な命令セットやアーキテクチャを決定する必要がある．しかし，多様なプロセッサ・アーキテクチャの中から短期間で適切な命令セットやアーキテクチャを決定することは困難である．

そこで，命令セットの仕様に基づいて特定用途向けプロセッサを生成する研究が行われている．プロセッサ生成に関する研究は，パラメータ化されたプロセッサ・コアを用いる方法と，プロセッサの仕様記述言語を用いる方法の 2 種類に大きく分けることができる．

プロセッサ・コアを用いる方法としては，PEAS-I[1]，Satsuki[2]，MetaCore[3]，CASTLE[4]などが提案されている．これらのアプローチでは，基本となる命令セットに，用意された命令またはユーザ定義の命令を追加することで用途に応じた命令セットを作成し，用途に応じたプロセッサを生成する．この方法はコンパイラ生成に適しており，記述量や設計工数が少ないが，パイプライン・ステージ数の変更や特殊な命令の追加に対応することが困難である．

プロセッサ仕様記述言語を用いた方法では，プロセッサ設計用に定義した言語を用いてプロセッサの仕様記述を行い，RT レベルの HDL 記述を生成する．この手法は，プロセッサ・コアを用いる方法と比較して，多

† 大阪大学大学院基礎工学研究科情報数理系専攻
Department of Informatics and Mathematical Science, Graduate School of Engineering Science, Osaka University
†† 静岡大学情報学部情報科学科
Department of Computer Science, Shizuoka University
††† 鶴岡工業高等専門学校電気工学科
Department of Electrical Engineering, Tsuruoka National College of Technology

様なプロセッサ・アーキテクチャに対応できるが，プロセッサ仕様記述の記述量が多く設計工数が大きい．ただし，プロセッサを汎用 HDL を用いて直接記述するよりは設計工数はかなり少ない．プロセッサ仕様記述言語を用いた方法として，AIDL[5]，PDL[6]，濱辺らの提案[7]がある．しかし，AIDL や文献 7) の手法では，設計者はプロセッサのパイプライン動作を考慮し，パイプライン・レジスタやパイプライン制御を記述する必要がある．また，AIDL，PDL，文献 7) の手法では，パイプライン・ハザードの発生する条件を求め，ハザードを解決する制御機構の自動生成は行われておらず，割込み制御機構も含まれていない．

筆者らはこれまで後者のアプローチの 1 つとして，命令のクロック単位の動作記述から論理合成可能なパイプライン・プロセッサ HDL 記述を生成する手法を提案してきた[8]．提案した手法では，命令形式の定義と使用する演算器などのリソースの宣言，および命令のクロック単位の動作記述に基づいて，パイプライン・レジスタなどを自動的に挿入し，プロセッサのデータパスとパイプライン制御機構を生成し，論理合成可能な VHDL 記述を生成する．文献 8) の手法では，設計するプロセッサの性能や面積に影響を及ぼすパイプライン・ステージ数，演算器構成，命令セットを容易に変更できる．また，論理合成可能な VHDL 記述が生成されるため，設計したプロセッサの面積や動作周波数を得ることができる．設計変更に要する工数が小さいため，設計者は短期間に多くの命令セットや演算器構成を評価でき，評価結果を比較することによって適切な命令セットや演算器構成を持つプロセッサの設計を行うことができる．

筆者らがこれまでに提案した手法[8]では，単サイクルで演算を終了する演算器を使用しており，分岐方式として分岐成立後フェッチ済みの命令をすべて実行する遅延分岐方式を採用していた．そのため，生成するプロセッサでは，パイプラインを停止しないことを仮定していた．また，割込みを持たないプロセッサを仮定していた．したがって，文献 8) の手法は，命令が単純でコプロセッサなどを持たない比較的小規模な RISC プロセッサの生成には有効であるが，乗除算などの複雑な演算やコプロセッサなどの外部モジュールの制御を行うプロセッサには適さないという問題がある．

そこで本論文では，マルチサイクル演算，遅延分岐スロット数のパラメータ化，外部割込みに対応し，マルチサイクル演算や分岐によって発生するパイプライン・ハザードに対し，HW インタロックやフェッチ済み命令の破棄などによってハザードを解決する制御機構の自動生成を行う方法を提案する．マルチサイクル演算，遅延分岐スロット数のパラメータ化，外部割込みに対応する利点は以下のとおりである．(1) マルチサイクル演算に対応することで使用できる演算器の範囲が広がる．(2) 遅延分岐スロット数をパラメータ化することで，分岐方式の選択範囲を拡張できる．遅延分岐スロットとは，分岐の成否によらず実行する後続命令が配置される分岐命令に続くスロットである．(3) 外部割込みに対応することにより，プロセッサ外部に接続されるコンポーネントを制御できる．これらの利点より，生成されるプロセッサに対して，使用する演算器の組合せ（演算器構成）や分岐方式などの点で設計可能なプロセッサの範囲を拡張でき，プロセッサの適用範囲を拡張できる．また，遅延分岐スロット数や使用する演算器の種類や個数および各演算器の実現方式を容易に変更できるため，多くの設計候補に対して論理合成可能な HDL 記述を生成し，動作周波数や面積を評価できる．したがって，評価結果を比較することで短期間で用途に応じたプロセッサの設計ができる．

本論文の構成は次のとおりである．まず，2 章では，設計者によって記述される，プロセッサの設計仕様記述について述べる．3 章では，従来のプロセッサ・モデルに対し各ステージの制御部を拡張し，HW インタロック，分岐制御を実現する方法について述べる．また，割込み制御部のモデルを定義し，外部割込み制御を実現する方法について述べる．4 章では，3 章で述べたプロセッサ・モデルに基づいて，ステージの制御部と割込みの制御部を生成し，プロセッサを生成する方法を提案する．5 章では，MIPS 社 R3000 のサブセット命令を持つプロセッサを生成する実験を行い，提案手法の有効性を評価する．6 章で提案手法について考察し，最後にまとめと今後の課題について述べる．

2. 設計仕様記述

設計仕様記述は，設計者によって記述されるアーキテクチャ・レベルのプロセッサの仕様記述である．従来より筆者らが提案してきた手法[8]では，設計仕様記述は命令形式の定義，使用するリソースの宣言，およびクロック単位の命令のマイクロ動作記述から構成されていた．本論文で提案する手法では，外部割込みの制御部を生成するために，上記の 3 項目に割込み定義を追加する．本章では，まず文献 8) で提案した命令形式定義とリソース宣言およびマイクロ動作記述について説明し，次に本論文で追加する割込み定義について説明する．

```
ALU0{
  class{"alu"},
  parameter{
    gate,
    bit_width{32},
    algorithm{cla}}}
```

図 1 リソース宣言の例
Fig. 1 Example of resource declaration.

2.1 従来の設計仕様記述

命令形式の定義では，命令コードのビットフィールドと命令固有の命令コードの値が定義される．

リソース宣言では，プロセッサ内で使用される演算器やレジスタなどのリソースの宣言が行われる．提案手法では，使用される演算器として，ALU，加算器，乗算器のように，入力データが与えられ，あるサイクル後に演算結果が出力される演算器を想定している．リソースのモデルとして，FHM（Flexible Hardware Model）[9]を用いる．FHM は，演算器やレジスタなど同一の機能を有するリソースに対して，抽象度レベル，ビット幅，演算のアルゴリズムなどがパラメータ化されたモデルである．FHM は，パラメータの値を変更することによりパラメータ値に応じたインスタンス生成を行うことができる．FHM を用いることで，パラメータ値の変更によって使用するリソースを容易に変更できる．**図 1** のリソース宣言の記述例では，ALU0 は alu というモデルで，抽象度レベルは Gate レベル，演算のビット幅は 32 bit，加算のアルゴリズムは Carry Look Ahead であると宣言されている．

命令のマイクロ動作記述では，各命令の動作がクロック単位で定義される．命令の動作記述は，(1) データの転送，(2) リソースを用いた処理（演算，書き込み，読み出しなど），(3) 演算や転送を実行する条件式の 3 要素から構成される．**図 2** に示す SLT（Set on Less Than）命令の記述例を用いて説明する．SLT 命令は，レジスタ rs の内容がレジスタ rt の内容より小さい場合レジスタ rd に 0 を，それ以外は 1 を格納する．図中の IR，IMEM などはリソース宣言で宣言されたリソースの名前である．記号 ":=" はデータの転送を表す．IR，IMEM などの記憶機能を有するリソースが右辺に現れる場合はデータの読み出し，左辺に現れる場合は書き込みを表す．第 2 ステージの "DECODE(IR);" は，レジスタ IR に格納された命令をデコードするという意味である．また，"ALU0.cmp($rs,$rt)" はリソース ALU0 を用いて比較演算 "cmp" を行い，結果を $flag に代入することを示す．第 5 ステージの if 文は条件付き実行の例となっている．

```
SLT{
 clk(1){"IR := IMEM[PC]; PC.inc();"},
 clk(2){"DECODE(IR);
         $rs := GR.read1(rs);
         $rt := GR.read2(rt);"},
 clk(3){"$flag := ALU0.cmp($rs, $rt);"},
 clk(4){""},
 clk(5){"if ($flag(1) = '1') then
           GR[rd] := \"00000001\";
         else
           GR[rd] := \"00000000\";
         end if;"}}
```

図 2 SLT 命令のマイクロ動作記述の例
Fig. 2 Example of micro-opeation description of instruction SLT.

```
Exception{
 int0{
  Condition{"INT = '1'"},
  Cycles{1},
  MOD{
   clk(1){"EPC:=PC;
           CSW:=CSW(31 downto 8)&\"00\";
           PC :=\"80000080\";"}},
```

図 3 割込み定義の例
Fig. 3 Example of exception definition.

2.2 割込み定義

割込みの定義では，各割込みについて発生条件と実行サイクル数，および処理の内容を定義する．割込みの発生条件は外部ポート名と入力される値を条件式の形で記述する．本論文では，割込み発生時にプロセッサが行う処理を割込み処理と呼び，特定レジスタの値を操作し，オペレーティング・システムなどが提供する割込み処理ルーチンに分岐するなどの一連の処理を指す．割込み処理の内容は，クロック単位のマイクロ動作記述として定義する．**図 3** に，int0 という割込みの記述例を示す．同図は，外部ポート INT に '1' が入力された場合に int0 割込みが発生し，1 サイクルでプログラム・カウンタ PC の内容を例外プログラム・カウンタ EPC に退避，割込み処理ルーチンのアドレスへ分岐，および制御ワードレジスタ CSW へ割込み原因などを格納する処理を行うことを表している．

3. プロセッサ・モデル

本章では，まず，筆者らが提案していたプロセッサ・モデルと生成できるアーキテクチャの限界について述べる．次に，本論文で行ったプロセッサ・モデルの拡張について述べ，HW インタロック，分岐制御および外部割込みに対応する方法について説明する．

3.1 従来モデル

文献 8) では，与えられるステージ数や各ステージでの処理内容に対して柔軟に対応できる単相同期方式のパイプライン・プロセッサのモデルを提案した．生

成するパイプライン・プロセッサのステージ数や各ステージで実行する処理の内容は，設計仕様記述として設計者から与えられるため，ステージ数や処理内容に対する柔軟性が要求される．プロセッサ・モデルは，**図 4** のパイプライン・ステージの集合で構成される．プロセッサ生成では，必要なステージの数だけパイプライン・ステージを連結することで，与えられたステージ数を持つプロセッサを生成する．

図 4 の任意の 1 ステージのデータパスと制御部のモデルは，演算器やレジスタなどのリソース “Resource” と有向枝で表されるリソース間の接続，パイプライン・レジスタ “Pipeline Register” およびステージの制御部 “Ctrl” から構成される．制御部は，実行中の命令の種類に応じて，リソースを制御する制御信号を出力する．命令の動作はステージごとに，(1) データの転送，(2) リソースを用いた演算などの処理，および (3) 演算や転送を実行する条件式の 3 つの要素を用いて記述される．(1)〜(3) の記述を図 4 のモデルに対応付けることで，パイプライン・プロセッサを生成する．リソースを用いた演算などの処理はリソースと，制御信号を出力する制御部によって実行される．また，リソース間のデータ転送はリソース間の接続に対応付けられる．また，制御部は条件式を評価して転送路やリソースの制御信号を選択することで，条件式による実行制御を実現する．

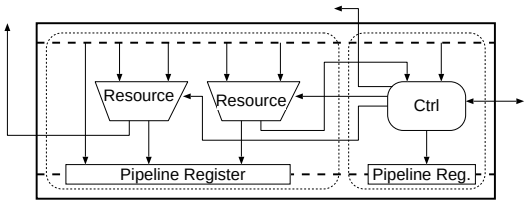


図 4 パイプライン・ステージ
Fig. 4 Pipeline stage model.

図 4 のモデルでは，各ステージは伝搬された命令を順次実行し，次ステージに結果を伝搬させることで，フェッチされた命令を順次実行することができる．しかし，命令の実行停止や破棄を行う機構を持たないため，マルチサイクル演算や分岐などによるパイプラインの乱れに対応できない．また，割込みに対応する機構を持たない．

3.2 モデルの拡張

本論文で提案する手法では，前節で述べたプロセッサ・モデルを拡張し，マルチサイクル演算，遅延分岐，および外部割込みを扱えるようにする．まず，図 4 のパイプライン・ステージの制御部を拡張し，命令の実行や実行中の命令の次ステージへの伝搬を停止する機構および命令の破棄を行う機構を追加する．命令の次ステージへの伝搬を停止することで，マルチサイクル演算時に後続命令をインタロックできる．また，命令を破棄することで指定された遅延分岐スロット数に応じた分岐の制御を可能にする．さらに，プロセッサ・モデルに割込み制御部のモデルを追加し，外部割込みに対応する．**図 5** に 5 ステージのパイプライン・プロセッサの例を示す．図 5 の例では，プロセッサは連結された 5 つのステージ（図 4）と割込み制御部から構成される．

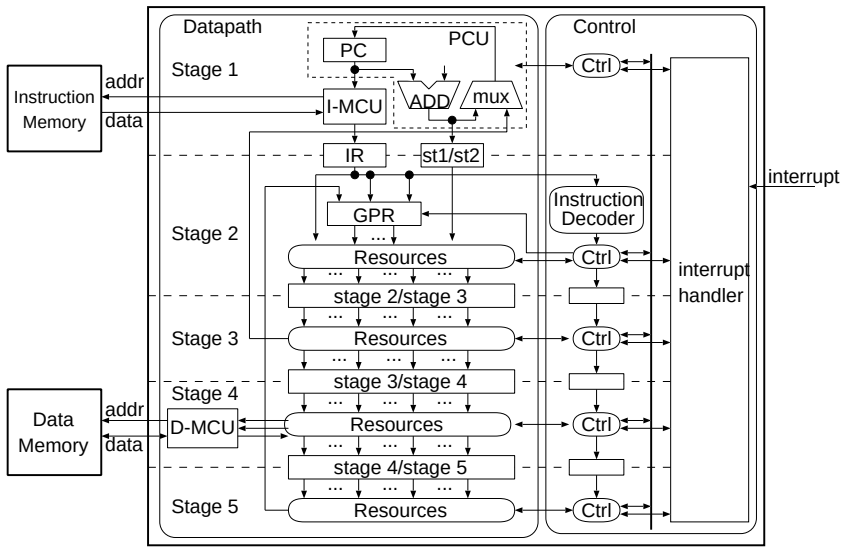


図 5 生成されるプロセッサの構造
Fig. 5 Structure of generated processor.

拡張されたモデルで生成できるプロセッサ構成は，次の特徴を持つ．(1) 同時に発行する命令数は 1 で，in-order 発行，in-order 完了する．(2) 外部割込み，マルチサイクル演算器を持ち，遅延分岐などの分岐制御を行える．

3.3 ステージの制御部

本節では，まず，命令の実行と次ステージへの命令の伝搬および実行中の命令の破棄などの制御を行うステージの制御部について述べる．次に，ステージ制御部を用いて HW インタロックを実現する方法を，さらに，分岐制御を実現する方法を説明する．

3.3.1 ステージ制御部のモデル

設計仕様記述によって与えられるパイプライン・ステージ数を n とし，ステージ番号を k $(1 \leq k \leq n)$ で表すことにする．第 k ステージの制御部は，命令の伝搬や破棄を制御する順序回路 $M_k = (q_k, I_k, O_k, \delta_k, \rho_k)$ と，データパスの各リソースへの制御信号を出力する制御信号出力関数で構成される．

順序回路 $M_k = (q_k, I_k, O_k, \delta_k, \rho_k)$ の各項 q_k，I_k，O_k，δ_k，ρ_k を，次のように定義する．ここで，命令の次ステージへの伝搬などを行う制御論理は文献 10) で提案された制御論理を参考に作成し，分岐制御時の命令破棄や割込み時の命令フェッチの停止に拡張した．

状態変数： $q_k \in \{0, 1\}$

入力信号集合：

$$I_k \triangleq \{branch, lock_k, go_{k-1}, go_{k+1}, valid_{k-1}, valid_{k+1}\}$$

出力信号集合： $O_k \triangleq \{valid_k, go_k\}$

状態遷移関数：

$$\delta_k(q_k, branch, lock_k, go_{k-1}, go_{k+1}, valid_{k-1}, valid_{k+1}) \triangleq (\overline{branch} + \overline{cancel(k)}) \cdot (valid_{k-1} \cdot go_{k-1} + q_k \cdot (lock_k + valid_{k+1} \cdot \overline{go_{k+1}}))$$

出力関数集合： $\rho_k \triangleq \{\rho_{valid_k}, \rho_{go_k}\}$

$$\rho_{valid_k}(q_k) \triangleq q_k$$

$$\rho_{go_k}(q_k, lock_k, valid_{k+1}, go_{k+1}) \triangleq q_k \cdot \overline{lock_k} \cdot (\overline{valid_{k+1}} + go_{k+1})$$

初期状態： 0

(1) 状態変数 q_k は $q_k = 1$ のとき，第 k ステージに実行すべき命令が存在することを示す．$q_k = 0$ のときは，第 k ステージに実行すべき命令がないことを示しており，HW インタロックや分岐後の命令の破棄などによって発生する空き状態であることを示す．

(2) 入力信号 $branch$，$lock_k$，$valid_k$，go_k は，それぞれ以下の場合に 1 となる信号である．

$branch$： 分岐が成立する．

$lock_k$ ： 第 k ステージでインタロックが発生する．

go_k ： 第 k ステージで実行中の命令が第 $k+1$ ステージに遷移する．

$valid_k$ ： 第 k ステージの制御部の状態変数 q_k の値が 1 である．

$go_0 = go_{n+1} = 1$，$valid_{n+1} = 0$ と定義する．$valid_0$ は，割込み制御部から出力される信号とする．割込み制御部は，命令フェッチを実行する場合に 1，フェッチを停止する場合に 0 を出力する．

(3) 状態遷移関数 δ_k において，$q_k^+ = 1$ すなわち，次サイクルで第 k ステージが実行すべき命令が存在する条件は，以下の条件 (a) と条件 (b) を同時に満たす場合である．ここで，q_k^+ は，状態 q_k の次状態を指す．

(a) 分岐が成立していない，または第 k ステージは分岐の成立時に命令が破棄されるステージでない．

(b) 第 k−1 ステージで実行中の命令が第 k ステージに遷移する，または第 k ステージで実行中の命令が第 k+1 ステージへ遷移しない．

関数 $cancel(k)$ はステージ番号を引数にとり，分岐が成立したときに第 k ステージの命令を破棄する場合に 1 を，それ以外は 0 を返す．関数 $cancel(k)$ の詳細は，3.3.3 項で述べる．

ここで，割込み制御部が命令フェッチを停止するために信号 $valid_0$ に 0 を出力している場合の第 1 ステージの動作について説明する．次サイクルで第 1 ステージの状態が $q_1 = 0$ になる場合は，第 1 ステージは実行されないので命令フェッチが実行されない．また，$q_1 = 1$ になる場合は，第 1 ステージでフェッチした命令が次ステージへ遷移していないため，新規命令はフェッチされない．したがって，$valid_0 = 0$ の間，第 1 ステージで新規命令はフェッチされない．

(4) 出力信号 $valid_k$ は，第 k ステージの状態変数 q_k の値を出力する．$valid_k = 1$ のとき，第 k ステージに実行すべき命令が存在することを示す．$valid_k = 0$ のときは，第 k ステージに実行すべき命令がないことを示す．

出力信号 go_k が 0 を出力するのは以下のいずれかの場合である．

(a) 第 k ステージでインタロックが発生している．

(b) 第 k+1 ステージで実行中の命令が第 $k+2$ ステージに遷移できない．

したがって，第 k ステージでインタロックが発生した場合，$go_k = 0$ より，第 1〜第 $k-1$ ステージについても $go_j = 0$ $(1 \leq j < k)$ となり，第 1〜第 k ステー

ジに存在する命令も次ステージに遷移しない．

制御信号出力関数

制御信号出力関数は，各命令のマイクロ動作記述の第 k ステージで記述された処理を実行するために，データパスのリソースに与える制御信号を出力する関数である．制御信号出力関数は，実行中の命令の種類とデータパスの出力信号および順序回路 M_k の出力信号 $valid_k$，go_k の関数である．$valid_k = 1$ の場合は，実行中の命令のマイクロ動作記述において第 k ステージで記述された処理の実行に必要な値を出力する．ただし，条件式 $valid_k \cdot go_k = 0$ が真となる場合，すなわち命令を実行していない場合あるいは実行中の命令が次ステージへ遷移できない場合，データパス中の記憶リソースへは，書き込みを抑制する値を出力する．

ステージ制御部の設計仕様記述に依存する項目は，信号 $lock_k$，$branch$ の論理関数，関数 $cancel(k)$，およびリソースへの制御信号の値の 4 種類である．プロセッサ生成では，命令のマイクロ動作記述を解析し，これら 4 項目の情報を抽出することで制御部を生成する．

3.3.2 HW インタロック

提案手法では，マルチサイクル演算命令を実行する場合，マルチサイクル演算が終了するまで後続命令をストールする方式を採用する．提案モデルでは，インタロックが発生したことを表す信号 $lock_k$ に 1 を出力することによって，後続命令をストールする．**図 6** の 5 段パイプラインの例では，命令 D は第 3 ステージで m サイクルのマルチサイクル演算が実行される．時刻 T において命令 D が第 3 ステージでマルチサイクル演算を実行すると，信号 $lock_3$ に 1 が出力される．信号 $lock_3$ の値が 1 へ変化すると，信号 go_3，go_2，go_1 の値は，各ステージの制御部 M_k 出力関数 ρ_{go_k} より，すべて 0 に変化する．これにより，第 1〜第 3 ステージの命令は，次ステージに遷移せずインタロックされる．第 3 ステージの命令がインタロックされるため，時刻 $T+1$ において，第 4 ステージに命令 D は伝搬されず，第 4 ステージは空き状態（$q_4 = 0$）となる．マルチサイクル演算が終了する時刻 $T+m$ まで信号 $lock_3$ に 1 が出力され，時刻 $T+m$+1 で $lock_3 = 0$ となると，go_3，go_2，go_1 はそれぞれ 1 に戻り，第 1〜第 3 ステージの命令が次ステージに遷移する．

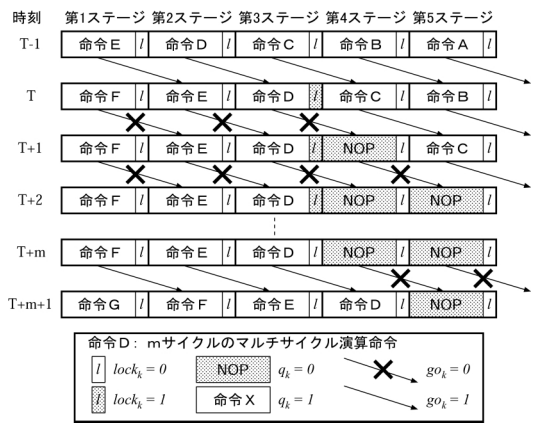


図 6 HW インタロックの例
Fig. 6 Example of hardware interlock.

3.3.3 分岐制御の実現

提案手法では，生成するプロセッサの分岐方式として，遅延分岐スロット数をパラメータ化し，分岐不成立予測と遅延分岐に対応する．設計者が指定する遅延分岐スロット数 d に応じて，分岐の成否によらず分岐命令の後続命令を d 個実行する．分岐が不成立の場合は後続命令を引き続き実行，成立した場合はフェッチ済みの $d+1$ 以降の後続命令を破棄する方式とする．なお，$d = 0$ の場合は，分岐不成立予測のみを採用したアーキテクチャとなる．

ここで，提案するモデルが仮定している項目は次のとおりである．まず，プログラム・カウンタ PC の値が分岐先アドレスに変更されるステージの番号 b を分岐ステージ番号と呼ぶと，分岐ステージ番号 b がすべての分岐命令で共通であると仮定する．また，すべての命令について第 $b-d$ ステージより前のステージでレジスタ書き込みが実行されていないものと仮定する．第 $b-d-1$ ステージ以前にレジスタ書き込みが行われると，命令を破棄する際に，書き込まれたレジスタの値を書き込まれる前の値に復元しなければならない．提案手法では，第 $b-d-1$ ステージ以前にレジスタ書き込みが行われないと仮定し，値を復元する回路の生成は行わない．また，第 1 ステージ〜第 $b-1$ ステージの間でマルチサイクル演算を行われないことを仮定する．これは，第 1 ステージ〜第 $b-1$ ステージの間でマルチサイクル演算により HW インタロックが発生すると，第 $b-d$ ステージから第 $b-1$ ステージの間で実行されている命令数が d より少なくなるからである．

命令フェッチが第 1 ステージで実行されるとすると，分岐時にフェッチ済みの命令数は $b-1$ である．指定可能な遅延分岐スロット数の範囲は，$0 \leq d < b$ となる．分岐が成立した場合に，第 i ステージ（$1 \leq i < b-d$）で実行中の命令を破棄することで上記の分岐方式を実現する．第 $i+1$ ステージの制御部は，分岐が成立すると次状態 q^+_{i+1} を 0 とすることで第 $i+1$ ステージで実行する命令を空にし，第 i ステージから伝搬され

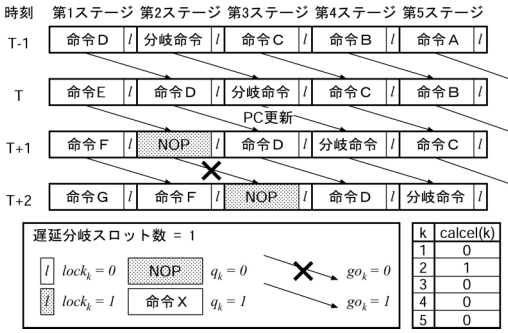


図 7 分岐成立の例
Fig. 7 Example of branch.

る命令を破棄する．したがって，ステージ番号を引数にとり，分岐成立時に命令を破棄するステージか否かを返す関数 $cancel(k)$ は $1 < k \leq b - d$ の場合に 1 を，$k = 1$ または $b - d < k$ の場合に 0 を返す関数とする．

分岐が成立した場合のパイプラインの動作の例を**図 7** に示す．図 7 では，PC が更新されるステージ $b = 3$，遅延分岐スロット数 $d = 1$ と仮定する．この場合，時刻 T の分岐が成立すると，制御部は後続命令 D を引き続き実行し，後続命令 E を破棄することになる．ここで，関数 $cancel(k)$ は，$b = 3$，$d = 1$ より，$k = 2$ の場合にのみ 1 を返す．また，時刻 T で分岐が成立すると，信号 $branch$ に 1 が出力される．状態遷移関数 δ_2 より，$branch = 1$，$cancel(2) = 1$ なので，第 2 ステージの次状態 q_2^+ は 0 となる．したがって，時刻 $T+1$ において命令 E は破棄される．なお，$cancel(3) = 0$ より，第 3 ステージの次状態 q_3^+ は 1 となり命令 D は引き続き実行される．

3.4 割込み制御部

提案手法では，外部割込みが発生した場合，実行中の命令の完了を待って割込みの処理を開始する方式を採用する．以下に具体的な割込み処理の手順を示す．

(1) 割込みの発生を検出する．
(2) $valid_0 = 0$ とすることで，第 1 ステージの制御部に対して新規命令のフェッチを停止させる．
(3) フェッチ済み命令の実行が完了するまで待機する．
(4) 定義された割込み処理を行う．
(5) 割込み処理の終了後，$valid_0 = 1$ とし，新規命令のフェッチを再開させる．

上記の手順の割込みを実現する割込み制御部は順序回路 $M_{intr} = (q_{intr}, I_{intr}, O_{intr}, \delta_{intr}, \rho_{intr})$ で実現される．q_{intr}，I_{intr}，O_{intr}，δ_{intr}，ρ_{intr} を次のように定義する．

状態変数： $q_{intr} \in \{intr, exe, wait\}$
入力変数集合：
$I_{intr} \triangleq \{interrupt, restart, complete\}$
出力変数集合： $O_{intr} \triangleq \{valid_0, int\}$
状態遷移関数：

$$\delta_{intr}(q_{intr}, interrupt, restart, complete) \triangleq \begin{cases} intr & when\ (q_{intr} = wait) \cdot complete \\ exe & when\ (q_{intr} = intr) \cdot restart \\ wait & when\ (q_{intr} = exe) \cdot interrupt \\ q_{intr} & otherwise \end{cases}$$

出力関数集合： $\rho_{intr} \triangleq \{\rho_{valid_0}, \rho_{int}\}$

$$\rho_{valid_0}(q_{intr}) \triangleq (q_{intr} = exe)$$
$$\rho_{int}(q_{intr}) \triangleq (q_{intr} = intr)$$

初期状態： $intr$

状態変数 q_{intr} は $q_{intr} = intr$ のとき割込み処理を実行中であることを，$q_{intr} = exe$ のとき命令の実行が行われていることを，$q_{intr} = wait$ のときフェッチ済みの命令が実行完了待ちであることを示す．また，$interrupt$ は外部割込みが発生した場合に，$restart$ は割込み処理が完了した場合に 1 となる信号である．$complete$ は，フェッチ済みの命令の実行が完了したことを表す信号で，各 k ステージの制御部の出力 $valid_k$ を用いて次式で表される．

$$complete = \overline{valid_1 + \cdots + valid_k + \cdots + valid_n}$$

状態遷移関数 δ_{intr} により割込み制御部の状態は次のように遷移する．まず，命令実行中に割込みが発生すると割込み処理待ち状態に遷移し（$q_{intr} = wait$），次に，フェッチ済みの命令の実行が完了すると待ち状態から割込み処理状態（$q_{intr} = intr$）に遷移する．また，割込み処理が完了すると再び命令の実行中の状態（$q_{intr} = exe$）に遷移する．

出力信号 $valid_0$ は，命令フェッチの実行を制御し，先に述べたの割込み処理の手順 (2)，(5) における命令フェッチの停止および再開を制御するために，割込み制御部が出力する．出力関数 ρ_{valid_0} は，命令実行を行う $q_{intr} = exe$ の間，信号 $valid_0$ に 1 を出力する．また，int は割込み処理を実行中であることを表す信号で，出力関数 ρ_{int} は，$q_{intr} = intr$ の間信号 int に 1 を出力する．

割込み制御部 M_{intr} を用いて上述の割込み処理を実現する方法について説明する．まず，割込みが発生すると，状態変数 q_{intr} が $wait$ となり，$valid_0$ に 0 を出力して新規命令のフェッチを停止する．第 2 ステージ以降で実行されているフェッチ済みの命令は引き続き実行する．フェッチ済みの命令の実行が完了すると，

すべてのステージについて $valid_k = 0$ となり，割込みの処理を開始する（$q_{intr} = intr$）．割込み処理開始からのサイクル数 x について，割込みの定義の第 x サイクルで記述された処理を実行するような制御信号を出力することで，実行中の割込みの処理を実現する．割込み処理が終了すると，状態変数 q_{intr} が exe となり，$valid_0$ を 1 に戻し，新規命令のフェッチを再開する．プロセッサはリセット割込みによって初期化されるため，割込み制御部の初期状態は，割込み処理の実行中を表す $q_{intr} = intr$ である．

設計仕様記述に依存する項目は，信号 $restart$, $interrupt$ の論理関数，および割込み処理の内容に応じたリソースの制御信号の値の 3 種類である．設計仕様記述の割込み定義を解析し，これら 3 項目の情報を抽出することで割込み制御部を生成する．

4. プロセッサ生成

本章では 2 章で述べた設計仕様記述と 3 章のプロセッサ・モデルに基づいて，プロセッサのデータパスと制御部を生成し，論理合成可能な VHDL 記述を出力する方法について説明する．プロセッサ生成では，最初にマイクロ動作記述を解析し，プロセッサのデータパスを生成する．同時に，ステージの状態と命令の種類および条件式の値に応じてリソースへ与える制御信号の値を求める．また，信号 $lock_k$，$branch$ の論理関数と関数 $cancel(k)$ を決定し，ステージの制御部である順序回路 M_k を生成する．さらに，割込みの定義に基づいて信号 $restart$，$interrupt$ の論理関数を決定し，各割込み処理の内容に応じた制御信号の値を求め，割込み制御用の順序回路 M_{intr} を生成する．なお，マイクロ動作記述に基づくプロセッサのデータパスおよび制御信号の値の生成方法は，文献 8) で詳細に述べられているため，ここでは省略する．本論文では，ステージの制御部および割込み制御部の生成方法について述べる．

4.1 ステージ制御部の生成

ここでは，ステージ制御部の生成に必要となる信号 $lock_k$，$branch$，および関数 $cancel(k)$ の決定方法について述べる．

4.1.1 $lock_k$ 信号の生成

マルチサイクル演算が実行されていることを表す信号 $lock_k$ の生成は，以下の手順で行われる．まず，個々の命令に対してマルチサイクル演算の実行条件と終了条件を求める．次に，すべての命令について実行条件と終了条件をまとめ，実行条件を満たし終了条件を満たさない場合に 1 となるように信号 $lock_k$ を生成する．なお，マルチサイクル演算器の実行サイクル数が固定の場合と不定の場合の両方に対応する．

(1) 命令ごとに実行条件・終了条件を求める

マルチサイクル演算が実行される命令 $inst$ およびステージ k に対して，マルチサイクル演算の実行条件と終了条件を求める．マルチサイクル演算の終了条件として，実行サイクル数が固定の場合は実行サイクル数 m を求める．また，実行サイクル数が不定の場合は，終了条件として終了信号の出力ポート p とポート p から出力される値 v の組を求める．命令 $inst$ ごとに，実行サイクル数固定のマルチサイクル演算の終了条件の集合 MC^k_{inst} および実行サイクル数不定のマルチサイクル演算の終了条件の集合 MV^k_{inst} を求める．

$$MC^k_{inst} = \{m | m : \text{必要サイクル数}\}$$
$$MV^k_{inst} = \{(p, v) | p \in \text{ポートの集合}, v : \text{値}\}$$

また，各マルチサイクル演算終了条件 $mc^k \in MC^k_{inst}$，$mv^k \in MV^k_{inst}$ について，演算を実行する条件 $exp_{mc^k,inst}$，$exp_{mv^k,inst}$ を命令のマイクロ動作記述から求める．

(2) マルチサイクル演算の実行条件・終了条件をまとめる

各命令ごとに求められたマルチサイクル演算の終了条件に対し，各ステージ k について終了条件の和集合を求める．ステージ番号を k，命令の集合を I とする．

$$MC^k = \bigcup_{inst \in I} MC^k_{inst}$$
$$MV^k = \bigcup_{inst \in I} MV^k_{inst}$$

また，終了条件 $mc^k \in MC^k$，$mv^k \in MV^k$ について演算の実行条件の集合 $Cond_{mc^k}$，$Cond_{mv^k}$ を求める．

$$Cond_{mc^k} = \{(exp_{mc^k,inst}, inst) | \exists inst \in I : mc^k \in MC^k_{inst}\}$$
$$Cond_{mv^k} = \{(exp_{mv^k,inst}, inst) | \exists inst \in I : mv^k \in MV^k_{inst}\}$$

$Cond_{mc^k}$，$Cond_{mv^k}$ の各要素 $(exp, inst)$ は，それぞれ mc^k サイクルの演算が実行される条件，および mv^k で表される終了条件で演算を終了するような演算が実行される条件を表す．

(3) 信号 $lock_k$ を生成する

求められたマルチサイクル演算の終了条件 MC^k，MV^k および実行条件 $Cond_{mc^k}$，$Cond_{mv^k}$ に基づいて信号 $lock_k$ の論理関数を求める．サイクル数固定のマルチサイクル演算終了条件の各要素 $mc^k \in MC^k$

に対して，信号 $lock_{mc^k}$ を生成する．信号 $lock_{mc^k}$ は，マルチサイクル演算実行条件 $Cond_{mc^k}$ が真となった場合に，mc^k サイクルの間 1 を出力し，それ以外の場合は 0 を出力する．また，サイクル数不定のマルチサイクル演算終了条件の各要素 $mv^k = (p, v) \in MV^k$ について，信号 $lock_{mv^k}$ を生成する．信号 $lock_{mv^k}$ は，マルチサイクル演算実行条件 $Cond_{mv^k}$ が真となった場合ポート p の出力が v と等しくなるまで 1 を出力し，それ以外の場合は 0 を出力する．信号 $lock_k$ は，各 $lock_{mc^k}$ 信号および各 $lock_{mv^k}$ 信号の論理和として実現される．

4.1.2 分岐制御の生成

分岐成立時の命令の実行制御に用いる信号 $branch$ および関数 $cancel(k)$ の生成方法について述べる．提案手法では，プログラム・カウンタ PC に分岐先アドレスが書き込まれる場合に分岐が成立すると判断する．したがって，分岐成立条件は命令のマイクロ動作記述より PC へアドレスが書き込まれる条件 $Cond_{br} = \{(exp, inst)\}$ となり，分岐ステージ番号 b はアドレスが書き込まれるステージの番号となる．

分岐成立条件 $Cond_{br}$ は，分岐を行う命令 $inst$ と命令 $inst$ で分岐が成立する条件 exp の組の集合となる．分岐の成立を表す信号 $branch$ は，分岐成立条件 $Cond_{br}$ のいずれかの要素 $(exp, inst)$ について，実行中の命令が $inst$ かつ条件 exp が真の場合に 1 が出力される．また，実行する遅延スロット数 d は設計者によって与えられるため，分岐ステージ番号 b を用いると，3.3.3 項より，関数 $cancel(k)$ は $1 < k \leq b - d$ の場合に 1 を返し，それ以外の場合は 0 を返す関数として実現される．

4.2 割込み制御部生成

ここでは，3.4 節の割込み制御部を生成するために必要となる，設計仕様記述に依存する信号 $interrupt$, $restart$ の生成方法について述べる．外部割込みが発生したことを表す信号 $interrupt$ は，割込み定義によって与えられた各外部割込み発生条件の論理和となる．また，割込み処理の終了を表す信号 $restart$ は，状態変数 q_{intr} が状態 $intr$ 遷移し，割込み処理を開始してから，処理中の割込みに必要なサイクルが経過すると 1 を出力する信号として実現される．割込み制御部は，設計仕様記述の割込み定義で定義された動作を行うために，状態変数 q_{intr} の値が $intr$ の間，割込みの種類と割込み処理開始後のサイクル数に応じて，リソースへの制御信号を出力する．なお，割込み処理を行うためのデータパスの接続情報とリソースへの制御信号の値は，命令のマイクロ動作記述からデータパスと制御信号を求める方法と同様の方法で，割込みのマイクロ動作記述に基づいて求められる．

5. 実　　験

本提案手法の妥当性と有効性を評価するために，プロセッサ生成を行う処理系を試作し，MIPS 社の R3000 命令セット[11]のサブセット 52 命令を持つプロセッサに対して VHDL 記述の生成を行った．マルチサイクル演算器使用の効果を評価するために，設計仕様記述として単サイクル演算器（組合せ回路で実現される乗算器および除算器）のみから実現されるプロセッサ A と，34 クロックのマルチサイクル演算を行う乗算器，除算器を使用したプロセッサ B を用意し，生成されたプロセッサの比較を行った．実験では，まずプロセッサ A の設計仕様記述を作成し，次に使する演算器を変更し，プロセッサ B の設計仕様記述を作成した．

生成されたプロセッサ A，B の VHDL 記述を，既存の VHDL シミュレータ上でシミュレーションを行った結果，設計仕様記述どおりにマルチサイクル演算，遅延分岐および割込み処理が動作していることを確認した．

設計仕様記述の作成と変更に必要とした工数を**表 1** に示す．マルチサイクル演算器を使用するか，単サイクル演算器を使用するかの選択は，使用するリソースのパラメータ値を変更することで行えるため，設計の変更に要する時間は数分である．プロセッサ A の新規設計には，500 分という設計時間を要するが，演算器を変更してプロセッサ B を設計する時間は短い．プロセッサ B の生成結果の記述量は，マルチサイクル演算の制御部を含むため，ステージ制御信号を生成する制御部などの記述量の分増加している．また，生成に必要となる時間は約 15 秒（UltraSPARC 167 MHz）と短い．

生成されたプロセッサの HDL 記述を，Synopsys 社の Design Compiler を使用して論理合成を行った結果を**表 2** に，プロセッサ A，B で使用した演算器の諸元を**表 3** に示す．プロセッサ B はマルチサイクル演算器を使用したため，プロセッサ A より面積が小さく，約 9.5 倍の動作周波数で動作する．本実験より，マルチサイクル演算器に対応することにより，面積が

表 1 設計工数
Table 1 Amounts of design time and quantity of descriptions.

	記述時間	生成時間	記述量	生成結果の記述量
A	500 分	14.6 秒	1026 行	1212 行
B	+3 分	17.4 秒	1026 行	1239 行

表 2 生成されたプロセッサの諸元
Table 2 Logic synthesis results of processor designs.

	面積（Gate）			動作周波数
	全体	制御部	演算器合計	（MHz）
A	104528	987	85804	5.46
B	59896	1099	42255	52.2

〔VLSI テクノロジー社 VSC753d（CMOS 0.5 μm）ライブラリ〕

表 3 乗算器・除算器の諸元
Table 3 Logic synthesis results of multiplier and divider.

	面積	動作周波数	サイクル数
乗算器 A	26810 Gate	10.9 MHz	1
除算器 A	28554 Gate	5.52 MHz	1
乗算器 B	5820 Gate	101 MHz	34
除算器 B	5995 Gate	113 MHz	34

〔VLSI テクノロジー社 VSC753d（CMOS 0.5 μm）ライブラリ〕

小さく，動作周波数の高いプロセッサの生成が可能になったといえる．

6. 考　　察

生成されるプロセッサの範囲に対して，マルチサイクル演算器や遅延分岐スロット数のパラメータ化への対応が及ぼす影響について考察する．まず，設計するプロセッサの演算器にマルチサイクル演算器を用いると，プロセッサの面積が減少し，動作周波数が向上するが，応用プログラムの実行サイクル数が増加する．演算器の選択肢にマルチサイクル演算器が加わることで，使用できる演算器の組合せが増加するため，様々な面積，動作周波数，実行サイクル数を持つプロセッサの設計が可能になる．また，遅延分岐スロット数をパラメータ化することで，分岐方式の選択範囲が広がる．遅延分岐スロットに有効な命令を配置できるプログラムの場合は，遅延分岐スロットが存在しない場合と比較して，実行サイクル数を短縮できる．しかし，遅延分岐スロットに有効な命令を配置できない場合はNOP 命令を配置するためにコードサイズが増大する．設計するプロセッサで実行する応用プログラムの性質に合わせて，性能とコードサイズのトレードオフを考慮できる．

次に，提案手法での設計空間の探索の容易性について考察する．提案手法では，FHM を用いて使用するリソースのインスタンスを生成するため，使用する演算器の選択はパラメータ値の変更だけで行うことができる．命令のマイクロ動作記述では演算器の実行サイクル数を意識せずに命令を記述できるため，マルチサイクル演算器を含めて演算器の変更に要する時間は数分である．プロセッサの設計仕様記述に基づいた論理合成可能なプロセッサの VHDL 記述は約 15 秒で生成でき，生成された VHDL 記述の論理合成を行うと，数時間で動作周波数や面積などを得ることができる．したがって，演算器構成，遅延分岐スロット数，割込み制御といった構成要素を含めたプロセッサ構成を変化させた多くのプロセッサ構成に対し，1 構成あたりの設計 TAT（Turn Around Time）は論理合成の時間を含めて数時間となる．設計 TAT が短いため，様々な候補に対して実行時間や面積などを短期間に評価することができ，用途に応じて適切な演算器構成や遅延分岐スロット数を選択できる．

演算器構成や遅延分岐スロット数の変化に応じて，設計したプロセッサの性能を評価する場合，応用プログラムの実行サイクル数やコードサイズの評価も必要となる．実行サイクルやコードサイズの評価には，設計仕様記述に応じたコンパイラが必要となる．指定された遅延分岐スロット数に応じて命令をスケジューリングするコンパイラを用いると，遅延分岐スロット数に応じてコードサイズの増減を評価できる．また，コンパイルの結果得られるオブジェクト・コードを用いて生成されたプロセッサの VHDL 記述を既存の VHDL シミュレータ上でシミュレーションすることで，実行サイクル数を評価できる．設計仕様記述に基づいたコンパイラ生成が望まれる．

7. おわりに

本論文では，マルチサイクル演算を実現するための HW インタロック機構，パラメータ化された遅延分岐スロット数，および外部割込みを考慮したプロセッサ・モデルの拡張と，その HDL 記述生成手法を提案した．マルチサイクル演算器および遅延分岐スロット数のパラメータ化に対応することで，生成可能なプロセッサの範囲が拡大した．マルチサイクル演算器に対応することにより，生成されるプロセッサの動作周波数が向上し，面積が削減された．

提案手法では，アーキテクチャの探索範囲として，命令の追加/変更，ステージ数の変更，ステージの処理内容の変更，演算器の変更，遅延分岐スロット数の変更などを行うことができる．生成実験より，短時間で論理合成可能な VHDL 記述が生成できることが確認された．また，生成された論理合成可能な VHDL 記述から動作周波数や面積などの設計品質指標を得ることができる．したがって，様々なアーキテクチャ候補に対して設計結果の動作周波数や面積などを容易に得ることができ，用途に応じて適切なアーキテクチャ

候補を選択できる．

今後の課題として，データハザード・インタロックとデータフォワーディング回路生成やパイプライン演算器への対応があげられる．また，生成されるプロセッサの動作周波数を考慮した生成アルゴリズムの改良があげられる．さらに，設計仕様記述に基づいたコンパイラの生成が望まれる．

謝辞 本研究を行うにあたり討論していただいた豊田工業高等専門学校の仲野巧助教授ならびに大阪大学今井研究室の北嶋暁博士，檜垣茂明氏に感謝いたします．なお，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．

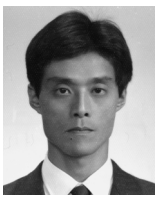
参 考 文 献

1) Sato, J., Alomary, A.Y., Honma, Y., Nakata, T., Shiomi, A., Hikichi, N. and Imai, M.: PEAS-I: A Hardware/Software Codesign System for ASIP Development, *IEICE Trans. Fundamentals*, Vol.E77-A, No.3, pp.483–491 (1994).
2) Shackleford, B., Yasuda, M., Okushi, E., Koizumi, H., Tomiyama, H. and Yasuura, H.: Satsuki: An Integrated Processor Synthesis and Compiler Generation System, *IEICE Trans. Inf. & Syst.*, Vol.E79-D, No.10, pp.1373–1381 (1996).
3) Yang, J.-H., Kim, B.-W., Nam, S.-J., Cho, J.-H., Seo, S.-W., Ryu, C.-H., et al.: MetaCore: An Application Specific DSP Development System, *35th DAC*, pp.800–803 (1998).
4) Camposano, R. and Wilberg, J.: Embedded System Design, *Design Automation for Embedded Systems*, Vol.1, Nos.1–2, pp.5–50 (1996).
5) Morimoto, T., Saito, K., Nakamura, H., Boku, T. and Nakazawa, K.: Advanced Processor Design Using Hardware Description Language AIDL, *ASP-DAC '97*, pp.387–390 (1997).
6) 鶴田三敏，安部公輝：マイクロプロセッサ記述言語 PDL に基づく設計支援システム，情報処理学会研究報告 DA 82-9, pp.65–72 (1996).
7) 濱辺雅哉，能勢 敦，戸川 望，柳澤政生，大附辰夫：パイプラインプロセッサのハードウェア記述生成手法，信学技報，VLD97-117，pp.33–40 (1997).
8) 伊藤真紀子，武内良典，今井正治，塩見彰睦：マイクロ動作からの合成可能パイプライン・プロセッサ HDL 記述生成手法の提案，第 12 回回路とシステム（軽井沢）ワークショップ論文集，pp.121–126 (1999).
9) Imai, M., Shiomi, A., Takeuchi, Y. and Sato, J.: Hardware/Software Codesign in the Deep Submicron Era, *IWLAS '96*, pp.236–248 (1996).
10) 古渡 聡，岩下洋哲，中田恒夫，広瀬文保：パイプラインプロセサの制御論理自動合成，信学技報，VLD94-41, pp.17–24 (1994).
11) Kane, G.: mips RISC アーキテクチャ– R2000/R3000，共立出版 (1992).

(平成 11 年 9 月 27 日受付)
(平成 12 年 2 月 4 日採録)

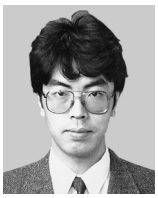
伊藤真紀子（学生会員）

平成 10 年大阪大学大学院基礎工学研究科博士前期課程修了．現在同大学院博士後期課程在学中．プロセッサ設計支援，ハードウェア/ソフトウェア協調設計手法に関する研究に従事．電子情報通信学会学生会員．



塩見 彰睦（正会員）

昭和 62 年豊橋技術科学大学情報工学課程卒業．平成 4 年同大学院博士後期課程修了．博士（工学）．平成 8 年より静岡大学情報学部情報科学科講師．設計自動化，ハードウェア/ソフトウェア協調設計手法，最適化設計支援に関する研究に従事．



佐藤 淳（正会員）

昭和 63 年豊橋技術科学大学大学院工学研究科情報工学専攻修士課程修了．博士（工学）．平成 3 年より鶴岡工業高等専門学校電気工学科助手．現在同校講師．ASIC の設計手法，ハードウェア/ソフトウェア協調設計手法に関する研究に従事．IEEE，ACM，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．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．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．