ハードウェア記述言語からの情報抽出
── コンパイラの自動生成への適用 ──

赤星博輝† 安浦寛人†

ハードウェア/ソフトウェア協調設計では，様々な設計支援が必要とされる．従来はコンパイラの生成，ハードウェア作成，高速シミュレーションモデルの作成などの支援ごとに，ハードウェアの情報を記述して支援ツールに与える必要があった．このような記述を設計者が行う場合には，1）複数の記述間での無矛盾の保証，2）設計変更の場合すべての記述を変更する必要性といった2点の問題がある．本論文では，ハードウェア記述言語による設計記述からの情報抽出という技術を提案する．情報抽出とは，ハードウェア記述言語による設計記述を本来の目的である論理合成や機能/ゲートレベル・シミュレーションに用いるだけではなく，その他の設計支援のために必要な情報を抽出する技術である．本論文では，ハードウェア記述言語によるプロセッサ記述から情報抽出を行い，そのプロセッサに対するコンパイラの生成を行った．それらの実験から，情報抽出の設計支援としての可能性を確認した．

Information Extraction from HDL Descriptions
– Application to Compiler Generation –

HIROKI AKABOSHI† and HIROTO YASUURA†

A lot of supports are required for hardware/software co-design, such as system partitioning, compiler generation (retargetable code generation), generation of a fast simulation model, and so on. These support tools require various kinds of information different each other. If a designer writes a description of information for each support, it has two major problems, 1) If the design is changed, all descriptions for support tools must be changed consistently by hand, 2) inconsistency between descriptions must be verified. We propose a technique of information extraction. Our information extraction performs the transformation from an HDL description of processor into another form of description required by each tool. This paper presents an algorithm of the information extraction and shows that a compiler generation is performed using extracted information.

1. はじめに

これまでにディジタル回路設計支援に関する研究が進み，論理合成やレイアウト合成といった技術に関しては実用的なレベルになってきた．このような技術を利用して，さらに，上位レベルでの設計支援の研究が行われている．特に，今までの設計支援のようにハードウェアだけを扱うのではなく，ハードウェア/ソフトウェアの両者を視野に入れた設計支援である“ハードウェア/ソフトウェア協調設計”に関する研究が行われている[1)~4)]．

ハードウェア/ソフトウェア協調設計では，システムの機能を，プロセッサを中心としたハードウェアとソフトウェアに分割しインプリメントを行う．システムに求められる動作だけではなく，システムに対する要求（コスト最小や実行速度など）に応じて，ハードウェア/ソフトウェアの境界を変更する必要がある．

ハードウェア/ソフトウェア協調設計に対する支援では，次の3点が重要なポイントとして挙げられる[5)]．

- 設計支援環境
- 性能評価環境
- 検証環境

設計支援環境では，今までの論理合成，レイアウト合成，シミュレーションなどのハードウェアの設計支援だけでなく，さらに，ソフトウェアの設計支援も必要となる．実際には，ソフトウェアの開発支援では，C言語などの（クロス）コンパイラ，デバッガなどが必要とされる．さらに，ハードウェア/ソフトウェアをシ

† 九州大学大学院総合理工学研究科情報システム学専攻
Department of Information Systems, Interdisciplinary Graduate School of Engineer Sciences, Kyushu University

ステムの要求に応じて分割する支援なども必要となる．

ハードウェア/ソフトウェア協調設計ではシステムに対する要求によってシステムの構成を変えるために，要求を満たすかをチェックするために性能評価が必要となる．性能評価は実用のソフトウェアを用いてハードウェアのシミュレーションによって行うことが多い．そのために，高速なシミュレーション・モデルなどが必要となる．

さらに，システムの構成を要求に応じて変えるために，検証環境が重要である．

現在，個別の技術に関しては，さまざまな研究が行われている．設計支援に関しては，アーキテクチャの自動生成[4),6),7)]，ハードウェア/ソフトウェアの分割[1),8)]，コンパイラの生成[4),5),9),10)]，ハードウェア/ソフトウェア協調設計用シミュレーション環境[3)]などの研究が行われている．

ハードウェア/ソフトウェア協調設計では，このように多くの支援が必要とされており，これらのさまざまな支援に対して，情報を与える必要が生じる．多くの場合，同じ情報を異なった形で与えることが多い．例えば，ハードウェアの情報を与えるために，ハードウェア記述言語を用いたり，コンパイラの生成などを行うための記述，あるいは高速シミュレーション・モデルを作成するための記述を別々に与える．同じ情報を異なった形で与えることの問題として，次の2点がある．

- それぞれの記述の間の無矛盾の保証
- 変更した時に，すべての記述を変更する必要がある点

本論文では，ハードウェア記述言語（HDL: Hardware Description Language）によるプロセッサの設計記述からそのプロセッサの命令の動作を抽出する手法を提案し，コンパイラの生成に適用した結果について報告する．2章で情報抽出について述べ，3章でHDLによるプロセッサの記述から，命令の動作を抽出するアルゴリズムについて述べ，4章でコンパイラの生成に適用した例を示す．

2. 情報抽出

情報抽出としては，今まで大村らがネットリストから機能回路を抽出する研究を行っている[11)]．我々は，RTレベルのハードウェア記述言語による設計記述から，さらに上位の情報を抽出する研究を行っている．実際の設計はRTレベルのハードウェア記述言語で行われることが多いため，RTレベルの設計記述から情報抽出を行うことは，有効であると考えられる．ハードウェア/ソフトウェア協調設計では様々な支援を必要とする．一つの記述から情報抽出を行い様々な支援に用いることは，1章で述べた二つの問題に対して有効である．本論文では，ハードウェア記述言語によるプロセッサ記述から，命令セットの動作を抽出する．ハードウェア記述言語によるプロセッサ記述からの情報抽出が利用できる支援としては，高速シミュレーション・モデルの生成，コンパイラの生成，ドキュメントの作成支援などがある（**図1**）．既存の汎用シミュレータのシミュレーション・スピードは，実用のソフトウェアを用いてシミュレーションを行うには遅い．RTレベルのHDLで記述されたプロセッサの記述から，より上位レベルの情報を取り出すことで，高速シミュレーション・モデルの生成を行うことができる．命令セットの動作を取り出すことで，コンパイラの生成に利用することができる．さらに，ドキュメントの作成支援のために，抽出した情報を用いることができる．今回は，その抽出した情報を用いてコンパイラを生成する．

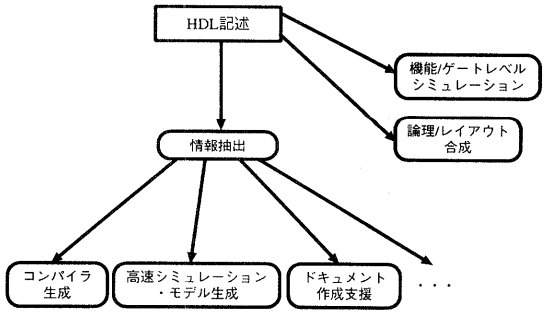

図1 情報抽出を中心とした支援
Fig. 1 Design supports with information extraction.

3. HDLからの情報抽出

本論文では，対象とするプロセッサのアーキテクチャに制約を与えて，HDLによるプロセッサ記述からの情報抽出について述べる．まず，その制約を述べた後，情報抽出時に用いるアーキテクチャの中間表現形式について説明を行い，具体的な情報抽出アルゴリズムを示す．

3.1 アーキテクチャの制約

現段階では，以下のアーキテクチャをもつプロセッサに対して情報抽出を行っている．

- 汎用レジスタ方式
- 逐次処理方式（パイプラインやスーパスカラなどは許さない）
- すべての制御は一つの有限オートマトンで行う
- 条件分岐はフラグではなく，汎用レジスタの内容

で行う

- 命令語長固定
- 命令フィールド固定
- 割込みは無し

3.2 中間表現形式

情報抽出では，HDL 入力を中間表現形式に変換し処理を行う．中間表現はツリー表現で，ARchitecture Expression in Tree（ARET）と呼んでいる．ARET は，終端ノード T，非終端ノード N，ノードを結ぶエッジ $E = \{(u,v)|u \in N, v \in N \cup T\}$ の三つで構成され，$< T, N, E >$ で定義される．

非終端ノードおよびエッジは，HDL の条件文を表現するために用いる．非終端ノード N には，ラベルとして変数とそのタイプを与え，エッジには，その変数が成立した場合の条件を与える，タイプとしては，グローバルな制御を行う信号，命令のフィールドの信号，フラグなどの信号について与える．実際には，グローバルな制御のタイプとしては，

GLB_RESET: リセット信号
GLB_PRESET: プリセット信号
GLB_WAIT: ウェイト信号
GLB_OTHER: その他のグローバルな信号

命令のフィールドの信号のタイプとしては，

IR_OP: 命令を指定するフィールドを指定する信号
IR_DST: 演算結果を収納するレジスタを指定する信号
IR_SRC$_i$: 演算のソースを収納する i 番目のレジスタを指定する信号
IR_OTHER: その他の信号を指定

終端ノードには，ルートから終端ノードに至る経路の条件が成り立つ時に，実行される HDL のステートメント群を与える．

図 2 に，ARET で表現した例の一部を示す．この図では，reset 信号が 0 で，ir<5:4>（ir という信号線の 6，5 ビット）が 00 の時，加算を行っている場合を示している．この例では，2 オペランドの計算方式をとっており，ir<3:2>でソースおよびデスティネーション・レジスタを指定し，ir<1:0>でソースレジスタを指定している．

ルートノードに reset という 1 ビットの変数があり，そのタイプとしては GLB_RESET が与えられている．そのルートノードからはエッジが 2 本出ており，条件に 0 と 1 が与えられている．条件が 0 の時のエッジは，reset が 0 の場合を表す．そのエッジからは，2 ビットの ir<5:4>の変数で IR_OP のタイプの非終端ノードが接続されている．終端ノードでは，ルートノードから

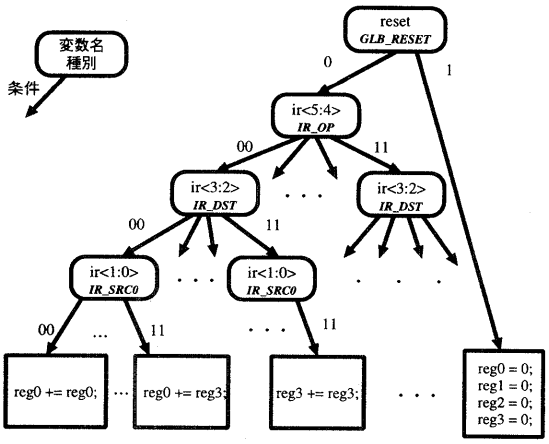

Fig. 2 ARET expression.

条件が満たされる場合に，実行されるステートメントが示される．

3.3 問題の定義

現段階では，HDL によるプロセッサ記述のみからの命令セットの抽出は困難なため，付加情報を与えることで，命令セットの動作を抽出する．

[プロセッサの命令セットの抽出]

入力： 1. HDL によるプロセッサ記述
2. 演算器の情報
3. プロセッサの基本レジスタおよびメモリの指定（プログラム・カウンタ（PC），命令レジスタ（IR），演算用レジスタの種類と構成要素，メモリ）
4. ARET を作成する際の，各変数のタイプおよび，その ARET を構成する時の変数の順序（ルートからの位置）

出力： 命令の動作

命令 i の動作 f_i は，次のように定義される．

$$R_{t+1} = f_i(R_t).$$

（命令サイクル t の時の，基本レジスタとメモリの内容を R_t とする）

入力の 2，3，4 が付加情報である．

3.4 抽出アルゴリズム

情報抽出の流れを**図 3** に示す．入力として与えられた HDL の記述を，3.2 節で述べた ARET 形式に変換する．

Step1: ターゲットとするプロセッサは有限オートマトンで制御される．有限オートマトンは有限個の状態（ステート）を持ち，各ステートで実行されるハードウェアの動作がステートメント群によって記述されるために，各ステー

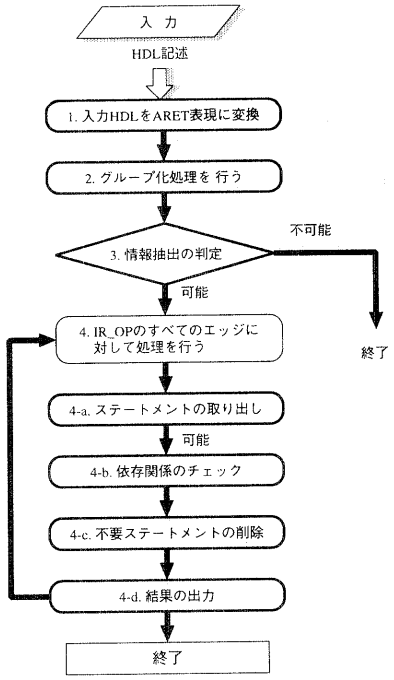

図 3 情報抽出の流れ
Fig. 3 Information extraction flow.

トごとに ARET のツリー表現を作成する.

Step2: 汎用レジスタ，浮動小数点レジスタは，個別のレジスタとして扱うよりも，グループとして扱った方が良いものについては，この段階で個別のレジスタに対する操作でなくグループに対する操作に変更する．これを，グループ化処理と呼ぶ.

Step3: 情報抽出が不可能であるか判定を行う（判定の方法は，3.4.3 項で述べる）.

Step4: すべての IR_OP から出ているすべてのエッジに対して，*Step4-a*〜*Step4-d* の処理を行う.

Step4-a:注目している IR_OP と，同じ条件を持つエッジの終端ノードのステートメントを取り出す.

Step4-b:取り出したステートメント間の依存関係を調べる．もし，依存関係があるなら，変数の代入展開を行うことで，新しいステートメントを作成する.

Step4-c:*Step4-b* で新たにステートメントを作成することで冗長なステートメントが生じたり，すべての命令で実行されるステートメントがある．そのために，不要なステートメントを削除する.

Step4-d:*Step4-c* が終った段階で，残っているステートメントを結果として出力する.

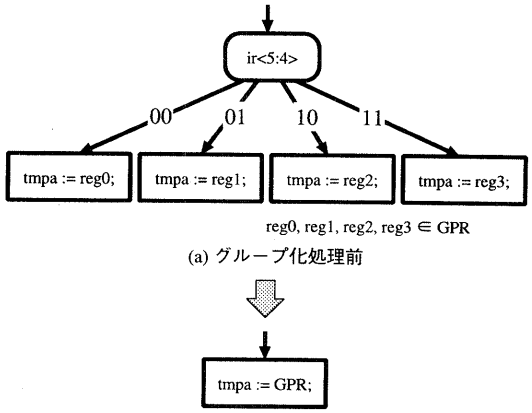

図 4 グループ化処理
Fig. 4 Grouping process.

それぞれについて，説明する.

3.4.1 ARET 表現への変換

現段階の ARET では，一つのツリー表現で，一つのステートしか表現できない．そのために，複数のステートを表現するために，複数のツリーによって表現する.

3.4.2 グループ化処理

汎用レジスタなどを個別に扱うよりも，まとめてグループとして扱うことでツリーを小さくでき，次に述べる情報抽出の可能性判定を容易に行うことが可能となる．このグループ化処理について**図 4** で説明する．図 4 (a) がハードウェア記述言語による記述から ARET 表現へ変換した段階である．ここで，reg0, reg1, reg2, reg3 が汎用レジスタ要素であるとする．この時，非終端ノード（ir<5:4>）は，汎用レジスタの選択だけに使われているだけである．この場合には汎用レジスタからの代入のステートメントを持つ終端ノードのみ（図 4 (b)）に変換する.

3.4.3 可能性判定

現段階のアルゴリズムでは，情報抽出が必ず可能であることを保証していない．また，記述の仕方によっては，情報抽出ができない場合があるため，この段階で，情報抽出が可能か判定を行う.

ターゲット・アーキテクチャが有限オートマトンによって制御されるために，複数のツリーができる．その各ツリーに対して，以下の判定を行う.

- ツリーの非終端ノードで IR_OP のタイプを持つものを選び出す.
- タイプが IR_OP の非終端ノードの次のノードは必ず終端ノードでなければ，現段階の情報抽出アルゴリズムでは抽出はできない.

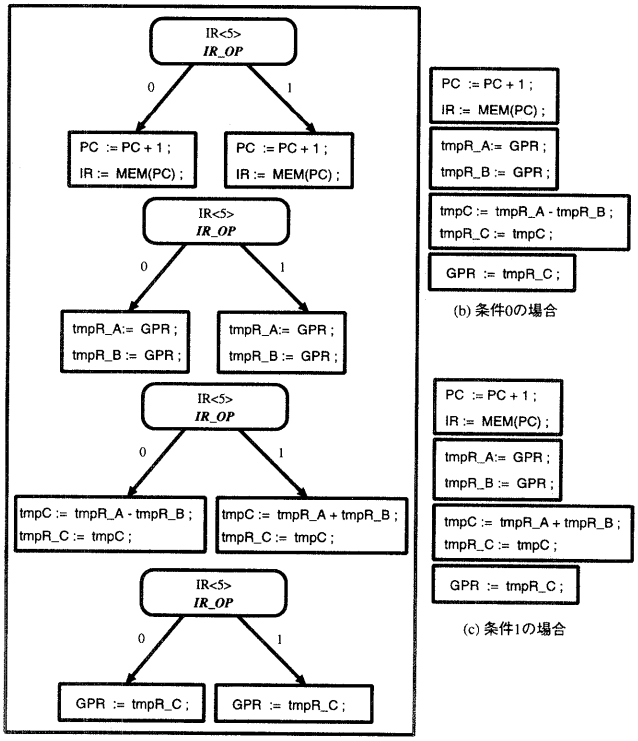

図5 ステートメントの取り出し
Fig. 5 Statements extraction.

抽出できないと判定された場合には，その場で情報抽出は中止される．

3.4.4 ステートメントの取り出し

グループ化処理が終った段階のツリーは，IR_OP のノードの下に，終端ノードが直接エッジによって結びつけられている．

ステートごとにツリーが作成されている．各ツリーの IR_OP のノードから出ているエッジの条件をすべて調べる．条件一つに注目し，ツリーすべてから，IR_OP のノードの下の注目している条件の終端ノードを取り出す．終端ノードには，ステートメントが含まれている．

これを**図5**の例で示す．この場合には，二つの条件，0，1がある．条件0に注目した場合は，図5(b)のステートメントが取り出され，条件1に注目した場合に，図5(c)のステートメントが取り出される．これ以降は，このステートメントに対する処理を行う．

3.4.5 依存関係のチェック

ステートメント中の変数（定数の場合もある）が，定義されたレジスタ（IR，PC，GPR）や定数や外部ピンでない場合には，中間変数を使っていると考えられる．依存関係をチェックし，可能ならば展開を行うことによって，新しいステートメントを作成する．

変数は次の四つメモリ変数，レジスタ変数，組合せ変数，定数，外部ピン変数に分類できる．レジスタ変数は，定義されたレジスタ変数であるか否かで，2種類に分類できる．

各ステートメントの右辺値である変数に対して，次の処理を行う．

```
foreach i （‘右辺値の集合‘）  {
 switch ($i) {
  case 定義されたレジスタ変数：
  case メモリ変数：
  case 定数：
  case 外部ピン変数：
    /* なにもしない */
    break;
  case 定義されていないレジスタ変数：
    if （一つ前のステートで $i に対する書込みがある）
      一つ前の動作を代入する
    break;
  case 組合せ変数：
    if （同一ステートで，$i に対する書込みがある）
      その代入を行う
    break;
  }
}
```

これを**図6**の例で示す．State 0 のステートメント

(1) の，右辺はPCと定数1であるので，何もしない．ステートメント *(2)* の，MEM（PC）のPCは定義されたレジスタであるので（MEMは，PCが指すアドレスの内容の値を返す関数とする），何もしない．

State 1 のステートメント *(3)(4)* の右辺の変数はGPRで，定義されたレジスタであるために何もしない．State 2 のステートメント *(5)* は，tmpR_A，tmpR_Bは，定義されていないレジスタであるために，一つ前のステートの動作を代入する．この代入により，*(A1)* のステートメントが生成される．State 2 のステートメント *(6)* は，tmpCは組合せ変数であるため，同一ステートメント内の代入を調べる．tmpCは，同一ステートで代入が *(A1)* のステートメントで行われているために，*(A2)* のステートメントが生成される．

State 3 のステートメント *(7)* の右辺値tmpR_Cは，定義されていないレジスタ変数であるために，*(A2)* で生成されたステートメントを使って *(A3)* のステートメントを生成する．

3.4.6 不要ステートメントの削除

不要のステートメントとは2種類に分けることができる．一つは，それまでの処理により冗長なステートメントとなったもの．もう一つは，すべての命令で実行されるために，コンパイラを生成するためには必要ないステートメントである．

不要ステートメントは，次の条件を満たすものである．

1. ステートメントの中の変数（定数）で，1）定義されていないレジスタ，2）組合せ変数が存在するもの
2. すべての命令で実行されるもの（プログラムカウンタのインクリメント，インストラクションのフェッチなど）

図6では，依存関係チェックが済んだ時に，*(1)* から *(7)* と *(A1)* から *(A3)* の10のステートメントがある．1.の条件を満たすステートメントは，*(3), (4), (5), (6), (7), (A1), (A2)* である．2.の条件を満たすステートメントは，*(1), (2)* である（*(1)* がプログラムカウンタのインクリメント，*(2)* がインストラクション・フェッチ）．残ったステートメントは，*(A3)* になる．

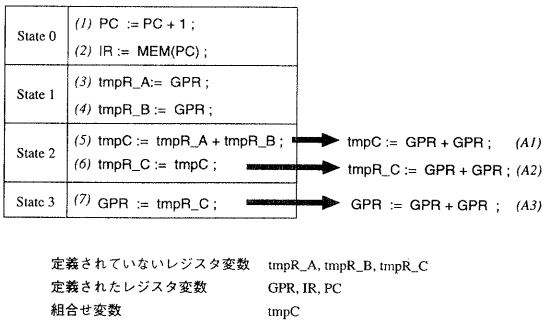

図6 依存関係のチェック
Fig. 6 Dependency check.

3.4.7 結果の出力

最終的に，残ったステートメントを出力する．図6では，*(A3)* が出力される．この3.4.4から3.4.6項の処理を繰り返すことで，すべての命令の抽出を行う．

4. コンパイラの自動生成への適用

3章で，ハードウェア記述言語によるプロセッサの設計記述から命令の動作の抽出について述べた．本章では，抽出した動作をコンパイラの生成に利用する．

4.1 プロトタイプシステム

命令の動作を抽出するプロトタイプを作成した．入力するハードウェア記述言語としては，UDL/I[12] のサブセットを利用する．入力として与える演算器の情報は，UDL/Iのシステム関数を演算器として用いた．UDL/Iのシステム関数は，基本的な加算，減算，乗算，除算や比較関数などが用意されている．それに加えて，STORE，LOAD，BEQZ，BNEZというシステム関数を追加した[13]．これらは，UDL/Iに容易に変換することが可能である．

設計者が与える情報は，次に示す通りである．

- HDLによるプロセッサ記述
- プロセッサの基本的なレジスタの指定．
- ARETを構成する時の情報

プロトタイプは，C++およびyacc, lexを用いて作成し，約10,000行でインプリメントした．今回の評価は，すべてSparc Station 10上で行った．

4.2 設計したプロセッサ

3種類のプロセッサCPU1，CPU2，CPU3をUDL/Iを用いて設計を行った．基本的な仕様は，**図7**に示す．演算は整数型のみで，汎用レジスタの数は

	CPU1	CPU2	CPU3
# of GPR	4	8	8
Instruction Set	*NOP* *ADD* *SUB* *LOAD* *LOADI* *STORE* *JUMP* *BEQZ* *BNEZ*	*NOP* *ADD* *SUB* *LOAD* *LOADI* *STORE* *JUMP* *BEQZ* *BNEZ*	*NOP* *ADD* *SUB* *LOAD* *LOADI* *STORE* *JUMP* *BEQZ* *BNEZ* *ADD3*

図7 設計したプロセッサ
Fig. 7 Designed processors.

OP-Code	Extracted Behavior
0000	----
0001	reg := ADD(reg, reg, 1B0)
0010	reg := SUB(reg, reg, 1B0)
0011 *	reg := ADD(reg, ADD(reg, reg, 1B0), 1B0)
0100	STORE(.d_mem, reg, reg, 1B0)
0110	reg := LOAD(.d_mem, reg, 1B0)
0111	reg := 10B0 !! ir<5:0>
1000	pc := reg
1000	pc := BEQZ(reg, 16B0, reg)
1010	pc := BNEZ(reg, 16B0, reg)

* CPU3のみ抽出

図8 抽出した動作
Fig. 8 Extracted instruction sets.

表1 記述量と情報抽出時間
Table 1 Behavior extraction time.

	CPU1	CPU2	CPU3
HDLの記述量	147行	163行	180行
基本レジスタ，メモリの情報	14行	14行	15行
ARETに関する情報	5行	5行	6行
抽出時間	2.6秒	2.8秒	2.8秒

CPU1は4本，CPU2とCPU3は8本とした．CPU1とCPU2の命令セットは同じで，CPU3のみ*ADD3*という三つのレジスタ和 ($D = S1 + S2 + S3$) を実行できる命令を加えた．各命令は，4クロックで実行を終了する．

4.3 情報抽出の結果

図8に抽出した結果を示す．抽出した結果は，UDL/Iのシンタックスで示されている．OP_Codeが，2進数で0000の時には，NOP（No OPeration）で何も抽出されない．OP_Codeが0011の場合は，CPU3の場合のみ抽出された．現段階では，簡単なプロセッサに対するアルゴリズムであるが，実際に情報抽出を行うことができた．また，抽出するに要した時間を**表1**に示す．

4.4 情報抽出に必要な情報に関する考察

本論文で提案したアルゴリズムは，HDLの入力だけではなく，演算器，基本レジスタとメモリ，ARETを作成するための情報を与えている．演算器に関する情報に関しては，大村らの手法[11]を用いることで，自動化することが可能であると考えられる．

基本レジスタとメモリに関する情報に関しては，現段階では，任意のHDLの入力からプログラム・カウンタや命令レジスタなどの自動識別ができないために，入力として与えている．プロセッサの設計を行う場合，設計者は，プログラム・カウンタや命令レジスタといった基本レジスタを中心に設計を行うために，これらの情報を入力として与えることは容易である．

表2 コンパイラの生成時間
Table 2 Compiler generation time.

	CPU1	CPU2	CPU3
生成時間	14.4秒	15.0秒	15.0秒

ARETに関する情報を自動抽出するためには，基本レジスタの情報を自動的に抽出可能となり，さらに，命令レジスタのビット割り付けが自動判別する必要がある．これらもアーキテクチャの重要な情報であるために，設計者が入力として与えることは容易である．

HDLに記述量とその他の付加情報の記述量を比較すると，付加情報はHDLの記述量に比べて約12%と少ない（表1）．

4.5 コンパイラの生成

抽出した情報を用いて，コンパイラの生成を行った．冨山らが作成しているコンパイラ・ジェネレータ[14]を使用して，コンパイラを生成した．このコンパイラ・ジェネレータは入力として，以下のものを必要とする．

- 命令の動作
- 汎用レジスタ数
- 命令の実行時間（コスト）
- 出力するニモニック

今回は，命令の動作以外は人手で与えた．

コンパイラを生成するためにかかった時間を**表2**に示す．HDLによるプロセッサの記述から，情報抽出を行いコンパイラの生成が短時間で可能であった．この短時間でコンパイラが生成可能となることで，ハードウェアとソフトウェアの両面から性能評価を行うことが容易となる．

4.6 ハードウェア/ソフトウェア協調評価

本論文で提案をしている情報抽出の長所は，ハードウェア設計をハードウェア記述言語で行うことで，今までの論理合成などの設計支援の環境をそのまま利用できることである．これにより，ハードウェアの評価を行うと同時に，ソフトウェアの評価を行うことができる．ハードウェアの評価項目としては，ゲート数，レイアウト面積，動作周波数などがある．ソフトウェアの評価項目としては，実行ステップ数，命令の実行頻度，ハードウェア資源の利用効率などがある．

ここでは例として，UDL/Iの合成系を用いて，ハードウェアの評価としてゲート面積を求め，ソフトウェアの評価として実行ステップ数を求めた．ソフトウェアの評価には，**図9**の2種類のループを用いた．Loop1とLoop2では，ループが2重になっている以外に，3変数の加算を行っており，CPU3のADD3が入ることでハードウェア/ソフトウェアの性能にどのような

```
/* Loop 1 */
main ( ) {
    int sum=0, n = 5,i;
    for ( i = 0; i < n, i++ ) {
        sum = sum + i ;
    }
}
```

```
/* Loop 2 */
main ( ) {
    int sum=0, n = 5, i, j;
    for ( i = 0; i < n ; i++ ) {
        for ( j = 0; j < n ; j++ ) {
            sum = sum + i + j ;
        }
    }
}
```

図9 使用したプログラム
Fig. 9 Benchmark programs.

表3 ハードウェア/ソフトウェア協調評価
Table 3 Hardware and software evaluation.

		CPU1	CPU2	CPU3
HW	ゲート面積（基本ゲート換算）	2,755	3,607	6,647
SW	Loop1（実行命令数）	128	41	41
	Loop2（実行命令数）	858	256	231

影響がでるかの評価を行う必要がある．また，最適なレジスタ数などの検討を行う必要がある．

ゲート面積の見積りと実行命令数を，**表3** に示す．Loop1 では，ADD3 を使用できないために，実行した命令数は CPU2 と CPU3 は同じになるが，加算器を余分に持つためにゲート面積では，CPU2 に比べて 84％増加している．CPU1 は，CPU2 や CPU3 に比べてレジスタの数が半分の 4 本と少ないために Load/Store の回数が増加するために，実行命令数は約 3 倍に増加しているが，CPU2 に比べて面積は 24％減少している．

Loop2 では，3 変数の加算があり CPU3 の ADD3 を使用できるために，CPU2 に比べ CPU3 は実行命令数が 10％減少している．

実際には，システムに対する要求などから，アーキテクチャを決定したり，アーキテクチャの変更を行う．

このように，ハードウェア記述言語による設計記述から情報を抽出して，コンパイラの生成を行うことで，今までのハードウェア記述言語による設計記述で利用できた支援はそのまま利用でき，さらに，ソフトウェアの支援を行うことができた．情報抽出とコンパイラの自動生成を組み合わせることにより，ハードウェア/ソフトウェア両面から性能評価が可能となった．今回は，性能評価という形でコンパイラを利用したが，用途によっては，このコンパイラによって最終的なプログラムをコンパイルすることなどが考えられる．

5. まとめ

ハードウェア記述言語による設計記述を中心に置くことで，今までの論理合成や機能/ゲートレベル・シミュレーションといった支援はそのまま利用可能となる．さらに情報抽出という技術を用いることで，さらに支援範囲を広げることが可能となる．本論文では，情報抽出のアルゴリズムについて述べ，プロセッサ記述から情報抽出を行った例を示した．さらに，抽出した動作から，コンパイラ・ジェネレータを用いてコンパイラを生成することができた．そのコンパイラを用いることで性能評価をハードウェアとソフトウェアの両面から行うことが可能となった．

提案したアルゴリズムでは，記述によっては抽出できない可能性もある．ハードウェア記述言語では，設計記述の自由度が高く，現状ではさまざまな記述から命令セットの動作を取り出すことは困難であるためである．今後アルゴリズムの改善を行うことで，アーキテクチャに対する制約を含めて制約を緩和することができると考えている．

本手法は，HDL で記述されたコアとなるプロセッサを用途に応じてカスタマイズする際に，命令セットの追加や削除，汎用レジスタの変化に対応したコンパイラを生成する技術として利用可能であると考えている．

今回は，コンパイラを作成するために必要なすべての情報を抽出していない．その他の情報に対する自動抽出についても考えていきたい．今後は，コンパイラの生成以外の支援，例えば，抽出した命令の動作を用いて，高速シミュレーションモデルの生成などについての研究を行っていく．

謝辞 本研究の一部は，情報処理振興事業協会「独創的情報技術育成事業」の一環として行われたものである．現在，筆者らとハードウェア/ソフトウェア協調設計支援システムの作成を行っている冨山宏之氏，橋本孝幸氏，井上昭彦氏，および，日頃から討論いただく安浦研究室の諸氏に感謝します．また，UDL/I 処理系に対する助言をいただきました京都高度技術研究所の神原弘之氏，横田吏司氏に感謝します．

参考文献

1) Gupta, R.K. and De Micheli, G.: Hardware-Software Cosynthesis for Digital Systems, *IEEE Design & Test of Computer*, pp.29–41

(Sep. 1993).
2) Thomas, D.E., Adams, J.K. and Schmit, H.: A Model and Methodology for Hardware-Software Codesign, *IEEE Design & Test of Computer*, pp.6–15 (Sep. 1993).
3) Kalavade, A. and Lee, E.A.: A Hardware-Software Codesign Methodology for DSP Applications, *IEEE Design & Test of Computer*, pp.16–28 (Sep. 1993).
4) Sato, J., Alomary, A., Honma, Y., Nakata, T., Shiomi, A., Hikichi, N. and Imai, M.: PEAS-I: A Hardware/Software Codesign System for ASIP Developement, *IEICE Trans.*, Vol.E77-A, No.3, pp.483–491 (1994).
5) Akaboshi, H. and Yasuura, H.: COACH: A Compute Aided Design Tool for Computer Architects, *Trans. IEICE*, Vol.E76-A, No.10, pp.1760–1769 (1993).
6) Pyo, I., Su, C., Huang, I., Pan, K., Koh, Y., Tsui, C., Chen, H., Cheng, G., Liu, S., Wu, S. and Despain, A.M.: Application-driven Design Automation for Microprocessor Design, *Proc. of the 29th ACM/IEEE DAC*, pp.512–517 (Jun. 1992).
7) Jiang, Y.-M., Lee, T.-F., Hwang and Lin, Y.-L.: Performance-Driven Interconnection Optimization for Microarchitecture Synthesis, *Trans. on CAD*, Vol.13, No.2, pp.137–149 (1994).
8) Menez, G., Auguin, M., Boéri, F. and Carrière, C.: A Partitioning Algorithm for System-Level Synthesis, *IEEE ICCAD-92* (Nov. 1992).
9) Marwedel, P.: Tree-Based Mapping of Algorithms to Predefined Structures, *Proc. ICCAD-93*, pp.586–593 (Nov. 1993).
10) Huang, I. and Despain, A.M.: High Level Synthesis of Pipelined Instruction Set Processors and Back-End Compilers, *Proc. of the 29th ACM/IEEE DAC*, pp.135–140 (Jun. 1992).
11) 大村昌彦, 安浦寛人, 田丸啓吉：組合せ回路の機能情報抽出, 電子情報通信学会論文誌, Vol.J74-A, No.2, pp.247–255 (1991).
12) UDL/I Committee: *UDL/I Language Reference Manual Version 2.0.2*: Japan Electronic Industry Development Association (Jun. 1993).
13) Akaboshi, H. and Yasuura, H.: Behavior Extraction of MPU from HDL Description, *Second Asian Pacific Conference on Hardware Description Languages*, pp.67–74 (Oct. 1994).
14) Tomiyama, H., Akaboshi, H. and Yasuura, H.: Compiler Generator for Hardware/Software Codesign: *Second Asian Pacific Conference on Hardware Description Languages*, pp.267–270 (Oct. 1994).

(平成6年11月10日受付)
(平成7年 9 月 6 日採録)

赤星 博輝（正会員）

平成3年九州大学工学部情報工学科卒業．平成5年同大学大学院総合理工学研究科情報システム学修士課程修了．現在，同大学大学院博士課程在学中．コンピュータ・アーキテクチャ設計用CAD，コンパイラの研究に従事．平成7年度情報処理学会山下記念研究賞を受賞．

安浦 寛人（正会員）

昭和51年京都大学工学部情報工学科卒業．昭和53年京都大学工学研究科修士課程（情報工学専攻）修了．昭和55年より同大工学部助手．同大工学部電子工学科助教授を経て，平成3年より九州大学大学院総合理工学研究科情報システム学専攻教授．VLSIシステムの設計手法とCADの研究およびハードウェアアルゴリズムの研究に従事．昭和57年電子通信学会学術奨励賞，昭和63年および平成6年電子情報通信学会論文賞，平成4年情報処理学会論文賞，平成5年情報処理学会坂井記念特別賞およびBest Author賞をそれぞれ受賞．電子情報通信学会，IEEE，ACM，EATCSなど各会員．