設 計 自 動 化 42－1
（1988. 5. 17）

専用プロセッサ設計における制御系の検討

竹沢寿幸　池永　剛　白井克彦
早稲田大学理工学部

専用プロセッサ設計における制御系合成法について検討した。ハードワイヤドロジック制御方式およびマイクロプログラム制御方式による制御系の合成法とそれらの特徴を整理し、特にルールベースによるインストラクションセット合成について検討した。さらに、合成された回路に対するソフトウェア開発ツールとしてコード生成系を自動的に提供する枠組みを検討した。例として、ハードワイヤドロジックによる制御系を持った回路の合成結果、インストラクションセットを合成するための予備実験結果、コード生成系を提供するための予備実験結果を示す。このような環境が実現されれば、システム設計者が回路の可能性を考慮できる範囲が飛躍的に広がると期待できる。最後に、今後の課題についてふれる。

Investigation of Controller Synthesis in Designing Special Purpose Processors

Toshiyuki Takezawa　Takeshi Ikenaga　Katsuhiko Shirai

Department of Electrical Engineering, Waseda University
3-4-1 Ohkubo, Shinjuku-ku, Tokyo 160, Japan

This paper discusses the controller synthesis method in designing special purpose processors. Two types of controller which are the hard wired logic one and the micro-programming one should be considered. Especialy in them, the instruction set synthesis by the rule based approach is examined. We also propose a framework of a software development tool for synthesized circuits to prepare a code generator automatically. An example to synthesize a circuit with a hard wired logic controller is shown. Results of preliminary experiments to synthesize an insturction set and to prepare a code generator are shown. The method proposed in this paper is expected to be very useful for system designers to consider various posibilities in the design process. Future problems are also discussed.

1．はじめに

高級言語による動作や機能に関する仕様記述と様々な制約記述をもとに専用回路の設計を行う研究を進めている[1][2][3][4]。ハードウェア設計を行うとともにソフトウェアの効果的な開発を支援する環境を目指している。

図1に構築中のシステムの構成を示す。専用の仕様記述言語に従って記述した高位の動作仕様を入力する。入力された動作仕様に対して、字句解析、構文解析、意味解析を行った後、中間表現に変換する。中間表現と制約記述から入出力部とデータパスを合成してＲＴレベルのハードウェア記述を出力する。制御系としては基本的にはハードワイヤドロジック制御方式を仮定したものが出力される。

本稿では、それをマイクロプログラム制御方式による制御系へ変換し、インストラクション・セットを合成する手法を検討する。さらに、そのようにして合成された回路に対してコード生成系を自動的に提供する枠組みを提案する。

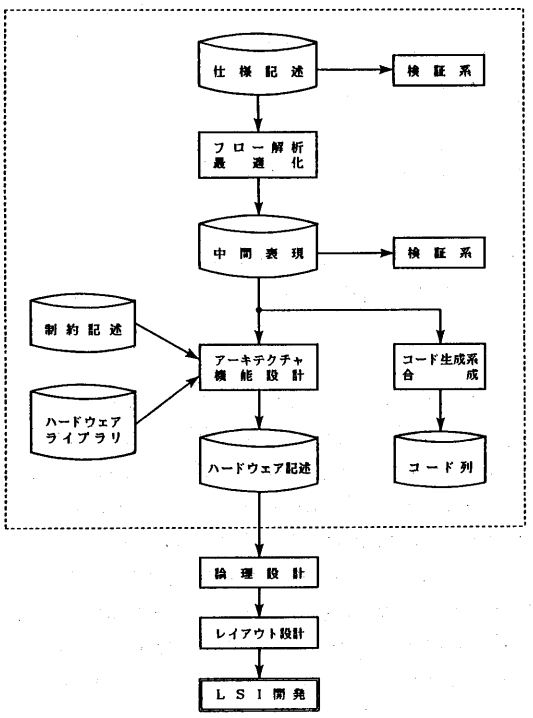

図1　システム構成

制御回路の自動合成に関する研究には、多くのものがある[5][6]。例えば、ＲＴレベルのハードウェア記述からMealy 型の制御回路をＰＬＡで実現するもの[5]や、マイクロプログラムのアドレス順序情報と分岐機能を定義する情報からマイクロプログラムのモデルを詳細化するもの[6]がある。本稿で論ずるものは、制御回路を実現するための情報を高位の動作記述からいかに詳細化するかということである。特に、インストラクション・セット合成については、高位の動作仕様から直接ＲＴレベルに変換するのではなく、その間でＩＳＰレベルに変換することに相当すると考えられる。

2．制御系のモデルと合成法の検討

入力されたアルゴリズムと制約をもとに詳細化されるハードウェア記述は、入出力部を含んだデータパス系と、その動作順序、タイミングを表す状態遷移形式の動作フローである。データパス系は、入力されたアルゴリズムの制御フロー、データフローと制約をもとにある程度の最適化を施したものである。動作フローは、入力されたアルゴリズムを合成されたデータパス系でいかに実行するか、その順序とタイミングを状態遷移形式で表したものである。

比較的小規模でかつ専用の回路については高速化のためにも制御系をハードウェア化した方が望ましい。しかし、これは規模が大きくなるにつれて状態数が増加する上に、専用のものには適するが汎用性に欠けている。規模の増大と汎用性の向上のためには、マイクロプログラム制御方式の制御系が有効である。

アーキテクチャ設計レベルにおける制御系合成の手法について整理する。

（１）ハードワイヤドロジック制御方式による制御系

状態遷移形式の動作フロー中のステート要素から状態フリップフロップを合成し、その遷移と制御信号出力を実現する順序回路を合成することで、ランダムロジックやＰＬＡに

よるハードワイヤドロジック制御方式の制御系を合成することができる。

この方式は高速である利点はあるが、汎用性に乏しい。また、入力されたアルゴリズムが複雑あるいは大規模になるにつれて状態数が増加し制御系も複雑あるいは大規模になる欠点がある。したがって、比較的小規模なアルゴリズムでかつ高速処理の要求される専用回路に向く。

（２）ダイレクト・コントロール方式マイクロプログラム制御による制御系

各制御点の制御信号をダイレクトにマイクロ命令のビットに対応させてマイクロ命令を構成することができる。マイクロ命令のデコーダは簡単となる上に、合成されたマイクロ命令の柔軟性が高い。しかし、マイクロ命令語長はマイクロ操作の数だけ必要であるため長くなる。

このような制御回路を実現するには、例えば、図２のような制御回路モデルを仮定し、状態遷移形式の動作フロー中のステート要素毎にすべての制御点に対して必要な制御信号を合成すればよい。ただし、分岐命令に対しては、条件と次のアドレスを指定する必要がある。さらに、制御系のモデルに従い、マイクロ・インストラクション・レジスタ（ＭＩＲ）のフェッチ、デコード、実行といった状態遷移記述に変換する。

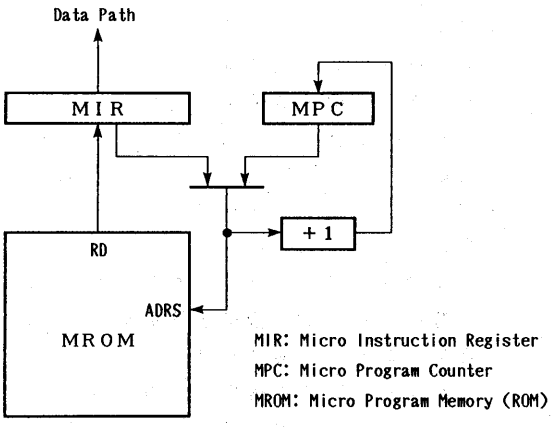

図２　マイクロプログラム制御による制御系のモデル

（３）インストラクション・セット

マイクロ命令語長の短縮化について検討する。

異なるマイクロ操作がＮ個ある時に最小のビット構成で符号化すると$\log_2 N$ ビットとなる。しかし、語長は最小となるが、柔軟性に欠ける上に、デコーダが複雑になり、実行速度も遅くなる。

状態遷移形式の動作フロー中のステート数をＭ個とすると、$\log_2 M$ ビットの状態フリップ・フロップが合成されるため、それをＭＩＲとみなして制御系を構成することも考えられる。ステート数Ｍが大きい場合は、状態数を見かけ上大幅に減らすことが可能である。しかし、ＭＩＲのどのビットが何を表現しているかが分かりにくい上、必要なデコーダは、直接制御方式の順序回路とほぼ等価であると考えられる。

したがって、分かりやすさと汎用性を考慮して、マイクロ命令セットとそのフォーマットを決定することが望まれる。マイクロ命令語長を短くするには、同時に実行されないマイクロ操作をまとめればよい。しかし、やみくもにまとめるのではなく資源との対応を考えて分類すべきである。必要により、データパス系を修正することもある。

人間がマイクロ命令セットを決定する場合も、様々な要因を考慮して決定していると考えられる。そこで、ルールベースの手法によりマイクロ命令セットを決定することを検討する。制御系のモデルは、パイプライン制御などは考慮せず図２のような比較的単純なものとする。また、マイクロ命令は、分岐命令と演算命令に分離し、かつ、分岐先数を二つに限定した水平型のマイクロ命令とする。ここで、利用可能なインストラクションのパタンを分離、整理し、それから選択するものとする。さらに、この場合、データパス系も制御系に合わせて変更する必要がある。例えば、そのルールとして次のようなものが考えられる。

○同時にアクセスされないレジスタは、できるだけ一つのレジスタファイルにまと

める。

○同時に実行されない演算や比較などの機能は、できるだけ１個のＡＬＵで実行させる。

ここで検討した手法と通常のマイクロプログラム開発との違いは、マイクロプログラムの最適化の場合、ハードウェアが固定であるのに対し、自動設計の場合、ハードウェアが可変である点があげられる。

３．ソフトウェア開発ツールの検討

インストラクションを持つハードウェアが合成されるとすれば、高位の仕様記述から合成するような環境においては、ソフトウェア開発ツールとして自動的にコンパイラが提供できれば非常に有効である。この場合、プロセッサのインストラクションなどアーキテクチャの記述を与えることによってコンパイラを提供する技術が必要不可欠である。

なるべく容易にコンパイラを提供するために、処理過程を目的プロセッサに独立な部分と依存した部分に分ける構成とすることにした。コード生成部では、プロセッサのアーキテクチャと命令セットに関するデータベースを書き換えることにより、それぞれのプロセッサに対応した、効率的なコード生成を行えることが必要である。そこで、ハードウェア記述をデータベースとして与え、最適化をルールベースで行うエキスパートシステムの構成とすることとした（図３）。

プロセッサ独立部には、本設計支援システムにおいて、字句解析、構文解析、意味解析を行った後、中間言語に変換し、最適化を行う部分を流用して利用する。

プロセッサ依存部では目的プロセッサに関する情報を参照し、ストレージ割付を行った後、ルールベースの手法を用いてコード生成および最適化を行う。

目的プロセッサに固有な情報について、次の２種類のデータベースを参照する。アーキテクチャとしては、レジスタの種類や個数、ＲＡＭの構成などのハードウェア・アーキテクチャに関する情報を記述する。命令セットとしては、演算命令、転送命令、制御命令など、命令の種類毎にその動作を記述する。

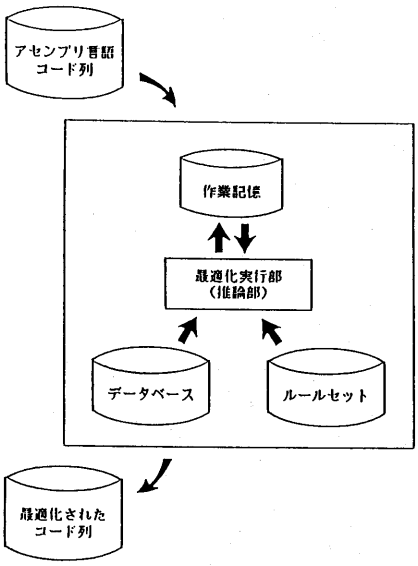

図３　コード生成系の構成

ただし、扱うことのできるプロセッサのアーキテクチャ、および命令セットは次に示す範囲内のものとする。

○アーキテクチャ
- 加減演算を実行可能なＡＬＵを持つ。
- 汎用レジスタを１個以上持つ。
- 変数領域として使用できるＲＡＭを持つ。

○命令セット
- レジスタ・レジスタ間、およびレジスタ・メモリ間の転送命令を持つ。
- レジスタ間接モードのアドレッシングが可能。
- イミディエット・データのロード命令を持つ。
- ３番地文で表現された式を実行するための算術演算および論理演算命令を持つ。
- 無条件ジャンプ命令を持つ。
- フラグレジスタを持ち、ゼロフラグ、サインフラグによる条件ジャンプ命令を持つ。

処理はストレージ割付け、コード生成、コード最適化の３つからなる。

（１）ストレージ割付け

３番地文、宣言、ライフタイム解析情報、および目的プロセッサのデータベースを参照して、変数をレジスタおよびメモリに割り付ける。この結果、変数名とその変数が割り付けられたストレージを示すシンボルテーブルが生成される。

（２）コード生成

表駆動（Table Driven）方式を基本としている。具体的には３番地文で表現されたそれぞれの文について、３番地文の型、演算子の種類、およびアドレッシングモードにより分類した命令あるいは命令列をデータベースに蓄えておき、コード生成時にこのデータベースから適切な命令を検索する。プロセッサ独立な最適化はすでに行われているため、この段階で３番地文、あるいは式の木構造を組み変えるような最適化は行わない。３番地文を変えない範囲での最適化、例えば“ADD 1”を“INC”で置き換えるなどの操作はデータベース中に記述しておく必要がある。

（３）ルールベースによる最適化

さらに、局所的最適化を施す。これは、一般に覗き穴的最適化－peephole optimization－と呼ばれている手法である。ここで、プロセッサのアーキテクチャの特徴を利用した最適化を記述しやすくするために、最適化をそれぞれ独立したif-thenルールとして記述することにした。

４．予備実験

（１）ハードワイヤドロジック制御系を持つ回路の合成例

例題として、クィック・ソートを取り上げる。クィック・ソートのアルゴリズムは、ソートされるべき要素列をE(i)（i=1..N)とすると、そのうちの一つの要素をXとし、E(i)≦XなるE(i)をXより前に、E(i)≧XなるE(i)をXより後に来るように並べ換える操作を基本としている。次に、この分割操作をXの前の部分列とXの後の部分列に対して、未処理の部分がなくなるまで繰り返す。

図４に仕様記述例を示す。各要素は、32ビットの整数であり、昇順にソートを行うものとする。現在、仕様記述として、関数や手続きを記述することは許しているが、それらの

```
program Quick_Sort_Soft (eport);
const N = 65536;
      M = 2048;
type integer = unsigned bit 16;
     real = unsigned bit 32;
input eport : real;
output eport : real;
var e : array [N] of real; { 64Kw * 32bit }
    stack_h, stack_l : array [M] of integer; { 4Kw * 16bit }
    i, j, k, l : integer;
    x, w : real;
    p : unsigned bit 12;
begin
{ input data from port(eport) to RAM(e) }
  for i := 0 to N-1 do
    e[i] := eport;
{ body of quick sort }
  p := 0; stack_h[0] := 0; stack_l[0] := N-1;
  repeat
    k := stack_h[p]; l := stack_l[p]; p := p-1;
    repeat
      i := k; j := l; x := e[i];
      repeat
        while e[i] < x do i := i+1;
        while x < e[j] do j := j-1;
        if i <= j then begin
          w := e[i]; e[i] := e[j]; e[j] := w;
          i := i+1; j := j-1;
        end
      until i > j;
      if i < l then begin
        p := p+1; stack_h[p] := i; stack_l[p] := l
      end;
      l := j
    until k >= l
  until p = 0;
{ output data from RAM(e) to port(eport) }
  for i := 0 to N-1 do
    eport := e[i]
end.
```

図４　クイック・ソートの仕様記述例

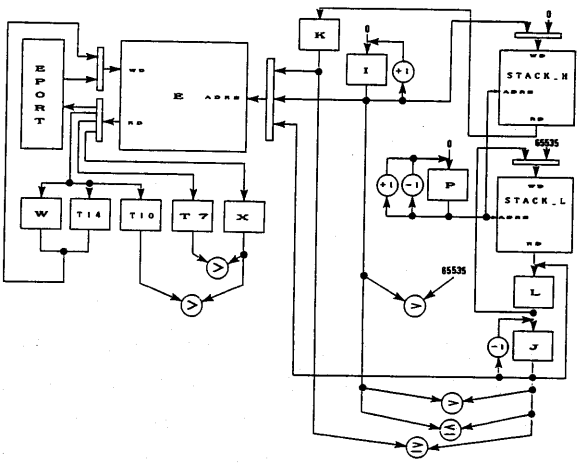

図５　データパス系自動合成例

```
(PROGRAM QUICK_SORT_SOFT
  (INPUT
    (CLOCK (UNSIGNED BIT 1))
    (EXEC (UNSIGNED BIT 1))
    (RESET_INCTL (UNSIGNED BIT 1))
    (EPORT_OUTEN (UNSIGNED BIT 1))
    (EPORT_INEN (UNSIGNED BIT 1)))
  (OUTPUT
    (EPORT_OUTREQ (UNSIGNED BIT 1))
    (EPORT_INREQ (UNSIGNED BIT 1)))
  (BIDIRECT
    (EPORT (UNSIGNED BIT 32)))
  (REGISTER
    (T0014 (UNSIGNED BIT 32))
    (P (UNSIGNED BIT 12))
    (W (UNSIGNED BIT 32))
    (X (UNSIGNED BIT 32))
    (L (UNSIGNED BIT 16))
    (K (UNSIGNED BIT 16))
    (J (UNSIGNED BIT 16))
    (I (UNSIGNED BIT 16)))
  (TERMINAL
    (T0010 (UNSIGNED BIT 32))
    (T0007 (UNSIGNED BIT 32)))
  (RAM
    (STACK_L (2048) (UNSIGNED BIT 16))
    (STACK_H (2048) (UNSIGNED BIT 16))
    (E (65536) (UNSIGNED BIT 32)))
  (EXTERNAL-RAM)
  (ROM)
  (STATIC)
  ;;
  (STATE QUICK_SORT_SOFT_BLOCK (CLOCK)
    (B0002
      (IF (I > 65535)
          (THEN
            (P := 0)
            (STACK_H (0) := 0)
            (STACK_L (0) := 65535)
            (GOTO B0005))
          (ELSE
            (EPORT_INREQ := 1)
            (GOTO B0003-1))))
    (B0003-1
      (IF (EPORT_INEN = 0)
          (THEN
            (GOTO B0003-1))
          (ELSE
            (E (I) := EPORT)
            (EPORT_INREQ := 0)
            (GOTO B0003-4))))
    (B0003-4
      (I := INC I)
      (GOTO B0002))
    (B0005
      (K := STACK_H (P))
      (L := STACK_L (P))
      (P := DEC P)
      (GOTO B0006))
    (B0006
      (I := K)
      (J := L)
      (X := E (K))
      (GOTO B0007))
    (B0007
      (T0007 := E (I))
      (IF (T0007 < X)
          (THEN
            (I := INC I)
            (GOTO B0007))
          (ELSE
            (GOTO B0010))))
    (B0030
      (GOTO B0009))
    (B0031
      (GOTO B0010))
    (B0010
      (T0010 := E (J))
      (IF (X < T0010)
          (THEN
            (J := DEC J)
            (GOTO B0010))
          (ELSE
            (GOTO B0013))))
    (B0013
      (IF (I <= J)
          (THEN
            (W := E (I))
            (GOTO B0015-1))
          (ELSE
            (GOTO B0016))))
    (B0015-1
      (T0014 := E (J))
      (GOTO B0015-2))
    (B0015-2
      (E (I) := T0014)
      (E (J) := W)
      (I := INC I)
      (J := DEC J)
      (GOTO B0016))
    (B0016
      (IF (I > J)
          (THEN
            (GOTO B0018))
          (ELSE
            (GOTO B0007))))
    (B0029
      (GOTO B0018))
    (B0018
      (IF (I < L)
          (THEN
            (P := INC P)
            (STACK_H (P) := I)
            (STACK_L (P) := L)
            (GOTO B0021))
          (ELSE
            (GOTO B0021))))
    (B0021
      (L := J)
      (IF (K >= J)
          (THEN
            (GOTO B0023))
          (ELSE
            (GOTO B0006))))
    (B0023
      (IF (P = 0)
          (THEN
            (I := 0)
            (GOTO B0026))
          (ELSE
            (GOTO B0005))))
    (B0026
      (IF (I > 65535)
          (THEN
            (GOTO INIT))
          (ELSE
            (EPORT_OUTREQ := 1)
            (GOTO B0027-1))))
    (B0027-1
      (IF (EPORT_OUTEN = 0)
          (THEN
            (GOTO B0027-1))
          (ELSE
            (EPORT := E (I))
            (EPORT_OUTREQ := 0)
            (GOTO B0027-4))))
    (B0027-4
      (I := INC I)
      (GOTO B0026))
    (INIT
      (IF (EXEC = 1)
          (THEN
            (I := 0)
            (GOTO B0002))
          (ELSE
            (GOTO INIT)))))
  ;;
  (EXCEPTION
    (IF RESET_INCTL
        (THEN
          (GOTO INIT)))))
```

図6　出力ハードウェア記述例

再帰的な呼び出しは許していないため、stack を宣言して、非再帰的に仕様を記述している。実験システムにより自動合成されたデータパス系の例を図5に示す。現在のシステムからは、制御系としてはハードワイヤドロジック制御方式のみ合成可能である。出力されたＲＴレベルのハードウェア記述の一部を図6に示す。

（２）マイクロプログラム制御系とインストラクションセットの合成例

予備的に入手により、図５のデータパス系をマイクロプログラム制御方式の制御系を持つように変更したものを図７に示す。この時の命令のフォーマットを図８に示す。

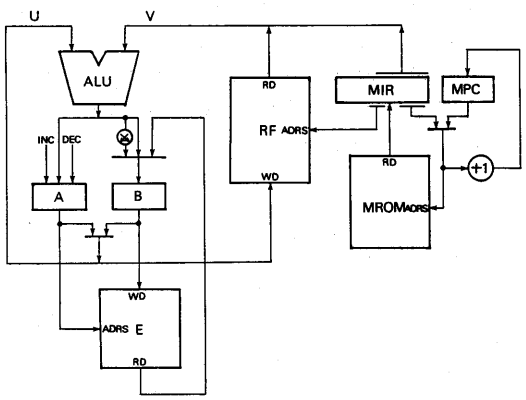

図７　マイクロプログラム制御向けデータパス系合成例（主要部）

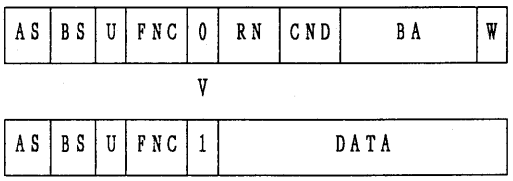

図８　マイクロ命令フォーマット例

（３）コード生成系の試作

予備実験として、既存の信号処理用プロセッサを対象に、アーキテクチャに適応可能なコンパイラを試作した。実際には、フロー解析された中間表現から、コード生成とルールベースによる最適化を行うものを試作した。例として取り上げたのは日本電気の信号処理用プロセッサμPD77230である[2]。

コード生成は、図９に示すようなコード生成テーブルを参照して、３番地文の型、演算子の種類、ストレージ割付の結果などにより、マッチングをとることで行う。

```
Pattern : (A:memory = B:register(dead) op C:memory)
Code    : Calc Ix-pointer (Address C);
          OP   B, RAM1;
          Calc Ix-pointer (Address A);
          MOV  RAM1, B;

Pattern : (A:register = B:register op C:register)
Code    : MOV  A, B;
          MOV  NON, C;
          OP   A, IB;

Pattern : (A:register = B:memory * C:register)
Code    : Calc Ix-pointer (Address B);
          MOV  KLR1, C;
          NOP;
          MOV  A, ML;
```

図９　コード生成テーブル

ルール：命令の入れ替え

if　３命令が次の順で並んでいる．
①ブランチ以外の命令
②ブランチ命令
③ｎｏｐ命令

then ③のｎｏｐ命令を削除する．
①と②の命令を入れ換える．

```
MOV  RAM1, WR0;      3ステップ
JMP  LABEL;
NOP;
      ↓
JMP  LABEL;          2ステップ
MOV  RAM1, WR0;
```

図１０　最適化ルール例

図１０は局所的最適化のためのルールの一例を示したものである。このルールでは３ステップを要している転送、および分岐命令を２ステップに短縮し、効率化をはかっている。

要素技術として、このようなコード生成系を提供することが可能であることは確認できた。これは、コンパイラの自動合成[7]と関連がある。

これを設計支援システムと接続するためには、ＲＴレベルのハードウェア記述をもとにコード生成用のテーブルを合成することと、局所的最適化をよりマクロなレベルで記述したルールが必要である。マクロなルールを用意することは、知識処理技術の面からも難しい問題を含んでいる。

5．むすび

専用プロセッサ設計における制御系合成法について報告した。直接制御方式およびマイクロプログラム制御による制御系の合成法を検討し、それらの特徴を整理した。また、コード生成系を自動的に提供する枠組みについても述べた。専用回路設計においてハードウェアのみならずソフトウェアも含めて、その開発支援環境を構築することを目的としている。

今後は、この方針に基づき制御系合成部を構築し、現在整備中の設計支援システムに接続する予定である。ただし、細かい点については、さらに調整する必要がある。汎用性のあるマイクロ命令セットの合成として複数のアルゴリズムへの適応も検討したい。

現在、整備の進んでいる部分については、既存の論理合成システムと接続する実験を行い、実用性に関する評価を行う予定である。また、本システムはハードウェア、ソフトウェアの開発支援環境を構築することを目的としているため、そのためのヒューマン・インタフェースについても検討する必要がある。

【参考文献】

[1] 白井，竹沢，永井：“高級言語による仕様記述に基づく回路のデータパス系自動設計法”，情報処理学会論文誌，Vol.28，No.1，pp.99-108，(1987-1).

[2] 竹沢，上野，池永，白井：“信号処理プロセッサのための知識処理手法を用いたコンパイラ”，電子情報通信学会第２回ディジタル信号処理シンポジウム，A-6-5，(1987-12).

[3] 竹沢，上野，桑原，白井：“ＶＬＳＩアーキテクチャ設計における制御系およびデータパス系合成”，電子情報通信学会技術研究報告，CPSY87-44，(1988-3).

[4] 竹沢，上野，白井：“専用プロセッサ設計における制御回路合成”，情報処理学会第３６回全国大会，5X-6，(1988-3).

[5] 高木茂：“制御回路自動合成の一手法”，情報処理学会設計自動化研究会，27-1，(1985-7).

[6] 戸次，横田，浜田：“制御論理回路自動合成方式の検討”，情報処理学会設計自動化研究会，35-1，(1986-12).

[7] 佐々政孝：“コンパイラ生成系”，情報処理，Vol.28，No.10，pp.1359-1367，(1987-10).

[8] 相磯，飯塚，坂村：“ダイナミック・アーキテクチャ”，共立出版，(1980).