設 計 自 動 化 46－10
（1989. 2. 21）

# アーキテクチャ設計支援システムにおける知識の利用と評価

竹沢寿幸　　白井克彦
早稲田大学 理工学部

ユーザの要求にあった回路を短期間で正確に効率よく作るためには、より高位レベルからの自動合成ないしは設計支援が望まれる。そのためには、知識処理技術を利用することと、対象を限定して設計システムを構築することが有効と考えられる。このような観点から、これまでに高位レベルからの電子回路設計支援への知識処理技術の適用法の検討を行い、信号処理を中心とした専用回路を対象として扱える実験システムの構築を行ってきた。これらの準備のもとに、自動論理合成システムと接続した環境を構築し、取り扱える回路の範囲を拡大するためのシステムの拡張、そこでの知識の充実、システムにおけるツールとその利用法について検討した。本論文では、システム全体の概要と各種適用実験について報告する。本システムは、抽象的な動作仕様記述を入力し、初期回路合成、回路の修正を行うことにより、よい設計を探索するものである。様々なレベルで評価ツールを持つので、設計者はシステムを利用して積極的にみずからフィードバックをかけることによって広い範囲からよい設計をみつけることができる。比較的簡単なプロセッサなどの実験結果を示し、システムの評価を行なうとともに、今後の課題を論ずる。

# Evaluation and Usage of Knowledge in an Architecture Design System

Toshiyuki Takezawa and Katsuhiko Shirai

Department of Electrical Engineering, Waseda University
3-4-1 Ohkubo, Shinjuku-ku, Tokyo 169, Japan

It is expected to support the design from higher level so as to fabricate circuits that satisfies requirements of users quickly. In order to solve this problem, it is useful to use knowledge-based technique in design process and to build a DA or CAD system limiting application area. In such view point, we have examined how to apply knowledge-based technique to the high-level synthesis and built an experimental system for application-specific integrated circuits. Based on these preparation, we have built integrated design environment from high-level synthesis to logic synthesis and improved it so as to deal with wider application area. We also show the usage of knowledge and design tools like the cost estimation with high precision. This paper presents overview of this system and several experiments. Inputs to this system are abstract behavioral specifications. This system synthesizes an initial circuit based on flow graphs generated from a input specification, and searches good design according to circuit modification by the rule-based technique. This system prepares several tools for design-estimation and verifiers. Designers can find better design from wider design space by self-feedbacks. Some examples including PDP-8 architecture are shown. System performance and future problems are also discussed.

## 1 はじめに

ユーザの要求にあった回路を短期間で正確に効率よく作るためには、より高位レベルからの自動合成ないしは設計支援が望まれる。このような観点から、近年、米国を中心に、ディジタルシステムの抽象的な動作仕様からそれを効率よく実行するレジスタ・トランスファ・レベルの回路構成を合成するという高位レベル合成に関する研究の重要性が認識され、そのための基本技術の研究が進められつつある [1]。

ここでいう高位レベル合成とは、アルゴリズム・レベルと呼ばれる動作仕様からの合成である。入力仕様は、データの入力系列から出力系列へのマッピングを表現するアルゴリズムであり、システム内の構成を制限しないようにできるだけ抽象的に与えるものとする。合成システムは、この入力仕様をもとに、その動作を実現するレジスタ・トランスファ・レベルのハードウェア記述を出力する。出力には、データパス系の情報、および、その動作を制御する制御系の情報を含まなければならない。制御系がデータパス系に統合化されていないのなら、データパスを駆動する有限状態マシンの仕様も合成しなければならない。

システムの最初の段階は、入力された動作仕様記述を内部表現にコンパイルすることである。内部表現としては、データフローと制御フローを含んだグラフ表現とする研究アプローチが多い [1]。仕様は、人間が見て分かりやすいように記述されており、ハードウェア記述言語のように、直接ハードウェアを意識して記述されるわけではない。したがって、内部表現に対する適度な最適化が望まれる。これには、ソフトウェア・コンパイラ的な最適化と、ハードウェア合成において特殊で局所的な変換が含まれる。

次の設計段階は、演算のスケジューリングとアロケーションであり、この二つは、動作仕様を回路構成に変換する上で最も重要である。また、スケジューリングとアロケーションは密接に関係があり、互いに独立に決めることは一般にはできない。スケジューリングは、演算をコントロール・ステップに割り当てることである。コントロール・ステップは同期システムの基本シーケンス単位であり、クロック・サイクルに対応する。アロケーションは、演算をハードウェア、すなわち、機能ユニット、記憶要素、通信バスに割り当てることである。

スケジューリングとアロケーションに関する基本的なアルゴリズムについては、多くの研究により十分な理解が得られてきている [1]。しかし、実際的には、与えられた動作仕様を実行する回路構成として多くの異なる種類のものが考えられる。そこで、サイクルタイム、面積、消費電力などの制約を与えて、それに適合する回路構成を選択するという設計が必要である。そのような設計を支援する上で最大の問題は、制約条件にできる限り適合した回路構成を設計するためには、非常に多くの設計可能性を検討しなければならないということである。『設計空間』は、多次元でかつ断続的な空間である上に、その形は、対象とするアルゴリズムごとに極めて個別的であるため、すべての場合を網羅する手法はない。

この解決策として、2つのものが考えられる。1つは、専門家の知識を設計過程で利用するという、いわゆるエキスパート・システムの手法を適用するものである [2]。また、各種設計ツールを有機的に組み合わせた統合的なVLSI 設計環境に知識処理手法を利用しようという試みもある [2]。もう1つは、対象とする領域を信号処理というように限定することにより、仕様や制約の記述法を制限した上で、実用的なシステムを開発しようというものである [3][4]。

このような研究の代表例としては、米国カーネギーメロン大学のシステム [5] があげられる。しかし、これはISPS[6] というインストラクション・レベルの動作仕様を入力するため、アルゴリズム・レベルよりも抽象度が若干低いといえる。また、ルールベースの手法で機能回路を自動生成する DAA[7] というツールもある。

関連する研究として、設計過程に設計者が積極的に介入することにより、機能設計レベルでの探索を実現しようというアプローチも多い [8]。代表的なものとして、米国スタンフォード大学の会話型機能論理設計システムがある [8]。これは、C 言語ライクな仕様記述を入力し、設計者と対話的によい設計結果を選択しようというものである。我々の研究は、システム側でもルールベースによりよい設計の可能性を探索するとともに、設計者もそこへ協調的に介入することを目指すものである。

対象を限定した上で実用的なものを構築しようという試みとしては、信号処理用シリコン・コンパイラ [3][4] があげられる。この代表例として IMEC 社の Cathedral-II [4] がある。これは、乗算器などの要素をハードウェア・ライブラリにもつという点で実用的である。我々の研究は、面積を精度よく見積ったり、必要があれば論理合成もできるという意味で実用性の高いものである。

我々は、これまでに高位レベルからの電子回路設計支援への知識処理技術の適用法の検討を行い [9]、信号処理を中心とした専用回路を対象として扱える実験システムの構築を行ってきた [10]。これらの準備のもとに、今回、自動論理合成システムと接続した環境を構築し、取り扱える回路の範囲を拡大するためのシステムの拡張、そこでの知識の充実、システムにおけるツールとその利用法について検討した。本論文では、システム全体の概要と各種適用実験について報告する。本システムは、抽象的な動作仕様記述を入力し、初期回路合成、回路の修正を行うことにより、よい設計を探索するものである。様々なレベルで評価ツールを持つので、設計者はシステムを利用して積極的にみずからフィードバックをかけることによって広い範囲からよい設計をみつけることができる。まず、システムの概要を説明した上で、比較的簡単なプロセッサなどの実験結果を示し、システムの評価を行なうとともに、今後の課題を論ずる。

## 2 機能論理統合設計支援システムの概要

VLSIのような大規模ディジタルシステムの機能・論理設計レベルからの設計効率の向上を目的として、近年、多くの自動論理合成システムが開発されている[11]。今回、機能論理統合設計支援システムを構築するにあたり、その中から、階層的ハードウェア記述言語H$^2$DL[12][13]を入力として論理回路を自動生成する自動論理合成システムLUNA[11]を使用した。実験システムの構成図を図1に示す。

システムへの入力となる仕様記述は、Pascalライクな手続き型のプログラミング言語による動作仕様記述である[10]。入力となる仕様記述言語の言語仕様を定めた時には、むしろユーザがビットを意識することなくアルゴリズムを容易に記述できるように、ビット操作に関する構文を用意しなかった。しかし、場合によっては、ビットを意識した記述が必要になることがある。本設計用仕様記述言語では関数が利用できるように処理系を構成しているため、ビット操作に関する関数を標準関数として定義することにより、今回それを実現した。ビット位置指定や連結の関数が用意されたため、ユーザにとっても仕様が書きやすくしかもリーダビリティもよい。これにより、純粋にアルゴリズム的なもののみではなく、かなりハードウェアを意識したものまで対象として扱えるようになった。

入力仕様記述は、ソフトウェア・コンパイラ的な手法により、中間表現に変換される[10]。中間表現は、基本的に制御フローとデータフローグラフからなる。この中間表現をもとに、制約条件を考慮してRT情報を合成する。RT情報は、データパス系と制御系の情報からなる。データパス系は、データパス表示ツールにより確認できる。制御フローも自動的に図示することができる。

本システムにおける回路合成は、中間表現からの初期回路合成とルールベースによる回路の修正からなる。初期回路合成時の演算のスケジューリングは、ASAP (as soon as possible) スケジューリングを現在行っている。初期回路合成時のアロケーションは、スケジューリングの結果をもとに、異なるコントロール・ステップの演算で共有化できるものは並列度最大で共有化するようにファシリティを割り付ける。その後、ルールによる回路変更をヒューリスティックに行い、ユーザの制約にあった回路構成を探索するものである。プロダクション・システムのパタンマッチングにはReteアルゴリズム[14]を用いて効率化を図っている。ここでの変更ルールとしては、1を加える式をインクリメントに置換するとか、0.5を掛ける式を1ビット右シフトにするといった局所的でかつ基本的な変換が主である。あるいは、並列度をさらに下げて演算器の数を減らすことによりコストを下げてみるというルールもある。

RT情報はトランスレータによりH$^2$DLのフォーマットに変換される。この段階で、検証系によりRTレベルでの詳細な動作などの検証が可能である。また、自動論理合成を施すことにより、セル数、ゲート数の精度よい

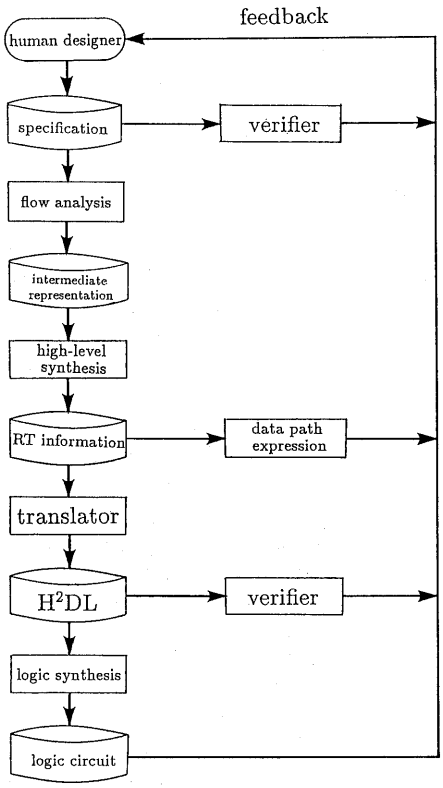



図1 機能論理統合設計支援のための実験システムの構成

評価を行うことができる。

現在、設計過程でのユーザとのインタラクションは十分であるとはいい難いが、得られた結果に対して各種検証系やツールを用いて所望の回路が得られたかどうか確認することは可能であり、さらに、中間ファイルやH$^2$DL記述ファイルを直接修正したり、変換・合成ルールを変更して設計し直すことも可能である。しかしながら、通常の利用形態では、仕様記述を書き改めて設計し直すことにより、フィードバックをかける方がふつうであろう。フィードバックループを図1中に示す。

また、論理合成システムと接続するにあたって機能の追加あるいは若干の制限を行った。主な項目を次に列挙する。

(1) 入出力データの制御方法の生成

一般に、高位レベル合成においては、モジュール外部とのデータ授受の制御方法に関する考慮はなされていないものが多い。いくつかの典型的な制御方法をテンプレートとして用意し、ユーザが選択できるようにした。ユーザが意識的に仕様中に記述するか、指定がなければ、『入力データは、処理開始から完了までの期間たえず入力されており、出力データは、処理完了から次の処理開始までの期間たえず出力する』という簡単な制御法を仮定し、RT情報に反映した。

(2) システム制御信号の生成

システム制御信号についても、一般の高位レベルの合成では、あまり考慮されていない。ここでは、処理の起動信号、リセット信号、処理終了信号、さらにクロック信号などを標準的に自動合成するようにした。

(3) 高機能演算器の処理

乗算器のような高機能演算器の処理は、H²DL のハードウェア・プロセス記述を利用して変換するようにした。ハードウェア・プロセス記述のままでは、自動論理合成はできないが、レジスタ・トランスファ・レベルでのシミュレーションは可能である。それをもとに、階層的に設計を進めるか、あるいは、利用可能な乗算器、ALU をハードウェア・ライブラリに登録し選択するようにすればよい。

(4) 同一演算器の抽出

本設計環境では、RT 情報中で既に演算と演算器の対応がついている。しかし、通常のレジスタ・トランスファ・レベルのハードウェア記述言語では、それらの区別ができないものが多い。今回は、同一演算器の式を抽出しスタティック宣言に変換して表現した。

```
program sum(n);
type    integer = unsigned bit 8;
input   n: integer;
output  o: integer;
var     j: integer;
        s: integer;
begin
   s := 0;
   for j := 1 to n do begin
      s := s + j;
   end;
   o := s;
end.
```

図 2 総和を求める例題の仕様記述例

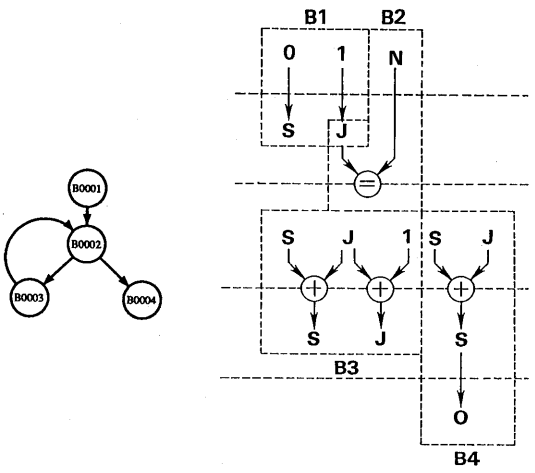



図 3 データフローと制御フロー

## 3 適用例

### 3.1 総和を求める簡単な例題

まず、システムの基本的な動作を説明するための簡単な例題として、総和を求める例題を示す。仕様記述例を図 2 に、データフローおよび制御フローを図 3 に、合成されたレジスタ・トランスファ・レベルのハードウェア記述を図 4 に、そのデータパス系を図 5 に、自動合成された論理回路 (一部) を図 6 に示す。

ここで、図 4 中の @ マークで示したスタティック宣言が同一演算器による動作を RT レベルのハードウェア記述中に含ませるために合成した部分である。

```
<INTS> SUM;
<IN>
      CLOCK,
      EXEC,
      RESET,
      N< 0 : 7 >;
<OUT>
      FIN,
      O< 0 : 7 >;
<REG>
      S< 0 : 7 >,
      J< 0 : 7 >;
<TER>
      FADD1O< 0 : 7 >,
      FADD1A< 0 : 7 >,
      FADD1B< 0 : 7 >,
      FINC1O< 0 : 7 >,
      FINC1A< 0 : 7 >;
<STC>
      O=S;
<ENDSTC>;
<STT> SUM_B: CLOCK;
      B0002:
         IF J==N
            THEN S=FADD1O;
                 FADD1A=S;
                 FADD1B=J;
                 FIN=1;
                 NEXT INIT;
            ELSE FIN=0;
                 S=FADD1O;
                 FADD1A=S;
                 FADD1B=J;
                 J=FINC1O;
                 FINC1A=J;
                 NEXT B0002;
         ENDIF;
      INIT:
         IF EXEC==1
            THEN S=0;
                 J=1;
                 FIN=0;
                 NEXT B0002;
            ELSE FIN=1;
                 NEXT INIT;
         ENDIF;
<ENDSTT>;
<EXC> RESET:
      NEXT SUM_B.INIT;
<ENDEXC>;
<STC>
@{    FADD1O=ADD(FADD1A,FADD1B,0);
      FINC1O=INC(FINC1A);
<ENDSTC>;
<ENDINTS>;
```

図 4 自動合成されたハードウェア記述 (H²DL 記述)

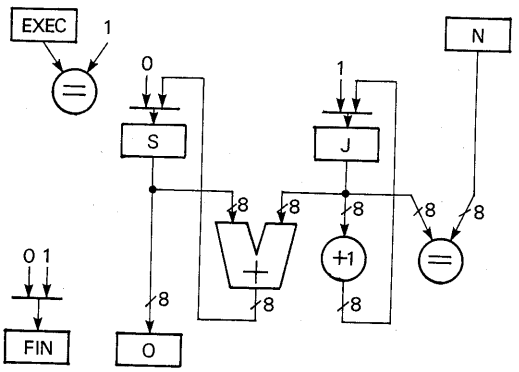



図 5 自動合成されたデータパス系

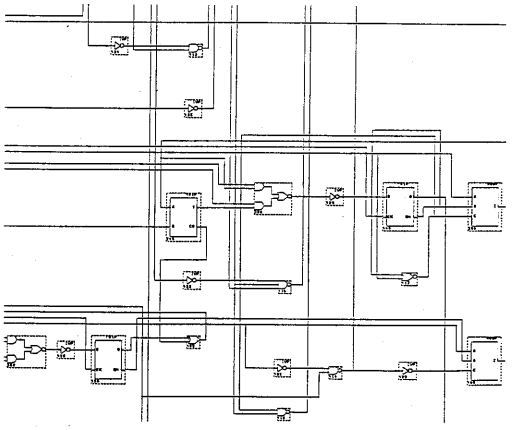


図 6 自動合成された論理回路 (一部)

```
program pdp_8(ir_req,switches);
const    n=1;
type     sig = unsigned bit 1;
         int3 = unsigned bit 3;
         int12 = unsigned bit 12;
input    ir_req : sig;
         switches : int12;
output   go : sig;
var      L,ir_enable,ed : sig;
         IR : int3;
         AC,PC,last_PC,MA,MB : int12;
         dm,m : int12;          { int }
         memory : array[n] of int12;
begin

  { fetch the instruction and decode }
    go := 1;
    MB := memory[PC];
    last_PC := PC;
    PC := PC+1;

{get instruction and effective address }
    IR := range(MB,2,0);

{effective address calculation }
   if IR <= 5
   then
     begin
       if range(MB,4,4) = 1
       then  MA := range(MB,11,5)
       else  MA := cat(range(last_PC,4,0),range(MB,11,5));
       if range(MB,3,3) = 1
       then
         begin
          if range(MA,8,0) = 1
          then memory[MA] := memory[MA]+1;
          MA := m;
         end;
     end;

{ instruction decode and execute instruction }
    case IR of
     0: begin
          MB := memory[MA];
          AC := band(AC,MB);
        end;
     1: begin
          MB := memory[MA];
          AC := AC + MB;
        end;
     2: begin
          MB := memory[MA];
          MB := MB+1;
          memory[MA] := MB;
          if MB = 0
          then PC := PC+1
        end;
     3: begin
        memory[MA] := AC;
          AC := 0;
        end;
     4: begin
          memory[MA] := PC;
          PC := MA+1;
        end;
     5: PC :=MA;
     6: begin
          if range(MB,11,3) = 1
          then
            begin
              ir_enable := 1;
              ed := 1
            end;
        end;
     7: begin
          if range(MB,3,3) = 1
          then
            begin
              if range(MB,11,11) = 1
              then
               begin
                 if range(MB,4,4) = 1
                 then AC := 0;
               end
              else
               begin
                 if ( range(MB,8,8) = 1) and (( range(MB,7,7) = 1) and (L = 1)
                                                 or (range(MB,5,5) = 1)
                                                 and (range(AC,11,11) = 1))
                 then PC := PC+1
                 else
                  begin
                    if (range(MB,8,8) = 0) and ((range(MB,7,5) = 0)
                                                 or (range(MB,7,7) = 1)
                                                 and (L = 0)
                                                 or (range(MB,6,6) = 1)
                                                 and (AC <> 0)
                                                 or (range(MB,5,5) = 1)
                                                 and (range(AC,11,11) = 0))
                    then PC := PC+1;
                    if range(MB,4,4) = 1
                    then AC := 0;
                    if range(MB,9,9) = 1
                    then AC := bor(AC,switches);
                    if range(MB,10,10) =1
                    then go := 0;
                    end;
               end
            end
          else
           begin
            case range(MB,5,4) of
            1: AC := 0;
            2: L := 0;
            3: begin
                 AC := 0;
                 L := 0;
               end;
            end;
            case range(MB,7,6) of
            1: AC := 4095-AC;          {AC = not AC}
            2: L := 1-L;               {L = not L}
            3: begin
                 AC := 4095-AC;        {AC = not AC}
                 L := 1-L              {L = not L}
               end;
            end;
            if range(MB,11,11) = 1
            then AC := AC+1;
            if range(MB,10,10) = 1
            then
              begin
                case range(MB,9,8) of
                1: AC := shl(AC,2);
                2: AC := shr(AC,1);
                end;
              end
            else
              begin
                case range(MB,9,8) of
                1: AC := shl(AC,1);
                2: AC := shr(AC,1);
                end;
              end;
           end;
        end;
    if (ed <> 1) and (ir_enable=0) and (ir_req =1)
    then
      begin
        memory[0] := PC;
        PC := 1;
      end;
end.
```

図 7 PDP-8 の仕様記述例

## 3.2 PDP-8

本システムは、アルゴリズムをソフトウェア形式で記述し、そのアルゴリズムを制約条件のもとで効果的に実行する回路を合成することを目的に開発されたものであるが、ビット演算を関数として記述することを許したために、通常のインターフェース回路やインストラクションを持つマイクロプロセッサの記述も、ある程度行なうことができるようになった。現実的には、このような周辺回路をカスタム LSI 化する要求も多いため、極めて有効であると期待できる。

そこで、このような例として PDP-8 を選び、合成実験を行なった。これを選んだ理由としては、例題として適当な規模であることと、比較評価がしやすいことがあげられる [8]。

PDP-8 の仕様記述例を図 7 に示す。仕様を書く場合、並列性を記述する構文が用意されていないため、予め分

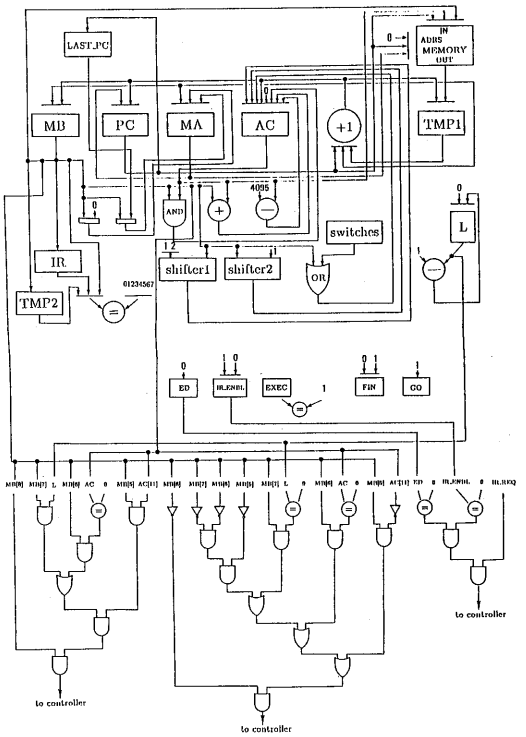
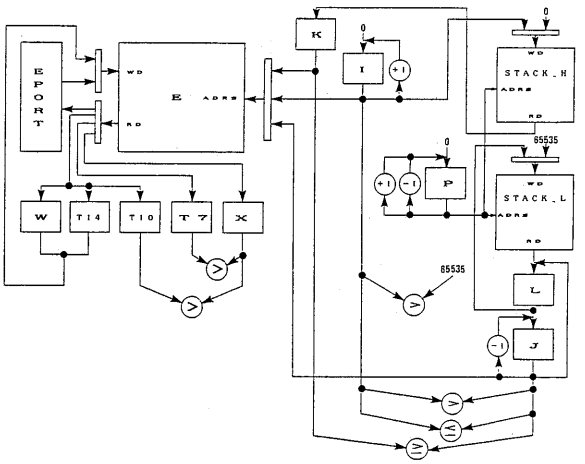



図 8 合成されたデータパス系 (主要部)

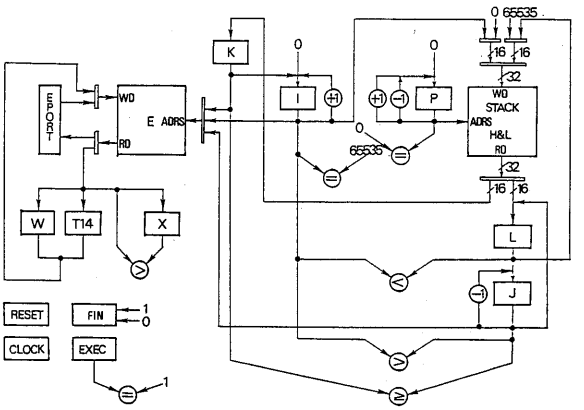
かっている並列性を記述することができない。したがって、フロー解析の結果抽出された並列性の部分を除き、ハードウェアとしての効率がよくない。合成されたデータパス系を図 8 に示す。この例題について、TC17G シリーズのマクロセルを用いて自動論理合成の実験を行ったところ総セル数で約 800 となった。このように、論理合成を施すことにより、精度良くコストを見積ることができる。

### 3.3 クィック・ソート

以前の論文 [10] で扱ったクィック・ソートの例をさらにルールベースを用いて修正した結果のデータパス系を図 9 に示す。初期回路に比べて、比較器の数が減っていることが容易に分かる。また、$stack_h$、$stack_l$ という 2 つの RAM のアドレスの値が常に同じで参照、代入が実行されるため、それらを連結して良いというルールを加え、併合することができた。データパス系については、このようなルールベースによる修正により、人手設計により一層近づけることができる。しかしながら、制御系を決定する制御フローは、もともとの仕様から ASAP スケジューリングにより作られたものを修正する機能を付加していないため、現状では大幅にリファインすることができない。したがって、状態数が人手設計よりも若干増える傾向にある。ビット長をすべて 4 ビットに書き改めて、




(a) 初期データパス系 (主要部)




(b) 修正されたデータパス系

図 9 クイックソートの初期回路と修正された回路

TC17G シリーズのマクロセルを用いて自動論理合成の実験を行ったところ総セル数でやはり約 800 となった。

## 4 評価

ここにあげた例題の他に、数多くの信号処理アルゴリズムの仕様を記述して、実験を行ってみた。しかし、信号処理アルゴリズムは乗算を含むことが多く、現状では仕様中に乗算を記述するとハードウェアプロセスに変換することにしているため、レジスタ・トランスファ・レベルで検証を行うことはできても、論理合成はすぐにはできない環境にある。したがって、現状で定量的な評価をするのは難しい。

また、信号処理の場合演算精度が重要であるが、現状では浮動小数点形式を論理合成まで接続することは許していない。ただし、仕様としては浮動小数点も書けるの

で、それを展開する規則を追加することにより処理することはできる。なお、今回の実験では、ユーザに固定小数点の小数点位置も意識して書き改めてもらったものを入力とした。単純な演算については内部でビット位置の補正をするようにしてあるが、一般的にそれを補正することは困難であるためである。

さらに、システム全体を知識ベースシステムという観点からとらえると、本設計支援システムは、フロー解析のようなアルゴリズミックなツールとルールベース的なツールを適切に組み合わせた構成をとっている。設計の自由度として、仕様記述を与える際の自由度、フロー解析(スケジューリング)を行う際の自由度、初期回路合成時の自由度、ルールベースによる回路の最適化の自由度が許される。本システムの仕様記述は手続き型プログラミング言語の形で与えるため、同じアルゴリズムでも多くの異なる記述法がある。このような仕様記述を与える自由度は、ユーザが操作することができる。フロー解析の部分は、手続き的に作成してあり、初期回路合成もテンプレート的に実現している。回路の修正はシステムが初期回路をもとに半自動的に行う。しかしながら、個々の変換プロセスとそこでの設計の自由度のみならず、全体的な協調動作についても十分検討する必要があると考えられる。

例えば、フロー解析部は、通常のソフトウェア・コンパイラ的な手法により実現されているため処理効率が良い反面、中間表現生成が固定化されているため、回路設計において有用な情報が失われることがある。その例を図10に示す。図10 (a) と図10 (b) は、動作としては同じであるが、表現が異なるものである。図10 (a) の表現

$$
\begin{aligned}
&Y = A0 \times X - B1 \times W1; \\
&\qquad \downarrow \\
&T001 = A0 \times X; \\
&T002 = B1 \times W1; \\
&Y = T0001 - T0002;
\end{aligned}
$$

(a)

$$
\begin{aligned}
&Y = 0; \\
&Y = Y + A0 \times X; \\
&Y = Y - B1 \times W1; \\
&\qquad \downarrow \\
&Y = 0; \\
&T001 = A0 \times X; \\
&Y = Y + T0001; \\
&T002 = B1 \times W1; \\
&Y = Y - T0002;
\end{aligned}
$$

(b)

図10 仕様記述による中間表現の違い

の方が、式の数が少ないが、図10 (b) の方がパイプライン処理の可能性が表現されている。この図10 (a) と図10 (b) の違いは同じアルゴリズムの異なる記述法の1つの例でもある。

## 5 むすび

アルゴリズム・レベルの動作仕様記述から機能設計を行うシステムを自動論理合成システムと接続し、機能論理統合設計支援環境を構築した。取り扱える回路の範囲を拡大するための仕様記述言語の記述能力を高めた。そこでの知識の充実を図るとともに、様々なレベルあるいは精度での評価ツールを設けた。

本システムは、抽象的な動作仕様記述を入力し、初期回路合成、回路の修正を行うことにより、よい設計を探索するものである。様々なレベルで評価ツールを持つので、設計者はシステムを利用して積極的にみずからフィードバックをかけることによって広い範囲からよい設計をみつけることができる。本システムのもつ回路の修正規則は局所的な変更であるので、設計空間の広い範囲を探索するには、探索の初期値にあたる初期回路を変更することが必要である。設計者は、仕様記述の自由度を利用したり、必要により設計過程に介入することにより、よい設計を得ることができる。つまり、本システムにおける知識の利用とは、システム内の知識の利用と設計者の知識の利用という2つの側面をもつ。

比較的簡単なプロセッサの実験結果を示し、システムの有効性を示した。仕様記述言語にビット操作を記述できるような関数を用意することにより、周辺回路やインストラクションレベルの記述もある程度可能なように拡張したため、簡単なプロセッサ的な回路も扱えることは確認できた。

最後に、今後の研究課題をあげる。まず、仕様記述の問題があげられる。ユーザがハードウェアをあまり意識せずに、ソフトウェアの感覚で抽象的に仕様を記述するという点では、本研究のようなアプローチは有効である。しかし、場合によっては、入出力の詳細なタイミング、あるいは、予めわかっている並列動作を記述する構文を用意し、並列性およびビット操作を仕様中に明記したいことがある。ソフトウェアレベルからインストラクションレベル，レジスタ・トランスファ・レベルまで任意に仕様を書ける自由度をユーザに与えることが望まれる。

また、ヒューマン・インターフェースを含めた設計者との対話的な協調動作について、ますます研究していく必要があると感じる。現状でも簡単な表示ツールや検証系は存在するが、設計者の介入をより積極的に許すためには必ずしも十分であるとはいえない。特に、設計者に対して、設計段階も含めて、入力と出力との対応付けが分かりやすく示すことが望まれる。

## 謝 辞

自動論理合成システムを利用するにあたり協力して下さった(株)東芝総合研究所情報システム研究所の西尾誠一氏、黒澤雄一氏、開發貴久氏に感謝します。

## 参考文献

[1] Macfarland, M. C., Parker, A. C. and Camposano, R.: "Tutorial on High-Level Synthesis," Proc. of 25th Design Automation Conf., pp. 330-336 (1988).

[2] 川戸, 角田: "VLSI CAD への知識工学の応用," 人工知能学会誌, Vol. 2, No. 4, pp. 410-418 (1987-12).

[3] Gajski, D. D. (ed.): "Silicon Compilation," Addison-Wesley, U.S.A. (1988).

[4] De Mann, H., Pabaey, J., Six, P. and Claesen, L.: "Cathedral-II: A Silicom Compiler for Digital Signal Processing," IEEE Design & Test of Comput., pp. 13-25 (December 1986).

[5] Thomas, D. E., Dirkes, E. M., Walker, R. A., Rajan, J. V., Nestor, J. A. and Blackburn, R. L.: "The System Architect's Workbench," Proc. of 25th Design Automation Conf., pp. 337-343 (1988).

[6] Barbacci, M. R.: "Instruction Set Processor Specifications (ISPS): The Notation and Its Applications," IEEE Trans. Comput., Vol. C-30-1, pp. 24-40 (January 1981).

[7] Kowalski, T. J. and Thomas, D. E.: "The VLSI Design Automation Assistant: What's in a Knowledge Base," Proc. of 22nd Design Automation Conf., pp. 252-258 (1985).

[8] 尾谷, Sun Young Hwang, Tom Blank, Tom Rokicki: "会話型LSI機能論理設計システム Hermod," 信学技報, VLD88-25 (1988).

[9] 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).

[10] 竹沢, 上野, 白井: "特定用途向け回路アーキテクチャ設計支援システム," 信学論 D, Vol. J71-D, No. 10, pp. 1931-1938 (1988-10).

[11] 黒沢, 増淵, 西尾, 上田, 宮田: "自動論理合成システム LUNA の適用と評価," 情報処理学会設計自動化研究会, 37-1 (1987-5).

[12] 宮田, 西尾, 山崎: "階層的ハードウェア設計言語 $H^2DL$ の思想," 情報処理学会設計自動化研究会, 22-1 (1984-7).

[13] 西尾, 宮田, 山崎: "階層的ハードウェア設計言語 $H^2DL$ の言語仕様 — 内部仕様記述とハードウェア・プロセス記述 —," 情報処理学会設計自動化研究会, 22-2 (1984-7).

[14] Forgy, C. L.: "Rete: A Fast Algorithm for the Many Pattern/Many Object Pattern Match Problem," Artificial Intelligence, Vol. 19, pp. 17-37 (1982).