設計自動化 27－2
（1985. 7. 16）

信号処理プロセッサの機能設計支援システム

竹沢　寿幸　　永井　保夫　　白井　克彦
（早稲田大学　理工学部）

1．はじめに

近年、ディジタル信号処理技術およびハードウェア技術の発達に伴って、種々の信号処理プロセッサが製作され、実用に供せられるようになった。さらに、信号処理プロセッサにおいては、ユーザ毎にインプリメントしたい信号処理アルゴリズムや、入出力条件、処理時間に対する制約が異なるため、カスタムＬＳＩへの潜在的な需要は多いと思われる。また、信号処理に限らずシステム設計のための有用な支援ツールの開発が望まれている所である。そこで、我々は信号処理プロセッサを対象例として取りあげ、設計支援システムを開発することにした。

信号処理プロセッサのユーザにとっては、必要な信号処理アルゴリズム、入出力条件、処理時間に対する要求等は既に明確であろう。しかしながら、これだけの情報で信号処理プロセッサを設計することは非常に難しい。従来の設計自動化の研究は、ハードウェア記述言語を用いてハードウェアの動作を記述して論理設計を行い、さらに、実装設計を行なうという形のものが多いためである。

そこでまず、信号処理アルゴリズム、入出力条件、処理時間に対する要求を入力として、信号処理プロセッサの機能設計を行なうことを考えてみた。今回は、特に、データパス系の設計を対象にしている。

2．システム概要

本システムの処理の概要は、図１のようになっている。本システムは、まず、入力された仕様の解析を行い、データフローやコントロールフローを表す中間表現を生成する。次に、その中間表現をもとに機能回路を合成する。そして、その回路の性能評価を行い、仕様中に記された制約を満足する範囲内で、回路の変更を繰り返す。このような回路合成及び変更の過程に、知識工学的手法を適用している。

仕様は LISP のＳ式で記述される。これは、主として、レジスタのようなハードウェアを直接意識しないで記述された信号処理のアルゴリズムから成る。

前処理部では、この仕様として入力された LISP プログラムに対して、通常のコンパイラのように、字句解析、構文解析等の処理を行い、機能回路を合成するためのもととなるデータフローグラフ及びシンボルテーブルを生成する。信号処理プロセッサという特殊性から、演算の並列実行可能性、アルゴリズム中に出現する演算の種類、頻度、順序、さらに変数の属性や求められる精度といったものが重要であるため、このような中間表現を生成するのである。

次に、機能回路合成部における回路合成について述べる。まず、中間表現中のデータフローグラフに対して、演算器や記憶要素のような構成要素を１対１に写像する。構成要素に関する知識はすべて、知識ベース中に蓄えておく。

信号処理プロセッサでは、通常、高速な処理が望まれるため、合成された回路の性能評価を行なう。ここでは特に、入力されたアルゴリズ

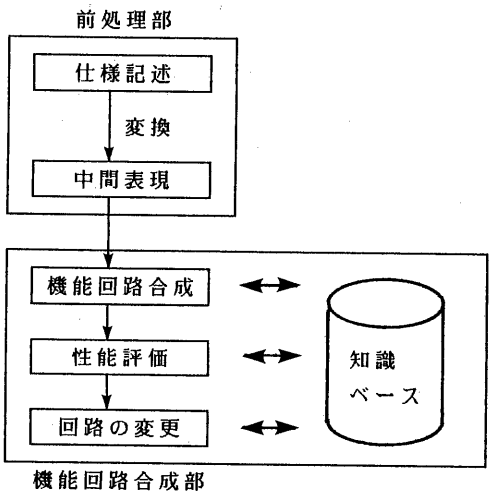

図１　システムの処理の流れ

```
specification ::= (PROGRAM program-name declaration-block algorithm-block) ;
declaration-block ::= (DCL-IO
                        {(I/O-variable {parameter-description})}) |
                       (DCL-RESTRICTION
                        {(condition-variable {parameter-description})}) ;
algorithm-block ::= (PROCESS {statement}) ;
statement ::= action-atement | control-statement ;
action-statement ::= (variable = expression) ;
expression ::= constant |
               (unary-operator expression) |
               (expression binary-operator expression) |
               variable ;
operator ::= arithmetic-operator | logic-operator ;
arithmetic-operator ::= + | - | * | // | mod | ^ ;
logic-operator ::= ! | & | < | > | =< | => | = ;
control-statement ::= (COND {(condition {statement})}) |
                      (WHILE-DO condition {statement}) |
                      (DO (index-part exit-condition) {statement}) ;
```

図２　仕様記述言語構文規則（要旨）

```
(program parcor-filter
    (dcl-io (s   (input  (stream    (serial)))
                 (width  (8))
                 (type   (array     (n))
                         (fixed     (7)))
                 (role   (port      (input))))
            (e   (output (stream    (serial)))
                 (width  (8))
                 (type   (array     (n))
                         (fixed     (7))
                         (precision (7)))
                 (role   (port      (output))))
            (rk  (input  (stream    (serial)))
                 (width  (8))
                 (type   (array     (m + 1))
                         (fixed     (7)))
                 (role   (port      (input))))
            (m   (step   (12)))
            (n   (point  (128)))
            ((ft gt gto)
             (type   (array   (m + 1))))))

    (dcl-restriction
            (sampling-frequency ((12.5 ^ 3))))

    (process
            (do ((i 1 (i + 1)) (i = 16))
                (gto{i} = 0.)
                (ft{i} = 0.))
            (e{1} = s{1})
            (m1 = m + 1)
            (do ((j 2 (j + 1)) (j = n))
                (ft{1} = s{j})
                (gto{1} = s{j - 1})
                (do ((i 2 (i + 1)) (i = m1))
                    (ft{i} = ft{i - 1} - rk{i - 1} * gto{i - 1})
                    (gt{i} = gto{i - 1} - rk{i - 1} * ft{i - 1})
                    (gto{i} = gt{i}))
                (e{j} = ft{m1})))))
```

図３　ラティス・フィルタの記述例

ムの大体の処理時間を予測する。評価した処理時間に余裕があれば、合成された回路に対して演算器の併合などの回路の変更を行なう。回路の変更規則は、知識ベース中にルール形式で記述しておく。さらに、変更した回路に対して性能評価を行い、要求される処理時間を満足する範囲内で回路の変更を繰り返す。ハードウェアのコストについては、演算器のような構成要素の数が少ない方が、コストも小さいとみなして、回路の変更は演算器の数を減らす方向で行なう。

図１の機能回路合成部の知識ベースには、構成要素に関する知識、回路の変更規則が蓄えられている。

本システムの出力としては、入力されたアルゴリズムに対する機能レベルの最適な回路構成を出力するというよりは、そのアルゴリズムに対して考えられるいくつかの回路構成と、それに対する大体の性能を出力するのである。

３．各部の処理

３－１．仕様の記述

本システムの入力となる仕様の記述について説明する。本システムは LISP 言語で作成されているため、仕様も LISP のＳ式の形となっている。これは大きく分けて PROGRAM という頭書き、プログラム名、変数、特に入出力変数の属性の宣言部、処理時間その他に関する制約条件の宣言部、信号処理アルゴリズムの記述部から成る。

仕様記述言語の構文規則を図２に示し、仕様の記述例を図３に示す。

図３の例で parcor-filter というシンボルが、このプログラムの名前である。dcl-io に続くリストが変数の属性の宣言部、dcl-restriction に続くリストが制約条件に関する宣言部、process に続くリストが処理アルゴリズムの記述部である。

dcl-io 部において、s、e、rk といったシンボルが変数名で、それに続くリストがその変数の属性を表している。例えば、変数ｓは stream 形式シリアル入力で、８bit 幅、そして１からＮまでの一次元配列で、入力ポートから入力されるという宣言である。

dcl-restriction 部では、このフィルタのサンプリング周波数が 12.5 kHz であることを宣言している。つまり実時間で処理するためには、80μsec 以内ですべての処理が完了しなければならないことを意味する。

process 記述部では、構造化プログラミングに似た形式でアルゴリズムを記述する。制御構造として許されるのは、条件分岐に相当する cond 節、条件が成立する間繰り返す while-do 文、ある決った回数だけ繰り返す do 文である。例えば、

```
(do  ((i 1 (i + 1)) (i = 16))
     (gto{i} = 0)
     (ft{i} = 0))
```

という文は、ｉが１から１ずつインクリメントしてｉ＝16 まで gto{i} と ft{i} に０の代入を繰り返すことを意味する。なお、LISP 言語で記述するために、配列は中括弧で表現している。

ところで、この仕様記述言語と通常のハードウェア記述言語とを比較すると、process 記述部がほぼ動作記述に相当すると言える。また、入出力部に関しては、すでにほぼその構造が決定されているといえる。ただし、その内部の構造については、かなりの設計の自由度が許されている。

３－２．フロー解析

図４に解析プロセス、図５にフローグラフの例を示す。

図４に示すように、アルゴリズムの動作記述である process 部から、データフローグラフ及びコントロールフローグラフが作成され、それ以外の宣言部（dcl-io 及び dcl-restriction 部）から、シンボルテーブルが作成される。

フローグラフの生成手順を説明する。まず、アルゴリズムの大域的な制御の流れを表現するために基本ブロックに分割する。基本ブロックとは、シーケンシャルに実行される式の集合である。例えば DO 文のような繰り返し文の内容

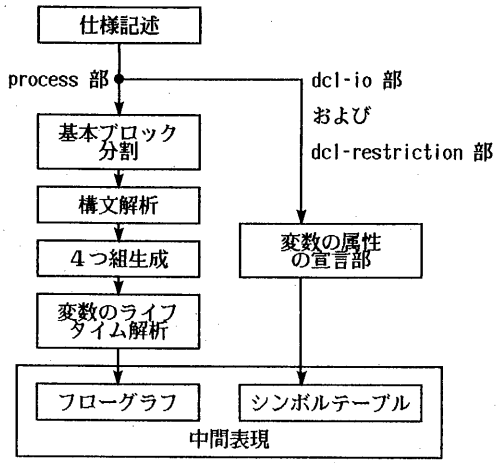

図4　解析プロセス

がそれに当る。

次に、基本ブロック中の個々の文に対して、字句解析及び演算子順位法に基づく構文解析を行なう。そして実行すべき演算の順序に4つ組の中間表現形式を作成する。この4つ組は

（OP　α_1　α_2　α_3）

の形式をしており、α_1 と α_2 に演算 OP をほどこした結果が α_3 であるということを意味している。

最後に、変数のライフタイム解析を行なう。これは、変数の値がどの時点で定義されるか、さらにその後参照されるかどうかを調べるものである。例えば、

```
(blk2 0 (= (e{1} 3) (s{1} 0) t0003)
      1 (+ (m 0) 1 t0004)
        (= (m1 5) t0004 t0005))
```

というフローグラフにおいて、(e{1} 3) というリスト中の3という数字が変数 e{1} に対する新たな定義がなされたことを示し、t0003 に対応する。また、(s{1} 0) の中の0という数字は s{1} が入力変数であることを示している。また、下線を引いた数字0、1はフロー中におけるその演算の順位を示す。

このようにして作成されたフローグラフの例が図5である。図5のうち、上のものが大域的な制御情報を示す。下のものが、4つ組の中間

```
(process
 (do ((i 1 (i + 1)) (i = 16)) (blk1))
 (blk2)
 (do ((j 2 (j + 2)) (j = n))
     ((blk3)
      (do ((i 2 (i + 1)) (i = m1))
          (blk4-1 blk4-2 blk4-3))
      (blk5))))

((blk1 1 (= (gto{i} 1) 0.0 t0001)
       1 (= (ft{i} 2) 0.0 t0002))
 (blk2 0 (= (e{1} 3) (s{1} 0) t0003)
       1 (+ (m 0) 1 t0004)
         (= (m1 5) t0004 t0005))
 (blk3 0 (= (ft{1} 6) (s{j} 0) t0006)
       0 (= (gto{1} 7) (s{j-1} 0) t0007))
 (blk4-1 1 (* (rk{i-1} 0)
              (gto{i-1} 7)
              t0008)
           2 (- (ft{i-1} 6) t0008 t0009)
             (= (ft{i} 10) t0009 t0010))
 (blk4-2 1 (* (rk{i-1} 0)
              (ft{i-1} 6)
              t0011)
           2 (- (gto{i-1} 7) t0011 t0012)
             (= (gt{i} 13) t0012 t0013))
 (blk4-3 3 (= (gto{i} 14)
              (gt{i} 13)
              t0014))
 (blk5 3 (= (e{j} 15) (ft{m1} 10) t0015)))
```

図5　データフローグラフの例

表現形式によるデータフローを示している。図中の blk1、blk2 といったシンボルが、基本ブロックを表している。なお、これは制御フローとデータフローを対応付けるポインタとしての働きを兼ねている。

また、仕様記述中の変数の属性の宣言部などから、シンボルテーブルが作成される。

3－3．構成要素の記述とハードウェア変更規則

機能回路合成及び変更手順について説明する。

前節で述べたようにして作成された中間表現中の制御フローグラフとデータフローグラフをもとに、知識ベース中の構成要素の知識を参照して、まず機能回路を合成する。

知識ベース中の構成要素は、フレーム表現で記述される。Adder の表現例を図6に示す。フレームとはもともと Minsky により提案されたもので、人工知能、知識工学の分野で一般的に良く用いられている知識表現方法の一つである。図6に示すように、この中に Adder の持つべき属性とその標準的な値を蓄えておく。このよ

うなものをクラスと呼ぶ。各属性に実際の値を持つものをインスタンスと呼ぶ。

最初の回路合成はデータフローグラフの各演算や変数に、適当な演算器や記憶要素のインスタンスを対応付けることで行われる。

このようにして合成された回路の変更について説明する。

回路の変更規則は IF-THEN 形式で記述され、知識ベース中に蓄えられる。そして、インタプリタでそれを解釈、実行するというプロダクションシステムのアプローチを採用している。この場合の回路変更は、むしろ試行錯誤的な探索が有効であろうと考え、前向き推論を採用している。つまり、条件部とのマッチングをとり、もし適合するものがあれば、その動作部を実行するというものである。実際の変更手順は図７のような流れとなる。

まず、回路の変更が失敗した場合に備えて回路データのバックアップをとる。次に知識ベース中の変更規則とのマッチングをとる。もし適合するルールが存在すればそれを発動し、回路の変更を行なう。そして変更された回路の性能を評価し、仕様中に記述された制約つまり処理時間に対する要求を満足するかどうかを調べる。もし満足するようであれば変更を繰り返す。もし満足しなければその回路変更は失敗であるので、回路のデータをバックアップに戻す。そしてその失敗したルールを取り除いたり、修正したりしてから回路の変更を繰り返す。また、変更規則とのマッチングが取れなかった場合には、別の変更規則のセットを選んでから回路の変更を繰り返す。もはや、変更規則の修正や変更ができなくなった場合には自動的に終了する。

回路の変更規則は、現在、

○もし加算機能のみの Adder が複数個あったら、どれか適当な２つを選んで１つの Adder に置き換えよ。

○もし加算と減算があったらＡＬＵにまとめてみよ。

というような、ハードウェア量を減少する方向のルールが全部で約 25 個用意してある。つまりコストをハードウェア量で見積ることにして、コストを減少させるような規則から成る。

従って、最初の回路の性能評価時に既に仕様中の制約を満足しない場合には、そのアルゴリズムは要求される制約内で設計するのは不可能として、回路の変更は行われない。

ただし、今後は、対象が信号処理プロセッサであることを考慮して、例えば高速化を計るための規則といったもっと信号処理プロセッサのアーキテクチャ向きの変更規則を付加していく必要があると思われる。

```
(adder (type (value (class)) (to-fill (mktype)))
       (need-attribute (value (type)
                              (ako)
                              (width)
                              (function)
                              (input)
                              (output)
                              (cost)
                              (speed)))
       (ako (value (function-unit)) (to-fill (mkako)))
       (width (default (16)))
       (function (default (add) (sub)))
       (input (default (2)))
       (output (default (1)))
       (cost (default ((add 2)) ((sub 4))))
       (speed (default ((add (1 14) (2 15) (4 20)))
                       ((sub (1 15) (2 16) (4 22)))
                       ((delay (1 5) (2 5) (4 5)))))
       (instance (value (adder0001)))
       (number (value (2))))
```

図６　Adder の知識表現例

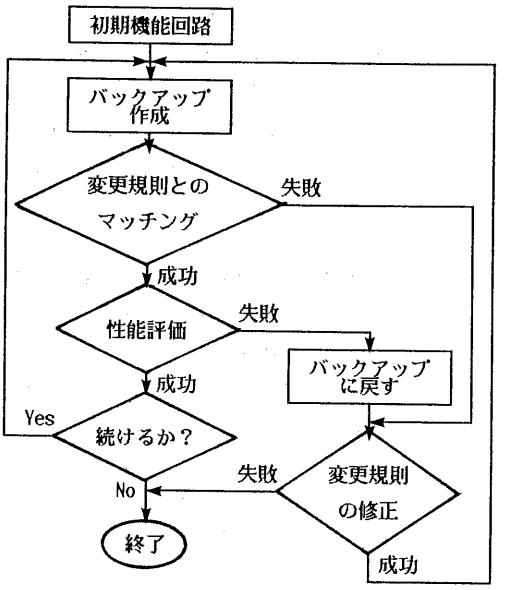

図7　回路の変更手順

3－4．合成された回路の性能評価

設計された回路の性能評価について述べる。対象が信号処理プロセッサであるので、性能の中でも特にその処理時間を評価の対象とした。

知識ベース中の構成要素のクラス表現中に、その構成要素の標準的な処理時間あるいは遅延時間が蓄えられている。ある種の記憶要素のように、標準的な処理時間にさえ幅があるものは、クラスの階層的な表現により、その管理を可能としている。また変更規則により、そのような構成要素の高速なものから低速なものへの変換が可能である。そのようなクラス中の標準的な値を用いて、インスタンス生成時に、そのインスタンスの処理時間なり遅延時間なりを算出する。

このようにして、データフローグラフに対応づけて作成されたインスタンスを用いて、アルゴリズム中のある基本ブロックないしは、全体の処理時間を評価する。

図8を例にすると、実際には次のような式に従って処理時間の評価がなされる。

$$T_{total} = T_i(OP2) + T_o(OP2)$$
$$T_i(OP2) = \mathrm{Max}\,[T_o(OP1) + T_i(OP1),\ T_s(S3)]$$

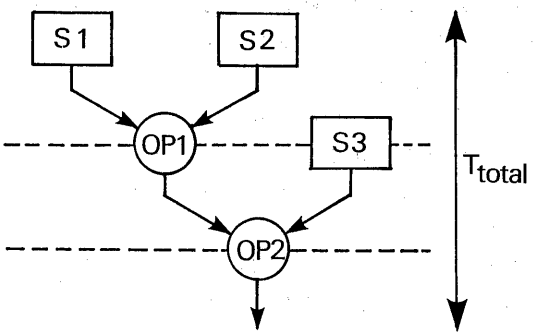

図8　性能評価の例

$$T_o(OP2) = T_{op}(OP2) + T_d(OP2)$$
$$T_o(OP1) = T_{op}(OP1) + T_d(OP1)$$
$$T_i(OP1) = \mathrm{Max}\,[T_s(S1),\ T_s(S2)]$$

ここで、T_{total} がこの例の全体の処理時間、T_i は入力が決定されるまでの時間、T_o は入力されてから出力が決定するまでの時間、T_s がストレージへのアクセス時間、T_{op} が演算時間、T_d が遅延時間を意味している。

ある基本ブロック内では、その基本ブロックに対する入力変数、または定数までさかのぼって、その出力の値が決定される時間を評価する。

また、あるデータフローと別のデータフローが並列に並んでいる場合については、その間で演算器などが共通でない場合には２つのうちの最大値をもってその処理時間とし、演算器などが重複している場合にはその加算したもので処理時間とみなす。

繰り返す回数の分っている繰り返し文の場合には、その回数倍したものでその繰り返し文全体の処理時間とみなす。ただし、並列に処理される場合にはその限りではない。

現在行っている性能評価は厳密なシミュレーションを行っているわけではない。言い換えれば、得られた回路構成に対して予想される大体の処理時間を求めているのである。

4．実験例

4－1．ラティス・フィルタ

本システムの実験例として、まず、ラティス

表1　ラティス・フィルタの回路例

回路番号	主な構成要素	処理時間
A	乗算器2個 加算器2個 RAM，ラッチ	4 μs
B	乗算器1個 加算器2個 RAM，ラッチ	7 μs
C	乗算器1個 加算器1個 RAM，ラッチ	7 μs
D	乗算器1個 ALU1個 RAM，ラッチ	33 μs
E	ALU1個 RAM，ラッチ	6ms

・フィルタをとりあげ、そのデータパス系の設計を行った。システムへの入力は、図3にあげた仕様記述例のようになる。仕様はフィルタのアルゴリズムと入出力条件として 8 bit シリアル、サンプリング周波数 12.5 kHz、フィルタの段数 12 段、1フレーム当りのサンプリング・ポイントを 128 ポイントとした。制約条件としては、実時間で処理するために、80 μsec 以内で処理が終了する必要がある。

本システムの処理結果をまとめたものを表1に示す。この例では、主な回路の構成としては表1に示すような5種類のものが得られた。ただし、表1中の回路Eについては、処理時間の評価が約 6 msec と予想されるため、失敗となったものである。

4－2．DPマッチング

別の例としてDPマッチングを考えてみた。DPのアルゴリズムは、図9のようなDPパスによって絶対値距離で計算し整合窓幅5とした。

入力パタンは、10 msec 毎に、8 bit 精度で 16 個の係数がパラレルに入力されるとした。標準パタンは6種類で 16 個の係数が 20 フレーム分あると仮定した。制約条件は、実時間で処理するとすれば 10 msec 以内ですべての処理が終了する必要がある。

DPのアルゴリズムを実現する上で、2つの数の最小値をとるとか最大値をとるといった演算の処理が問題となる。今回はとりあえず、はじめのうちは比較演算を行なう MIN と MAX という機能ユニットを仮定してハードウェアに写像した。MIN および MAX という仮想的な機能ユニットの演算時間は約 250 nsec と仮定して、性能評価に用いた。そして、変更する過程でALUに変換するようにした。

絶対値の処理は、条件分岐を用いてアルゴリズム中に記述した。また、多次元の配列の取り扱いは、それを1次元に置き換えて処理を行った。

このようにして処理した結果を表2に示す。MIN や MAX の機能ユニットの仮定をやめると、あまり時間に余裕のないことが分る。回路構成例を図 10 に示す。

表2　DPマッチングの回路例

回路番号	主な構成要素	処理時間
A	加算器12個 MIN 3個，MAX 1個 RAM，ラッチ	2.6ms
B	加算器12個 MIN 2個，MAX 1個 RAM，ラッチ	3.2ms
C	加算器11個 MIN 2個，MAX 1個 RAM，ラッチ	3.4ms
D	加算器9個 MIN 2個，MAX 1個 RAM，ラッチ	3.5ms
E	加算器6個 ALU1個 RAM，ラッチ	6ms
F	ALU2個 RAM，ラッチ	11ms
G	ALU1個 RAM，ラッチ	11ms

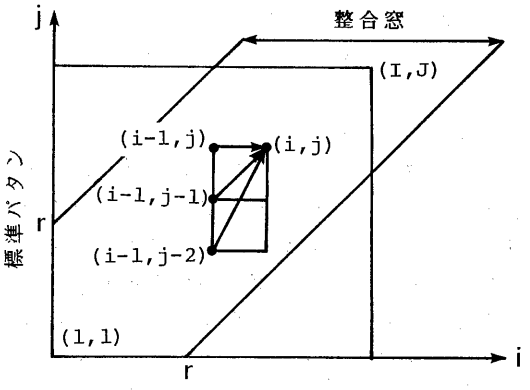

図9　DPパス

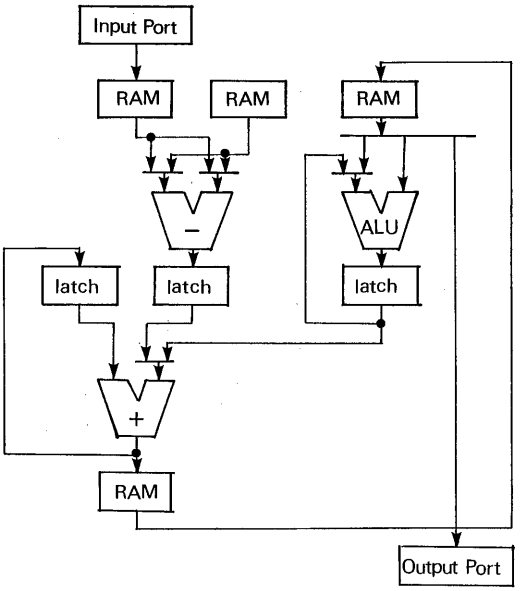

図10　ＤＰマッチングの回路例
（表２中の回路Ｅ主要部）

５．まとめ

信号処理アルゴリズムと、処理時間に対する要求を入力とした信号処理プロセッサの機能設計支援システムの概要と、データパス系を対象としたいくつかの設計例について報告した。

今後は、もっと複雑な例が取り扱えるように知識ベース中の知識を強化する必要がある。また、得られたデータフロー系に対する制御回路の設計は将来の課題である。さらに、仕様記述言語の記述能力の見直しや厳密なフロー解析も必要であるが、本システムの全体の枠組は信号処理以外にも勿論適用できるものである。現在、マン・マシン・インターフェースの観点から、設計された回路図を表示することを準備している。

参考文献

[1] Kahrs, M., "Silicon compilation of a very high level signal processing specification language", in: Van Valkenburg M.E. et al. (eds.), "VLSI Signal Processing", IEEE PRESS, New York, (1984).
[2] Tseng, C. and Siewiorek, D.P., "Facet: A Procedure for the Automated Synthesis of Digital Systems", Proc. of 20th Design Automation Conference, pp.490-496, (1983).
[3] Hafer, L.H. and Parker, A.C., "A Formal Method for the Specification, Analysis, and Design of Register-Transfer Level Digital Logic", Trans. on Computer-Aided Design, Vol.CAD-2, No.1, pp.4-18, (1983).
[4] Stefik, M., "Planning with Constraints", Artificial Intelligence 16, pp.111-140, (1981).
[5] Minsky, M., "A Framework for Representing Knowledge", in: Winston, P.H. (ed.), "The Psychology of Computer Vision", McGraw-Hill, USA, (1975).
[6] Winston, P.H. and Horn, B.K.P., "LISP", Addison-Wesley, USA, (1981).
[7] "The Bipolar Digital Integrated Circuits Data Book", Texas Instruments, USA, (1978).
[8] Shirai, K., Nagai, Y., Takezawa, T., "Functional Level Design System for Digital Signal Processors", Proc. of IFIP Int. Conf. on VLSI85, Aug., (1985).
[9] 永井保夫，竹沢寿幸，白井克彦：”知識工学的手法を用いた信号処理プロセッサの機能設計支援システム”，昭和60年信学会全国大会，Part 6, pp.250, (1985).
[10] 中村行宏，小栗　清：”ハードウェア記述言語とその応用”，情報処理，Vol.25, No.10, pp.1033-1040, (1984).
[11] 平山正治：”シリコン・コンパイラ”，情報処理，Vol.25, No.10, pp.1153-1160, (1984).
[12] 川戸信明，斎藤隆夫：”論理装置のＣＡＤにおける知識工学の応用”，情報処理，Vol.25, No.10, pp.1161-1168, (1984).
[13] 内田俊一：”信号処理用プロセッサ”，情報処理，Vol.18, No.4, pp.402-409, (1977).
[14] 内田俊一，森　秀樹：”信号処理プロセッサ”，電子通信学会誌，Vol.82, No.11, pp.1281-1288, (1979).
[15] 丸田力男：”音響処理用ＬＳＩ”，日本音響学会誌，Vol.39, No.11, pp.750-755, (1983).
[16] 迫江博昭：”連続発声した単語音声を効率的に認識する２段ＤＰマッチング”，日経エレクトロニクス，1983年11月 7日号，pp.171-208, (1983).
[17] 斎藤収三，中田和男：”音声情報処理の基礎”，オーム社，(1981).
[18] 中田育男：”コンパイラ”，産業図書，(1981).