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†

白井克彦†† 竹沢寿幸†† 永井保夫††*

今後の VLSI 設計においては，個別的で多様な要求に対してどのようにしてその機能を的確に実現するかが重要な問題の一つである．例えば，信号処理のユーザにとっては，ユーザごとにアルゴリズムや処理時間に対する制約が異なるうえに，高速化，小型化の要求が強いため，高級言語で記述された仕様から回路の設計支援がなされれば非常に有効であろう．そこで，高級言語で記述されたアルゴリズムと処理時間に対する制約からデータパス系を設計する方法を検討してみた．まず，入力されたアルゴリズムのフロー解析を行い，演算器や記憶要素を割り当てる．次に，プロダクション・システムの手法により要求される制約を満足する範囲内で回路の変更を繰り返す．回路の性能としては入力されたアルゴリズムの処理時間を評価し，設計の妥当性を保証した．処理時間を満たす範囲内であればコストが小さい方がよいと考え，構成要素数を減らすルールを与えた．実験システムを作成し，ラティス・フィルタ等の仕様を与え，いくつかの実験を行った．その結果，この手法がデータパス系の設計に関してはある程度有効であることが確認できた．

1. まえがき

回路の設計，製造技術が高まるにつれて，信号処理プロセッサ[1]に見られるような様々な個別的 LSI が製作され，実用に供せられるようになった[2]．さらに，VLSI のような大規模な回路の設計，製造に関する研究が盛んに行われており，そのような回路のユーザにとっても，一つのチップで高い処理能力が得られるために，大きな期待が寄せられている．

集積度が高まることによって，どのようにして最終的にユーザが必要とする機能を持たせるかが重要な問題の一つになる．そこで，VLSI 設計における今後の一つの方向として，ユーザが自分の欲する回路仕様を表現して回路を設計する，あるいはしてもらうというものが考えられる．自分が計算機上で実行しているソフトウェアを高速に処理したいとか，小型化したいという理由でハードウェア化したいという要求は高い．特に，信号処理のユーザにとっては，ユーザごとにアルゴリズムや，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．

ところが，通常の回路の設計自動化の研究では，回路の動作仕様が比較的明確に記述できるレジスタ・トランスファ・レベルの仕様記述を入力とするものが多い[3),4)]．しかしながら，ユーザにとって現在のハードウェア記述言語は，必ずしも，扱いやすいものとは言えない．ユーザが比較的簡単に明確化できる仕様は，自分の処理したいアルゴリズム，入出力条件，処理時間に対する要求等であろう．しかし，これだけの情報で回路を設計するには，難しい問題が多い．

そこで，我々は，高級言語により記述されたアルゴリズムによる動作記述と処理時間等の制約条件を入力とした回路の設計支援システムを開発することにした．本論文では，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アルゴリズムと処理時間等の制約条件を入力として，主にデータパス系の設計を行う方法を検討した．

類似した研究として，CMU の DAA システム[5]がある．その場合は，ISPS と呼ばれるインストラクション・レベルの仕様記述から回路を自動合成するもので，比較的汎用的な回路の設計を対象としていると考えられる．これに対して，我々は，通常ソフトウェアで実行されるようなアルゴリズムを入力としてユーザの要求する制約を満足する比較的小規模な専用回路の設計を対象としている．入力記述は比較的容易であるが，通常のマイクロプロセッサのような汎用的なものの設計は，現在のところ対象としていない．

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．

まず，前述したように，高級言語によるアルゴリズムと制約条件から成る仕様記述を入力する．実際に

† Automatic Data Path Synthesis Method Based on High Level Specification Description by KATSUHIKO SHIRAI, TOSHIYUKI TAKEZAWA and YASUO NAGAI (Department of Electrical Engineering, Waseda University).

†† 早稲田大学理工学部電気工学科

* 現在 (株)東芝から(財)新世代コンピュータ技術開発機構へ出向.

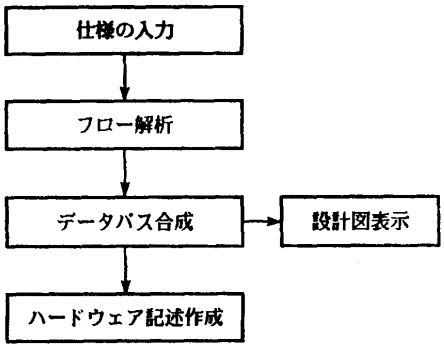

図 1 システムの処理フロー
Fig. 1 Block diagram of the system.

は，仕様は LISP のS式で記述される．

次に，そのアルゴリズムのフロー解析を行い，設計する上で重要な並列実行可能性等の特徴抽出を行う．仕様として入力された LISP プログラムに対して，通常のコンパイラと同様に，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フローグラフおよびシンボルテーブルを生成する．この部分は，構造化プログラミングの形式で記述された信号処理アルゴリズムで1チップで実現できる程度のものは十分扱えるものとなっている．

このようにして得られた情報をもとにして，回路の機能設計を行う．これには，知識工学の手法を適用している．知識ベース中には回路の構成要素に関する知識や回路の変更規則が蓄えられている．まず，フロー解析によって得られた中間表現に対して演算器や記憶要素をほぼ1対1に対応させる．このようにして合成された回路に対してその性能を評価し，入力仕様中の制約を満たす範囲内で回路の変更を繰り返す．性能評価としては，特に，入力されたアルゴリズムの大体の処理時間を予測する．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．回路の変更は，コストが小さくなるものとみなして，構成要素を減らす方向で行う．

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，その大体の性能を出力するのである．つまり，入力仕様中のアルゴリズムを入力仕様中の制約の範囲内で実現するためには，演算器や記憶要素がどの程度必要かという情報を出力すると言える．実際の出力形式は，現在，状態遷移形式の動作記述に回路の構成要素とその制御情報を付加したものとしている．このような出力形式とすれば，これまでに開発された多くの CAD システムやツールが利用できる可能性がある．

2.1 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式とした．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，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はストリーム形式シリアル入力で，8 bit 幅，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

```
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-statement | 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）
Fig. 2 Syntax of the specification description language.

```
(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・フィルタの仕様記述例
Fig. 3 Sample of the specification description.

いる．つまり実時間で処理するためには，80 μs 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，ある決まった回数だけ繰り返す DO 文である．また，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，すでにその概略がほぼ決定されているといえるが，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．

2.2 フロー解析

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および制御フローグラフが作成され，それ以外の宣言部（DCL-IO および DCL-RESTRICTION 部）から，シンボルテーブルが作成される．

制御フローグラフは，アルゴリズム中の DO ループや COND 節等の制御の流れを示すものである．DO ループや COND 節の条件部とそれに対応する基本ブロック名から構成される．

基本ブロックは，DO ループや COND 節を含まない一連の式の単位である．したがって，二重ループの時は，外側のループの中は細分化され，内側のループの前と後に標準的には一つずつ基本ブロックが作られ，内側のループを前後の基本ブロック名とともに制御フローグラフに登録し，その中が再帰的に処理される．

動作記述からフローグラフへの変換は，手続き的に行う．フローグラフの作成手順を説明する．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し，基本ブロック名が付けられる．基本ブロック名は制御フローグラフに登録される．データフローグラフも同時に作成され，基本ブロック名自身に，それに対応するデータフローグラフを検索できる情報（属性）が与えられる．

次に，データフローグラフ中の個々の文に対して，字句解析および演算子順位法に基づく構文解析を行う．そして実行すべき演算の順序に四つ組の中間表現形式を作成する．この四つ組は

(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．

最後に変数の解析を行う．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．

このようにして作成されたフローグラフを模式的に描いた例が**図 4** である．これは，図 3 の仕様記述例から作成されたものである．

なお，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．

2.3 機能回路合成

機能回路およ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

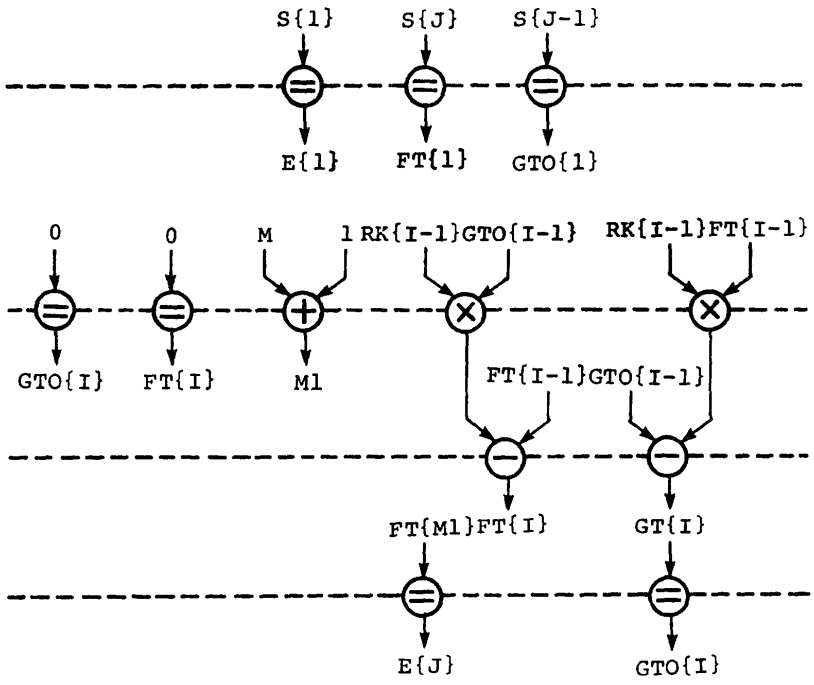

図 4 データフローの例
Fig. 4 Sample of the data flow.

れる．Adder の記述例を図 5 に示す．図 5 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．以下では，このようなものをクラスと呼び，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 1 ノードに対し 1 演算器を設け，各変数ごとに記憶要素を設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・システムの手法を採用している．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 6 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，別のルールセットを選んで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．

```
(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 5 Adder の記述例
Fig. 5 Knowledge representation for an adder.

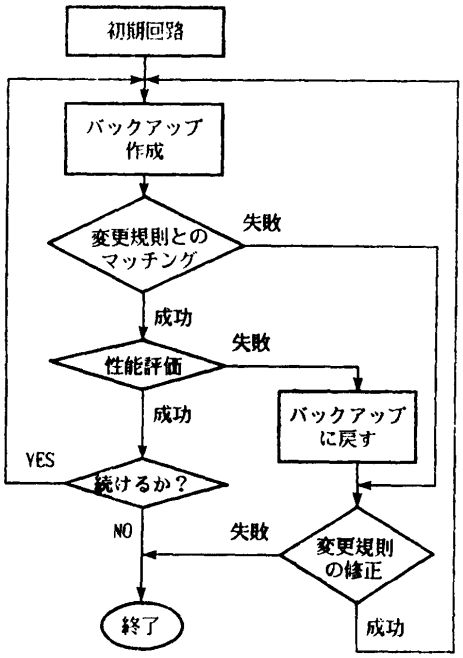

図 6 回路の変更手順
Fig. 6 Flow of the design process.

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，回路の変更は行わない．今回の実験においては，高速化を図るルールは用意されていない．

2.4 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．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や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，同じ機能の構成要素を高速なものから低速なものへ変換することが可能である．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．

アルゴリズムの処理時間算定の基礎データとなっているのは，一般の規格表を参考にして標準的な値と思われるものに基づいている．その表現法は，各クラス表現中の SPEED と COST というスロットに蓄えておく．なお，この値自体は，より実際的な実験の際には，容易に変更することが可能である．

このようにして，データフローグラフに対応づけて

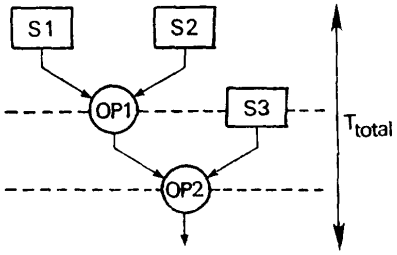

図 7 性能評価の例
Fig. 7 An example of performance estimation.

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 7 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．

$$T_{total} = T_i(OP\,2) + T_o(OP\,2)$$
$$T_i(OP\,2) = \mathrm{Max}\,[T_o(OP\,1) + T_i(OP\,1), T_s(S3)]$$
$$T_o(OP\,2) = T_{op}(OP\,2) + T_d(OP\,2)$$
$$T_o(OP\,1) = T_{op}(OP\,1) + T_d(OP\,1)$$
$$T_i(OP\,1) = \mathrm{Max}\,[T_s(S\,1),\ T_s(S\,2)].$$

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二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．

繰り返す回数の分か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，繰り返し文を並列展開した場合の性能評価は，回数倍せずに 1 回分の処理時間でそれとみなす．

つまり，性能評価としては，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って，厳密なシミュレーションを行っているわけではない．

制約値の近傍にある場合の性能評価については，おおまかな処理時間を予測しているために，考え得る多くの回路を出力し，最終的には利用者の判断を仰ぐこ

とになる.

2.5 ハードウェア記述の出力

フロー解析や機能設計部により得られる情報は，主に，データフローグラフ，制御フローグラフ，回路の構成要素，および，構成要素とフローグラフとの対応である．一般には，入力されるアルゴリズムの規模や複雑さには幅があるため，全体が一つの回路にできるものもあれば，複数に分割すべきものもあるので，入力仕様中のアルゴリズムのどの範囲を扱うか指定する必要がある．本論文では，そのような分割問題は考えずに，例題として信号処理アルゴリズムを対象としアルゴリズム全体を一つの回路にすることを検討した．

作成する状態遷移形式の動作記述の基本形は次のような四つ組である．

(State Condition Next-State Operation)

State はユニークな状態名を示す．Condition は，条件分岐により次の状態が異なる場合の条件である．Next-State は次の状態名である．Operation は，その状態で実行すべき操作である．

これの作成手順を説明する．制御フローグラフをもとに，まず，COND 節の条件判断や DO ループのインデクスの処理の状態を生成する．COND 節の各条件に対応する動作や DO ループ内の動作は，この段階では，基本ブロック名のままとなっている．次に，基本ブロックに対応するデータフローを状態遷移形式にしたもので，この基本ブロック名を置き換える．

この状態遷移形式の作成は，手続き的に行うため，別の状態とまとめられるものが生じることがある．また，フロー解析を行う際に一時変数が生成されるため，そのまま状態遷移形式に変換すると，実際には不要な一時変数が残ることがある．そこで，このようにして作成された状態遷移形式に対して，若干の最適化を行う．まず，データフロー中に出現する一時変数のうち冗長なものを消去し，冗長な状態を一つにまとめる．次に，対応する演算器が重複しない，複数の状態にわたる同時に実行可能な操作を一つにまとめる．

回路の記憶要素は，アルゴリズム中の変数名ごとに合成したものである．演算器系についてはルールベースの手法により回路の変更を行うが，記憶要素系については，このような変更を行っていない．さらに，演算器系を変更することによって，記憶要素系の調整が必要となろう．そこで，この段階である程度の調整を行うために，変数のライフタイム解析を行った．

上で述べたようにして得られた状態遷移形式のもとに各状態ごとに次のような形式のテーブルを作成する．

(STATE (VAR LIVE)...)

STATE が状態名，VAR が生きている変数名，LIVE が S, E, L の記号である．Sは代入された時，Eは参照された時，Lはその間で生きていることを示す．

これの作成手順を述べる．まず，状態遷移表現中の Operation 部に対して代入と参照のすべての候補を挙げる．次に Condition 部に対して参照のすべての候補を挙げる．最後に，参照から代入へ後向きに伝播させて，生きている変数をライフタイム・テーブルに追加する．

伝播の順序は状態遷移の順ではなく逆順とした．代入，参照の様々なパタンに対して逆順の方が伝播の方法が一様ですむためである．伝播の方法としては，

○現状態で参照される変数と生きている変数は，その前の状態では生きている．

○現状態で代入される変数と死んでいる変数は，その前の状態では死んでいる．

というものである．状態遷移の基本パタンは逐次的，ループ，条件分岐の3通りであるが，本手法においては，現状態に遷移してくるすべての前状態に伝播させるので，いずれのパタンにも適応可能である．

このようにして作成された変数のライフタイム・テーブルをもとにレジスタ数を決定する．最も効率を上げるためには，この表の各状態の中で生きている変数の数の最大値をとればよい．ただし，変数への代入と参照が同時に起こる場合があることに注意する必要がある．この場合，それは一つと数える．現在は，このようにして，必要なレジスタ数の上限を決定することのみ行っている．もちろん，この数より少なくてもハードウェアとしては成立する．

このようにして，むしろインストラクション・レベルに近いハードウェア記述を出力する．これは，この段階で作成するとすればどの程度の情報を出力できるか検討するために行ったものである．

3. 知識処理技術について

ここでは，特に機能設計を行う上で採用した知識処理技術について述べる．この段階の設計は，考え得る非常に数の多い回路構成から適切な回路構成を決定することである．この問題は従来あまり研究されていなかったものなので，必ずしも，有効なアルゴリズムは

明らかでない．さらに，今回選んだ対象ではユーザの要求する仕様が多種多様であるため，一様に多くの場合に適応可能なアルゴリズム的解法を考えるよりは，知識工学的手法を適用することが一つの有効な手段である．ただし，アルゴリズムによる解法は最適解を保証するが，ヒューリスティックな問題解決手法には最適性の保証はない．しかし，適当なアルゴリズムが分かっていなくとも，知識工学の手法では，比較的簡単にルールを変更して実験することが可能である．そのような実験を経て，うまい解法が見つかれば，それは一つのアルゴリズムと言えるかもしれない．

ここで，本システムが対象としている設計問題，特に，機能設計を考えてみる．例えば，機能設計の中の一つに演算器系決定という段階が存在すると考えられるが，それについて次のようないろいろなレベルの規則群が考えられる．

○並列動作しない演算器を併合する規則

○並列動作するものでも演算器を併合する規則

○演算器を同一機能で高速あるいは低速のものに置換する規則

そこで，これらをルールセットに分けることにすると図8のような構成となる．合成された回路のデータはブラックボードに置き，ルールとのマッチングや内容の変更ができるようにする．そして，推論機構でルールセットの選択等の管理も行う．このようなスケジューラとしての機能は一種の設計計画とも考えられる．

この枠組みの利点としては，次のようなものが挙げられる．

○全体としての規則数が増大しても，個々のルールセットではそれほど爆発的に規則数が増加しない．

○ルールセットに分割しその管理を行うというアプローチを採用することにより，システム全体としての動作の制御が見通し良く行える可能性がある．

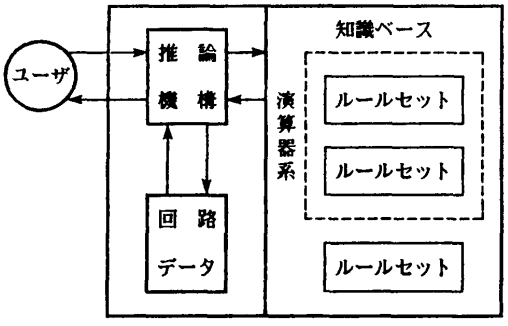

図8　知識ベース・システムとしての構成
Fig. 8　Construction of the knowledge-based system.

```
<rule> ::= (rule-name <antecedent> => <consequent>) ;
<antecedent> ::= (AND {<condition>}) |
                 (OR {<condition>}) ;
<condition> ::= <pattern> | (NOT <pattern>) ;
<pattern> ::= (<object> {^<attribute> <value>}) |
              (<predicate> <pattern>) ;
<consequent> ::= {<action>} | {<procedure>} ;
<action> ::= (MAKE <object> {^<attribute> <value>}) |
             (MODIFY <object> {^<attribute> <value>}) |
             (DELETE <object> {^<attribute>}) ;
<procedure> ::= (function-name {<argument>}) ;
<element> ::= <vector-element> | <av-element> ;
<vector-element> ::= {<value>} ;
<av-element> ::= (<object> {^<attribute> <value>}) ;
```

図9　プロダクション・ルール記述文法（要旨）
Fig. 9　Syntax of the production rules.

現在試作した部分は，まず演算器系が重要であると考え，図8において破線で囲んだ部分である．ルールセットの選択についてはまだ複雑な選択の制御は行っておらず，数通りの順序でそれを選択，実行するようにした．ブラックボード中の回路のデータは，フレーム形式で記述される．

次に，本システムのプロダクション・ルールの記述文法を図9に示す．連想三つ組が基本となっており，これとブラックボードに記述されたフレーム形式の回路データとの間でマッチングがとられる．前向き推論の際には，ルールの右辺にLISPで記述した関数を埋め込むことが可能である．なお，このインタプリタ自身は前向き推論，後向き推論いずれも可能なものを作成したが，設計システムとしては前向き推論を採用している．実際のルール記述例を図10に示す．#OPERATOR1のようにアトムの先頭に#マークの付いているものは変数を表す．この例の右辺に現れるOPMGDは関数名である．

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，どれか適当な二つを選んで一つの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

```
(RULE4 (AND (#OPERATOR1 ^FUNCTION ADD)
            (#OPERATOR2 ^FUNCTION SUB))
       =>
       (OPMGD OPERATOR1 OPERATOR2))
```

図10　プロダクション・ルール記述例
Fig. 10　An example of production rules.

ルを全部で約30個用意した．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．

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・フィルタ[6]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フレーム当たりのサンプリング・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は 80 μs 以内で処理が終了する必要がある．

表1 ラティス・フィルタの設計実験例
Table 1 Results of an experiment for the filter.

回路	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 ALU 1個 RAM, ラッチ	33 μs
E	ALU 1個 RAM, ラッチ	6 ms

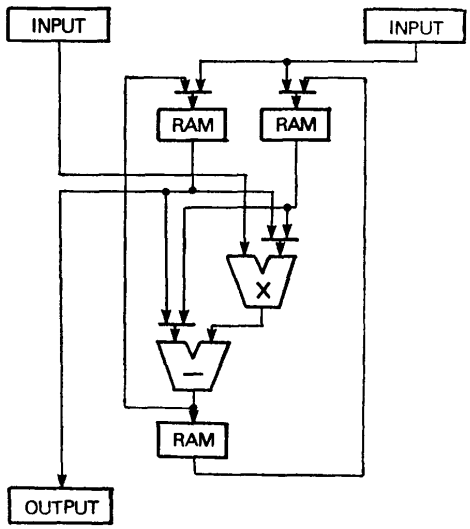

図11 フィルタの回路例（表1中の回路C主要部）
Fig. 11 A circuit of the filter (shown as circuit C in Table 1).

表2 ラティス・フィルタの設計過程例
Table 2 Design process of an experiment for the filter.

ステップ数	変更規則	回路変更	性能評価	回路構成
1	規則群1	成功	成功	回路A
2		成功	成功	回路B
3		失敗		
4		失敗		
5		成功	成功	回路C
6		失敗		
7		失敗		
8		失敗		
9		失敗		
10		失敗		
11	規則群2	失敗		
12		失敗		
13		失敗		
14		成功	成功	回路D
15		失敗		
16		成功	失敗	回路E
17		成功	失敗	回路E
18		失敗		

本システムの一つの設計結果をまとめたものを**表1**に示す．この場合は，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 ms と予想されるため，失敗となったものである．合成されたデータパスの例を**図11**に示す．これは，表1中の回路Cである．この場合の設計プロセスを**表2**に示す．ここで使用したルールは10数個ずつルールを含んだルールセット2個である．このほかにもいくつかルールを変えて実験を行った．その結果，この例においては途中経過で異なる回路が得られたが，最終的な回路構成は同じものとなった．

4.2 DP マッチング

別の例として DP マッチング[7]を考えてみた．これは，近年音声認識の分野で利用されているもので，動的計画法により補正をしながら音声の入力パタンと標準パタンの距離計算を行い，音声を認識するものである．DP のアルゴリズムは，**図12**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 ms ごとに，8 bit 精度で16個の

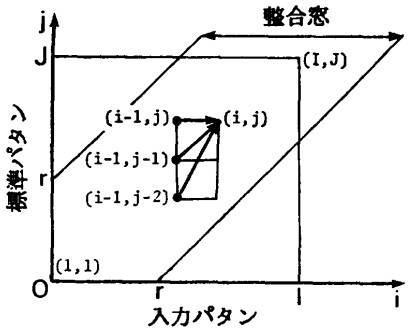

図 12 DP パス
Fig. 12 DP path.

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 ms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，二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．今回の実験では，比較演算を行う 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 ns と仮定して，性能評価に用いた．そして，変更する過程で ALU に変換するルールを与えた．

絶対値の計算は，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．

一つの設計結果を表3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かる．回路構成例を図13に示す．

5. 評　　価

本システムの一つの評価として UTILISP の TIME 関数により CPU 時間を測定した結果，フィルタの例では約7秒，DP マッチングの例では約23秒であった．これにはガベッジ・コレクションの時間を含む．使用した計算機は，東京大学大型計算機センター HITAC M-280 H である．早稲田大学リモート・データ・ステーションを介して TSS 環境で利用した．

本システムの設計の質の評価は，必ずしも現段階で十分な評価はできないが，例えば，今回の実験結果を，既に設計され使用されている信号処理プロセッサの構成と比較すると，妥当な構成となっていると言えよう．逆に，本システムは，製作された LSI の中で入力されたアルゴリズムに対して適切なものを選択することに使用できる可能性がある．

また，知識処理技術については，フィルタの例においていくつか条件を変えて実験を行った結果，最終的に得られた回路は同じものであったが，これは必ずし

表 3 DP マッチングの設計実験例
Table 3 Results of an experiment for DP matching.

回　路	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	2.6 ms
B	加算器12個 MIN 2個，MAX 1個 RAM，ラッチ	3.2 ms
C	加算器11個 MIN 2個，MAX 1個 RAM，ラッチ	3.4 ms
D	加算器 9 個 MIN 2個，MAX 1個 RAM，ラッチ	3.5 ms
E	加算器 6 個 ALU 1個 RAM，ラッチ	6 ms
F	ALU 2個 RAM，ラッチ	11 ms
G	ALU 1個 RAM，ラッチ	11 ms

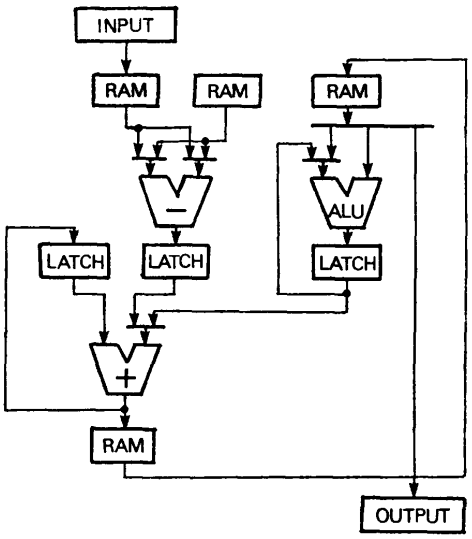

図 13 DP マッチングの回路例(表3中の回路E主要部)
Fig. 13 A circuit of the DP matching (shown as circuit E in Table 3).

も一般に言えることではない．ただし，およその性能評価がある程度有効に作用しているものと考えられる．

ルールの数と内容は，今回の例を扱うためならば十分と考えられるが，より複雑な問題に対する場合むだがなく，しかも十分賢いルールが用意されているかは検討を要する．また，例えば高速化を図るためのパイプライン構成法といったもっと様々な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．

6. むすび

高級言語により記述されたアルゴリズムと，処理時間に対する要求を入力とした回路のデータパス系設計支援システムの概要といくつかの設計例，および，そこでの知識処理技術について報告した．フィルタ等の実験によりある程度の有効性が確認できた．

システムを改良する方法としては，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ものが考えられる．また，途中の回路図を適切に表示して，対話的に設計を行うことも必要であろう．

今後の課題としては，さらに実用的な質と量を持つ設計に関するデータを取り扱う必要があろう．さらに，出力記述の記述能力を高めて，制御回路設計を考えることが必要である．また，高速化を図るルールを付加した時の全体のルールの管理を適切に行う方法を検討する必要もあろう．

参考文献

1) 内田俊一，森　秀樹：信号処理プロセッサ，信学誌，Vol. 82, No. 11, pp. 1281-1288 (1979).
2) 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).
3) 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).
4) 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).
5) Thomas, D. E. et al.: Automatic Data Path Synthesis, *IEEE Comput.*, Vol. 16, No. 12, pp. 59-70 (1983).
6) 斎藤収三，中田和男：音声情報処理の基礎，p. 268, オーム社，東京 (1981).
7) 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).
8) Shirai, K., Nagai, Y. and Takezawa, T.: Functional Level Design System for Digital Signal-Processors, *Proc. of IFIP Int. Conf. on VLSI 85*, pp. 203-212 (1985).
9) 竹沢寿幸，白井克彦：アーキテクチャ設計支援エキスパート・システムの構成と設計環境，情報処理学会「VLSI CAD への知識工学の応用」シンポジウム，pp. 1-10 (1986).

(昭和61年3月31日受付)
(昭和61年10月8日採録)

白井　克彦（正会員）

昭和14年生．昭和38年早稲田大学理工学部電気工学科卒業．昭和43年同大学大学院博士課程修了．昭和40年より同大学勤務．現在，同大学電気工学科教授．工学博士．音声情報処理，自然言語処理，CAI などマン・マシン・システムの諸問題の研究に従事．電子通信学会，日本音響学会，IEEE 等会員．

竹沢　寿幸（正会員）

昭和36年生．昭和59年早稲田大学理工学部電気工学科卒業．昭和61年同大学大学院博士前期課程修了．現在，同大学大学院博士後期課程在学中．回路設計支援システムの研究に従事．電子通信学会，人工知能学会各会員．

永井　保夫（正会員）

昭和36年生．昭和58年早稲田大学理工学部電気工学科卒業．昭和60年同大学大学院理工学研究科電気工学専攻修士課程修了．在学中は知的 CAI システム，知的 DA システムの研究を行う．同年(株)東芝に入社，情報通信システム技術研究所に勤務．同年11月より(財)新世代コンピュータ技術開発機構へ出向中，同機構研究所第5研究室に勤務．現在，エキスパートシステム，特に構築ツールの研究に従事．電子通信学会会員．