解　説

VLSI 設計エキスパートシステムにおける知識獲得†

渡辺正信††　岩本雅彦††

1. はじめに

近年，VLSI 設計者の経験的知識をフルに活用することで，アルゴリズムを主体とする従来手法の限界を打ち破ろうとする VLSI 設計エキスパートシステムが，非常に注目を集めている．そこでは，熟練設計者のケースバイケース的設計ノウハウをいかに多く知識ベースに保持し適宜活用できるかで，その機能や処理速度が決まる．したがって，VLSI 設計エキスパートシステムの成否は，設計者の設計ノウハウをいかに獲得するかにかかっているといっても過言ではない．

本稿の目的は，VLSI 設計エキスパートシステムを構築する上での知識獲得の問題を整理し，その現状と自動化に向けた研究事例を紹介し，今後の課題について解説することにある．以下，第2章において，VLSI 設計知識を獲得する上での問題点を整理し，その問題を解決するために必要な要素技術を明らかにする．第3章で，設計知識を三つのレベルに分類し，各知識レベルでの知識獲得手法について概観する．第4章では，知識獲得の事例を紹介する．そこでは，設計者へのインタビュを基にした獲得手法から，設計者の試行をモニタし自動学習する手法の代表例を紹介する．最後に，今後の課題を整理する．

2. VLSI 設計知識の獲得問題

VLSI 設計エキスパートシステムは，従来手法では達成困難とみなされていた種々の課題（設計時間の短縮，設計品質の向上，階層設計の下で分断された設計ノウハウの集約など）を実現するものとして非常に注目され，活発に開発が進められている[1)~8)]．しかしながら，VLSI 設計エキスパートシステムを実現するためには，熟練設計者の経験的知識をどのようにして獲得するかという大きな問題がある．特に，VLSI 設計分野における知識獲得では，次の問題に対処しなければならない[9)]．

（1） 大規模な知識ベースが要求される．

（2） ひとりではなく複数の設計者のノウハウが必要である．特に，階層設計の各レベルに分割された設計者のノウハウを集約する必要がある．

（3） VLSI 技術の急速な変化に追随して，知識ベースの内容を変更できなければならない．通常，VLSI 技術は，2年ごとに刷新されるといわれる．

（4） 設計者の本来の仕事を中断させることなく，設計ノウハウを知識ベースに組み込めることが望まれる．

以上の問題を解決するためには，次の技術が重要となる．

〔A〕 大量の知識を収集し，知識ベース化する技術．たとえば，地理的に分散した複数の設計者から時間的に変化していく知識を収集していく技術が必要となる．

〔B〕 段階的に追加，更新されていく知識ベースの整合性/論理的一貫性を管理維持する技術．

上記〔A〕の技術として，最近の知識獲得/学習の研究でいくつか注目すべきものがある．たとえば，問題や対象知識ごとにモデルを仮定しその下で知識を収集していく手法[10)~12)]や，設計者の設計プロセスをモニタし，システムに欠けている知識を検出し一般化した形に変換し，知識ベース化する手法[13),14)]などが注目される．なお，エキスパートシステムにおける知識獲得全般に関しては，文献 15) を参照されたい．

一方，〔B〕の技術としては，述語論理をベースにした TMS (Truth Maintenance System)[16)]，ATMS (Assumption-based TMS)[17)]などが注目される．これらは，知識ベースの部分的な変更にともない，その影

† Knowledge Acquisition in VLSI Design Expert Systems by Masanobu WATANABE and Masahiko IWAMOTO (C&C Systems Research Laboratories, NEC Corporation).

†† 日本電気(株) C&C システム研究所

響を効果的に伝播し，必要に応じてほかの部分を変更したり，矛盾を検出し解消する技術である．

なお，以下本稿では，VLSI 設計エキスパートシステムを実現する上で，設計者の設計ノウハウを収集し知識ベース化することが急務の問題であることから，〔A〕の技術に的を絞り，現状の技術と今後の課題を詳述する．

3. 設計知識のレベルと獲得方式

VLSI 設計エキスパートシステムは，アーキテクチャ/方式設計[1]，論理設計[2)~5)]，レイアウト設計[6)~8)]に分けて開発が進められているが，ここでは，設計知識の特性に従って図-1 に示す三つの知識レベルに分けてその獲得方式を概観する．

まず，〔設計オブジェクトレベル〕の知識には，モジュールの動作仕様，要求仕様に関する制約条件自身，制約条件を違反していることの検出方法，違反を解消する方法なども含まれる．このレベルの知識の獲得事例には，LSI やバスの動作を記述した自然言語と図式からなるテキストを解析して，有限オートマトンを基にした形式的な仕様記述を獲得する方式[12]や，モジュールやデータストリームの動作条件，入出力写像関係を形式的に記号表記して，下流のモジュールの動作条件を上流へ制約伝播させ上流のモジュールの動作条件を記号的に導出/獲得することができる方式[18]や，タイミング設計において，不完全な動作仕様と一部の要求仕様などから，パラメータ間の制約条件に基づき，論理的に一貫性をもつ完全な動作仕様を獲得する方式[14]などがある．

次の〔設計基本ステップレベル〕では，モジュール詳細化ルール，最適化ルールの獲得が中心となる．このレベルの獲得事例としては，設計者から直接一般化されたルールを獲得する方式，論理最適変換における変換前と変換後の部分回路図から最適化ルールを獲得する方式[2]，実際の設計過程の途中で設計者が，明示的に部分回路を変更しその情報を提供することでシステムがその部分回路に関する合成ルールを獲得する方式[20]，実際の設計過程において設計者の試行をモニタし，システムに欠けるルールを抽出し自動的に一般化する方式[13),14)]などがある．なお，このレベルの獲得方式は，次章で詳述する．

知識レベル	内　容	獲得事例
設計オブジェクトレベル	モジュール，データストリームなどの動作仕様・要求仕様・制約条件に関する知識．	西田モデル CRITTER VILLA
設計基本ステップレベル	モジュール詳細化ルール，最適化ルール，設計における基本合成ルールなどの知識．このレベルの知識は互いに独立性が強く，それ自身が設計過程の基本ステップとなる．	SOCRATES CONSTELLATION LEAP VILLA
制御レベル	設計過程を大局的にみることで，代替案の中で最適なものを選出する制御知識・ヒューリスティックルールなど．	

図-1 設計知識のレベルと獲得事例

最後の〔制御レベル〕における知識とは，設計プロセスを大局的にみることによって獲得することができる制御知識である．たとえば，チップ面積を最小化する基準で論理合成を最適化するための知識などがこのレベルのものである．なお，設計者が実際に行っている試行錯誤（バックトラック）制御を忠実に計算機上で実現する方法は，いまだ確立されていない．そこで，多くのエキスパートシステムでは，大局的なバックトラックをしないモデルを仮定して全体制御が実現されている．そこでの知識獲得では，設計基本ステップレベルのルールが複数適用できる状態で最適なルールを選出するヒューリスティックスルールの抽出がポイントとなる．

4. 知識獲得の事例

ここでは，VLSI 設計エキスパートシステムにおける知識獲得方式を，次の4つの代表事例 (DAA, SOCRATES, COSTELLATION, LEAP) を通して紹介する．つまり，DAA では，知識工学者（エキスパートシステムの開発者）が VLSI 設計者に直接インタビュして設計ノウハウをルール化する．SOCRATES は，知識工学者または設計者が，論理最適化ルールを簡便に効率良く入力することができるグラフィックインタフェースを提供する．そこでは，最適化できる部分回路構成図と最適変換後の部分回路構成図を入力すると実行可能形式の最適化ルールが自動的に検証され知識ベースに組み込まれる．CONSTELLATION では，設計者が通常の設計作業時にある部分回路図を典型的な回路合成，回路解析の具体例として明示すると，これを先例としてシステムがルールを作成し，以降の設計に活用する．ただし，ここでは，ルールの一

般化は行われない．LEAP では，設計者の試行をモニタし，システムに不足している有効な合成ルールを自動抽出し，さらに，そのルールを検証後，自動的に一般化することによってルールの汎用性を高める．

4.1 DAA[1)]

DAA (The VLSI Design Automation Assistant) は，VLSI システムのアルゴリズム的記述 (ISPS 言語による記述) からテクノロジ独立のレジスタ，オペレータ，データパス，制御信号などからなる構造記述 (SCS 言語による記述) を生成するアーキテクチャ/方式設計エキスパートシステムである．ここでは，DAA を構築するときにとられた知識獲得方式を紹介する．この方式は，専門家に対するインタビュ方式として最も代表的なものであり，次の二つのフェーズからなる．

(1) ケーススタディインタビュ

プロトタイプシステムを作成することを目的として4人の方式設計者（初級1名，中級2名，上級1名）にインタビュが行われた．このフェーズのインタビュでは，教科書的知識から設計方法論に至る幅広い設計知識を得ることはできるものの，具体的かつ緻密な設計ノウハウを十分に獲得することは困難である．DAA でのインタビュの要約を図-2 に示す．図中の yes は，それを実行していると述べたこと，no はそれをしないと述べたこと，ブランクはそれについて述べなかったこと，数値は実行順序や年数を示す．4人に対する計約4時間のインタビュをテープに記録し，これを分析することで，DAA のプロトタイプシステム (OPS 5 を使って約 70 個のルールを保持するもの) が作成された．

	設計者1	設計者2	設計者3	設計者4
Background				
Level	novice	moderate	expert	moderate
TTL & ECL	yes	yes	yes	yes
NMOS	no	yes	yes	yes
CMOS & PMOS	no	no	yes	no
Industrial years	3	4	12	4
Total years	7	9	12	7
Allocation				
Clock Phases	1		2	
Operators	2	3		
Registers	3	2	3	
Data Paths	4	1	1	1
Control Logic	5	4	4	2
Iterating				
Constraint Violations	yes	yes	yes	yes
Global Improvements	yes			yes
Technology Increase	yes	yes	yes	yes
Functionality Decrease		yes	yes	yes
Constraints				
Speed	yes	yes	yes	yes
Area		yes	yes	yes
Power			yes	yes
Schedule			yes	
Cost	yes	yes	yes	
Drive	yes			yes
Width	yes	yes		

図-2 設計者へのケーススタディインタビュの要約

(2) 知識獲得インタビュ

プロトタイプシステムが出力した具体的設計例を，熟練（前述の上級クラス）設計者に批評してもらい，システムに不足している知識を補充していくことを目的とするインタビュである．DAA では，MOS テクノロジのマイクロプロセッサ (MCS 6502) を例にとり，この知識獲得インタビュを行った．このインタビュ結果を要約したのが図-3 である．そこでは，DAA が段階的に設計ルールを改良・増加することにより設計の質を向上していったことを示す．設計1は，約70個のルールをもつ最初のプロトタイプによる設計例である．設計1から設計2への改良は，異なるサイズやタイプの算術オペレータ (Cmp, Minus, Plus, Shifts

	設 計 1	設 計 2	設 計 3
And	20	20	20
Cmp	177	1	1
Minus	64	0	0
Or	9	9	9
Not	21	21	21
Plus	540	0	0
Shifts	35	1	1
Xor	9	9	9
Alu	0	35	35
Dreg	450	281	210
Treg	1,227	0	62
Mux In	2,122	2,657	473
Mux Out	293	377	84
Bus In	0	0	769
Bus Out	0	0	210

図-3 MCS 6502 に対する三つの設計例

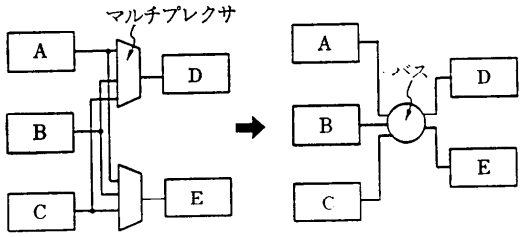
など）を ALU を使って集約化すべきという設計ノウハウを追加することで実現された．さらに，設計2から設計3への改良は，マルチプレクサ（Mux In, Mux Out）をバス（Bus In, Bus Out）に変換するルールを追加することで実現された．このとき追加されたルールを図-4 に示す．設計3の時点でのルール数が約 130 であった．以上のインタビュを熟練設計者に数百時間にわたって繰り返すことで DAA システムは約 300 個のルールを保持するに至った．その後，IBM 370 プロセッサに対する試行が行われ IBM 設計者によりテクノロジ独立レベルの設計として妥当な評価を受けている．

なお，DAA の作成者は，論文を次の興味深い言葉でしめくくっている．『エキスパートシステムアプローチによって，VLSI 設計者の設計ノウハウを収集し検証する時間が短縮された…』つまり，DAA が，設計ノウハウを獲得するための強力なツールの役（CAD システム開発者と設計ノウハウを保持する設計者の仲介役）を果たした．

一方，DAA では，この知識獲得を上記のインタビュにより全面的に人手で行った．以下のシステムでは，この人手による知識獲得作業を半自動化，自動化しようとするものである．

4.2 SOCRATES[2]

SOCRATES は，組み合わせ回路を特定のテクノロジ向けに最適化する（与えられた遅延の範囲でゲートサイズを最小化する）ルールベースエキスパートシステムである．その全体構成は，図-5 に示され，図中の Rule entry モジュールが知識獲得サブシステムに当たる．なお，獲得しようとするルールは，図中の Extractor と Comparator で，その正当性が検証される．ここで述べる最適化ルールとは，図-6 で示された点線で囲った部分の回路変換（たとえば AND と NOR



（a）

IF
（1） 現在のアクティブコンテクストがバス配置であり，
（2） マルチプレクサであるモジュールが存在し，
（3） マルチプレクサであるほかのモジュールが存在し，
（4） バスでないモジュールから最初のマルチプレクサへのリンクが存在して，
（5） そのモジュールから2番目のマルチプレクサへのリンクが存在して，
（6） 他のバスでないモジュールから最初のマルチプレクサへのリンクが存在して，
（7） そのモジュールから2番目のマルチプレクサへのリンクが存在する．
THEN
これらの接続をバスに置き換えよ．
（b）

図-4 マルチプレクサをバスに置き換えるルール

をその複合ゲートに変換すること）を実行するものである．この最適化問題のむずかしさは，同じルールが微妙に異なる複数箇所で適用可能な場合どちらに適用するかでその結果が全然異なってくるという点にある．たとえば，図-6 の左側と右側では，NOR の二つの入力信号のうちどちらを対象と考えるかでその結果（サイズ）が異なってくる例を示している．

SOCRATES では，最適化用の基本的変換ルールと全体の制御を行うメタルール機構を保持する．ここで述べる Rule entry は，この前者のルールの獲得を半自動化するものであり，その獲得手順を以下に述べる．

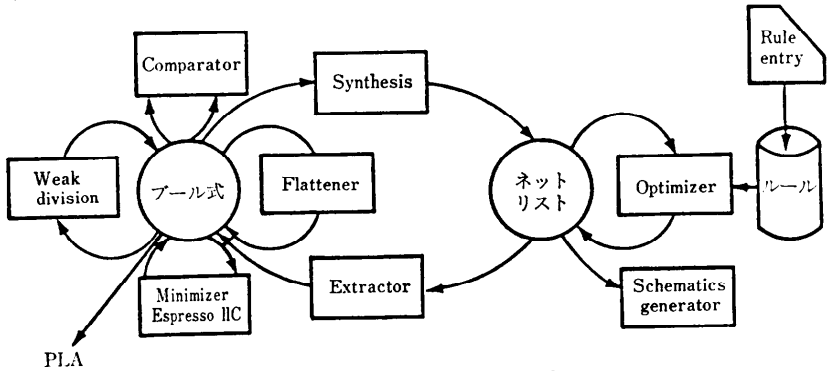


図-5 SOCRATES システム構成

〔SOCRATESでのルール獲得方式〕

まず，入力として，ネットリスト（ゲートとその接続による論理回路の構造記述）で記述された最適化の対象となる回路と変換後の回路のペアをとる．獲得結果は，条件部に図-7(b)で示される回路パターン，帰結部に条件部が満足したときの変換後回路構造をもつ最適化ルールである．図-7(a)の回路は，図-7(b)のパターンにマッチする具体的回路図例である．なお，SOCRATESでは，現在，図-7(a)のイメージをグラフィックインタフェースから入力すると自動的に図-7(b)のネットリストが生成される．以上の入力情報から次の手順でルールが作成される．

（1） 検証：入力された二つのネットリスト（変換前と変換後）が論理的に等しいことを，おのおのをブール式に変換して比較することで検証する．

（2） パターンとアクションの生成：最適化の対象とマッチするようなパターン（図-7(b)参照），及び置き換え，接続，削除などのアクションを行うコマンドを生成する．また，面積をどれだけ削減できるかなどに関する情報が計算されて専用のテーブルに蓄えられる．

（3） 分類：ルールがゲートサイズを削減するものか，時間遅延を減少させるものかなどの観点から，分類及び順序付けなどを行う．この処理は，ルールベース全体の再編成を必要とする（なお，この分類は，VAX 11/780で約3秒のCPU時間を要する）．

（4） 知識ベースへの挿入：以上の獲得方式の効果として，従来完全に人手で行った場合には約45〜90分かかっていたルールの作成と検証時間が，約3分に短縮された．特に，最適化法は，NMOS, CMOS, バイポーラといったテクノロジに依存しているため，新しいテクノロジが使われるごとに動的に変更される．このテクノロジの変更に追随するためにも最適化ルールの獲得を効率化することは非常に重要であり，SOCRATESのアプローチは実用的観点から示唆に富むものである．

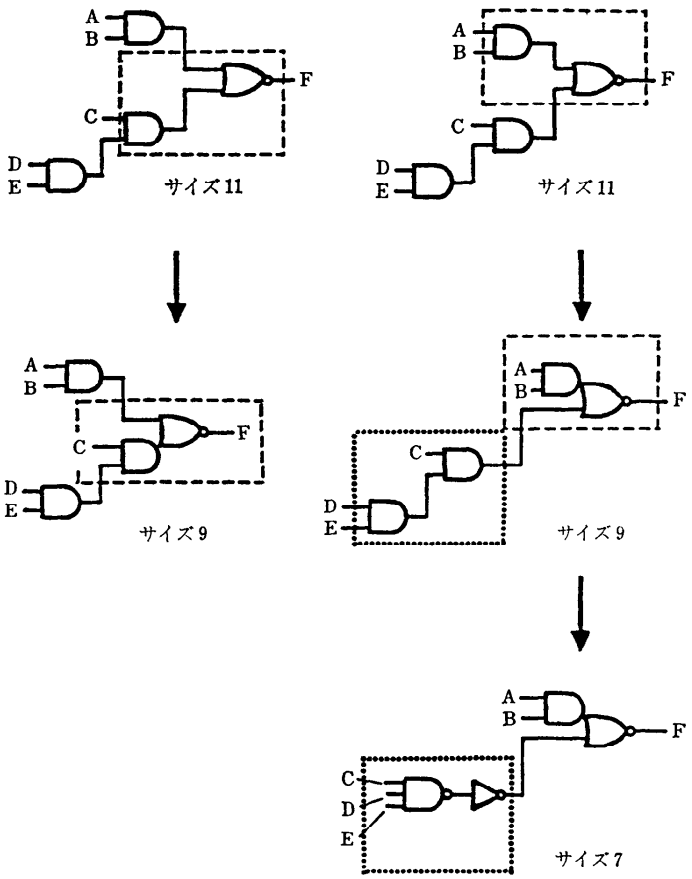


図-6 組み合わせ回路最適化例

一方，対象が組み合わせ回路に限定されているためすべての回路をこれで論理合成できないこと，設計者との対話的な最適化が行われないためルールの獲得方式が最適化処理と完全に分離されていることなどは，今後の課題となろう．また，回路最適化の大局的制御を行うメタルールの獲得を半自動化/自動化することも今後の課題である．

4.3 CONSTELLATION[19),20)]

CONSTELLATIONは，レイアウト前のネットリスト記述による構造レベルの設計の解析（より抽象度の高いモジュールへの変換）や最適化をグラフィックインタフェースを通して対話的に支援するLispベースCADツールである[19)]．そこでは，設計者が，グラフィックエディタで，ある部分構造（たとえば，図-8で黒くぬりつぶされた部分）を指示すると，CONSTELLATIONが，その指示された部分に適用可能な

解析ルール (design precedent と呼ばれるルール) を自動的に探索する．生成されるルールはマッチした部分をより高いレベルのモジュールに置き換えるルールであるが，設計者はそのルールのアクション部を変更することで種種の使い方(回路変更，最適化など)ができる．その後，必要であれば，最初に指示された部分構造の近傍で同様の部分構造パターンをもつ部分を自動的にアクション部に従って変更する．つまり，CONSTELLATION は，その解析ルールを適用できる部分を探索していく．以上の手順を実現する処理フローを図-9 に示した．ここで，STAR システムとは，同一設計オブジェクトに対するグラフィック表現とテキスト表現（Y言語という構造記述言語）の両方の表現を一貫して支援するものである．つまり，グラフィック表示された部分回路から，それに対応するテキスト記述を生成したり，この逆の変換を自動的に行う．

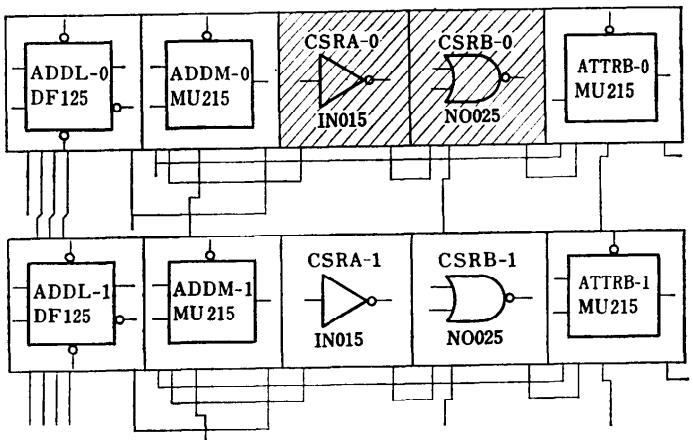
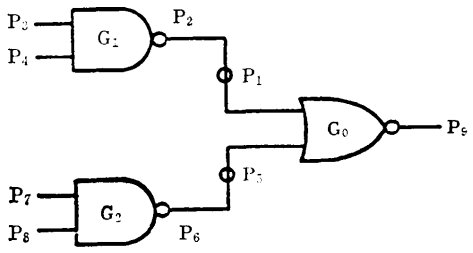


図-8 グラフィックウィンドウでの部分回路の指示例



(a)

```
1  node_class   G0  nr2
2  pinof        G0  P1
3  pinclass     P1  nr2_inputs
4  net_size     P1  1
5  con_pos      P1  G1
6  assign_con   P2  G1
7  pin_class    P2  nd2_output
8  net_size     P2  1
9  node_class   G1  nd2
10 pinof        G1  P3
11 pinclass     P3  nd2_inputs
12 pinof        G1  P4
13 pinclass     P4  nd2_inputs
14 pinof        G0  P5
15 pinclass     P5  nd2_inputs
16 net_size     P5  1
17 con_pos      P5  G2
18 assign_con   P6  G2
19 pin_class    P6  nd2_output
20 net_size     P6  1
21 node_class   G2  nd2
22 pinof        G2  P7
23 pinclass     P7  nd2_inputs
24 pinof        G2  P8
25 pinclass     P8  nd2_inputs
26 pinof        G0  P0
27 pinclass     P0  nr2_output
```

(b)

図-7 回路例とその回路パターン記述例 (文献2) より)

さて，CONSTELLATION が，どのようにしてルール (design precedent) を獲得するかを紹介しよう．この獲得手法において，前述の DAA，SOCRATES と異なる点は，設計者自身が，通常の設計作業中にルールを作成すること，つまり，システムが獲得することにある．

〔CONSTELLATION でのルール獲得方式〕

まず，グラフィックウィンドウ上に表示された回路上で設計者が部分回路(たとえば，図-8 上の二つのセル) をマウスを使って選択する．その後，Precedent 用ポップアップメニューを使って Make-precedent を選択すると以下のことが図-10 の処理フローに従って自動的に行われルール (design precedent) が獲得される．

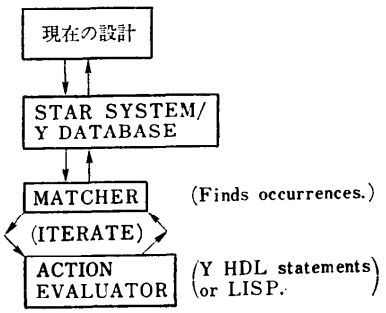


図-9 Precedent を使った設計支援の処理フロー

(1) 部分回路の分離：Isolator が指示された部分回路を外部モジュールから分離する．

(2) テキスト記述への変換：Y HDL Generator がグラフィック表現の precedent から Y言語表現に変換しライブラリ化する (図-8 の Invertor と NOR から図-11 の記述を得る)

(3) 機能シミュレーション環境 (SIMMER) をセットアップ：(2)で定義されたモジュールのシミュレーションモデルを作る．

(4) Precedent の作成：Precedent Builder が，前述の部分回路のテキスト記述 (図-11) から Precedent のパターン部 (図-12 の Y-PATTERN) を生成する．(2)で定義された構造ブロック記述を基に，Precedent Builder が Precedent のアクション部 (図-12 の Y-ACTION) を生成して，Precedent を完成する．

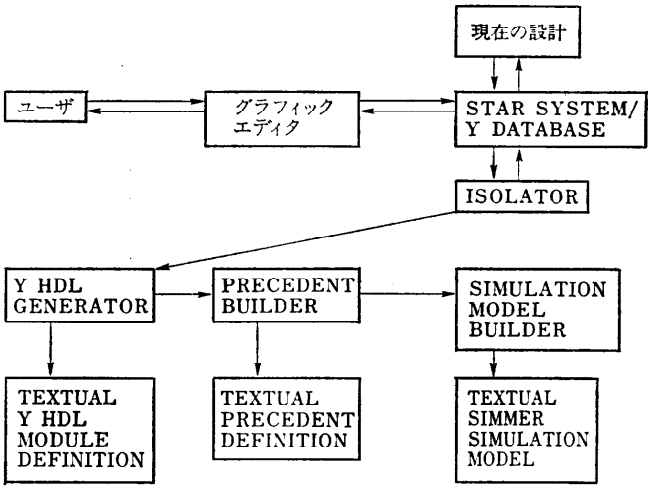


図-10 Precedent を獲得する処理フロー

```
(DEFMODULE INVNOR 0
  (MODULE 'CSRB-0 'NO025 0)
  (MODULE 'CSRA-0 'IN015 0)
  (PORT 'CB 'BIT 0)
  (PORT 'D2-0 'BIT 0)
  (PORT 'DA-0 'BIT 0)
  (PORT 'DEE-0 'BIT 0)
  (BUS 'CB 'BIT 0)
  (BUS 'D2-0 'BIT 0)
  (BUS 'DA-0 'BIT 0)
  (BUS 'DEE-0 'BIT 0)
  (CONNECT ()〉 'CSRB-0.NO025.MODULE 'B.BIT.PORT)
           ()〉 'CB.BIT.BUS))
  (CONNECT ()〉 'CSRB-0.NO025.MODULE 'A.BIT.PORT)
           ()〉 'D2-0.BIT.BUS))
  (CONNECT ()〉 'CSRA-0.IN015.MODULE 'Q.BIT.PORT)
           ()〉 'D2-0.BIT.BUS))
  (CONNECT ()〉 'CSRB-0.NO025.MODULE 'Q.BIT.PORT)
           ()〉 'DA-0.BIT.BUS))
  (CONNECT ()〉 'CSRA-0.IN015.MODULE 'A.BIT.PORT)
           ()〉 'DEE-0.BIT.BUS))
  (CONNECT ()〉 'CB.BIT.PORT) ()〉 'CB.BIT.BUS))
  (CONNECT ()〉 'D2-0.BIT.PORT) ()〉 'D2-0.BIT.BUS))
  (CONNECT ()〉 'DA-0.BIT.PORT) ()〉 'DA-0.BIT.BUS))
  (CONNECT ()〉 'DEE-0.BIT.PORT)
           ()〉 'DEE-0.BIT.BUS)))
```

図-11 Y言語によるモジュール定義例 (文献20) より)

ただし，以上の手順で獲得されたルールは，設計者により陽に提示された設計例を，基本的にそのままの形でルール化したものである．つまり，ルールのパターン部やアクション部が，一般化されることはない．しかしながら，この precedent ルールは，低レベルの構造記述をより抽象度の高い記述に変換したり，回路の検証，シリコンコンパイラでの最適化などに利用可能である．また，次のような興味深い経験談を述べている．『CONSTELLATION の利用によって，(design precedent の利用頻度から) セルライブラリのどのプリミティブが最もよく利用されるかとか，どのような構造をマクロセルジェネレータとして作成すべきかなどを学ぶ上で有効なツールであることが分かった．』

4.4 LEAP[13]

LEAP は，有効なルールの抽出，及び，ルールの一般化をも自動化する．

LEAP のルール獲得方式を理解するためには，まず，その基本思想である "Learning Apprentice System" の考え方を押えておく必要がある．Learning Apprentice System とは，対話的な問題解決コンサルタントシステムで，かつ，利用者が問題解決に寄与した部分をモニタしそれを分析することによって新しい知識を獲得するシステムである．具体的には，LEAP では，利用者が，対話的 VLSI 設計支援システム (VEXED[3]) を使っていることを前提にする．VEXED では，まず，ある抽象モジュールに要求仕様が与えられる．VEXED は，このモジュールを詳細化するために利用可能なルールを探し，そのルールをすべて利用者に提示する．利用者は，適当なルールを選

```
〈DEFPRECEDENT INVNOR-PRECEDENT
      "A NOR gate with one input inverted."
  :Y-PATTERN
  ((MODULE 'CSRB-0 'NO025 0)
   (MODULE 'CSRA-0 'IN015 0)
   (BUS 'CB 'BIT 0)
   (BUS 'D2-0 'BIT 0)
   (BUS 'DA-0 'BIT 0)
   (BUS 'DEE-0 'BIT 0)
   (CONNECT ()) 'CSRB-0.NO025.MODULE 'B.BIT.PORT)
            ()) 'CB.BIT.BUS))
   (CONNECT ()) 'CSRB-0.NO025.MODULE 'A.BIT.PORT)
            ()) 'D2-0.BIT.BUS))
   (CONNECT ()) 'CSRA-0.IN015.MODULE 'Q.BIT.PORT)
            ()) 'D2-0.BIT.BUS))
   (CONNECT ()) 'CSRB-0.NO025.MODULE 'Q.BIT.PORT)
            ()) 'DA-0.BIT.BUS))
   (CONNECT ()) 'CSRA-0.IN015.MODULE 'A.BIT.PORT)
            ()) 'DEE-0.BIT.BUS)))
  :Y-ACTION
  ((MODULE (UNIQUE-NAME 'INVNOR) 'INVNOR 0)
   (CONNECT (*)) 'CB.BIT.BUS)
            ()) (UNIQUE-NAME 'INVNOR)
                'CB.BIT.PORT))
   (CONNECT (*)) 'D2-0.BIT.BUS)
            ()) (UNIQUE-NAME 'INVNOR)
                'D2-0.BIT.PORT))
   (CONNECT (*)) 'DA-0.BIT.BUS)
            ()) (UNIQUE-NAME 'INVNOR)
                'DA-0.BIT.PORT))
   (CONNECT (*)) 'DEE-0.BIT.BUS)
            ()) (UNIQUE-NAME 'INVNOR)
                'DEE-0.BIT.PORT))
   (ZAP (*)) 'CSRB-0.NO025.MODULE))
   (ZAP (*)) 'CSRA-0.IN015.MODULE))))
```

図-12 Precedent 記述例 (文献20) より)

択するとVEXEDは，そのルールを実行してモジュールを詳細化する．次に，利用者は，詳細化したいモジュールを選択し，VEXEDがそのモジュールに適用可能なルールを探索するというように，上記のことが繰り返される．ただし，利用者は，VEXEDが選出したルールに従うことなく，自分の好みのモジュール詳細化を，モジュールエディタを使って実行できる．ここに，LEAPが介在する場面が発生する．つまり，利用者がエディタをとおして実現する設計例を観察して，それをモジュール詳細化ルールとして抽出するわけである．LEAPでは，この抽出したルールをさらに一般化する．以下，この獲得手順を図-13の例を使って説明する．

〔LEAPでのルール獲得方式〕

設計者がVEXEDの設計案を拒否して，自らの設計を行ったときに，LEAPが起動される．たとえば，図-13(a)のモジュール仕様に対してVEXEDが提示した図-13(b)の回路構成案を設計者が否定して，図-13(c)の解を作成したとしよう．つまり，正論理回路(ORやAND)ではなくMOS向けに負論理回路(ここではNOR)を使った分解を好んだわけである．LEAPは，この状況をモニタし，図-13(a)と図-13(c)から，図-14のルールを獲得する．LEAPが獲得したルール(図-14)で，注目すべき点は，ルールの条件部に記述された関数仕様(Functionの部)が図-13(a)の例のものより一般化されている点である．つまり，図-13(a)での(Or (Value Input 1 (i)) (Value Input 2 (i)))と(Or (Value Input 3 (i)) (Value Input 4 (i)))がそれぞれ，〈bool-fn 2〉と〈bool-fn 1〉に一般化されている．ここで，〈bool-fn i〉は，一般のブール関数を示す．ただし，関数仕様のトップレベルのAndは，そのまま保持される．逆に，このAndはこのルールで重要な役割りを果たすキーワードになっている．

次に，LEAPが，図-13(a)と図(c)から図-14のルールを獲得する手順を詳述しよう．

(1) 検証過程：まず，設計者が提示した解(図-13(c))が，もとの要求仕様(図-13(a))を満足するかを，図-15(b)に示したドメイン理論を使って検証する．図-15(a)は，この場合の検証過程を示す．ステップ1で，まず，設計者の解からその関数仕様(〈composed-spec〉)を生成し，これと元の要求仕様(〈original-spec〉)が等価なことを証明する．この証明は，図-15(b)のドモルガンの定理や，2重否定を削除する定理を使って行われる．

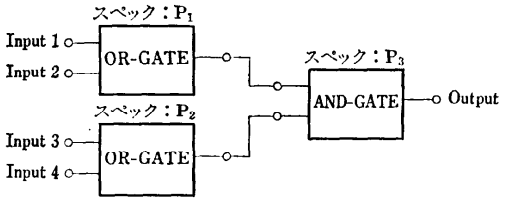
(2) 一般化過程：検証過程で生成された証明木を使って，サブモジュール(図-13(c)の各モジュール)を一般化する．この一般化のポイントは，検証過程で使われた二つの変換ルール(De-Morgan と Remove-Double-Neg)のそれぞれの前提条件(Precondition)を，証明木と逆向きに(後向きに)伝播させ，対応する箇所を一般化することである．最後に，元の要求仕様の一般形を，一般化されたサブモジュールの仕様から再度証明過程と同じ変換を行って求める．これら，一般化された関数仕様を使って図-14のルールを作成する．

なお，LEAPが獲得するルールは，設計基本ステップレベルの知識であり，制御レベルのルールではない

モジュール要求仕様:
Function to be Implemented:
Inputs: Input1. Input2. Input3. Input4
Outputs: Output
Function: (Equals (Value Output (i)) (And
(Or (Value Input1 (i)) (Value Input2 (i)))
(Or (Value Input3 (i)) (Value Input4 (i)))))

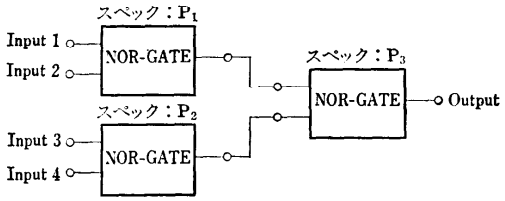
(a) モジュール機能仕様

VEXED の設計案:



(b) VEXED の設計案

設計者の解



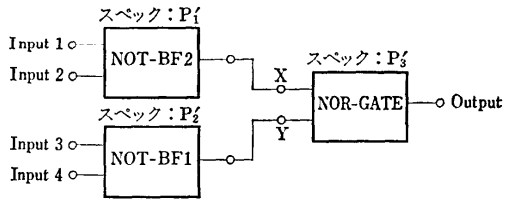
(c) 設計者の解

図-13 LEAP がモニタした設計例

ことに留意されたい．制御レベルのルールの獲得方式は，ここでも今後の課題となっている．また，LEAP では，検証過程を必須とするが，この検証が困難であったり，証明ができたとしても証明木が膨大となった場合に一般化過程が困難となるなどの問題がある．

次に，LEAP に関連した研究と，最近の動向を紹介

IF 関数仕様が次の形式である
Inputs: *any*
Outputs: Output
Function: (Equals (Value Output (i))
(And <bool-fn2> <bool-fn1>))
THEN 次のモジュール構成で実現できる



三つのモジュールのスペックはそれぞれ次のようになる．
P1': (Equals (Value X(i)) (Not <bool-fn2>))
P2': (Equals (Value Y(i)) (Not <bool-fn1>))
P3': (Equals (Value Output(i)) (Not (Or X(i) (Y(i)))))

図-14 LEAP が獲得したルール

Step 1: Forming the Composed Specification from rule RHS
(EQUALS (VALUE Output (i))
(NOT (OR (NOT (OR (VALUE Input1 (i)) (VALUE Input2 (i))))
(NOT (OR (VALUE Input3 (i)) (VALUE Input4 (i)))))))
Step 2: Verifying the Circuit Functionality
(EQUALS <composed-spec> <original-spec>)
or in this case
(EQUALS
(NOT (OR (NOT (OR (VALUE Input1 (i)) (VALUE Input2 (i))))
(NOT (OR (VALUE Input3 (i)) (VALUE Input4 (i))))))
(AND (OR (VALUE Input1 (i)) (VALUE Input2 (i)))
(OR (VALUE Input3 (i)) (VALUE Input4 (i)))))
VERIFICATION
(NOT (OR (NOT (OR (VALUE Input1 (i)) (VALUE Input2 (i))))
(NOT (OR (VALUE Input3 (i)) (VALUE Input4 (i))))))
↓
De-Morgan
↓
(AND (NOT (NOT (OR (VALUE Input1 (i)) (VALUE Input2 (i)))))
(NOT (NOT (OR (VALUE Input3 (i)) (VALUE Input4 (i)))))))
↓
Remove-Double-Negation
↓
(AND (OR (VALUE Input1 (i)) (VALUE Input2 (i)))
(NOT (NOT (OR (VALUE Input3 (i)) (VALUE Input4 (i))))))
↓
Remove-Double-Negation
↓
(AND (OR (VALUE Input1 (i)) (VALUE Input2 (i)))
(OR (VALUE Input3 (i)) (VALUE Input4 (i)))))

(a) 検証過程

ドモルガンの定理
(De-Morgan's Law)
Precondition:
(NOT (OR <bool-fn1> <bool-fn2>))
Postcondition:
(AND (NOT <bool-fn1>) (NOT <bool-fn2>))

2重否定の削除定理
(Remove-Double-Negation:)
Precondition:
(NOT (NOT <bool-fn>))
Postcondition:
<bool-fn>

(b) ドメイン理論 (Domain Theory)

Step 3: Determining the Generalized Composed Specification
COMPUTING THE GENERALIZED COMPOSED SPECIFICATION
(AND (OR (VALUE Input1 (i)) (VALUE Input2 (i)))
(OR (VALUE Input3 (i)) (VALUE Input4 (i)))))
↓
Remove-Double-Neg
↓
(AND (OR (VALUE Input1 (i)) (VALUE Input2 (i)))
(NOT (NOT <bool-fn1>))))
↓
Remove-Double-Neg
↓
(AND (NOT (NOT <bool-fn2>))
(NOT (NOT <bool-fn1>))))
↓
De-Mogan
↓
(NOT (OR (NOT <bool-fn2>)
(NOT <bool-fn1>)))
Step 4: Determining the Generalized Original Specification
GENERALIZED SPECIFICATIONS OF SUBMODULES
P_1': (EQUALS (VALUE X(i)) (NOT <bool-fn2>))
P_2': (EQUALS (VALUE Y(i)) (NOT <bool-fn1>))
P_3': (EQUALS (VALUE Output (i)) (NOT (OR X(i) Y(i))))
Step 5: Forming the New Implementation Rule

(c) 一般化過程

図-15 LEAP におけるルール獲得過程

する.

〔LEAP との関連研究と動向〕

(1) Ellman らの研究[21]: LEAP と同様に, 一般化手法としては, Explanation-Based Generalization 法を使っている. この手法は, ラトガース大の Mitchell や, イリノイ大の DeJong らにより提案されたもので, Ellman も DeJong のグループのメンバである. Ellman は, 循環シフトレジスタの仕様から, 任意の入れ替えを計算できるシフトレジスタの一般的回路のスキーマを得る方式を示している. ここでも, まず, 証明木をつくるが, その後の一般化過程は, LEAP のものとは異なる. そこでは, まず, 仕様に出現する定数を変数に変換して, 仕様を一般化する. 一般化された仕様を, 証明木のルートからリーフへ伝播させ, リーフでの制約条件を得るという手法である. Ellman の方式の限界として, 仕様の表現能力が十分でないことと, 一般化手続きが単純であることから, シフトレジスタの長さを一般化することができないなどをあげている.

(2) VILLA[14),22)]: LEAP では, ブール関数を前提とした2値の世界での論理回路の検証を前提としていたが, 多値を扱う MOS の回路に関するルールを獲得するときには, サブモジュールの内部に MOS の特殊な構造を保持する回路を検証したり一般化することが必要になり, LEAP の方式がそのまま適用困難となる. そこで, VILLA では, モジュールの動作条件を入力と出力信号の両方を用いて表現することに基づき, LEAP での問題点を解決する方式を提案し, MOS の構造を陽に意識したモジュール詳細化ルールの獲得を実現した.

(3) Hall の研究[23]: LEAP では, 設計者が実現した解を, システムが検証できなかったときは, その解から新しいルールを作成することをあきらめる. しかしながら, 現実の回路では, ループ構造を部分回路にもつものや, 内部状態を保持するモジュールなどがある場合, 全体の回路動作を数式的に検証することは, 非常に困難である. つまり, LEAP での検証は失敗する. Hall らは, この検証/説明が失敗したときを新しいルールを獲得できるチャンスと考え, 次のようなルール獲得方式を提案した.

まず，同じ機能を果たすが異なる構造をしている二つの回路と既知の設計ルールが入力される．入力された二つの回路に既知の設計ルールを適用し，それらが等価であることの説明を試みる．**図-16** は，この適用後のイメージを示す．ここで，z^{-1} は，1クロック分の時間遅れを示す．つまり，そこでは，二つの回路を入出力の両方からパターンマッチングをとることで，二つの回路の対応関係を説明していく．この対応の説明が困難になったとき，説明できない部分の回路（図-16 で点線で囲った部分）が等価とみなされ，その部分に対応するルールが作成される．この場合，両方の部分回路（点線で囲った部分）が，複数のモジュールで構成されているため，いずれもルールの条件部にはなれず，次の二つのルールとして獲得される．つまり，$f(x,y)\rightarrow z^{-1}(\mathrm{XOR}(x,y))$ と $f(x,y)\rightarrow \mathrm{XOR}(z^{-1}(x), z^{-1}(y))$ である．ここで，$f(x,y)$ は，システムが自動的に生成した関数である．また，このほか，すでに得られているルールを再分析することにより，より一般的なルールを得る手法も提案している．

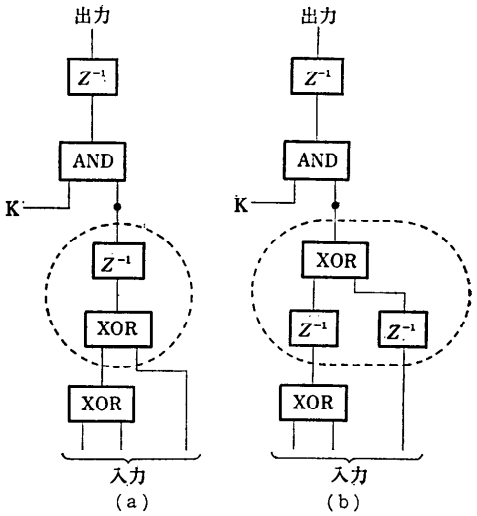


図-16 検証の失敗からのルール獲得

5. 今後の課題

以下，VLSI 設計エキスパートシステムでの知識獲得における今後の課題を各知識レベルごとに整理する．

5.1 設計基本ステップレベル

このレベルの知識をルールとして獲得する方式の現状を，前章で事例をとおして詳述した．このレベルの知識は，互いに独立性が高く，ルールとして獲得するのに適しているが，次の課題がある．

（1） ルールの粒度 (grain size) の問題：ルールの条件部と帰結部のギャップに関する問題である．つまり，条件部でマッチするモジュールの抽象度と帰結部で生成されるモジュールの抽象度のギャップが大きすぎるとルール適用効率は上がるが，変換途中の変更に対する柔軟性が欠けてくる．したがって，ルールを獲得するとき，その粒度をいかにして適正化するかが課題となる．もちろん種々の粒度をもつルールを保持することは可能である．

（2） ルール作成のための検証問題：このレベルのルールでは，基本的に，条件部にマッチする回路と，帰結部で生成される回路は機能的に等価でなければならない．したがって，ルールを作成するときは，変換前後の回路の等価性を検証しなければならない．しかし，現時点では，ブール関数に対する検証技術は確立しているものの，より複雑な回路（MOS，ループ，状態をもつ構造など）に対する検証技術は十分であるとはいえない．ルールの正当性を保障するために，より高度の検証技術が望まれる．定性的推論による検証技術などは，この意味で重要となる．さらに，高度な検証技術は，ルールを一般化するための Explanation-Based Generatization 手法の適用可能範囲を拡大する上でも重要となる．

（3） ルールの一般化手法の問題：従来の帰納的学習手法に代わって LEAP などで提案された演繹的学習手法（説明ベースの学習）は単一例から一般化を行うものとして注目を集めているが，前述したように説明のためのドメイン理論が不完全であったり，それを基にした証明木が膨大になると現実的に適用することが困難となる．したがって，現実的な一般化手法として，各種手法の長所を生かし，欠点を補い合うような統合方式を確立することが望まれる．

（4） 獲得環境を整備する問題：現時点では，SOCRATES, CONSTELLATION は，実用レベルに最も近いルール獲得環境を提供しているといえるが，今後，LEAP などで提案されている Learning Apprentice 的な獲得環境へ向けた各種技術の確立が望まれる．たとえば，設計エキスパートシステムと獲得システムをよりタイトに統合化し設計者との協調的作業環境を充実するようなマンマシンインタフェース系の確立が重要となる．

5.2 設計オブジェクトレベル

ここでは，モジュールライブラリの整備支援，作成

自動化とともに，設計者からの抽象度の高い不完全な要求仕様を，完全な要求仕様に変換する技術が重要となる．つまり，完全な要求仕様を生成するためのドメイン依存知識の獲得と，その知識ベース化が必要となる．なぜならば，その知識も，各種ドメイン，テクノロジに依存したものになると予想されるからである．

5.3 制御レベル

制御レベルの知識を，設計基本ステップレベルの知識と明確に区別することはエキスパートシステムを発展させていく見通しを良くする点で重要である．

ただし，制御レベルの知識は，設計基本ステップレベルのものとは違って互いに依存性が強く，その作成においてはほかへの影響を十分に考慮する必要がある．たとえば，SOCRATES でのメタルールを作成するときは，個々の最適化ルールの役割や互いの優先度関係など大局的観点が要求される．したがって，このレベルの知識を自動または半自動で獲得する技術は重要でありニーズも高いが，設計基本ステップレベルでの獲得技術を確立するより困難となると予想される．ただし，推論プロセスの緻密なトレーサ，説明機能，評価機能を充実させることによって間接的に制御レベルの知識作成を支援することができるであろう．さらに，これらの機能を強化することにより自動化への道が開かれる．

6. おわりに

VLSI 設計エキスパートシステムを開発する上での知識獲得の問題点と獲得事例に関して解説した．そこではまず，VLSI 設計での知識獲得の問題点を整理し，設計知識を設計オブジェクト，設計基本ステップ，制御の三つのレベルに分類し，それぞれのレベルに関する獲得方式を概観した．その後で，設計基本ステップレベルの知識をルールとして獲得する方式について，事例をまじえて詳述した．そこでは，設計者へのインタビュを重ねて設計ノウハウを人手により収集しルール化する方式から，ルールを簡便に入力できるグラフィックインタフェースを提供しルールの検証や入力を効率化したもの，さらに，人間の設計プロセスをモニタして自動的に有効なルールを抽出し，一般化する自動獲得方式などを紹介した．さらに，各種知識レベルごとにそこでの今後の課題について整理した．今後ともこの分野に対する活発な研究開発活動が期待される．

最後に，本稿をまとめるにあたり有益な示唆をいただいた当研究所山本昌弘部長，小池誠彦課長に深謝します．

参考文献

1) Kowalski, T. J.: An Artificial Intelligence Approach to VLSI Design, Kluwer Academic Publishers (1986).
2) de Geus, A. J. and Cohen, W.: A Rule-Based System for Optimizing Combinational Logic, IEEE Design and Test of Computers, Aug., pp. 22-32 (1985).
3) Mitchell, T. M., Steinberg, L. I. and Shulman, J. S.: A Knowledge-Based Approach to Design, IEEE Trans. on Pattern Analysis and Machine Intelligence, Vol. PAMI-7, No. 5, Sep., pp. 502-510 (1985).
4) Birmingham, W. P. and Kim, J. H.: DAS/Logic: A Rule-Based Logic Design Assistant, Proc. of Conference on Artificial Intelligence Applications, pp. 264-268 (1985).
5) Saito, T., Sugimoto, H., Yamazaki, M. and Kawato, N.: A Rule-Based Logic Circuit Synthesis System for CMOS Gate Arrays, Proc. of the 23rd Design Automation Conference, pp. 594-600 (1986).
6) Kim, J. and McDermott, J.: TALIB: An IC Layout Design Assistant, Proc. of AAAI, pp. 197-201 (1983).
7) Mori, H. et al.: WIREX: VLSI Wiring Design Expert System, Proc. of VLSI 85, pp. 303-312 (1985).
8) Joobbani, R.: An Artificial Intelligence Approach to VLSI Routing, Kluwer Academic Publishers (1986).
9) Stefik, M. et al.: Towards the Principled Engineering of Knowledge, the AI Magazine, Summer, pp. 4-16 (1982).
10) Marcus, S., McDermott, J. and Wang, T.: Knowledge Acquisition for Constructive Systems, Proc. of IJCAI '85, pp. 637-639 (1985).
11) Eshelman, L. and McDermott, J.: MOLE: A Knowledge Acquisition Tool That Uses Its Head, Proc. of AAAI '86, pp. 950-955 (1986).
12) 西田，堂下：LSI の動作記述からの知識獲得について，情報処理学会，論文誌，Vol, 26, No. 6, pp. 1057-1068 (Nov. 1985).
13) Mitchell, T. M., Mahadevan, S. and Steinberg, L. I.: LEAP: A Learning Apprentice for VLSI Design, Proc. of IJCAI '85, pp. 573-580 (1985).
14) 渡辺，岩本，山之内，松田：VILLA: VLSI 設計知識獲得システム，信学会，人工知能と知識処理研究会資料，AI 86-1 (1986).
15) 渡辺：エキスパートシステムにおける知識獲得，情報処理，Vol. 28, No. 2, pp. 167-176 (Feb.

1987).
16) McAllester, D. A.: A Three Valued Truth Maintenance System, MIT AI Memo 473 (May 1978).
17) deKleer, J. : An Assumption-Based Truth Maintenance System, Artif. Intell., Vol. 28, pp. 127-162 (1986).
18) Kelly, V.: The CRITTER System-Automated Critiquing of Digital Circuit Design, Proc. of the 21 st Design Automation Conf. pp. 419-425 (1984).
19) Lathrop, R. H. and Kirk, R. S.: Precedent-Based Manipulation of VLSI Structures, Proc. of the 23 rd Design Automation Conf., pp. 667-670 (1986).
20) Lathrop, R. H. and Kirk, R. S.: A System Which Uses Examples To Learn VLSI Structure Manipulations, Proc. of AAAI '86, pp. 1024-1028 (1986).
21) Ellman, T.: Generalizing Logic Circuit Designs by Analyzing Proofs of Correctness, Proc. of IJCAI '85, pp. 643-646 (1985).
22) Watanabe, M, : Learning Implementation Rules in VLSI Design by Monitoring Design er's Solutions, NEC Res. & Develop., No. 83, pp. 1-8 (Oct. 1986).
23) Hall, R. J.: Learning by Failing to Explain, Proc. of AAAI '86, pp. 568-572 (1986).

(昭和62年3月10日受付)