設計自動化30－2
（1986. 1. 21）

論理設計における回路の自動合成、および回路最適化のための一手法

松尾　健治　　久野　巧　　諏訪　基
（㈱安川電機製作所）（電子技術総合研究所）（電子技術総合研究所）

1．はじめに

近年、回路の大規模化、複雑化、および回路に対する高信頼化の要請に伴い回路設計を支援するシステムに対する要求が高ってきた。

筆者らは、このような要求に対し、トップダウンな回路設計に基づいたアルゴリズムレベルからレイアウトレベルまでの設計を支援するシステムの研究を進めている。

設計を支援するためには、設計対象の階層的構造を適切に表現することが重要である。設計対象の適切な表現は、設計者と設計支援システムとの円滑な対話を促し、設計者の設計活動を効果的に支援するための必要条件になる。また、設計対象表現はひとつの設計対象に関する複数の抽象化レベルでの記述からなる。ここで考える抽象化レベルとしては、アルゴリズムレベル、レジスタトランスファレベル（ＲＴレベル）、ゲートレベル、サーキットレベル、およびレイアウトレベルである。設計対象における階層化には、異なる抽象レベル間の関係を表わす上位－下位階層と、ある抽象レベルにおける設計対象を記述する全体－部分階層が考えられる。この階層化のうち全体－部分階層を実現する手段として、ｔｙｐｅ　ａｎｄ　ｃｏｍｐｏｎｅｎｔの概念［１］を用いた。この概念は、どの抽象レベルにも有用であること、設計過程における種々の目的に応用できること等にその力点が置かれている。

このようなｔｙｐｅ　ａｎｄ　ｃｏｍｐｏｎｅｎｔの概念に基づいて、ここでは論理レベル（ゲートレベル、レジスタトランスファレベル）を対象にした設計対象のモジュール化と階層化、およびＡＩ手法を導入して専門家の知識を容易に利用できるようなシステムを考える。

モジュール化は、適当な大きさに分割された対象回路を１つの独立した世界と見なしてＵｒａｎｕｓ［６］の多重世界機構を用いて実現している。

階層化は、異なる抽象レベル間の上位－下位階層（ｉｓａ階層）、および同一レベル内の全体－部分階層（ｐａｒｔ－ｏｆ階層）として実現している。また、上位－下位階層ではｉｓａ－ｃｏｎｎｅｃｔという概念を導入し、抽象レベルの異なる階層間の整合性を保ち、設計者の負担軽減を計っている。

現在ゲートレベル、ＲＴレベル、ミックスレベルにおけるイベントドリブン方式によるシミュレータ、回路検証機能（ベリファイヤー）、および回路の自動合成機能（シンセサイザー）を実現しているが、本論文では特に、ＲＴレベルにおける回路の自動合成手法、および回路の最適化手法について論じる。、

回路の自動合成では、回路の動作記述と回路データベースに格納されている回路記述から回路の構造を合成する手法と、与えられた回路の構造記述から最適化された回路構造を合成する手法について述べる。また、最適化処理では回路全体を考慮した最適化が、非常に困難なため回路の機能単位レベルでの局所的な最適化を回路の動作記述に適用することによって実現している。

この手法を用いることにより、回路の再利用が容易になっている。

以下、２章で回路の仕様記述を説明した後、３章、４章では回路の自動合成と最適化手法、５章でまとめについて述べる。

2．論理設計における仕様記述

現在、ハードウェア記述言語としては、ＣＤＬ、ＳＤＬ、ＤＤＬ等多くの言語が提案されているが、ここでは少なくとも次のような条件を満足するような仕様記述を考える。

(1) 階層レベルに依存しない表現能力を持つ。
　（単一の記述を行うだけで様々なレベルの記述を同時に可能にするような記述言語）
(2) 設計過程において生ずる種々の目的に共通な仕様記述であること。
　（シミュレーション、ベリフィケーション、シンセシス等の目的に対して一貫して使用できること）

以下、対象となる論理回路の仕様記述を回路の静的な構造を宣言する構造記述と、回路の動的なふるまいを宣言する動作記述について簡単に説明する。

なお、ここでは中島［６］によって開発されたUranus(PROLOG/KRの拡張版）を用いて記述しており、図１、２、３にそれぞれゲートレベルにおけるfull-adderの回路記述例、AND ゲートの回路記述例、およびＲＴレベルにおけるadder 回路の記述例を示す。

2.1 構造記述

これは、外部入出力端子記述、コンポーネント記述、内部接続情報の記述、内部状態の記述よりなる。以下、簡単に説明する。

(1) 外部入出力端子記述

これは、次に示すように述語 extと入出力端子記述よりなる。さらに、入出力端子記述は、対象とする端子が入力か、もしくは出力であるかの情報、端子名、および回路名によって表現する。

（ext　入出力端子記述）

入出力端子記述：(in or out　端子名　回路名）

(2) コンポーネント記述

ここでは、回路内部で使用されているコンポーネント（部品）の宣言を行う。

（parts　回路名　コンポーネント名）

たとえば、図１に示すfull-adderはX1,X2 というExclusive-ORゲートと、a1,a2 というAND ゲート、およびO1というORゲートがコンポーネント（部品）として存在することを示す。

(3) 内部接続情報の記述

これは、対象回路内部の接続情報を記述するもので、述語connect とネット名、入出力端子記述１、および入出力端子記述２によって表現される。

（connect　ネット名　入出力端子記述１
　　入出力端子記述２）

(4) 内部状態の記述

これは、順序回路における内部状態を記述するもので、述語internal-stateと状態名、およびコンポーネント名と、そのコンポーネントにおける内部状態名よりなる。

（internal-state 状態名１　（状態名２　コンポーネント名））

たとえば、
(internal-state cont1 (cont jk-ff1))の場合、対象回路がcont1 という内部状態を持ち、JKフリップフロップJK-ff1の内部状態contと、１対１に対応することを示す。
但し、primitive 回路は、コンポーネントを持たないために状態名１のみで表現される。

2.2 動作記述

これは、回路遅延の記述と回路の動作記述よりなるが、ゲートレベルとＲＴレベルでは、その記述方法が若干異なる。以下、簡単に説明する。

(1) 回路遅延の記述

これは、$behavior という述語と遅延時間により記述され、ゲートレベルシミュレーション時の回路遅延時間の指定に用いる。但し、ＲＴレベルシミュレションでは、回路遅延を考えないために遅延時間の指定は行わない。

（$behavior 遅延時間）

(2) 動作記述

ゲートレベルにおける動作記述は、プロダクションシステムにおけるＩＦ－ＴＨＥＮルールの形式で記述する。たとえば、「２入力ＡＮＤゲートの入力が共に１ならば、出力は１である」ことを表わすには、次のように記述する。

```
(behavior and
    (if ((state (in 1 and) on)
         (state (in 2 and) on)
    (then ((state (out 1 and) on))))
```

また、ＲＴレベルにおける動作記述は、図３に示すように機能素子単位レベルの代数式で表現する。

```
;
;;;  Full-Adder Description
;

(with fa

;
;;  External Description
;
        (az (ext (in a fa)))
        (az (ext (in b fa)))
        (az (ext (in ci fa)))
        (az (ext (out s fa)))
        (az (ext (out co fa)))

;
;;  Parts Description
;
        (az (PARTS xor x1))
        (az (PARTS xor x2))
        (az (PARTS and a1))
        (az (PARTS and a2))
        (az (PARTS or o1))
;
;;  Connection
;
        (az (connect net1 (in a fa) (in 1 x1)))
        (az (connect net2 (in a fa) (in 1 a1)))
        (az (connect net3 (in b fa) (in 2 x1)))
        (az (connect net4 (in b fa) (in 2 a1)))
        (az (connect net5 (in ci fa) (in 2 x2)))
        (az (connect net6 (in ci fa) (in 1 a2)))
        (az (connect net7 (out 1 x1) (in 1 x2)))
        (az (connect net8 (out 1 x1) (in 2 a2)))
        (az (connect net9 (out 1 a1) (in 2 o1)))
        (az (connect net10 (out 1 a2) (in 1 o1)))
        (az (connect net11 (out 1 x2) (out s fa)))
        (az (connect net12 (out 1 o1) (out co fa)))
        )
```

図１　full-adderの回路記述例

```
;
;;;  And-Gate Description
;

(with and

;
;;  External Description
;
        (az (ext (in 1 and)))
        (az (ext (in 2 and)))
        (az (ext (out 1 and)))

;
;;  Behaviornal Description
;

        (az ($behavior 1))

        (az (behavior and
                (if ((state (in 1 and) on)
                     (state (in 2 and) on)))
                (then ((state (out 1 and) on)))))

        (az (behavior and
                (if ((state (in 1 and) off)))
                (then ((state (out 1 and) off)))))

        (az (behavior and
                (if ((state (in 2 and) off)))
                (then ((state (out 1 and) off)))))
        )
```

図２　AND ゲートの回路記述例

```
;
;;;  Adder Description(RT level)
;

(with adder

;
;;  External Description
;

        (az (ext (in a adder)))
        (az (ext (in b adder)))
        (az (ext (out s adder)))

;
;;  Behaviornal Desciption
;

        (az ($behavior))
        (az (behavior adder (s := a ++ b))))
```

図３　ＲＴレベルにおけるadder 回路の記述例

３．回路の自動合成１

回路の自動合成では、与えられた回路の動作記述(behavior)から回路合成に必要な情報を抽出した後、回路データベース内に格納されているprimitive 回路の適用を行うことによって目標とする回路の構造(structure) を自動合成する。但し、ここではユーザによってボトムアップに組立てられた回路もprimitive 回路と呼んでいる。

また、与えられたbehaviorに対して専門家のヒューリスティックを利用して最適化処理を行うことにより、最適回路の合成が可能である。

図４に回路の自動合成過程を示し、以下、回路データの構造、回路データベースの検索、最適化処理について述べる。

3.1 回路データの構造

回路データは、加算器、乗算器といったすでに与えられている回路の情報であり、回路合成時には対象回路のコンポーネントになる。

この回路データは、対象回路のbehaviorを、各機能単位に分割した後、prologのホーン節（ルール節）に変換することによって得られる。

たとえば、test1 という回路のbehaviorが

s = (a + b) * c という代数式で与えられた時、次のような手順で回路データを得る。

①　まず、与えられたbehaviorを接頭辞記法に変換する。

```
prefix --> (equal s (times (plus a b) c))
```

②　次に、入出力情報を付加する。

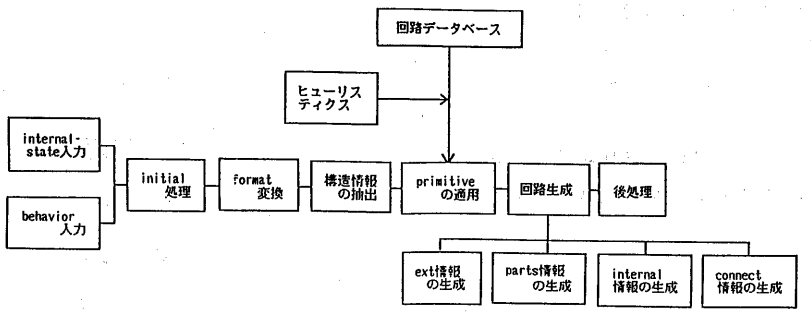

図４　回路の自動合成過程

```
(equal (out s test1) (times (plus
  (in a test1) (in b test1)) (in c test1)))
```

③　各コンポーネント単位に分割する。ここでのコンポーネントは、加算器、および乗算器である。

```
(plus (in a test1) (in b test1) r001),
(times r001 (in c test1) (out s test1))
```

④　prologのホーン節に変換する。

```
(assert (circuit test1)
  (struc (plus (in a test1) (in b test1) r001)
  (times r001 (in c test1) (out s test1))
    $var))                    ---（３－１）
```

3.2 回路データベースの検索

対象とする回路を合成する際に問題となるのが、回路データベースを如何に効率よく検索するかということである。これは、前述した回路データの構造とともに回路合成時に入力となる対象回路の構造、および回路データベース検索手法に大きく関係する

以下、対象回路の構造について記述した後、回路データベースの検索、いわゆるprimitive 回路の適用法について説明する。

(1) 対象回路の構造

回路データの場合と同様に構造情報を抽出するために、与えられた対象回路のbehaviorは、コンポーネント単位（素子）に分割されるが、回路データが、分割された各コンポーネントをprologのルールとしてPrologのデータベース内にアサーションされるのに対して、対象回路の場合は、各コンポーネントをそれぞれfact形式でアサーションされる。たとえば、対象回路のbehaviorが、s = (a + b) * c として与えられた場合、次のようになる。

```
(assert (struc (plus $a $b $s1) (plus (in a
                    ①
  test1) (in b test1) s1))) ---（３－２）
(assert (struc (times $s1 $c $s) (times s1 (
                    ②
  inc test1)(out s test1))) ---（３－３）
```

なお、①②は、次項で説明する回路データベース検索時に回路データがユニファイされる。

(2) 回路データベースの検索

回路データベースの検索は、回路データのルール節を評価し、分割された対象回路のコンポーネントをPrologのユニフィケーション機能を使って逐次取り出していくことによって行われる。

たとえは、回路データベースに格納されている前記（３－１）式(circuit test1) を評価した場合（３－２）、（３－３）式とユニファイに成功し、対象回路がコンポーネントtest1 より構成されていることがわかる。

しかし、単純にルール節を評価した場合、「合成回路の冗長性」、「最適なコンポーネントの選択が

できない」等の問題が生じ、効率のよい回路データベースの検索ができない。

たとえば、図５に示すように回路データベースにp1,p2,p3といった回路データが格納されており、目標とする対象回路がplus（加算器）、times（乗算器）というコンポーネントにより構成されていた場合、組み合わせとしては、p2（２個）とp3（１個）、p1（１個）とp2（１個）が考えられるが、前者を選択した場合、後者に比しコンポーネント数が多くなり回路としては冗長になる。この問題は、いわゆるプロダクションシステムにおけるconflict resolution（競合解消）問題として扱うことができる。

このような、競合を解消する知識は次のようなものが考えられる。

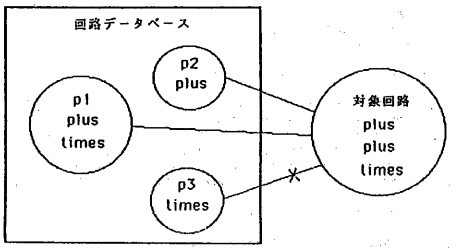

図５

① より複雑な回路から適用していく

図５で説明したように、回路内のコンポーネント数を最小にするように回路データを適用するための知識である。

② 回路データ⊂対象回路（適用する回路データが、対象回路の部分集合でなければならない。）

回路データベースと対象回路が図６に示すように与えられていた場合、p2は対象回路の部分集合にならないため適用されず、p1が適用される。

③ 数値情報があれば、それを優先する。

たとえば、図７に示すように対象回路内にインクリメンターが含まれていたとしよう。
この場合、合成回路としては加算器を用いて実現する場合と、インクリメンター自身を用いて実現する場合が考えられるが、回路データベース内に両方共に格納されていた場合は数値情報を含むインクリメンターを優先する。

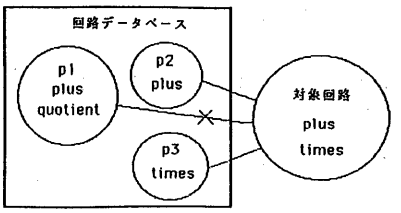

図６

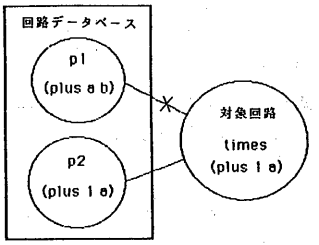

図７

3.3 最適化処理

回路における最適化処理には、回路内の配線数を最小にするような処理、レイアウトレベルにおける配線長を最小にするような処理等が考えられるが、ここでは、レジスタトランスファレベルにおける素子数が最小になるような処理を考える。

これは、回路内の素子数を最小にするような回路最適化ルール（ヒューリスティック）を対象回路のbehaviorに適用して行う。

また、合成された対象回路が最小素子数であることを保証するには、回路の取り得るすべての組み合わせを調べる必要があるが、回路の組み合わせ爆発という問題を生じ、実際には非常に困難である。

従って、ここでは、数学公式を利用したbehaviorの変換を行うことによって準最適解を求めることにする。

たとえば、対象回路のbehaviorが（３－４）式で与えられた時、前記最適化ルールを適用することによって準最適解である（３－５）式を得る。回路合成は、この（３－５）式を用いて行われるが、これはコンポーネントとして加算器と乗算器が各１個づつ少なくなっている。

s = (a + b) *c + (a + b) * b－－－（３－４）

s = (a + b) * (b + c)　　－－－（３－５）

なお、現在６１の最適化ルールを用意している。

3.4 回路の自動合成例１

回路の自動合成例を図８（ａ）に示すＢａｒｒｏｗ［３］のＣｏｕｎｔｅｒモデルで説明する。

この回路は、マルチプレクサ（ＭＵＸ１）、インクリメンター（ＩＮＣ１）、およびレジスタ（ＲＥＧ１）の３コンポーネントで構成されている。各コンポーネントのbehaviorは、（３－６），（３－７），（３－８）式のようになり、すでに回路データベース内に格納されているものとする。

ＭＵＸ：out = if switch then in0 else in1
－－－（３－６）

ＩＮＣ：out = in + 1　　－－－（３－７）

ＲＥＧ：(internal-state contents) －（３－８）
contents = in1
out1 = contents

なお、(internal-state contents) はレジスタの内部状態を表わす。

ここで、入力情報として図８（ｂ）に示す動作記述を与える。

（３－９）式は、対象とする回路が内部状態contentsl を持つことを宣言するものであり、（３－１０）式は合成しようとする対象回路の動作記述本体である。

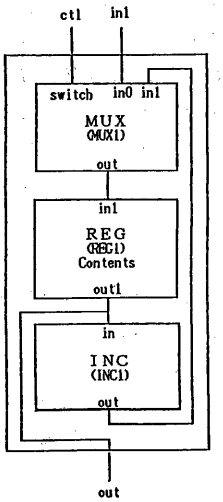

図８（ａ）　Ｃｏｕｎｔｅｒモデル

前記、入力情報に従って回路を自動合成した結果を図８（ｃ）に示す。

```
Input : internal-state ---> contents1 －（３－９）

behavior ---> if ctl then in1 －（３－１０）
                 else contents1 + 1,
              out1 = contents1
```

図８（ｂ）　動作記述

```
;
;; Circuit Generation
;

(with GENE1

;; **- External in or out Definition -**

(az (EXT (IN CTL GENE1)))
(az (EXT (IN IN1 GENE1)))
(az (EXT (OUT OUT1 GENE1)))

;; **- Parts Definition -**

(az (PARTS INC I1020))
(az (PARTS MUX M1021))
(az (PARTS REGISTER R1022))

;; **- Connect Definition -**

(az (CONNECT OUT-OUT1 (OUT OUT1 R1022) (OUT OUT1 GENE1)))
(az (CONNECT IN-CTL (IN CTL GENE1) (IN SWITCH M1021)))
(az (CONNECT IN-IN1 (IN IN1 GENE1) (IN IN0 M1021)))
(az (CONNECT OUT-OUT (OUT OUT I1020) (IN IN1 M1021)))
(az (CONNECT OUT-OUT (OUT OUT M1021) (IN IN1 R1022)))
(az (CONNECT OUT-OUT1 (OUT OUT1 R1022) (IN IN I1020)))

;; **- Internal Definition -**

(az (INTERNAL-STATE CONTENTS1 (CONTENTS R1022)))

;; **- Behavior Definition -**

(az ($behavior))
(az (BEHAVIOR GENE1 (CONTENTS1 := IF CTL THEN IN1
                                  ELSE CONTENTS1 ++ 1)))
(az (BEHAVIOR GENE1 (OUT1 := CONTENTS1)))
```

図８（ｃ）　自動合成結果

４．回路の自動合成２

前章では、対象回路の動作記述(behavior)から、その構造を合成する手法について述べたが、本章では与えられた構造記述(structure) から最適化された回路を合成する手法について説明する。

図９に示すように、これは与えられたstructureから、まずbehaviorを抽出する。この抽出されたbehaviorに対して前章で述べた最適化ルールを適用し、最適化された回路構造を得る。

4.1 動作記述(behavior)の抽出

回路の構造記述より、動作記述を抽出する過程では、まずｐａｒｔｓ記述より対象回路を構成しているコンポーネント情報、およびコンポーネントの動

作記述を得る。

次に、ｅｘｔ，ｃｏｎｎｅｃｔ記述よりそれぞれ、外部端子の入出力情報、コンポーネント間の情報が得られるが、この情報より前のコンポーネントの動作記述に対して代入操作を逐次繰り返すことにより対象回路の動作記述を得る。

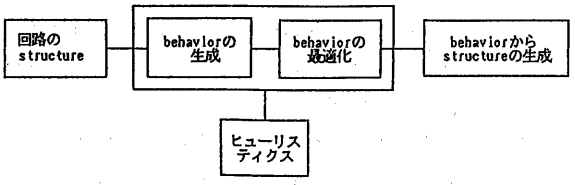

図９　回路の自動合成過程

4.2 回路の自動合成例２

以下、回路の自動合成例を図１０（ａ），（ｂ）で説明する。

（１）まず、前記アルゴリズムに従ってstructure (a) よりbehavior（４－１）式を抽出する。

（２）（４－１）式に最適化処理を適用して（４－２）式を得る。

（３）この（４－２）式から３章で示した回路合成手法を用いて回路図１０（ｂ）を得る。

```
(behavior circuit1 (s = a * b + a * c))－（４－１）

(behavior circuit1 (s = a * (b + c)))　－（４－２）
```

```
;
;; Strucure
;

(with circuit1

;; **- External in or out Definition -**

(az (ext (in a circuit1)))
(az (ext (in b circuit1)))
(az (ext (in c circuit1)))
(az (ext (in s circuit1)))

;; **- Parts Definition -**

(az (parts multi mul1))
(az (parts multi mul2))
(az (parts adder add1))

;; **- Connect Definition -**

(az (connect in-a (in a circuit1) (in a mul1)))
(az (connect in-b1 (in b circuit1) (in b mul1)))
(az (connect in-b2 (in b circuit1) (in a mul2)))
(az (connect in-c (in c circuit1) (in b mul2)))
(az (connect out-s1 (out s mul1) (in a add1)))
(az (connect out-s2 (out s mul2) (in b add1)))
(az (connect out-result (out s add1) (out s circuit1)))
```

図１０（ａ）　構造記述

```
;
;; Structure Optimization
;

(with STRUC1

;; **- External in or out Definition -**

(az (EXT (IN A STRUC1)))
(az (EXT (IN B STRUC1)))
(az (EXT (IN C STRUC1)))
(az (EXT (OUT S STRUC1)))

;; **- Parts Definition -**

(az (PARTS ADDER A0995))
(az (PARTS MULTI M0996))

;; **- Connect Definition -**

(az (CONNECT IN-B (IN B STRUC1) (IN A A0995)))
(az (CONNECT IN-C (IN C STRUC1) (IN B A0995)))
(az (CONNECT IN-A (IN A STRUC1) (IN A M0996)))
(az (CONNECT OUT-S (OUT S A0995) (IN B M0996)))
(az (CONNECT OUT-R (OUT R M0996) (OUT S STRUC1)))

;; **- Internal Definition -**


;; **- Behavior Definition -**

(az ($behavior))
(az (BEHAVIOR STRUC1 (S := A X ( B ++ C ))))
```

図１０（ｂ）　自動合成結果

５．まとめ

レジスタトランスファレベルにおける回路の動作記述から回路を合成する手法、最適化手法、および回路の構造記述から最適回路を合成する手法について述べた。今後は、他の階層レベルにおける回路合成、回路全体を考慮した最適化等について考えていきたい。

謝辞

本研究の機会を与えて頂いた柏木　寛電子計算機部長、また種々の教示を頂いた人間機械システム研究室の方々に深謝します。

参考文献

[１] W.M.Vancleemput :AN HIERACHICAL LANGUAGE FOR THE STRUCTUAL DESCRIPTION OF DIGITAL SYSTEMS , 14th Design Automation Conference , 377-385

[２] Harry G.Barrow : VERIFY A program for Proving Correctness of Digital Hardware Design , Artificial Intelligence vol24 (1984)473-491

[３] Michel R.Genesereth : The Use of Design Description in Automated Diagnosis , Artificial Intelligence vol24 (1984) 411-436

[４] Saito,Uehara,Kawato : A CAD SYSTEM FOR LOGIC BASED ON FRAMES AND DEMONS , 18th Design Automation Coference , 451-456

[５] Kawato , Uehara , Hirose , Saito : AN INTERACTIVE LOGIC SYNTHESIS SYSTEM BASED UPON AI TECHNIQUES , 19th Design Automation Conference , 858-864

[６] Nakasima:Uranus Reference Manual,ETL-RM-85-1E(1985)

[７] 久野、中島、大表、田村、諏訪：設計支援システムにおける設計対象の階層的表現方法，情報処理学会，第２９回全国大会論文集

[８] 中島、久野、田村、大表、諏訪：多段階の抽象化を許す設計支援対象の記述，情報処理学会，第２９回全国大会論文集

[９] 南谷崇：論理合成，情報処理，VOL.25,NO.10 (1984)1071-1077

[１０] 川戸、斎藤：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 (1984)1161-1168