設計自動化28−4
(1985・9・17)

論理型言語による論理設計支援システム

Logic Design Support System in a Logic Programming Language

林　一司　丸山　文宏　真野　民男　角田　多苗子
Kazushi HAYASHI　Fumihiro MARUYAMA　Tamio MANO　Taeko KAKUDA
川戸　信明　上原　貴夫
Nobuaki KAWATO　Takao UEHARA
富士通株式会社
FUJITSU LTD.

はじめに

論理設計においては，最近，色々なCAD プログラムが実用化されてきているが，それらは個別的なものであり，論理設計全体を支援しているものは少ない．特に，ハードウェアとしての動作の決定を行う機能設計は，人間の設計者でなければできないと考えられていた．我々は設計者のノウハウを用いることにより，機能設計から回路設計までの論理設計全体を支援するシステムの試作を行った．システムをインプリメントするには，論理型言語であるPrologを用いた．

このシステムは従来のCAD システムではサポートされていない機能設計までカバーしているので，設計支援の出発点のレベルはハードウェアの動作アルゴリズムへと上げられている．即ち，仕様（動作アルゴリズム）を与えると，システムがそれを満たすハードウェアの動作の決定（機能設計）を行い，さらに，その動作を実現する回路を設計し，各素子間の接続関係までを決定する（回路設計）．ハードウェアの動作アルゴリズムは，並列動作が明確に記述される点を除けば，通常のプログラミング言語による記述と同様であり，ユーザはハードウェアを詳しく知らなくても書き下すことが出来る．このことは，一般のユーザでも，並列性を利用したアルゴリズムを設計すれば，このシステムを利用してLSI が設計できることを意味している．

仕様を記述するためには，並列性を表現し易いように設計されたプログラミング言語occam[1]を採用した．システムはoccam によって記述されたハードウェアの動作アルゴリズムを入力として受取り，CMOSによる回路(CMOS基本セル・CMOS複合セル及びそれらの接続関係）を出力する．

このようなシステムにおいては，設計者のノウハウを有効に利用するとともに，従来のCAD システムによって支援されてきたアルゴリズムに基づく処理もうまく組み入れることが重要である．両者をうまく結合できれば，信頼性の高い質の良い設計をおこなうシステムを構築することが可能となる．本研究は，論理型言語の上に知識ベース及びアルゴリズミックな処理を統合した知的CAD システムの実現をめざしている．

1．システム構成

このシステムは機能設計と回路設計の2つのフェイズから構成されている．2つのフェイズの中間段階としてレジスタ・トランスファ・レベルの記述が設定されている．この記述にはハードウェア記述言語DDL[2]を採用した．機能設計フェイズはoccam による仕様の記述からDDL によるレジスタ・トランスファ・レベルの記述の生成を行い，回路設計フェイズはDDL からCMOS複合セルによる回路の設計を行う．

各フェイズは更にいくつかのサブシステムから構成されている（図1）．

ハードウェア化サブシステムはoccam による並列アルゴリズムを解析して，DDL による有限状態機械(finite-state machine)の記述を生成する．ステートマシン最適化サブシステムは，DDL による有限状態機械の記述を分析して最適化を行い，それらを再構成し完全なDDL の記述とする．

回路を設計するにはハードウェアの構造に関する情報が必要であるが，DDL による記述は直接にはそのような情報は提供していない．従って，DDL によるハードウェアの機能記述から，実際のハードウェアの構造記述への変換を行わなければならない．ここでは，トランスレータサブシステムがそれを行なっている．このサブシステムはDDL を解析して，データパスに関する情報と状態遷移に関する情報を抽出している．

制御回路設計サブシステムは，有限状態機械を実現するために，トランスレータサブシステムの出力する状態遷移に関する情報に基づいて状態遷移の制御を行う回路を設計する．データパス設計サブシステムはレジスタ，メモリ，アダー，カウンタ等のコンポーネントの周辺回路を設計する．

ここまでの処理は，使用するテクノロジとは独立に行うことができる．

制御回路設計サブシステムとデータパス設計サブシステムは共に論理式を生成するが，これらの論理式は組合せ論理回路としてCMOS複合セルによって実現される．関数分割サブシステムは，論理式を複合セルを実現するのに適した小論理式へと分割する．これらの小論理式は複合セル設計サブシステムによって，実際の CMOS 複合セルとしてのレイアウト[3]が行われる．

レジスタ，メモリ，アダー，カウンタ，デコーダ，I/O ピン等のコンポーネントは，基本セル割当サブシステムによって，基本セル

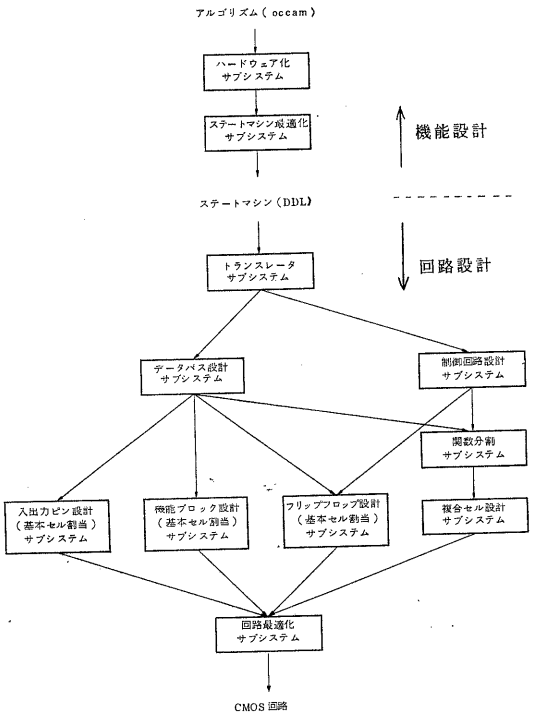


図1　システム構成

ライブラリから割り当てられる．基本セル割当サブシステムは，可能であればライブラリからのセルを割り当てるが，適当なセルが登録されていない場合は基本セルを組合せて生成する．

回路最適化サブシステムがさらに最適化を行う．これは，複合セル及び基本セルの接続関係等を調べて，冗長なセルの除去などを行う．

2．機能設計

機能設計は，今までほとんど計算機による支援が行われていなかった部分であり，設計者の持っているノウハウに大きく依存している．従来の設計においては，機能設計の過程は主に設計者の頭の中で行われ，明確な知識の形では外部には明らかにされていなかった．我々は，機能設計の過程を分析し，そこで用いられている設計者の知識を抽出した．抽出した知識は，ルールの形で知識ベースに登録している．知識をルール化するにあたっては，論理型言語の述語を用いて設計対象の間の関係が自然に表現できることに注意し，これを活かすように知識の表現を行った．

基本的な推論メカニズムとしては，前向き推論を採用しており，各処理の知識をルール（具体的にはPrologの節）の形で表現したものを動的につないでいくことにより，機能設計の過程を実現している．

人間の設計者が設計を行う場合は，一度にすべての設計を行うわけではなく，全体の仕様を見ながら徐々に具体的な機能を明らかにしてゆくが，この過程は，ルールの実行に伴う副作用が環境（ワーキングメモリ）を変化させてゆくことにより計算機上に実現している．このワーキングメモリはPrologのfactを用いて構成しており，環境を表現するために40個ほどの述語を使用している．このワーキングメモリの更新は，Prologのassertとretract を用いて行なっている．

このシステムにおける機能設計は，occamによるソフトウェアとして記述された仕様にハードウェアを対応付ける過程と考えることができる．従って，そこで使われている知識は，ソフトウェアとハードウェアの間の橋渡しをするものが中心となっている．

たとえば，ハードウェアの最大の特徴はその並列動作にあるから，ソフトウェアの仕様からできる限り多くの並列性を引き出すことが望ましい．このためには，図2 に示すような知識がある．これはoccam における逐次的な代入操作を，ハードウェアにおける並列なレジスタ転送操作に圧縮するための知識である．これは，

(1) 引き続く２つの操作がともに代入である．

(2) ２つの変数ともレジスタで実現されることななっている．

(3) 互いに干渉していない．

ならば，この２つの操作はハードウェアの同一サイクルで実行できる，ことを表している．

この知識の適用の例として，図3 のoccamの記述をあげる．この記述は，変数r に変数t の値を代入したのち，変数t をTRUEに初期

```
compatible (Operation_1, Operation_2):-
    store_typed_operation (Operation_1, Variable_1, Source_1, ...),
    store_typed_operation (Operation_2, Variable_2, Source_2, ...),
    Variable_1 \ == Variable_2,
    implementation (Variable_1, register, .....),
    implementation (Variable_2, register, .....),
    not (appear (Variable_1, Source_2)).
```

図２　オペレーション圧縮の知識

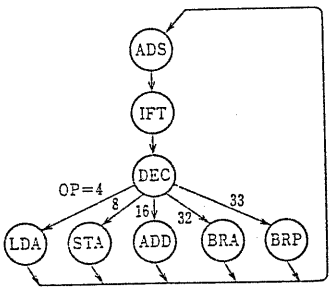
化することを示している．このソフトウェアにおける逐次動作は，ハードウェアにおいては図4 に示される2 つのレジスタ転送の並列実行によって実現できる．このDDL 記述は，変数r とt が両方ともレジスタで実現される場合，同一サイクルでそれぞれにt と1 (TRUE)を転送できることを示している．

3．回路設計

回路設計の部分の処理は，機能設計の部分に比べるとアルゴリズミックに行われるものが多い．例えば，トランスレータサブシステムは，DDL によるハードウェアの機能記述をハードウェアの構造記述に変換するが，この変換はアルゴリズムに従って機械的に行われている．しかし，すべての処理がアルゴリズミックに行なえるわけではなく，やはり知識が必要である．

たとえば，制御回路設計サブシステムでの状態数の最小化を考える．

図5 のような状態遷移図が与えられた場合，これは図6 のような制御回路によって実現することができる．ところで，状態遷移図を見ると分るように，ステートDEC から次の5 つのステートへの分岐はレジスタOPの値によって決まっている．LDA ～BRP の5 つのステートを命令の実行状態ということで1 つにまとめても，個々のステートはレジスタOPの値で区別することができる．実行状態を一括して



ADS：番地設定 (Address Set)
IFT ：命令読出し (Instruction Fetch)
DEC：命令解読 (Decode)
LDA：LOAD 命令の実行
STA：STORE 命令の実行
ADD：ADD 命令の実行
BRA：BRANCH 命令の実行
BRP：BRANCH-POSITIVE 命令の実行

図 5　状態遷移図

```
SEQ
      .
      .
   r := t
   t := TRUE
      .
      .
```

図 3　occam の記述

```
      .
      .
r <- t,  t < - 1.
      .
      .
```

図 4　DDL の記述

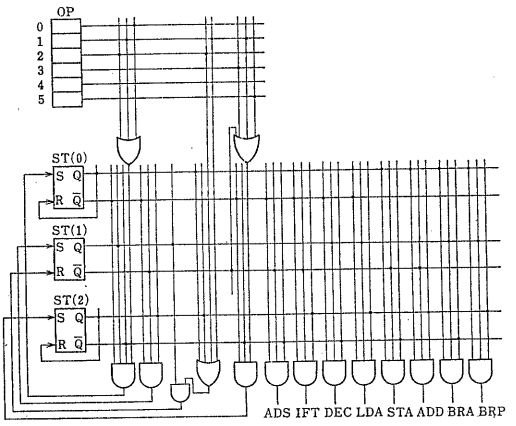


図 6　制御回路

EXC とすれば，状態遷移図は図7 のように簡単化される．これに対する制御回路は図8 に示すようなものになるが，これは図6 に比べるとかなり簡単になっている．

しかしながら，この方法が利用できるのは次の4 つの条件が満たされている時に限られている．

(1) 分岐がレジスタの値で区別されている．

(2) 分岐先のステートには2 つ以上のステートから遷移して来ることはない．

(3) 分岐先への遷移時にレジスタの値を変更していない．

(4) 分岐先のステートで，レジスタの値の変更を伴うような自分自身への遷移を行っていない．

このルールをPrologで記述したものが図9に示してある．(1)～(4)で示している述語はそれぞれ条件の(1)～(4)に対応している．

また，複合セル設計サブシステムは，ヒューリスティックなアルゴリズムによりCMOS複合セルの内部のレイアウトを決定している．これは，アルゴリズミックな処理の良い例である．

このシステムでは論理式が与えられた時，適切にFET の並べ換えを行い，それを実現するCMOSセルの大きさを最小にすることを行い，従来は設計者の人手で行われていたセル内の最適配置まで自動化している．

最適配置を求めることは，論理回路のグラフモデル中のオイラーパス(一筆書きが可能な経路) を求めることに帰着される．与えられた論理式の等価回路を求めて，そのオイラーパスを探してゆく．たとえば，図10の回路が与えられたとする．今仮に，CMOSのNANDとNOR ゲートしか使用ができないとすると，実

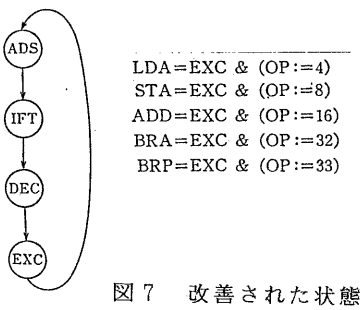


図7　改善された状態遷移図

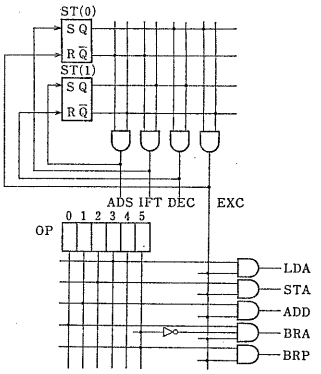


図8　改善された制御回路

```
branch_state_reduction(State) :-
    next_states(State, Next_states),
    for(X in Next_states,
        setof(Y, (transition_condition(State, Next_states, Register, Condition), .... (1)
                  unique_predicessor(X, State), ..................................... (2)
                  unaffected(State, X, Register), ................................... (3)
                  unaffected(X, X, Regsiter), ....................................... (4)
                  Y = (X, Condition)),
              Z)),
        update_table(State, Z).
```

図9　Prologによる状態数最小化の知識の記述

際にセルを割り当てると図11のようなレイアウトが得られる．これに対して，図12のような等価回路に変換すると，複合セルの性質を利用することによりもっと少ない面積で実現できる．図13に最適なCMOS複合セルのレイアウトを示すが，図11のレイアウトのほぼ半分となっている．

しかし，すべての等価回路を作り出し，そのオイラーパスを見つけ出すことは，コンピュータにとっても非常に負荷が大きい．そこで，ヒューリスティックなアルゴリズムによる最適化を考える．そのアルゴリズムは以下のようなものである．

(1) 偶数個の入力をもつすべてのゲートに，擬似入力を付加する（図14 (a)）．

(2) 論理回路表現において，擬似入力と実入力との交差が極小になるように，入力の並びを入れ換える（図14 (b)）．

このヒューリスティックなアルゴリズムは必ずしも最適配置を与えるとはかぎらない．しかし，結果的にセパレーション領域がない場合，最適配置となる．

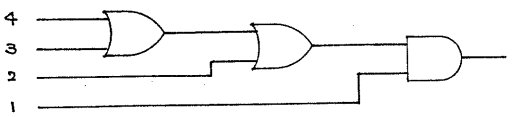


図10 回路例

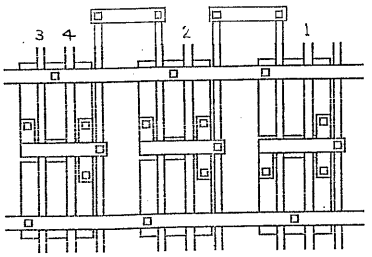


図11 NAND, NORによるインプリメント

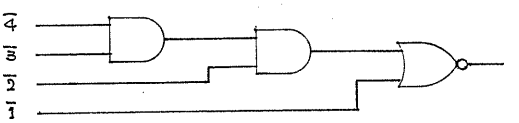


図12 等価回路

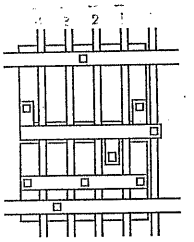
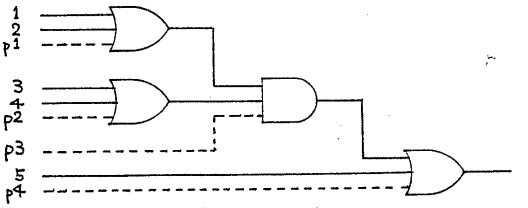
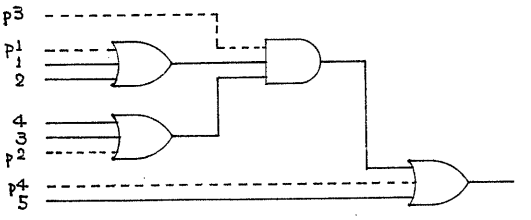


図13 複合セルによるインプリメント



(a) 元の回路



(b) 極小交差の等価回路

図14 極小交差アルゴリズム

4．実行例

この章では，実際にこのシステムを用いた設計の例を示す．例題としては，シストリックアレイによるパターンマッチャ[43]をとりあげる．これは，ある有限長の文字列（PATTERN）が，与えられた文字列(STRING)中に埋め込まれているかどうかを検査するものである（図15）．

まず，動作アルゴリズムをoccam によって記述する（図16）．システムはこれを入力として機能設計を行う．不明な点についてはユーザに対して質問を行う（図17）．

機能設計フェイズはその結果としてDDL によるハードウェアの機能記述を出力する（図18）．トランスレータサブシステムはこのDDL 記述を解析し，ハードウェアの構造に関する情報を取り出す．それらの情報から，テクノロジとは独立な論理回路図が図19のように求まる．この図の左半分はコンパレータを，右半分がアキュムレータを表している．

レジスタ等は基本セルから割り当て，組合せ回路は複合セル設計サブシステムによって最適なCMOS複合セルのレイアウトが生成される（図20）．．

この例においては，occam によるアルゴリズムの記述からCMOS複合セルのレイアウトを求めるのに，VAX11/780 のCPU 時間で約12分を要した．ここで生成された複合セルの個数は57個であった．

5．まとめ

Prologを用いて，論理設計全体の支援を行うシステムを試作した．　この作業を通じて，論理設計支援エキスパートシステムの構築における論理型言語の有用性を確認した．

通常，設計作業は複数個の対象間の関係を参照しながら進められて行く．つまり，ここで用いられる知識の多くは，対象間の関係を表現するようなものである．論理型言語の性質（述語が対象間の関係を表している）は，このような知識の記述を行うのにたいへん有効であった．

また，ルールと手続きが同一のインタフェースで呼びだせるので，両者の結合を自然に行うことができた．

今後は，さらにルールの充実を計ることが必要である．また，回路設計情報，基本セル情報などは異なるレベルの設計情報である．現在はこれらのレベルを特に区別してはいないが，これらの設計情報を構造化すれば関係する知識も構造化され，さらにわかりやすくなると思われる．さらに，学習機能などについても考慮してゆかなければならない．

謝辞

本研究は，第５世代コンピュータプロジェクトの一環として行われたものである．御支援頂いたＩＣＯＴ第一研究室古川室長に深く感謝します．

［参考文献］

［１］Taylor, R. and Willson, P.
OCCAM : Process-oriented language meets demands of distributed processing, Electronics, Nov.30, 1983

［２］Dietmeyer, D.L.
Logic Design of Digital System, Allyn and Bacon, 1971

［３］Uehara, T. and vanCleemput, W.M.
Optimal Layout of CMOS Functional Arrays, IEEE trans, Vol.30, No.5, 1981

［４］Foster, M.J. and Kung, H.T.
Design of Special-Purpose VLSI Chips : Example and Opinions, CMU-CS-79-147, 1979

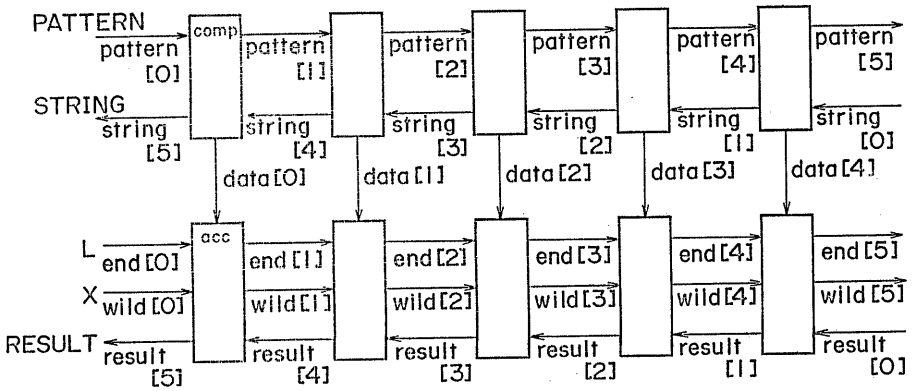


図15　パターンマッチャ

```
CHAN pattern [6]:
CHAN string [6]:
CHAN data [5]:
CHAN end [6]:
CHAN wild [6]:
CHAN result [6]:
PROC comp (CHAN pin, sin, pout, sout, dout ) =
  VAR p, s :
  SEQ
    PAR
      p := 0
      s := 0
    WHILE TRUE
      SEQ
        PAR
          pout ! p
          sout ! s
        PAR
          pin ? p
          sin ? s
        dout ! p = s:
PROC acc (CHAN xin, lin, rin, din, xout, lout, rout )=
  VAC d, x, l, r, t:
  SEQ
    PAR
      x := FALSE
      l := FALSE
      r := FALSE
      t := TRUE
    WHILE TRUE
      SEQ
        PAR
          xout ! x
          lout ! l
          rout ! r
        PAR
          din ? d
          xin ? x
          lin ? l
          rin ? r
        IF
          l = TRUE
            SEQ
              r := t
              t := TRUE
          l = FALSE
            t := t /\ (x \/ d):
PAR i = [1 FOR 5]
  PAR
    comp (pattern [i-1], string [5-i],
          pattern [i], string [6-i], data [i-1])
    acc (wild [i-1], end [i-1], result [5-i],
         data [i-1], wild [i], end [i], result [6-i])
```

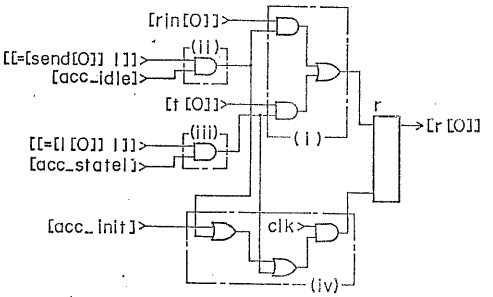
図16　occam による仕様の記述

```
yes
| ?-functional_design.
Parsing your specifications in OCCAM...

Implementing OCCAM variables...

Should variable l have only one bit ? y/n
|:y
How many bits should variable s have ?
|:8
Should variable p have as many bits as variable s ? y/n
|:y
Should variable r have only one bit ? y/n
|:y
Should variable x have only one bit ? y/n
|:y
Compressing a sequence of operations...

Implementing inter-process communication...

Can the entire system be controlled by a single clock ? y/n
|:y
Would you prefer faster overall communication ? y/n
|:y
Generating partial DDL descriptions...

Constructing the final DDL code from partial DDL descriptions...

yes
| ?-
```

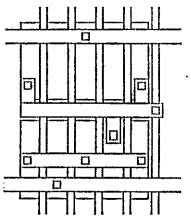
図17　インタラクション

```
〈system〉 pm.
 〈time〉  clk.
 〈entrance〉 pin(8), sin(8), xin, lin, rin, din.
 〈exit〉 pout(8), sout(8), dout, xout, lout, rout.
 〈terminal〉 sendl.
 〈automaton〉 comp:clk:
  〈register〉 p(8), s(8).
  〈states〉
    init: p←0, s←0, →idle.
    idle: pout=p, sout=s,
          p←pin, s←sin, →state2.
    state2: sendl=1, dout=(p:=s), →idle.
  〈end〉.
 〈end〉 comp.
 〈automaton〉 acc:clk:
  〈register〉 d, x, l, r, t.
  〈states〉
    init: x←0, l←0, r←0, t←1, →idle.
    idle: sendl: xout=x, lout=l, rout=r,
                 x←xin, l←lin, r←rin,
                 d←din, →statel.
    statel: |*l*| r←t, t←l
                ; t←(t & (x | d))., →idle.
  〈end〉.
 〈end〉 acc.
〈end〉 pm.
```

図18　occam から得られたDDL 記述



(a)　レジスタr の周辺回路



(b)　(iv) の部分のCMOS複合セル

図20　複合セルによる回路の実現

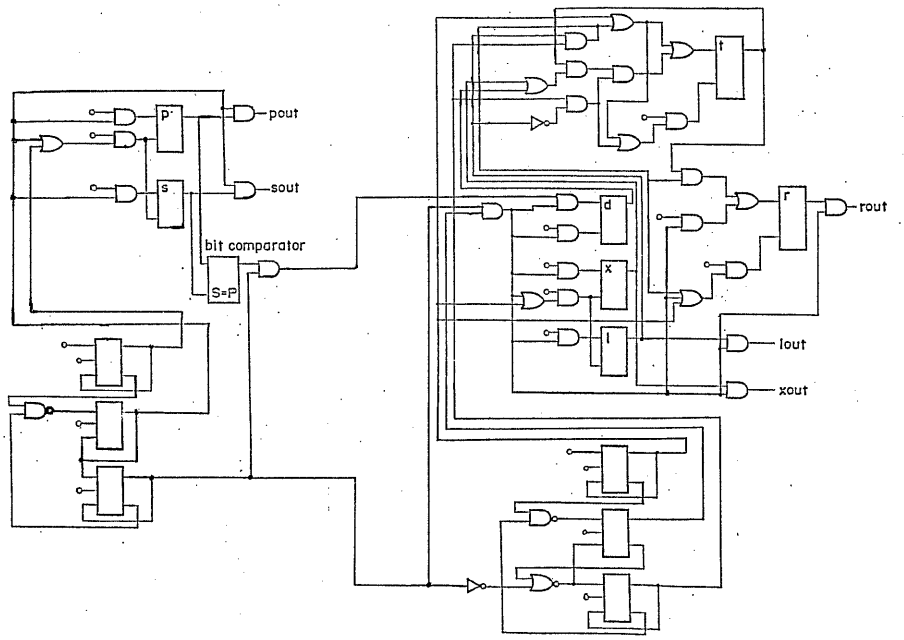


図19　論理回路図