計算機設計の自動化*

元岡　達**

1. 序　論

計算機の設計，製造の一部に計算機を利用しようという試みは，製造会社を中心に近年活発に進められてきたが，利用できる計算機システムからの制約もあり，少なくともわが国においては，実用化された例のほとんどが計算機のオフライン利用に限られている．したがって実用化されている分野も，基板の配置布線設計，シミュレーションによる誤設計の検出といった比較的設計方式が確立しており，人間の助けを設計途中の段階であまり借りずに，計算機だけで設計を終らせてしまうことのできるような分野に限定されてきた[1),2)]．

しかし，計算機の設計，製造に本格的に計算機を応用し，その能率を向上させるためには，システム設計から自動製造技術に直結される実装設計にいたるまでの一貫した設計自動化システムを完成する必要がある．設計業務の一部がとびとびに，独立に自動化されたのでは，それぞれの段階に対する入力を人手によって作成することとなり，その労力や入力作成途中での誤りの発生のために自動化の利点がほとんど失なわれてしまう結果となる．

一貫した設計自動化システムを完成するためには，創造性が要求されるという設計業務本来の性格からいっても，また計算機技術全般の急速な進歩，発展に即応する必要性からいっても，人間が設計自動化システムの中心となって自由にその能力が発揮できるシステムとしなくてはならない．このためには，今日各方面でその重要性が指摘されている TSS (Time Sharing System) を用い，人間が計算機を設計の道具として使うという CAD (Computer Aided Design) の思想に立った柔軟性に豊んだ設計システムを確立することが必要である．一口にいえば，計算機のオンライン利用を前提とした設計自動化システムということになる．

したがって，ここでは計算機のオンライン利用を前提として，計算機設計の自動化システムの性格や，そのあるべき姿について論じてみることとする．

計算機に関する設計自動化の分野は，まず計算機自体の具体設計に先立って行なわれるシステム設計とその評価，第2に具体的なハードウェアに関連する論理設計と実装設計，第3にハードウェアの完成と共に必要となるソフトウェアの三つの分野に大別できる．第3のソフトウェアに関しては，設計というよりむしろ製造の自動化といった方が適切かもしれない．

第1のシステム設計の分野は，設計の定式化が最も困難な分野であり，シミュレーションによるシステム評価が計算機利用の役立つ領域と考えられている．

第2のハードウェアの設計は設計の定式化が最も進んでおり，設計そのものへの計算機利用が考えられる．この分野をさらに細分してみると，回路設計，論理設計，実装設計にわけられる．論理設計には，論理の生成，最簡化，解析，合成，素子への割付けといった問題があり，評価や誤設計検出のための論理シミュレーションもこの段階の設計のために必要である．論理設計の出力データは直接次に述べる実装設計の入力となるべきものであり，実装設計との一貫性が特に望まれる．

実装設計は各論理素子の基板への割当て，基板の架への割当てと位置ぎめ，架の背面布線といった問題にわけられ，自動設計の実用化が最も進んでいる分野である．設計の評価基準が比較的明確であり，しかも大量の情報を間違いなく処理する必要があるといった計算機に適した応用分野であることや，自動製造機への入力を作る必要があることもあって，実用化が急ピッチで進められている．集積回路を採用するためには実装設計の自動化が不可欠なものであることについての認識は高まってきたが，これに対して論理設計の自動化の必要性についてはその実現が困難なこともあり，疑問視している向きも多い．しかし LSI (Large Scale Integration) を全面的に採用するためには論理設計段階の自動化が必要であり，今後大きな問題となってくると考えられる．

回路設計の自動化については，ECAP, NETI とい

---

* Computer Aided Design of Digital Computers, by Tohru Moto-Oka (Faculty of Engineering, University of Tokyo)

** 東京大学工学部

った電子回路の汎用シミュレータが用いられている程度で，最適化やオンラインシミュレーションの問題は未解決である．トランジスタなどのシミュレーションが容易でないこと，実験が比較的容易に行なえることがこの分野の発展をおくらせてきた大きな原因と考えられるが，集積回路やその大容量化が進むにつれて，実験的手法による設計が困難となることが予想され，論理設計の自動化と共に，今後その重要性が認識され始めることと思う．

ソフトウェアの自動設計及至製作に関しては，ハードウェアの設計と同様なシステムを考えることができるが，Compiler's compiler の研究が進められている段階で，supervisor などについては未だ試みられたという話を聞かない．しかし故障点検出用プログラムで使用するテストデータのように，ハードウェアの設計技術と密接な関係のある分野で自動製作技術の研究が活発に進められていることは興味深い．

このような設計自動化が進められる理由，あるいは設計の自動化によって得られる利点と考えられるものを列挙してみると，次のようになる．

（1） 価格の低減；直接的な設計経費の低減だけでなく，製造，保守を始めその影響を及ぼす範囲は非常に大きい．

（2） 設計の質の向上；いろいろな設計をし，比較検討することによって，より良い設計ができる．

（3） 人間の創造能力の向上；人間-機械相互作用が密になることによって，人間の創造活動が活発となり，また計算機に他の仕事を分担させることによって創造的な仕事により多く時間がかけられる．

（4） 設計時間の短縮；

（5） 設計変更の処理；設計変更の影響する範囲をおさえ，資料を整備することは設計システムが大きくなるにつれ，ますます厄介な仕事となるが，計算機の最も得意とする仕事である．

（6） 書類の整備；常に最新の資料を整備し，供給することは協同作業の場合特に重要な問題であり，計算機で容易に達成できる業務である．

（7） 設計者の能率向上；

（8） 特殊目的計算機の設計；従来，設計業務が大変なために汎用機に頼っていたような用途に，専用の特殊計算機を設計することが有利となる可能性が生じる．

（9） 協同作業の容易化；書類が整備され，設計者相互の情報交換を計算機が受持てば，協同作業は容易になり，これまで不可能視されていたような大規模なシステムの設計が可能となる．

(10) 誤設計の除去；

設計業務に計算機を導入する場合，現在の設計業務で設計者が困まっており，しかも計算機に比較的楽に仕事を移行できる項目から入ってゆくのが自然といえよう．このような観点から上記の項目をみてみると，10) 誤設計の除去，5) 設計変更の処理，6) 書類の整備，7) 設計者の能率向上，9) 協同作業の容易化，2) 設計の質の向上，といったことを目標に，第1段階の設計自動化システムは計画されるのが適当といえる．

ここではハードウェアに関連した設計自動化の技術，特にその中では遅れている論理設計の問題を中心に考えてみることとする．

2. システム設計の自動化

計算機の助けを設計業務に取入れる第1の段階として考えられることは，シミュレーションに計算機を用いることであり，その結果を評価し，設計の最適化をはかることは人間にまかせることである．計算機システムの設計は，実際の計算機製作に先立って，その外部仕様を決定する段階で行なうものと，システム導入に当たって行なうものとに大別できるが，前者はその影響する範囲が広く，特に慎重に行なわなければならない．このようなシステム設計で実用されている手法はシミュレーションであり，GPSS, SIMSCRIPT などの汎用ディジタル・システム・シミュレータが用いられるのが普通である．

シミュレーションでは，各部に生じるデータの Queue に関する統計情報や処理速度，応答時間などが求められ，またシステムを構成する各装置の利用率，記憶の利用状況などを出力する．このシミュレーションの結果から，システムの隘路を検出して，システム変更を提案するという形で設計が進められ，処理能力の評価や優先順位の割当て法，記憶の割当て法などハードウェアとソフトウェアとが総合されたシステムの評価も行なわれる．TSS のように大規模かつ複雑なシステムを設計する際には，ハードウェアの設計に先立ってこの種シミュレーションによるシステム評価を各方面から綿密に行なうことが是非とも必要である．

装置内部の動きをやや詳細にシミュレートするためには，シミュレートするシステムを記述するための問題向き言語を用いるのが普通で，SOL, LOCS, LOTIS など多くのシステム記述用語が提案されている．これ

らの言語はレジスタ間の情報転送の記述に主眼をおいたシステム記述用言語であって，register transfer language と呼ばれている．シミュレーションは後述する論理設計の各段階においても，誤設計の検出や設計評価のために必要であり，類似のシステム記述用言語が用いられる．ただ論理設計の段階では，設計のためのシステム記述言語としても使えることが望まれるから，シミュレーションに主眼をおいたものとは変ってくる．

3. 実装設計の自動化

金物の設計の中もっとも自動化の進んでいる部分はデータ量が多く，人間－機械間の相互作用をあまり必要としないと考えられる実装設計に関連した部分であり，計算機のオフライン利用による自動化システムを実用化している例が多い．

実装に関連した自動化として考慮されている問題を列挙すると，次のようになる．

（1） 論理回路の分割と各部の標準回路や標準基板への割り付け

（2） 標準基板の架への割り付けと位置の決定

（3） 基板間並びに架端子の相互接続のための布線設計

（4） 図面や資料の作成と更新

（5） 基板の設計（基板中での基本回路の位置決定相互接続路の決定）

これらの設計は相互に密接な関係があるために，問題は複雑となる．それぞれの設計が大きな規模を持っているので，一緒にして考えることには多くの困難があるが，配置と布線とは同時に取扱っている例が多い．

設計を行なう手順は算法的なものと，heuristic なものとにわけられるが，最適解をめざす算法的手順は問題が大きいために非常に大きな計算時間を必要とすることになり実用的でない．たとえば位置ぎめの問題にしても，基板上での基本回路の位置ぎめであれば，すべての可能な配置について接続路の長さを比較し，最小となるような配置を抽出することも可能であるが，架上への基板の配置の問題となると，M個の場所にN個の基板を配置する組み合わせの数は $M!/(M-N)!$ となり，基板数が数百といった典形的な問題では莫大な数の配置法があることになり，上述のような手法はとることができないことは明らかである．布線の問題にしても，n個の端子を相互に接続し，線長を最小にしたいという問題が与えられたとき，ピン間の接続だけを考えても，$n-1$ 個の枝路からなる n^{n-2} 個の樹が対象となる（結合線路の途中からの分岐を考えることのできる印刷基板のような場合には対象となる樹の数はさらに増す）．しかも配置の問題では，布線の樹が指定されていることが前提となる．

そこで最適解ではなく，近似解を求める方式がいろいろ試みられるわけであるが，配置の問題では maximum conjunction, minimum disjunction rule に従って，関係の深い要素を次々に近傍に配置していく方式が一般的である．ただ結合の強さを示す関数形にいろいろなものがある．この方式では $W_1(C_i)$, $W_2(C_i, \tilde{C})$, $W_3(C_i, \tilde{C})$ の三つの関数が評価に用いられる．C_i は i 番目の要素，$\tilde{C}$ は既に配置された要素の集合を表わし，W_1 は C_i 要素に関係する信号できまる関数，たとえば信号の数である．W_2 は C_i 要素と $\tilde{C}$ 要素群が共通に持つ信号できまる関数，W_2 は C_i 要素にあって $\tilde{C}$ 要素群にない信号できまる関数である．この規則では W_2 が最大のもののうち W_3 が最小の要素を選んで次々に配置する．この原則が使えないときには W_1 が最大の要素を選んでおき，上記の操作を繰り返えす．

このようにして選ばれた要素を配置する場所は布線長などのような距離関数を最小とするようにきめるのが普通である．このような初期配置で不満足であるときには，要素の入れかえを順次行なって最適解に近ずける．要素の入れかえのうち一番簡単なものは一対の要素を選んで場所を交換することであるが，この繰り返えしでは必ずしも最適解に収斂せず，3個以上の要素を一度に入れかえる必要の起ることがある．距離関数 d_{ij} には次のような式が用いられる．

$$d_{ij}{}^m = |x_i - x_j|^n + |y_i - y_j|^n$$

ただし (x_i, y_i), (x_j, y_j) はそれぞれ要素 C_i および C_j の座標であり，d_{ij} は C_i, C_j 間の距離である．m, n についてはそれぞれの場合に適当にきめられており 1, 1; 2, 2; 1, 2 などの例がある．この選び方は布線のとり得る経路に対する制限にも影響される．たとえば布線には縦，横の線しか認めない場合や，ななめの布線が認められる場合があり，配置や布線設計に影響を及ぼす．特に印刷配線の場合は布線経路に対する制約が多く，一般的なアルゴリズムの決定はむずかしい．とり得る経路に対する制限，径路にゆるされる導線数，ピンに接続できる導線数などが設計の条件となる．径路は heuristic な手法できめられてゆくが

この手法が使えなくなると迷路の問題をとくことになる．多層プリント板を使うときには，何層にすれば配線できるかを決定する必要がある．

布線径路をきめて行く順序としては，径路をとり難い長い布線からきめて行くのが普通であるが，長い布線はそれだけ多くの径路をふさいでしまうので，短かいものからきめてゆくという考えもある．

4. 論理設計の自動化

計算機の設計ではまず命令の目録を作り，それぞれの命令の機能を記述する．この命令機能の記述を出発点としてこれらを実現するに必要な論理式及至論理図を作成する過程が論理設計の対象であり，論理の生成，論理シミュレーションなどに計算機の利用が考えられる．

論理の生成については，**register transfer language** を使って記述された計算機の記述を入力として，それを実現する論理式を出力する方式が考えられるが，この方面の研究に関する発表は数少ない．

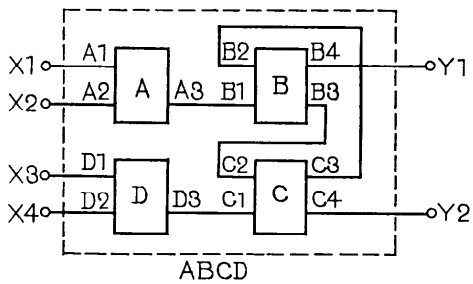
ここでは筆者らのグループで進められている研究を中心に論理の生成の問題をとり上げてみよう．このようなシステムのおもな目的は，あきあきするような単純な問題や，繰り返えし作業から設計者を開放することであり，論理的な矛盾や時間関係の衝突，論理の誤りといったものを除くことである．またファイルの自動管理や，計算機と人間との対話を通して設計の協同作業を円滑にすることも大きな目標といえる．

まず論理設計自動化システムで用いられるシステム記述言語であるが，設計をいくつかの段階にわけ，各段階で人間と計算機が自由に対話できることが好ましい．これは CAD の基本的な思想である．しかも各段階の言語が全く独立にきめられたものでは変換に手数がかかり，言語の意味を明確にする上にも多くの支障をきたす．またシステムの各部をいろいろな段階の詳しさで書き得ることは，無駄な手数を省いたり，協同作業の場合，他人に自分の意志を正確に伝える上に，重要な性質である．このような見地から，五つのレベルからなる設計用言語を提案した．

レベル 1 の言語は，論理図に近いものを表現するのに用いられ，この論理設計システムの最終出力となるものである．従来の論理記述言語と異なる点は，多重構造をみとめている点で，実装設計で論理基本回路や基板に割り付ける際に必要となる情報を多重構造の形である程度伝えるのに役立つものと考える．レベル 1

```
LEVEL 1
  ABCD (X1, X2, X3, X4; Y1, Y2)
  LEVEL 1/*EXAMPLE*/
        A(A1, A2; A3);
        B(B1, B2; B3, B4);
        C(C1, C2; C3, C4)
        ATTRIBUTE
            OPERATION=RSFF;
            DELAY=0.05
          ENDA;
        D(D1, D2; D3);
      A.A1=X1$          A.A2=X2$
      B.B1=A.A3$        B.B2=C.C3$
      C.C1=D.D3$        C.C2=B.B3$
      D.D1=X3$          D.D2=X4$
      Y1=B.B4$          Y2=C.C4
 END;
END
```

（a） 言語記述例



（b）（a）の記述が意味する論理接続図

第 1 図　レベル 1 の言語による記述例

の言語は**第 1 図**に例示するように論理モジュールを宣言する部分と，モジュール間の接続を示す部分とからなり，この論理モジュールに多重構造が認められていて，一番内側の論理モジュールの性質を示すのは ATTRIBUTE によっている．第 1 図 (a) は言語記述例で，同図 (b) はこの記述が意味する論理図を示している．レベル 1 の言語は論理設計と実装設計の中間的性格を持っている．

レベル 2 の言語は，論理式を表わすためのもので，論理操作をステートメントで表現する．分離符号には単なる接続，すなわち論理的には遅れなしの情報転送を意味する＝のほかに，1 単位時の遅れのあとに情報が転送されることを示す := が用いられる．したがって:=の左辺の変数は遅延フリップフロップに対応し，＝の左辺の変数は中間端子を表わす．レベル 2 の記述中にレベル 1 の記述を含ませることができ，レベル 1 では変数として formal parameter しか認めていない

が，レベル2では中間端子を宣言してステートメント中で使うことができる．

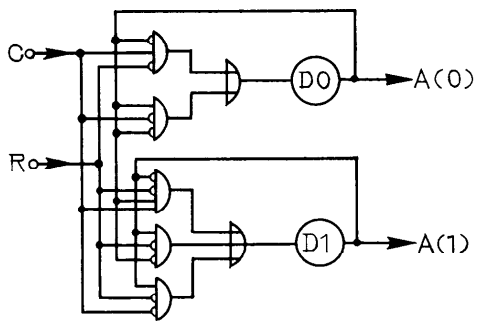
レベル2の言語による記述例を**第2図**に示す．これは同図 (b) に示す sequential machine を記述したものである．

レベル3の言語は，タイムチャートのような論理操作の時間的推移を表現するためのものである．レベル2のすべてのステートメントが1単位時ごとに意味を持っているのに対して，レベル3では1単位時ごとに実行するステートメントを指定することができる．

compound statement が1単位時の動作を表わし，その label を状態に，basic statement の list を出力に，transition part を遷移入力に対応させると，状態図との対応をつけることができる．

```
LEVEL 2 ELEMENT
  COUNTER (R, C; A(2))/*R=RESET,
    C=COUNT, A=OUTPUT*/
LEVEL 2
  TERMINAL D0, D1;
    D0:=C&¬D0&¬R|¬C&D0&¬R$
    D1:C&¬D1&D0&¬R|¬D0&D1&¬R|
    ¬C&D1&¬R$
    A(0)=D0$
    A(1)=D1
END;
END
```

（a）言語記述例



（b）（a）の記述が意味する sequential machine

第2図　レベル2の言語による記述例

宣言できるものは TERMINAL, REGISTER, 関数, モジュールの4種であり，REGISTER は通常のプログラム言語の記憶と同様に新たに値を入れない限り，前の状態が保持される記憶である．関数には MONO と UNIT があり，遅延のない論理操作で，ブール表現中で使われる．UNIT は単なる論理操作を表わす MONO は単一の金物に対応し，呼ばれるごとに同じものが使われる．モジュールは独立した制御系を表わすのに使われ，一単位時以上の遅れのあるモジュールには ELEMENT をつける，その名前が変数としても用いられ，モジュールが動作中か否かを示すのに使われる．レベル3の言語による記述例を**第3図** (a) に示す．これは 10 進計数器を記述したもので同じものを同図 (b), (c) にレベル4および5で記述したものを示す．

```
LEVEL 3 ELEMENT
  DECIMALCOUNTER (X; Y)
    LEVEL 3
    REGISTER A(4);
    TERMINAL C(3), D(4);
  ST: A:=D&¬(D(0)&D(2))$
    D=¬A&(C, X)|A&¬(C, X)$
    C=A(1: 3)&(C(0: 1), X)$
    Y=D(0)&D(2)
      :ST
END;
END
```

（a）レベル3言語による記述例

```
LEVEL 4 ELEMENT
  DECIMALCOUNTER (X; Y)
    LEVEL 4
      REGISTER A(4);
      TERMINAL C(3);
  ST: A:=¬A&(C, X)|A&¬(C, X)$
      C=A(1: 3)&(C(0: 1), X)$
      Y=A(0)&A(2);
      A:=A&¬(A(0)&A(2));
      GOTO ST
END;
END
```

（b）レベル4言語による記述例

```
LEVEL 5 ELEMENT
  DECIMALCOUNTER (X; Y)
    LEVEL 5
      REGISTER A(4);
  ST: A:=A+X;
      IF A(0)&A(2)
      THEN BEGIN Y=1; A:=0 END
      ELSE Y=0;
      GOTO ST
END;
END
```

（c）レベル5言語による記述例

第3図　10 進カウンタの記述例

レベル4の言語は，論理動作の実行順序を記述する

ためのものである．レベル3以下との相異点としては実行順序のみを示し，実行時間を制限をつけないステートメントを設けたこと，ステートメント形式やブール演算子の種類をふやしたことなどがあげられる．レベル4の言語による記述例を第3図（b）に示す．

レベル5の言語はよりマクロな記述を可能とするように作られたもので，他の register transfer language と多くの共通点がある．memory や switch などの宣言も認められ，arithmetic expression や conditional expression もある．また DO, ON, WAIT conditional statement がある．レベル5の言語による記述例を第3図（c）に示す．

各レベルの言語による記述の中にはより下位のレベルの言語による記述を含むことができるようにして，システムを記述する際，各部を任意の細かさで記述できるように配慮してある．

この言語による設計自動化システムは analyzer—converter—generator から構成される．計画中のものでは analyzer は文法解析を行なって記述文をリスト構造に分解する．converter はこのリストについて変換を行ない，一つ下位のレベルのリストにする．generator はリストを言語に変換する役割を果たす．analyzer と generator は各レベルの言語に対して共通のものを準備すればよく，converter は各レベルごとに別個に作らなくてはならない．

オンラインシステムを用いた CAD システムを構成する場合の問題点を，この言語を基礎にしたシステムについて以下に考察してみよう．

オンラインシステムの特色を発揮するためには会話モードでの使用が望まれ，このためには analyzer は interpretive な形にして入力したものから順次 list を作り，文法上の誤りが発見されれば analyzer は直ちに error message をオンラインで出力する必要がある．また設計資料はすべてファイルに記憶されることが前提となり，従来の設計で得られている標準的な方式や論理回路構成もライブラリの形でファイルに保存しておき，仕様にあったものが取り出され，利用できることが重要である．ライブラリの利用は主としてレベル5から下位のレベルに落す段階で行なわれる．

部分的な設計変更は設計途中のみならず設計終了後にも絶えず発生する問題であるが，その影響する範囲が正確に把握できなかったり，設計資料の整備に不備をもたらす原因となったりして設計業務上解決のせまられている大きな問題点であり，設計自動化システムを考える上に是非とも考慮しなくてはならぬ点である．この言語は module 構成となっているので，設計変更を module の削除・付加の問題に還元でき，また多重レベルに分かれているので，他の module が下位のレベルで記述されていればそのレベルまでは他の module に変更を加えることなく修正ができる．

協同作業のためには，ファイルの管理による資料の整備が最も重要な問題であり，TSS を設計に用いる理由のうち man-machine interaction の問題と双璧をなす重要項目である，協同作業の困難さの一つは作業者相互の情報交換が不足し，連絡の不備による誤り，無駄な作業の続発することである．計算機によって各人のファイルが常に最新のものとなっており，他の人が自由に見ることができれば，この困難性の多くは解消される．この資料の updating を計算機にまかせるということが CAD を採用する大きな理由となっている．

言語の面からみると，module, block, equivalence name といった概念が協同作業に役立ち，module や block が分担の単位となる．このため module には，interface を記述するための response sequence を書くことができるようになっている．

人間－計算機間の相互作用を密にし，計算機を設計の道具として使いやすいものにするためには，状態図 Time table, 論理図といったものを通信の手段として使うことが考えられる．CRT display と light pen といったものが金物として準備されていれば，言語の各レベルはこのような出力とよい対応がついているので，比較的簡単に変換できよう．

論理シミュレーションを行なうことが自動設計の一環として重要であることについては前にも述べた．製造に先立って論理設計の誤りを検出するためにも，故障点検出用プログラムの自動作成のためにも用いられる．

論理シミュレーションは論理式の値を求める操作に帰着するが，計算機全体の要素の数が非常に多いために1単位時ごとの各要素の値を逐次に求めることには非常に時間がかかる．しかも，かなり多くの場合について計算する必要があり，シミュレーションの能率を高め計算機の使用時間を短かくすることがまず第一に要求される．このためには論理表現の値が決定すれば計算を途中で打切るような計算法が用いられたり，また各単位時ごとにすべての要素が値を変える可能性があるわけではないから，値を変える可能性のある要素

のみについて計算する方式などが工夫されている．また信号の伝播遅れや論理的な帰還ループの存在を考慮して，racing や hazard の問題を取扱えるようにしたものもある．

前述の言語をシミュレーション用に用いたるめにはなお二，三の機能を言語の定義中に加える必要があるが，大きなシステムの一部をシミュレートしたい場合，その部分だけ低いレベルで書き，他の部分は高いレベルの言語で書くことによって能率のよいシミュレーションが行なえる可能性がある．module のattribute に書く step sequence や ALGOL の CODE の思想を取り入れたことは，このような場合にも役立つものと思われる．入力データとして論理値でなく変数を入れ，論理表現を出力する logic tracer も誤り検出に有効な手法であるが，リストを使って比較的容易に実現できる可能性がある．

converter で行なう仕事は文法の書きかえのほかに同じレベルの記述中での最適化の問題がある．逐次操作を時間的，空間的につめることは状態数最小化の問題に関連し，状態割当ての問題，論理式の最簡化の問題など，オートマトン理論や論理回路論で議論の対象となっている問題がすべてここに含まれる．論理式の最簡化にしても fan-in, fan-out の制約下での多段多出力問題であって，最適解を求めることは困難である．整数線形プログラミング技術の利用も提案されているが，計算時間の点から現状では非実用的である．heuristic な手段により，最終的には人間の判断にまかせるのが実用的であろう．

5. 結　　言

実装設計には計算機がオフラインで利用され，かなり実用化されており，集積回路の導入とも関連して今後ますます盛んになるものと思う．実装設計については一応の解決が得られた段階であるが，改良すべき多くの問題点が残されていることはいうまでもない．これに対しシステム設計，論理設計についてはシミュレーション技術が実用されている段階である．論理設計については，論理回路の作成，最簡化に計算機の利用が考えられ，この方面の研究の一例とオンラインシステムとする際の問題について述べた．実用にいたるまでにはなお解決すべき多くの問題が残されており，基礎的な研究成果の蓄積が必要とされようが，集積回路の採用などによってハードウェアの規模は今後一段と複雑化，大規模化の方向をたどることが予想され，計算機の応用面が拡大することを考えると，論理設計の自動化は設計能率向上のためにも，協同設計を円滑に進める上にもより良い計算機設計のためには是非とも必要な技術と考えられる．実装設計用データ，故障点検出用プログラムの自動作成用データを人手によって作ることは誤りの混入の恐れもあり，はん雑な仕事であって，論理設計自動化システムの出力がそのまま使えるように一貫したシステムを作る必要がある．

ここで述べたシステム記述用語は，システムの仕様を論理設計者に正確に伝える手段としても利用でき，またはソフトウェアの製作者にシステムの仕様を伝える手段としても利用できるものと思う．

故障点検出用プログラムの自動作成の問題は研究もかなり進んでおり，非常に重要な問題であるが，紙面の都合で割愛した．ソフトウェアの自動作成の問題はハードウェアの自動設計に匹敵する重要な問題であり，ソフトウェア作成の人・時の短縮や時間遅れの解決に役立つものと思われ，今後に残された大きな問題である．

終りに本文の作成に当たり，いろいろ御討論いただいた岡田康行氏に深甚なる謝意を表する．

文献紹介

(1) 高島：計算機の設計自動化，信学誌，**50**，4，p. 590.（昭和42年4月）

(2) M. A. Breuer: General Survey of Design Automation of Digital Computers, Proc. of IE^3 **54**, 12, p. 1708. (Dec. 1966)
（設算機の自動設計に関連する文献約 300 のリストがついている）

（昭和42年9月19日受付）