報　告

電子装置のCAD (2)†

—テスト・パターン設計と故障診断の最近の話題—

電子装置設計技術研究委員会

1. まえがき

論理装置のLSI化に伴い，論理回路の故障診断は，重要さを増している反面，ますます，複雑，困難なものになって来ている．本章では，CADの中で大きな位置を占めるテストパターン設計と故障診断について，その概要と最近の話題の幾つかについて紹介する．

論理回路の故障診断は，基本的には，装置にテストパターンを入力し，それに対する正常な出力と故障が生じた時の出力を比較することにより，故障の検出と故障位置の指摘を行おうとするものであり，通常，次のような手順からなっている．

(1)　検出すべき故障の集合を決める．

(2)　テストパターンの集合を求める（テストパターンの生成）．

(3)　一つのテストパターンに対して，検出できる故障の集合を見い出す(テストパターンの検証)．

(4)　テストパターンの最適化を行い，故障検出および故障位置の為の故障診断用資料を作成する．

(5)　テスタなどを用いて，被検査回路にテストパターンを印加して，故障診断を行う．

実際には，上記(2)，(3)の絡みにより，実現方法として，各種の手法が存在する．

論理回路に発生する故障には多くの種類があるが，実際に故障診断の対象とされているのは，単一固定故障が大部分である．テストパターンの自動生成の手法には，乱数および各種アルゴリズム(一次元活性化法，Dアルゴリズム，ブール微分など)が用いられており，それぞれ利害得失がある[1]．現在，使用されているものは，Dアルゴリズムを中心として，それに人手や乱数を併用しているものが多い．テストパターンの検証には，ソフトウェアにより故障シミュレータを用いて行うものと，ハードウェアによるシミュレーションで行うものとがある．故障シミュレータには，並列，ディダクティブ，同時シミュレーションなどの手法が用いられている．故障位置指摘に対しては目的に応じて，機能単位，ブロック単位，ゲート単位などの区別があるが，LSIに関しては，ゲート単位の指摘は一般的に，困難になって来ている．

故障診断については今までに多くの成書[1)-4)]や，報告書[5)-8)]等が出されているので，詳細についてはそれらを参照されたい．　　(向殿政男：明大)

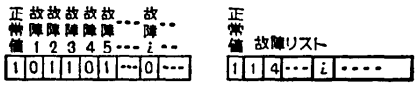
† Computer Aided Design of Electronic Equipments (2) —The Recent Topics of Test Pattern Generation and Fault Diagnosis—by Research Committee on The Design Technology of Electronic Equipments

2. 故障シミュレーション

2.1 ディダクティブシミュレーション

論理装置の故障動作を調べる場合（例えば，テストパターンの検証），正常回路と n-1 種の故障回路の論理値を，ホスト計算機の1ワードのビット位置と対応づけて各信号線の状態表を作り，シミュレートする並列故障シミュレータが一般的であった．Armstrong[11)]によって提案され，その後実用化されたディダクティブシミュレータは，並列故障シミュレータと比べ，大規模故障をシミュレートするのに適している，と報告されている[14),15)]．

ディダクティブシミュレータでは，各信号線に対し，正常値と故障によってその信号値を誤らせる故障名（故障リスト）とが記憶される（図-1）．実行時，素子の入力信号線の正常値と故障リストから，出力信号線の正常値と故障リストが演算される．素子の出力信号線の故障リストを求める式は，故障リスト伝搬式と呼ばれ，入力故障リストの集合演算式で表わされる．図-2



(a) Status table in Parallel Simulator　　(b) Faults list in Deductive Simulator

図-1　故障リスト

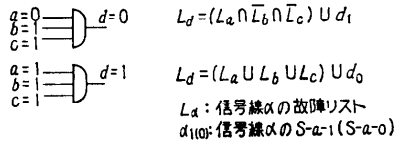
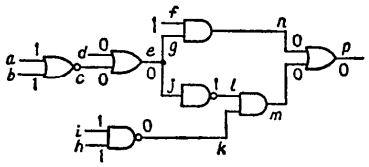


図-2　素子の評価



(a) Simple Circuit
$L_a=\{a_0\}$, $L_b=\{b_0\}$, $L_c=\{L_a\cap L_b\}\cup\{c_1\}=\{c_1\}$
$L_d=\{d_1\}$, $L_e=L_d\cup L_c\cup\{e_1\}=\{c_1,d_1,e_1\}$
$L_f=\{f_0\}$, $L_g=L_e\cup\{g_1\}=\{c_1,d_1,e_1,g_1\}$
$L_h=\{h_0\}$, $L_i=\{i_0\}$, $L_j=L_e\cup\{j_1\}=\{c_1,d_1,e_1,j_1\}$
$L_k=L_i\cup L_h\cup\{k_1\}=\{i_0,h_0,k_1\}$
$L_l=L_j\cup\{l_0\}=\{c_1,d_1,e_1,j_1,l_0\}$
$L_m=\{L_k\cap\bar{L}_l\}\cup\{m_1\}=\{i_0,h_0,k_1,m_1\}$
$L_n=\{L_g\cap\bar{L}_f\}\cup\{n_1\}=\{c_1,d_1,e_1,g_1,n_1\}$
$L_p=L_n\cup L_m\cup\{p_1\}=\{c_1,d_1,e_1,g_1,n_1,i_0,h_0,k_1,m_1,p_1\}$.
(b) Apply ($a=1$, $b=1$, $d=0$, $f=1$, $h=1$, $i=1$)
h: 1→0; $L_h=\{h_1\}$,
k: 0→1, $L_k=(L_h\cap\bar{L}_i)\cup k_0=\{h_1,k_0\}$
m: 0→1, $L_m=(L_k\cup L_l)\cup m_0=\{h_1,c_1,d_1,e_1,j_1,l_0,k_0,m_0\}$
p: 0→1, $L_p=(L_m\cap\bar{L}_n)\cup p_0=\{h_1,k_0,m_0,j_1,l_0,p_0\}$.
(c) Change (h: 1→0)

図-3　ディダクティブ・シミュレーションの例

に示すように，故障リスト伝搬式は，素子の論理機能と入力論理値によっても異なる．

小規模回路上での実行を**図-3** に示す．この結果，出力端子 p を観測することによって，第1入力パターンで故障 (c_1, d_1, e_1, g_1, i_0, h_0, k_1, m_1, p_1) が，第2入力パターンで故障 (h_1, k_0, m_0, j_1, l_0, p_0) が検出できることが確認できる．

順序回路では，フィードバックループを介して，故障 α_0 (α_1) が α 自身に伝搬する場合の配慮が必要である．そのとき，α の正常値が 0 (1) であれば，α_0 (α_1) によって α が 1(0) になるという矛盾が生じるため，$\alpha_0(\alpha_1)$ を α の故障リストから削除せねばならない．FF の故障リスト伝搬式は，かなり複雑なものになるが[2]，FF 自身の故障を無視すれば，入力論理値に依らないで次式となる (SR ラッチは簡単化できない)．

$$L_Y=L_D \quad \cdots\cdots (D\text{-FF})$$
$$L_Y=L_T\oplus L_y \quad \cdots\cdots (T\text{-FF})$$
$$L_Y=(L_J\cap\bar{L}_y)\cup(L_K\cap L_y) \quad \cdots\cdots (JK\text{-FF})$$

ここで L_y は FF の出力信号に対する演算前の故障リストである．

ディダクティブシミュレータの大規模な運用結果が文献13)で報告されている．(安藤　宏：沖電気(株))

2.2　同時シミュレーション

同時シミュレーション (Concurrent Simulation)[17),18)]は，正常回路と故障回路とが同じ入力系列に対して通常ほとんど同じ動作をするという性質に着目して考案された手法で，正常回路のシミュレーションを基本とし，故障回路の動作が正常回路のそれと異なる時点でのみ，故障回路のシミュレーションを正常回路のシミュレーションと“同時に”実行するところから同時シミュレーションと呼ばれる．原理的にはワン・パス・シミュレーション法であるところは前節のディダクティブ法と同様であるが，故障回路も実際にシミュレートされるところが重要な違いである．以下例によってこの手法の概要を説明する．

図-4 のごとく，3個のゲート A, B, C で構成された部分回路を考える．この部分回路を含む対象回路が故障していない場合の，ある時点におけるゲート A, B, C の入力信号 (正常信号) がそれぞれ (1, 1, 0), (0, 1, 1), (0, 0) であったと仮定し，故障 m が存在する場合には A の入力 a_1 が 0 に，また故障 n が存在する場合には a_3 が 1 に縮退すると仮定する．この時故障 m に対応して入力 a_1 を 0 に縮退させた故障ゲート A(m) を，また故障 n に対応して入力 a_3 を 1 に縮退させた故障ゲート A(n) を生成し，正常ゲート A ともどもシミュレートする．故障ゲート A(m) の出力は正常ゲート A の出力と同一であるが，A(n) の出力は異なるため，故障 n に対応して故障ゲート B(n)，および C(n) が生成される．このように一時的に故障ゲートを生成しながら，正常回路については回路 A-B-C が，故障 m については回路 A(m)-B-C が，そして故障 n については回路 A(n)-B(n)-C(n) がシミュレートされる．

故障ゲート格納に必要な記憶容量は，ディダクティブ法における故障リスト格納に要する記憶容量よりも大きく[2]，かつディダクティブの場合と同様，使用量がシミュレーションの進行につれダイナミックに変化し，事前に予測することが困難であるという欠点が予

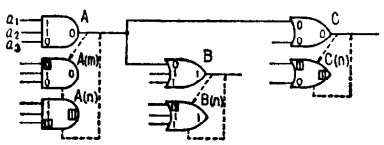


図-4　同時シミュレーション法

想されるものの，一方でディダクティブ法が集合演算という複雑な演算法を用いるのに対し，同時法では故障ゲートを生成して独立に処理しているため，正常回路のシミュレーションに使用できるあらゆる手法がそのまま故障シミュレーションにも使用でき，テーブル索引等を用いた高速シミュレーションが可能であること，更には多値シミュレーション，複雑な遅延時間モデル，機能モデル素子の取り扱いが容易といった他の手法では得がたい利点があり，近年急速に注目を集めているが[19]，いまだ実施結果の報告例[20]が少なく，最終的な評価はディダクティブ法ともども今後の課題であろう．（村井真一：三菱電機(株)）

2.3 機能ブロック故障シミュレータの実施例

IC カードのテストパターン生成等に使用する故障シミュレータでは，高集積 IC の回路モデル作成の簡単化とシミュレーション時間の短縮が課題である．これを解決するには，MSI やメモリ素子を機能ブロックのまま処理することが有効である．以下に機能ブロック故障シミュレータの一実施例を述べる．

2.3.1 対象故障

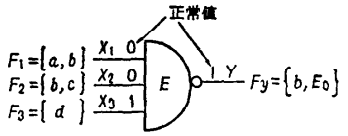
対象とする故障は，素子の入出力縮退故障で，かつ単一故障とする．機能ブロックの故障を内部ゲートの縮退故障として扱うと，機能ブロックの入出力縮退故障では表わせないものもあるが，MSI 程度の機能ブロックでは，入出力端子の縮退故障のみ扱うこととしても実用上問題はないと考えられる．

2.3.2 故障シミュレーションの手法

本故障シミュレーションの基本的手法には，ディダクティブ法の基礎となっている故障リスト伝搬法を用いている．その中の故障リスト演算については，ディダクティブ法が機能ブロックに対して実用的でないため，同時シミュレーション法の概念を用いて機能ブロックの処理を可能にしている．

故障リスト伝搬法では，回路の各素子は正常論理値と故障リストをもつ．故障シミュレータは，論理値の演算とともに素子の入力に伝搬してきた故障リストの中から，その素子においても検出可能な故障を求める故障リスト演算を行う．

図-5 に故障リスト伝搬の例を示す．この例では a, b, c, d のうち Y において検出可能な故障は，Y を 0 に変える故障，即ち入力 X_1, X_2 を同時に 1 に変える故障であるから b がこれに相当する．したがって故障リスト演算の結果は，b と E 自身の出力 0 故障を加えたものであり，$F_y=\{b, E_0\}$ となる．



a, b, c, d: 前段以前の素子の故障
F_1, F_2, F_3: 入力 X_1, X_2, X_3 に伝搬してきた故障リスト
F_y: 出力 Y における故障リスト
E_0: E の出力の 0 縮退故障

図-5 故障リスト伝搬の例

本故障シミュレータでは，故障リスト演算法として，ゲートに対してはディダクティブ集合演算法を，また機能ブロックに対しては，同時シミュレーション法の概念を用いた繰返し故障値シミュレーションによる方法を採用している．

（1） 集合演算法

ゲートに対しては，出力故障リストは入力故障リストの集合関数として表わされ，シミュレータはこの関数を決定し，その演算を行う．図-5 の例では

$F_y(E_0$ を除く$)=F_1 \cap F_2 \cap \bar{F}_3=\{b\}$

しかし，機能ブロックに対しては，このような集合演算式を求めることが実際上不可能であるため，この方法は適用できない．

（2） 繰返し故障値シミュレーション法

機能ブロックに対しては，次の手順で出力リストを求める．

① 機能ブロックの入力に伝搬してきた故障の中から 1 個を取り出し，その故障が含まれる入力端子には故障値を，その他の入力には正常値をセットし，論理演算を行い出力値を求める．

② 求められた故障出力値を正常出力値と比較し，異なっていれば，検出可能故障として出力故障リストに加える．

以上の処理を入力に伝搬してきた全故障に対して，繰返せば求める出力故障リストが得られる．図-5 の例では，入力故障リスト中の b を取り出した時，出力値は 0 となり，正常値 1 と異なるため，伝搬故障として b が求まる．

2.3.3 適用結果

本故障シミュレータによる IC カードのテストパターン生成への適用例を表-1 に示す．シミュレーション時間は，ゲートレベル処理に比して，20% 程度短縮された．故障シミュレーションでは，機能ブロックの処理の効率化が重要であり，その手法としては，同時シミュレーション法が有効であると考えられる．

表-1 適用例

ICカード	素子数	パターン数	故障数	検出率(%)	CPU タイム
A	204 (25)	35	445	94.8	58″
B	289 (9)	100	661	94.7	5′34″
C	199 (40)	151	1,372	92.2	8′45″

・ホストマシンは IBM 370/158 相当
・素子数の ( ) は機能ブロック数を示す
・テストパターンは乱数および人手作成

(湯浅克男：富士通(株))

3. ランダムおよびコンパクトテスト法

3.1 ランダム・テスト法

論理回路の機能を検査して，故障を検出するためには，テストパターンの系列を生成する必要がある．論理回路にソフトウェア的に発生した乱数をテストパターンとして与えて故障を検出しようとする試みは古典的な手法であり，乱数の発生および故障シミュレーションを CAD 的（ソフトウェア的）に実行しようとするものである．

従来のソフトウェア的手法では，計算処理時間の制約から，生成されるテストパターンの数はせいぜい数百個から数千個であり，検査の完全性を目指す立場から見て故障検出率にばらつきが大きく，問題点が指摘されていた．

これらの問題を解決するために最近新たな工夫が試みられている．その手法はソフトウェア的と言うよりもむしろハードウェア的な色合いが濃い．手法の要点を以下に述べる．

（1） 乱数発生器による乱数の発生

乱数を発生するハードウェアによって乱数を発生する．乱数とは0と1の発生する確率がいずれも 0.5 で等しいことを意味する．この乱数発生に際して1の発生頻度を例えば 0.25（1と0の発生頻度の比率が 1：3）として，重みづけを与えることもできる．

（2） ハードウェアによる論理シミュレーション

乱数発生器の出力をテストパターンとして採用して，予め正常な論理機能の確認された回路に与えてその出力をテスタで吸い上げる．このテストパターンは数百万個に及ぶこともある．

（3） テストパターンの出力の圧縮

従来の方法では，テスタにおいて被検査回路にテストパターンを1個与えるごとに，被検査回路からの出力を予め計算された期待出力と比較照合する．一回でも一致しないことが起ると，故障が検出されたことになる．これに対して，毎回比較するに要するテスト時間を短縮するため，予め出力パターンの系列を圧縮しておき，テスタによって論理回路を検査する時に同じ方法で被検査回路からの出力を圧縮して，圧縮された結果の値同士を比較照合することによって，比較照合を一度で済まそうとする．しかしこの圧縮により，故障の検出を見逃す可能性が生じてくる．

以下に本手法における種々の工夫および問題点について述べる．

（1） 出力の圧縮の方法

出力端子において，“1”の個数を数える，出力の変化 (0→1, 1→0) の回数を数えるなどの方法がある．この場合は圧縮の比率は，$n \to \log_2 n$ となるので，$n/\log_2 n$ である．データの圧縮により，テスタにおける出力を期待出力と比較照合する回数が，n 回から，$\log_2 n$ ビット長の計数器の値を最後に1回だけ比較照合することに簡略化できるが，その代り故障の検出を見逃す可能性がある．この見逃しの可能性を少なくするには，データ通信で使われている CRC 法が有力である．

（2） 適用可能な回路

組合せ回路に向いている．組合せ回路に於て入力端子からの論理の段数を 10 段，ゲートの入力端子数がすべて2の例で考えてみる．テストパターンの個数を $(2^2)^{10}=2^{20}\simeq 10^6$ 程度にするとゲートのすべての入出力端子の縮退故障を検出することができよう．組合せ回路では，10^6〜10^8 程度のテストパターンによって実用的には十分に高い故障検出率が得られるであろう．

（3） 順序回路への適用

順序回路に適用する場合には，ハザードが生じるのをできるだけ避けるため，引き続くテストパターンにおいて1個の端子においてのみ，値を変化させる，即ち，次に入力値を変化させる入力端子を乱数により決めると言ってよい．

（4） 乱数の重みを与える

入力端子の値を決める場合に乱数の重み（1の発生する確率を 0.5 以外の値とする）を与えることによって故障検出率を高くしようとする試みがある．重みを変えることは容易であるが，適当な重みを選ぶ方法が問題であり，今後の課題である．

（5） 市販のテスタ

Fluke Trendar（東陽通商）を始めとして，数社のテスタメーカから市販されている．本手法は従来の CAD による手法と異なり，困難かつ高価なテスト系列の生成・検証を必要としない点で有力な手法ではあ

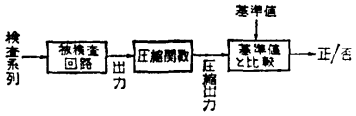


図-6　検査方式（コンパクトテスト法）

るが，故障検出率が不明であり，技術的に解明不十分である．今後の進歩が期待される．

（加納　弘：(株)日立）

3.2　コンパクトテスト法

図-6 に示す検査法について考察する．検査においては，被検査回路（ここでは組合せ回路のみを対象とする）に検査系列を加え，その時に得られる出力系列を正常な場合と比較して，回路に故障があるか否かを判定するが，図のように出力系列を短く圧縮し，それと基準値を比較して正否を判定する方法をコンパクトテスト法という．

そこで，出力系列をうまく圧縮するために，検査入力系列の作り方を工夫することが問題となる．出力系列の圧縮だけに重点を置くと，反って入力系列が長くなるということもあるから，検査入力系列と圧縮出力系列の和を小さくするということを考えなければならない．

2値系列 $R=r_1r_2\cdots r_m$ に対して，次のような圧縮関数について考えよう．

$$C_1(R)=\sum_{i=2}^{m} r_i$$

$$C_2(R)=\sum_{i=2}^{m} r_{i-1}\oplus r_i$$

$$C_3(R)=\sum_{i=2}^{m} \overline{r_{i-1}\oplus r_i}$$

$$C_4(R)=\sum_{i=2}^{m} \overline{r_{i-1}}\cdot r_i$$

$$C_5(R)=\sum_{i=2}^{m} r_{i-1}\cdot \overline{r_i}$$

C_1 は R の中の1の数を，C_2 は R の中で0，1又は1，0の変化の回数，C_3 は R の中で同じ値の続く回数，C_4 は0から1へ変化する回数，C_5 は1から0へ変化する回数，を求める関数である．

さて，テスト系列に含まれるテストで，正常なときその出力値が 1(0) となるテストの集合を $T_1(T_0)$ とする．そして，T_0 の元と T_1 の元を交互に含みかつそのすべての元を含むように並べた系列を α_T とする．α_T の長さは $\max\{n_0, n_1\}$ で最悪の場合には $2n-2$ となる．ここで，n_i は T_i の要素の数で $n=n_0+n_1$ である．初期には，α_T と各種の関数に関するコンパクトテスト法が考えられていたが[25)-27)]，α_T の長さ自身が元の検査系列長 n より長くなり，先に言った意味での圧縮にはなっていない．

そこで，次の β_T，γ_T を導入する．

$$\beta_T=t_{01}t_{02}\cdots t_{0n_0}t_{11}t_{12}\cdots t_{1n_1}$$

$$\gamma_T=t_{01}t_{02}\cdots t_{0n_0}t_{01}t_{11}t_{12}\cdots t_{1n_1}t_{11}$$

ここで，$T_0=\{t_{01}, t_{02}, \cdots, t_{0n_0}\}$，$T_1=\{t_{11}, t_{12}, \cdots, t_{1n_1}\}$ である．$n=n_0+n_1$ とおく．

このように定義すると，β_T の長さは n，γ_T の長さは $n+2$ であり，ほぼ元の検査系列の長さに等しい．

検査入力を β_T として，圧縮関数に (C_1, C_5) を用いると，正常な出力系列に対しては，$C_1(R)=n_1$，$C_5(R)=0$ となり，元の検査入力で検出出来る故障の存在するときには，圧縮関数の値はこれと異なる．即ち，この場合には入力系列長は同じ，圧縮出力として $n_1(<n)$ と0を判定すればよいから，コンパクトテストとなっている[28),29)]．

更に，γ_T を用いる場合を考えると，圧縮関数として，(C_4, C_5) を用いれば，正常のときには $C_4(R)=1$，$C_5(R)=0$ で，故障の存在によって C_4，C_5 のいずれかの値が必ず変化する．この場合には，入力系列はもとのものに比べて2だけ長くなっているが，圧縮出力の基準値が (1, 0) であるから，元の長さ n に比べて $2/n$ に圧縮されたコンパクトテストとなっている[29)]．

このように入力系列を γ_T の形に変形することにより，0から1の変化と，1から0への変化の回数をかぞえるというような簡単な圧縮関数でコンパクトテストが実現出来る．圧縮関数 (C_2, C_4)，(C_2, C_5) についても同じようなことが言える．

以上の議論は，検査回路が単一出力の場合についてであり，多出力回路の場合には上の議論を若干修正，拡張して考えなければならない．この場合には圧縮率は悪くなる．詳しくは文献 29) を参照されたい．

一般の組合せ回路の場合には多出力関数を考えなければならないが，例えばセルマトリックスメモリの場合には，読出される値が0，1の系列であるから，単一出力の場合のコンパクトテスト法を有効に利用することが出来る．

（樹下行三：広島大）

4.　グラフ理論による故障診断

コンピュータなどのシステムの構成要素をグラフの節点，枝に対応させ，グラフ理論的な分析によってシステムの故障診断を行う方法がいくつか提案されてい

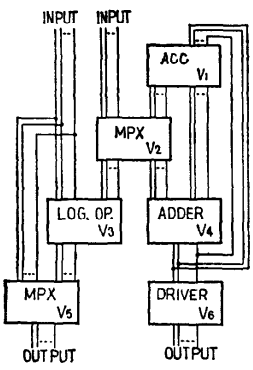
る．このグラフ理論によるシステムのモデル化は，さまざまなレベル（システム・レベル，ゲート・レベルなど）について行うことができ，グラフによる理論的なテスト生成の手法が統一的に適用される可能性がある．

その一例として，大型コンピュータ・システムのレジスタ，アダー，マルチプレクサなどの機能ブロックをグラフの節点に対応させ，これらの機能ブロック間を接続するデータ線（バイト又はワード単位）を1個の枝に対応させる．最近の大型コンピュータでは，ほとんどマイクロ・プログラムによって，各機能ブロック間のデータ伝送，レジスタの制御を行っているので，このようにモデル化したグラフにおいてそれぞれの節点，枝を独立に制御できる．したがって一つのコンピュータの内部をグラフによりモデル化し，さらにこのグラフ上で，データの流れに沿った有向グラフを考えることができる．もし入力，出力がそれぞれ一つ以上存在するときにはデータの入力，出力端を表わす節点を付加する．このような有向グラフを SEC (Single entry-Single exit) グラフと呼び一例を**図-7** に示す．

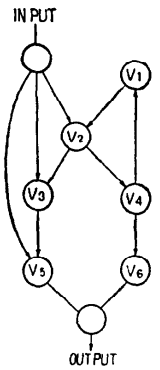
このとき入力から出力まで一つの道を作り，その道（必ずしも一筆書きの条件，すなわちオイラー経路である必要はない）がすべての節点と，すべての枝を通るようにできれば，一つの道のテストだけで故障の検出が可能である．道の途中にレジスタのような記憶素子がある場合には，そこで一時的にデータをホールドさせる．どの機能ブロックに故障が存在するかを決定するためには，仮に n ブロックあれば，$\log_2 n$ 以上のテストする道を作らなければならない．これは必要最小限の道であってすでに設計されたシステムにおいて，実際にはさらに多くの道を必要とし，又いかなる道を作っても分離不可能な故障も存在する．

モデル化されたグラフの上で，もっとも効率よくテストを行うための道の作り方には，現在のところ二つの方法が示されている．その一つは[30]，有向グラフの上で，グラフを張る木を Breadth-First Search 法で作り，この木を用いて入力端から出力端にいたる道を見い出す方法である．このアルゴリズムは節点の数を n とすれば，n 程度の手数でよいので，モデル化さえできればコンピュータを用いて容易に道を求めることができる．

一方ある一つの節点の入力として複数本の有向枝の入力を許すように道を作るアルゴリズムが提案されている[31]．ある特定の節点を取り除いて(Blocking gate)



システムの例



SECグラフ

図-7　システム例　　SEC グラフ

	V_1	V_2	V_3	V_4	V_5	V_6
P_1	0	0	1	0	1	0
P_2	0	0	0	0	1	0
P_3	0	1	1	0	1	0
P_4	0	1	0	1	0	1
P_5	1	1	1	1	1	0
P_6	1	1	0	1	0	1

木によって生成された道

	V_1	V_2	V_3	V_4	V_5	V_6
P_1	0	1	1	1	1	1
P_2	0	0	1	0	1	0
P_3	1	1	0	1	1	1
P_4	0	1	1	0	1	0
P_5	1	1	0	1	0	1
P_6	1	1	1	1	1	0

Blocking gate によって生成された道

図-8

できる道を作り，この道に含まれる節点に関してベクトルを作る．さらに別の節点を取り除いてできる道を作り，同様にこの節点に関してベクトルを求める．これをすべての節点について行い，これらの道のベクトルから行列を求め，この行列を縮退化して，もっとも能率のよいテストの道を求める手法である．図-7 の SEC グラフを例にそれぞれの手法によって与えられるテストのための道の行列を**図-8** に示す．道 P_i に節点が含まれるときを1，含まれないときを0で表わす．

上の二つの方法のいずれも，故障の位置を決定するためには，与えられたテストの道（P_1, P_2, $\cdots P_n$）を用いる．いまテストの道 P_i, P_j について故障であることがわかれば，P_i と P_j に共通に含まれ，しかも P_i, P_j 以外には含まれない節点およびこれらの節点を接続する P_i, P_j の枝に故障があることが示される．

ここに示した方法は，ゲート・レベルのテストにも使える[32]．また最近のように LSI 化されたマイクロ・コンピュータにも適用できる．しかもグラフ化モデルを用いることによって，能率よくテストの生成できることが大きな特徴である．

すでに与えられたシステムについてグラフによるモデル化を行うには，それぞれ独立に枝，節点が制御できるものとして扱っているので，LSI によるマイクロ・

コンピュータのように命令系が一意に定まってしまうときには，必ずしも目的通りの道ができるとは限らない．したがってマイクロ・コンピュータの場合はテスト目的のみの命令系も必要となることが考えられる．

以上ここで述べた手法は各節点とも受動的な能力のみしかないものとした．最近では，ある節点が別の節点をテストする能力（すなわち自己診断能力）を持つような研究[33),34)]も進められており，複数個のプロセッサからなるコンピュータ・システム，あるいはコンピュータ・ネットワークの診断法に応用できる興味深い研究分野の一つである．　（古賀義亮：防衛大）

5. テスト容易化設計法

集積回路の発達にともない論理回路の大規模化，複雑化はますます進み，テストパターン生成およびテストが非常に困難になりつつある．これを解決するためにはテストパターン生成方式，テスト方式とともに，テスト容易化設計法が重要である．特に最近 IBM が LSI に対してスキャンパスを適用した Level Sensitive Scan Design (LSSD) と呼ばれるテスト容易化設計法を発表したことから，一層注目されるようになった[38),42)]．

（1） スキャンパス方式

順序回路のテストを行う際に組み合わせ回路として取り扱うスキャン方式の考え方は，W. Carter 等により IBM System/360 のシステム診断のために導入されたものであるが[35)]，テストデータのスキャンイン（セット），テスト結果のスキャンアウト（読み出し）の回路構成が複雑になる欠点があった．1968 年に小林等により図-9 に示すようなシフトレジスタ構成のスキャンパス方式が考案され[36)]，NEAC 2200/700 に適用された[37)]．IBM が 1977 年に発表した LSSD も構成的にはほぼ同じものである．

図-9 において DFF はシフト機能を有する D タイプフリップフロップ，CLK 1 は平常動作用クロック，CLK 2 はテストデータのスキャンイン／アウト（シフト）のためのクロックである．テストの実施時には，SI よりテストデータが CLK 2 によりスキャンイン（シフト）され，フリップフロップに格納される．CLK 1 が 1 クロック分動作後，テスト結果が CLK 2 により SO へスキャンアウト（シフト）され読み出される．したがってフリップフロップはプライマリ入出力と同じ働きをすることになり，テスタビリティを大幅に改善することが出来る．

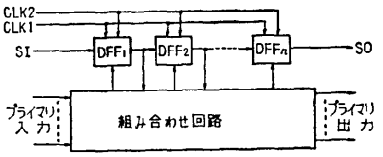


図-9　スキャンパスの構成

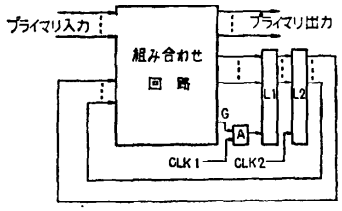


図-10　LSSD (Level Sensitive Scan Design)

IBM の LSSD 方式では，図-10 に示すように DFF 内のマスタ・フリップフロップ（L 1）およびスレーブ・フリップフロップ（L 2）を独立させ，スレーブフリップフロップ（L 2）に与えるクロック（CLK 2）をコントロールすることにより遅延テストも可能な構成としている．

スキャンパスの有効性については，スキャンパスを有する順序回路のテスト自動生成をスキャンパスを用いた場合と用いない場合について比較した例があり，スキャンパスを用いることにより，テスト生成時間が 1/2～1/4 に短縮されたことが報告されている[39)]．

なおシフトレジスタ方式と異なるスキャンパス方式としては UNIVAC のマルチプレクサ方式がある[40)]．

（2） 回路分割方式

テスト生成時間は素子数の 2～2.5 乗に比例するが，回路を分割し分割回路単位にテスト生成を行うことにより，素子数に比例した時間に近づけることが出来る．回路分割の方法としては物理的分割と論理的分割があり，シフトレジスタ方式のスキャンパスを用いている場合は論理的分割を容易に実現出来る．すなわち，1 個あるいは複数個のフリップフロップからファンイン方向へフリップフロップ又はプライマリ入力に到達するまでトレースすることにより分割回路が得られる．この方式により論理的分割を行う場合は，分割回路間の重複を少なくするよう配慮が必要である．

回路分割方式は LSI やカードのテストのみでなく装置やシステムのテストにも適用出来る[41),42)]．

以上，テスト容易化設計法の例として，スキャンパス方式と回路分割方式について述べたが，高密度メモリ素子を含む論理回路のテスト対策としてもテスト容

易化設計法がますます重要になろう.

(山田昭彦: 日電(株))

6. あ と が き

論理装置の LSI 化が, 故障診断にもたらした影響は大きい. 最大の問題は, 取扱うべき回路規模の増大である. 一般に, ゲート数が増加すると, 故障診断の為に必要な計算機の処理時間は, 指数関数的に増加すると考えられる[43]. 現在, 実際に CAD で取り扱っている回路規模は, せいぜい, 5,000 ゲート前後である. この様な回路規模の増大に対する問題を解決する一つの方向は, 本報告でも述べた様なテスト容易化法であり, これは, 今後, 重要な手法になると思われる. 又, LSI に適した故障検出アルゴリズムの開発が急がれると共に, LSI の内部の構造が明確に分からない場合もあり, この為に, 機能素子とゲート素子との両方を含む論理回路に対する有効なテスト方法の開発も望まれる.

(向殿)

参 考 文 献

1) 猪瀬編: コンピュータ・システムの高信頼化, 情報処理学会(昭和 52 年).
2) Breuer, M. A. and Friedman, A. D.: Diagnosis & Reliable design of digital systems, Computer Science Press, Inc. (1976).
3) H. Y. チャン他: ディジタル・システムの故障診断, 産業図書(昭和 46 年).
4) M. A. ブルーア: ディジタル計算機の自動設計, 産業図書(昭和 48 年).
5) 情報処理学会: 計算機設計自動化研究委員会報告(昭和 48 年).
6) 北村, 田代: 故障診断の技術と方法, 電子通信学会誌, Vol. 57, No. 10 (昭和 49 年).
7) 倉地: コンピュータの設計自動化(4), 情報処理学会誌, Vol. 18, No. 9 (昭和 52 年).
8) Bennetts, R. G. and Scott, R. V.: Recent Developments in the theory and practice of testable logic design, Computer, Vol. 9, No. 6 (1976).
9) Ibarra, O. H. and Sahni, S. K.: Polynomially complete fault detection problems, IEEE Trans. Vol. C-24 (March 1975).
10) Chang, H. Y. 他: Deductive techniques for simulating logic circuits, Computer Vol. 8, pp. 52-59 (March 1975).
11) Armstrong, D. B.: A deductive method for simulating faults, IEEE Trans. C-21, pp. 464-471 (1972).
12) Chappell, S. G. 他: Logic circuit simulators, BSTJ, Vol. 53, pp. 1451-1476 (1974).
13) Buther, T. T. 他: Application to switching system development, BSTJ, Vol. 53, pp. 1535-1555 (1974).
14) Chang, H. Y. 他: Comparison of parallel and deductive simulation techniques, IEEE Trans. C-23, pp. 1132-1138 (1974).
15) 村上, 天野: 凝似コーディングによる故障シミュレーション手法のパフォーマンス比較, 本委員会夏期シンポジウム (1978).
16) Chappell, S. G. 他: Functional simulation in LAMP system, 13-th DA Conf. pp. 42-47 (1976).
17) Ulrich, E.G. and Baker, T.: The Concurrent Simulation of Nearly Identical Digital Networks, Proc. 10th Design Automation Workshop, pp. 145-150 (1973).
18) Ulrich, E. G. Baker, T. and Williams, L. R.: Fault Test Analysis Techniques based on Logic Simulation, Proc. 9 th Design Automation Workshop, pp. 111-115 (1972).
19) Abramovici, M. Breuer M.A. and Kumar K.: Concurrent Fault Simulation and Functional Level Modeling, Proc. 14th Design Automation Conference, pp. 128-137 (1977).
20) Schuler, D. M. and Cleghorn, R. K.: An Efficient Method of Fault Simulation for Digital Circuits Modeled from Boolean Gates and Memories, ibid, pp. 230-238 (1977).
21) 田中善一郎: ロジック回路の故障が手軽にできるコンパクト・テスト法, 日経エレクトロニクス pp. 50-64 (1977. 10. 3).
22) Schnurmann, H.D. et al.: The Weighted Random Test-Pattern Generator, IEEE Trans. on Comp. pp. 695-700 (July 1975).
23) Agrawal, P. et al.: Probabilistic Analysis of Random Test Generation Method for Irredundant Combinational Logic Networks, ibid pp. 691-694.
24) Peterson, W. W. et al.: Error Correcting Codes, second edition, MIT Press (1972).
25) Hayes, J. P.: Transition count testing of combinational logic circuits, IEEE Trans. on Comput., C-25, 6 (June 1976).
26) Hayes, J. P.: Check sum methods for test data compression, J. DA & FTC, Vol. 1, No. 1 (Oct. 1976).
27) Seth, S. C.: Data Compression technique in logic testing: An extention of transition counts, J. DA & FTC, Vol. 1, No. 2 (Feb. 1977).
28) Reddy, S. M.: A note on testing logic circuits by transition counting, IEEE Trans. on Comput., C-26, 3 (March 1977).
29) Fujiwara, H. and Kinoshita, K.: Testing logic circuits with compressed data, 8th-Confer-

ence on Fault Tolerant Computing (June 1978).
30) Carroll, A. B., Kato, M. Koga, Y. and Mine, H.: A Method of Diagnostic Test Generation, AFIPS SJCC, pp. 221-228 (1969).
31) Ramamoorthy, C. V. and Mayeda, W.: Computer Diagnosis Using the Blocking Gate, IEEE Trans. COM, Vol. C-20, 11, pp. 1294-1299 (Nov. '71).
32) Koga, Y. CHen, C. and Naemura, K.: A Method of Test Generation for Test Location Incombinational Logic, AFIPS FJCC, pp. 69-78 (1970).
33) Preparata, F. P. Metze G. and CHien, R.T.: On the Connection Assignment Problem of Diagnosable Systems, IEEE Trans. EC, Vol. EC-16, No. 6, pp. 848-854 (Dec. '67).
34) Russel, J. D. and Kime, C. R.: System Fault Diagnosis: Clousure and Diagnosability with Repair, IEEE Trans. COM, Vol. C-24, No. 11, pp. 1078-1089 (Nov. '75).
35) Carter, W.C. et al.: Design of Serviceability features for the IBM system/360, IBM J. Res. Develop., Vol. 8, pp. 115-126 (1964).
36) 小林 亮他: FLT に適したフリップフロップ回路, 昭 43 電子通信学会全国大会 p. 692 (1968).
37) 宮城嘉男他: NEAC シリーズ 2200 モデル 700 のハードウェア・システム, 電子通信学会計算機研究会 EC 71-3 (1971).
38) Eichelberger, E. B. et al.: A Logic Design Structure for LSI Testability, Proc. 14th Design Automation Conf. pp. 462-468 (1977).
39) Funatsu, S. et al.: Designing Digital Circuits with Easily Testable Consideration, Proc. 1978 Annual Test Conf. pp. 98-102 (1978).
40) Stewart, J. H.: Application of Scan/Set for Error Detection and Diagnostics, Proc. 1978 Annual Test Conf. pp. 152-158 (1978).
41) Yamada, A et al.: Automatic System Level Test Generation and Fault Location for Large Digital Systems, Proc. 15th Design Automation Conf., pp. 347-352 (1978).
42) Bottorff, P. S.: Automatic Test Generation for LSI Chips and Printed Circuit Boards, Proc. 1979 IEEE International Solid-State Circuits Conf. pp. 252-253 (1979).

(昭和 54 年 6 月 11 日受付)