設計自動化20－3
（1984. 2. 21）

原因・結果グラフを用いた ハードウェア機能テスト支援手法

小林　一夫

（日本電信電話公社　武蔵野電気通信研究所）

1　まえがき

電子交換用処理装置のようなハードウェアの開発では、設計された論理や製造された製品と仕様とが一致していることを調べるために、テスト（機能テスト）が行われる。機能テストは、テストプログラムとテストデータで構成され、現在、それらの設計に、かなりのコストを要している。今後、ＶＬＳＩによる装置の小型化やＬＳＩマスク設計の自動化等が進むにつれ、実装設計と製造のコストが更に減少する可能性がある。そのため、相対的にこのテストコストの比重が増大する。さらに、ＶＬＳＩでは、製造後に見つかった論理誤りの修正には、多大の期間とコストがかかる。そのため、装置製造前のテストによって、論理誤りを取除いておくことが、従来以上に必要となってくる。

このような背景のもとに、テストコストの削減や高品質設計をねらって、次のような研究が行われている。

（ａ）テストプログラム用高水準言語[1],[2] …従来のアセンブラ言語に換えて、テストプログラム専用の高水準言語を開発し、テストプログラムの設計容易化を図る。

（ｂ）論理の自動検証[3]-[5] …記号実行（シンボリック・シミュレーション）や背理法を援用して論理の正当性の証明を自動化する。機能設計段階のような設計の前期にデバックが可能なため、装置製造後のテストに要するコストを大幅に軽減させる可能性を持っている。

しかし、いづれの方法にもテストデータ作成に問題が残されている。

すなわち、（ａ）におけるテストプログラムの入力データや正解データ、および、（ｂ）における記号実行のテストデータや、背理法の〝表明〟は、従前通りハードウェア仕様を人手で解析して作成しなければならない。そのため、テストによって仕様をどこまで調べられるか（テストの網羅率）等は、テストデータの設計者の能力や、対象の複雑さに左右され、工数を費やしてもかならずしも高い網羅率が得られるとは限らない。

ここでは、このような問題を解決するため、テストデータをハードウェア仕様から自動生成する方法を報告する。

2　ハードウェア機能テスト

ハードウェアのテストは、設計された論理、または、製造された製品と仕様とが一致していることを調べる機能テストと、正しく設計、製造された装置に生じた故障箇所を指摘する故障診断とに大別できる（図２．１）。ここ

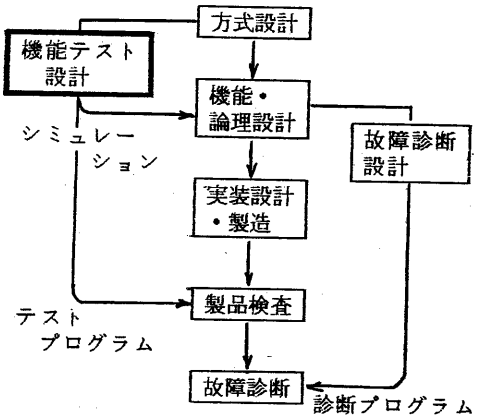


図２．１　設計手順とテスト

では、前者の機能テスト（機能・論理設計段階でのシミュレーションを用いた論理チェック、および、製造後の製品検査）を扱う。

故障診断に使われるテストデータがハードウェア内部の論理構成を解析して選定されるのに対して、機能テストで使われるテストデータは、外部仕様（ハードウェア仕様）のみから選択される。その機能テストデータをハードウェア仕様から直接生成するため、次の3章、4章に示すように、ハードウェア仕様の記述方法と、その仕様を解析してテストデータを生成する方法とを検討した。

3　ハードウェア仕様記述法

方式設計支援のためのハードウェア仕様記法は、次の条件を満たす必要がある。

（1）計算機処理のため、形式的であること。

（2）ソフトウェア設計者や分担して設計する他のハードウェア設計者にわかりやすく、仕様の内容を記述できること。

（3）ハードウェア特有の動作を、明確に表現できること。

一方、ハードウェア仕様は、これまで、文章と、それを補う状態遷移図や動作フローチャート等を用いて表現されてきた。これらの従来の表現法は、内容の理解は容易であるが、計算機処理のための形式性を持たない。そのため、ここでは、ハードウェア仕様を形式的に、かつ判読性が良く記述できるようにするため、次の特徴が活用できるＨＣＰ(Hierarchical and ComPact description) チャート[6]を取上げた。

（ａ）論理を図的に記述できる一種のフローチャート記法であり、形式性と判読性を備えている。

（ｂ）概要と詳細を対応づけながら動作を階層的に記述できる。

（ｃ）処理とデータとの関係を明示で

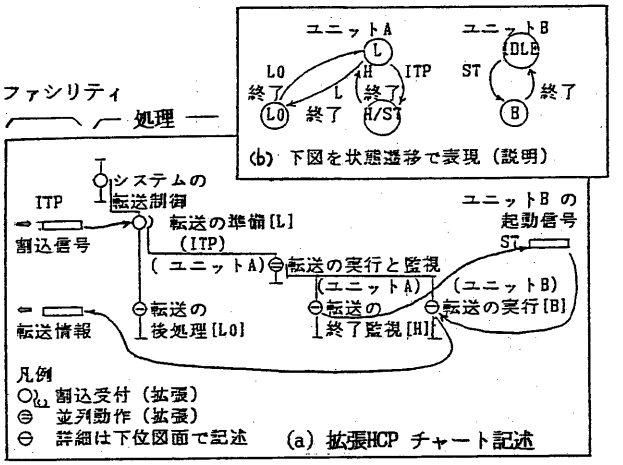


図3．1　ハードウェア仕様記述例

表3．1　拡張ＨＣＰチャートの主な図記号

	表現内容	図記号	備考
ファシリティ	装置＊	○○○	○○○ 装置名
	装置内メモリ、レジスタ等＊	○○○	○○○ メモリ名、レジスタ名等
	入力（出力）データ	○○○	⇨ 入力　⇦ 出力 ⇦⇨ 入出力共用
処理	並列動作＊	A　B	A，Bが並列動作をする
	割込動作＊	(×××)	××× 割込条件
	通常の処理	×××	××× 処理内容
	繰返し処理		
	モジール化		別ページに詳細な処理手続きが表わされる
	分岐		矢印の方向が条件成立時の処理の流れ
	終了		× X階層上に戻る

＊拡張した図記号

きる。

しかし、ＨＣＰチャートは、本来ソフトウェアの記法であり、そのままではハードウェアの記述にかならずしも適さない。そのため、次のような記法を追加して、ハードウェア特有の動作を陽に記述できるようにした[7]。

（１）割込動作：ある処理の実行中に外部からの割込信号があると、実行順序を強制的に切換えて、別な処理を行うことを明確に規定できる。

（２）並列動作：複数の処理が同時に行われることを明示できる。

図３．１（ａ）に拡張したＨＣＰチャートによるハードウェア仕様の記述例を示す。一方の機能（ユニットＡ）からの割込みで、他方の機能（ユニットＢ）が並列動作を始めることを表している。同図（ｂ）は、従来の状態遷移図による、その動作の説明である。表３．１に主な図記号を示す。

４　機能テストデータ生成法

４．１　原因・結果グラフ

仕様を解析して機能テストデータを作成する方法に、ソフトウェアのテスト分野で提案されている原因・結果グラフ手法[8]がある。

この手法では、図４．１に示すような図記号を使って、仕様に含まれる入力（原因）と出力（結果）との間の論理関係を表現する。（例えば、２つの装置の間の制御信号の仕様は図４．２のように表現される。）その後、このグラフを『結果』の方から『原因』に向かって追跡することにより、ある『結果』を調べるために必要な『原因』の組合わせ（テストデータ）が、系統的に作成できる。

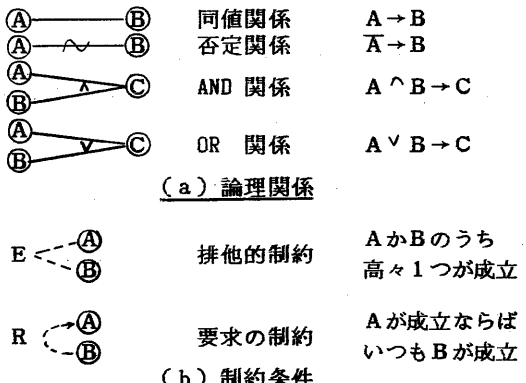


図４．１　原因・結果グラフの記号

しかし、原因・結果グラフ手法では、いかにして元の仕様から正確にグラフを作るかが課題であった[9]。

この課題を解決するため、次のように、仕様から原因・結果グラフを自動的に生成する手順を新たに考案した。

４．２　テストデータ生成手順

図３．１に示したように、仕様は、処理を表すノードのシーケンスと各ノードが参照するファシリティとで構成される。この仕様から原因・結果グラフを作るには、各処理ノードを『原因』と『結果』に分類し、ある『結果』は、どの『原因』の組合わせにより生じるか等の論理条件を付ける必要がある。

そのため、次の①、②の新たな手順を付加えて、図４．３に示す手順で実現した。

①処理ノードとファシリティの参照関係から『原因』と『結果』を抽出する。

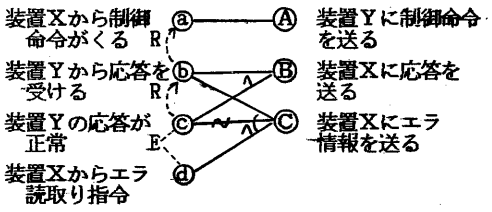


図４．２　原因・結果グラフの例

②各処理ノードの処理順序から論理条件と制約条件を決定する。

その後、既存の手順[8]により、原因・結果グラフからテストケースを生成し（③）、このテストケースを、具体的な実行命令の組合せ（テストデータ）に置換する（④）。

以下では、①、②を中心に自動化処理の手順を述べる。

4．2．1原因と結果の抽出

元の仕様には、階層記述（表3．1）のようにテストデータ生成に不要な情報も含まれる。そのため、まず、テストデータ生成に有効な次の情報を取出す（『原因』と『結果』の抽出）必要がある。

（ア）『原因』または『結果』となる処理ノード。

（イ）『原因』と『結果』の間の因果関係を決める、各処理ノードの順序関係。

このとき、処理ノードの順序は、次のいづれかに分類できる。①ある処理の後、無条件で次の処理が行われる（連続）。②ある処理で条件を判定して、次に行われる処理が決まる（分岐）。これに着目して、『原因』と『結果』の抽出結果を表現するために、図4．4に示すようなツリー構造（原因・結果ツリー）を新たに導入した。

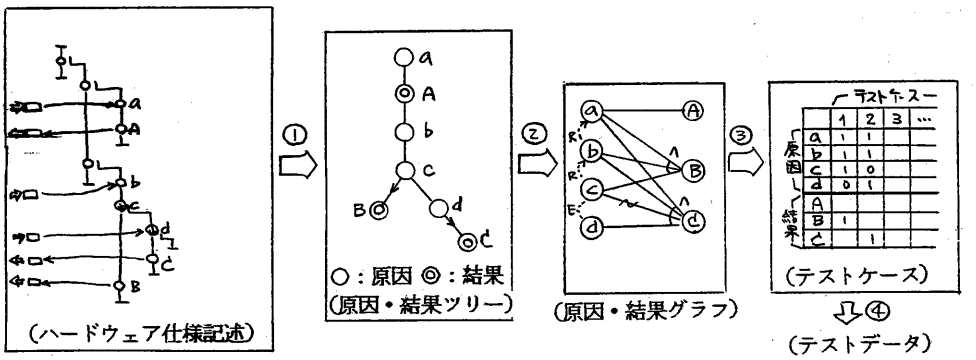


図4．3　処理手順

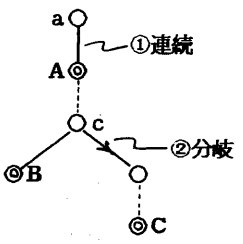


図4．4　原因・結果ツリー

表4．2　原因と結果の対応付けルール

	拡張HCPチャート	原因・結果ツリー
処理の連続	(a) 通常の連続　(b) 階層間に渡る連続	
分岐		
多分岐		
割込み		

注）並列動作は、各動作に分離後、上記のルールを適用する。

例えば、図４．４①『連続』では、ａが『原因』、Ａが『結果』とすると、『結果』Ａは、ａが『原因』となって生じることを表す。一方、②『分岐』は、頂点の『原因』ｃが持つ判定処理が成立すれば、矢印付きの辺の『結果』Ｃが起こることを表す。判定処理が不成立ならば、もう一方の『結果』Ｂが起こることを表す。

このような『原因』と『結果』の抽出と、原因・結果ツリーの表現は、前記の（ア）、（イ）に対応した次の抽出ルールで実現した。

［抽出ルール］

（ア）ファシリティから入力しているノードを原因、ファシリティへ出力しているノードを結果とする。同一ノードが入出力共用の場合は、該当ノードを分割して、それぞれ原因と結果に割振る。

（イ）『原因』と『結果』の間の因果関係は、処理ノードのタイプに応じて表４．２のように対応させる。なお、対応付けに当って、並列動作は、独立の動作に分離して、それぞれの動作ごとにテストケースを作る。

①連続した処理ノードは、原因・結果ツリーの『連続』に対応させる。その際、拡張ＨＣＰチャート上で階層間に渡る処理ノード同志（表４．２（ｂ））でも論理的に連続であれば、同様に対応させる。

②分岐の処理ノードは、原因・結果ツリーの分岐に対応させる。

③多分岐の場合は、拡張ＨＣＰチャート上の分岐条件を新たに二分木構造の頂点として分岐に対応させる。

④割込みは、割込まれる処理ノードを頂点、割込み後実行する処理ノードを判定成立方向とする分岐に対応させる。

４．２．２論理条件と制約条件の解析

原因・結果グラフを作成するには、次に、各『原因』と『結果』の間の論理条件と制約条件を明らかにする必要がある。この論理条件と制約条件の抽出は、前段階で作った『原因』と『結果』との因果関係を解析する次のルールで実現した。抽出した条件は、原因・結果グラフに表現する。

［条件抽出ルール］

①ある結果の前に発生した『原因』は、すべてその『結果』を生じさせる条件（論理積）とする。

②分岐ノードの否定側を経て『結果』が生じる場合、分岐ノードの否定を条件に組み込む。

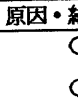
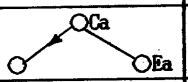
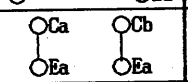
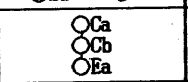
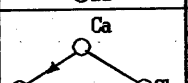
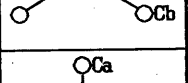
③異なる『原因』の組合せから同一の『結果』が生じる場合、それらのいづれかが『結果』を生じさせる条件（論理和）とする。

④多分岐の各分岐条件を持つノードは、相互に排他とする。

⑤ある『原因』Ａの次に別の『原因』Ｂが必要な場合、ＡにはＢの要求（Ｒ）の制限を付ける。

これらを表４．３にまとめた。

表４．３　条件抽出ルール

		原因・結果ツリー	原因・結果グラフ
論理条件	連続		
	分岐		
	並列		
	縦列		
制約条件	原因の分岐		
	原因の従属		

注）Ci(i=a,b)：原因を表すノード　Ea：結果を表すノード

４．２．３　テストケースの生成

原因・結果グラフから以下の手順でテストケースを作る。

①ある結果を選ぶ。

②原因・結果グラフを逆にたどりながら、この結果を真『１』あるいは偽『０』に設定する原因（図４．５の制約のもとで）すべての組合せを見つける。

③原因の各組合せのためのデシジョンテーブルを作る。

④すべての結果に対し、①～③を繰返す。

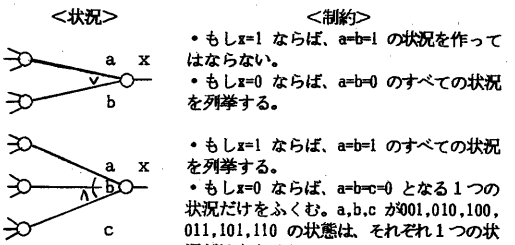


図４．５　グラフを追跡するときに利用される制約規則

５　ＰＲＯＬＯＧによる実現

４章で述べた手順をプログラム化するに当って、今後、各処理に設計者の知識を組込んでより効率的なテストデータ生成ができるようにするため、プロダクションシステム手法の適用を図った。そのため、ＰＲＯＬＯＧを用いて、図５．１の構成で実現した。

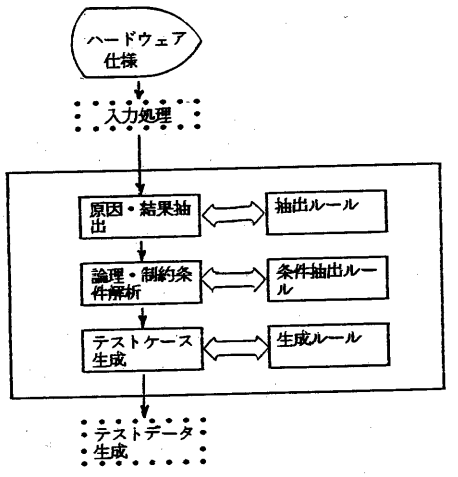


図５．１　プログラム構成

すなわち、グラフィック端末から入力された仕様を、ＰＲＯＬＯＧの『事実』の集合（データデース）に変換する。４章の各処理ルールを、ＰＲＯＬＯＧの『ルール』で構成し、先のデータベースに順次このルールを適用して変換していく。

図５．２に実行例を示す。同図（ａ）がモデル仕様である。ただし、この例では、処理の内容は省略し、代りに英記号（ａ，ｂ，…Ａ，Ｂ…）を用いてある。同図（ｂ）～（ｄ）が入力および処理の途中のデータベースの内容をＰＲＯＬＯＧの構文形式で表したものである。同図（ｅ）が処理結果を表す。

６　手法の評価と今後の課題

（１）評価

今回試作したプログラムは、処理手順の正当性の確認とＰＲＯＬＯＧの適用性の評価を主眼としたプロトタイプであり、約２００ＰＲＯＬＯＧ節で作成した。

処理規模に制限があるため、ノード数２０程度までの約１０種のモデルを用いて、ハードウェア仕様記述から直接テストデータを作成できることが確認できた。

さらに、ノード数と処理時間の関係の一例として図６．１に示す結果が得られた。ＰＲＯＬＯＧ処理系では、パタンマッチを常にデータベースの頭から行うため、全数検索に近い形で目的のデータが得られる。そのため、ノード数の増加に伴い、処理時間が急激に増加する傾向が見られる。

（２）今後の課題

これらの結果をふまえて、今後の課

題としては以下がある。

1）テストデータと仕様記述の量的把握：本手法を実用規模の仕様に適用して、生成されるテストデータ量と記述量を評価する。その結果、必要ならば、テストケース生成時の制約条件の他に対象を限定する適当なルールを設定して、規模の適正化を図る。さらに、ＰＲＯＬＯＧとＬＩＳＰの併用等の処理の高速化手法を採用する。

2）処理ルールの拡充：現在は前後の動作のみ考慮しているが、実用的なテストケースには、並列動作間の競合のような複雑な時間条件の考慮が必要である。このような時間条件を考慮した処理ルールの実現を図る。

7　むすび

ハードウェア仕様設計支援の一環として、機能テスト支援手法の検討結果を述べた。ここでは、①仕様の形式的

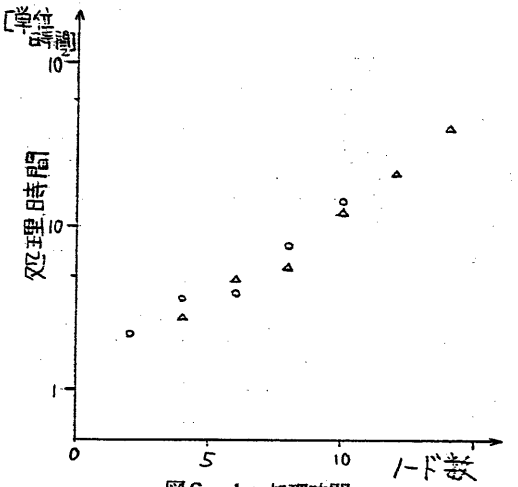


図６．１　処理時間

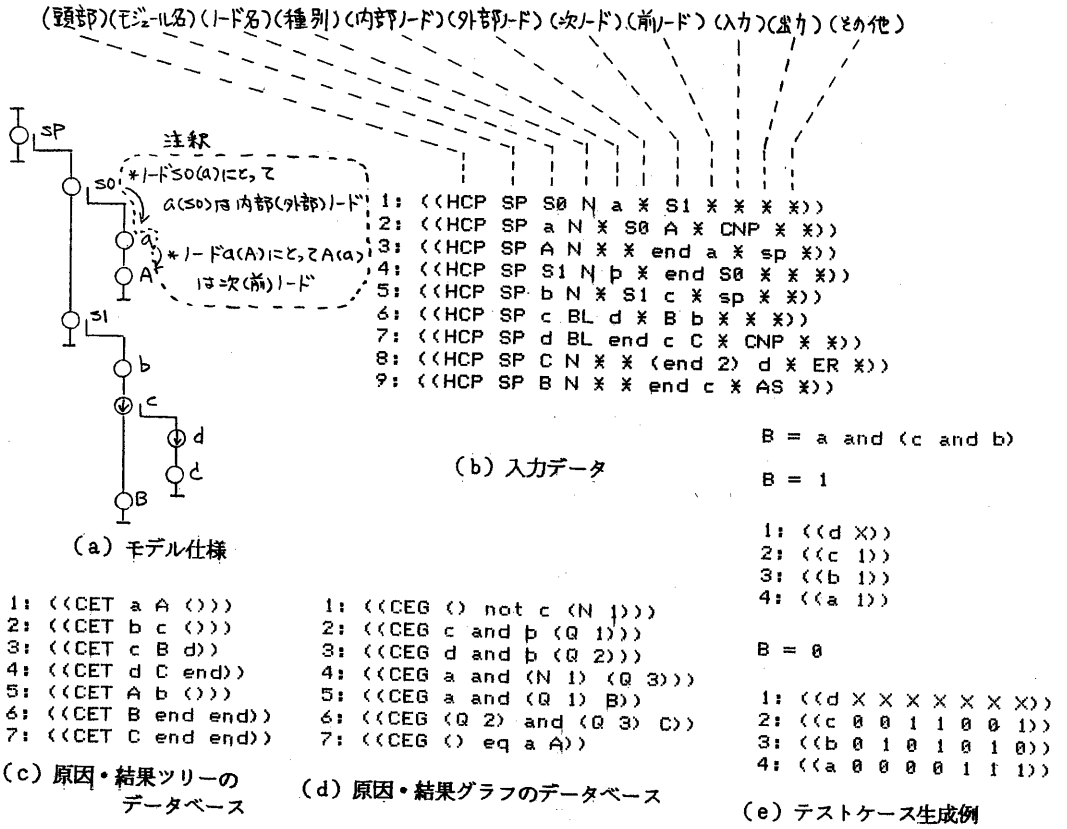


図５．２　実行例

記述のため、ＨＣＰチャートをハードウェア向きに拡張し、②原因・結果グラフ手法を適用して、仕様からテストデータを直接生成する手順を考案した。さらに、③約２００ＰＲＯＬＯＧ節のプログラムにより、この手順を実現し、本手順の正当性とＰＲＯＬＯＧの適用性を明らかにした。

今後の課題は、実用規模のハードウェア仕様で、この手順を評価することと、さらに機能を拡充するため、時間条件の考慮や、テストケースの規模の適正化を図ることである。機能の拡充については、プロダクションシステムの場合と同様に、ルールの追加という形で実現することが考えられる。

最後に、本検討に当り、御指導、御助言を賜わった、処理装置研究室、丹羽昭男室長、松浦洋征調査員、および、室員各位に深謝いたします。また、本プログラムのテストケース生成部の設計は、森田浩氏（大阪大学大学院工学研究科）によるものであり、ここに、感謝の意を表します。

『参考文献』

[1] 田代，津田：テスト用高級言語TPL ；信学会論文誌Vol.J63-D No.11 p923-p930 (1980)

[2] 菊地，村上，平川他：LSI テストプログラムの自動発生；情処第23回全国大会予稿集 1D-7 (1981)

[3] J.A.Darringer:The Application of Program Verification Techniques to Hardware Verification;16th DA Conf. p375-p381 (1979)

[4] 丸山：ハードウェアの機能設計段階における検証；情処論文誌Vol.21 No.5 p493-p503 (1980)

[5] G.L.Smith,R.J.Bahnsen,H.Halliwell:Boolean Comparison of Hardware and Flowcharts ; IBM J.RES.DEVELOP. Vol.26,No.1 p106-p116 (1982)

[6] 花田，佐藤，松本他：コンパクトチャートを用いたプログラム設計法；情処論文誌Vol.22,No.1 p44-p50 (1981)

[7] 小林，若林：ハードウェア仕様記述の一手法；情処第27回全国大会予稿集 5L-4 (1983)

[8] G.J.Myers(長尾他訳)：ソフトウェアテストの技法；近代科学社 (1980)

[9] 古川，野木，越智：機能テストのためのテスト項目作成手法について；情処ソフトウェア工学研資料16-2 (1980)

[10]斉藤、溝口：知的情報処理の設計；コロナ社(1982)