# テスタビリティ解析プログラムCOAP

河村 匡彦　　平林 莞爾
東京芝浦電気(株) 総合研究所

## あらまし

論理LSIのテスタビリティ解析プログラムCOAPをGoldsteinのSCOAPの手法を参考にして開発した。このCOAPの特徴は、
i) 論理シミュレーション用の接続データをそのまま入力できる,
ii) ライブラリ登録形式のため任意のゲートさらには機能ブロックをも扱える,
iii) 最大10Kノードまでの回路を実用的なコア容量,時間内で扱える,
である。
本稿では、COAPについての概要を述べるとともに、COAPをいくつかの回路に適用した結果について議論する。

## 1. まえがき

論理LSIの高集積化が進んでいる中で、テストの困難性が急激に増大しつつある。そこで、初めからテスト容易化を意図して設計されたLSIは別として、通常の論理LSIでその回路がテストしやすいか否かを設計の段階で簡単に判定することができれば、その判定に基づいていくらかでもテスト容易に改良することが可能となる。

この判定のメジャーとして最もよく用いられるのが、テスタビリティ（テスト容易性）というものであり、このテスタビリティはコントローラビリティ（可制御性）とオブザーバビリティ（可観測性）とによって決定される。コントローラビリティとは外部入力（PIと略す）から回路中のノードを"0"または"1"に制御する難易度を表しており、オブザーバビリティは逆に外部出力（POと略す）から回路中のノードが"0"であるか"1"であるかを観測する難易度を表している。

今般、筆者らはこのコントローラビリティとオブザーバビリティをGoldsteinの提案したSCOAP[1]に基づき計算するプログラムCOAP（Controllability and Observability Analysis Program）を開発したので報告する。

このCOAPの特徴は以下に挙げる3点である。

i) 入力データとして論理シミュレーション用の接続データをそのまま用いる。従って、COAPの解析結果を速やかに論理設計にフィードバックできることになる。
ii) ライブラリ登録形式を採っているため、ユーザが定義すれば任意のゲートのみならず機能ブロックをも扱える。
iii) プログラム編集方式により、常に対象回路に適合した計算ルーチンを発生させているため、10Kノードまでの回路を実用的なコア容量内で扱える。また、計算に要する時間も高々2～3分である。

本稿では、2. でCOAPにおける諸量の定義を簡単な例を用いて示した後、3. で処理フローを説明し、4. ではCOAPを加算器,乗算器等に応用した実

行結果例を示す。5. では、テスタビリティの解釈を4. の結果を基に議論し、更に再収斂回路などへの応用に対する問題を提起する。

## 2. COAPにおける諸量の定義

### 2.1 コントローラビリティ

コントローラビリティには、各ノードを"0"に制御する難易度を表す0-コントローラビリティ（CNTL0と略す）と、"1"に制御する難易度を表す1-コントローラビリティ（CNTL1と略す）とがある。各ノードのコントローラビリティは外部入力（PI）のコントローラビリティをもとに次々に計算されていくことになるが、外部入力のコントローラビリティは次のように定義される。

**定義1**

自由な外部入力 ; $CNTL0 = 1$, $CNTL1 = 1$

0に固定された外部入力（GNDなど）;
$CNTL0 = 0$, $CNTL1 = \infty$

1に固定された外部入力（VDDなど）;
$CNTL0 = \infty$, $CNTL1 = 0$

ノードNの0-コントローラビリティをCNTL0(N)，1-コントローラビリティをCNTL1(N)と表すことにすれば、図1のようなANDゲートの出力ノードDの各コントローラビリティは次のように定義される。

$$\begin{cases} CNTL0(N) = \min(CNTL0(A), CNTL0(B), CNTL0(C)) \\ CNTL1(N) = CNTL1(A) + CNTL1(B) + CNTL1(C) \end{cases}$$

つまり、Dを"0"に制御するには、A，B，Cの少なくとも一つが"0"になればよいから、それぞれのCNTL0の最小値をとる。

また、Dを"1"に制御するためには、ノードA，B，Cのすべてを"1"にしなければならないから、それぞれのCNTL1の総和をとることにする。

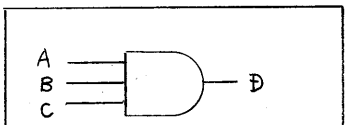


図1 ANDゲート

SCOAPとの違いは、SCOAPの場合、各ノードごとの計算において外部入力からの深さということで1を加えているが、COAPではそれを加えていない。

同様にして、ORゲート，NANDゲート，NORゲート，XOR（排他的論理和）ゲートなどが定義できる。また、MOS回路固有のTG（トランスファゲート），スイッチング機能素子BDD[2]，nビットカウンタなどもそれぞれの機

能を考慮して定義できる。

## 2.2 オブザーバビリティ

オブザーバビリティ（$\bar{O}BSRV$と略す）は、外部出力から回路中のノードの状態を観測するのに要する手数により定量化される。各ノードのオブザーバビリティは外部出力のオブザーバビリティと2.1で求められた各コントローラビリティを用いて計算される。外部出力のオブザーバビリティは次のように定義される。

**定義 2**

すべての外部出力 ； $\bar{O}BSRV = 0$

さて、図2のような回路を例に、ノードA，Bのオブザーバビリティをノード C，Dのオブザーバビリティ $\bar{O}BSRV(C)$、$\bar{O}BSRV(D)$ を用いて表すと、

$$\begin{cases} \bar{O}BSRV(A) = \bar{O}BSRV(C) + CNTL0(B) \\ \bar{O}BSRV(B) = \min(\bar{O}BSRV(C) + CNTL0(A),\ \bar{O}BSRV(D)) \end{cases}$$

となる。ノードBのようにファンアウト数が2以上のゲートの計算では、各ファンアウト先から観測する難易度のうち最小のものをとることにする。この計算においてもSCOAPとの違いは、1を加えていないことである。

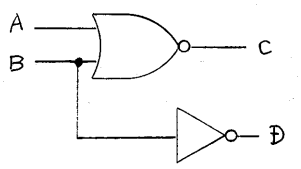



図2　回路例

## 2.3 テスタビリティ

以上のようにして求められたコントローラビリティとオブザーバビリティを用いて、各ノードのテスタビリティ（TESTAと略す）を次のように定義する。

**定義 3**

$$TESTA = (CNTL0 + \bar{O}BSRV) + (CNTL1 + \bar{O}BSRV)$$
$$= CNTL0 + CNTL1 + 2 * \bar{O}BSRV$$

また、以下ではTESTAの全ノード平均値を *overall testability* と呼ぶことにし、回路全体のテスタビリティを代表させている。

## 3. COAPの処理フロー

図3にCOAPのための処理フローを示す。まず、ライブラリ登録プログラムにより、各ゲート種類に対するコントローラビリティとオブザーバビリティの計算式をライブラリに登録しておく。

COAPでは入力データとして論理シミュレーション用接続データを用いるが、次に前処理プログラムにより、後の計算を効率よく行うために、この接続データをライブラリに登録されているゲート種類順に再配列する。

この再配列された接続データを用いて、編集プログラムにより対象回路に適合した計算ルーチン発生させ、コントローラビリティとオブザーバビリティなどの計算を実行する。

コントローラビリティとオブザーバビリティの計算では、初期値としてPIとPOにのみ定義1，2に従う有限値を割り当て、残りのノードの各値を∞とし、計算式に従いすべての値が収束するまで計算を繰り返す。

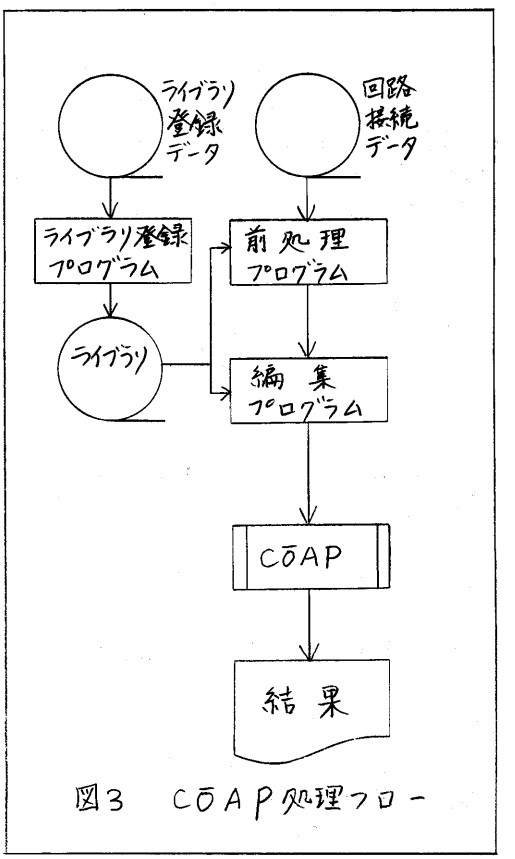


図3　COAP処理フロー

結果はリストに各ノードのCNTL0，CNTL1，OBSRV，TESTAの値が表の形で出力される。また、各値の全ノード平均値も併せて出力される。図4に結果例を示す。

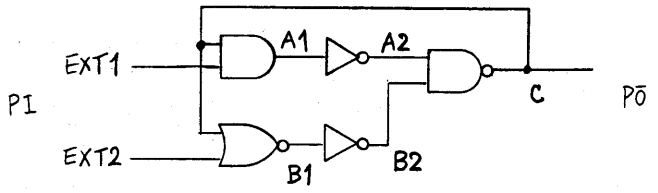
表1はCOAPを回路規模の異なる4種類の回路に応用した場合のCPU時間とコア容量である。CPU時間，コア容量ともにCOAP本体に要するもので、ライブラリ登録，前処理に要するものは含まれていない。

表より、計算時間は回路規模に対してほぼ線形に増大しており、コア容量の方もプログラム編集方式を採っているため比較的小さくてすむ。

表1　COAP実行例

| 回路名 | 総ノード数 | CPU時間(秒) | コア容量(KW) | overall testability |
|---|---|---|---|---|
| A | 466 | 5.0 | 20 | 38 |
| B | 909 | 8.3 | 23 | 40 |
| C | 1795 | 15.5 | 31 | 42 |
| D | 3567 | 33.1 | 45 | 44 |

(ACOS 700による)




```
************************************************************
 TOSHIBA CONTROLLABILITY & OBSERVABILITY ANALYSIS PROGRAM
************************************************************

RESULTS OF COAP

NODE NAME       CNTL0     CNTL1     OBSRV      TESTA

EXT1     ...        1         1         4         10
EXT2     ...        1         1         3          8
A2       ...        4         1         1          7
B2       ...        3         1         1          6
B1       ...        1         3         1          6
C        ...        2         3         0          5
A1       ...        1         4         1          7

NUMBER OF NODES =        7
NUMBER OF PI'S  =        2
NUMBER OF PO'S  =        1

AVERAGE CONTROLLABILITY-0 =       1.86
AVERAGE CONTROLLABILITY-1 =       2.00
AVERAGE OBSERVABILITY     =       1.57

OVERALL TESTABILITY       =       7.00
```

図４　ＣＯＡＰ結果例

## 4. ＣＯＡＰ応用例

表２～４にＣＯＡＰを加算器，乗算器，二進カウンタの各について、それぞれビット数を変化させた回路に適用した結果を示す。それぞれの回路を構成しているゲート種類は、

i) 加算器　；　NOT, AND, OR
ii) 乗算器　；　NOT, AND, OR
iii) 二進カウンタ；　NOT, NAND

である。また、二進カウンタはリセット付きで、最終ビットの１出力のみPO扱いにしている。

また、図５に表２～４の結果を図示する。このグラフを見ると、乗算器，二進カウンタともに overall testability がビット数に対して線形に増大しているのに比べ、加算器は 20 以下で飽和する傾向にある。

表2　加算器への応用結果

| ビット数 | ノード数 | | | CNTL0の平均値 | CNTL1の平均値 | OBSRVの平均値 | overall testability |
|---|---|---|---|---|---|---|---|
| | トータル | PI | PO | | | | |
| 1 | 15 | 3 | 2 | 1.3 | 1.9 | 2.8 | 8.9 |
| 2 | 29 | 5 | 3 | 1.4 | 2.3 | 3.5 | 10.7 |
| 4 | 57 | 9 | 5 | 1.4 | 2.6 | 4.1 | 12.2 |
| 8 | 113 | 17 | 9 | 1.4 | 2.7 | 4.5 | 13.0 |
| 16 | 225 | 33 | 17 | 1.4 | 2.7 | 4.6 | 13.4 |

表3　乗算器への応用結果

| ビット数 | ノード数 | | | CNTL0の平均値 | CNTL1の平均値 | OBSRVの平均値 | overall testability |
|---|---|---|---|---|---|---|---|
| | トータル | PI | PO | | | | |
| 1 | 17 | 4 | 2 | 1.4 | 2.1 | 2.9 | 9.4 |
| 2 | 58 | 6 | 4 | 1.7 | 3.5 | 5.9 | 17.0 |
| 4 | 218 | 10 | 8 | 2.4 | 6.4 | 13.7 | 36.2 |
| 8 | 850 | 18 | 16 | 3.3 | 10.2 | 30.8 | 75.2 |
| 16 | 3362 | 34 | 32 | 4.1 | 13.5 | 65.5 | 148.6 |

表4　二進カウンタへの応用結果

| ビット数 | ノード数 | | | CNTL0の平均値 | CNTL1の平均値 | OBSRVの平均値 | overall testability |
|---|---|---|---|---|---|---|---|
| | トータル | PI | PO | | | | |
| 1 | 12 | 2 | 1 | 2.8 | 1.6 | 4.8 | 13.9 |
| 2 | 22 | 2 | 1 | 4.0 | 2.4 | 9.3 | 24.9 |
| 4 | 42 | 2 | 1 | 6.4 | 4.1 | 18.3 | 47.0 |
| 8 | 82 | 2 | 1 | 11.2 | 7.2 | 36.2 | 90.8 |
| 16 | 162 | 2 | 1 | 20.9 | 13.5 | 72.3 | 179.0 |

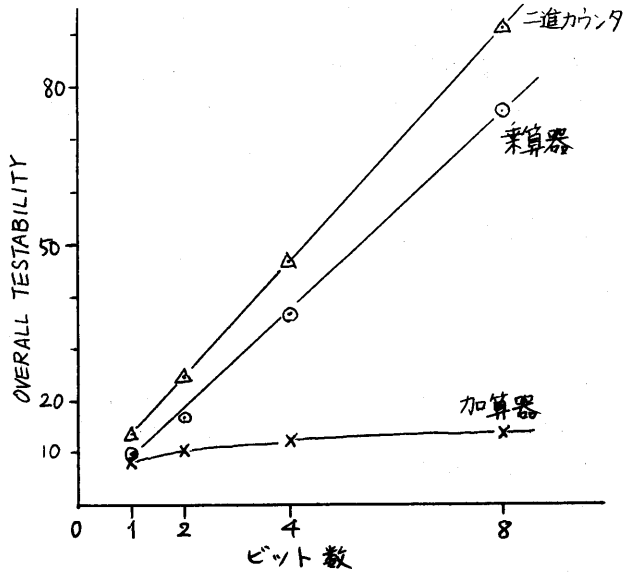



図5　COAP応用結果

## 5. テスタビリティの解釈

4.でCOAPを3種類の回路に適用した結果を示したが、これらの結果をどう解釈し、どう設計にフィードバックするかが一番問題である。結論からいうと、まだ結論めいたものはでていないのが現状であるが、次のような解釈をすることは可能である。

図5の結果を見ると、加算器と乗算器ではビットの増加に対するoverall testabilityの伸びがかなり異なる。これは加算器の場合、ユニットである全加算器を一次元的にcarryチェーンでつないだ回路であり、overall testabilityの増加が小さいのに対し、乗算器の場合は、全加算器のユニットを二次元的に配列したものであり、ビット数に比例してoverall testabilityが大きくなっていくからであろう。また、二進カウンタの場合は、前者2回路と違い順序回路のため、回路規模の割にはoverall testabilityがより大きくなっている。

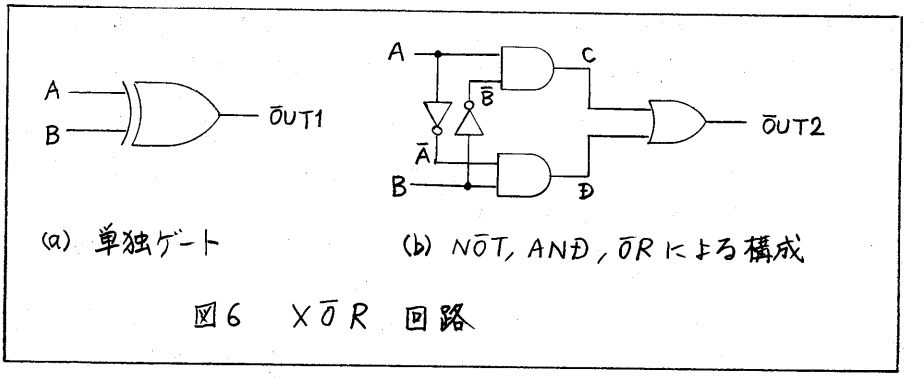
さて次に、同じ機能を有する回路を異なる構成要素で実現した場合、結果がどうなるかを乗算器と二進カウンタを例にとり比べてみる。

乗算器の場合、一つは4.で用いられた回路であり、もう一つはXOR, BDD[2], ANDから構成されるものである。表5に結果を示す。

表5 構成の異なる乗算器のoverall testability

| 構成要素＼ビット数 | 2 | 4 | 8 |
|---|---|---|---|
| NOT, AND, OR | 17.0 | 36.2 | 75.2 |
| XOR, BDD, AND | 13.5 | 81.3 | 3793.3 |

表5を見ると、前者がビット数に対してほぼ線形に増大しているのに対して、後者は指数関数的に増大している。この主な原因は、乗算器回路で多用されているXORにおけるコントローラビリティの計算法にある。




(a) 単独ゲート　(b) NOT, AND, ORによる構成

図6 XOR 回路

XORを図6(a)のように単独ゲートで表すと、出力のCNTL0は、

$$CNTL0(\overline{OUT1}) = \min(CNTL0(A)+CNTL0(B),\ CNTL1(A)+CNTL1(B))$$

となるのに対して、図6(b)のように構成した場合には、

$$CNTL0(\bar{O}UT2) = CNTL0(C) + CNTL0(\bar{D})$$
$$= min(CNTL0(A), CNTL0(\bar{B})) + min(CNTL0(B), CNTL0(\bar{A}))$$
$$= min(CNTL0(A), CNTL1(B)) + min(CNTL0(B), CNTL1(A))$$

となり、値によっては CNTL0(A) + CNTL1(A) または CNTL1(B) + CNTL0(B) という矛盾した計算をしてしまうことになる。従って、このような構成による乗算器の overall testability はビット数が大きくなるほど小さめに出てしまう。XOR などを用いた乗算器の overall testability の方が正しいのはいうまでもない。

次に、二進カウンタについて同様の比較をしてみる。一つはこれも4.で用いた回路であり、もう一つは機能ブロックとして定義したものである。この定義に際しては、各ビットの出力を"0","1"に制御する手数(ステップ数)を基に計算式を作成した。表6に結果を示す。

表6 構成要素の異なる二進カウンタの overall testability

| 構成要素＼ビット数 | 2 | 4 | 8 | 16 |
|---|---|---|---|---|
| NOT, NAND | 24.9 | 47.0 | 90.8 | 179.0 |
| 機能ブロック | 14.5 | 50.7 | 811.6 | 229409.6 |

表6を見ると、乗算器の場合と同様、前者が線形であるのに対して、後者は指数関数的に増大している。この原因は、カウンタ回路のように再収斂のある回路をゲートレベルで記述した場合には、この影響がテスタビリティの計算には全く現れなくなるためである。この場合も後者の方が正しい。

乗算器の場合もXORをNOT, AND, ORで構成した場合には、テスタビリティの計算において再収斂の影響を無視していることになり、乗算器、カウンタとも再収斂を考慮しないとビット数が大きくなるにつれて誤差が非常に大きくなってしまう。

従って、COAPをLSIに適用する場合には、この再収斂に留意しないと値として意味のないものになってしまう危険性がある。しかし、実用上はすべての再収斂を考慮に入れるのは不可能であり、これらの例のように入力側に強い相関のある場合のみ、より大きなブロック単位で記述すれば、回路全体のテスタビリティには大きな影響を与えないように思われる。

最後に、テスタビリティの値そのものの評価であるが、初期化できない回路やPOまで達していないノードなどのテスタビリティが∞となり注意を要するのはいうまでもないが、それ以外の有限な値の評価は難しい。テスタビリティの値が大きいほどテストしにくいということになるが、回路の規模、順序性、機能などによりテスタビリティの値は様々であり、ある値をもってテストしやすいか否かを判断することは困難である。これについては、適用例を増やして経験的に傾向をつかむ他によい解決策はないようである。

## 6. むすび

以上、我々が開発したテスタビリティ解析プログラムCOPについて機能，処理フロー，未解決の問題点などを中心に述べた。

テスタビリティの解釈については、まだまだ適用例が少なく、これという結論は下せないが、COPが手軽に使える有効なツールであることには変わりなく、今後も適用例を増やしていく予定である。

また、overall testability とテスト系列の長さ，テスト系列作成に要する時間等には相関があるといわれており、この点に関する調査，検討が今後の大きな課題である。

謝辞　第4章のCOPの応用例に関しては、豊橋技科大三井修氏の協力を得ましたのでここに感謝します。

## 文献

(1) L.H.Goldstein,"Controllabilty/Obserbability Analysis of Digital Circuits," IEEE Trans., CAS-26, No.9,pp.685-693, 1979.

(2) S.B.Akers,"Binary Decision Diagrams," IEEE Trans., C-27, No.6, pp.509-516, 1978.