故障検出困難度を利用したコンパクトなIDDQテスト集合生成法

渡邉猛夫[†] 篠木剛[†] 林照峯[†]

本論文は，IDDQ テスト法を用いてブリッジ故障を検出するための，コンパクトなテストパターン集合を生成する手法について述べる．すでに提案されている乱数逐次改善法を用いた greedy 生成法[5),6)] では，検出が容易な故障から優先して検出するテストパターンが生成されていくという性質があるが，新手法では，それとは反対に，できるだけ検出が困難な故障から優先して検出されるようにテストパターンを生成することによって，テストパターン数がより少ない IDDQ テスト集合を生成する．評価実験結果により有効性を示す．

A Compact IDDQ Test Set Generation Method Using Hard-to-detect Measure

Isao Watanabe,[†] Tsuyoshi Shinogi[†] and Terumine Hayashi[†]

This paper presents a generation method of compact IDDQ test sets for detecting bridging faults. In the previously proposed method (a greedy method using iterative improvement of random patterns[5),6)]), easy-to-detect faults tend to be detected by the patterns generated earlier in the test set. On the contrary, in the new method proposed in this paper, hard-to-detect faults are detected by the patterns generated earlier in the test set, which leads to decrease the number of test patterns. The experimental results demonstrate its effectiveness.

1. はじめに

IDDQ テスト法[1)] は，CMOS の特徴である信号値の静止時にはほとんど電流が流れないことを利用し，短絡系の故障を，静止時の電源電流を測定すること（IDDQ 測定）により検出するテスト手法である．IDDQ テスト法は，論理テスト法では検出困難な故障も検出できることや，故障あたりのテスト生成計算コストが小さいことなどの特徴を持つ．しかし，IDDQ 測定に時間がかかるため，LSI の検査速度が遅いという問題がある．このため，IDDQ テスト用のテストパターン数を小さくしたい（コンパクトなテスト集合）という要求が強い．

本論文は，組合せ回路を対象とし，論理ゲート回路レベルにおける 2 つのライン間のブリッジ故障の検出に焦点を絞った，IDDQ テスト用のコンパクトなテスト集合の生成法に関するものである．ブリッジ故障とは，2 つのライン間が短絡する故障で，それら 2 つのラインの論理値が互いに背反になるような外部入力パターンを与えれば，IDDQ テスト法により検出できる[1)]（**図 1**）．このようなブリッジ故障検出のためのコンパクトな IDDQ テスト集合の生成法としては，ある補助回路を付与し，deterministic ATPG（Automatic Test Pattern Generation）によってテスト集合を直接生成する手法[2)]，遺伝的アルゴリズムを用いた手法[3)]，与えられたテスト集合を（拡張）必須故障に着目して圧縮する手法[4)] などが，提案されている．さらにその後，乱数逐次改善法を用いた greedy 生成法[5)] およびそれに基づく並列生成システム[6)] が提案され，より良い結果を出している．本手法については詳しくは 2 章で説明するが，できるだけ多くの未検出故障を新しく検出するテストパターンの greedy な生成と故障ドロップを繰り返す手法で，テストパターン生成は，乱数パターンを，より多くの残未検出故障を検出する方向に逐次改善していくことにより行う．

本論文では故障検出困難度を利用した重み付き greedy 生成法およびそれに基づく生成システムを提案する．greedy 生成法[5)] は，できるだけ多くの残未検出故障を検出するテストパターンの greedy な生成を繰り返す手法であるため，検出しやすい故障を多数検出するテストパターンがはじめの方で生成され，検出困難な故障は後回しにされるという性質がある．そのため，後の方で生成される検出困難な故障を検出す

† 三重大学大学院工学研究科
Graduate School of Engineering, Mie University

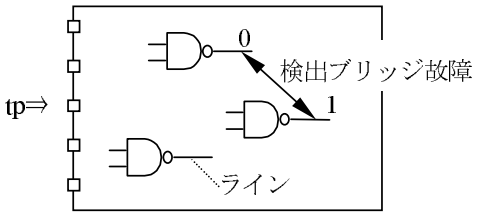

図 1 検出ブリッジ故障
Fig. 1 Detected bridging fault.

るテストパターンは，検出しやすい他の故障を検出する余力があるにもかかわらず，すでに検出しやすい故障は前の方で生成されたパターンによって検出されてしまっているために，後の方のパターンは検出困難な故障を検出するためにしか利用されないという欠点がある．本論文で提案する手法はこの問題点に着目し，greedy 生成法とはまったく反対に，できるだけ検出困難な故障をはじめの方のテストパターンで多く検出するようにし，検出しやすい故障を後回しにするようにして，テスト集合を全体的に見たときのコンパクト性に関する非効率さを解決し，その結果，よりテストパターン数が少ないテスト集合を生成する．

本論文は，次のように構成する．2 章で乱数逐次改善法を用いた greedy 生成法[5] およびそれに基づく生成システム[6] について説明する．3 章で従来手法およびシステム[5),6)] の問題点を明らかにする．4 章で解決手法について述べ，5 章で新しい生成システムを再構築する．6 章では実験結果を示し，7 章でまとめる．

2. greedy 生成法[5)]

手法 5) は，greedy 法（欲張り法）を基本としている．新しく検出する故障数をテストパターンの評価値とし，この評価値ができるだけ大きいテストパターンの生成を繰り返す（以降，これを “greedy 生成法” と呼ぶ）．具体的な手法を以下に示す．

本手法は，できるだけ多くの未検出故障を新しく検出するパターンの greedy な生成と故障ドロップを繰り返す手法で，パターン生成は，乱数パターンを，より多くの残未検出故障を検出する方向に逐次改善していくことにより行う．**図 2** に手続きを示す．関数 main は，乱数逐次改善によりできるだけ多数の未検出故障を新しく検出するテストパターンの greedy な生成と検出故障ドロップを繰り返す．関数乱数逐次改善中のループで，現最良パターン *pattern*（最初は乱数パターンを与える）と，改善ビットポインタ *pin* が指す *pattern* 中の 1 ビット値を反転した対抗パターン *pattern′* を比較し，良い方（未検出故障を数多く検出

```
main(){
  u←故障集合;
  for(i←0;!打ち切り条件*1;i++)  {
    tp[i]←乱数逐次改善();
    新規検出故障を u から削除;
}}
乱数逐次改善(){
  tps←φ; pin←ランダム選択した外部入力ピン番号;
  pattern←乱数パターン; max←nNew_detF(pattern)
  while(1) {
    pattern'←pattern 中の第 pin ビット値を反転;
    temp←nNew_detF(pattern');
    if temp > max then {
        pattern←pattern'; max←temp;
        }
    else {  /* 非改善ステップ */
      if 改善が収束した*2 then {
        tps←tps∪{pattern};
        if  |tps|≧nseeds  then  goto epilog;
        pattern←乱数パターン;
        max←nNew_detF(pattern);
        }}
    pin←pin+'1;  /* i+'j は，(i+j) mod n. */
    }
  epilog:
    tps の中で，
      新検出故障数が最も多いパターンを返す;
}
nNew_detF(pattern) {
  論理シミュレーション(pattern);
  return  pattern が検出した新規検出故障数;
}
```

*1:打ち切り条件は，(1)充分な検出率が得られた，または，(2)これ以上新しく検出故障が見つけられそうもない．
*2:非改善ステップばかりが続いたまま，pin が外部入力ピンを一周したとき．

図 2 greedy 生成法[5)]
Fig. 2 Greedy generation method.

する方）を新しく現最良パターン *pattern* とし，*pin* を 1 つ進める操作を，“収束する” まで繰り返す（“収束する” とは，現最良パターンのいずれの 1 ビットを反転しても，現最良パターンより多くの未検出故障を検出するパターンが得られなくなること）．これで得られる収束パターンは最初に与えた乱数パターンに依存するので，最も多くの数の残未検出故障を検出するようなパターンが得られるとは限らない．このため，多数（*nseeds* 個）の乱数パターンを改善し，最も良い収束パターンを採用する．関数 nNew_detF は，パターンの評価関数であり，論理シミュレーションにより，パターンの新検出故障数を求める．

以上の手続きでは，乱数パターンをもとに改善するため，未検出故障数が少なくなってくると検出パターン

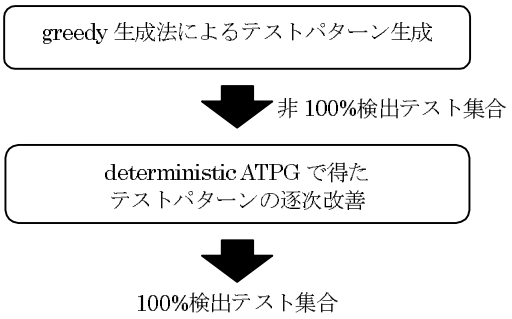

図 3 greedy 生成法に基づく生成システム[6)]
Fig. 3 Test generation system using greedy generation method.

が得にくくなり，複雑な回路や大規模回路に対しては，検出困難な未検出故障（random resistant 故障）が残る．そのため，残った未検出故障を検出するパターンを，deterministic ATPG を用いて求め，このパターンをもとに新検出故障を逃さないように逐次改善する手法を用いることで，最終的に検出効率 100%のコンパクトテスト集合を得る（**図 3**）[6)]．この従来システムを “greedy 生成法に基づく生成システム” と呼ぶことにする．

3. 従来手法[5),6)] の問題点

2 章で紹介した従来法[5),6)] には，次の 3.1，3.2 節に述べるように検出困難な故障の検出を後回しにする性質があり，このことが生成されるテスト集合全体のコンパクト性を下げてしまう要因となっている．

3.1 greedy 生成法[5)] の問題点

故障の中には，多数の外部入力値を必要な値に決めてやらないと検出できないような故障がある．このような故障に対しては，多数の外部入力値が制約を受けるため，検出することができるテストパターンは限られる．したがって，このような故障は乱数パターンによって検出することは容易でなく（検出困難故障），また，1 つのテストパターンで一度に，このような故障を多数検出することは困難である．これらの故障とは反対に，外部入力値のほとんどが “don’t care” でよく，わずかの外部入力値を決めるだけで検出できるような故障がある．ほとんどの外部入力値が不要であるため，このような故障は乱数パターンによって検出しやすいといえる（検出容易故障）．また，このような故障は各々がテストパターンへの制約が小さいので，1 つのテストパターンで多数の異なるこのような故障を検出することができる．

greedy 生成法[5)] では，1 つのテストパターンを生成するにあたって，乱数パターンを 1 ビットずつ変更しながら，新しく検出する故障の数をできるだけ増やしていく．上で述べたようなことから，1 つのパターンの検出故障数を増やすためには，できるだけ検出容易故障を検出する方が有利である．そのため，greedy 生成法においても自然に，検出容易故障から優先されてどんどん取り込まれていくように動作し，検出困難故障は後回しにされる．その結果，生成されたテストパターン集合の中でも，最初の方で生成されたテストパターンとしては，検出しやすい故障を多数検出するものが作られ，後の方で生成されたテストパターンとしては，後回しにされた検出困難な故障を検出するものが作られる．しかし，次の (a)，(b) で述べるように，後の方で生成されたパターンは，検出困難な故障を検出するためだけにしか働いておらず，検出しやすい故障を検出する能力があるにもかかわらず，それが生かされない．

(a) 後の方で生成されたパターンは，残っていた検出困難な故障を検出するために働いている外部入力値と，それ以外の部分，すなわち，新しい故障検出のためには働いていない外部入力値に分けられる．後者は，一般には，さらに別の故障を検出するために使うことができる．しかし，検出困難な故障検出のためにすでに多くの外部入力が使われたため，新しい故障検出のために働いていない外部入力の数は少なく，さらに，未検出故障としては，多数の外部入力値が制約を受ける検出困難な故障しか残っていない．そのため，まだ働いていない外部入力を，新しい故障検出のために有効に使うことができず，有効的に働いていない外部入力値が多い（**図 4**）．

(b) 後の方で生成されたパターン中の，検出困難な故障を検出するために必要な外部入力値は，同時に他の故障も偶然に検出する．しかし，これらの同時偶然検出故障は検出しやすい故障である場合が多いため，すでに前の方で生成されたテストパターンによって検出されていることが多く，しかも前の方のテストパターンが意図的に検出してしまっていることも多い．ここで “テストパターンがある故障 f を意図的に検出する” とは，別の故障を検出するために生成されたテストパターンによって，その故障 f が偶然検出されたものではなく，その故障 f を検出するために，テストパターン中の一部を必要な値に定めたことをいう．検出困難な故障を検出する外部入力値による同時偶然検出故障は，前の方のパターンで意図的に検出する必要はなく，そのために使われた前の方のパターン中の外部入力値は別の故障検出のために使うべきである．

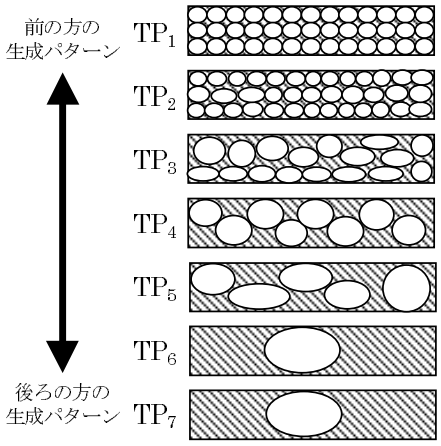

図 4 greedy 生成法によって生成されたテスト集合
Fig. 4 A test set generated by greedy generation method.

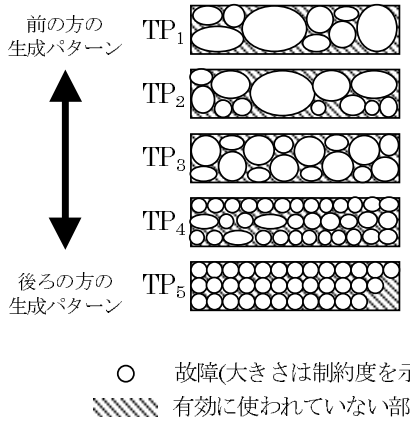

図 5 重み付き greedy 生成法によって生成されるテスト集合
Fig. 5 A test set generated by weighted greedy generation method.

これらの問題点 (a), (b) はともに，はじめの方で生成するテストパターンから検出困難な故障を優先的に検出するテストパターンを生成するようにすれば解決できる．すなわち，(a) については，はじめの方のパターンから，検出困難な故障を優先的に詰め込むようにし，1 つのパターン中に検出困難な故障を詰め込めなくなったらしだいに検出容易な故障も詰め込むようにしていけば，(a) で述べたような最後に検出困難故障だけが残ってしまうようなことはなくなり，働いていない外部入力に関する無駄を少なくできる（**図 5**）（これは，様々な形や大きさの引越し荷物をできるだけ少ない数のダンボール箱に詰め込む問題に似ている）．

さらに，(b) についても，はじめの方のパターンで検出困難故障をできるだけ検出するようにすれば，その検出困難な故障を検出するために必要な外部入力値が同時に偶然検出する故障は，故障ドロップにより残未検出故障リストから除かれるので，それ以降それらの同時偶然検出故障を意図的に検出するパターンは作られない．

3.2 システム構成[6] 上の問題点

greedy 生成法は乱数パターンをベースとした手法であるため，未検出故障数が少なくなってくると検出パターンが得にくくなる．そこで手法 6) では，新しい故障を検出するパターンが生成しにくくなると，残未検出故障を検出するパターンを deterministic ATPG で得，そのパターンを改善して，テストパターンを補足している（図 3）．

しかし，ここで deterministic ATPG を用いて得たテストパターンによって検出される故障はきわめて検出困難な故障である．つまり，乱数逐次改善法によりテストパターンを生成した後で，きわめて検出困難な故障を対象にして，deterministic ATPG を用いてテストパターンを生成することは，上述 3.1 節の greedy 生成法の問題点と同様，検出困難な故障が後回しにされ，効率的ではない．deterministic ATPG で補足されるテストパターンが検出するきわめて検出困難な故障は，3.1 節で述べたように，テスト集合生成の過程で最初の方で検出されるべき故障である．

4. 重み付き greedy 生成法

3.1 節で示したような，残っている検出困難な故障を優先的に検出するテストパターンを greedy 生成法を用いて生成するには，検出困難な残故障を検出するテストパターンを高く評価する必要がある．このために，まず，あらかじめすべての故障の検出の難しさを測定する．検出困難な故障は，多数の乱数パターンを与えてもなかなか検出されないが，検出しやすい故障は何度も検出される．これを利用すれば故障の検出の難しさを評価できる．故障の検出の難しさを “検出困難度” と呼び，以下のように求める．

- 故障の検出困難度
  乱数パターンを多数発生させ，その乱数パターンで故障 f_i を n 回検出したとき，
  $$\text{検出困難度 } (f_i) = 1/n^k$$
  $k : k > 0$ の定数 (経験的に定める)
  （$k = 0$ の場合，検出困難度 $(f_i) = 1$ であり，greedy 生成法での評価方法となる）

パターンの評価値は，新しく検出したすべての故障の検出困難度を総和する．

- パターンの評価値
  パターン P が残未検出故障集合のうちの故障 $\{f_i : i = 1, 2, \cdots, m\}$ を新しく検出したとき，

$$\text{評価値}\ (P) = \sum_{i=1}^{m} \text{検出困難度}\ (f_i)$$

この評価関数を用いて greedy 生成を繰り返せば，図 5 のようなテスト集合を生成することができる．この手法を，“重み付き greedy 生成法” と呼ぶことにする．

5. 重み付き greedy 生成法に基づく生成システム

重み付き greedy 生成法によって 3.1 節の greedy 生成法の問題点は解決することができる．もう 1 つの 3.2 節で述べた問題点（deterministic パターンの能力が有効に利用されていない）についても，deterministic パターンを最初に生成するようシステムを変更することによって解決できる．未検出故障として残るきわめて検出困難な故障は，最初に予測する．

図 6 に新しい生成システムの流れを示す．まずはじめに行われる未検出故障として残る検出漏れ故障の予測には，検出困難度を利用する．すなわち，多数の乱数パターンをそれぞれ，全故障に対して故障シミュレーションし，検出された回数が 0 回かまたはきわめて少なかった故障を対象として deterministic ATPG でパターンを生成する．そして，このパターンを deterministic ATPG で対象となった故障を保持したまま，評価値（重み付き）が大きくなるように改善したテストパターンを作る．その後，残った未検出故障を対象とした重み付き greedy 生成法を実行する．もし，重み付き greedy 生成法の実行後，未検出故障が残った場合は，その故障を最初の deterministic ATPG の対象故障に加え，図 6 の第 2 ブロックからやり直す．

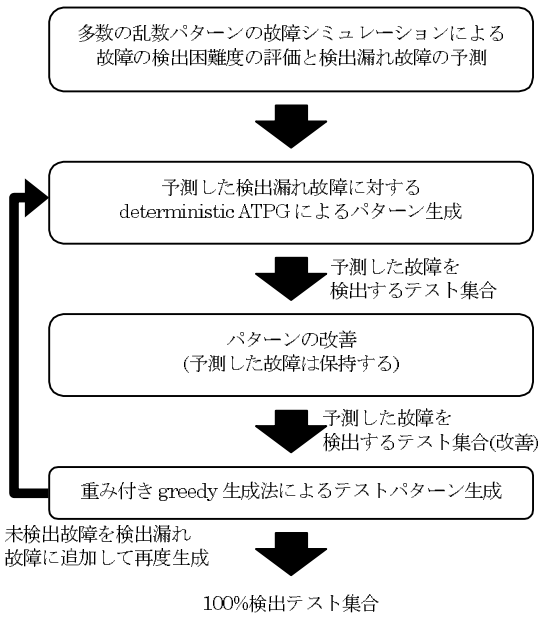

図 6 重み付き greedy 生成法に基づく生成システム
Fig. 6 Test generation system using weighted greedy generation method.

6. 実験結果

重み付き greedy 生成法の性能を評価するために，新しい生成システム（図 6）と greedy 生成法に基づく旧生成システム（図 3）との性能比較実験を行った．本実験では，適当に選択したブリッジ故障を対象故障としている（100,000 個）．greedy 生成法および重み付き greedy 生成法は図 2 の手続きにおいて，$nseeds = 32$；重み付き greedy 生成法における故障の検出困難度評価関数の分母中の $k = 3$；打ち切り条件は，乱数逐次改善により生成されたパターンが新しい故障を 1 つも検出しないということが 100 回連続して発生したときとした．使用した計算機の CPU は Pentium III（850 MHz）である．実験結果は**表 1** のようになった．検出効率はすべて 100%である．

実験結果より，重み付き greedy 生成法は greedy 生成法とほぼ同じ程度の計算時間で小さなテスト集合が生成できている．また，回路規模が大きくなるにつれて重み付き greedy 生成法で生成されたテスト集合がより小さくなっており，重み付き greedy 生成法の有効性が確認できた．

表 1 実験結果
Table 1 Experimental results.

circuit	test set sizes		CPU time [s]	
	greedy	重み付き greedy	greedy	重み付き greedy
c880	15	12	25	85
c1355	62	58	21	38
c2670	18	13	264	261
c3540	29	23	42	67
c5315	13	12	185	231
c6288	15	14	18	50
c7552	21	12	243	212
s5378	30	21	280	314
s9234	42	31	333	373
s13207	52	38	1,346	1,326
s15850	33	21	1,163	881
s35932	9	7	3,508	3,727
s38417	24	11	3,773	4,330
s38584	46	25	4,452	2,604

（ISCAS’89 の回路はフルスキャンを仮定）

7. まとめ

本論文では，ブリッジ故障を検出するコンパクトな

IDDQ テスト集合を生成する重み付き greedy 生成法およびそれに基づく生成システムを提案した．

greedy 生成法[5] では，検出しやすい故障を多数検出するテストパターンから順に作成していき，検出困難な故障は後回しにされるという性質があった．そのため，後の方で生成される検出困難な故障を検出するテストパターンは，他に検出しやすい故障を検出する能力があるにもかかわらず，検出しやすい故障はすでに生成したテストパターンによって検出されてしまっているために，検出しやすい故障を検出する能力が有効に使われていないという欠点がある．そのため，コンパクト性の観点からテスト集合を全体的に見ると，効率を低下させてしまっていた．また，greedy 生成法に基づく生成システム[6] は greedy 生成法で検出漏れとなったきわめて検出困難な故障を検出するテストパターンを deterministic ATPG で後から補足するため，同様の理由で効率を低下させていた．

そこで，検出困難な故障を優先的に検出するテストパターンを生成するための重み付き greedy 生成法とそれに基づく生成システムを新しく提案した．新生成法では，greedy 生成法とはまったく反対に，できるだけ検出困難な故障をはじめの方のテストパターンで検出するようにし，検出しやすい故障を後回しにするようにして，テスト集合を全体的に見たときの非効率性を解決した．さらに，新システムでは，重み付き greedy 生成法で検出漏れになる故障を前もって予測し，この故障を検出するパターンをはじめに deterministic ATPG で生成し，改善することによって非効率性を解決した．

実験結果より，重み付き greedy 生成法は，greedy 生成法に比べ，特に回路規模が大きい回路において，よりコンパクトなテスト集合が生成できることが確認できた．

謝辞 本研究の一部は，財団法人岡三加藤文化振興財団からの研究助成によるものです．同財団に感謝いたします．

参 考 文 献

1) Chakravarty, S. and Thadikaran, P.J.: *Introduction to IDDQ Testing*, Kluwer Academic Publishers (1997).
2) Reddy, R.S., Pomeranz, I., Reddy, S.M. and Kajihara, S.: Compact Test Generation for Bridging Faults under IDDQ Testing, *VLSI Test Symposium*, pp.310–316 (1995).
3) Thadikaran, P. and Chakravarty, S.: Fast Algorithms for Computing IDDQ Tests for Combinational Circuits, *9th International Conference on VLSI Design*, pp.103–106 (1996).
4) Kondo, H. and Cheng, K-T.: An Efficient Compact Test Generator for IDDQ Testing, *Asian Test Symposium*, pp.177–182 (1996).
5) Shinogi, T. and Hayashi, T.: A Simple and Efficient Method for Generating Compact IDDQ Test Set for Bridging Faults, *VLSI Test Symposium*, pp.112–117 (1998).
6) Shinogi, T. and Hayashi, T.: A Parallel Generation System of Compact IDDQ Test Sets for Large Combinational Circuits, *8th Asian Test Symposium*, pp.164–169, (1999).

(平成 12 年 9 月 13 日受付)
(平成 12 年 12 月 1 日採録)

渡邉 猛夫

1977 年生．1998 年三重大学工学部電気電子工学科卒業．現在，同大学院工学研究科修士課程在学中．LSI テスト生成システムの研究に従事．

篠木 剛（正会員）

1954 年生．1977 年東京工業大学理学部情報科学科卒業．1979 年同大学院理工学研究科修士課程修了．同年株式会社富士通研究所入社．1985 年より 1 年間米国オレゴン大学客員研究員．Lisp マシン Facom α，並列推論マシン PIM/p の研究開発に従事．1998 年三重大学大学院工学研究科博士後期課程修了．工学博士．1998 年三重大学工学部講師．1999 年助教授．LSI のテスト生成システムや設計支援システム，並列/分散計算機システム等に興味を持つ．電子情報通信学会，IEEE 各会員．

林 照峯（正会員）

1947 年生．1969 年名古屋大学工学部電気学科卒業．1971 年同大学院工学研究科修士課程修了．同年株式会社日立製作所日立研究所入社．1993 年三重大学工学部電気電子工学科教授．工学博士．LSI の設計自動化，論理回路のテストおよびテスト容易化設計等の研究に従事．電子情報通信学会，IEEE 各会員．