Consiglio Nazionale delle Ricerche

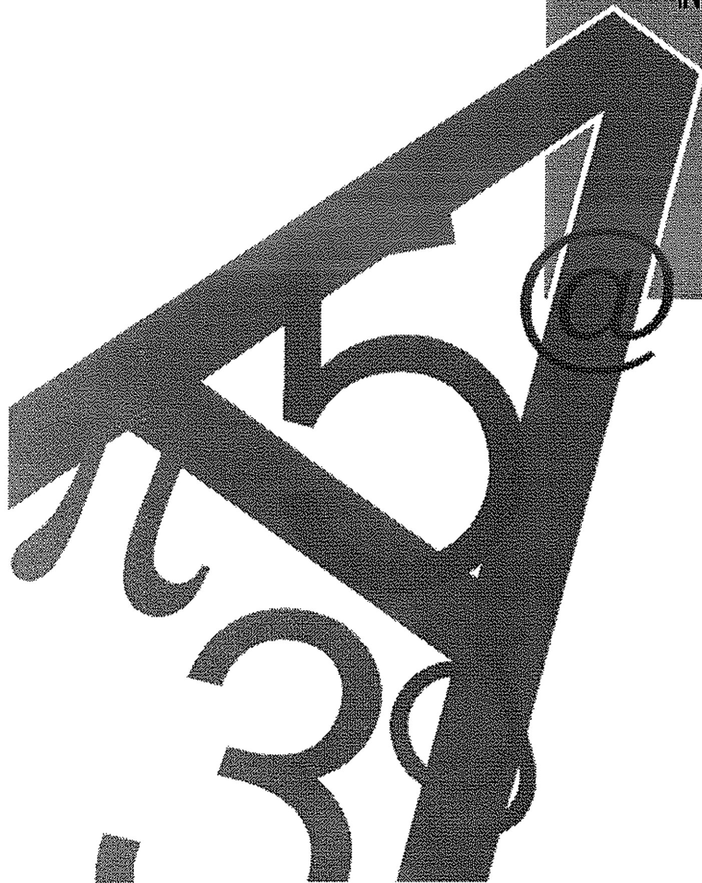
Technical Report

A Comparison-Based Diagnosis Algorithm Tolerating Comparator Faults

Balázs Sallay, Piero Maestrini, Paolo Santi

IEI: B4-26-10-98

I.E.I.
ISTITUTO DI ELABORAZIONE DELLA INFORMAZIONE



Pisa

I.E.I. via S.Maria, 46 - 56126 PISA
Tel + 39 50 593414 Fax + 39 50 554342 E.mail biblio@iei.pi.cnr.it

Consiglio Nazionale delle Ricerche

A Comparison-Based Diagnosis Algorithm Tolerating Comparator Faults

Balázs Sallay, Piero Maestrini, Paolo Santi

IEI: B4-26-10-98

A Comparison-Based Diagnosis Algorithm Tolerating Comparator Faults

Balázs Sallay*, Piero Maestrini, and Paolo Santi
Istituto di Elaborazione dell'Informazione del CNR, Pisa, Italy

Abstract

A promising application of system-level diagnosis is the testing of VLSI chips during the manufacturing process. A comparison-based diagnosis is easier to implement on the wafer than the PMC-conform one. However, existing comparison models essentially overlook the test invalidation due to the physical faults in the comparators. This paper proposes a comparison-based model and a diagnosis algorithm which takes into account the effects of faults that affect the comparators. In order to deal with faults in the comparators, a preliminary comparator test session is included. This session requires the adjacent units to be able to feed the comparator with all needed patterns independently of each other. As shown in the paper, this requirement can be satisfied at a relatively small wafer design cost. The test session can also be extended to handle the faults in the syndrome collection circuitry.

Categories and Subject Descriptors: B.7.m [**Integrated Circuits**]: Miscellaneous; B.8.1 [**Performance and Reliabiltity**]: Reliability, Testing, and Fault-Tolerance

General Terms: Algorithms, Design, Reliability

1 Introduction

System level diagnosis, also called self-diagnosis, has been introduced by Preparata et al. in 1967 [1]. In self-diagnosis, a system composed of several units connected by bidirectional links can be diagnosed using the information provided by *tests.* These tests are performed by the units comprising the system itself, and each of them involves two units, called the *testing* and the *tested* unit. The testing and the tested unit must be adjacent (that is, interconnected by bidirectional links). Essentially, a test is performed as follows:

- the testing unit requests the tested unit to run a test;
- the tested unit returns a result to the testing unit;
- the testing unit compares the actual and the expected results and provides a binary test outcome. The outcome is 0 if the actual and the expected results match (the test passes), 1 otherwise (the test fails).

The set of tests utilized for the purpose of diagnosis is defined by the directed graph $DG = (V, E)$, where V is the set of units and $E = \{(u, v)$ such that unit u tests unit $v\}$. DG is called the *diagnostic graph* of the system. Observe that the tests involving different units can be run simultaneously. The set of all test outcomes is called *syndrome.*

The test results are not necessarily reliable, since testing units themselves may be faulty. Different hypotheses upon the test outcome generated by faulty units lead to different *invalidation rules,* and consequently to different *diagnostic models.* The most widely used diagnostic model is the PMC model introduced in [1], which assumes arbitrary test outcomes for tests performed by faulty units. The invalidation rule of the PMC model is shown in Table 1.

Comparison models for self-diagnosis have also been introduced in literature. In comparison models, tests are performed by comparators which compare the outputs of pairs of units that run the same test. The diagnostic graph corresponding to a comparison model has an undirected edge between a pair of nodes if there exists a comparator between the corresponding units. The outcome of the comparison is 0

*The work of this author was supported by the European Research Consortium for Informatics and Mathematics

testing unit	tested unit	test outcome
fault-free	fault-free	0
fault-free	faulty	1
faulty	fault-free	0 or 1
faulty	faulty	0 or 1

Table 1: Invalidation rule in the PMC model

if the outputs agree, and 1 if they disagree. Again, different assumptions on the behaviour of the faulty units and comparators lead to different diagnostic models. In Malek's model [2] the comparators are implicitly assumed to be fault-free and the comparison outcome is 0 only if both units being compared are fault-free. The model introduced by Chwa and Hakimi [3], which also assumes reliable comparators, allows arbitrary comparison outcomes when both the compared units are faulty. In [4], Maeng and Malek proposed a modification of Malek's model, in which comparisons are performed by the system units themselves: if unit i is adjacent to units j and k, then i may be utilised to compare the outputs of j and k. This requires the system units to be homogeneous and able to perform comparisons, which condition is satisfied when the units are processors.

Rangajaran, Fussel and Malek [5] suggested that system level diagnosis may find application in the testing of VLSI chips on the wafer during the manufacturing process (*wafer-scale testing*). In this case the main goal of the diagnosis is identification of good integrated circuits (ICs) within the wafer, which will be packaged, while the faulty circuits will be discarded. This approach reduces the complexity and cost of the Automated Test Equipment (ATE) needed to test the chips with current technology. Moreover, the time needed to test the ICs on the wafer can be significantly reduced as well.

If the nature of the manufactured chips is unrestricted, the comparison model is the best choice for this application. In order to implement self-diagnosis on the wafer, hardware support is necessary. First, interconnection links need to be introduced and comparators need to be wired on each link. To keep the number of comparators relatively small, the interconnection structure should be regular and the degree of nodes small. Additional hardware support is required to provide the test sequence for the chip inputs and to collect the outcomes of the comparators. The latter task may be executed by the probing unit, which reads the outputs of the comparators and transmits them to a reliable external computer. Once the external computer receives the syndrome, it will execute the diagnosis algorithm.

Unfortunately, the comparators and the hardware added for the purpose of diagnosis may be faulty. This may lead to incorrect diagnosis if the diagnostic model does not take such faults into account: therefore a modification of the existing models is necessary.

In this paper we introduce an implementation of a comparison model in which faults in the comparators and in the additional hardware are considered.

The paper is structured as follows. In Section 2 we introduce a diagnosis algorithm assuming the PMC model. Section 3 extends the algorithm to comparator-based models and identifies the problems caused by faults in the comparators. Section 4 and 5 describe a technique which ensures correct diagnosis even in the presence of comparator faults. Some additional implementation issues are discussed in Section 6, and the implementation of the circuitry needed for the safe collection of test outcomes is described in Section 7.

2 A self-diagnosis algorithm for the PMC model

In order to perform the self-diagnosis, we need a *diagnosis algorithm* which, given a diagnostic graph and a syndrome, provides a diagnosis of the system; i.e. classifies every unit as either *faulty* or *fault-free* or *unknown*. The diagnosis algorithm usually runs on an external and reliable computer, called the *diagnoser*. The diagnoser is also assigned the task of collecting the test outcomes from the units in the system.

The diagnosis is said to be *correct* if no faulty unit is diagnosed as fault-free and no fault-free unit is diagnosed as faulty. The diagnosis is said to be *complete* if every unit is classified as either faulty or fault-free. It is known that a correct and complete diagnosis is possible only when the number of faults in the system is no more than the so-called *diagnosability*, which is bounded above by the minimum indegree of the nodes in the diagnostic graph [6].

In large scale systems where units are connected to a small number of neighbours, correct and complete diagnosis is guaranteed in the presence of a small number of faults. This is the case of wafer-scale testing,

where, however, the expected percentage of faulty ICs may be as large as 50%. For this reason, algorithms aiming at providing correct and complete diagnosis are not suitable for wafer-scale testing. In fact, the algorithms proposed so far provide a diagnosis which is either complete but possibly incorrect (*probabilistic diagnosis*) or correct but in general not complete (*incomplete diagnosis*). Huang et al. [7], Agarwal et al. [8], and LaForge et al. [9] use probabilistic PMC-based algorithms. Maestrini et al. [10] [11] aim at the proven correct diagnosis whose completeness is not guaranteed.

In all cases, PMC-based diagnosis algorithms extensively exploit some immediate consequences drawn from Table 1:

1. If two units A and B declare each other fault-free (we denote this by $A \overset{00}{\longleftrightarrow} B$, or simply call it a 00 result), then they are of the same state.
2. If B is faulty and there exists a 0-labelled path in *DG* ending at B and, then all the units in the path are faulty.
3. If unit A accuses B to be faulty but B declares A fault-free ($A \overset{01}{\longleftrightarrow} B$), then B is faulty.
4. If units A and B accuse each other to be faulty ($A \overset{11}{\longleftrightarrow} B$, or a 11 result), then at least one of them is faulty.

Maestrini's algorithm, which will be further developed in this paper, performs the diagnosis of bi-dimensional grids. However, it can be adapted to perform the diagnosis of other structures (arrays, octagonal meshes, hypercubes and so on) as well. The algorithm proceeds as follows:

- Initially, units proven to be faulty are identified using property 3.
- Then pairs of units of a still undetermined state, declaring each other faulty, are selected and classified as *dual* units. The goal of this step is to trade a faulty unit for an unknown one, as one of them is surely faulty; this way, some faults are separated and more reliable assumptions can be made about the number of faults in the remaining part.
- The remaining units are built up into *aggregates* based on 00 test results, and the largest aggregate (or the collection of those with the largest cardinality) is selected as the *fault-free core* (FFC).
- In the last step of the algorithm, the fault-free core is extended recursively, relying on the test results made by the units in the FFC.

The algorithm is proven to be correct if the number of faults in the system does not exceed a certain number (the *syndrome-dependent bound*, denoted by T_σ), asserted by the algorithm itself. The algorithm has the favourable property that this fault limit is bounded below by the *syndrome-independent bound*, which is $O(N^{2/3})$, where N is the number of the units in the system.

3 Comparator-based diagnosis

Under the assumption that no faults occur in comparators, a comparator-supplied syndrome can be easily converted into a PMC syndrome. A comparison result can be regarded as if mutual tests were performed by adjacent units with the outcomes shown in Table 2.

comp. output	PMC equivalent	situation in effect
0	00	both units faulty or both units fault-free
1	11	at least one unit faulty

Table 2: Equivalent PMC syndromes

With this transformation, which is basically the Chwa-Hakimi model, PMC-based algorithms will work without modification. Note that no comparator-based syndrome is converted to a 01 PMC syndrome, but any PMC algorithm should work without 01 mutual outcomes, since faulty units that would produce 0 "have the right" to declare other units faulty any time. Regarding the distribution of possible syndromes

over a specific system state, the loss of 01 results entails a somewhat decreased average efficiency, but the worst-case performance of a PMC diagnosis algorithm is still the same as that achieved by the same algorithm in a comparator-based implementation.

3.1 Impact of comparator faults

Using the outcome conversion described in the previous section leads to serious problems if comparators, containing physical faults, produce incorrect results. Table 3 describes how the PMC model is disturbed in the presence of comparator faults (CFs).

case	$unit_1$ state	$unit_2$ state	comp. state	comp. result	PMC conform
a	good	good	good	0	yes
b	good	good	faulty	0	yes
c	good	good	faulty	1	no
d	good	faulty	good	1	yes
e	good	faulty	faulty	1	yes
f	good	faulty	faulty	0	no
g	faulty	faulty	any	any	yes

Table 3: CFs and PMC compatibility

Note that *masked* CFs, such as in cases *b* and *e*, do not present a problem in the wafer-scale testing application, because the comparators will never be used again after the diagnosis is performed. Case *g* is also compliant with the PMC assumptions, since the test results of faulty units are regarded as unreliable. On the other hand, *active* CFs (cases *c* and *f*) may prevent test results from complying with the PMC invalidation model.

Case *c* is not critical in itself, because still a correct diagnosis can be obtained by a certain degradation of the diagnosis algorithm. The degradation consists in restating Assumption 4 as "at least one among units A, B, and the associated comparator is faulty." Since this means that both units A and B might be good, this degradation prevents the algorithm from falsely diagnosing a cycle of good units and one faulty comparator as faulty (Figure 1).

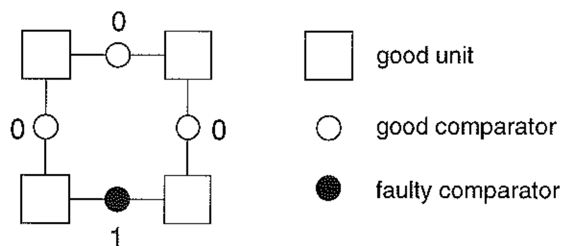


Figure 1: Cycle of good units with one faulty comparator

Weakly correctness means that faulty units are not declared fault-free. With only case *c*, Maestrini's algorithm [10] [11] is still *weakly correct* if the condition on the total number of faults (affecting units and/or comparators) holds. However, there is a chance that units declared faulty are actually fault-free when 11 results are not reliable. The syndrome-dependent bound T_σ also accounts for both unit and comparator faults, thus fewer unit faults are sufficient to occur to make the diagnosis incorrect.

Case *f* poses a much more serious problem. The correctness of every diagnosis algorithm known to the authors depends on the validity of the *aggregation* step, which is based on Assumption 1: units testing each other with 0 are in the same state. In this hypothesis, units in any set where every pair is connected by a 00 path can be declared as having identical states. The possible occurrence of case *f* entirely invalidates this hypothesis, since an erroneous 0 comparison may lead to the aggregation of a good and a faulty unit.

4 Pre-diagnosis comparator test session

In this section a preliminary test session is proposed which is capable of eliminating the situation described in line *f* of Table 3, for a wide class of comparator faults. For the sake of simplicity, single-bit comparators are examined first. The multiple-bit case will be discussed in the next section.

Our goal is to detect an erroneous 0 comparison between a fault-free and a faulty unit. We assume that we can feed any two adjacent units with independent inputs, and that we can force desired values to the output of any fault-free unit (these properties can be granted by a special wafer design, as shown later). If both units are good, the comparator can be tested exhaustively. Faulty units, however, may fail to feed the comparator with proper test patterns, and the fault of a unit may mask the fault of the comparator. Even in this situation, the test can still be performed, provided that a faulty unit always produces the same responses for identical inputs. (This condition means that we limit our consideration to permanent faults, and that units should be combinational. The latter requirement can be easily met by using the specific wafer design described in Section 6, where sequential units are bypassed with a combinational part.)

Figure 2 depicts the behaviour of the adjacent units during a complete comparator test session when everything is fault-free (Fig. 2/a) and when one of the components, say B, may be faulty together with the comparator (Fig. 2/b). (We assume that fault-free units produce outputs 0 and 1 for inputs in_0 and in_1, respectively, while faulty units produce outputs x and y for the same inputs.)

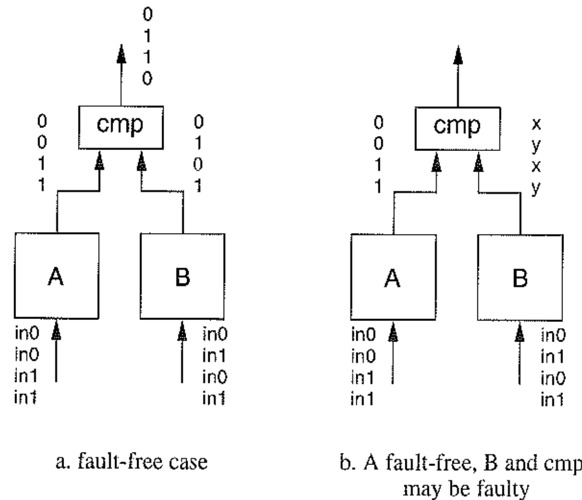


Figure 2: Comparator test session

Table 4 shows what effect a faulty unit may have on the comparator test. If the faulty unit B does not alter the comparator test patterns (line 1), then the comparator will undergo an exhaustive test. If B produces identical responses for in_0 and in_1 (lines 2 and 3), then the comparator will fail to produce the expected sequence 0110. Line 4 is, however, problematic, because the faulty unit B negates its responses for in_0 and in_1, and the faulty comparator might invert them again, thus passing the test in spite of faults. This situation will be referred to as inverting behaviour.

case	x	y	effect
1	0	1	full comparator test
2	0	0	no 0110 output sequence
3	1	1	no 0110 output sequence
4	1	0	problematic

Table 4: Impact of the behaviour of B

Fortunately, most practical comparator designs have the property that an arbitrary unit fault together with a wide class of comparator faults cannot remain undetected. For example, this is the case for the simple 1-bit comparator in Figure 3.

Lemma 1: In the 1-bit comparator in Figure 3, the inverting behaviour cannot occur for any multiplicity of gate-level stuck-at faults.

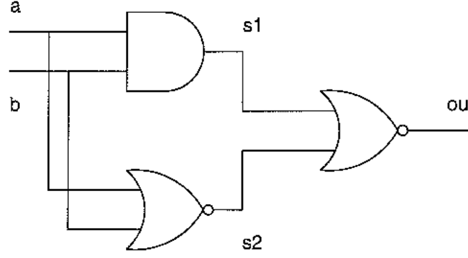


Figure 3: A simple 1-bit comparator

Proof: Table 5 enlists all stuck-at fault combinations, together with their corresponding detection patterns.[1] Some fault combinations (deriving from *don't care* entries in Table 5) require two input patterns to be detected. Although the faulty unit B may fail to drive the expected values to line b, this case will still produce a discrepancy. Since for every row the expected value of b in the input pattern(s) is constant (0 or 1), the "inverting" behaviour of unit B, displayed in row 4 of Table 4, is detected by detecting an equivalent stuck-at fault (*sa-1* or *sa-0*) of line b. For example, in row 3 pattern 11 detects the case of line s_1 stuck at 0 combined with any assignment of faults to *don't care* entries, except s_2 stuck at 1 or, equivalently, both a and b stuck at 0. The input pattern 01 is needed to handle such exceptions. The inverting behaviour of B is equivalent in this row with the fault of b *sa-0*, which is covered by assigning *sa-0* to the *don't care* entry of column b. □

out	s_1	s_2	a	b	pattern (ab)
0					01
1					00
ff	0				11 and 01
ff	1				01
ff	ff	0			00 and 10
ff	ff	1			01
ff	ff	ff	0		10 and 00
ff	ff	ff	1		01 and 11
ff	ff	ff	ff	0	01
ff	ff	ff	ff	1	00

Table 5: Comparator faults and detecting patterns

4.1 Multiple-bit comparators

A single-bit output is a rare practical case. However, considering the "bit-sliced" design of Figure 4, we will show that the technique of *Lemma 1* can be extended to an arbitrary width, while the test length increases proportionally with the number of slices.

Theorem 1: Consider the n-bit comparator of Figure 4. Assume that unit A is fault-free and unit B produces identical responses for identical inputs. Then, the test patterns in Table 6 will detect every combination of stuck-at faults in the comparator combined with any faulty behaviour of unit B.

Proof: The table contains a 4-vector long test sequence for each comparator bit slice, line 0 (a shared vector for each slice), and lines $3i + 1$ to $3i + 3$ for the ith slice ($i = 0, ..., n - 1$). They trivially detect any fault combination involving a stuck-at fault on line *out*. For the remaining faults, we will show that the test of slice i can be performed in spite of the effect of faults in other slices. Two cases may occur during the slice test:

1. As an effect of a fault in unit B or in the comparator, a faulty 1 value appears on some OR-gate input line, belonging to a slice other than i. Let us call the vector producing it v. Since there are 4 different vectors, differing only in input values feeding slice i (the others are all 0), these four vectors will contain another one (let us call it w) where only a_i differs from v. The slices other than i will

[1]The meaning of the table entries is the following:
0 (1): sa-0 (sa-1) fault; ff: fault-free state;
empty box: any state (don't care)

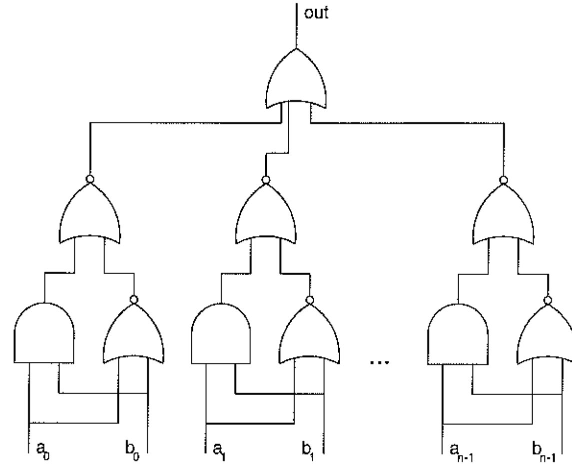


Figure 4: A simple n-bit comparator

a_0	b_0	...	a_i	b_i	...	a_{n-1}	b_{n-1}
0	0		0	0		0	0
0	1		0	0		0	0
1	0		0	0		0	0
1	1		0	0		0	0
...	...		...	...		...	...
0	0		0	1		0	0
0	0		1	0		0	0
0	0		1	1		0	0
...	...		...	...		...	...
0	0		0	0		0	1
0	0		0	0		1	0
0	0		0	0		1	1

Table 6: Test patterns for n slice

receive the same actual input values (even if modified by B) when we apply v and w, therefore the comparator output will be 1 for both patterns. On the other hand, the fault-free outputs should be different for v and w, i.e. the fault is detected.

2. The OR-gate input lines belonging to slices other than i always hold the value 0. In this case, the presence of faults in unit B or in the other comparator slices does not disturb the test of slice i, which detects any stuck-at fault in it according to the previous lemma.

Repeating the test for all slices, every combination of stuck-at faults in the comparator will at least once produce an output different from the expected one. □

In conclusion, it has been shown that the patterns of Table 6 are a complete test sequence for this comparator, and additionally, that a simple comparator has the property that a faulty B unit cannot mask comparator faults. ¿From the point of view of the diagnosis algorithm, we can summarize the results above as follows. *If only permanent faults are present in the units and only stuck-at faults are present in the comparators, then after a comparator passes the proposed preliminary test, the diagnosis algorithm can rely on its 0 output during the normal diagnostic session.* This eliminates the problem of case f in Table 3, so the aggregation step can be done safely.

It is also true that the applied preliminary test session excludes the undetected occurrence of line c, so the algorithms can also rely on the 1 results of comparators having passed the test. However, this fact has smaller practical significance, because a faulty unit may fail to provide one or more test vectors, and this will most probably cause the test to fail regardless of the state of the comparator: thus reliable 1 comparisons may occur rarely. A failed comparator test could mean therefore either case c, d, e, or g, in contrast to our intention of distinguishing between PMC-conform and non-compliant cases. On the other hand, if the comparator is fault-free and the fault in the unit does not modify the comparator

test patterns, then the comparator test will pass, and the comparison result will be a reliable 1. The comparator test implementation, suggested in Section 6, highly exploits this feature, achieving a good ratio of meaningful and reliable 1 results.

5 Diagnosis strategy

The introduction of the comparator test phase allows for a refinement in the handling of outcomes. The combined outcome of the comparison-based test of units (in short, diagnostic test) and the comparator test is determined according to Table 7:

comparator test	diagnostic test outcome	combined result
passed	0	reliable 0
passed	1	reliable 1
failed	0	unreliable
failed	1	unreliable

Table 7: Ternary combined outcomes

Since the *reliable 0* and *1* results indicate a passed comparator test, i.e. a comparator proven not to contain stuck-at faults, the diagnosis algorithm can act by regarding these results as valid. However, it takes no actions based on *unreliable* outcomes, because they may have been produced by a faulty comparator.

As mentioned earlier, *reliable 1* combined results will exist only when faulty units do not modify the comparator test patterns. This property can be achieved with good probability by a careful design of the wafer environment. It should be observed that if the occurrence of *reliable 1s* is not exploited, i.e. the case when a good comparator is declared *unreliable* in any case, the correctness of the algorithm is not impaired, although its performance may de somewhat decreased.

5.1 The CF tolerant algorithm

The comparator-based adaptation of Maestrini's algorithm (see Section 2 and [10]), which tolerates multiple stuck-at CFs, is as follows. Note that the indicated $\overset{0}{\longleftrightarrow}$ and $\overset{1}{\longleftrightarrow}$ diagnosis results are always *reliable* ones, whereas no action is taken at all for *unreliable* results.

1. Perform the comparator test and perform the diagnosis session. Remove diagnosis test results wherever the comparator test failed, and treat the remaining ones as *reliable*.
2. *(search for faulty units)*
    For every unit u
        $status_u := initial.$
    Let the fault set $\mathcal{F}$ be empty.
    While there exist *initial* units
        Choose a unit u of status *initial*.
        Let set $\mathcal{S}$ be empty.
        Let $sflag$ status flag be $nonfaulty$.
        $searchfaulty(u)$.
        If $sflag = faulty$
            $\mathcal{F} := \mathcal{F} \bigcup \mathcal{S}$.
            For every unit v in set $\mathcal{S}$
                $status_v := faulty$.

procedure $searchfaulty($ unit $u)$
    Put u into $\mathcal{S}$.
    $status_u :=$ *undetermined.*
    For every neighbour n of u

If $u \overset{0}{\longleftrightarrow} n$ and $status_n = initial$
$searchfaulty(n)$.
If $u \overset{1}{\longleftrightarrow} n$ and $n \epsilon \mathcal{S}$
$sflag := faulty$.

3. *(determination of dual units)*
Let the dual set $\mathcal{D}$ be empty.
For every unit u
if $status_u = undetermined$ and there exists a neigbour n of u that $status_n = undetermined$ and $u \overset{1}{\longleftrightarrow} n$
$status_u := dual$.
$status_n := dual$.
Put u and n into $\mathcal{D}$.

4. *(aggregation)*
$i := 0$.
While there exist *undetermined* units
$i := i + 1$.
Choose a unit u of status *undetermined*.
Let aggregate $\mathcal{A}_i$ be empty.
$aggregate(u)$.

procedure $aggregate($ unit $u)$
Put u into set $\mathcal{A}_i$.
$status_u := aggregated$.
For every neighbour n of u
If $status_n = undetermined$ and $u \overset{0}{\longleftrightarrow} n$
$aggregate(n)$.

5. *(determination of the fault-free core)*
Select aggregate $\mathcal{A}_i$ where $|\mathcal{A}_i| \geq |\mathcal{A}_j|$ for $\forall j$.
(If more than one such aggregates exist, select all of them.)
Let $\mathcal{A}_{max}$ be the union of the selected aggregates.
For every unit u in set $\mathcal{A}_{max}$
$status_u := faultfree$.
$T_\sigma = |\mathcal{A}_{max}| + |\mathcal{D}|/2 + |\mathcal{F}|$.

6. *(augmentation of the FFC)*
While there exist $faultfree$ units
Select a unit u of status $faultfree$.
$status_u := augmented$.
For every neighbour n of u
If $status_n = dual$ or $status_n = aggregated$
If $u \overset{0}{\longleftrightarrow} n$
$status_n := faultfree$.
If $u \overset{1}{\longleftrightarrow} n$
$augmentfaulty(n)$.

procedure $augmentfaulty($ unit $u)$
$status_u := faulty$.
For every neighbour n of u
If $u \overset{0}{\longleftrightarrow} n$ and $status_n \neq faulty$
$augmentfaulty(n)$.

7. *(conclusion)*
Units of status *augmented* are fault-free. Units of status $faulty$ are faulty. This diagnosis is correct if $t < T_\sigma$, where t is the actual number of unit faults in the system. Units of status *aggregated* or *dual* can be either fault-free or faulty. If such units exist,

the diagnosis is incomplete.

6 Implementation issues

To be able to carry out the proposed comparator test session, the wafer diagnosis environment should meet the following requirements.

- The fault-free circuits should be able to provide the comparator with every comparator test pattern (see Table 6).
- A circuit containing permanent faults should produce identical responses for identical inputs.
- Although the units are fed by the same input during the diagnostic test session, during the comparator test phase two adjacent units should be driven with different input values.

The first two conditions, if not already fulfilled by the actual unit design, can be easily satisfied by a simple modification in the circuit design, e.g. by multiplexing the output of the unit.

The third requirement is a more difficult one. It requires either that the input bus (or at least one bit) be duplicated in the wafer diagnosis circuitry, or that the chips on the wafer be non-identical (in case of a rectangular grid diagnosis structure, there must be two kinds of chips).

A comprehensive solution to these problems could be the addition of some simple combinational logic that generates the comparator test patterns for the comparator test session (Figure 5). One bit, encoding the position of the chip ($A/\overline{B}$), is introduced only in the final stage of the design of the wafer masks. During the comparator test session, the diagnosis circuitry receives a wafer-wide signal which tells it to switch to the output of the test pattern generator (TPG) instead of that of the unit. The function of the TPG can be implemented with a small ROM module indexed by the common input, or with a more concise combinational logic, since the test patterns are easily compressable. It should be emphasized that a sequential design for the TPG logic would contradict our assumption that a repeated test pattern is always altered the same way by permanent faults.

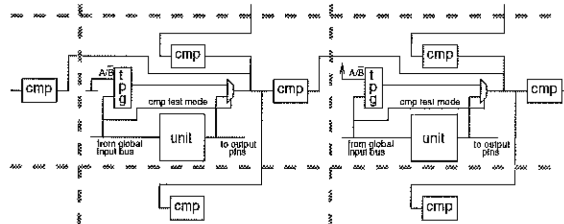


Figure 5: Non-identical chips on the wafer

This solution has a number of favourable features:

- The chips are capable of playing the role of either unit A or B without the intervention of the diagnosis algorithm and without requiring the duplication of the input bus. The additional cost is percentually small, especially in the case of complex chips.
- The wafer-based testing implementation remains hidden to the circuit designer, because the TPG and comparator subcircuitry can be added by the silicon factory, and the circuit user, because during packaging diagnostic links are not connected to pins.
- The design of the TPG and the comparator is independent of the nature of the IC to be manufactured, since including this additional logic does not require information other than the output width.

An even more desirable property of the TPG implementation is the satisfactory treatment of case c in Table 3. If the unit is faulty but the TPG logic is intact, a fault-free comparator will pass the comparator test and will signal the different behaviour of the units during the diagnosis session. In other words, a fault-free comparator will fail the test and be declared as unreliable only if a feeding TPG is faulty, which has relatively low probability.

7 Observing results

The bottleneck in the performance of comparison-based testing is the observation of comparison results. Since the diagnostic test can be very long, the diagnoser should not be required to read the entire sequence of comparator outputs; instead, a syndrome collection circuitry should be wired to every comparator. Excluding the faults in the diagnostic circuitry, this could essentially be a 1-bit RS flip-flop, which is reset before executing the diagnostic test, and set by any mismatch the compared chips produce.

This technique can also be used in the approach extended with a comparator test session, but some problems must be solved. First, a simple 1-triggered flip-flop is not sufficient, because the comparator produces 1 results during the comparator test session even if everything is fault-free. Furthermore, potential faults in the flip-flop must be taken into account, therefore the comparator test must be extended to cover faults in the collection circuitry as well. Finally, the number of read-out operations should be minimized.

We will present a syndrome collection strategy and a general overview of the supporting hardware, which handles all these problems. We suggest that an RS flip-flop (Figure 6) be used for syndrome collection, with the following features:

- It contains a *guard* signal, controllable by the external diagnoser, which masks the transients that occur on the *set* input of the FF.
- An auxiliary signal *aux*, also controllable, is used to control the *set* line. During the comparator and diagnostic test sessions, this line always holds the same value as the expected output of the comparator, so that one flip-flop is sufficient for collecting both 1s and 0s. Furthermore, during the flip-flop test (see later), the signal improves controllability.

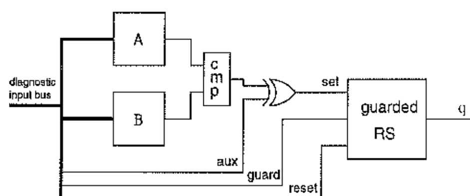


Figure 6: Syndrome collection circuitry

We extend the comparator test session to include a flip-flop test, which, if passes, ensures that the collection circuitry is also free from faults. We will prove that the flip-flop test detects *all* combinations of stuck-at faults in the collection circuitry as well.

The complete test session, including the extended comparator test session and the diagnostic test session, requires only 3 read-out operations, and proceeds as follows.

- Initially, we feed units A and B with a comparator test pattern that normally produces a 0 comparator output (e.g. the first line of Table 6), we set *aux* to 1, and disable *guard*. This should make line *set* hold value 1 in the fault-free case. We issue a *reset* command, then enable *guard* (which should set the output, since *set* still holds 1), and check if the FF has been indeed set.
- Next, we disable the *guard*, reset again, and perform the comparator test, taking care that *aux* always holds the same value as what is expected from the comparator, and that *guard* is only active when the comparator output settles down. After the comparator test, we switch *aux* without enabling the *guard*, and read out the comparator test result, which is expected to be 0.
- Finally, we perform the diagnostic test, keeping *aux* 0 and enabling *guard* after each vector, and read out its result.

The next theorem says that the result of the diagnostic test is reliable if we receive a 10 sequence for the first two read-outs.

Theorem 2: For the gate level model shown in Figure 7, all combinations of gate-level stuck-at faults in the syndrome collection circuitry will cause the combined flip-flop–comparator test to fail.

Proof: We show in particular that every combination of stuck-at faults entails a test result other than 10. Table 8 lists all stuck-at fault combinations, along with the possible read-out results for the first two read operations. The meanings of the table entries are the same as in Table 5.

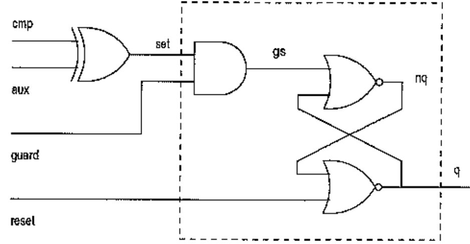


Figure 7: RS flip-flop at the gate level

q	reset	nq	gs	set	aux	guard	equiv. fault	read out
1								11
0								00
ff	1						q sa-0	
ff	0							0X 11
ff	ff	1					q sa-0	
ff	ff	0						00
ff	ff	ff	1				nq sa-0	
ff	ff	ff	0					00
ff	ff	ff	ff	0			gs sa-0	
ff	ff	ff	ff	1				00 11
ff	ff	ff	ff	ff	0			0X 11
ff	ff	ff	ff	ff	1			0X 11
ff	ff	ff	ff	ff	ff	0	gs sa-0	
ff	ff	ff	ff	ff	ff	1		11

Table 8: Gate level fault coverage of the test

Some faults may produce several output sequences, depending on the actual behaviour of the driving circuitry (A, B, and the comparator), and on the signals of which the state is listed as *don't care* in the given line. For example, the line where signals q, *reset*, nq, gs, and *set* are fault-free but *aux* is sa-0 summarizes the following:

- If the *guard* is sa-0, the read-out sequence will be 00.
- If the *guard* is sa-1 or fault-free, and the comparator correctly produces 0 for the initial pattern, then the first bit read out will be 0.
- If the *guard* is sa-1 or fault-free, and the comparator incorrectly produces 1 for the initial pattern, then the read-out sequence will be 11, since the initial pattern is part of the comparator test, which will therefore fail.

Note that we exploit again the fact that the circuitry comprising the comparator and the TPG part of the two units is combinational, therefore the same faults with the same input will produce an identical output. Since no fault set listed in the table allows for a 10 read-out sequence, a passed FF–comparator test guarantees that the circuitry is free from stuck-at faults. □

8 Conclusion

In the present paper the sensitivity of comparison-based wafer diagnosis algorithms to comparator faults has been examined. It has been pointed out that the most serious problem is the possibility of incorrect

aggregation. A preliminary comparator test session has been proposed to eliminate this risk. It has been proven that this session will prevent a false 0-comparison for *any* permanent unit fault with *any* combination of stuck-at faults in the comparators. This approach excludes the need of a complex fault-tolerant comparator circuitry. The comparator test session requires the chips to be capable of driving the comparator test patterns, and any two adjacent chips to behave in a different way during the test. A simple wafer design solution, ensuring this property, has been shown as well.

The incorrect 1-comparison has been identified as a less serious problem. With the application of the comparator test session, situations with this risk are also detected, at the price that perhaps fault-free comparators are also declared unreliable. The proposed wafer implementation strongly reduces the probabaility of this conservative diagnosis.

An extension of the comparator test has been also given which detects multiple stuck-at faults not only in the comparators but in the syndrome collection circuitry as well.

References

[1] F.P. Preparata, G. Metze, and R.T. Chien: "On the connection assignment problem of diagnosable systems," *IEEE Trans. Electron. Comput.*, vol. EC-16, pp. 848-854, Dec. 1967.

[2] M. Malek: "A comparison connection assigment for diagnosis of multiprocessor systems," *Proc. 7th FTCS*, pp. 31-35, May 1980.

[3] K.Y. Chwa, S.L. Hakimi: "Schemes for fault-tolerant computing: a comparison of modulary redundant and t-diagnosable systems," *Information and Controls*, vol. 45, No. 3, pp. 212-238, 1981.

[4] J. Maeng and M. Malek: "A comparison assignment for self-diagnosis of multicomputer systems," *Proc. 11th FTCS*, pp. 173-175, 1981.

[5] S. Rangajaran, D. Fussell, and M. Malek: "Built-in testing of integrated circuit wafers," *IEEE Trans. Comput.*, vol. 39, pp. 195-205, Feb. 1990.

[6] S.L. Hakimi and A.T. Amin: "Characterization of connection assignment of diagnosable systems," *IEEE Trans. Comput.*, vol. C-23, pp. 86-87, Jan. 1974.

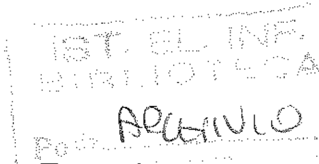
[7] K. Huang, V.K. Agarwal, L. LaForge, and K. Thulasiraman: "A diagnosis algorithm for constant degree structures and its application to VLSI circuit testing," *IEEE Trans. Parallel and Distr. Systems*, vol. 6, pp. 363-372, Apr. 1995.

[8] A.K. Somani and V.K. Agarwal: "Distributed diagnosis algorithms for regular interconnected structures," *IEEE Trans. Comput.*, vol. C-41, No. 7, pp. 899-906, Jul. 1992.

[9] L.E. LaForge, K. Huang, and V.K. Agarwal: "Almost sure diagnosis of almost every good element," *IEEE Trans. Comput.*, vol. C-43, pp. 295-305, March 1994.

[10] S. Chessa and P. Maestrini: "Self-Test of Integrated Circuit Wafers," *Proc. ETW-96 European Test Workshop '96*, pp. 54-58. Sete-Montpellier, France, June 1996,

[11] P. Maestrini and P. Santi: "Self Diagnosis of Processor Arrays Using a Comparison Model," *Proc. 14th Symposium on Reliable Distributed Systems*, pp. 218-228, Sept. 1995.



B4-26 T. R

A Comparison-Based Diagnosis Algorithm Tolerating Comparator Faults

Balázs Sallay*, Piero Maestrini, and Paolo Santi
Istituto di Elaborazione dell'Informazione del CNR, Pisa, Italy

Abstract

A promising application of system-level diagnosis is the testing of VLSI chips during the manufacturing process. A comparison-based diagnosis is easier to implement on the wafer than the PMC-conform one. However, existing comparison models essentially overlook the test invalidation due to the physical faults in the comparators. This paper proposes a comparison-based model and a diagnosis algorithm which takes into account the effects of faults that affect the comparators. In order to deal with faults in the comparators, a preliminary comparator test session is included. This session requires the adjacent units to be able to feed the comparator with all needed patterns independently of each other. As shown in the paper, this requirement can be satisfied at a relatively small wafer design cost. The test session can also be extended to handle the faults in the syndrome collection circuitry.

Categories and Subject Descriptors: B.7.m [**Integrated Circuits**]: Miscellaneous; B.8.1 [**Performance and Reliabiltity**]: Reliability, Testing, and Fault-Tolerance

General Terms: Algorithms, Design, Reliability

1 Introduction

System level diagnosis, also called self-diagnosis, has been introduced by Preparata et al. in 1967 [1]. In self-diagnosis, a system composed of several units connected by bidirectional links can be diagnosed using the information provided by *tests*. These tests are performed by the units comprising the system itself, and each of them involves two units, called the *testing* and the *tested* unit. The testing and the tested unit must be adjacent (that is, interconnected by bidirectional links). Essentially, a test is performed as follows:

- the testing unit requests the tested unit to run a test;
- the tested unit returns a result to the testing unit;
- the testing unit compares the actual and the expected results and provides a binary test outcome. The outcome is 0 if the actual and the expected results match (the test passes), 1 otherwise (the test fails).

The set of tests utilized for the purpose of diagnosis is defined by the directed graph $DG = (V, E)$, where V is the set of units and $E = \{(u, v)$ such that unit u tests unit $v\}$. DG is called the *diagnostic graph* of the system. Observe that the tests involving different units can be run simultaneously. The set of all test outcomes is called *syndrome*.

The test results are not necessarily reliable, since testing units themselves may be faulty. Different hypotheses upon the test outcome generated by faulty units lead to different *invalidation rules*, and consequently to different *diagnostic models*. The most widely used diagnostic model is the PMC model introduced in [1], which assumes arbitrary test outcomes for tests performed by faulty units. The invalidation rule of the PMC model is shown in Table 1.

Comparison models for self-diagnosis have also been introduced in literature. In comparison models, tests are performed by comparators which compare the outputs of pairs of units that run the same test. The diagnostic graph corresponding to a comparison model has an undirected edge between a pair of nodes if there exists a comparator between the corresponding units. The outcome of the comparison is 0

*The work of this author was supported by the European Research Consortium for Informatics and Mathematics