BUILT-IN TESTING
OF INTEGRATED CIRCUIT WAFERS*

Sampath Rangarajan, Donald Fussell, and
Miroslaw Malek**

Department of Computer Sciences
The University of Texas at Austin
Austin, Texas 78712-1188

TR-87-27 July 1987

Abstract

Production testing of a potentially faulty digital circuit consists of two types of tests, parametric tests to determine whether the production process has produced the proper electrical parameters for the features on the wafer, and at-speed logic testing which identifies the effects of random defects on the behavior of individual circuits. The latter type of testing requires the generation of a sequence of test vectors and their application to the circuit being tested. Traditionally, in the second step the output of the circuit under test is compared with a known correct output for each test vector by a *test computer*. This method has some drawbacks in certain circumstances which promise to become more critical in the near future. These drawbacks include difficulties in producing test computers that can drive circuits under test at their rated operating speed, excessive time spent in testing an entire wafer circuit by circuit, complex probing hardware for circuits with many I/O pins, and the difficulty of using test computers to find failures in wafer-scale systems during their actual operation. In homogeneous systems of identical integrated circuits like these, testing can be done in another way—by applying a common test vector to several processing elements at once and comparing the results produced by them. In such a scheme, neither the test computer nor even a set of *a priori* correct results is needed. The amount of special testing logic required is thus minimized, and all of the problems mentioned above are solved. These features make this scheme attractive as an alternative means for production testing of silicon wafers consisting of an array of identical circuits. We analyze such schemes and show that perhaps surprisingly they are inherently as accurate as the traditional method for production testing. We also show that this approach should allow wafers to be tested for production faults significantly faster than by using a probe tester.

*This research was supported in part by the Office of Naval Research under Contracts N00014-86-K-0597 and N00014-85-K-0763.

**Department of Electrical and Computer Engineering.

1 Introduction

Production testing of a potentially faulty digital circuit consists of two types of tests, parametric tests to determine whether the production process has produced the proper electrical parameters for the features on the wafer, and at-speed logic testing which identifies the effects of random defects on the behavior of individual circuits. The latter type of testing requires the generation of a sequence of test vectors and their application to the circuit being tested. Traditionally, in the second step the output of the circuit under test is compared with a known correct output for each test step by a *test computer* which is assumed to be fault-free. Moreover, the computers used for computing the known correct results are also assumed to be fault-free.

The use of IC test computers for production testing has always been considered to be obviously the only way to proceed, but in fact it has some drawbacks in certain circumstances which promise to become more critical in the near future. Potentially the most damaging of these drawbacks is tester speed, since at-speedtesting must be done at the rated operating speed of the circuit under test to be accurate. With the advent of very high speed circuits such as those made with gallium arsenide, the rated speed of the circuit may be greater than that achievable by current test computers, which are made of ECL parts. This will require a new generation of test computers based on the same technology as the circuits under test. It is likely to be difficult to build such testers cost-effectively, especially if every advance in performance at the IC level causes the performance of the parts to outstrip that of the exising generation of test computers.

Another drawback is the time required to test an entire wafer by probe testing a single circuit at a time. As individual circuits become ever more complex, the size of the set of test vectors required to provide good fault coverage increases, and so does the testing time per circuit. Since a wafer typically contains hundreds of copies of circuits, testing the entire wafer a single circuit at a time, with a physical repositioning of the probes from one circuit to the next at each step, can take a very long time and add a great deal to the production cost of the circuits. A related problem is the ever increasing number of I/O pins on modern circuits, which requires increasingly complex probing hardware on the tester.

A final drawback is related to the use of a probe-testing technique in wafer-scale integrated systems, which are a candidate for a new generation of semiconductor technology. Prototypes of such systems have been built in which a working system is configured out of working parts after all the individual modules have been tested using traditional probe testing techniques. A major problem with this approach is that once the wafer is tested and configured, it is difficult or impossible to retest and reconfigure it in the event of a post-production failure of one or more of the modules that passed the production testing step.

In this paper we examine a different approach to production testing which provides solutions to all of these problems. It begins with the presumption that the system to be tested consists of an array of identical digital sequential or combinational circuits that can communicate among themselves. These circuits can be tested simultaneously with the same test vectors and the results compared locally within each module on the wafer to determine whether a fault has been detected in that module. If an ordinary wafer containing an array of identical copies of an integrated circuit is augmented with some simple testing hardware for each module and communication paths between the circuits, then the requirements of our approach are met. In this regard, we quote the following from [10].

> *"Let us mention here another possible use of the net of buses and switches on a wafer, that was suggested to us by someone in the industrial community. Since the width of the buses and switches turn out to be narrower than the scribe lines themselves , one could incorporate them on ordinary wafers, at no extra cost, and use them to test the chips on the wafer before dicing it. In this way, the cost of testing can be reduced, because there is no need to probe the individual chips."*

Other than the cost savings mentioned above, we would show that our algorithms will lead to considerable time saving too.

While comparison-based or *voting* schemes related to this approach have been used for *fault masking* during the operation of highly-reliable systems [1], [8], [11], [13], [15], rarely if ever have they been considered seriously for production testing. Our analysis shows that perhaps surprisingly this approach is potentially very practical in its application to production testing. Comparison based algorithms that have been proposed for System Level Fault Diagnosis [14], [2], [7] seem to be designed with a more pessimistic view of the situation and do not model the situation realistically. Also, Systolic array fault diagnosis algorithms based on a Comparison approach have been proposed in [4] and [5]

Because we are concerned with an analysis of the production fault testing process as distinguished from the test vector generation process, we base our analysis on a view of an integrated circuit as a "black box" computing a function, and therefore the testing problem is simply to determine whether the function performed is the desired one. This approach detaches our analysis from any particular method of test vector generation which may be used to generate the test patterns so that we can perform a generic analysis of the performance of comparison-based testing. It should be noted that this should not be interpreted as meaning that this approach depends in any way for its operation on random test vector generation, but rather that we have no hard statistical data on the characteristics of real test sets and thus must rely on such a generic probabilistic analysis.

We analyze the performance of voting schemes in terms of the degree to which they can be relied upon to accurately discriminate between faulty and fault-free modules and in terms of the time they require to do so. Our results show that these schemes are at least as accurate as the traditional method using a test computer in this regard, especially when used in wafer-scale systems. We also show that voting schemes are attractive in terms of the time they require to perform the testing. Since built-in testing eliminates problems of matching test computer speed to module speed and requires probing of a wafer only to receive test results and possibly to broadcast test vectors, this approach is therefore shown to solve the problems detailed above for at-speed testing.

2 A Functional Model of Fault Diagnosis

As indicated above, we are interested in studying the *application* of test vectors rather than their *generation*. Since we are concerned with production fault diagnosis primarily in a VLSI environment, we will assume that we are not in a position to repair a faulty circuit and therefore that our tests need only tell us whether or not a circuit functions correctly and not how or where it may have failed. Thus we will not be concerned with how one associates the results of a test with a particular fault in a circuit. Circuits to be tested can be either combinational or sequential in nature. In either case, signature generators can be used to combine the outputs produced by each module from sequence of test vectors to reduce the internal communications required for transmission of results to be compared and in the case of sequential circuits to support an LSSD scheme [9]. This allows us to equate a test of a sequential circuit with that of a combinational circuit and thus to define a single, simple functional model which covers both types of circuits. It is also worth noting at this point that while we frequently refer to the circuits being tested as processing elements, they can in fact be any type of digital circuit.

A *target circuit* C_t is a function mapping a domain I of input values to a range O of output values. C_t is intended to represent an ideal or correct circuit, in other words, the function that is computed by a fault-free physical realization of the circuit. Let C_a donate an *instance* or *realization* of C. C_a is thus also a function from I to O, but not necessarily equal to C_t. When C_a is not equal to C_t, we say that C_a is *faulty*. In production testing, our goal isto check whether or not a given C_a is faulty i.e. to check whether or not $C_a = C_t$.

Of course, the obvious way that this can be done is to exhaustively generate $C_t(e)$ and $C_a(e)$ for every element e in I and then check whether $C_a(e) = C_t(e)$, but frequently the size of I makes this *a priori* impractical.

A *failure model* for a target circuit C_t is a family of functions $\{C_t, C_1, C_2, ..., C_n\}$, where the subset $\{C_1, C_2, ..., C_n\}$ is the set of possible functionally distinct faulty realizations of C_t. For $0 < i \leq n$, the ith *fault state* C_i is a unique function on domain I of C and is therefore uniquely characterized by the subset $I_i = \{e|C_a(e) \neq C_t(e)\}$ of I, i.e. the set of input values for which faulty outputs are produced. Each member of I_i is called a *characteristic input* for faulty circuit C_i. For any particular input value $e \in I$ there is a *fault coverage set* $s = \{i|e \in I_i\}$ of faulty realizations of C_t for which e is a characteristic input. Then $\frac{|s|}{n}$, the fraction of the total number of fault states that input e characterizes, is called the *coverage* of e. A fault coverage set s can be partitioned into a set of *failure modes* where each failure mode is the set of all faulty circuit realizations in s which produce the identical faulty output on input e (*Figure 1*).

A *test* for a circuit C is a sequence $t = \{v_1, v_2, ..., v_m\}$ of *test vectors*, each of which is an element of the input set I of C_t. Each such element v_i has an associated fault coverage set s_i. We can let $S = \cup_{i=1}^{m} s_i$, the union of these fault coverage sets, be the fault coverage set of t. A *test set* for C is a sequence $T = \{t_1, t_2, ..., t_r\}$ of tests, and the *coverage of a test set* is

$$\frac{|\cup_{i=1}^{r} S_i|}{n}$$

which is the cardinality of its fault coverage set divided by the number of fault states.

If the test set being used to determine the correctness of a circuit is all of I, then its coverage is obviously 1, but as noted above the the size of this set generally makes such exhaustive testing impossible in practice. Research on the generation of test sets is concentrated on methods that select small test sets with the highest possible coverage, but how these sets are generated is not our concern here. We will see later that not only does the coverage of the entire test set being used bear on the quality of the testing, but that the coverage of the individual test vectors in these test sets can be very important as well.

Having briefly defined the production testing problem, we need to look at the process of performing the tests. Note that if signature generation is used, the outputs considered for each test are the outputs of the signature generator rather than the outputs of the circuit under test but for notational simlicity we will consider the output domain of the signature generator to be the same as the output domain of the IC itself. Consider an n-ary *evaluation predicate* $V(o_1, o_2, ..., o_n)$ on $\{O \times O \times ... \times O\}$. The *i*th *test* of circuit realization C_a is the value of $V(C_a(t_i), o_{i,2}, ..., o_{i,n})$. If $V(C_a(t_i), o_{i,2}, ..., o_{i,n}) \neq 0$ for some

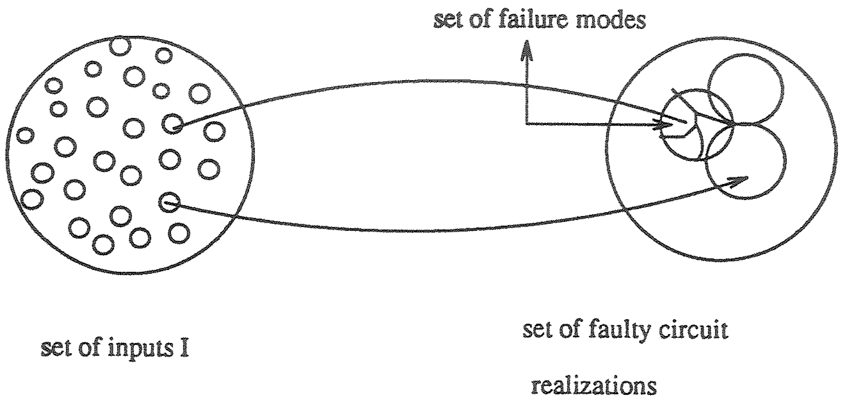


Figure 1: Functional Fault Model

$0 < i \leq n$ then we conclude that C_a is faulty with respect to test vector t_i. A *diagnosis* of a circuit realization C_a is a sequence $R = \{r_1, r_2, ..., r_r\}$, where $r_i = V(C_a(t_i), o_{i,2}, ..., o_{i,n}))$, that is R is the sequence of r results of the tests performed by applying V to the outputs of C_a for each of the test vectors. A circuit realization C_a is deemed fault-free only if $r_i = 0, \forall i, 1 \leq i \leq r$ i.e. the diagnosis vector does not contain any ones.

Suppose that we have fabricated n instances of C, $C_1, C_2, ..., C_n$ and we are testing C_i by comparing its output on each test with the outputs of all C_j for which $j \neq i$. Then we can define an evaluation predicate $V(C_i(t), C_1(t), C_2(t)..., C_n(t)) = 1$ iff at least k of the $C_j(t), 1 \leq j \leq n$, are all equal to $C_i(t)$ for some k, called the *threshold* of V. More generally, *voting algorithms* for functional fault testing are characterized by having evaluation predicates of this form for some values of n and k. Of course, the traditional testing method is also a voting algorithm, as we will see in detail later.

3 Accuracy of Testing by Voting

In evaluating a mechanism for testing, it is necessary to determine how closely it will approximate the ideal testing algorithm. In order to compare a particular testing algorithm to the ideal algorithm, we will examine the differences in the diagnosis of a particular unit that would beproduced by the testing algorithm and the ideal algorithm. For each element of the diagnoses produced by testing algorithm and the ideal algorithm for a particular unit under test, four conditions are possible. Both results could be 0 when the tests indicate no fault for the unit or both could be 1 when the tests indicate that a fault has been detected. These are the two conditions under which the evaluation function of a testing algorithm has made a correct determination. The other two conditions occur when the testing algorithm indicates that the unit under test is faulty (a 1 in the diagnosis vector) when the ideal test finds no fault (a 0 diagnosis vector element), a result which we call a *pessimistic diagnosis*, and the condition where the testing algorithm finds no fault when the ideal algorithm does detect a faulty condition, which we call an *optimistic diagnosis.* We will now analyze the probabilities that these types of diagnostic errors occur.

Let $p_{pes,i}$ designate the probability of a pessimistic diagnostic error for the ith test in a diagnosis, and let $p_{opt,i}$ denote the corresponding probability of an optimistic error. Also, let $\overline{p}_{pes,i}$ denote the probability that a pessimistic error does not occur for the ith test. For a voting algorithm with threshold k, a pessimistic error *does not* occur when at least k voters return results identical to the correct result produced by the unit being tested, i.e. out of the n voters a combination of at least k, fault-free voters or faulty realizations of the voters for whom the ith test vector t_i is not a characteristic input, could be found, and an optimistic error occurs when the unit being tested is not covered by t_i (so that the faulty unit produces the correct output for the test vector t_i) and at least k voters return results identical to the correct result produced by the unit under test, or when the unit being tested is covered by t_i (so that the faulty unit produces an incorrect result) but at least k voters return results identical to a faulty result produced by the unit being tested, i.e. out of the n voters, k voters belong to the same failure mode as the unit being tested for the test pattern t_i. Let us assume that there are f_i failure modes which are covered by t_i and that these failure modes are all equally likely to occur in any faulty circuit realization of C. If we further assume that each circuit C_j can fail independently of all other circuits with probability p, then the probability that circuit C_j produces an incorrect output for input t_i is $c_i p$, the probability that the circuit C_j produces a correct output for input t_i is $1-c_i p$, and the probability that an incorrect output produced falls into a particular failure mode is $\frac{c_i p}{f_i}$, where c_i is the coverage of the test t_i.

From the above definitions it is clear that for the ith test vector t_i, the probability that pessimistic error does not occur will be

$$\overline{p}_{pes,i} = \sum_{i=k}^{n} \binom{n}{i} (c_i p)^{n-i} (1 - c_i p)^i$$

and the optimistic error probability will be

$$p_{opt,i} = (1 - c_i)\overline{p}_{pes,i} + c_i \sum_{j=k}^{n} \binom{n}{j} (\frac{c_i p}{f_i})^j (1 - \frac{c_i p}{f_i})^{n-j}$$

Generally, a test will be made up of a set $T = \{t_1, t_2, ..., t_r\}$ tests . Let r be the cardinality of the test set T so that $r = |T|$. Also, let $p_{pes}(t)$ and $p_{opt}(t)$ denote the probabilities of pessimistic error and optimistic error after the application of t tests . The test will proceed in such a way that the r tests are applied in sequence to the system of units under test, the result of the test being evaluated after all the tests have been applied. Initially, all the units are assumed to be fault-free so that the pessimistic error probability is at its minimum value, which is zero and the optimistic error probability is at its maximum value. Every application of a test vector weeds out some units as faulty. This would mean that the pessimistic error probability increases with the application of more and more tests and simultaneously, the optimistic error probability decreases with the application of more and more tests. That is, both $p_{pes}(t)$ and $p_{opt}(t)$ are functions of the number of tests that have been applied so far, during the course of the test. Units once deemed faulty by a test vector do not make decisions on their fault status any more, but still function as voters by deciding on the results of other units for the remaining tests . This observation leads to a new set of equations for $p_{pes}(t)$ and $p_{opt}(t)$ different from the ones derived before for a single test vector.

Assume that the coverages of all the tests are equal, so that $c_i = c, \forall i, 1 \leq i \leq r$ and also the number of failure modes covered by all the tests are equal, so that $f_i = f, \forall i, 1 \leq i \leq r$ $\overline{p}_{pes}(t)$ is bounded from below by the probability of finding at least k fault-free voters. If at least k fault-free voters cannot be found, then, the probability of finding a combination of k fault-free and faulty voters giving the correct output is a function of the tests. Consider a particular instant during the test when t out of the r tests have been applied so far. At this instant of the test, $\overline{p}_{pes}(t)$ will be

$$\overline{p}_{pes}(t) = \sum_{i=k}^{n} \binom{n}{i} p^{n-i}(1-p)^i + \sum_{i=0}^{k-1} \binom{n}{i} p^{n-i}(1-p)^i \left(\sum_{j=k-i}^{n-i} \binom{n-i}{j} c^{n-i-j}(1-c)^j \right)^t$$

Note that the first component of the above equation, which is independent of the number of tests, bounds $\overline{p}_{pes}(t)$ from below. The above equation could also be written as

$$\overline{p}_{pes}(t) = \sum_{i=0}^{n} \binom{n}{i} p^{n-i}(1-p)^i \left(\sum_{j=k-i}^{n-i} \binom{n-i}{j} c_i^{n-i-j}(1-c_i)^j \right)^t$$

The pessimistic error probability will be given as a function of the number of tests as

$$p_{pes}(t) = 1 - \overline{p}_{pes}(t)$$

An optimistic error would have been made on a faulty unit under test after the application of t tests if

1. This faulty unit was not covered by any of the t tests and a combination of k fault-free and faulty voters giving a correct result identical to the unit under test could be found for each of the t tests . The probability of this occurence will be

$$h0 = (1-c)^t \overline{p}_{pes}(t)$$

2. This faulty unit was covered by a set of m tests $1 \leq m \leq T$ but it could find k voters which had the same failure mode so that it gave identical faulty outputs for those test tests and for the remaining $t - m$ tests which did not cover this particular unit, it could find at least k voters that gave the correct result. The probability of this occurence will be

$$h1 = R \sum_{i=1}^{t} \binom{t}{i} c^i (1-c)^{t-i} \overline{p}_{pes}(t-i)$$

where R is the probability that there are at least k voters out of the n voters which are (1) faulty and (2) have the same failure mode as the faulty unit in question. This will be given by

$$R = \sum_{j=k}^{n} \binom{n}{j} \left(\frac{pc}{f}\right)^j \left(1 - \frac{pc}{f}\right)^{n-j}$$

where $\frac{pc}{f}$ gives the probability that a particular voter is faulty, is covered by a particular vector and has the same failure mode as the faulty unit in question.

So the optimimistic error proabability will be given as

$$p_{opt,i} = h0 + h1$$

In all of the above analysis, we have assumed a uniform distribution of coverage for all the tests over the set of all failure modes.

In order to study the implications of the above analysis, we define two more parameters as follows:

Locatability (L): This is the probability that a fault-free unit has been identified as fault-free at the end of the test after the required number of tests have been applied. Locatability is considered undefined during a test after the application of only a limited number of the total tests to be applied.

Identifiability (I) : This is the probability that a unit is identified correctly as either faulty or fault-free at the end of the test.

These two parameters are given in terms of $p_{pes}(t)$ and $p_{opt}(t)$ as

$$L = 1 - p_{pes}(r)$$

and

$$I = (1-p)L + p(1 - p_{opt}(r))$$

From the above analysis, we observe that there is a definite trade-off between $p_{pes}(t)$ and $p_{opt}(t)$. When we apply more and more tests, although $p_{opt}(t)$ decreases, $p_{pes}(t)$ increases so that L decreases. The critical number here is $p_{opt}(t)$ because an incorrect decision on a fault-free IC leads to the removal of that fault-free IC from the set of working ICs, whereas an incorrect decisison on a faulty IC will lead to the inclusion of that IC into the set of working ICs. In practice we should choose the parameters of the particular voting algorithm such that $p_{opt}(t)$ is made very small.

Average Test Length :

Average Test Length could be defined as the expected number of tests that has to be applied to the circuit before a fault could be diagnosed. From the above analysis, it could be seen that for our voting algorithm, the average test length will be given as

$$\sum_{t=0}^{\infty} t(1-c)^{t-1} c\overline{p}_{pes}(t-1)$$

and for the instance of our algorithm with a threshold of *one* this is equal to

$$TL_{av} = \frac{(1-p^k)}{c} + \frac{cp^k}{(1-c^k)^2(1-c)^2}$$

It is interesting to note that the above analysis tell us that we require a shorter average test length to diagnose a unit compared to the diagnosis using a test computer and this is because of the inherent unreliability present in our algorithm due to the pessimistic error probability.

Note that $TL_{av}(k->\infty) = \frac{1}{c}$.

We do not worry about the generation of tests and in this sense we can assume that a test generator will give us an "average number of tests " to work with. That means we can find out the number of voting units required on an average to diagnose the ICs. We would assume that we are given,

$$TL_{trav} = \sum_{t=0}^{\infty} t(1-c)^{t-1}c$$

tests which is equal to $\frac{1}{c}$. Then, for a specific accuracy in diagnosis of fault-free units (Locatability), we can calculate the number of voting units required. Assuming that $(1 - c^{k-1})$ is very small, we note that

$$L = (1 - p^k) + p^k(1 - c^{k-1})$$

so that

$$k = \frac{\log(1 - L) + \log(c)}{\log(pc)}$$

The above analysis gives us a clear idea as to the number of voters that need to be used given that a test generator generates an average number of tests for a particular accuracy of the voting algorrithm.

Pseudo-Random Model:

The above analysis, assumed that tests are applied in a random fashion and the same test can be repeated due to the randomness in the selection of the test involved. To make the analysis more precise we consider the scenario where the test is conducted without test vector duplication [16]. Assume that a circuit could be affected by any of the faults in the fault set $F = \{f_1, f_2, ..., f_m, \lambda\}$ where λ denotes the *null fault.* Let us also assume that the test set consists of r tests $T = \{t_1, t_2, ..., t_r\}$ as before. We could not, obviously, test any circuit exhaustively. Let us also assume that the r tests are generated randomly out of N possible tests. Let M_i be the number of tests, out of the possible N tests which will cover fault f_i. Noting that no test is repeated during the test process, we see that the probability that fault f_i is detected by exactly x of the r tests is given by the hypergeometric distribution function

$$P_i(x) = \frac{\binom{M_i}{x}\binom{N-M_i}{r-x}}{\binom{N}{r}}$$

The hypergeometric distribution is valid is valid only if the tests are picked up as sets (say a set of k tests out of N possible tests without repetition) of tests. But in reality, tests will be picked up sequentially with an earlier picked test being not picked again. (We would refer to the model which reflects this situation as the *Sequential Selection Model*). So, we note that modelling the practical situation using hypergeometric distribution is only an approximation. But, using an interesting property of hypergeometric distribution, we would show that the model derived gives an upper bound on the real situation.

From the equation given above, we note that the probability that none of the tests covered this fault f_i is given by

$$P_i(0) = \frac{\binom{N-M_i}{r}}{\binom{N}{r}}$$

It is intuitively clear that in the worst case, the coverage is limited by the fault which is most difficult to detect. Let $min_i M_i = M$ so that M gives the number of tests which covers the fault which is most difficult to detect. Then, an upper bound on the probability that a faulty unit is not detected to be faulty is given by

$$Y = max_i P_i(0) = \frac{\binom{N-M}{r}}{\binom{N}{r}}$$

We will use the above pseudo-random coverage model and develop equations for *pessimistic error probability* and *optimistic error probability*. The equations will be seen to involve a combinatorial enumeration to calculate the coverage of each individual test. It is intuitively obvious that the *expected coverage* of each individual test in the pseudo-random case will be smaller than that in the random case which will be equal to $\frac{M}{N}$. We will show that this indeed is true so that the coverage of each test on an average is upper bounded and the *Locatability* is lower bounded by the random model case.

Let $X(i)$ denote the probability that the ith test detects the fault when the tests are applied in sequence. As mentioned previously, using hypergeometric distribution to model the pseudo random case for the calcualtion of the coverages of individual tests will be an approximation. We would show that this distribution givbs an upper bound on the coverage without calculations using enumeration.

The expected coverage for the ith test will be given as

$$C_{exp}(i) = \frac{M - AV_{i-1}}{N - (i-1)}$$

where AV_{i-1} is the average number of tests that covered this fault out of the previous $i-1$ tests. AV_j will be given as

$$AV_j = 1(X(1) + X(2) + ... + X(j-1)) + 2(X(1)X(2) + ...) + ...+$$
$$(j-1)(X(1)...X(j-1)$$

The Locatability value, in this case, will be

$$L = (1-p^k) + p^k((1 - C_{exp}(1)^k)...(1 - C_{exp}(t)^k))$$

and the optimistic error probability will be

$$\frac{\binom{N-M}{r}}{\binom{N}{r}} \overline{p}_{pes}(r)$$

Let $P(i)$ give the probability that the last i tests, when the tests are selected in sequence, cover the fault. Then

$$P(i) = \frac{\binom{M}{i}\binom{N-M}{r-i}}{\binom{r}{i}\binom{N}{r}}$$

which means that the hypergeometric distribution model fits the sequential selection model when the last set of i tests in the sequential selection model cover the fault. Calculation of AV_j would involve calculating the probability of different permutations of $1, 2, ..., j-1$ tests covered the fault. But it can be shown that each of these probabilities for each of the permutations of m tests will be *lower bounded* by $P(m)$ (see *Appendix*) so that AV_{i-1} is lower bounded and consequently $C_{exp}(i)$ is upper bounded by the hypergeometric distribution case.

Thus for upper bound calculations of coverages of individual tests, we can use the hypergeometric distribution. Also, the mean of hypergeometric distribution is such that $AV_j = (j-1)\frac{M}{N}$, so that

$$C_{exp}(i) = \frac{M - (j-1)\frac{M}{N}}{N - (i-1)} = \frac{M}{N}$$

which is the same as for the random model. Thus, we can show that the *expected coverage* of individual tests is bounded from above by the value for the random model which is $\frac{M}{N}$.

The fact that the coverage of individual test is upper bounded by the random model tells us that the *Locatability* value is lower bounded by the random model case. This means that pseudo-random model will give a locatability value which is no less than the one given by the random model.

Average Test Length (Pseudo Random Model):

The average test length could be calculated as

$$TL_{avps} = \sum_{r=0}^{\infty} r \frac{\frac{M}{r}\binom{N-M}{r-1}}{\binom{N}{r}} \overline{p}_{pes}(r-1)$$

A closed form solution for the above does not seem to exist, but we note that an upper bound on the above will be

$$\sum_{r=0}^{\infty} r \frac{\frac{M}{r}\binom{N-M}{r-1}}{\binom{N}{r}}$$

which converges to $\frac{N+1}{M+1}$.

4 Communication of Test Results

The voting algorithms we have discussed so far enable the units to decide whether they are faulty or fault-free. Once this decision has been made, this information has to be conveyed to the outside world. Perhaps the simplest way to accomodate this requirement together with the interprocessor communication requirement for performing voting is to link the units together into a linear array, one end of which can be probed by an external tester. The interprocessor communications during voting can take place along this link, and the final diagnostic results can be transmitted to the external environment at the end of the diagnostic procedure through the probe link. While this method is not particularly fast for interprocessor communication, it is simple and should allow far more rapid testing than individual probing of ICs. If a more sophisticated communications structure is desirable, a method related to the adaptive testing strategy of [6] using a tree-structured network connecting the processors could be used. Such a technique would allow sublinear times for reporting the locations of faulty or good units in many cases and would thus be much faster than a single serial link. On the other hand, in the worst case the reporting of results can be slower than it is using the simple serial technique.

For the above adaptive strategy, we can show, that if p is the probability that a IC is faulty, then when $p \leq 0.5$, a certain decision strategy should be followed and when $p > 0.5$ a complement decision strategy should be followed.

The technique given in [6] borrows its idea from the adaptive tree walk probing followed in data communication protocols to give access to a sender to transmit messages

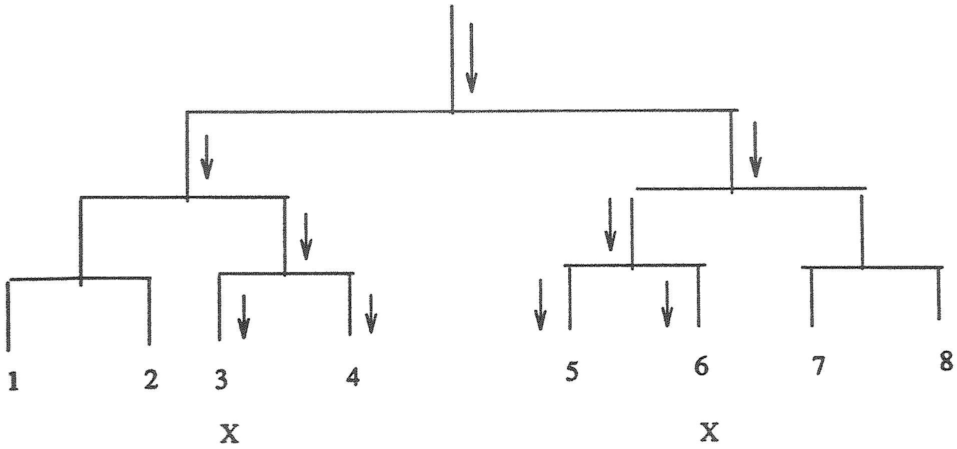


Figure 2: Group Probing of ICs

on to the network. In the subsequent discussion, we assume that each of the ICs have identified themselves to be faulty or fault-free using an instance of the voting scheme we have proposed. So, an outside tester needs to poll these ICs and find out their status. It has already been shown elsewhere that ICs have to be divided into groups of 2^j units each where $(1-p)^{2^j} > \frac{1}{2}$ for optimality. We derive an expression for the average number of inquiries required to unambiguously identify faulty and fault-free ICs. Each of the groups of 2^j ICs is polled in the fashion shown in *Figure 2.*

There is always one enquiry at the root node. If at least one of the ICs is faulty, the tree is divided into two sub trees and more inquiries are conducted and so on. The number of inquiries can be expressed in terms of a recurrence relation as follows. Let $T(n)$ be the average number of inquiries for a tree polling strategy on a set of n ICs. Let P be the probability that a IC is fault-free. That is, $P = 1 - p$. Also, $T(1) = 1$. Then,

$$T(2^j) = 1 + 2T(2^{j-1})(1 - P^{2^j})$$

because, once the root node is probed, the sub-trees are probed only if at least one of the ICs contained in the leaf nodes of the tree is faulty. Solution of this recurrence relation gives

$$T(2^j) = 1 + \sum_{i=1}^{j-1} 2^i \prod_{x=1}^{i} (1 - P^{2^{j-(x-1)}}) + 2^{j+1} \prod_{i=0}^{j} (1 - P^{2^{j-i}})$$

The strategy to be followed will be as follows. Let us assume that the ICs store a 1 in their status register if they find themselves to be faulty and store a 0 if they find themselves to

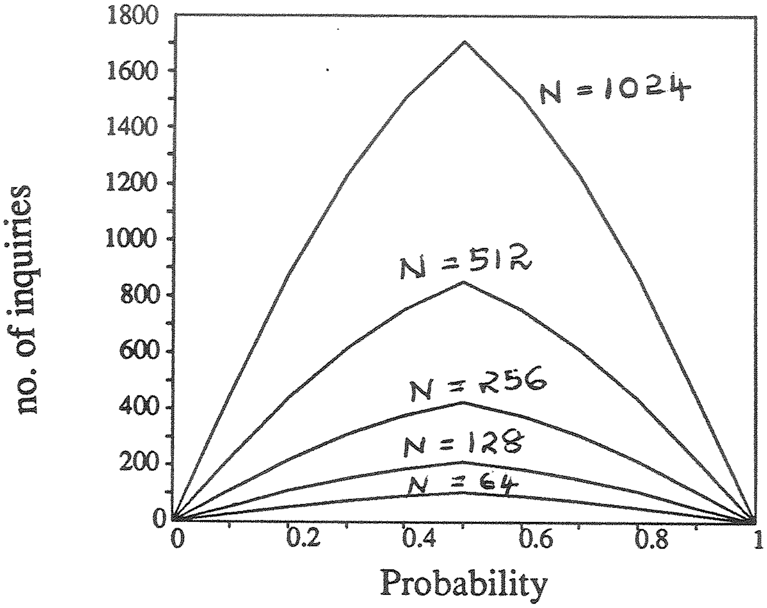


Figure 3: Number of Inquiries Required

be fault-free. If $p < 0.5$ we know that on an average, there will be more fault-free ICs than faulty ICs. So, the polling algorithm should function in such a way that ICs send the OR of their status registers up the tree so that the root of the tree receives a 1 if any of the ICs in the tree are faulty so that a search is made for the faulty ICs. If $p > 0.5$, we know that there will be more faulty ICs on the average. That means, the polling algorithm should search for fault-free ICs which will be fewer in number. For this case, the equation for the average number of inquiries is the same with P being replaced by p. In this case, the ICs should send the OR of the complement of their status register up the tree so that the root of the tree receives a 1 if any of the ICs in the tree are fault-free. When $p = 0.5$, it does not matter which strategy is followed, because both strategies will involve the same number of test inquiries on the average. The number of inquiries required with respect to failure probability is shown in *Figure 3.*

The above technique requires further investigation, we feel at present that simplicity of interconnect and the concomitant low hardware overhead for testing is of primary importance.

In the next section, we will consider the trade-offs involved in the application of the voting algorithm to both heterogeneous and homogeneous systems and also look at the efficiency of the algorithm with respect to traditional algorithms.

5 Voting Versus Traditional Testing

In this section, we will examine the equations derived in the last two sections and try to understand them in greater detail. We will show that the efficiency of the algorithm depends on the way the parameters are chosen and that there are trade-offs involved among them. To reiterate the results obtained in the previous sections we note the following

1. A voting scheme for testing can be can be applied to the production testing of circuits.
2. A priori correct results are not needed for testing.
3. A test set is assumed to be made up of multiple tests , each with a coverage equal to c. It is shown that the diagnostic resolution of individual tests affect the efficiency of the algorithm.

The analysis of the results from *Section 3* requires thought as the diagnostic resolution of tests have a large effect on the efficiency of the algorithm. From the analysis in the previous section, we observe that there is a definite trade-off between the probabilities $p_{pes}(t)$ and $p_{opt}(t)$. When we apply more and more tests, although $p_{opt}(t)$ decreases, $p_{pes}(t)$ increases so that L decreases. The critical number here is $p_{opt}(t)$ because an incorrect decision on a fault-free circuit leads to the removal of that fault-free circuit from the set of working circuits, whereas an incorrect decisison on a faulty circuit will lead to the inclusion of that circuit into the set of working circuits. In practice we should choose the parameters of the particular voting algorithm such that $p_{opt}(t)$ is made very small.

The equations we developed in *Section 3* are for a generalised class of voting algorithms with a threshold V. The only reason that we may need to use a threshold greater than *one* is that there might be very few failure modes so that for a particular test, there is a good chance that a faulty tester and one or more faulty testee have the same failure mode so that they give the same faulty result. In practice, for circuits of reasonable complexity, this will not be true. This fact is shown later in this section with numerical figures. In our analysis, we use simpler equations where it is assumed that no two faulty circuits give out the same faulty result for any test and also; we use a threshold value of *one*.

For an algorithm with threshold $k = 1$, *Figure 4* shows the variation of Identifiability with respect to the number of voting units for a fixed value of p which is set to 0.9 and

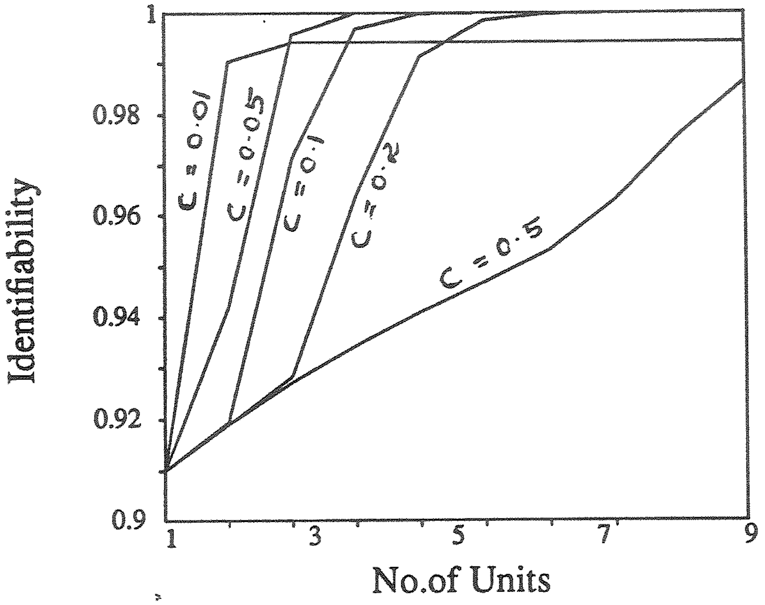


Figure 4: Variation of Identifiabilty with Voting Units when p=0.9

with the number of tests applied being 500 for different values of the coverage parameter c. For a small value of c, it is seen that I increases rapidly with n. All the curves tend to 1, but will saturate at some value < 1 for a fixed value of T.

The curve for $c = 0.01$ is seen to saturate at a value smaller than 1. For this value of c, it is clear that the *Optimistic Error probability* has saturated and does not decrease any more with increase in n. Only if the number of tests is increased will the I value increase in this case. This proves our point that all the parameters should be chosen correctly according to the circumstances.

Figure 5 shows the variation of I with n for various values of p with a fixed value of c which is set to 0.01 and for the application of the same number of tests. A higher probability of failure p leads to a higher value of I initially. As before, all the curves saturate at a value less than 1 and only an increase in the number of tests will lead to an increase in I.

Also, *Figure 6* shows the variation of I with respect to p for various values of c for a fixed value of n which is set to 2. A smaller value of c leads to a larger value of I for all values of p. But the saturation effect is seen in this graph also.

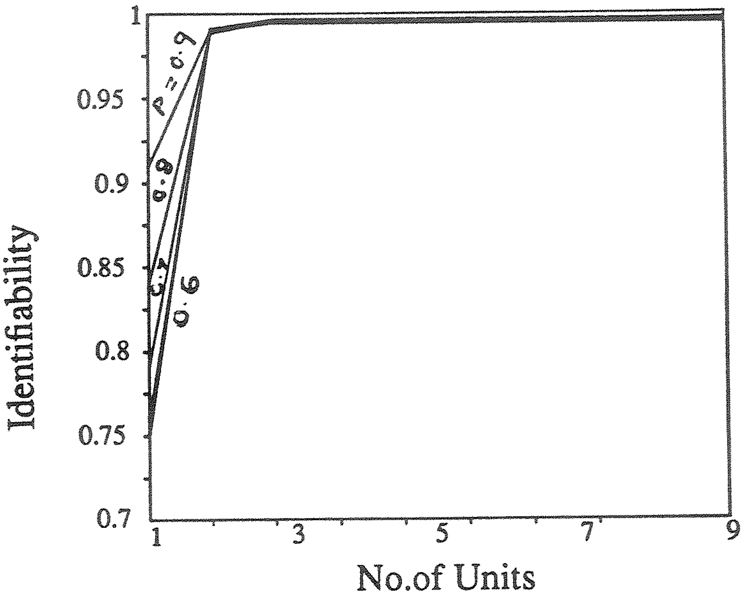


Figure 5: Variation of Identifiability with Voting Units when c=0.01

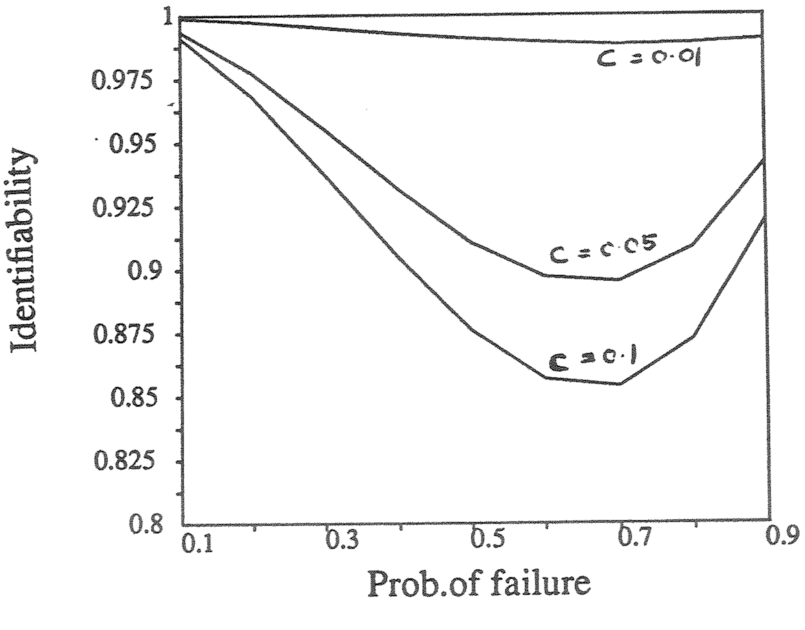


Figure 6: Variation of Identifiability with the Prob.of Failure when n=2

As far as the production testing problem is concerned, traditional testing methods involve a test computer which compares the output of the system under test with a known correct output for each of the tests. We observed previously, that two assumptions–one, that the test computer was fault-free and the other, that the computers that were used to compute the known correct results were fault-free, were made. We also mentioned that the above testing method using a test computer, in a way, belonged to the class of *Voting Algorithms* we have proposed. We will now compare this traditional algorithm with the general class of voting algorithms.

An instance of a Voting Algorithm is characterized by an evaluation predicate $V(C_a(t_m), C_1(t_m), ..., C_n(t_m))$ which is 1 iff atleast k of the $C_j(t_m)$, $j \neq i$ are equal to $C_i(t_m)$, and by the parameters k,n and r. The traditional test algorithm is an instance with the evaluation predicate $V(C_a(t_m), C(t_m))$ which is the equality predicate, which is true if the two values are equal and false otherwise, and parameters $k = 1$,$n = 1$ and some r.

Let the non-zero probability of the traditional test algorithm making an incorrect decision, either due to a fault in the test computer or due to a fault in the computers that evaluated the known correct output be ϵ, where $0 < \epsilon < 1$. It is obvious that the same equations derived above for $p_{pes}(t)$,$p_{opt}(t)$,L and I could be used to evaluate the efficiency of the traditional algorithm. In this case, the parameter p will take a value of ϵ. The value of ϵ will be much smaller than the probability of failure of an individual circuit C_j that should be used in our evaluation of the general class of voting algorithms that we have proposed, but it should be noted that the parameter n is a fixed value in the traditional algorithm, whereas in the general class of algorithms that we have proposed, the parameter n could be varied as availability and time permits. Also, our algorithm is distributed so that the diagnostic unit in each functional unit is simple compared to the complexity of the computer which would calculate the correct output and the test computer, which are used in a traditional algorithm. Let us assume that the probability that the diagnostic unit in the functional unit is faulty to be δ. Then $\delta << \epsilon$. From the above discussion, we note that in our algorithm, I which is the probability that that a circuit is correctly identified to be fault-free or faulty would asymptotically reach a value higher than that attainable with a test computer when the various parameters are chosen correctly. In a test algorithm using a test computer, it is obvious that the computer which evaluates the correct result and the test computer itself are the components which are prone to failure and they control the proper functioning of the test algorithm. We have taken note of this fact in our algorithm and have split up our diagnostic logic so that the algorithm is no more dependent on the proper functioning of some complex circuitry. Even though the diagnostic unit in each circuit is not perfectly reliable, due to its reduced complexity, it is much more reliable than a central test computer. In fact, if we assume that the diagnostic logic is perfectly reliable due to its reduced complexity and also that the circuit is complex enough so that no two faulty circuits within the set of n circuits have the same failure mode so that no two faulty circuits give out the same faulty result for any test, with sufficient number of voters in the comparison process and with sufficient number of tests, our algorithm would asymptotically give an I of 1. It should be noted that even if the diagnostic unit belonging to a particular circuit is faulty, it will affect the decisions made only on that particular circuit and not the decisions made on the other circuits under test. Contrary to this, a fault in the test computer or in the assumed correct result will affect the decisions made on all the circuits that are under test so that the fault is propagated globally.

Let us look at the equation for identifiability and assume that a test set consists of r tests. Then identifiability $I = (1-p)L + p(1 - p_{opt}(r))$. $I = 1$ when $L = \frac{1-p(1-p_{opt}(r))}{(1-p)}$. Obviously, the only solution to the above equation for $L \leq 1$ occurs when $p_{opt} = 0$ and the solution is $L = 1$. That means, $I \to 1$ only when $L \to 1$ and $p_{opt} \to 0$ simultaneously.

From the equation for $p_{opt}(t)$, $p_{opt}(t) = h0 + h1$

$$= (1-c)^r \overline{p}_{pes}(t) + R \sum_{i=1}^{r} \binom{r}{i} c^i (1-c)^{r-i} \overline{p}_{pes}(r-i)$$

When $r \to \infty$ we see that p_{opt} is bounded from above by R. When the circuit is of reasonable complexity so that f is high, then we can say that no two faulty realizations of C will have the same failure mode so that no two faulty circuits will give the same faulty output for any test. Then $R \to 0$ so that p_{opt} asymptotically reaches 0. Also, for a threshold of $k = 1$, $R = 1 - (\frac{pc}{f})^n$ and for a small value of $n(\frac{pc}{f})$, this could be approximated to $R = 1 - n(\frac{pc}{f})$. For example, for $f = 1000$, $n = 6$, $c = 0.1$, and $p = 0.9$, we get $R \approx 0.00054$ and this value decreases with the decrease in p.

From the equation for locatability L, we observe that if the test contains a set of r tests, then from the equation for p_{pes} when $r \to \infty$, $p_{pes} \to (1 - \sum_{i=k}^{n} \binom{n}{i} p^{n-i}(1-p)^i)$ so that Locatability $L \to \sum_{i=k}^{n} \binom{n}{i} p^{n-i}(1-p)^i$. From this we observe that when $n \to \infty$, $L \to 1$, so that we can simultaneously satisfy the conditions $L \to 1$ and $p_{opt} \to 0$ and this would make $I \to 1$ asymptotically. In practice, obviously we cannot increase r and n without bound. But we will show that we can get very good results with $n \geq 6$ and with $c = 0.1$. In fact, we need to us even a smaller value of n for a smaller c. That means, we should choose all our parameters properly so that we can get a result close to 1.

Let us make a numerical comparison between the traditional algorithm and our generalised class of voting algorithms. As mentioned previously, the traditional algorithm is an

$\epsilon = 0.01$

r p	1000	10000
0.5	3.84	4.61
0.6	4.09	4.91
0.7	4.33	5.20
0.8	4.56	5.46
0.9	4.78	5.73

Table 1: Number of Voters required for performance equal to that of the traditional algorithm

instance of the generalised class of voting algorithms with the number of voters n being 1 and the threshold value k being 1. The general equations for L and I for $k = 1$ reduce to the following.

$$L = (1 - p^n) + p^n(1 - c^n)^r$$

$$I = (1 - p)((1 - p^n) + p^n(1 - c^n)^r) + p$$

We have assumed that $p_{opt}(r)$ is very near zero, which will be true with a sufficiently high r and f (so that R is very small). Let I_{tra} and L_{tra} be the Identifiability and Locatability figures respectively, for the traditional algorithm. Then

$$L_{tra} = (1 - \epsilon) + \epsilon(1 - c)^r$$

$$I_{tra} = (1 - p)((1 - \epsilon) + \epsilon(1 - c)^r) + p$$

We equate I and I_{tra} to find out the point at which the performance of our class of voting algorithms equals the traditional algorithm with respect to the number of voters n required when the values of p,ϵ,c and r are kept constant. Also, we neglect the value of $\epsilon(1-c)^r$ in I_{tra} because it is very small due to the small value of ϵ and also because $(1 - c)^r$ will be very small with the application of a reasonable number of tests.

$$(1 - p^n) + p^n(1 - c^n)^r = (1 - \epsilon)$$

For rc^n sufficiently small, $(1-c^n)^r \approx (1-rc^n)$ and $1-r(pc)^n = 1-\epsilon$, so that

$$n = \frac{\log(\epsilon) - log(r)}{log(pc)}.$$

Table 1 gives the value of n as a function of p with $\epsilon = 0.01$, $c = 0.1$ and $r = 10000$. Note that with the application of 10000 tests each with a coverage of 0.1, we need $n = 6$ to achieve accuracy equal to that of the traditional algorithm when $\epsilon = 0.01$. In general we can use this approximation to easily determine the number of voting units for each test required to give better accuracy than a test computer with any given probability of failure.

The discussions in this section clearly show that a voting scheme is very efficient for the production testing of digital circuits. In the next section, we will apply our algorithm to the diagnosis problem of the functional units of a modular architecture at a wafer level.

6 An example Voting Diagnostic Algorithm for a Wafer Scale System

As an example of the implementation of our generalized voting algorithm to a specific architecture, we will now apply the algorithm to a modular architecture made up of functional units on a wafer. The explanation of the appplication of the algorithm assumes that the system is made up of homogeneous units, but it will apply to heterogeneous systems as well. The layout of the functional units on the wafer is as shown in *Figure 7*.

In this section we provide an example to clarify the intended application of the voting technique to wafer testing in terms of the overhead involved in implementing it and the speedups that it can provide over the traditional method. Of course, in a wafer-scale system, as is considered here, the communications overhead indicated would not be overhead at all since communications would be required in any case. Also, for ordinary wafers it should be noted that the communications wiring between processors can be expected to take less space than scribe lines.

In order to motivate the consideration of a heterogeneous modular architecture on a wafer, we look at how the functional units of such a modular architecture could be placed on a wafer and how the communication structure connecting these modules would look like. Katevenis [10] has provided a description of a Soft-Configurable Wafer Scale Integrated system where he uses the concept of a pipelined bus, where each partition(link) of the bus retains some information during a specific cycle and passes it on to the next partition, if

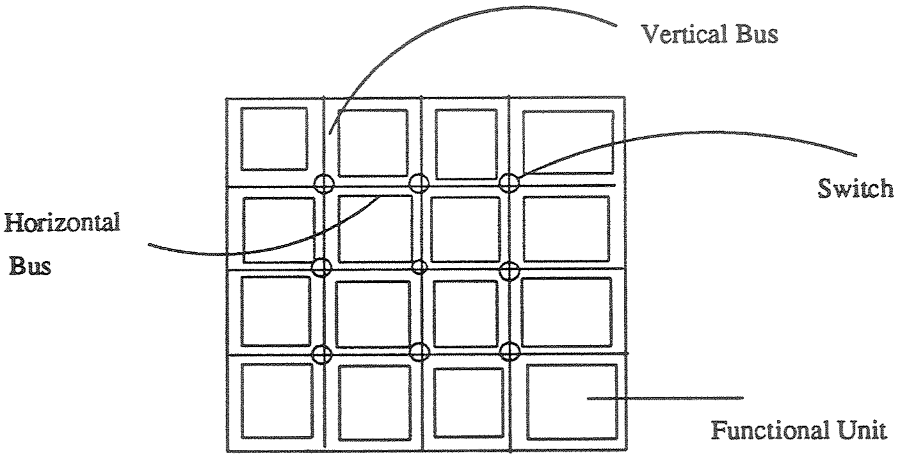
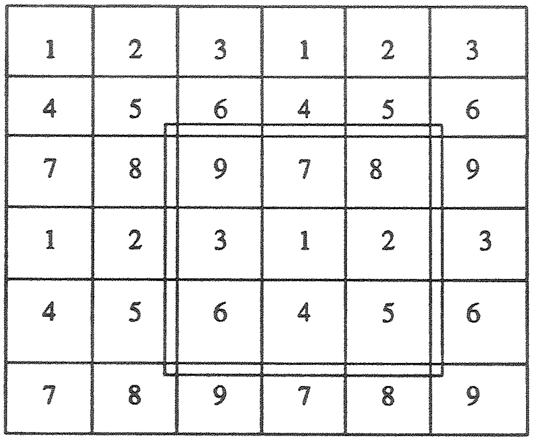


Figure 7: Two-Dimensional Bus-Connected Wafer

it is free, during the next cycle. Each partition of the bus is analogous to a pipeline stage. There are no buffers required, but the bus partitions themselves store the information. Such pipelining and the subsequent complexity of the hardware may not be justified if the communication is very localized. Further, some stages of the pipeline may fall vacant so that there may not be very efficient utilisation of the pipeline anyway.

In the case of a heterogenous system of units, we show that with clustering of functional units on the wafer, the communications on average take place between neighboring units which requires the information to pass through only a very small number of bus partitions. If this is the case, then a simple circuit switched scheme, where a circuit is formed through the partitions before the information is sent, might be easily implementable with very little overhead. The fact that self-timed communication protocols could be used would make the circuit switching idea practicable with very little time and logic overhead.

We look at a simple scheme where functional units of different types are clustered together on the wafer so that every functional unit has exactly one unit of all other types at a distance one from it. This would mean that there could be, on the whole, units of nine different types. Partitionable busses run horizontally and vertically through these units. There are switches between the partitions, but horizontal and vertical busses do not logically intersect so that a vertical partition cannot be connected to a horizontal partition



1	2	3	1	2	3
4	5	6	4	5	6
7	8	9	7	8	9
1	2	3	1	2	3
4	5	6	4	5	6
7	8	9	7	8	9

Figure 8: Clustering Scheme

and vice versa. This would mean that the switches need to be made up of simple pass transistors only. *Figure 8* shows the clustering scheme and *Figure 9* shows the range for any particular functional unit where the range is defined as the area on the wafer(functional units on this area) with which this particular unit can communicate. A unit in row i and column j can address the union of all the units on rows i-1,i and i+1 and columns j-1,j and j+1.

Regarding fault tolerance, we see that fault tolerance of the links is solely provided by the two circuit switched connections that could be formed between any two functional units. This would be adequate as link failures are very small as shown by Choi et.al[3]. There is no explicit functional unit redundancy as such in the system, but we see that if a unit is not functional, a communication meant for a unit of this type would be sent to another unit of the same type at a little distance away, so that the load on fault-free functional units increase with more and more faulty units. *Figure 10* specifies the range and the distance (the number of bus partitions required) between unit marked X and the other units in the range.

The load distribution algorithm would work in such a way that if the unit X (type 5) wants to make use of a functional unit of some type (say 2), thenit would try to use

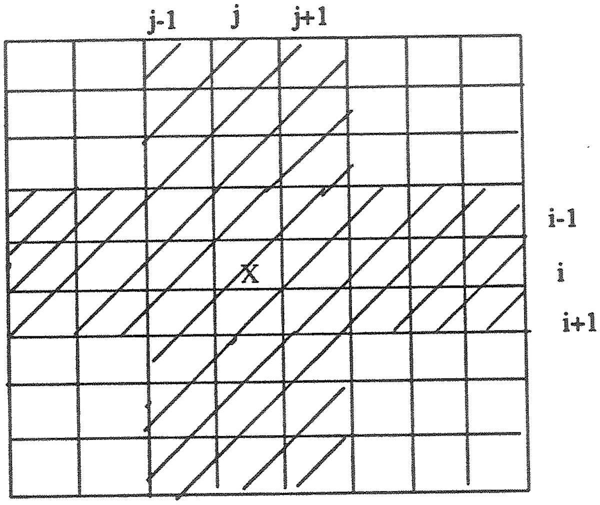


Figure 9: Range of a Unit

				4	4	4			
				3	3	3			
				2	2	2			
	4	3	2	1	1	1	2	3	4
	4	3	2	1	X	1	2	3	4
	4	3	2	1	1	1	2	3	4
				2	2	2			
				3	3	3			
				4	4	4			

Figure 10: Distance Between Units

a functional unit in its same cluster (distance 1) and if all these units are faulty, then it would go for units at neighboring clusters (distance 3) and if all these units are also faulty, then the next neighboring clusters (distance 6) and so on.

Below, we realize a bound on the average distance and subsequently the average number of bus partitions that a unit would use per communication. Let p be the probability that a particular unit is faulty. Then the average distance (if the above load distribution algorithm is followed) would be

$$\begin{aligned}
&(1-p) + p(1-p^4)3 + pp^4(1-p^4)6 + pp^8(1-p^4)9 + \cdots \\
&= (1-p) + 3p(1-p^4)(1 + 2p^4 + 3p^8 + \cdots) \\
&= (1-p) + 3p(1-p^4)\sum_{i=1}^{\infty} ip^{4(i-1)}
\end{aligned}$$

assuming that the wafer stretches infinitely on all the directions.

If we assume that the maximum number of clusters on all the directions from unit X is N, then the average distance would be

$$= (1-p) + 3p(1-p^4)\sum_{i=1}^{N} ip^{4(i-1)}$$

the second component of the equation for an infinite wafer converges to

$$\frac{1}{(1-p^4)^2}$$

and the second component of the equation for a finite wafer converges to

$$\frac{(1-p^{4N}) - Np^{4(N-1)}(1-p^4)}{(1-p^4)^2}$$

From this, we get the average distance to be

$$(1-p) + \frac{3p}{(1-p^4)}$$

for case 1 and

$$(1-p) + \frac{3p}{(1-p^4)}((1-p^{4N}) - Np^{4(N-1)}(1-p^4))$$

p	Case 1	Case 2
0.5	2.1	2.1
0.6	2.468	2.468
0.7	3.063	3.063
0.8	4.26	4.26
0.9	7.951	7.2268

Table 2: Average Communication Distance in the Presence of Faults

for case 2

Table 2 gives the average distance and hence the average wire length for various values of p for case 1 and also for case 2 with $N = 10$. We see that there is no difference between these values for $p \leq 0.8$ and the difference is seen only for $p = 0.9$. ie. the summation converges quickly for $p \leq 0.8$.

From these results, we see that even with a probability of failure of around 0.7, the average wire length (number of bus partitions) will be only around 3. It is interesting to note that the average wire length jumps up for a failure probability of 0.9 which shows a knee in the curve at that point. But a yield of more than 20% should be attainable most of the time.

A simple scheme could be thought of to realise the unit selection algorithm. After the initial testing for faults, all fault-free units should be updated with information regarding the nearest fault-free functional unit of each type which is available for use and each unit will address only these particular units to get some work performed by a functional unit of a particular type. Similarly, a simple scheme for fault-free route selection would be to try one of the two circuit switched connections possible and if this could not be formed, the other connection should be tried. Note that separate access tokens have to be passed around each row and column busses, so that a unit could attempt to form a switched circuit on a bus only if it has the access token for that particular bus. If none ofthe two switched circuits could be formed, then the unit should be able to direct its load to the next nearest fault-free functional unit of the required type with a fault-free path.

With the above scenario, a set of working functional units could be dynamically configured to form a working system. In this sense, a complete system need not be confined to a particular part of the wafer, but could spread out over the wafer. The mapping algorithm should work in such a way that a working system is mapped within the least area possible to avoid long communication delays. Our test algorithm identifies the working units that could be configured using such a configuration algorithm as a post test step.

Consider a wafer containing a square array of $\sqrt{N} \times \sqrt{N}$ homogeneous *Integrated Circuit Blocks* where each integrated circuit block contains the IC itself and also additional circuitry for testing. *Fig 6* shows the configuration of these blocks on the wafer with additional communication links that have to be configured on the wafer for testing purposes. All circuits must perform their IO at the rates expected during their normal operation. This may appear to require some complexity in the communications hardware on the wafer, although the communications could be done serially if the bus lines could be made to work fast enough and if the amount of data to be transmitted between circuits could be minimized. For this purpose, it may be useful to take the actual circuit outputs and route them through a communications module which does signature analysis and only transmits a compressed version of the data over serial communication links to effect the comparisons. This sort of scheme would present a full-speed environment to the circuit under test without undue silicon penalties for complex communication links.

A second serial link is required for clocking the units. It is assumed that the ICs are externally clocked by the same clock which will clock the running of the test algorithm. This would mean that the clock that would be used would run at the operational clock speed of the ICs themselves thereby testing the ICs at that speed.

An IC block consists of three sub-blocks other than the IC itself as shown in *Figure 11.* The *Signature Generator* block generates the signatures of the outputs of the ICs during the testing process. The voting unit consists of a comparator and a Vote Counter for keeping track of the *vote count* by which we mean the number of signatures from the other ICs which matched the signature produced by the corresponding IC for a particular test. There is also a single status bit in each voting unit and all the status bits in the wafer are connected to form a serial shift register. At the end of each test the status bits are set if the vote count crosses the threshold value which will be prestored in the *Threshold Register.* The communication monitor essentially serves as an interface between the bus

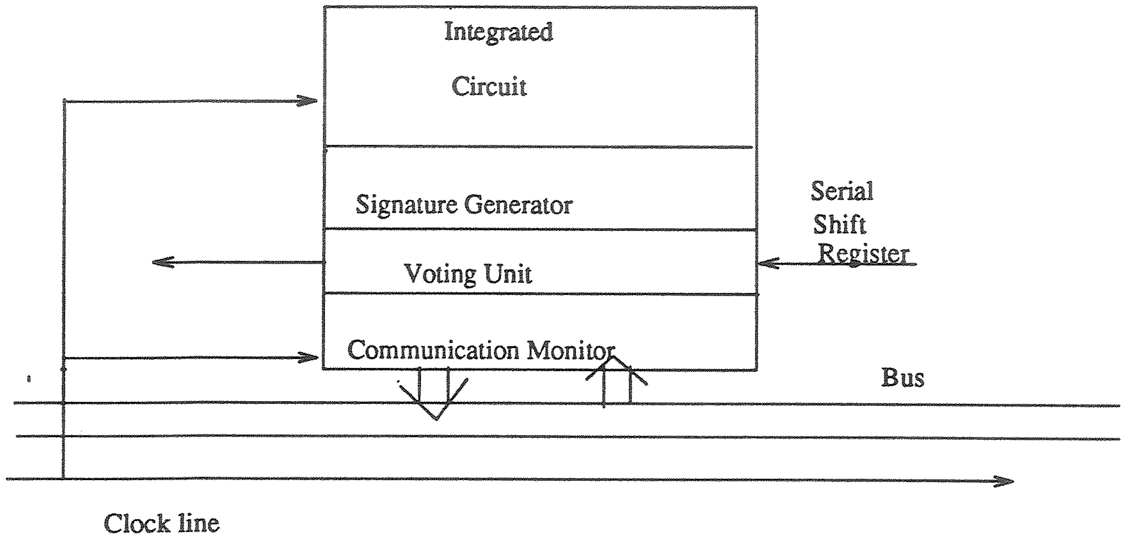


Figure 11: IC Block

partition and the other sub blocks in the IC block.

It is clear that the signature generators will require very little hardware. *Figure 12* shows the hardware involved with the design of the voting unit and obviously, the overhead is quite small. Also, the communication monitor hardware is expected to be quite trivial. It should be noted that the extra hardware used for production testing could also be used to test for post production failures in case of parallel systems which are made up of a multitude of these ICs. In that situation the hardware overhead is essentiallyzero.

Each test input is broadcast to all the IC blocks. ICs in each of the clusters compute results for the tests. Once a signature has been computed, each IC in a cluster, in turn, broadcasts its signature to the other IC blocks in the cluster, and these are observed and compared with their own signatures by the other IC blocks. (*Figure 13*). Such partitioned voting scheme is also shown in *Figure 14*. ICs for which the threshold is crossed are deemed fault-free for that particular test. This is done by setting the status bit to zero. ICs for which the threshold is not crossed even after all the signatures have been matched are deemed faulty by setting their status bits to one. Once the status bit is set to *one*, that particular IC does not vote on its own status any more, but it still generates signatures for broadcast to the other ICs in the partition. Once all the tests have been applied, the

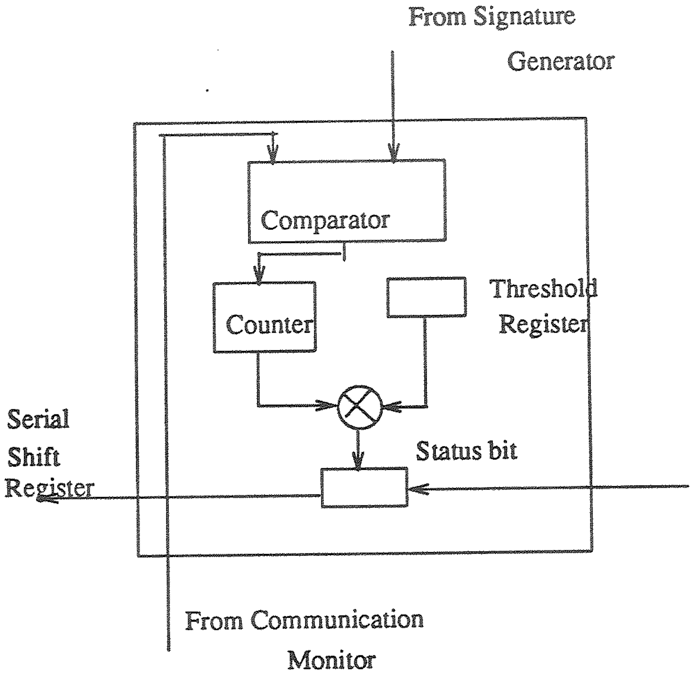


Figure 12: Voting Unit Hardware

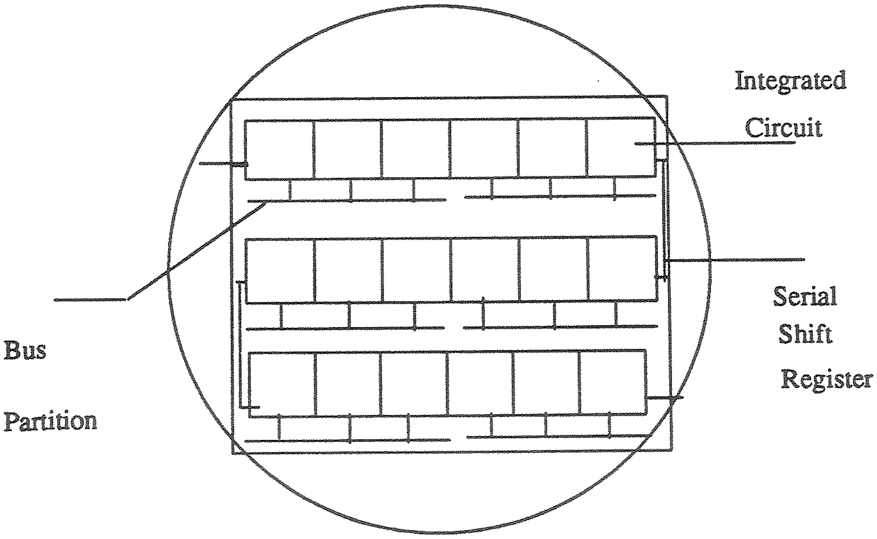


Figure 13: Clustered Testing of ICs

status bits are serially shifted out, thereby conveying the fault status of all the ICs to the outside world.

For a single test, the time taken for the test to be completed will be $t(1+n)$. assuming the broadcast time for the tests (the set up time) to be Lt for L tests, the total time is $T_t = Lt(1+n)$. If we assume that NLt is the time taken for a sequential algorithm like the traditional test algorithm using a test computer, then then the speedup is

$$S_p = \frac{NLt}{L(1+n)}$$

$$= \frac{N}{(1+n)}$$

Thus our algorithm's performance is linear in L, the number of tests, as is the traditional method, but it does *not* depend on the number of ICs being tested as is the case for probe testing. It should be noted, however, that reporting the test results for the whole wafer to an external monitor may well take time linear in the number of ICs tested, but this time will be orders of magnitude smaller than that required by the mechanical motionsof a probe tester.

To summarise, our voting algorithms have the following characteristics:

1. IC voting units observe results computed by other ICs and based on a threshold make decisions as to the fault status of their ICs.

2. All those ICs for whom the threshold could not be reached for a particular test are deemed faulty.

3. ICs, once deemed faulty, do not vote on their own fault status any more.

4. Each IC voting unit computes its vote count separately and there could not be a tie as the count is with respect to its own result.

5. Will identify both fault-free and faulty ICs with high efficiency when sufficient number of tests are applied.

The implementation of the partitioned voting algorithm on the wafer could be represented in the form of a pseudo-code as below.

```
/* status and present status are boolean variables */
```

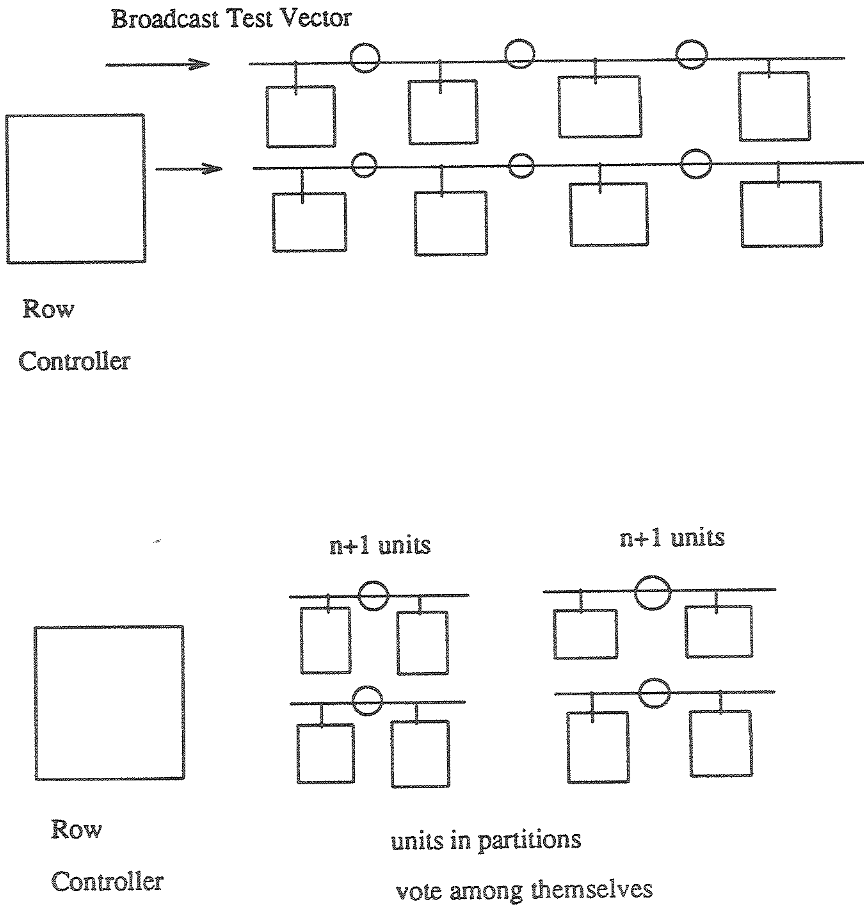


Figure 14: Partitioned Voting

```
    /* Test token denotes the test input. Decision token denotes the token which is passed
around the ICs to give them access to broadcast their results on to the bus partition*/

    /*after the ICs have received a single test token and calculated their results */
    for each unit do {
        if not decision token {
            for each value on bus do {
                if status=0
                compare/set presentstatus;
            }
        }
        else {
            broadcast result on the bus;
            wait for all IC Blocks to finish making a decision;
            pass the decision token to the IC block.
        }
        if at the end of the test for a particular test token
            if presentstatus=1 status=1;
        }
/* for all IC Blocks */

    /* at the end of the test, if status=1 then the IC is faulty */
```

7 Conclusion

We have proposed the solution of a number of problems inherent in the use of probe testing for at-speed testing of integrated circuits by the use of a simple, comparison-based scheme using circuitry built into the IC wafer itself. Analysis of a reasonable probabilistic model of the behavior of this method indicates that it performs at least as well as the traditional method using a test computer in terms of the accuracy of the distinctions made between faulty and fault-free circuits and of the speed with which they are produced. The analysis shows that essentially all of the fault-free units can be distinquished from faulty units with very high probability.

There are many potential advantages to using voting schemes for production testing in place of traditional probe testers. Accuracy of the test results equal to or better than that achieved with test computers is attained even with relatively unreliable voting units as a result of allowing many such units to participate in each decision as to whether a test has been passed or not.The effects of faults in the diagnostic hardware are localized since each unit has its own diagnosis logic which determines results only for that unit. Also the diagnostic units are simple compared to the complexity of a test computer and thus inherently less failure-prone. The fact that voting rather than test computers is used also means that we no longer need to assume that any computer produces *a priori* correct results for each test vector. The fact that only comparisons are used to determine correctness means that a test vector set could actually be stored once on a wafer and used to test all the circuits on the wafer without concern about whether a particular test vector is modified by a production flaw in the area of the test vector storage itself. Since a modified test vector in a properly designed system should stillbe a valid test vector and need not match any already computed result, a test sequence modified by some production flaws should still produce an accurate diagnosis. If randomly generated tests are to be used, then random number generators could be fabricated into the wafer which generate a test vector sequence automatically, resulting in a simple built-in system which performs the test generation as well as test application automatically.

The disadvantage of the voting approach is that communication paths among the circuits are required, along with some diagnostic and communication logic which represents hardware overhead on the wafer which may adversely affect yield. However, if a simple communication scheme is used, this overhead should be small, and in fact it is likely cause a smaller increase in the silicon area devoted to an individual circuit to be tested than does the introduction of LSSD circuitry into such a circuit. It is to be hoped that the inclusion of voting circuitry can have as much value in supporting effective production testing as LSSD techniques do in debugging physical designs.

Of course, the entire approach outlined here fundamentally relies on the assumption that production defects are independent random events. Given that we envision separately performing parametric testing, this assumption may be valid for at-speed testing. Certainly such sources of error as flaws in the silicon substrate and the effects of dust particles can be modeled as random independent errors, but such systematic problems as mask misalignments or failures of the weakest parts of a design when processing parameters are marginal cannot be. Nevertheless, it appears that this approach has much to recommend it in terms of testing efficiency, and the savings that may be realized in the testing stage of IC production may well motivate some changes in earlier stages of the production process to eliminate sources of systematic error so that voting methods can be used.

References

[1] M.A. Breuer and A.D. Friedman, *Diagnosis and Reliable Design of Digital Systems*, Rockville, MD. Computer Science Press, 1976.

[2] J.T. Butler, "Speed-Efficiency-Complexity Tradeoffs in Universal Diagnosis Algorithms," *IEEE Trans. Computers*, August 1981, pp. 590–596.

[3] Y-H. Choi, D.S. Fussell and M. Malek, "Fault Diagnosis of Switches in Wafer-Scale Arrays," ICCAD '86 pp. 292–295.

[4] Y-H. Choi, S. Han and M. Malek, "Fault Diagnosis of Reconfigurable Systolic Arrays," ICCD '84 pp. 451–455.

[5] Y-H. Choi, D. Fussell and M. Malek, "Token Triggered Systolic Diagnosis of Wafer Scale Arrays," Proceedings of the International Workshop on WSI, Southampton, England, July 1985 pp. 246–258.

[6] V. Cherkassky and L. Kinney, "A Group Probing Strategy For TestingLarge Number of Chips," Int'l Test Conference 1986, pp. 853–856.

[7] A.T. Dahbura, K.K. Sabani and L.L. King, "The Comparison Approach to Multiprocessor Fault Diagnosis," *IEEE Trans. Computers*, March 1987, pp. 373–378.

[8] P.T. Desousa and F.P. Mathur, "Sift-out modular Redundancy," *IEEE Trans. Computers*, July 1978, pp. 624–627.

[9] E.B. Eichelberger and T.W. Williams, "A Logic Design Structure forLSI Testability," Proc. 14th Design Automation Conf. 1977, pp. 462–468.

[10] M.G.H. Katevenis and M.G. Blatt, "Switch design for Soft-Configurable WSI Systems," 1985 Chapel Hill Conferenece on VLSI, pp. 199-219.

[11] Losq. J, "A Highly Efficient Redundancy Scheme: Self-Purging Redundancy," *IEEE Trans. Computers*, June 1976, pp. 569–577.

[12] S. Rangarajan, "Distributed Voting Algorithms for Fault Testing," M.S Thesis, Dept. of Electrical and Computer Engg., UT-Austin, May 1987.

[13] D.P. Siewiorek and R.S. Swarz, *The Theory and Practice of Reliable System Design*, Bedford, MA, Digital Press 1982.

[14] J.E. Smith, "Universal System Diagnosis Algorithms," *IEEE Trans. Computers*, May 1979, pp. 374–378.

[15] von Neumann. J, "Probabilistic logics and synthesis of reliable organisms from unreliable components," *Automata Studies*, in *Annals of Mathematical Studies*, No. 34, C. E. Shannon and J. McCarthy, eds., Princeton University Press (1956), pp. 43–98.

[16] K.D. Wagner, C.K. Chin and E.J. McCluskey, "Pseudorandom Testing," *IEEE Trans. Computers*, March 1987, pp. 332–343.

Appendix

To show that P(i) lower bounds the probability of coverage of a fault by any permutation of i tests in the sequential selection model.

We have noted that the hypergeometric distribution model fits the sequential selection model when the last i tests in the sequential selection model cover the fault. That means, the probability that that particular permutation where the last i tests out of m tests applied cover the fault in the sequential selection model is the same as $P(i)$.

The intuitive idea behind our claim as follows. If the tests that cover the fault occur earlier in the sequence, then the probability that a subsequent test does not cover that fault is higher because that the total number of tests remaining are the same where as the tests that cover the fault in the test set are less in number than before.

$$P(i) = \frac{\binom{M}{i}\binom{N-M}{m-i}}{\binom{m}{i}\binom{N}{m}}$$

Let us look at a case (say) when $i = 3$. Then

$$P(3) = \frac{\binom{M}{3}\binom{N-M}{m-3}}{\binom{m}{3}\binom{N}{m}}$$

$$= \left(\frac{N-M}{N}\right)\left(\frac{N-M-1}{N-1}\right)\cdots\left(\frac{M}{N-(m-3)}\right)\left(\frac{M-1}{N-(m-2)}\right)\left(\frac{M-2}{N-(m-1)}\right)$$

Let us look at the probability that some other permutation of 3 tests out of m tests cover this fault. Let the tests that cover the fault be (say) the first, second and fourth. This will be

$$= \left(\frac{M}{N}\right)\left(\frac{M-1}{N-1}\right)\left(\frac{N-(M-2)}{N-2}\right)\left(\frac{M-2}{N-3}\right)\cdots\left(\frac{N-(m-6)-(M-3)}{N-(m-3)}\right)$$
$$\left(\frac{N-(m-5)-(M-3)}{N-(m-2)}\right)\left(\frac{N-(m-4)-(M-3)}{N-(m-1)}\right)$$

The above equation could be rearranged to give

$$= \left(\frac{N-(m-6)-(M-3)}{N}\right)\left(\frac{N-(m-5)-(M-3)}{N-1}\right)\left(\frac{N-(M-2)}{N-2}\right)$$

$$\left(\frac{N-(m-4)-(M-3)}{N-3}\right)\cdots\left(\frac{M}{N-(m-3)}\right)\left(\frac{M-1}{N-(m-2)}\right)\left(\frac{M-2}{N-(m-1)}\right)$$

The denominator is the same in both the equations. Also, the terms M, $M-1$ and $M-2$ occur in the numerator of both the equations. But it is easy to verify that the product of the other terms in the numerator of $P(3)$ is smaller than that of the other equation. It is also easy to verify that the above is true of any permutation of any i faults when compared to $P(i)$.

Thus $P(i)$ is seen to lower bound the probability of coverage of a fault by any permutation of i faults in the sequential selection model.