Data Mining Integrated Circuit Fails with Fail Commonalities

Leendert M. Huisman, Maroun Kassab, and Leah Pastel

IBM Microelectronics, Essex Junction, VT

Abstract

We describe ways to use fail data from many failing Integrate Circuits (ICs) to determine which ICs failed because of similar causes, rather than to determine the cause of each individual failing IC. The purpose of finding clusters of similarly failing ICs is to focus on systematic defects, and to de-emphasize random ones. Once large groups of similarly failing ICs have been identified, a selection of the ICs in each group can be diagnosed using standard diagnostic routines.

1 Introduction

Part of the normal manufacturing process of ICs, is verifying that the ICs are free of defects. The immediate goal of this testing step is the identification of all defective chips. A secondary goal is collecting fail data for each defective IC. These fail data can then be used later, if so desired, to identify the defect that caused the fail [1].

The combined fail data for the set of defective ICs can also be used to obtain information about the defects, however, if it was collected in sufficient volume, and with sufficient detail. For example, the effectiveness of the various steps in the test sequence in detecting defects can be obtained, and fail probabilities can be determined for various embedded objects, like embedded RAMs [2].

Another obvious use of fail data is to diagnose the failing ICs using more or less standard diagnostic techniques [1]. Since the number of failing chips is large, the diagnosis has to be fast. Rapid volume diagnosis has been used to identify repetitive failure mechanisms (for example, in [3]). It can become even more effective by combining it with the results of inline inspection (for example, [4] and [5]).

But fail data can also be used in a less quantitative way. They can be used as a signature, so to speak, of the underlying defect, the one that caused the fail. Such signatures can be defined for each failing IC, using the raw fail data or some kind of summarization of them. Using the fail data itself as a signature, rather than a diagnostic result, looses accuracy, of course, but has the advantage of greatly improving the speed with which the signatures can be obtained. Also, since no expensive diagnosis is required, all the fails can be used, and no restrictions need to be imposed on the number of failing patterns that will be considered, or the types of faults that will be employed in the diagnosis.

This use of fail signatures is well known in dictionary based diagnosis [1], [4], in which fail signatures are compared to precalculated signatures, one for each fault in a fault list, to find the ones that best match the actually observed signatures. This paper focuses on the observed signatures themselves, without comparing them to precalculated ones as is done in dictionary based diagnosis.

Signature analysis has also been used recently in a different but related context, namely on the results of Iddq measurements [6]. The signatures used in the present article are largely based on the results of logic testing, and are discrete. They also correspond to definite fails, while the distinction between passing and failing Iddq signatures is based on some, more or less arbitrary threshold.

Using fail signatures is important because the signatures can be used to compare different chips, or the same chip under different test conditions, and these comparisons may then indicate if the fails were caused by the same defect mechanism. This type of comparison will be called commonality analysis. Once a way is found to reliably compare different ICs, the failing ones can be clustered into groups of ICs that seem to have failed because of the same or similar defects.

One reason for attempting such clustering is that, if all the ICs in a single group did fail because of similar defects, information like occurrence probabilities of such defects is available immediately. In addition, diagnosis can then be targeted to the more frequent defects.

More importantly, clustering failing ICs focuses on the presence of systematic defects. The Null hypothesis of manufacturing is that there are no systematic defects, that all defect producers act with equal strength on all wafers, and produce defects randomly and independently of each other. If true, commonality analysis would not succeed, or, at least, would fail to find any large clusters. Consequently, finding large clusters gives evidence of some systematic defect.

The goal, therefore, of comparing fail signatures, and of clustering failing ICs using their signatures, is to uncover the presence of systematic defects, if any exist.

0-7803-8580-2/04 $20.00 Copyright 2004 IEEE

	latch 1	latch 2	...	latch n-2	latch n-1	latch n	# latches	pass/fail
pattern 1	x		...	x			l_1	1
pattern 2			...				l_2	0
...	...	...	...	...	...	...	...	...
pattern m-2		x	...	x			l_{m-2}	1
pattern m-1	x		...	x		x	l_{m-1}	1
pattern m			...				l_m	0
# patterns	p_1	p_2	...	p_{n-2}	p_{n-1}	p_n	N	
distinct fail	1	1	...	1	0	1		

Table 1 Deterministic Test Results

Once identified, further diagnosis can provide more information about them. We will not address the subsequent diagnosis, since diagnostic techniques have been well described in the literature (for example, [1], [8] and [9]).

In this article, we will investigate commonality between ICs, whether on the same wafer or in the same lot. The fail data that will be employed to define signatures are the lists of failing measures. Measures are comparisons between observed and expected values at Primary Outputs (POs), and failing measures indicate that the corresponding observed and expected values differ. When a PO corresponds to the output of a scan chain, the measures are comparisons between observed and expected latch contents. Because the lists of failing measures may be very large, various ways to summarize them without losing essential information will be studied.

The basic steps to be taken are fairly straightforward. First, define a measure of commonality between the fail data of two failing ICs. Second, identify clusters based on the commonality measure. Commonality analysis can be performed on all ICs failing any of the deterministic tests, or can be done in a more detailed manner by differentiating between various environment parameters, like voltage and temperature. Finally, develop suitable cluster signatures to identify future occurrences of the same underlying defects.

This type of analysis is performed on large numbers of ICs. Consequently, the analysis has to be fast, which limits the amount of work that can be performed on the fail data. It is the opposite of logic diagnosis, in which accuracy of the result is the driving force, and performance can be sacrificed to it.

This article is divided into three parts. First, various types of signatures will be described in detail for various types of test results. Signatures can be defined for both single ICs and for wafers, and we will briefly describe some wafer signatures and their uses. Second, abstract commonality measures will be defined for ICs, as well as clusters. Finally, a clustering technique will be presented, and some results of applying such clustering on selected sets of fail data will be discussed.

2 Logic fails

The abstract, and rather brief description of fail signatures can be made clearer by a detailed example. We will describe various fail signatures that are suitable for deterministic, that is, scan based tests. Such tests are generally used to test the internal logic circuitry of a chip, as opposed to, for example, memories, which are tested by specific algorithmic tests.

The result of the deterministic tests is symbolically shown in Table 1. The rows in this table correspond to the patterns that were applied during the test, and the columns to the latches and POs whose values were inspected as part of the test procedure. A latch that, after a given pattern was applied, was found to have an incorrect value is marked by the symbol ☒ in the cell corresponding to that pattern and that latch.

The table contains two auxiliary columns and two auxiliary rows. The column labeled "# latches" contains, for each pattern, the number of latches that were found to have incorrect values, with 0 indicating that the pattern did not fail. The column labeled "pass/fail" merely shows whether the value in the previous column is 0 or not. The two auxiliary rows have similar functions. The one labeled "# patterns" contains the number of patterns that caused a particular latch to have an incorrect value, while the row labeled "distinct fail" shows whether the number of failing patterns is 0 or not.

Finally, the total number of failing values that were observed in any of the failing patterns is N, and is the col-

umn sum of the values in the "# latches" column, and is also equal to the row sum of the values in the "# patterns" row.

The contents of Table 1 represent all the available fail data, without omissions, but in a way that is more suitable for discussing usable signatures. Various types of signatures can now be defined using this table. The most obvious one is to keep the complete table. This is of course the best possible signature, but has the drawback of being rather voluminous.

Two approaches will be discussed here. The first one uses only information in the table, the second one uses the table, and, in particular the "distinct fail" row, as a starting point for further analysis. We will consider both in turn. We will close this section with a brief discussion of some alternate signatures that may also be useful in certain circumstances.

2.1 Signatures based on fail data only

A better choice than all the data in Table 1 is to summarize the fail data using one or more of the auxiliary rows and columns. In fact, the column labeled "pass/fail" is a well known summarization, used in dictionary based diagnosis [1]. The assumption in that type of diagnosis is that the vector of values in the "pass/fail" column can be used to identify the underlying defect almost uniquely. Upon further experimentation, it usually becomes clear that this vector gives a very poor diagnostic resolution, and is not used in practice. For the same reason, it is not a good signature to use in commonality analysis.

A considerably better choice, as it turns out, is the vector of ones and zeros in the "distinct fail" row (see Table 2),

	latch 1	...	latch n	# latches	pass/ fail
pattern 1	x	...		l_1	1
...	...	...	...	...	...
pattern m		...		l_m	0
# patterns	p_1	...	p_n	N	
distinct fail	1	...	1		

Table 2 Signature Based on Unique Fails

and this vector will be called the unique fails signature. A clearly even better signature is the vector of values in the "# patterns" row, possibly augmented with the vector of values in the "# latches" column (Table 3). It cannot be worse than the unique fails signature, and it requires only marginally more storage. This vector, or the combination of the two vectors, will be called the marginals signature.

	latch 1	...	latch n	# latches	pass/ fail
pattern 1	x	...		l_1	1
...	...	...	...	...	...
pattern m		...		l_m	0
# patterns	p_1	...	p_n	N	
distinct fail	1	...	1		

Table 3 Signature Based on Marginals

With these choices for signatures, choosing the appropriate commonality measures is straightforward (Chapter 4). Note that, in all cases, patterns that did not fail and latches that never contained fail data do not contribute to the commonality measure. In other words, the commonality measures are based solely on fails that occur in at least one of the two signatures.

2.2 Signatures based on backtracing

The unique fails signature works rather well in practice, in that a high commonality value based on this signature often indicates a common defect cause. The reason for this is probably that the defect that caused the fail is likely to be close to the latches that contained the incorrect values, because fault effects caused by the defect flow along signal wires that do not commonly cross large distances over the chip. In other words, when fault effects propagate away from the location of the defect, they will usually not travel far before they are stored into latches.

If a fault effect is stored in a latch, this latch will contain an incorrect value at the conclusion of the application of the test pattern. Such a latch is commonly called a failing latch, although, usually, the latch as a logic circuit is defect free.

If fault effects do indeed stay close to the originating defect, then it makes sense to follow the fault effects in the opposite direction: start from the failing latches and trace through the logic backwards until primary inputs, embedded memories, or other latches are encountered, and store all the nodes that were encountered during the tracing in a list.

Backtracing through combinational logic is well described in the literature [1], and will not be discussed here. During the backtrace, all the encountered nodes are stored in a list. The backtrace is repeated for each failing latch and each failing pattern, and, each time a node is encountered, that node entry in the list is incremented by 1 (or by some other value depending perhaps on the backcone, the number of failing latches, etc.).

The resulting signature is a list of (node, v) pairs, in which v is the number of times this node was encountered during the backtraces. A high v value shows that the corresponding node is in the backcones of many failing latches.

The v values, therefore, form a rough estimate of the likelihood that the defect is located on or near any of the nodes in any of the backcones.

It is useful to compare this signature with the result of a crude form of diagnosis that is sometimes employed, called intersection. In intersection, backcones are obtained as above, but instead of incrementing counters, the backcones are kept as sets and the intersection is taken of all these sets. The result is a set of nodes that are in all the backcones.

The theory behind that form of diagnosis is that only nodes in the intersection can be the location of the defect, because otherwise fault effects from the defect could not have propagated to all the failing latches. Unfortunately, not all defects affect single nodes. Bridges, for example, affect at least two, and the latches downstream from one leg of the bridge may not be the same as the ones downstream from the other leg. Consequently, intersecting backcones from the failing latches is likely to result in an empty set.

Using the backcones based signature, however, circumvents this problem. It lists all the nodes ever encountered in any of the backtraces, but it ranks them according to how often they were seen. The group of nodes most often encountered form then a generalization of the intersection, one that does not suffer from the problem of potentially being empty.

2.3 Signatures based on cells

Instead of keeping track of which nodes or blocks are encountered during backtracing, one can also notice their functional properties. These can include the logic function, drive strength, power level, delay times, etc. All these details can be retrieved using the cell name of the block, which is a reference to a specific cell in the design library of which this block is an instance.

Monitoring cells rather than nodes during backtracing is sometimes useful when the defect is not one that impacts a specific instance of a library cell, but, instead, one that impacts the library cell itself; perhaps a defect prone layout style, an underpowered driver, or any other design flaw that will affect all instances of that cell.

The resulting signature is very similar to the one discussed in the previous section, except that now the n component of the (n, v) pairs is not the name of a node or a cell instance, but the name of the cell itself. Signatures that have high counts of certain cells hint at problems with that cell, rather than at some point defect somewhere on the chip.

3 Other signatures

In some cases, fail measures can be interpreted as passes or fails of well defined objects, like embedded RAMs, with a failing measure indicating that the corresponding object failed some part of the test sequence [2]. This is particularly true when the pass/fail status of the embedded object is recorded in a scannable latch, because then that pass/fail status can be retrieved as if it were the result of some logic test. Commonality analysis can then be performed on either the failing measures themselves, or, if appropriate, on the corresponding failing objects. This leads to pairs (n, v), in which n refers to the object and v equals 1 or 0.

This paper focuses on IC level signatures, but, clearly, the idea of signatures is not restricted to single ICs. For example, a wafer level signature can be defined for embedded objects. Such a signature could take the form of a list of (n, v) pairs, in which n again refers to an embedded object, but v is now equal to the number of ICs on the wafer in which that object failed.

4 Commonality measures

In all cases considered in this article, the fail signatures that need to be compared are collections of pairs (n, v), in which n refers to an identifiable object, like an embedded array, a logic cell or a cell function, and v is a number. n can be an actual string, but it can also be a number, like a net index. v is always a number, but it can have arbitrary (non-negative) values, or it can be restricted to 0 and 1. Obviously, each signature has at most one (n, v) pair for any given n. By convention, if a given signature does not have a (n, v) pair for some n, it implicitly contains the pair (n, 0).

4.1 Pairwise commonality

There are several ways to compare two signatures. The general approach is to compare the (n, v) pairs in the two signatures one by one by comparing the v values of pairs with corresponding n values. This comparison should end with some number that, for the sake of convenience, will be between 0 and 1.

The two most relevant comparisons will be indicated by $\cos\theta_{IJ}$ and h(I,J), where I and J are two different signatures. The first one is most suitable when v can have arbitrary values. It starts from the notion of a vector $V = \{v_l\}$. The vector is that of the v values with the n values in some arbitrary but definite order that is the same for all the signatures. If n is a name, it could, for example, be their alphabetical order. v_l is then the v value of the lth (n, v) pair according to that order of n.

Two signatures can now be compared by using the notion of a cross product $V^I \otimes V^J$, defined between the vectors I and J by

$$V^I \otimes V^J = \sum_i v^I{}_i v^J{}_i, \tag{1}$$

where $v^I{}_i$ is the ith element of vector V^I. This cross product is in fact the same as the dot product used in vector algebra. To get a number between 0 and 1, we can again let vector algebra guide us towards the cosine of the angle between the two vectors. A length $^I = \sqrt{V^I \otimes V^I}$ is defined for each vector, and the commonality between two signatures is defined as

$$\cos\theta_{ij} = (V^I \otimes V^J)/(L^I L^J), \tag{2}$$

in which θ_{IJ} the purely symbolic angle between the two vectors. $\cos\theta_{IJ}$ is always well defined because all the components of the vectors are non-negative, so it is never negative, and some components in fact are positive because, otherwise, there would be no fails.

When signatures I and J are strongly correlated, that is, when corresponding v values are almost equal, $\cos\theta_{IJ}$ is close to 1. If I and J are uncorrelated, $\cos\theta_{IJ}$ is expected to be small because corresponding v values will be very dissimilar, and often one or the other will be zero

When the v values are restricted to 1 and 0, this commonality measure can still be used. However, in that situation it may be more natural to use a different one, indicated by h(I,J). The latter is obtained by considering only pairs where the corresponding v values differ. In other words,

$$\begin{aligned} h(I, J) &= 1 - \frac{\sum_i v^I{}_i(1 - v^J{}_i) + v^J{}_i(1 - v^I{}_i)}{\sum_i (v^I{}_i + v^J{}_i - v^I{}_i v^J{}_i)} \\ &= \frac{\sum_i v^I{}_i v^J{}_i}{\sum_i (v^I{}_i + v^J{}_i - v^I{}_i v^J{}_i)}, \end{aligned} \tag{3}$$

in which the sums are over all the n occurring in either I or J. This expression is in fact a very natural measure of commonality, since the numerator gets contributions only when corresponding elements in the two vectors are both 1, while the denominator gets contributions when either of the two corresponding elements is 1.

h(I,J) is a number between 0 and 1. It equals 1 when corresponding v values are the same. When the signatures have little in common, it is expected to be small, because, for each n, one or the other v value is likely to be zero.

If signatures are random, their lack of commonality should be reflected in the commonality measures being small. The average values of $\cos\theta_{IJ}$ and h(I,J), however, can be calculated only in exceptional circumstances, and we will essentially trust our intuition that both of them are small if the signatures having little or nothing in common.

4.2 Examples

The unique fails signature consists of ones and zeros only, and its appropriate measure is h(I, J). On the other hand, the marginals signature consists of arbitrary value and its appropriate commonality measure is $\cos\theta_{IJ}$.

The v component of a IC level embedded object signature can be equal only to 0 or 1, and the most appropriate commonality measure is $h(I, J)$. On the other hand, at the wafer level, the v component can have any non-negative value, and the appropriate commonality measure is then $\cos\theta_{IJ}$.

As embedded objects either fail or not, the natural commonality measure is h(I,J). This measure, however, ignores the information contained in the sizes of the objects. This size dependence can be accommodated by using a weighted commonality measure

$$h'(I, J) = \frac{\sum_i w_i v^I{}_i v^J{}_i}{\sum_i w_i(v^I{}_i + v^J{}_i - v^I{}_i v^J{}_i)}, \tag{4}$$

in which the weights w_i are arbitrary. Definition (4) is such that h'(I,J) still equals 1 when the two signatures match exactly, regardless of the weights.

The weights in Equation (4) should be such that larger objects have smaller weights than smaller objects. For example, if the object is an embedded RAM, and s_i is its size, then one possible choices is $w_i = 1/s_i$.

4.3 Commonality of more than two signatures

Next, the commonality measure needs to be extended to arbitrary sets of signatures, not just pairs. This can be done in a number of ways, but in this article, we will use only one.

Given a set of signatures, each pair of them will have a commonality measure, be it $\cos\theta_{IJ}$ or h(I,J). This measure will be indicated by the generic form o(I, J). Let now {I} be a set of signatures. $\mathcal{O}(\{I\})$, the commonality of the signatures in the set {I}, is then defined as

$$O(\{I\}) = \min_{I, J \in \{I\}} o(I, J). \tag{5}$$

As a result, O({I}) will be close to 1 if the signatures in the set are all mutually highly correlated, while it will be close to zero if at least two signatures in the set have low commonality.

It is also useful to have a measure for how little two different sets, say {I} and {J}, have in common. The measure to be used here will be indicated by D({I}, {J}) and is the opposite of that in Equation (5):

$$D(\{I\}, \{J\}) = \underset{I \in I, J \in J}{\mathrm{Max}} \; o(I, J) \; . \quad (6)$$

Finally, as a warning, note that $\mathcal{O}(\{I\})$ and $D(\{I\}, \{J\})$ depend on the particular choice made for o(I, J). Most of the time the differences will be minor, but there may be cases in which different commonality measures lead to very different $\mathcal{O}(\{I\})$ or $D(\{I\}, \{J\})$.

5 Clustering

Now that means have been defined to measure commonality between the fails of different ICs, we can turn our attention to clustering together those ICs that seem to share the same failing mechanisms. Intuitively, this is easy: just put those ICs in the same cluster that have high values of the chosen commonality measure for each pair of ICs. This, however, immediately runs into problems. Consider, for example, ICs A, B and C. A and B have a high commonality value, and so do A and C. B and C, however, have an medium commonality value, and not enough to qualify them for membership in the same cluster. Should the core cluster now be A and B, or A and C ? Either choice would be arbitrary.

To handle such conundrums, an algorithm is needed that reduces the number of arbitrary decisions to a minimum. The chosen algorithm is well known in the literature as the furthest neighbor method ([10]). A simplified program that implements this algorithm in a brute force way is shown in Figure 1.

The algorithm starts with as many clusters as there are signatures. At each step of the algorithm, the number of clusters is reduced by 1 by merging two clusters. The selection of the clusters uses the cluster commonality measure $\mathcal{O}(\{I\})$, defined above in Equation (5), where {I} is the set of signatures in the cluster. The two clusters selected for merging are such that, after merging, the commonality measure of the resulting cluster is larger than that of any other pair of clusters. If there is a choice between several pairs of clusters, each pair leading to the same commonality measure of the resulting cluster, then an arbitrary choice needs to be made to pick one of the pairs. The process of merging stops if there is no pair such that the commonality measure of the result of the merger is larger than some threshold t, a number between 0 and 1.

The result of the algorithm is a set of clusters for each one of which $\mathcal{O}(\{I\})$ is larger than t. The resulting clusters can be compared to how tight they are, using $\mathcal{O}(\{I\})$ (Equation (5)), or to how different they are using D({I}, {J}) (Equation (6)).

```
// S is set of clusters
S = ();

// put all signatures in S
for each I, put I in S;

// best_pair (S) finds pair of clusters in S that
// produce the highest combined commonality
// measure among all pairs in S.
// It returns a pair of clusters, or an empty set
// if no commonality measure exceeds the
// threshold t.

best_pair (S, t) {
  best = t;
  P = ();
  for all (c1 in S) {
    for all (c2 in S and c2 ne c1) {
      if ((m = measure (merge (c1, c2))) > best) {
        best = m;
        P = (c1, c2);
      }
    }
  }
  return (P);
}

// merge (c1, c2)  returns the union of the signatures
merge (c1, c2) {
  return ( c1 ∪ c2 );
}

// Main routine
while (P = best_pair (S, t) not empty) {
  C = merge (P);
  remove (S, P);    // remove clusters in P from S
  add (S, C);       // add merged cluster to S
```

6 Examples

The first example, Figure 2, is one caused by a photo mask defect, and is commonly called a repeater. The design is a large ASIC with many embedded SRAMs. The commonality was calculated using the pass/fails of these memories (see Section 3.) The largest cluster found using the commonality matrix is shown in Figure 2. This figure is a composite over several wafers. The wafer locations of the ICs in the cluster are indicated by the shaded cells. The numbers in the cells, and the corresponding intensities, indicate the number of wafers that contribute ICs at this

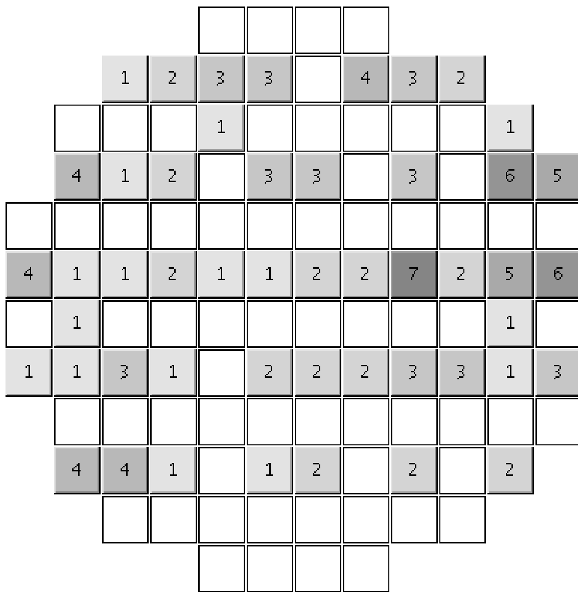

Figure 2: Repeater based cluster

location to the cluster. The empty cells show those wafer locations that have failing ICs on some wafers, none of which were in the largest cluster.

The first sign of a repeater is the clear striping in the largest cluster found using this commonality matrix. As the design is printed from a 1 by 2 reticle, a mask fail was suspected. Further analysis of the contents of this cluster revealed that one particular memory on all ICs in the cluster had failed, and that there was no other common fail among the ICs in the cluster. At the time of this writing, no failure analysis had been performed yet to confirm the diagnosis, but the signature is so strong, that no other explanation is likely.

Repeaters can be found in many ways, and using commonality analysis seems to be overkill. There are a number of reasons, however, why this example is still important. First, commonality analysis should be finding repeaters. Such fails obviously have very strong signatures, and, if commonality analysis would not have found them, its usefulness would be in question. Second, repeaters do not always stand out, for example when the yield is low and many other defects mask the otherwise obvious repeater. Commonality analysis can find those chip that suffered from the mask problem only, even when many other chips that manifest the same mask problem cannot be identified as such because of the presence of other defects.

The second example is that of a defect based cluster. The design is another large ASIC. The reticle in this case has size 2 by 2, with three A chips and one B chip (different versions of the same design.) Almost all A chips failed. Commonality analysis using unique fails (Table 2) was used to cluster the failing ICs, and three medium sized clusters were found, one of which is shown in Figure 3. Further

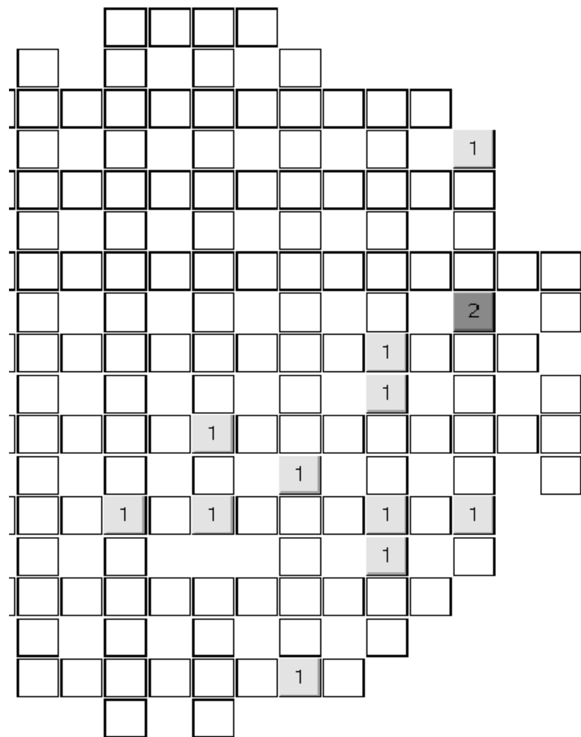

Figure 3: defect based cluster

diagnosis indicated a unique failing location in the design, which then, after failure analysis, was traced back to a physical design problem (misaligned via.)

The final example used cell based commonality to find the defect.The design is a medium sized microprocessor. It repetitively uses a group of logic gates, implemented with cells one of which was prone to fail. The fail signature was not very informative, because different instantiations of the same group of logic gates would fail on different ICs. Cell based commonality analysis, however, identified a single cluster of about ten ICs. Once this cluster was identified, it was straightforward to diagnose only the ICs in the cluster. Logic diagnosis then showed that in all cases different but similar groups of logic were involved in the fail.

7 Conclusion

We have shown several techniques for finding ICs that seem to have failed because of the same or similar defects. These techniques can be used to target different types of defect similarities. We have applied our techniques on various real life examples, and have identified in each one of them significant systematic problems.

Commonality analysis can be applied to any set of fail data, but its success depends on the choice of the commonality measure. In practice, some experimentation is

required to find a measure that brings out the large cluster that is, presumably, based on some systematic defect. Likewise, finding the right clustering threshold may require a number of trials.

Since commonality analysis compares every chip to every other chip, its run time is inherently quadratic in the number of chips. This has not been a problem yet, but it may well become one once we attempt to analyze, say, a hundred thousand or more chips.

We have also briefly alluded to extensions of commonality analysis to other than failing ICs. For example, we showed how the same techniques can be used to identify similarities in wafer level patterns of fails. Another possible extension is commonality analysis on the same IP core in different products.

8 References

[1] M. Abramovici, M. A. Breuer and A. D. Friedman, *Digital Systems Testing and Testable Design*, Computer Science Press, 1990.

[2] Greg Bazan, et al., “Using Embedded Objects for Yield Monitoring”, *Proceedings of the IEEE/SEMI Advanced Semiconductor Manufacturing Conference and Workshop*, pp124-128, 2004.

[3] Camelia Hora, Rene Segers, Stefan Eichenberger and Maurice Lousberg, “An Effective Diagnosis Method to Support Yield Improvement”, *International Test Conference*, pp. 260-269, 2002.

[4] Hari Balachandran, et al., Correlation of Logical Failures to a Suspect Process Step”, *International Test Conference*, pp. 458-466, 1999.

[5] Gary Maier, “Electronic Test Process Limite Yield”, *IBM Microelectronics,* pp. 17-22, First Quarter 2000.

[6] Sri Jandhyala, Hari Balachandran, Manidip Sengupta and Anura P. Jayasumana, “Clustering Based Evaluation of IDDQ Measurements: Applications in Testing and Classification of ICs”, *VLSI Test Symposium, pp. 444-449*, 2000.

[7] David B. Lavo, Brian Chess, Tracy Larrabee and F. Joel Ferguson, “Diagnosing Realistic Bridging Faults with Single Stuck-At Information,” in *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 17, no. 3, pp. 255-267, March 1998.

[8] Edward Eichelberger, Eric Lindbloom, John A. Waicukauski and Thomas W. Williams, “*Structured Logic Testing*,” Prentice Hall, New Jersey, 1991.

[9] Leendert M. Huisman, “Diagnosing Arbitrary Defects in Logic Designs Using Single Location at a Time (SLAT)”, *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, pp. 91-101, 2004.

[10] William R. Dillon and Matthew Goldstein, “*Multivariate Analysis Methods and Applications*”, John Wiley & Sons, 1984.