A Geometry Search Engine for the Analysis of Systematic Defects in VLSI

David L. DeMaris
IBM EDA
11400 Burnet Rd, CPYJ
Austin TX, 78758
01-512-838-6186
demaris@us.ibm.com

Dan Maynard
Bette Bergman Reuter
IBM Microelectronics
1000 River St. Essex Jct. VT, 05452
01-802-288-2042
danielm@us.ibm.com
breuter@us.ibm.com

Shi Zhong
Dept. of CSE
Florida Atlantic University
777 Glades Rd, S&E 366
Boca Raton, FL 33431
01-561-297-3168
zhong@ece.fau.edu

ABSTRACT
We describe a method for identifying layout regions in one or more designs ranked by similarity to layout implicated in systematic defects. Defect engineers examine these regions in hardware for presence of defects to analyze the cause of the failure. Feature extraction based on intersection filters with orthonormal basis patterns is performed offline at many possible defect scales.

Categories and Subject Descriptors
T5.5 [Design for yield robustness, design-to-manufacturing interface]: .

General Terms
Algorithms, Design

Keywords
Defect analysis, systematic defects, design rule checking, pattern recognition, pattern spaces

1. INTRODUCTION
Systematic defects in a semiconductor process are those defects which are not a result of particles interfering with the exposure or other process steps during manufacturing; instead, they are associated with optical, chemical, or mechanical process sensitivity to *specific shape configurations*. Yield engineers face a difficult task in identifying the cause of such defects, which exhibit correlations between design shape configurations and reduced process tolerance (i.e. the acceptable range of variation in process parameters). Contemporary semiconductor processes employ resolution enhancement techniques, like optical proximity correction (OPC), to compensate for shape deforming influences of neighboring shape features. This and other rocessing methods such as chemical-mechanical polishing (CMP) exhibit defects for which the root cause involves the interaction of a particular local pattern with the shapes in its context. The same pattern (layout shape) which results in a deterministic fail (or is not robust to process variations) in a particular part may not fail in other locations on the same design, or on a different design due to differences in context.

The problem faced by the engineers encountering such a failure can be formulated, in part, as a search task. The engineer first needs to find regions of interest with a pattern that is similar to some (possibly unknown) aspect of the pattern causing the defect. Ultimately, the coordinates of a set of such similar patterns is used to monitor the same part or other parts in manufacturing to see whether systematic failures, or distortion of the shapes which might be defects with process variations, occur in those patterns as well.

The second part of the engineer's task is to ascertain what is *different* about the similar but non-failing patterns, or what is in common among the failing patterns. Corrective action can then be taken, either in the form of process adjustments or updated design rules.

This paper describes the defect analysis problem and reports results of an approach to providing automated pattern recognition capability leveraging existing design rule checking tools. Detailed introductions to pattern recognition and data mining techniques are precluded by length constraints, but referenced throughout the text.

2. BACKGROUND AND METHODS
In this section we describe the characteristics and constraints of the problem as described by manufacturing engineers, and introduce terminology. We then outline the basic methods developed to date, problems with the previous approach, and the basic methods developed to apply pattern recognition techniques built on the existing tool infrastructure.

2.1 Problem Characteristics
The systematic defects of concern usually result in shorts, opens, or high resistance line thinning on single layers, but their root cause may involve shapes in a larger context. The exact scale of the defect shapes themselves may vary, as well as the scale of the relevant context which affects the shape. While the most common cases involves interactions between shapes on one or two layers,

Conference 04 Month 1-2, 2000, City, State.

defects involving specific patterns on up to four layers have been observed.

While the most common scenario is to search for similar patterns in the same design, patterns similar to the defect may also exist in other technology nodes. Thus, search techniques should be independent of the detailed dimensions of the target (failing) pattern.

This combination of multi-scale shapes on several layers has been referred to as *swamps*. To support process engineers analyzing such swamps, a system known as Swampfinder [1,2] has been deployed. That system requires that a unique design rule checking (DRC) *screen* is written to encode a working hypotheses of the defect configuration. By screen, we mean that the result of the computation has a binary pass/fail value. The returned shape configurations which pass the screen are turned into bitmaps and formatted as a web page for engineers to examine. There are several problems with this method:

1. It is labor intensive, requiring scarce design rule checking DRC coding resources devoted to every defect under analysis.
2. The checks resulting in "target matches" (similar patterns) must be precisely specified, while the exact relevant ranges of spacing or width values and especially contextual information are often not clearly known.
3. The check may return too many target matches. To address this, random selection among matches can be made. However, the random selections may not be representative, and may not effectively sample the space of patterns. Process engineers need to see the worst case examples to learn the process window tolerance.
4. It is iterative but not incremental. When the process engineer receives the results of this query, it may be obvious that a slightly refined search was necessary. However, the loading of the data into efficient search structures is generally lost and a new batch search must be executed, resulting in a long delay.
5. There is no measure of distance from match to defect pattern. Process engineers would like to examine a range of increasingly dissimilar patterns, if possible, to ascertain what aspects of the pattern cause the fail.
6. Finally, the DRC screen must often run on full chip data, entailing long runtimes (possibly multiple days even on multiprocessing servers). Defect patterns often occur in wiring layers or at the interface of wiring and device layers, making exploitation of hierarchy difficult.

Work commenced on a replacement system known as *Fuzzy Swampfinder* in 2002, with two generations of prototypes created. The system overcomes many of the problems described above by adapting techniques common in other domains such as multimedia search. Rather than writing specific screens for each defect, a generic feature extraction can be performed on the design *offlin*e, prior to defect discovery. The resulting databases for the smallest defects (i.e. single device shapes) may be quite large (many gigabytes). In the second generation prototype, we have introduced region sampling techniques to allow control over the amount of data collected.

To accelerate the search process for the largest offline generated feature databases, the system design and prototypes have used clustering and index methods.

A schematic of the staged system design is shown in Figure 1. Prototype work to date has addressed the components of the first stage only. Given a successful design, more direct interfaces to manufacturing imaging and analysis tools may be introduced to facilitate the workflow of process engineers.

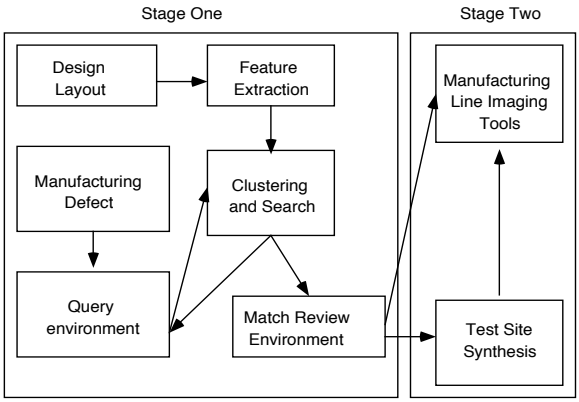

Figure 1. Schematic of Pattern Recognition Based Defect Analysis Flow

The remainder of section 2 describes the stage one components - feature extraction , clustering, and search methods - in more detail.

2.2 Feature Extraction

Many pattern recognition methods and descriptors have been developed for various classes of data including shapes and images. Two major families are structural and statistical pattern recognition. Both generally rely on decomposition of the input data into features. For the requirement of ordering matches by distance from the target, the statistical approach was chosen as more straighforward. Each configuration of shapes can be considered as a point in a pattern space, with distances in the space computed by standard methods.

VLSI layout on a single layer can be considered as binary image. However, the data is universally stored in a geometric vector description, with rasterization performed by layout editors and at final mask generation. While many feature extraction methods on images are known, it was desired to avoid the cost of rasterization; also the processing demands of most image based methods on fine-grained images of layout would be prohibitive and possibly unnecessary. Each pixel would have to be on the order of minimum design steps, since designers have flexibility to create shape borders with fine steps as long as minimum shape widths are exceeded, and specific shapes may be a source of lithography related defects. Instead, we have adapted the ideas of transform methods on images to work in the geometric data.

The Walsh Transform [3,4] is a well known method for decomposing signals or images into coefficients of an orthonormal basis, by measuring the energy of each component in an image; this can be considered a "square wave basis " version of the Fourier transform. Our method is to create two dimensional

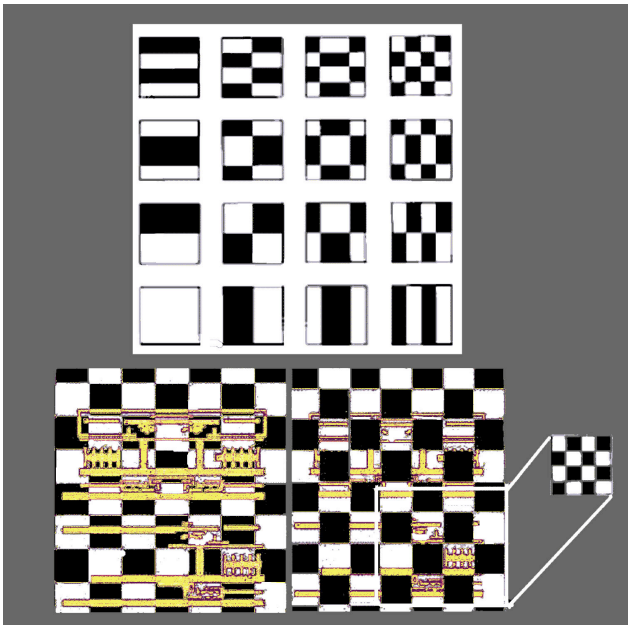

Figure 2. Walsh Patterns and Intersection.

See text for explanation.

Walsh patterns [5] and to *perform intersections of target and search layout regions* with a full orthogonal set of such patterns, scaled to match a standard set of target region sizes. The set of N 2-dimensional patterns are formed by creating matrix H, a Hadamard matrix [6] of dimension $\sqrt{N} \times \sqrt{N}$ then to apply the following procedure:

Rotate H by –90 to produce matrix had_h
Rotate H by 180 to produce matrix had_v
To produce i-th pattern W_i *{*
h'=Replicate each row of had_h to dimension of H
v'= Replicate each column of had_v to dimension of H
W_i *= Multiply the h' and v' matrices elementwise*
}

The matrix *Wi* now consists of values 1, -1, with +1 corresponding to the on (shape) regions and –1 as off (no shape). The "tiles"created from contiguous blocks in the pattern (see Fig. 2, upper region) are scaled to fit the desired defect region of interest height and width. These patterns are intersected with shapes in regions on each layer; the regions may completely cover the layout in a regular tiling, or may consist of sample regions randomly distributed over the layout. In Fig. 2, the layout shapes are intersected with the white regions of the shown tiled Walsh pattern; the remaining shapes after intersection with on tiles are shown on the lower right. The area of the intersected shapes over the entire region is the basic quantity constituting each field in a feature vector. Several functions the intersection area and other derived area have been tested as described in section 3.1

Some simple function of the intersection area is then stored as a numerical value; (functions are described in section 3.1). Each design layer of interest is stored as subset of the vector. A vector has dimension $d = l * w$, where l is the number of layers and w the number of Walsh patterns used. A feature database consists of N vectors, one for each region.

Since the spatial resolution of individual tiles in the patterns is greater than the minimum line position, the resulting vector is lossy and cannot recover the original shapes. By increasing the order of Walsh functions or replicating and scaling, we can trade off processing time and storage against feature extraction quality.

In the prototype we have used 16 Walsh patterns shown in Figure 2. These were initially formed as 4x4 tiles. Later, to increase the spatial resolution the underlying matrix was replicated and halved in spatial scale resulting in 8x8 tiles. This increases the spatial frequency, while reducing positional information. For arbitrary spatial configurations this would be of no benefit, but layout configurations are restricted by minimum width and spacing constraints and the imperative for design density. Both the subjective match quality and quantitative measures of clustering error (i.e. the measure in section 3.2) indicate that the high frequency 8x8 patterns produce better results for target shape configurations tested to date.

Walsh patterns are generated at several scales, and each scale is run on each layer of interest in the offline processing step. The scales are defined in a technology independent fashion, in terms of minimum wire pitch on the finest pitch layer for the technology. This supports a requirement to be able to search for defects in different technologies. DRC code and the corresponding Walsh patterns are synthesized for a particular technology from parameterized scripts.

In principle the regions need not be square, though in the prototype this has been the case. Walsh patterns may be synthesized at the time of target formation to match any aspect ratio, but require another batch feature extraction for non-standard aspect ratios.

In the second prototype stage, sampling of regions was introduced to reduce the data volume. Regions of the desired scale are generated at random locations; cells containing Walsh patterns are merged to those locations and intersections performed. The [institution] DRC system Niagara includes a "throw defect" feature originally developed for critical area analysis that was useful in this computation. Another feature known as *chaining* is critical for the correct operation of the sampling subsystem. Chaining modifies the default iteration character of the DRC engine to apply a sequence of operations on a shape by shape basis and accumulate the results for each shape, rather than the default behavior of applying shape operations to all shapes on a level. By using region boundaries and chaining we are able to implement the desired intersection filters, and to avoid storage costs of intermediate shape results.

2.3 Translation Invariance

The recognition of geometric transformations (translation, rotation in the plane) is always an issue in shape or image recognition. In the current system there are three aspects of the overall design which attempt to ameliorate the problem of recognizing translated or rotated variants.

1. The Walsh patterns are of relatively low spatial frequency. Shape patterns of higher spatial frequency can be translated within some of the patterns without changing the value of fields.
2. Sampling of regions for feature extraction (see section 3.3) randomizes the spatial phase of the regions with respect to underlying patterns, increasing the probability of finding matches.

2.4 Query Formation and Search

In the simplest search scenario, the user specifies a defect *target* by selecting a region on the design and the design layers of interest. Shapes are cut to the region boundary and run through the

Walsh pattern feature extraction process, resulting in a feature vector. The target vector and database of features for each <layer, Walsh pattern> tuple *i* in the d-dimensional vector are read from storage and a euclidean distance computation is performed between the database subset (i.e. feature columns) corresponding to the selected layers:

$$eucdist = \sum_{i=1}^{d} (t_i - v_i)^2$$

Other distance functions may be used, and have advantages for particular distributions, sensitivities to outliers, etc. During search, it is possible and in some cases helpful to use a weight multiplier on each term to emphasize certain aspects of the pattern. One may weight one layer more heavily, or emphasize high or low spatial frequencies preferentially, resulting in a better search for a particular target.

The results of the distance to each vector are sorted, and the user specifies a number of bins to return and the number of match results in each bin. An optional "fuzz" factor controls the number of retrieved matches. In our prototype, the center coordinates of the matching regions are returned, and the same layout editor used to form the query is used to step through the matching windows.

For some real world problems, the simple search performance has proven adequate. In a recent production example, a database of 500K feature vectors sampled from a large (18 mm x 18mm in .13 μ CMOS) ASIC was searched in under 3 minutes. The defect in this case required a relatively large search region of 9.6 sq. microns resulting in a modest database, with only 2 layers (32 features).

2.5 Clustering and clustered search

In a more complex but higher performance search scenario, one or more *clusterings* are performed offline on the feature vector database.

Clustering is a fundamental data analysis step that has been widely studied across multiple disciplines for over 40 years [7,8,9]. It has been successfully used in many exploratory pattern-analysis, grouping, decision-making, and machine-learning situations, including data mining, information retrieval, image segmentation, and pattern classification. It is sometimes referred to by alternative terminology, such as *unsupervised classification*, *unsupervised learning*, and *segmentation*, in different communities.

The general objective of a clustering problem is to discover coherent groups of similar data objects. In our application the result of each clustering is a set of representative vectors for patterns on some vertically adjacent subset of layer and an index set with the association of feature vectors to each cluster center. Normally adjacent pairs or triples of layers are clustered in a moving window fashion (i.e. M1+M2, M2+M3,M3+M4…). Several clustering methods have been examined, including k-means (km), balanced k-means (bkm), self-organizing map (som), neural gas (ng), and scalable versions of k-means and neural gas (skm, sng).

After clustering, the search process proceeds according to the following pseudocode. Again, the user specifies three search parameters: a fuzz factor for how distant the matches may be, the number of bins *i* to be examined, and the number of entries *j* to examine from each bin.

1. generate one or more target feature vectors (including translated or mirrored variants of pattern if desired).

2. for each target feature vector {
choose cluster set corresponding to selected levels
*compute n = fuzz * baselineReturnSet*
choose n nearest clusters according to distance measure
retrieve feature vectors for all n matching clusters
compute distance measure to each retrieved vector
sort return set by distance measure
divide return set into i bins
return top j entries in each bin
}

Cluster based search can be very fast for feature databases of many gigabytes, since distance comparisons are initially performed for only cluster centers, then later for the returned set; thus the entire database need not be read from disk for in-memory computations. We have implemented a scalable k-means algorithm which should be able to handle clustering of datasets limited only by mass storage capacity, though the quality or utility of searching such large databases is still to be determined.

3. RESULTS

In this section we report numerical results which were used to evaluate design choices and make course corrections during the first generation prototype.

3.1 Comparison of Feature Quality

As mentioned earlier, the raw intersection area is used in some function to create features for each level. In this section we describe some of the functions and a methodology for quantifying their effectiveness.

Let A_{tot} be the total area of shapes in a layout region of interest, A_w be the total area covered by each Walsh pattern, A_i be the area of intersection of the "on" areas of Walsh pattern i with layout, and $\overline{A}_i$ be the intersection with off areas. Intersection functions used, in order of increasing computational cost, have included:

1. A_i
2. A_i / A_{tot}
3. $(A_i - \overline{A_i}) / A_w$
4. $(A_i - \overline{A_i}) / A_{tot}$
5. $A_i (A_i - \overline{A_i}) / A_{tot}$

Due to the low spatial frequency of the patterns, each of these has some set of non-unique shape sets which will map to the same feature value for any one Walsh pattern, but not for all. Thus each is potentially viable, and the increasing computational cost must be measured. In order to assess the additional value for more complex but discriminatory functions, a study of correlations between feature dimensions was performed.

Correlation coefficients between features can be used to measure feature redundancy. Let v_i be the vector of feature dimension *i*, v_j the vector of feature dimension *j*. Both vectors are of length *N*,

which is the number of data points, and $v_i(n)$ represents the *i*-th dimension of the *n*-th data point.

Let

$$\bar{v}_i = \frac{1}{N}\sum_{n=1}^{N} v_i(n)$$

be the average value of feature $v_i(n)$.

The correlation coefficient between v_i and v_j is defined as

$$corrcoef(v_i, v_j) = \frac{\mathrm{cov}(v_i, v_j)}{std(v_i) \bullet std(v_j)}$$

where

$$\mathrm{cov}(v_i, v_j) = \frac{1}{N}\sum_{n} (v_i(n) - \bar{v}_i(n))(v_j(n) - \bar{v}_j(n))$$

is the covariance between v_i and v_j, and

$$std(v_i) = \sqrt{\mathrm{cov}(v_i, v_i)}$$

is the standard deviation of v_i. The correlation coefficient is within range [-1, 1]. A value of 1 means that v_i and v_j are 100% correlated. A value of -1 means 100% anti-correlated. A value of 0 means there is no correlation between v_i and v_j.

We use an average (absolute) correlation coefficient (ACC) to measure the average correlation between all pairs of feature dimensions.

Table 1. Cross Correlation between Feature Pairs

Reg .16	0.5883	RegHQ.16	0.0498
Reg .8	0.5608	RegHQ.8	0.0281
Reg. 4	0.5195	RegHQ.4	0.0316
Reg. 2	0.4750	RegHQ.2	0.0426
Reg.1	0.4661	RegHQ.1	0.0351

Table 1 shows cross correlation for various window sizes for two area functions corresponding to simply intersection area (1) and the higher quality difference filter (3). This study was performed on a large register file completely tiled with regions. 16 Walsh patterns were used with two layout layers, resulting in 32 feature dimensions.

It is evident that the improved Walsh features have much less correlation between the 32 feature dimensions, thus should contain more information. It is not clear at this point, however, whether or not more information leads to better clustering or search results for defective patterns.

3.2 Comparison of Clustering Methods

A study of the effectiveness of various clustering methods was performed on a large macro with diverse layout regions (floating point register file), during the first generation prototype when only small designs could be handled.

The clustering results on the register file layout data were collected for 5 different scales (16, 8, 4, 2, 1). The results for scale 2 (2 track x 2 track region) are shown in Table 2; the best algorithms, highlighted in bold are common across all scales. The number of feature vectors ≈ 560K and the number of clusters was fixed at 1000, i.e. 500 feature vectors/cluster. The objective function used is the mean-squared error

$$MSE = \frac{1}{N}\sum_{x} \|x - u_x\|^2$$

where u_x is the mean vector of the cluster to which *x* is assigned. The time recorded is purely clustering time (not including I/O time) on an IBM 397 (160 Mhz Power2) workstation.

Table 2. Clustering results for scale 2 (2 x 2 tracks)

	MSE	Time (min)
km	2.0001	110
bkm	Not avail.	Not avail.
ng	**0.403**	1579
som	1.552	59
skm	2.421	**54**
sng	6.414	1540

From the results, it is evident that the Neural-Gas algorithm always delivers the smallest MSE and the scalable k-means algorithm always runs the fastest. The blank entries for bkm (balanced k-means) are due to running out of memory on a 32 bit workstation; in general this algorithm is worst in runtime and memory, but the balanced property is desirable for search when the distribution of data is highly skewed (i.e. concentrated in subspaces). We have compiled 64 bit versions of these algorithms and plan to collect more results on larger datasets with more diverse shape patterns, but at the time of the clustering study such data was not available.

For the search application, it will always be necessary to examine many nearby clusters since the best match may not reside in the cluster closest to the target. Given this fact, it is probably not necessary to minimize the MSE measure, but to choose an algorithm balancing runtime vs. error. The SOM algorithm, which was used in our first prototype phase, seems to fare better as the size of dataset grows (e.g., it has better MSE but runs as fast as scalable k-means).

3.3 Runtime and Memory Using Sampling

Sampling was identified as a requirement early in the design stages. For technology nodes in volume production such as .13 μ, fully covering the design with 1 track wide windows on an 18 mm side die would result in over 2 x 10^9 regions; ideally one would like to use overlapping windows, quadrupling that data volume. Each window would require storage for 16 features per layer, with as many as 20 layers potentially of interest counting

vias. Even at 1 byte per field the storage requirements will become prohibitive for future technology nodes. Typical designs contain a great deal of redundancy in patterns, making sampling a reasonable strategy. Our current implementation uses sample densities ranging from 0.5% to 5% depending on the scale of regions, technology node, and die size.

Some example runtimes and feature database sizes are given in Table 3. The CPU times are for an S85 (600 MHz PowerPC) server with 100 Gbytes of real storage. Sample densities were adjusted to manage memory use with other production jobs. These runs do not reflect any use of SMP, though the underlying Niagara DRC system supports SMP processing and recent trials using SMP exhibit nearly linear speedup in number of processors with only about 50% increase in peak memory.

Table 3. Memory and Storage for 2 layer feature extraction

Design	Trks /region	Samp. Dens.	CPU Hrs/min	Peak mem.	Samples
PPC750	16	5%	1:15	400 M	27K
PPC750	4	1%	3:30	1.10G	271K.
PPC750	1	1%	12:39	8.44G	8.44M
Power5	4	0.5%	82:00	17.2G	4.37M

4. DISCUSSION

While the use of the existing DRC infrastructure has allowed rapid progress and produced a tool which shows promise for production use, the complexity of operations required is beyond the design point of existing DRC engines. Ideally, *adaptive sampling* would be performed rather than the random sampling. Samples which are already represented and do not add new information to the database would be rejected. This would require that DRC operation be interleaved with the clustering process or some similar computation comparing the accumulating database with candidate samples.

Othere applications of geometry search apart from defect characterization which preclude sampling. For example, it might be necessary to apply some manual or automated correction process to the design when process based solutions cannot be found for some defect. In this case, *all* instances of the defect or similar patterns must be found. To avoid storing features for all regions, it would again be desirable to interleave distance and clustering operations with DRC operations. Such intelligent search is beyond the historic scope of tasks envisioned for DRC engines, and requires a more flexible architecture and programming tools.

5. CONCLUSION

The key underlying concepts introduced in this paper are treating design configurations as pattern spaces, and translating regions of layout to points in such a pattern space. These concepts are anticipated to have other applications in yield enhancement and development of advanced processes. The geometric search engine for defect analysis is the first of several applications in the area of yield engineering and design for manufacturability.

We have demonstrated feasibility of a pattern recognition based approach to search and analysis of systematic defect patterns known as swamps. The system is currently being tested on real production line problems at full chip scales and evaluated for further development.

The introduction of effective pattern recognition capability into the yield engineer's toolkit will provide several benefits:

- The requirement for hand coded searches for each defect under analysis will be reduced. This should result in both a cost and turn-around time reduction.
- By avoiding the introduction of technology node specific constants in DRC code, it should be possible to search for a fail structure and context in a different technology.
- Today, there is no automated, quantitative capability to perform the comparison stage of the extended "search on similarity then compare for differences" process that the engineer engages in.

The capability to identify the coordinates of matches and quantify similarity to a failure provides the base for further integration with in-line analysis data and tools.

6. ACKNOWLEDGMENTS

William Leipold recognized the applicability of pattern recognition techniques to manufacturing and layout resolution problems. Paul Bassett put the right people in touch to initiate this project. Mark Lavin, Bob Sayah and Young Kim provided essential consultation in the use of Niagara DRC tools.

7. REFERENCES

[1] Maynard, D., Bergman Reuter, B. Rosner, R. Swampfinder, Proc. Advanced Semiconductor Manufacturing Conf. 2001, (Munich, April 2001), 151-155.

[2] Maynard, D.N., Bergman Reuter B., Patrik, J. A. . A Manufacturing Perspective of Physical Design Characterization. in Proc. Advanced Semiconductor Manufacturing Conf. 2002 (Boston, May 2002), 240-246.

[3] Walsh, J. L. A Closed Set of Normal Orthogonal Functions. Amer. J. Math. 45 , 1923, 5-24.

[4] Beauchamp, K. G. Walsh Functions and Their Applications. London: Academic Press, 1975.

[5] Optican, L. M., and Richmond, B. J. Temporal encoding of two-dimensional patterns by single units in primate inferior temporal cortex. III. Information theoretic analysis. J. Neurophysiol. 57, 1987, 162-178.

[6] Hedayat, A. and Wallis, W. D. Hadamard Matrices and Their Applications. Ann. Stat. 6, 1978, 1184-1238.

[7] Hartigan, J.A. Clustering Algorithms. Wiley, New York, 1975.

[8] Jain, A.K, and Dubes, C. Agorithms for Clustering Data .Prentice Hall, New Jersey, 1988.

[9] Zhong, S. Probabilistic Model-based Clustering of Complex Data. Dissertation. Dept. of ECE, U. Texas at Austin, 2003.

.