DEVELOPMENT OF A BRIDGE FAULT EXTRACTOR TOOL

A Thesis

by

NANDAN BHAT

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

December 2004

Major Subject: Computer Engineering

DEVELOPMENT OF A BRIDGE FAULT EXTRACTOR TOOL

A Thesis

by

NANDAN BHAT

Submitted to Texas A&M University
in partial fulfillment of the requirements
for the degree of

MASTER OF SCIENCE

Approved as to style and content by:

D. M. H. Walker (Co-Chair of Committee)	J. Hu (Co-Chair of Committee)
W. Shi (Member)	U. Cilingiroglu (Member)
C. Singh (Head of Department)	

December 2004

Major Subject: Computer Engineering

ABSTRACT

Development of a Bridge Fault Extractor Tool. (December 2004)

Nandan Bhat, B. Tech, Indian Institute of Technology, Bombay

Co-Chairs of Advisory Committee: Dr. D. M. H. Walker
Dr. Jiang Hu

Bridge fault extractors are tools that analyze chip layouts and produce a realistic list of bridging faults within that chip. FedEx, previously developed at Texas A&M University, extracts all two-node intralayer bridges of any given chip layout and optionally extracts all two-node interlayer bridges. The goal of this thesis was to further develop this tool. The primary goal was to speed it up so that it can handle large industrial designs in a reasonable amount of time. A second goal was to develop a graphical user interface (GUI) for this tool which aids in more effectively visualizing the bridge faults across the chip. The final aim of this thesis was to perform FedEx output analysis to understand the nature of the defects, such as variation of critical area (the area where the presence of a defect can cause a fault) as a function of layer as well as defect size.

To my parents

ACKNOWLEDGMENTS

I am greatly indebted to my advisor, Dr. D. M. H. Walker, for giving me an opportunity to work under him . I would like to thank him for all the invaluable guidance he provided me throughout the course of this research. I really appreciate the useful insights he gave me on the research topic and even otherwise.

I would also like to express my sincere appreciation to all my committee members for their interest in my research. Last, but not the least I would like to thank my parents for their constant support and encouragement. They have always been there for me and I shall always be grateful to them.

TABLE OF CONTENTS

LIST OF FIGURES

Page

LIST OF TABLES

Page

CHAPTER I

INTRODUCTION

In order to remain competitive, integrated circuit (IC) manufacturers must be able to handle customer demand for faster and more complex designs brought to market faster. This requires packing more and more transistors in a single chip; greater chip complexity, smaller transistor geometries and more interconnect layers [1].

Table 1 shows a comparison between past, current and future technologies [2]. Complex integrated circuit designs implemented in new technologies are more sensitive to manufacturing defects, which cause deformation to the ideal IC. Manufacturing defects are divided into two groups: *global* and *local* defects. Global defects cause global deformations such as variation in line width that results in parametric yield loss, such as inadequate speed or noise margin [3], and are also called *parametric defects*. They also affect electrical characteristics of the circuit such as bias voltages or leakage currents. On the other hand, local defects such as particles cause local deformations such as extra or missing material and affect functional yield, e.g. change circuit topology and cause the chip to fail. These are called *catastrophic* or *spot* defects.

Both, global and local defects can affect manufacturing yield, e.g. the ratio of the number of good chips per wafer and total chips on the wafer. With lower yield, the

---

The journal model is *IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems*.

manufacturing costs become higher, which affects company competitiveness in the marketplace. So it is very important for the manufacturer to get to a high yield before it starts manufacturing chips in volume, by quickly finding and removing defect sources in a new manufacturing process. Since parametric defects affect large areas of the wafer, small test structures can be used to identify many of them. For example, excessive variation in transistor effective channel length can be detected by measuring individual transistors.

Table 1. Previous, present and future semiconductor technology roadmap

Year	1997-2001	2003-2006	2009-2012
Feature size, nm	250-180	130-70	45-32
Millions of transistors per cm2	4-10	18-39	84-180
Number of wiring layers	6-7	7-9	9-10
Clock rate, MHz	200-1684	3088-5631	11511-19348
Voltage, V	1.2-2.5	0.9-1.2	0.9-1.0
Power, W	1.2-61	2.8-98	3-138
Pin cound	100-1200	500-1936	780-3616
Die size, mm2	50-385	60-520	70-750

In contrast, the source of spot defects can only be determined with certainty by analyzing defects. This is referred to as defect diagnosis. This first requires locating them within the chip. This process can be simplified through the use of test structures [2][3]. An example is static RAM whose bad bits are easily located. However the low defect densities required for competitive manufacturing mean that spot defect test structures are too large to be used during the manufacturing volume ramp. They can only be used during

process development. During production, defects must be located within the product itself. This process is referred to as *defect localization.*

Failure analysis is the process of determining the cause of detected failures, and a logical search to determine the likely source of error is called *fault diagnosis*. *Fault localization* or *fault isolation* is the process of identifying a region within an integrated circuit that contains a circuit fault, such as a short or open circuit. This region must be small enough that the defect causing the fault can be found and analyzed. This is very important for quickly debugging new products, ramping yields, identifying test and reliability problems in customer returns, and resolving quality assurance (QA) part failures. Use of advanced IC technology greatly increases the complexity of fault isolation. Often a direct view of the defect from the front or backside of the chip is not available. This makes it increasingly difficult to locate the defect using defect localization methods that detect light or heat given off by the defect. Fault isolation has become the most time-consuming part of defect diagnosis, and often a diagnosis cannot be performed since the fault cannot be localized. As a result, fault isolation is listed as a difficult challenge in the International Technology Roadmap for Semiconductors (ITRS), with its complexity projected to grow by 142 times by 2014 [2].

Industry experience has shown that a very common circuit fault is the bridging fault, which is caused by a short between two or more normally unconnected nets. Examples of some bridging fault models are shown in Figure 1. The most used bridge models are Wired-AND, when the gate pull-down network that drives one net is stronger than the gate pull-up network that drives the other net; Wired-OR, when the gate pull-up network

that drives one net is stronger than the gate pull-down network that drives the other net; and Dominant, when the gate that drives one net is stronger than the gate that drives the other net.

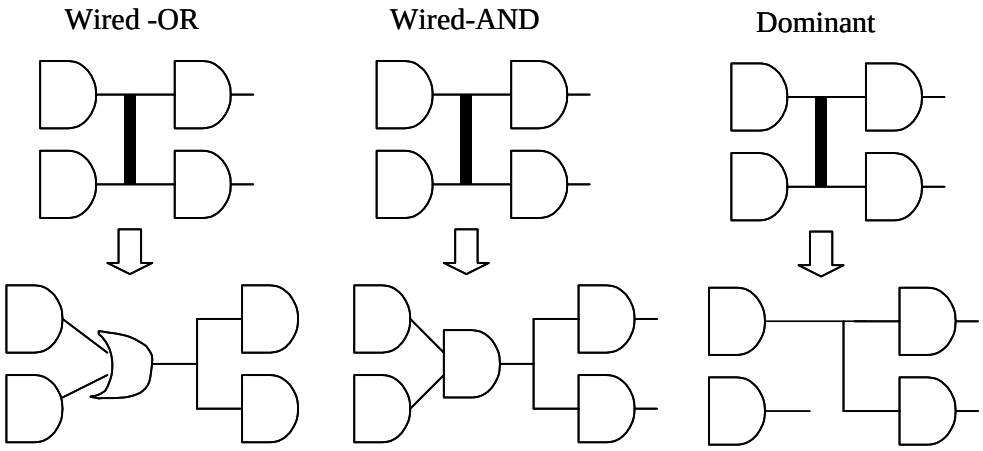


Figure 1. Different bridge models

FedEx, developed at Texas A&M University, [4] is a bridge fault extractor which extracts two-node intralayer as well as interlayer faults. An example of an intralayer fault is a bridge between two adjacent metal1 lines. An example of an interlayer fault is a short between overlapping polysilicon and metal1 lines.

The rest of this thesis is organized as follows: Chapter 2 gives an overview of previous work, including the motivation for FedEx. Chapter 3 outlines the performance enhancements to FedEx. Chapter 4 lists the applications for FedEx, including the development of a Graphical User Interface (GUI) and critical area sensitivity analysis. Chapter 5 finally concludes the report.

CHAPTER II

PREVIOUS WORK

Understanding how chips fail is the first step toward identifying and eliminating the causes of the failure. The objective is to understand failures well enough to prevent them from recurring. Diagnosing a failure means locating the fault and analyzing the defect causing it. Two types of approaches are available for fault diagnosis. The first approach is a *cause-effect analysis*, which enumerates all the possible faults existing in an applied fault model and determines, before the testing experiment, all their corresponding responses to a given applied test [5]. The second approach uses an *effect-cause analysis.* This approach processes the actual response of the chip and tries to determine exactly only the faults that could produce that response. Ideally the initial part of the analysis is done in a model-independent fashion to avoid diagnostic failure due to an inadequate fault model. We will have a brief look at previous work done using both approaches.

High real fault coverage [6] and accurate fault diagnosis [7] are most efficiently achieved when the software tools have a realistic list of the possible circuit faults. A number of software tools to identify potential realistic faults within a circuit have been developed [8][9][10][11][12][13][14]. These tools are termed *fault extractors*. They analyze the mask layout to determine what faults could realistically occur, given a description of possible manufacturing defects. Defects are assumed to occur randomly on the chip, with defects following a size distribution, and causing either *intralayer* or *interlayer* faults. An example of an intralayer fault is a bridge between two adjacent

metal1 lines. An example of an interlayer fault is a short between overlapping polysilicon and metal1 lines. A recent survey of fault extractors describes their different features [15]. Some tools such as VLASIC [16] attempt to provide an accurate circuit model for complex bridges and opens. In practice this is very expensive, and most subsequent tools cannot handle such complex models. Open circuits are often not extracted since they can be enumerated given a circuit topology, and because the stuck-at or stuck-open fault model is often a good model for their behavior.

Some fault extractors provide a list of possible faults, while others rank them based on their relative likelihood of occurrence. This is computed using the *critical area* and defect size distribution. The critical area is the area of the chip where the center of a defect must occur to cause a fault. The critical area is a function of the defect size - the larger the defect, the larger the critical area. The critical area between two adjacent wires is shown in Figure 2. The critical area is usually combined with the defect size distribution to compute the *weighted critical area* (WCA). Some tools use a surrogate for critical area, such as length of parallel wire runs that is correlated to critical area, but cheaper to compute.

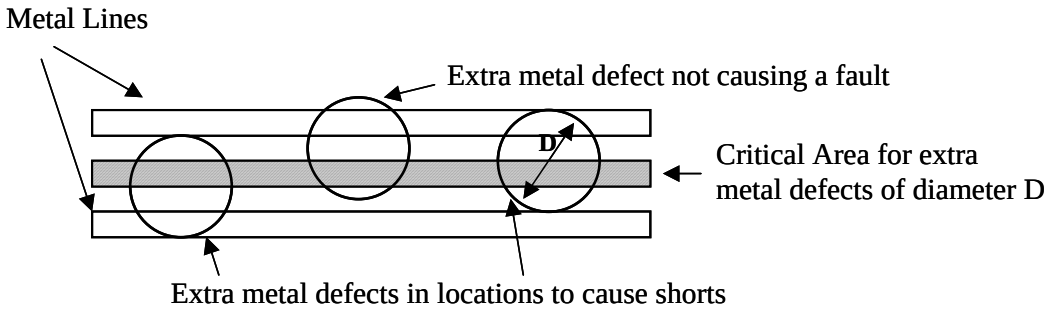


Figure 2. Critical area between two wires

For a given fault (e.g. a bridge between two nets), there could be several disjoint critical area regions at different locations and on different layers. We term these *fault sites*. It has been found that typically there are about two fault sites per intralayer bridging fault, while there is typically one site for interlayer bridges [1].

Catastrophic faults such as shorts and opens are caused primarily by spot defects, that is, regions of extra or missing material. In this work we restrict ourselves to bridging faults caused by spot defects of extra material. These are modeled as circular disks on different layers, with a diameter distribution. The process disturbance causing the defect is usually a three-dimensional particle, but modern chemical mechanical polishing limits their effect to primarily the mask layer in which they occur or neighboring layers [17], particularly in the metal layers.

The exact computation of critical areas for typical layouts is costly. This is primarily because the circular defect model implies a Euclidean polygon expansion to compute the critical area between two polygons, as shown in Figure 3. Each polygon is expanded by half the diameter in a Euclidean fashion and intersected. The intersection area is the

critical area. A common approximation is to assume a square defect and its associated orthogonal expansion, as shown in Figure 4.

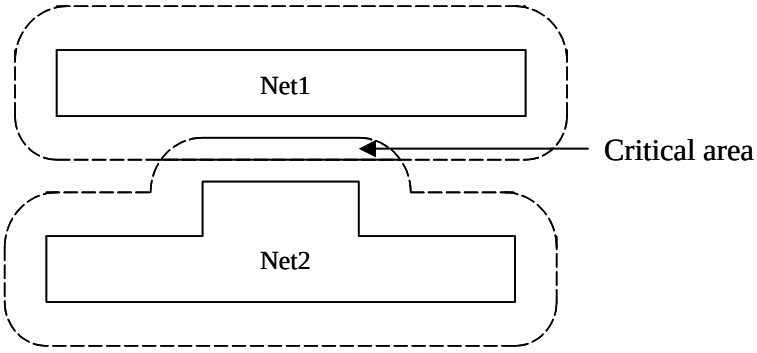


Figure 3. Euclidean polygon expansion

Orthogonal expansions are relatively inexpensive. An alternative is to use a circular defect and sample the layout with a Monte Carlo process to estimate the critical area [16]. The drawback of a Monte Carlo procedure is that large sample sizes are required to ensure that faults with small critical area are identified. The computation of weighted critical areas adds the further complication of computing the critical area as a function of defect diameter, and then convolving that with the defect diameter distribution. A Monte Carlo analysis can simplify this since the defects can be drawn from the diameter distribution.

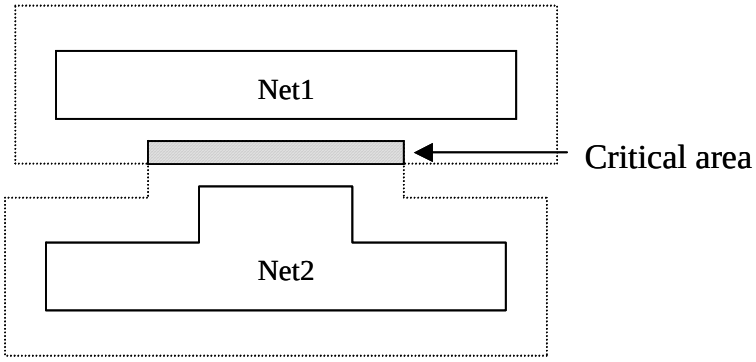


Figure 4. Orthogonal polygon expansion

Although much prior work has been expended on exact or near-exact critical area computations [18], in practice there is little need for them. The defect diameter distribution is poorly characterized, varies from factory to factory, and changes over time. The fabricated chip structures do not match the mask artwork due to process variations and the limitations of the patterning process. What is more important from a test and diagnosis viewpoint is ensuring that the more probable faults are on the fault list. ATPG will likely not target all possible realistic faults. In fact, it has been shown that when faults are sorted in descending order of critical area, only the top few ones which add up to around half the total critical area actually contribute to the faulty chip [19]. The remaining faults usually don't occur in the production. Some fault extractors use the ranking information to keep only a most likely subset of the faults [20]. However in most designs there are a very large number of similar-probability faults. The WCA uncertainty is such that faults of similar WCA can be treated as equally important, and there is no need to accurately compute which fault has slightly higher WCA. In other words, the fault list should be viewed as a ranked list of equivalence classes, with faults of similar WCA

within each class. Experimental results show that ranking information such as WCA is not useful for diagnosis [21]. What is very important for diagnosis is that all faults that could reasonably occur be on the fault list. This is essential for quality assurance failure or customer return parts, where diagnostic success must be quickly achieved on that particular part.

An alternative to a special-purpose bridging fault extractor is to use a coupling capacitance extractor. The list of coupling capacitances can be used as an unordered list of two-node bridging faults. Long, close parallel wire runs have both higher capacitance and higher critical area, so capacitance extractor rules can be used to target the most probable bridging faults [22]. If the extractor is sufficiently flexible, an approximate WCA can be computed by replacing the capacitance extraction rules with WCA extraction rules.

The advantage of using a capacitance extractor to generate a bridging fault list is that the capacitance extraction is part of the design flow, so no extra step is needed. The drawback is that capacitance extractors reduce their computational effort by making approximations, such as lumping many small capacitance values together. Hierarchical extractors may approximate the capacitance of cells when computing the capacitance of global nets. These approaches may be sufficient for obtaining a list of the most probable faults, but preclude obtaining a nearly complete list of realistic two-node bridging faults.

The DEFAM (Defect to Fault Mapper) fault extractor tool [23] was one of the first hierarchical fault extractors. The DEFAM tool consists of three main parts:

1) Hierarchy Identification

2) Hierarchical Circuit Extraction

3) Circuit Analysis

In the first step the design hierarchy is analyzed using the hierarchical chip database (HCDB) [24] in order to identify the unique parts of the design. HCDB stores the design as several levels of unique nonoverlaping *tiles*. Each tile is a corner-stitched rectangle database [25]. Since the whole design can be reconstructed by attaching together instances of these tiles, analysis of these tiles is equivalent to analysis of the whole chip. Hierarchical circuit extraction is applied on the tiles, identifying connections among tiles, and transistors and nets within each tile.

The critical area is calculated using a Monte Carlo method where defect sampling is applied on the tiles. The Monte Carlo method introduces defects uniformly on the layout using defects drawn from the size distribution. Defects of different sizes are modeled as circular regions of extra or missing material. The analysis first finds the probability that a defect of type i causes a fault of type f in tile k ($POF_{i,k,f}$) and then these results are combined to compute the same probability for the entire chip ($POF_{i,f}$).

The DEFAM system works well on very regular layout designs where the area of the tiles is small compared to the chip area. However in an ASIC design style, there is little regularity in the logic sections of a chip layout, particularly when above-the-cell routing is considered. DEFAM cannot handle this design style in reasonable memory or time, since the work of finding the regularity is not paid back by reduced analysis time. It is these irregular logic sections of the chip that were the subject of the initial research [1], since the memory arrays can be diagnosed by bitmapping [26][27].

The considerations above led to the development of a bridging fault extractor with the following characteristics:

Extract all two-node intralayer bridges.

Optionally extract two-node interlayer bridges.

Compute approximate weighted critical area for each bridge.

Compute approximate fault locations and layers.

Trade WCA accuracy for speed and memory.

Since the primary goal was to be able to handle the largest designs on a large workstation overnight, the fault extractor was named *FedEx*.

A. FedEx System

The FedEx system performs the following functions:

- *Parse mask layout.* The parser reads a hierarchical Calma GDSII Stream or Caltech Intermediate Form (CIF) [28] layout file.
- *Extract circuit topology using a technology file.* The netlist is extracted using the layout connectivity. The rectangles on a net are labeled using the text labels that intersect the rectangles on the same layer. The technology file specifies connectivity and transistor structure rules. Since we are extracting bridging faults, we do not retain transistor extraction information.

- *Identify fault sites.* All nets within the user-specified window are considered possible bridges.
- *Compute the weighted critical area of each bridge pair.* This analysis can be omitted if it is not needed for the application.
- *Write fault sites to output file.* As described below, this step must merge net numbers together, and record any equivalence information.

The information flow between these functions is shown in Figure 5.

.

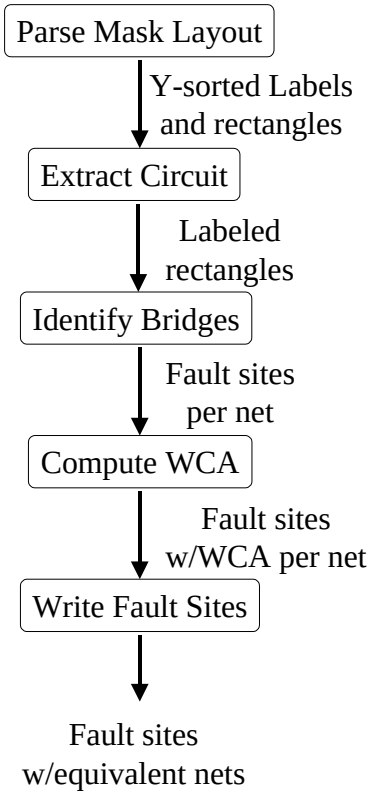


Figure 5. FedEx flow diagram

A *scanline algorithm* [29] was used for both circuit extraction and fault extraction, as shown in Figure 6. The parser reads the hierarchical layout description, converts polygons into rectangles, incrementally flattens the design, and sorts the rectangles by their top y-coordinate. These y-sorted rectangles enter the scanline where the circuit extraction is performed. The labeled rectangles then enter an array of bins where fault extraction is performed, and then rectangles exit the system. Fault sites associated with a net are written to the output file when the scanline passes the bottom of the net.

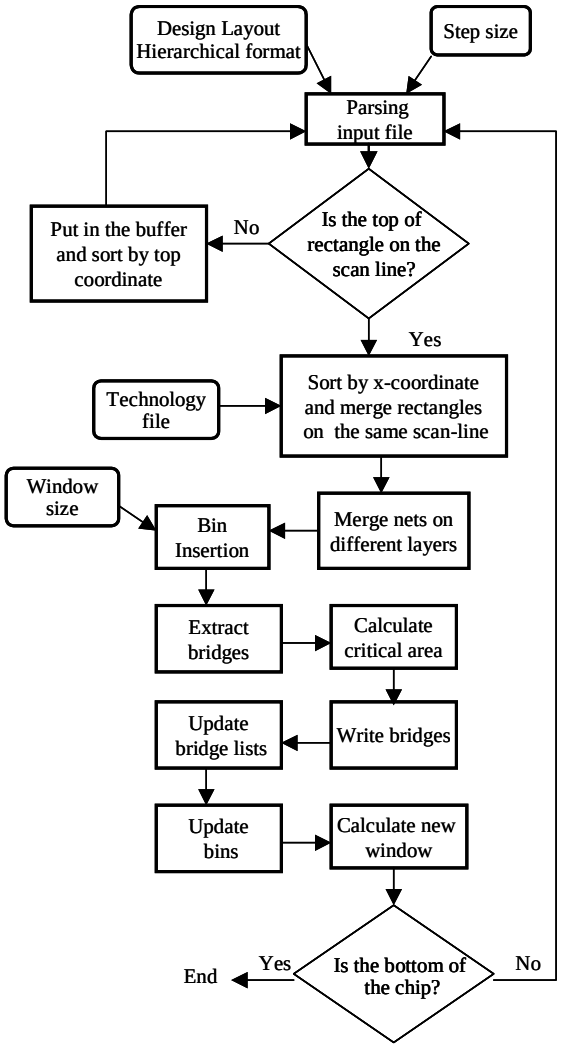


Figure 6. FedEx scanline algorithm

B. Parser

The parser reads the mask layout into memory, incrementally flattens, and sorts it by y-coordinate. A stairstep algorithm that horizontally slices any non-orthogonal section of the polygon converts polygons into rectangles. The user specifies the maximum vertical step size for this slicing process. The step must be small enough that two neighboring polygons do not touch after the conversion process, which is typically half the minimum spacing.

The parser stores cell definitions as unsorted linked lists. Once parsing is complete, the bounding boxes of all cell definitions are recursively computed by computing the bounding box of the geometry and cell instances contained within the definition. These bounding boxes are stored with each definition.

The layout is flattened and sorted by rectangle top y-coordinate as follows:

1. The top level cell instance is placed into a priority queue using the top y-coordinate. The cell definition bounding box is transformed to global chip coordinates first.
2. The rectangle, label or cell instance at the top of the queue (highest y-coordinate) is removed.
3. If it is a rectangle or label, it is given to the scanline processing code.
4. If it is a cell instance, the cell definition is opened, and its contents (rectangles, labels and cell instances) are transformed to global coordinates and inserted into the queue. Transformation of labels includes prefixing it with the global pathname.
5. Steps 2-4 are repeated until the queue is empty.

C. Circuit Extraction

Circuit extraction is performed using the scanline. The scanline is a scalar value that keeps track of the y-coordinate of rectangles currently being processed. Associated with the scanline are linked lists of rectangles, two for each mask layer - new and active. The procedure is as follows:

1. *Receive geometry.* A rectangle or label comes from the parser in y-sorted order.
2. *Insert into new list.* If the rectangle or label top y-coordinate is the same as the scanline value, it is inserted into the new list on the appropriate layer using an insert sort based on the left x-coordinate. If the top y-coordinate is below the scanline, the rectangle or label is saved in the y-sorted *buffer* list, and processing of the new list halts.
3. *The new list is merged into the active list.* The active list contains all rectangles that intersect the scanline. The idea of using a new and active list is that the superlinear cost of sorting is paid only for the smaller new list, and then the linear cost of merging is paid in the larger active list.
4. *The label and via lists are processed.* They are used to attach labels to nets and merge conducting layers together. The first label encountered on each net is recorded. Other labels could be recorded, but the assumption is that this is handled in the layout versus schematic (LVS) application that maps the bridging fault list to the netlist for use in ATPG or fault diagnosis. The circuit extraction does not make

use of the net labels, in that it does not assume that two nets with the same label are connected. Only geometry can connect two net segments together.

5. *The scanline is moved down.* It stops at the highest rectangle bottom y-coordinate, or the top of the buffer list, whichever is highest. A sorted list of rectangle bottom y-coordinates is used to quickly determine the new scanline value. Any rectangles that are now above the scanline are removed from it (and passed to the bins discussed below). All labels on the scanline are discarded. Any rectangles in the buffer list that now coincide with the scanline are moved to the new list.

The above procedure is repeated until all geometry is exhausted. If there are a finite number of scanline stops per rectangle, then the cost is linear in the number of rectangles, except for the insertion sort in step #2. In the original FedEx design, it was assumed that there are a relatively small number of rectangles in the new list, so the x-sorting cost is relatively small. Label processing in step #4 is also quadratic in time, but relatively small.

Hooks to extract directly overlapping interlayer critical area are located in extraction step #4, since the problem of identifying which nets overlap one another is essentially the same problem as determining whether a conductor overlaps a via. There is only a small additional cost to check for overlaps with adjacent conductor layers. The primary cost of interlayer bridges is in inserting them into the net data structures. The reason is that a net on one layer tends to be perpendicular to nets on adjacent layers, so the number of interlayer bridges will typically be much larger than the number of intralayer bridges. As above, a more complex data structure could be used to reduce this cost.

D. Fault Extraction

The rectangles leaving the scanline are inserted into an array of bins, shown in Figure 7. There are two rows of bins, each with height and width equal to the user-specified fault extraction window size S_{max}. This is the maximum defect size considered for bridges. The bin processing procedure is as follows:

1. As the scanline moves down, the bottom row of bins accumulates rectangles until it reaches full height (the scanline is more than S_{max} below the top row of bins).
2. Fault extraction is performed on the geometry within the bottom row of bins, including analysis of bridges to geometry in the top row of bins.
3. The geometry in the top row of bins is discarded.
4. The geometry in the bottom row of bins is moved to the top row with a pointer swing.

Each bin contains a linked list for each layer pointing to all rectangles that intersect the bin on that layer. Thus rectangles within the bin array have pointers to them from each bin they intersect. Rectangles that still intersect the scanline but protrude into the bins are pointed to as well.

Fault extraction is performed for each bin in the lower row. For each layer, each rectangle on that list is considered. It is checked against all the other rectangles on that layer within its bin, and the five neighboring bins (left, right, and three above). This means that rectangles as far as $2\sqrt{2}S_{\max}$ apart are considered. The analysis for rectangles that are

within this distance is shown in Figure 8. For each rectangle, only rectangles to the left, upper left corner, top, and upper right corner are considered for intralayer bridges. This is to avoid double-counting critical areas as the rectangles are moved from the bottom to the top row of bins.

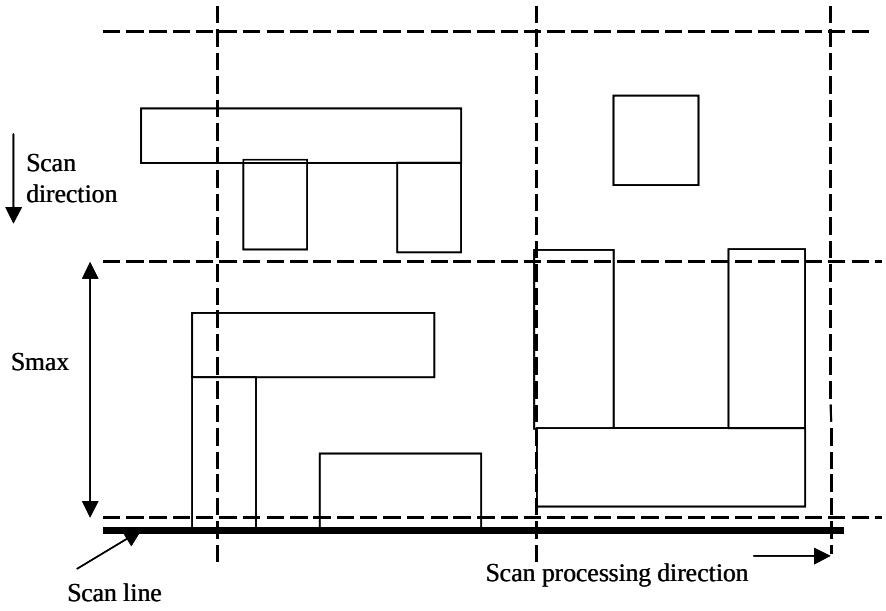


Figure 7. Fault extraction bins

In looking for other rectangles, the algorithm considers pairs, without considering intervening rectangles. These critical areas cannot occur in practice, but provide the two-node approximation for multi-way bridges. So given three adjacent parallel lines A, B, and C, bridges A-B, B-C, and A-C will be reported, even though the latter should be A-B-C.

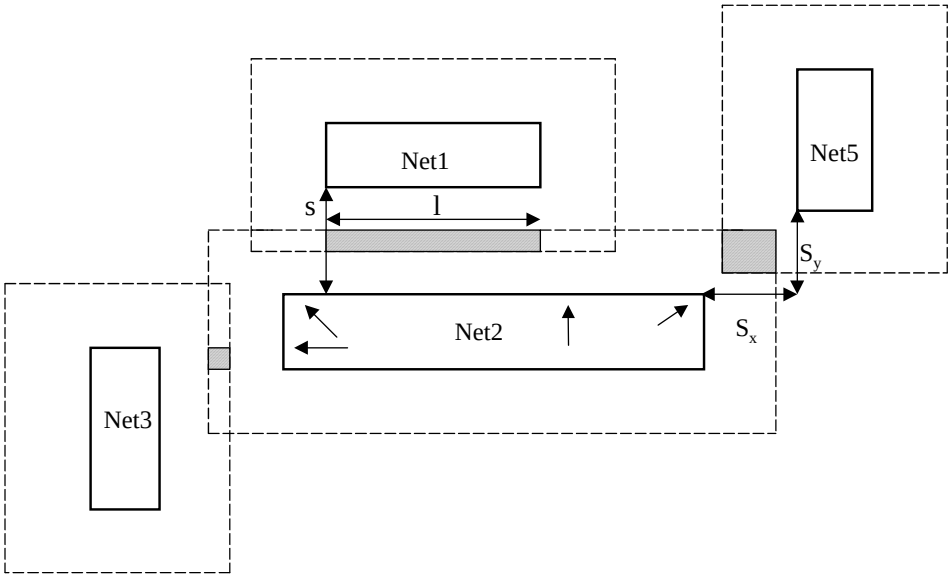


Figure 8. Critical area calculation using a geometrical method

The weighted critical area for intralayer bridges is calculated using a geometrical method. Geometry corresponding to each of the nets is inflated by $S_{max}/2$. The intersection region corresponds to the critical area, as shown in Figure 8. The spacing s and length l are used to compute the weighted critical area in Equation (1).

$$A = \int_{s}^{S\max} \chi_0^2 \frac{(x-s)}{x^3} l dx \tag{1}$$

This equation is used for the top and left critical areas. This equation assumes a $1/x^3$ defect size distribution. The x_0^2 term is the user-supplied proportionality constant for the WCA. Corner critical areas are computed with Equation (2), using the s_x and s_y values.

$$A = \int_{\max(s_x, s_y)}^{S\max} \chi_0^2 \frac{(x - s_x)(x - s_y)}{x^3} dx \qquad (2)$$

Equation (2) doesn't handle "end" effects though; it clips the critical area at the edges, so it underestimates the critical area for top side (Net1-Net2 in figure 8), but overestimates the critical area for corners (Net2-Net5 in figure 8).

E. Postprocessing

In order to minimize memory consumption, the FedEx algorithm writes out the bridges for a net as soon as the net has passed the scan line. Each entry in the bridge file contains the two bridged net numbers, critical area, bridge layer, and bridge bounding box. The label associated with the net number is written to a label file.

It can happen that two net segments start and then merge at a lower y-coordinate on the chip. Since we relabel the merged nets with the smaller of the two net numbers, as shown in Figure 9, all the bridges to the relabeled net that have already been written to the bridge file are incorrect. This is handled by recording net equivalence information with the net. When the net is completed, this equivalence information is written to an equivalence file.

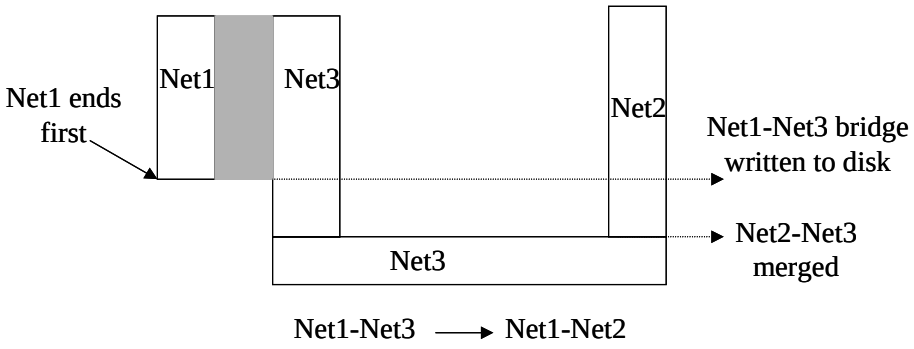


Figure 9. Nets merging

In order to generate the list of all two-node bridges in terms of unique net numbers or labels, it is necessary to first process the equivalence information. This was done with a combination of simple C programs and shell scripts. For a list of several million bridges this processing takes a few minutes.

CHAPTER III

FEDEX PERFORMANCE IMPROVEMENTS

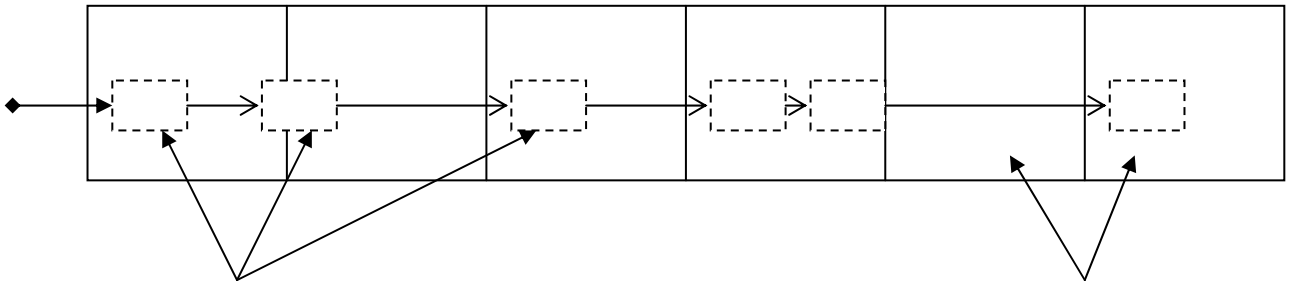
As explained in the previous chapter, the code written for FedEx is not optimal due to the use of inefficient data structures. This is not a big problem for small chips, but if large chips are to be handled in reasonable amounts of time, the existing data structures would limit the speed of the code considerably.

Code profiling showed two primary bottlenecks as far as algorithm efficiency was concerned. The first one was in the circuit extraction. After the rectangle or label comes from the parser in y-sorted order, it was inserted into the new list on the appropriate layer using an insert sort based on the left x-coordinate. This list is maintained as a doubly-linked list. This data structure is inefficient since the search time to insert a new rectangle or label is linear in the list length. The length is $O(\sqrt{N})$ for N rectangles in the chip [30]. Since insert time is $O(L)$ for list length L, then insertion time is $O(\sqrt{N})$, and total list processing time for the chip is $O(N^{3/2})$. As discussed later, for large chips, most time was spent on list insertion.

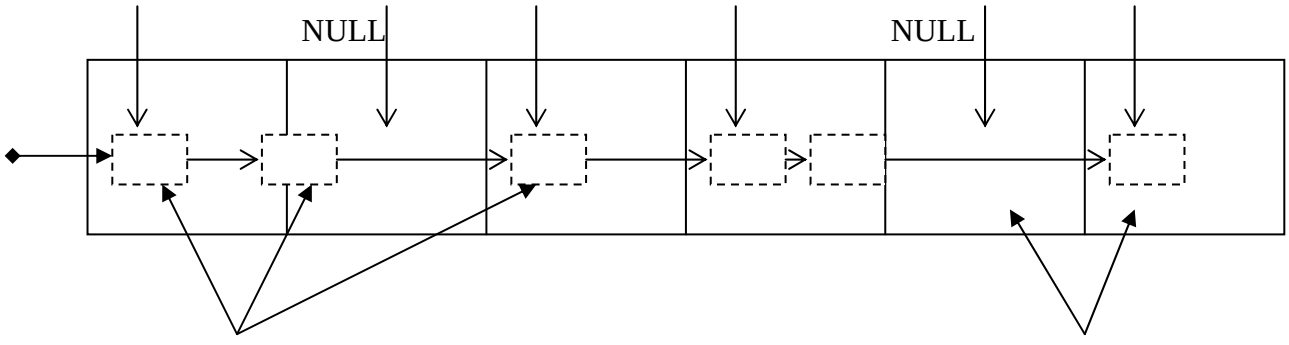
The insertion sort was replaced with a radix and then insertion sort as shown in Figure 10. For critical area analysis, the scanline is divided into an array of bins S_{max} in width. Insertion is performed by indexing to the bin that holds the left rectangle edge. Each bin has a pointer to the leftmost rectangle in the active list that intersects the bin. This is used as a starting point for insertion into the list. Since the rectangle density is relatively

constant, each bin has a roughly constant number of rectangles, so overall insertion time is constant. This reduces the total rectangle insertion time to *O(N)*.

If a bin contains no rectangles, the insertion performs a linear search of the bins to the left of the starting bin, to find the rectangle to the left of the new rectangle. Typically, there will be very few empty bins, so this bin search takes little time.



Rectangles (a) Bins



Rectangles (b) Bins

Figure 10. (a) Linked list for rectangles (original code) (b) Indexing into first rectangle for each bin (modified code)

The second and bigger bottleneck was related to the dumping of bridges in the bridge file. After a net has been processed, all the bridges to it are written out to the bridge file. These bridges to the net are maintained in a linked list in sorted order so as to help in the searching time when the same bridge is reported again. So every time a bridge is to be added, it has to be inserted into the sorted linked list. For nets with a small number of bridges (which happens in the case of small chips), this does not take much time. But when large chips such as Controller (see results table) are processed, there are many nets with a large number of bridges. Maintaining large linked lists for such nets slows down the code considerably. Code profiling also confirmed that bridge processing indeed takes up a significant percentage of the total time. The following modification was made to tackle this bottleneck.

Experimental data showed that most nets have few bridges (e.g. fewer than 10). Processing these nets takes relatively little time and the algorithm modification is not going to have an effect on these nets as far as speedup is concerned. However, there are some nets (mainly global) which have a large number (thousands) of bridges to them. These are the nets which consume a lot of time. The algorithm modification is directed primarily to take care of such nets. The algorithm uses dynamically grown hash tables since search time for hash tables is constant. The algorithm is as follows:

1. For each net, start off with a small size hash table. (Our code started off with size 1 since many nets have only 1 bridge).

2. If the number of bridges exceeds the size of the hash table, dynamically allocate a new hash table with size greater by a factor of 10. (By doing so, it is ensured that the number of bridges in a hash table is not very large and hence search time for a bridge in the hash table remains essentially constant.) Else go to step 5.

3. Insert all the bridges from the existing hash table into the newly allocated table. If more than one bridge maps to a hash table entry, maintain those bridges as an ordered linked list.

4. Free the previous hash table.

5. If no more bridge processing is to be done (this happens when the scanline has moved to the bottom of the chip), exit. Else go to step 2.

These two algorithm modifications resulted in a significant speedup of the code, by a factor of 8 in the best case. The experiments were run on a Linux machine (Pentium IV 2.26 GHz with 256 MB of memory). Tables 2, 3 and 4 show the results for several chips without extraction of the interlayer bridges for the insertion modification, bridging modification and the final (both put together) respectively. Tables 5, 6 and 7 show similar results with extraction of intra- as well as interlayer bridges.

Table 2: FedEx improvements (insertion – only intralayer bridges)

Chip	Transistor count	Modified code (insertion) (s)	Original code (s)
Serial	70k	25	26
Hopfield	22k	13	14
Frame	64k	33	33
Array	85k	45	54
Mosaic	1200k	255	273
Controller	500k	4025	4035

Table 3. FedEx improvements (bridging – only intralayer bridges)

Chip	Transistor count	Modified code (bridging) (s)	Original code (s)
Serial	70k	26	26
Hopfield	22k	13	14
Frame	64k	33	33
Array	85k	47	54
Mosaic	1200k	205	273
Controller	500k	2070	4035

Table 4. FedEx improvements (final - only intralayer bridges)

Chip	Transistor count	Final code (s)	Original code (s)
Serial	70k	25	26
Hopfield	22k	13	14
Frame	64k	33	33
Array	85k	44	54
Mosaic	1200k	193	273
Controller	500k	2055	4035

Table 5. FedEx improvements (insertion – intra- and interlayer bridges)

Chip	Transistor count	Modified code (insertion) (s)	Original code (s)
Serial	70k	37	38
Hopfield	22k	260	264
Frame	64k	104	107
Array	85k	94	99
Mosaic	1200k	765	771
Controller	500k	13753	13761

Table 6. FedEx improvements (bridging – intra- and interlayer bridges)

Chip	Transistor count	Modified code (bridging) (s)	Original code (s)
Serial	70k	34	38
Hopfield	22k	30	264
Frame	64k	90	107
Array	85k	78	99
Mosaic	1200k	502	771
Controller	500k	4441	13761

Table 7. FedEx improvements (final – intra- and interlayer bridges)

Chip	Transistor count	Final code (s)	Original code (s)
Serial	70k	34	38
Hopfield	22k	29	264
Frame	64k	90	107
Array	85k	76	99
Mosaic	1200k	499	771
Controller	500k	4439	13761

Multiple copies of the Serial chip were used to see how the time taken to process a chip varies with the size of the chip. The copies were laid out horizontally. This creates an effectively bigger scan line, so this tests the complexity of scanline processing. All other parts of the code only grow linearly in time (e.g. fault extraction, bridge processing, rectangle sorting) or are nearly constant time (parsing). These other parts of the code are already linear time on larger chips. Since the Serial chip is made up of 70k transistors, the 11-copies test has a scanline as wide as a 11*11*70k = 8.5 million transistor chip.

Table 8 shows the results for multiple copies of the Serial chip without extraction of the interlayer bridges. Figure 11 graphs the time taken versus the number of Serial chips. It can be seen that the time taken for processing is almost linear as was expected by the algorithm modifications.

Table 8. Results for serial chips (without interlayer bridges)

Copies of The serial chip	Final code (s)	Original code (s)
1	25	26
2	51	57
3	78	90
4	106	131
5	137	176
6	168	231
7	202	300
8	241	381
9	276	484
10	316	602
11	361	738

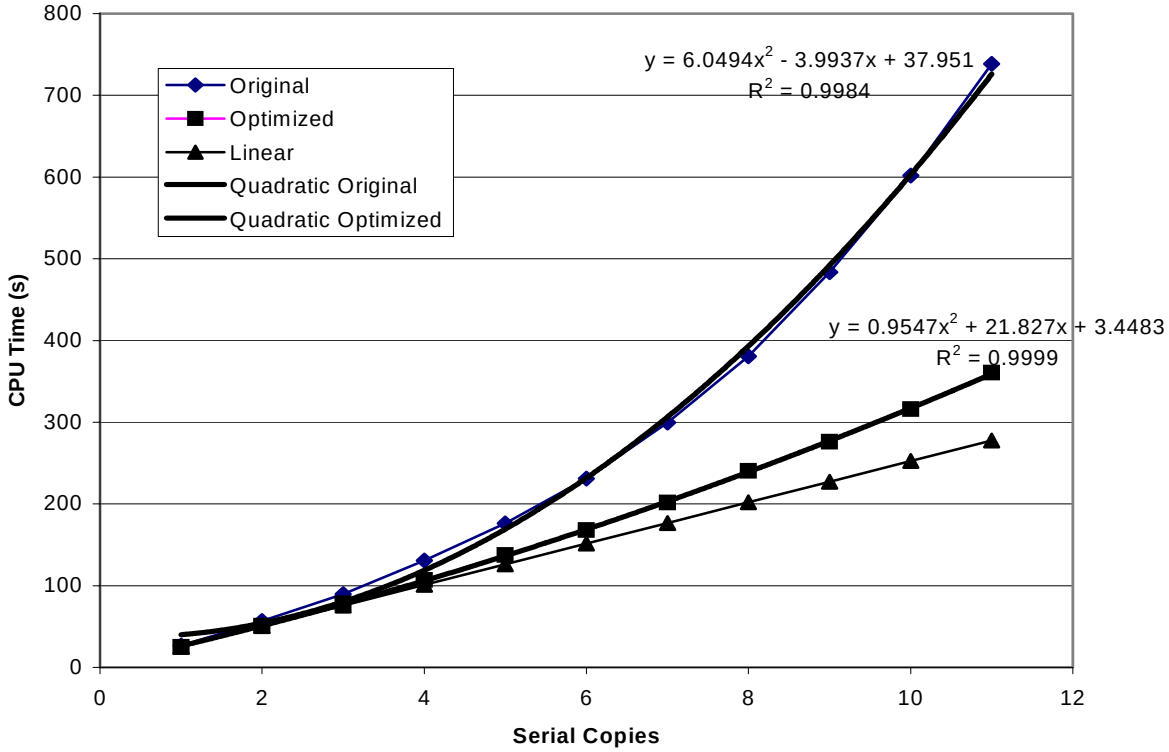


Figure 11. FedEx time for multiple copies of the Serial chip

From Figure 11, it can be seen that the algorithmic modifications have resulted in the near-elimination of the quadratic factor. The original code graph follows the equation:

$$y = 6.0494x^2 – 3.9937x + 37.951;$$

whereas the modified code graph follows the equation:

$$y = 0.9547x^2 + 21.827x + 3.4483.$$

Code profiling showed that this small quadratic factor is due to the label processing part in the code which uses an insertion sort. To address this problem, a modification

similar to the one used in the rectangle insertion part of the code can be used to eliminate the quadratic factor.

CHAPTER IV

GUI DEVELOPMENT AND SENSITIVITY ANALYSIS

A. Graphical User Interface

The second part of FedEx development was the building of a Graphical User Interface (GUI) using the Tool Command Language/Tool Kit (Tcl/Tk) [31]. This was done to aid in the more effective visualization of the FedEx output. The idea here was to provide the users with the option to view the variation of bridge fault densities across the chip. This would help in understanding which parts of the layout are more prone to defects. Once that is done, those particular parts could be targeted for defect diagnosis. This kind of information would not be obtained with just the textual output as was the case with the original version of FedEx.

The flow diagram for the tool created is shown in Figure 12. The user specifies the settings needed for FedEx through the GUI, selects the circuit and gives the command to run FedEx. This invokes FedEx, which uses the selected circuit file as input and writes the faults in the bridge fault output file. The input processor reads the generated output file of FedEx and extracts the bridge faults corresponding to the layer number specified by the user. The chip is divided into a grid and the bridge fault density is calculated for each grid cell. Colors are assigned to each grid cell depending on its bridge fault density. The grid is displayed on a canvas (which is the part of the GUI in which the chip layout is displayed). The canvas initially represents the entire chip area, and variation of color (grayscale) on the grid shows the bridge fault density.

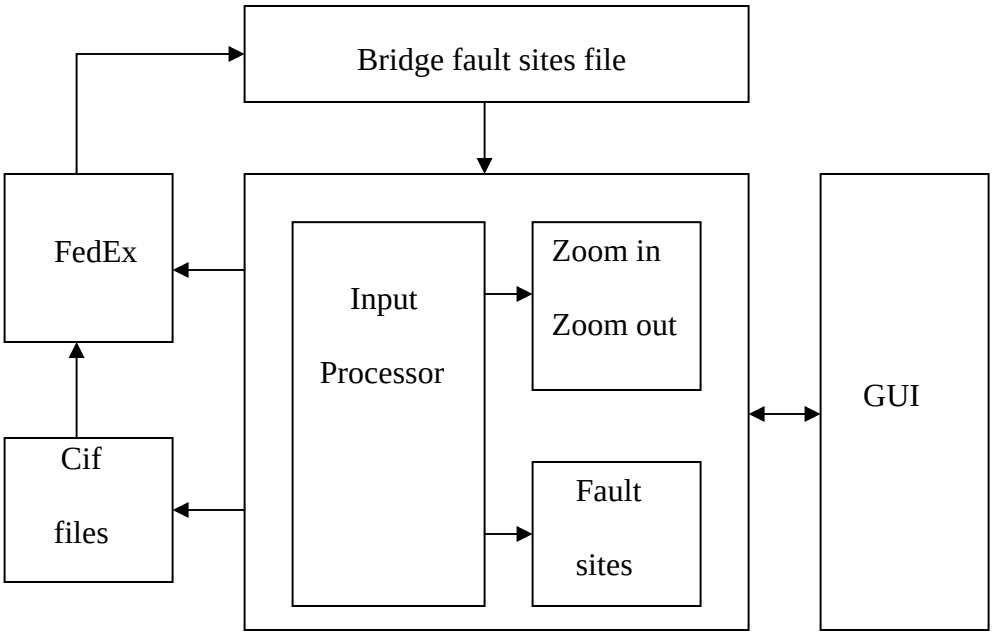


Figure 12. Flow diagram for GUI tool

Once the initial display is done, the user can zoom in and zoom out to reach a particular area of the chip and analyze the bridge fault density in that region. This zoom does not change the grid resolution but merely magnifies the grid. The bridge fault density is calculated again when a different layer number (or different chip altogether) is selected and FedEx is invoked again.

The bridge fault density is calculated in the following manner:

The chip is divided into a rectangular grid. The organization of grid cells is shown in Figure 13. The bridge fault density is calculated by finding the minimum and maximum row number and minimum and maximum column number corresponding to the bounding

box given in the FedEx output file for each bridge, and then incrementing the bridge fault density of all the grid cells that enclose the region.

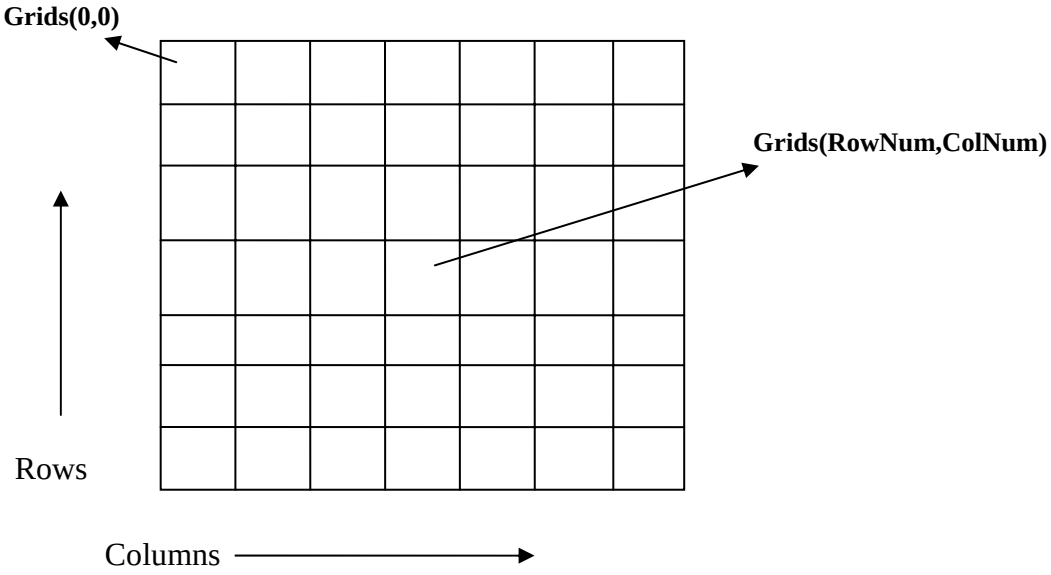


Figure 13. Grids organization on the chip

The algorithm for bridge fault density calculation is as follows:

Algorithm: *CalCriticalArea* (resolution)

1. Divide the chip into a grid according to the given resolution (with the appropriate grid cell height and width)
2. *foreach* bridge fault of the selected layer
   Find the values of RowMin, RowMax, ColMin
   and ColMax from the bounding box .
   for RowMin *to* RowMax *do*
   for ColMax *to* ColMin *do*
   increment Grid(RowNum, ColNum) by 1
3. *for* 1 *to* MaxRowOfChip *do*
   for 1 *to* MaxColOfChip *do*

Grid(RowNum, ColNum) += Grid value of neighbors
Grid(RowNum, ColNum) = Grid(RowNum, ColNum) /
(Number of neighbors + 1)

In the above algorithm, Step 2 calculates the bridge fault density and Step 3 performs the smoothening function. The smoothening ensures that the variation of bridge fault density across the grid is displayed in a continuous manner which makes the visualization of the variation easier. The organization of grid cells as rows and columns ensures that we traverse only those grid cells which enclose the bounding box of the fault.

Once the bridge fault density is calculated, a color coding scheme is needed through which density variation can be easily visualized. Initially, multiple color representation was considered, but that required a color look-up table. Instead, a gray scale color scheme was chosen, with intensity proportional to the bridge fault density. White represents the lowest fault density and black represents the highest fault density.

The algorithm for color coding first finds the maximum fault density of all the grid cells and normalizes the grayscale with respect to the maximum density. Ten equally spaced gray levels are used, covering the range 0 to 1, so as to normalize the densities. All the grid cells are traversed and each grid cell is assigned the appropriate color number depending on its fault density.

Algorithm: *ColorCoding* ( )

1. Find MaxFaultDensity, the maximum fault density of all the grids.
2. *for* 1 *to* MaxRowOfChip *do*
   for 1 *to* MaxColOfChip *do*
   Grids (RowNum, ColNum) /= MaxFaultDensity

IntervalNum = Grayscale Interval in which Grids(RowNum, ColNum) lies.

GridColor(RowNum, ColNum) = IntervalNum

The user interface with grids displayed is shown in Figure 14.

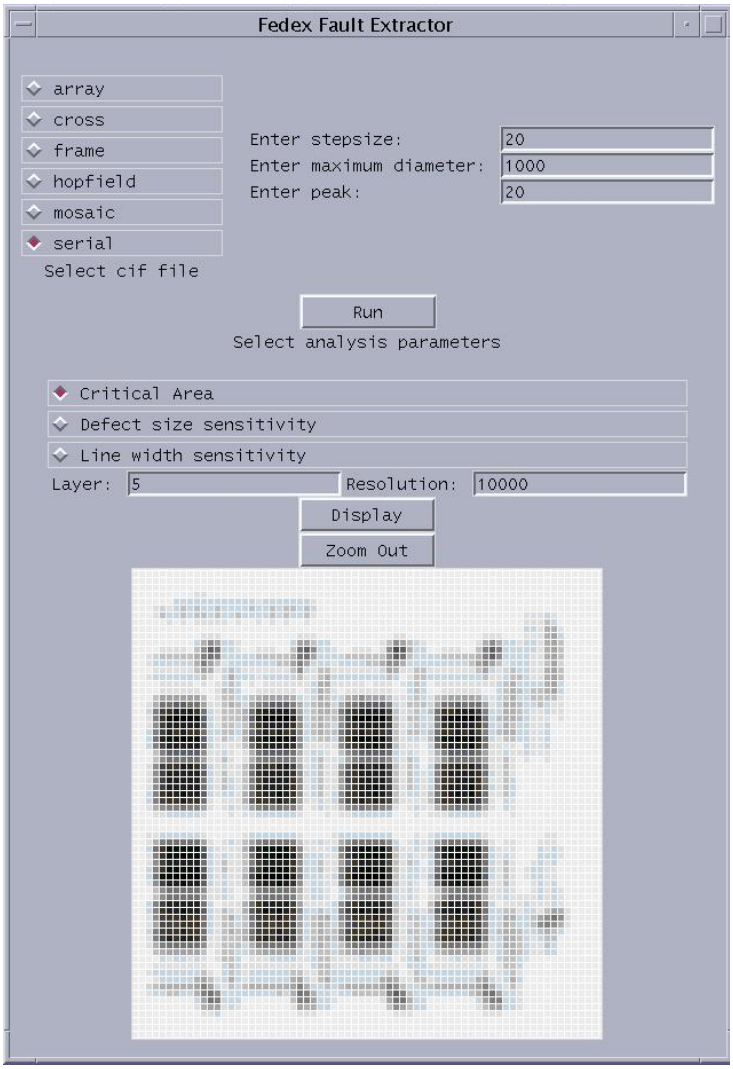


Figure 14. Initial display

The GUI provides the user with the option to select the layer number and resolution for which display is to be done. Once the display is done, zoom-in and zoom-out is considered next.

The row and column numbers corresponding to the area selected (by the mouse) is found. During this process, the following cases were considered for the user-selected area:

- Has some part outside the canvas (i.e. the area selected extends beyond the grid cells displayed on the canvas);
- The mouse selected area is from right to left;
- The mouse selected area is from left to right.

Once the starting and ending row and column numbers of the user-selected area are known, some rows or columns are added to make full use of the available canvas area. This resulted in two cases:

- **Case1**: The number of rows is greater than the number of columns in the user selected area. More columns are added on left and right of the user-selected area. During this process of addition of columns only valid column numbers are added. Update the start column and end column.
- **Case2**: The number of rows is smaller than the number of columns in the user selected area. More rows are added on top and bottom of the user-selected area. During this process of addition of rows only valid row numbers are added. . Update the start row and end row.

The updated starting row and column, and ending row and column are displayed on the canvas (Figure 15).

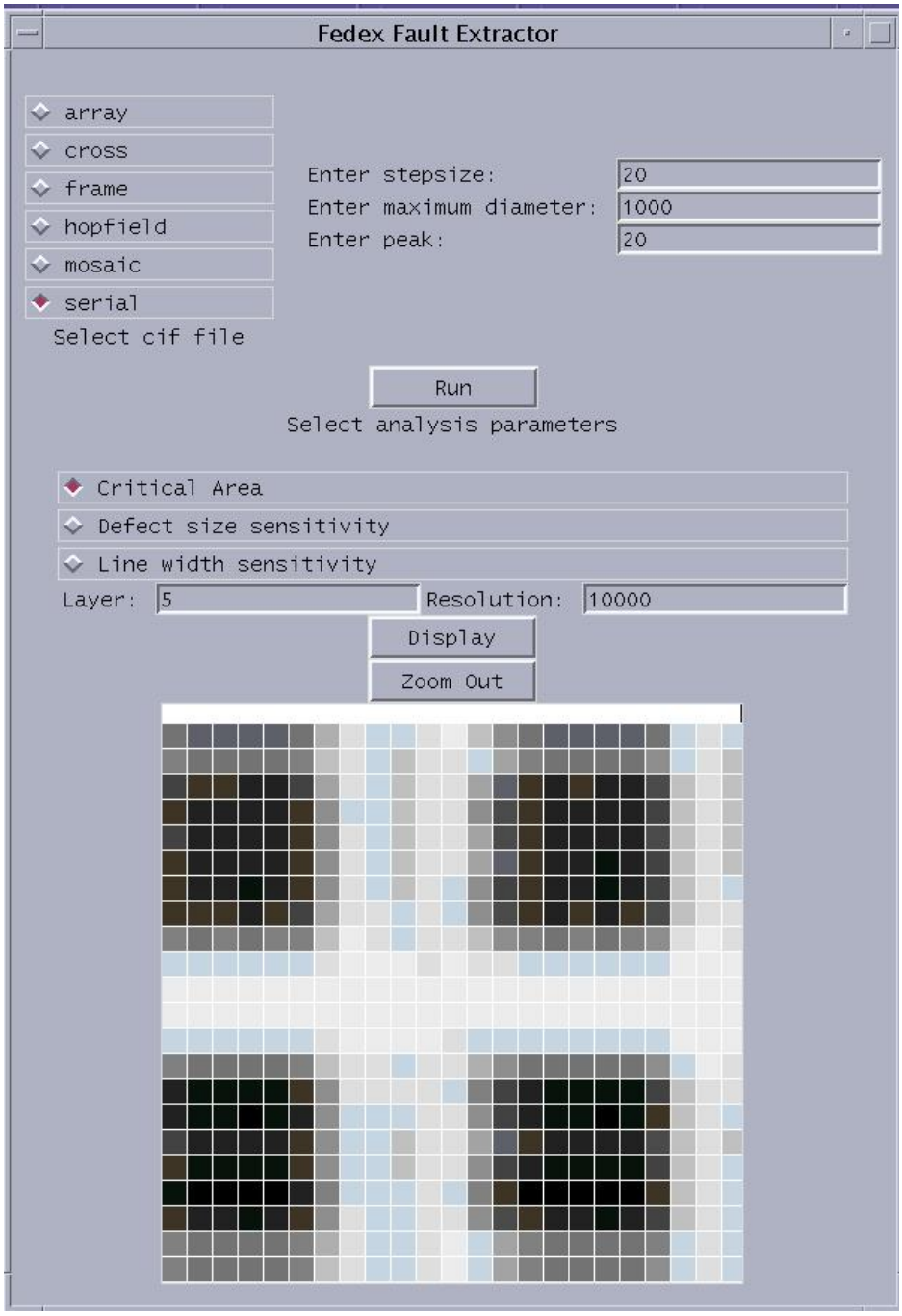


Figure 15. The display after zoom-in

Fixed zoom-out is implemented in which the zoom-out is by hundred percent of the current display (i.e. 2x zoom-out). Using the current row and column numbers displayed on the canvas, the new starting and ending row and column numbers are calculated. The row and column numbers are adjusted to make sure that the displayed row and column numbers are valid. The updated starting row and column, and ending row and column are displayed on the canvas (Figure 16).

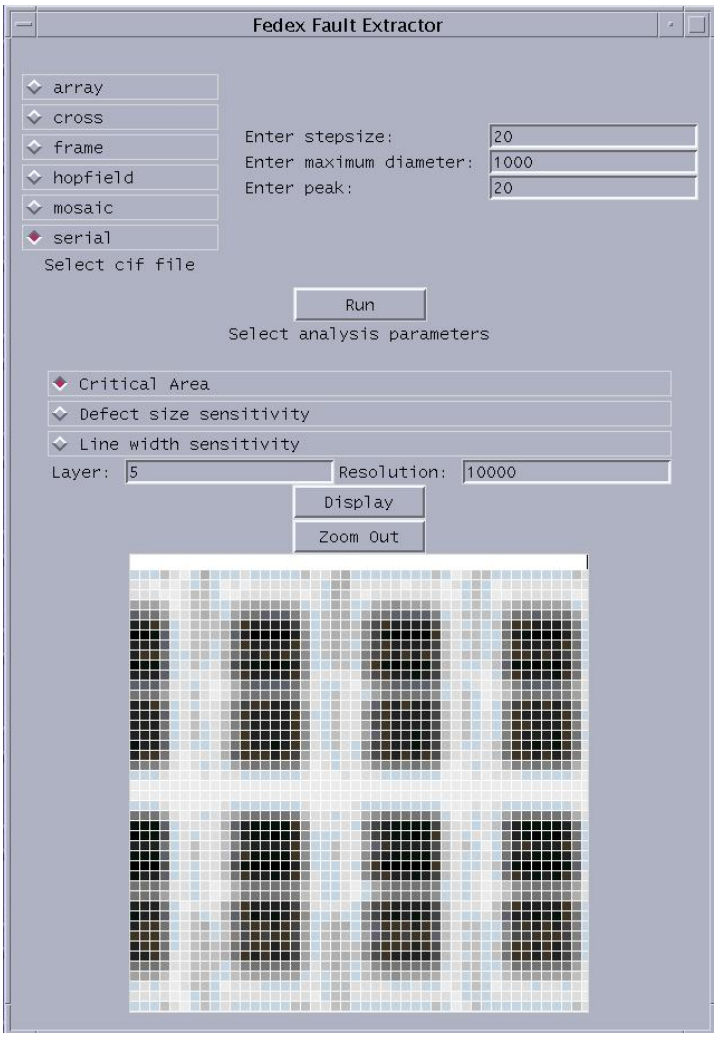


Figure 16. The display after zoom-out of Figure 15.

B. Sensitivity Analysis

FedEx, in its original from, writes out the list of bridges. However, there are certain kinds of information which can be useful to users who want to know which parts of the chip layout are more prone to defects. For example, users might want to know the variation of critical area as a function of defect size; or the variation of critical area as a function of line width. The goal of sensitivity analysis was to provide such information. Towards this end, the code was modified to incorporate variation in critical area as a function of defect size as well as line width in addition to calculating the critical area itself.

The equation for the critical area calculation used in the code is given in equation (3).

$$CA = x_0^2 * l * (S_{max} - s)^2 / (2 * S_{max}^2 * s) \qquad (3)$$

where x_0 and l are constants, S_{max} is the maximum defect size as given by the user and s is the spacing between nets. Equation (3) is the integrated version of equation (1).

To calculate the variation in critical area as a function of defect size, equation (3) is differentiated with respect to *def* to obtain equation (4).

$$d(CA)/d(S_{max}) = x_0^2 * l * (S_{max} - s)/(S_{max}^3) \qquad (4)$$

Similarly, to calculate the variation in critical area as a function of line width (which causes a variation in the spacing), equation (3) is differentiated with respect to s to obtain equation (5). Variation in line width by Δl corresponds to variation in line spacing by Δl.

$$d(CA)/d(s) = x_0^2 * l * (s^2 - S_{max}^2)/(2 * S_{max}^2 * s^2) \qquad (5)$$

The variations were computed and written out into the bridge file for each bridge. The percent variation in critical area as a function of defect size i.e. (variation in critical area due to unit change in defect size)*100/total area, and the percent variation in critical area as a function of line width i.e. (variation in critical area due to unit change in line width)*100/total area are shown in Tables 9 and 10 respectively.

Table 9. Sensitivity analysis – defect size variation

Chip	Transistor count	% change in critical area due to unit defect size variation		
		Poly	Metal 1	Metal 2
Serial	70k	0.0769	0.152	0.0554
Hopfield	22k	0.0574	1.13	0.413
Frame	64k	0.0809	0.323	0.444
Array	85k	0.0752	0.237	0.456
Mosaic	1200k	0.0448	0.0634	0.0168
Controller	500k	0.00341	0.00476	0.00220

Table 10. Sensitivity analysis – line width variation

Chip	Transistor count	% change in critical area due to unit line width variation		
		Poly	Metal 1	Metal 2
Serial	70k	0.546	0.793	0.226
Hopfield	22k	0.422	5.95	1.75
Frame	64k	0.549	1.99	1.94
Array	85k	0.539	1.23	1.83
Mosaic	1200k	0.910	0.594	0.119
Controller	500k	0.076	0.313	0.0747

The sensitivity analysis was incorporated into the GUI as well. So the user has options to not only view the critical area layerwise, but also the variations in the same with respect to defect size (Figure 17) and line width (Figure 18).

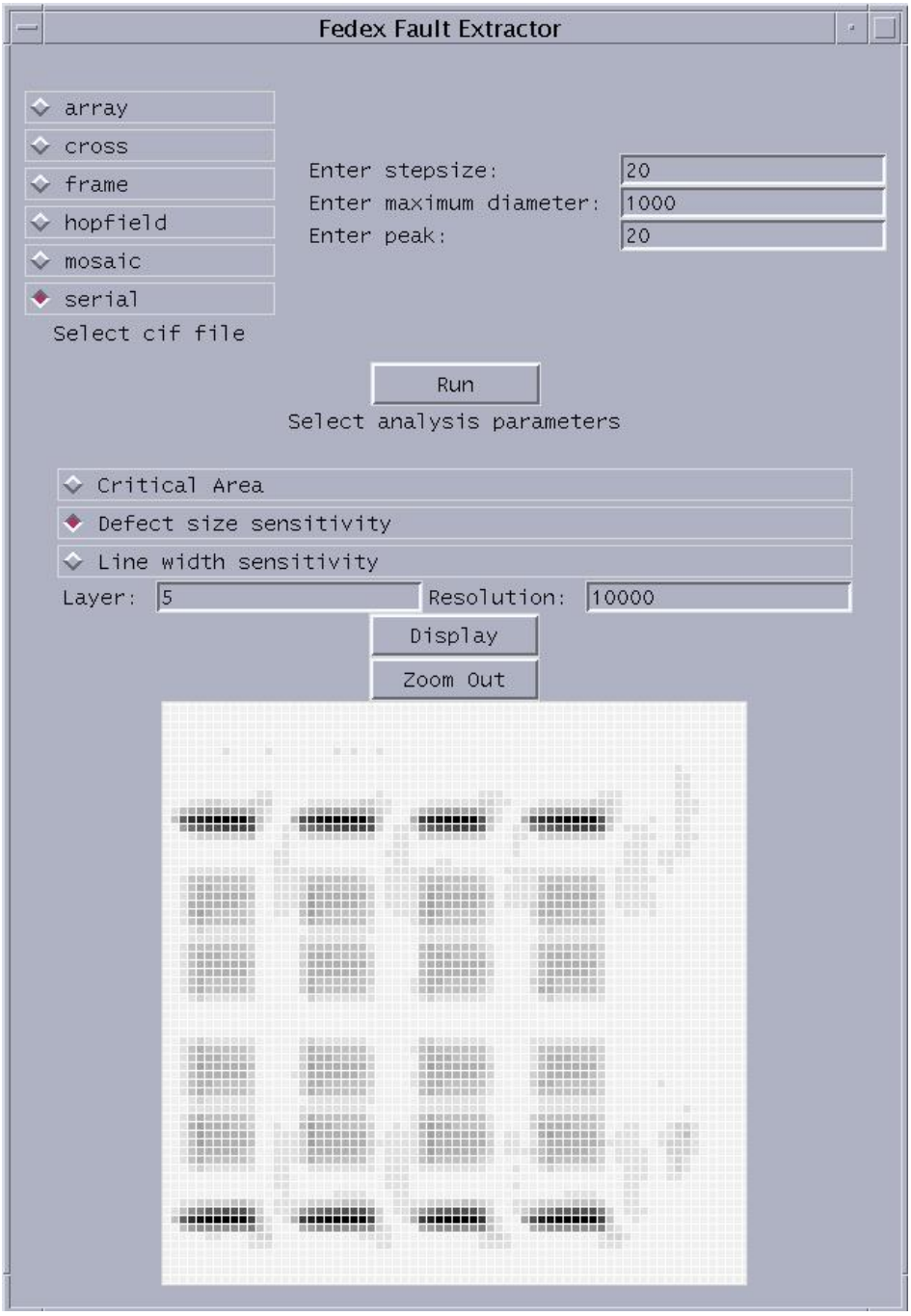


Figure 17. Variation in defect size sensitivity across the chip

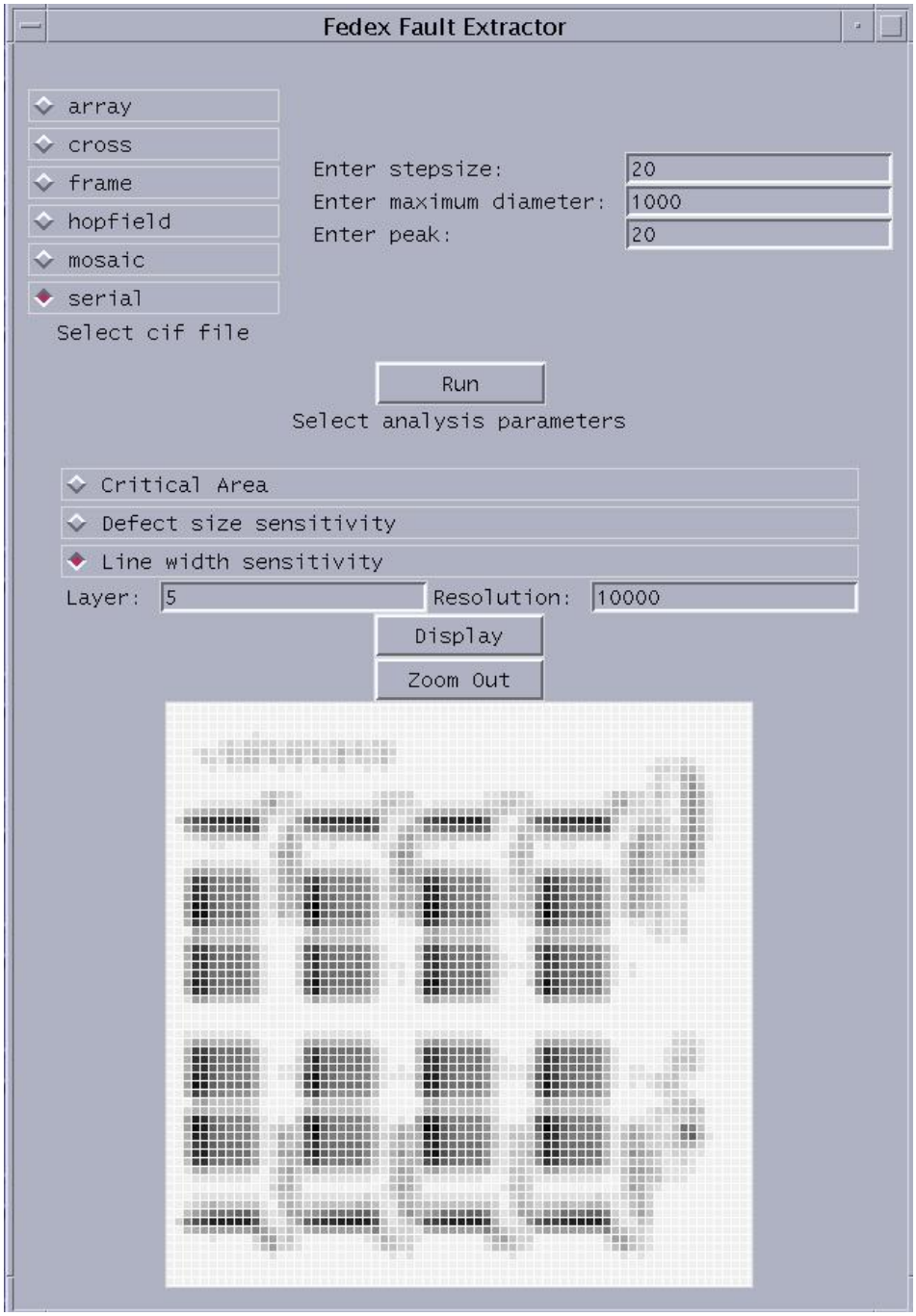


Figure 18. Variation in line width sensitivity across the chip

CHAPTER V

CONCLUSIONS AND FUTURE WORK

In this thesis, contributions were made to fault extraction by using the tool FedEx, which was previously developed here at Texas A&M University.

A. Contributions

The specific contributions of this work were as follows:

- The primary contribution was to speed up FedEx by algorithmic modifications since the original version could not handle large chip layouts in reasonable amounts of time. The performance improvement was by a factor of 8 in the best case. This improvement was obtained by modifying the parts of the FedEx code which handle the rectangle insertion and bridge processing routines. Due to the use of inefficient data structures, the code slowed down considerably while handling large chips. The modifications made to these routines removed these inefficiencies.

- A GUI was also developed which aids in effectively visualizing the bridging faults across the chip. Different portions of the chip are shown in different shades (grayscale) depending on the bridge fault density in those portions. This gives users an idea of which portions of the chip are particularly prone to bridging faults. Besides, critical area variation as a function of defect size and line width are also displayed. The GUI can be used to effectively target select portions of the chip. For example, if the memory part of a particular chip layout

is shown in darker color, it shows that there is a heavy congestion of potential bridging faults in that part and steps can be accordingly taken to target that particular part.

- Critical area analysis was performed to evaluate the sensitivity of the chip to variations in defect size and line width (this was incorporated into the GUI as well). This analysis gives the variation in critical area due to the variation in defect size or line width. This information can be used to understand, for example, the percent variation in critical area due to say, 1% variation in polysilicon line width. This can be used to estimate the accuracy of the critical area calculated.

B. Future Work

For future work, FedEx needs improvements primarily in the memory usage. As of now, FedEx stores all hierarchy in memory, so this dominates memory usage. One option would be to cache off of disk to reduce virtual memory usage. To further improve speed, one could also skip cells/nets that are not of interest. For example, many times one might not require the analysis of the entire chip but only some select portions. In such cases, provision would have to be made for the user to select specific cells of the chip and then analyze only those cells. This would definitely reduce the time taken for bridge fault extraction. Similarly, one could provide the user with the option to select specific nets.

One could then perform fast analysis of bridges to those specific nets. The idea here is to not analyze the entire chip in order to reduce the time taken.

With such improvements and the improvements to the memory usage, FedEx would be further optimized both on speed as well as on memory to effectively handle large chips in reasonable amounts of time.

REFERENCES

[1] Z. Stanojevic, Ph.D. Dissertation, “Computer-aided Fault to Defect Mapping (CAFDM) for Defect Diagnosis,” Department of Computer Engineering, Texas A&M University, College Station 2002.

[2] Semiconductor Industries Association, “International Technology Roadmap for Semiconductors,” Austin, TX, 1999.

[3] W. Maly, A. J. Strojwas, and S. W. Director, “Fabrication Based Statistical Design of Monolithic ICs,” in *Proc. IEEE Int. Symp. Circuits and Systems*, April 1981, pp. 135-138.

[4] Z. Stanojevic and D. M. H. Walker, “FedEx – A Fast Bridging Fault Extractor,” in *Proc. Intl. Test Conf.*, 2001, pp. 696-703.

[5] M. Abramoveic, M. A. Breuer, and A. D. Friedman, *Digital Systems Testing and Testable Design*, New York: IEEE Press, 1990.

[6] V. R. Sar-dessai and D. M. H. Walker, “Resistive Bridge Fault Modeling, Simulation, and Test Generation”, in *Proc. IEEE Int. Test Conf.*, Atlantic City, NJ, Sept. 1999, pp. 596-605.

[7] Z. Stanojevic, H. Balachandran, D. M. H. Walker, F. Lakhani, S. Jandhyala, K. Butler and J. Saxena, “Computer-Aided Fault to Defect Mapping (CAFDM) for Defect Diagnosis”, in *Proc. IEEE Int. Test Conf.*, Atlantic City, NJ, Oct. 2000, pp. 729-738.

[8] S. T. Zachariah and S. Chakravarty, “A Scalable and Efficient Methodology to Extract Two Node Bridges from Large Industrial Circuits”, in *Proc. IEEE Int. Test Conf.*, Atlantic City, NJ, Oct. 2000, pp. 750-759.

[9] D. M. H. Walker and S. W. Director, “VLASIC: A Catastrophic Fault Yield Simulator for Integrated Circuits,” *IEEE Trans. Computer-Aided Design*, vol. CAD-5, no. 4, pp. 541-556, Oct. 1986.

[10] D. D. Gaitonde and D. M. H. Walker, “Hierarchical Mapping of Spot Defects to Catastrophic Faults -- Design and Applications,” *IEEE Trans. Semiconductor Manufacturing*, vol. 8, no. 2, pp. 167-177, May 1995.

[11] P. K. Nag and W. Maly, “Hierarchical Extraction of Critical Area for Shorts in Very Large ICs”, in *IEEE Int. Workshop Defect and Fault Tolerance in VLSI Systems*, Lafayette, LA, Nov. 1995, pp. 19-27.

[12] A. L. Jee and F. J. Ferguson, “Carafe: An Inductive Fault Analysis Tool for VLSI Circuits”, in *Proc. IEEE VLSI Test Symp.*, Atlantic City, NJ, 1993, pp. 92-98.

[13] F. M. Goncalves, I. C. Teixeira and J. P. Teixeira, “Realistic Fault Extraction for High-Quality Design and Test of VLSI Systems”, in *IEEE Int. Symp. on Defect and Fault Tolerance in VLSI Systems*, Paris, France, 1997, pp. 29-37.

[14] S. T. Zachariah, S. Chakravarty, and C. D. Roth, “A Novel Algorithm to Extract Two-Node Bridges”, in *Proc. Design Automation Conf.*, Los Angeles, CA, June 2000, pp. 790-793.

[15] D. M. H. Walker, “Critical Area and Fault Probability Prediction”, in *Integrated Circuit Manufacturability: The Art of Process and Design Integration*, J. Pineda de Gyvez and D. K. Pradhan, eds., New York: IEEE Press, 1998, pp. 121-156.

[16] D. M. H. Walker and S. W. Director, “VLASIC: A Catastrophic Fault Yield Simulator for Integrated Circuits,” *IEEE Trans. Computer-Aided Design*, vol. CAD-5, no. 4, pp. 541-556, Oct. 1986.

[17] H. Balachandran and D. M. H. Walker, “Improvement in SRAM-Based Failure Analysis Using Calibrated IDDQ Testing,” in *Proc. IEEE VLSI Test Symp.*, Princeton, NJ, April 1996, pp. 130-136.

[18] D. M. H. Walker, “Critical Area and Fault Probability Prediction”, in *Integrated Circuit Manufacturability: The Art of Process and Design Integration*, J. Pineda de Gyvez and D. K. Pradhan, eds., New York: IEEE Press, 1998, pp. 121-156.

[19] M. Tripp, “The Trouble with Targeting Defects,” *Defect Based Testing Workshop*, Napa Valley, CA, April 2004.

[20] S. T. Zachariah and S. Chakravarty, “Extraction of Two-node Bridges from Large Industrial Circuits,” *IEEE Trans. Computer-Aided Design*, vol. 23, no. 3, March 2004, pp. 433-439.

[21] Z. Stanojevic, H. Balachandran, D. M. H. Walker, F. Lakhani, S. Jandhyala, K. Butler and J. Saxena, “Computer-Aided Fault to Defect Mapping (CAFDM) for Defect Diagnosis”, in *Proc. IEEE Int. Test Conf*, Atlantic City, NJ, Oct. 2000, pp. 729-738.

[22] C. E. Stroud, J. M. Emmert, J. R. Bailey, K. S. Chhor, and D. Nikolic, “Bridging Fault Extraction from Physical Design Data for Manufacturing Test Development”, in *Proc. IEEE Int. Test Conf*, Atlantic City, NJ, Oct. 2000, pp. 760-769.

[23] D. D. Gaitonde and D. M. H. Walker, “Hierarchical Mapping of Spot Defects to Catastrophic Faults -- Design and Applications,” *IEEE Trans. Semiconductor Manufacturing*, vol. 8, no. 2, pp. 167-177, May 1995.

[24] D. M. H. Walker, C. S. Kellen, D. M. Svoboda, and A. J. Strojwas, “The CDB/HCDB Semiconductor Wafer Representation Server,” *IEEE Trans. Computer-Aided Design*, vol. 12, pp. 286-293, Feb. 1993.

[25] J. Ousterhout, “Corner Stitching: A Data-Structuring Technique for VLSI Layout Tools,” *IEEE Trans. Computer-Aided Design*, vol. CAD-3, pp. 87-100, Feb. 1984.

[26] J. Khare and W. Maly, “Inductive Contamination Analysis (ICA) with SRAM Application,” in *Proc. IEEE Int. Test Conf*, Washington DC, 1995, pp. 552-560.

[27] S. Naik, F. Agricola, and W. Maly, “Failure Analysis of High Density CMOS SRAMs: Using Realistic Defect Modeling and IDDQ,” *IEEE Design and Test of Computers*, vol. 10, pp. 13 –23, June 1993.

[28] C. Mead and L. Conway, *Introduction to VLSI Systems*, Boston, MA: Addison-Wesley Publishing Company, 1980.

[29] A. Gupta, “ACE - A Circuit Extractor”, VLSI Document V105, Dept. of Computer Science, Carnegie Mellon University, Pittsburgh, PA, June 1982.

[30] J. L. Bentley, D. Hakenj and R. W. Hon, “Statistics on VLSI Designs”, Dept. of Computer Science, Carnegie Mellon University, Pittsburgh, PA, April 1980.

[31] J. K. Ousterhout, *Tcl and the Tk Toolkit*, Boston, MA: Addison-Wesley Publishing Company, 1994.

VITA

Nandan Bhat was born on May 10, 1980 in Mumbai (Bombay), India. He received his B.S. degree in Electrical Engineering in 2002 from the Indian Institute of Technology (IIT) Bombay. His permanent mailing address is: 14, Uma Irla, Irla Lane, Vile Parle (West). Mumbai – 400056. India.