FedEx – A Fast Bridging Fault Extractor

Zoran Stanojevic
Dept. of Electrical Engineering
Texas A&M University
College Station TX 77843-3124
Tel: (979) 862-6610
Fax: (979) 847-8578
Email: zoran@cs.tamu.edu

D. M. H. Walker
Dept. of Computer Science
Texas A&M University
College Station TX 77843-3112
Tel: (979) 862-4387
Fax: (979) 847-8578
Email: walker@cs.tamu.edu

Abstract

Test pattern generation and diagnosis algorithms that target realistic bridging faults must be provided with a realistic fault list. In this work we describe FedEx, a bridging fault extractor that extracts a circuit from the mask layout, identifies the two-node bridges that can occur, their locations, layers, and relative probability of occurrence. Our experimental results show that FedEx is memory efficient and fast.

1. Introduction

In order to remain competitive, integrated circuit 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. Such integrated circuit designs are more sensitive to manufacturing defects. This places a premium on high initial test coverage and fast and accurate fault diagnosis. In order to meet market windows, test sets must achieve high fault coverage at the start of manufacturing. Manufacturers must be able to quickly debug new products, and identify test and reliability problems in customer returns. Given the complexity of today's designs, such tasks must heavily rely on software tools to aid the engineer.

High real fault coverage [1] and accurate fault diagnosis [2] are most efficiently achieved when the software tools have a realistic list of the possible circuit faults. Most defects cause electrical shorts within the same layer or open vias between layers. Most shorts can be modeled as a bridging fault. A number of software tools to identify potential realistic faults within a circuit have been developed [3][4][5][6][7][8][9][10][11]. 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 either causing *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 [12]. Some tools such as VLASIC [3] attempt to provide an accurate circuit model for complex bridges and opens. This is too expensive for large circuits. The most common practice is to identify two-node bridging faults. Open circuits are less frequently extracted because the stuck-at fault model is often a reasonably good model for their behavior.

Some fault extractors provide an unordered 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 1. The critical area is usually weighted by 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.

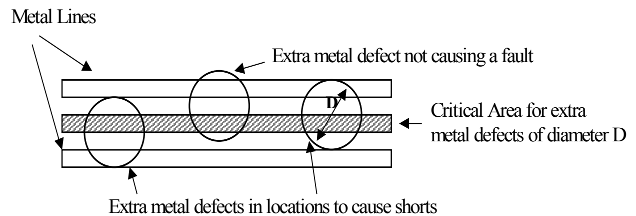

Figure 1. Critical area between two wires

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*. We have found that on average there are about two fault sites per intralayer bridging fault.

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 [13].

The exact computation of critical areas for typical layouts is costly. This is primarily because the circular

defect model implies a Euclidean polygon expansion, as shown in Figure 2 to compute the critical area between two polygons. Each polygon is expanded by half the defect 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 polygon expansion, as shown in Figure 3. Orthogonal expansions are relatively inexpensive. Another alternative is to use a Voronoi diagram of segments [14]. Another alternative is to use a circular defect and sample the layout with a Monte Carlo process to estimate the critical area. The drawback of a Monte Carlo procedure is that large sample sizes are required to ensure that faults with small critical area are identified for those applications needing a nearly-complete fault list. 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.

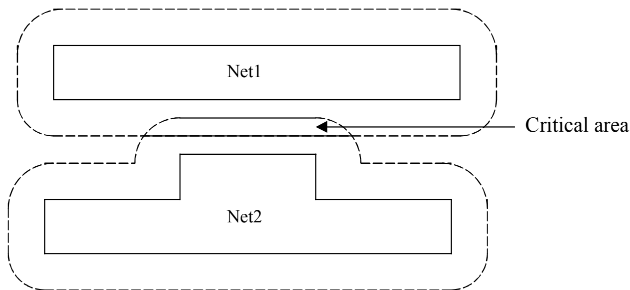

Figure 2. Euclidean polygon expansion

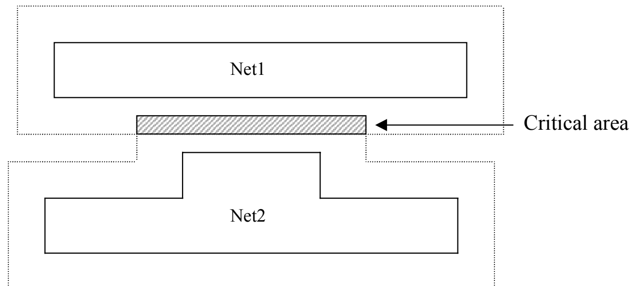

Figure 3. Orthogonal polygon expansion

Although much prior work has been expended on exact or near-exact critical area computations [12], 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 is ensuring that the more probable faults are on the fault list. ATPG will likely not target all possible realistic faults. Some fault extractors [9] use the ranking information to keep only a most likely subset of the faults. 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. We previously showed that ranking information such as WCA was not useful for diagnosis [2]. 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 [15]. 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 considerations above led us to develop a bridging fault extractor with the following characteristics:

- *Extract all two-node intralayer bridges.* Modeling multi-way bridges is too expensive for ATPG and diagnosis tools, and our expectation is that most can be tested and diagnosed by using all the two-node bridges that make up a multi-way bridge. All bridges to nets within a user-specified window around a net are considered as bridge pairs, even if there are intervening nets. So a three-way bridge will be extracted as the three two-node bridges. The window size is made large enough to consider all defect sizes likely to occur. As discussed above, we need to extract all bridging faults for diagnosis. Applications such as ATPG may choose to keep only the most probable bridges.
- *Provide hooks to extract two-node interlayer bridges.* Our past experience is that intralayer bridges are more important, so we targeted them first. We will discuss interlayer bridges later in the paper.

- *Compute approximate weighted critical area for each bridge.* As discussed above, exact WCA is not needed. Since we are keeping all bridges, we do not need the WCA to trim the fault list. It is only stored as a parameter with each fault for later use. Our goal is to estimate WCA to within 20% of the exact critical area. We assume a $1/x^3$ defect diameter distribution.
- *Compute approximate fault locations and layers.* Two nets can bridge at different fault sites - locations and layers where the defect can cause the fault. This information is required for diagnosis. Rather than computing an exact critical area shape, we compute the bounding box of the space between the bridged nets on each layer. This is sufficient to localize a fault in most cases.
- *Trade WCA accuracy for speed and memory.* The diagnosis application driving this research has little need for WCA accuracy. Of more importance is to quickly obtain the complete fault list of a new chip. The fault extractor should handle the largest designs in reasonable time. Specifically we want to be able to extract a 50-million transistor design in 10 CPU-hours on a high-end workstation. If we assume about 20 flattened mask rectangles per transistor, that is a billion rectangles. If a workstation executes one billion instructions per second, we can execute no more than 36 000 instructions per mask rectangle to meet our performance goal.

Since our primary goal is to be able to handle the largest designs on a large workstation overnight, we have named our fault extractor *FedEx*. In the sections that follow, we describe FedEx and our experimental results. Section 2 provides a description of the FedEx system. Section 3 provides experimental results on a number of chip designs, and Section 4 concludes.

2. 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) 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 4.

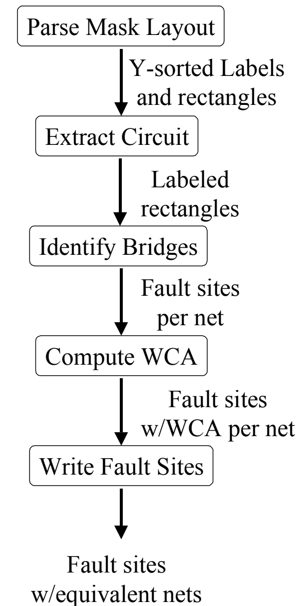

Figure 4. FedEx flow diagram

The fundamental design decision in FedEx is to start with a hierarchical design description, but to process the layout in a flat manner, and write bridging faults out in a flat manner. Our past experience with hierarchical fault extraction [4] is that the design must have a high degree of regularity to gain significant performance benefits from hierarchical analysis. Otherwise the overheads of handling the design hierarchy eliminate most of the benefits.

Our target ATPG and fault diagnosis applications focus on the logic part of a design. We assume test and diagnosis of large memory arrays is handled separately, as is current practice. There is relatively little layout regularity in most logic designs, particularly those implemented in an ASIC design style. The advantage of a flat analysis is that its performance is independent of the design style. Rather than concentrating on developing a complex hierarchical algorithm, we instead have focused on developing a fast and simple flat analysis.

We use a *scanline algorithm* [16] for both circuit extraction and fault extraction, as shown in Figure 5. 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.

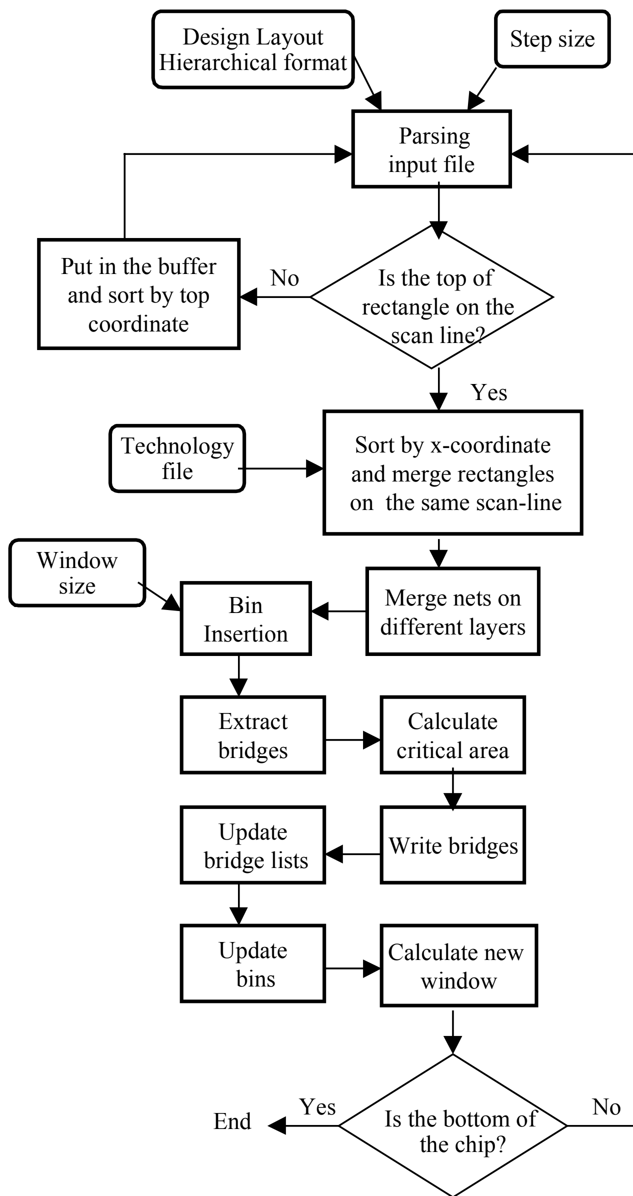

Figure 5. FedEx scanline algorithm

The memory consumption of the scanline approach is relatively small. Only a moving window of geometry is kept, which is $O(\sqrt{N})$ in the size for a chip with N rectangles. Only information about nets that intersect the scanline must be kept. When the scanline has passed below a net, that net can be discarded since there can be no further bridges to it. The number of local nets crossing the scanline will be relatively constant, following $O(\sqrt{N})$ behavior as with rectangles. The number of global nets will slowly rise towards the middle of the chip and then slowly fall. These are a small percentage of all nets. The only place where the flattened design is present all at once is in the bridge fault sites written to the output file.

2.1. Parser

The parser reads the mask layout into memory, flattens, and sorts it by y-coordinate. Polygons are converted into rectangles using a stairstep algorithm that horizontally slices any non-orthogonal section of the polygon. 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. Usually a smaller step size is chosen in order to improve the accuracy of the WCA computation. It is possible to use a separate step size for each mask layer, so that large numbers of rectangles are not created where design rules are coarse. In practice most polygons appear on lower mask layers, which have similar design rules, so using one step size for all mask layers results in little penalty.

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 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.

This sorting procedure takes advantage of the design hierarchy to reduce the sorting cost and memory consumption. If the hierarchy is a binary tree organized by y-coordinate, then the cost is only *O(N)* for *N* flat rectangles since the hierarchy already provides the sorting structure. In the worst case the cost is *O(NlogN)*, when one cell instance contains all chip rectangles. In a standard cell design style, the rectangles on most of the metal layers are described in a flat manner. However these layers are not as dense in terms of rectangles as the lower mask layers, so even in ASIC designs the priority queue greatly reduces the sorting cost. In very large ASIC designs, as well as custom designs, modules are placed and routed separately, and then assembled, which provides further hierarchy to reduce the sorting cost. Our experiments show that overall the parsing, flattening, and sorting cost is less than 8% of the total FedEx execution time.

2.2. 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 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 active list, and then the linear cost of merging is paid in the larger active list. If the inserted rectangle would overlap a neighboring rectangle, they are coalesced into one larger rectangle that retains the number of the smaller net (all nets are assigned a unique identifying number). Since the two rectangles can be of different height, the rectangle with the lower bottom coordinate is sliced off so that its height matches that of the shorter rectangle. This sliced off rectangle is placed in the buffer list. If a rectangle does not intersect any other, it is assigned a new net number. Layers that form transistors are processed in order to split active regions that are separated by a transistor channel.
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 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 practice there are a relatively small number of rectangles in the new list, so the x-sorting cost is relatively small. The cost is further reduced by skipping layers that did not have rectangle or label insertions between scanline moves. If the sorting cost became unacceptable, there are a number of O(1) time data structures that can be used to speed it up.

The 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.

2.3. Fault Extraction

The rectangles leaving the scanline are inserted into an array of bins, shown in Figure 6. There are two rows of bins, each with height and width equal to the user-specified fault extraction window size S_{max}. This is in effect 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.

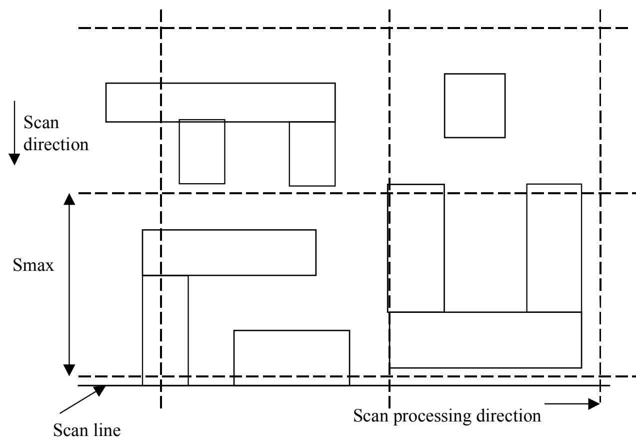

Figure 6. Fault extraction bins

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 apart as $2\sqrt{2}S_{\max}$ are considered. The analysis for rectangles that are within this distance is shown in Figure 7. For each rectangle, only rectangles to the left, upper left corner, top, and upper right corner are considered for bridges. This is to avoid double-counting critical areas as the rectangles are moved from the bottom to the top row of bins.

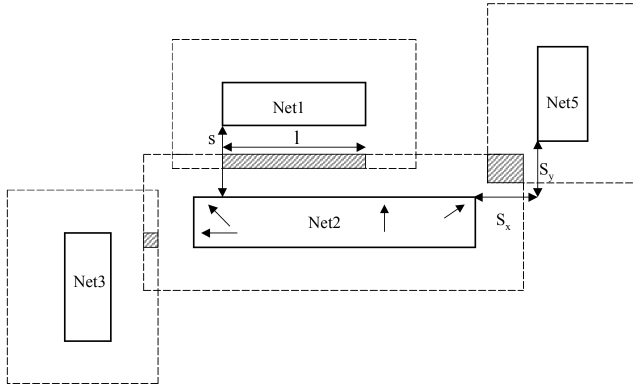

Figure 7. Critical area calculation using a geometrical method

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.

The weighted critical area is calculated using a geometrical method. Geometry corresponding to each of the nets is inflates by the half S_{max}. The intersection region defined by the bloated rectangles of two or more nets (the shaded areas) corresponds to the critical area, as shown in Figure 7. The spacing s and length l are used to compute the weighted critical area in Equation (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.

$$A = \int_{s}^{S\max} x_0^2 \frac{(x-s)}{x^3} l dx \quad (1)$$

The corner critical areas are computed with Equation (2), using the s_x and s_y values.

$$A = \int_{\max(s_x,s_y)}^{S\max} x_0^2 \frac{(x-s_x)(x-s_y)}{x^3} dx \quad (2)$$

As shown in Figure 6, rectangles on the same net will abut against another. Those parts of a rectangle edge that directly abut another rectangle on the same layer (same net) are not considered when computing critical area. Since rectangles are divided by horizontal slices, this only applies to the top and bottom edges of rectangles.

Equation (1) underestimates the critical area in that it does not include the "end effect" regions, such as to the left and right of the top critical area in Figure 7. This is not included because at each spacing change between two wires, this would result in overlapping end regions. An approximate value can be computed [8][9]. We have developed a number of approximation techniques, but felt that the improvement in accuracy was not worthwhile. The error is relatively small for longer parallel wire runs on upper metal layers. The error is largest for short wires within cells. In keeping with our goals, we only need to know that the former have large WCA and the latter do not. Therefore we did not include the end approximation in our computations.

Equation (2) overestimates the critical area due to diagonally adjacent nets. This is the same overestimate obtained through standard orthogonal expansion. Normally these are small critical areas, and so not significant. The exception is when a polygon is converted to rectangles, resulting in a net with many nearby corners. We analyze each corner separately, so the result is much more critical area than would be the case with polygons with smooth diagonal edges. Again, these normally only occur within cells, not the upper metal layers.

The result of the fault extraction for each bin is a set of fault sites, including net pair, bounding box of the critical area, the layer, and the WCA.

Interlayer bridges can be added to the analysis by extending the above algorithm to have the rectangle under consideration in the center bin check against rectangles in the bins on adjacent layers, out to some distance I_{max}, the maximum lateral interlayer bridge distance. The WCA calculation would have to add the overlapping critical areas identified during the circuit extraction.

When a row of bins is about to be discarded, the net data structures are checked to see if any will fall outside the top row of bins and should be discarded. The bridges associated with these nets are printed. This first requires checking all the net number equivalences due to merging of nets on the scanline. In some cases bridges will be discarded since they bridge to the same net. The incremental merging scheme speeds scanline processing, but the final net scan is currently implemented as an $O(N^2)$ algorithm. This can result in a significant overhead when long nets pass the scan line.

2.4. 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, 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.

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. We currently do this with a combination of C programs and shell scripts. For a list of several million bridges, this processing takes a few minutes. We do not include this time in the experimental results that follow since the amount of processing is highly dependent on the application using the bridge fault list.

3. Experimental Results

FedEx has been implemented in 14 000 lines of C code. A set of university, research laboratory, and industrial circuit designs were extracted using a Sun UltraSPARC 5/360 with 256 MB of memory. The design styles ranged from array to full custom to ASIC. The university and research laboratory designs used two and three metal layers. The extraction window size was fixed at 10 λ for all designs, where λ is defined as half the drawn transistor gate length. This distance considers defects large enough to bridge several adjacent wires together. The step size for polygon to rectangle conversion is set at 2/3 λ. Bridges on the metal, polysilicon and diffusion layers were considered.

The results are shown in Table 1. The memory consumption is not linear in transistor count. It is primarily a function of the design complexity in terms of the hierarchy and how many polygons must be converted to rectangles. The first five designs are fairly regular and their layouts contain only rectangles. The Mosaic design has parts that are very regular, but also contains a large flat microcode ROM. The controller is implemented using an ASIC design style in a four-metal, 180 nm technology. The standard cells contain polygons, so the design has both flat global wiring as well as many rectangles generated from polygon conversion. This is more typical of the fault extraction behavior on a current industrial design.

Note that for the two larger designs, the process size exceeded the main memory size. This did not cause significant paging since the working set is relatively small. Most of the memory is used to store the design data structures in the parser, and these are accessed relatively slowly compared to scanline processing.

Table 1. FedEx Experimental Results

Circuit	# Trans.	Mem (MB)	CPU Time (s)	Fault Sites
Hopfield	22K	32	31	71 223
Frame	64K	35	80	243 787
Serial	70K	18	66	194 319
Array	85K	36	125	352 166
Cross	164K	75	375	285 382
Mosaic	1200K	645	625	648 979
Controller	500K	305	4260	3 613 586

The CPU time is primarily correlated with the number of bridging fault sites extracted, which is a function of the layout density relative to the design rules. But it is also a function of the number of unique y-coordinates (scanlines) and windows (chip y dimension divided by window size) processed.

There is no common set of fault extraction benchmarks or extraction criteria, so it is not possible to directly compare the performance of FedEx with other fault extractors. For example, the L2 circuit in [9] is 8% larger than the controller circuit. Compared to [9], FedEx takes 13% of the CPU time on a machine that is 60% of the speed, so overall FedEx appears about 12 times faster. But [9] is doing more work since it extracts interlayer bridges, computes intralayer bridge WCA more accurately, and only keeps the 30 000 most probable bridges. Most importantly, the value of S_{max} used is not specified, and CPU time is very sensitive to this.

Due to the different circuit design styles and densities, it is not possible to draw any conclusions from Table 1 about the asymptotic complexity of the FedEx algorithm. The primary concern is the behavior of the scanline processing. To study this, a chip that consists of a row of two copies of the controller circuit was extracted. Since we wanted to avoid significant paging, the circuits were extracted on a HP 9000 Model V2200, which has a larger physical memory. The results are shown in Table 2. Larger circuits could not be run due to virtual memory limits. The memory usage grows sublinearly since the fixed overheads (e.g. design description) do not grow. This would be true of any hierarchical design with some regularity. The runtimes are approximately linear in the circuit size, so the $O(N^2)$ factor associated with the current scanline implementation is relatively small.

Table 2. FedEx Scaling Results

Circuit	Memory (MB)	CPU Time (s)
Controller	320	5000
2 x Controller	493	10888

On the controller chip, FedEx processed 10 743 000 flattened mask rectangles in 4260 CPU-sec on an

UltraSPARC 5/360, or about 140 000 instructions per rectangle. This is extracting about 7 fault sites and 3.5 unique faults per transistor, which is a relatively high fault density. Based on this we are within a factor of 4 of our ultimate performance goal in terms of instructions executed per mask rectangle, and that while FedEx cannot yet handle the largest designs overnight, it can handle them within a few days.

4. Conclusions

In this work we have developed the FedEx two-node bridging fault extractor. By trading accuracy for time, and using efficient data structures and algorithms, FedEx is able to perform circuit and fault extraction at high speed with reasonable memory consumption. Our experimental results show that it is very competitive with other fault extractors.

We have integrated FedEx into the Knights Technology Merlin™ database as part of the Computer-Aided Fault to Defect Mapping (CAFDM) project [2]. In the Merlin version, the circuit extraction has already been performed, the parser is replaced by database region queries, and critical area calculation is not performed since it is not needed for the diagnosis task.

The Merlin version of FedEx has been run on a 4M-transistor circuit implemented in six-metal 180 nm technology. The chip has 1M nets, of which 300 000 are between cells, and 25M bridge fault sites. A 20M transistor chip is also being extracted using this system. This version of FedEx is currently much slower than the version described here due to the way it was interfaced to the database. We expect it to be of similar performance to our parser-based FedEx when it is fully optimized.

We would like to speed up FedEx by at least another factor of 4. The netlist merging when a net is complete currently uses an $O(N^2)$ algorithm. This netlist merging takes almost half the execution time of the controller chip. Scan line manipulation can be improved by using a $O(1)$ bin structure, rather than a linked list. This ensures that the runtime behavior on very large chips is linear. The parser memory consumption can be significantly reduced. Multiple copies of some structures are kept, and structures could be freed immediately after they are used. This is the majority of the memory consumption in the benchmarks. The combination of these improvements should achieve our goal of extracting a chip with 50 million logic transistors (e.g. the IBM POWER4 microprocessor [17]) overnight on a large workstation.

FedEx is being implemented on a 16-CPU shared memory parallel processor. The parser runs on one processor and provides data to the scanline processing, which is done on the remaining 15 processors. Net merging and I/O is done on the first processor. Given the balanced load and independent nature of the bin processing, we expect a nearly linear speedup.

Acknowledgements

This research was funded in part by International SEMATECH under project DRTB002. This project has benefited from the continual guidance of the program manager, Fred Lakhani. This project has also benefited from many interactions with Hari Balachandran of Texas Instruments.

References

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

[2] 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”, *IEEE Int’l Test Conf.*, Oct. 2000, pp. 729-738.

[3] H. Walker and S. W. Director, “VLASIC: A Catastrophic Fault Yield Simulator for Integrated Circuits,” *IEEE Trans. CAD*, Oct. 1986, pp. 541-556.

[4] D. D. Gaitonde and D. M. H. Walker, “Hierarchical Mapping of Spot Defects to Catastrophic Faults -- Design and Applications”. *IEEE Trans. on Semi. Manuf.*, May 1995, pp. 167-177.

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

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

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

[8] S. T. Zachariah, S. Chakravarty, and C. D. Roth, “A Novel Algorithm to Extract Two-Node Bridges”, *Design Automation Conference*, June 2000, pp. 790-793.

[9] S. T. Zachariah and S. Chakravarty, “A Scalable and Efficient Methodology to Extract Two Node Bridges from Large Industrial Circuits”, *IEEE Int’l Test Conf.*, Atlantic City, NJ, Oct. 2000, pp. 750-759.

[10] C. H. Ouyang, W. A. Pleskacz and W. Maly, “Extraction of Critical Areas for Opens in Large VLSI Circuits”, *IEEE Trans. CAD*, Vol. 18, Feb. 1999, pp. 151-162.

[11] HPL, Inc., SAFARI System, 2001.

[12] 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, ed., IEEE Press, 1998.

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

[14] E. Papadopoulou and D. T. Lee, “Critical Area Computation Via Voronoi Diagrams”, *IEEE Trans. CAD*, Vol. 18, April 1999.

[15] C. E. Stroud et al, “Bridging Fault Extraction from Physical Design Data for Manufacturing Test Development”, *IEEE Int’l Test Conf.*, Atlantic City NJ, Oct. 2000, pp. 760-769.

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

[17] C. Anderson, J. Petrovich, J. Keaty and G. Nusbaum, “POWER4 Physical Design”, *IEEE Int’l Solid-State Circuits Conf.*, San Francisco, CA, Feb. 2001.