Some Future Directions in Fault Modeling and Test Pattern Generation Research

F. Joel Ferguson and Tracy Larrabee
Computer Engineering Department
University of California, Santa Cruz
Santa Cruz, CA. 95064

Abstract

This document presents the current state of fault modeling research and lists research options that leverage the Carafe–Nemesis software packages and the knowledge gained from their use.

1 Current State of Fault Modeling Research

The most widely accepted fault model is the single stuck-at (SSA) fault model. It is easy to use and it models many defects that occur in digital circuits, especially TTL circuits [TBG+83]. If all SSA faults are covered by a test set[1], each signal line in the circuit has its value observed through sensitized paths at least twice — once as a logic 0 and once as a logic 1. Due to this nice property, many non-SSA faults are fortuitously detected by SSA test sets. However, as quality level requirements become more stringent the non-SSA faults that remain undetected may cause an unacceptable number of faulty ICs to pass manufacturing test.

We now introduce some terminology for which no standards have been set. A *defect* or *spot defect* is a local (less than 10 microns long) perturbation during manufacture that changes the circuit to one whose behavior differs unacceptably from the ideal circuit[2]. A *circuit-fault* is a deviation in the connectivity of conducting and semiconducting regions and is caused by a defect. A *fault* is the resulting change in behavior that results due to the circuit-fault. The fault, or change in behavior, is what is exploited during the testing procedure to detect manufacturing defects. There are three important classes of faults:

[1] We define a *SSA test set* as a test set that was generated by targeting SSA faults. A *complete SSA test set* is defined as a test set that detects all SSA faults in the circuit.

[2] We choose not to consider spots of material that don't affect the circuit's behavior as defects. This makes the term defect coverage more intuitive.

changes in the logic function, which we call *logic faults*; increased propagation delay, which we call *delay faults*; and increased quiescent power supply current, which we call I_{DDQ} *faults*.

Consider as an example an opaque spot on an area of the metal fabrication mask that is normally transparent, causing an undesirable spot of metal to exist on the physical circuit. If this metal spot intersects other metalized areas in the circuit representing different circuit nodes, it is a defect. The resulting circuit-fault is that the two affected circuit nodes are joined to form a single node. The resulting fault is classified by its behavior. For instance, if one of the nodes was a node to V_{dd} and the other was the output of a gate then the fault would be a logic fault, more specifically, a stuck-at 1 fault. If the quiescent power supply current of the circuit is low enough and the output node static, then it is also an I_{DDQ} fault. In either case the detection of the defect involves applying the correct stimulus (inputs) and observing the result (either output value or power supply current, respectively).

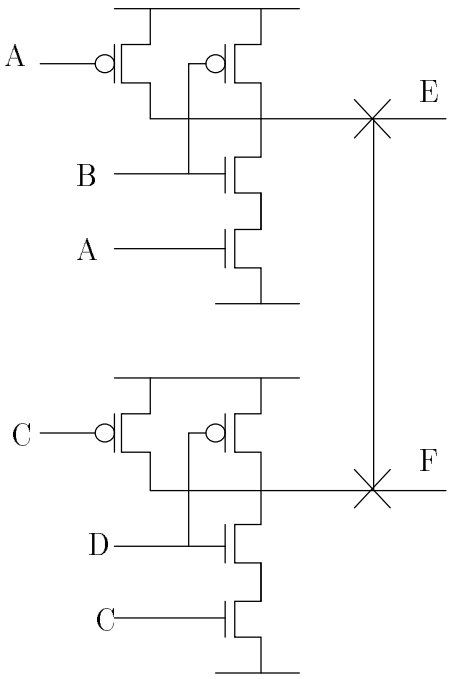
Defect simulation experiments show that most spot defects cause shorts between nodes or breaks in a node. We call the resulting circuit-faults bridges and breaks, respectively. Breaks within a CMOS circuit may cause sequential behavior. These are equivalent to transistor stuck-off faults. Empirical evidence shows that bridges are more prevalent than breaks or transistor stuck-off faults[TLPM85, MTCC87, WNS87].

If the circuit's state and inputs have specific values, many bridges and breaks can cause the circuit to use an abnormally large power supply current after the transients in the circuit have died down. In that case the defect can be modeled as an I_{DDQ} fault and be detected by measuring the power supply current[Ack83, HS86, MNN88, FL91a].

Many bridges and breaks that do not cause an incorrect logic function increase the propagation delay of the circuit. An excellent description of the problems of delay fault testing and categories of delay fault tests is given by Pramanick and Reddy[PR88] and are not be discussed here.

Traditionally bridge faults have been modeled as a wired-AND or a wired-OR function on the two bridged nodes, that is, the logic 0 is always stronger than the logic 1 or vice-versa. This is often an incorrect assumption in CMOS circuits, where a bridge may result in a voltage on the affected nodes that may be interpreted as a logic 0 by some cell inputs and a logic 1 by other cell inputs (these are called indeterminate values). Recent research in test pattern generation for bridge faults has focussed on obtaining a more accurate model of the logic value resulting from the bridge. For CMOS non-feedback bridge faults Acken's voting model is used[Ack88]. The voting model states that when there is a bridge between nodes and each node is being driven to a different value, the resulting voltage is determined by a "vote" between the pullup path(s) and the pulldown path(s), where not all paths have the same strength. An example of the voting model is shown in Figure 1. The figure shows that two PMOS transistors in parallel are stronger than the NMOS transistors in series for the NAND gates in the CMOS3 standard cell library [Hei88], but a single PMOS transistor is not. The transistor strength model used in switch-level simulators, such as COSMOS, cannot model this fault correctly by assigning any combination of strengths to the eight transistors.

Another issue is that different cells (or gates) and different inputs to the same cell (or gate)



Truth Table entries for E $\neq$ F

ABCD	EF	Wired-And	Wired-Or	Spice
0111	10	0	1	0
1011	10	0	1	0
0011	10	0	1	1
1101	01	0	1	0
1110	01	0	1	0
1100	01	0	1	1

Figure 1: Logic Function of Bridge Fault in CMOS3 NANDs using Spice.

tend to have different logic thresholds due to different gate-to-source voltages for conducting transistors in series. Our circuit simulations of the CMOSn cell library show a difference of almost a volt for different inputs of a 3-input NAND gate.

Bridge faults may also cause sequential behavior when occurring between two signal lines[Mei74]. Using circuit simulations we have shown this to be true within a single CMOSn standard cell.

Lastly there are bridge faults that occur within cells or gates that cause errors to occur at the inputs of the gate. An example of this class of bridge faults is illustrated in Figure 2. There is a bridge fault between the gate of the upper nMOS transistor and its source (this fault type has been described by Hong and McCluskey). If the value of {A,B} is {1,1} and the nMOS transistor being controlled by A is stronger than the pMOS transistor in the inverter being driven by B, the logic 1 on the input to inverter C will be forced to a logic 0. This causes the C output to be a function of A. Hong calls this a "pattern-dependent fault".

The next most common circuit-fault type is breaks. Breaks within a primitive or complex gate that do not involve the input and output nodes can be placed into one of two categories: Those that break all paths from the output node to either Vdd or ground (see A in Figure 3), and those that break one but not all paths between the output node to either Vdd or ground (see B in Figure 3). The former behaves as a stuck-at fault after the input acquires the appropriate charge, and the latter transforms the combinational circuit to a sequential circuit due to trapped charge[Wad78].

It is more difficult to predict the behavior of the circuit when a break is in the node serving as a signal line. This is because such breaks cause the gates of transistors to float

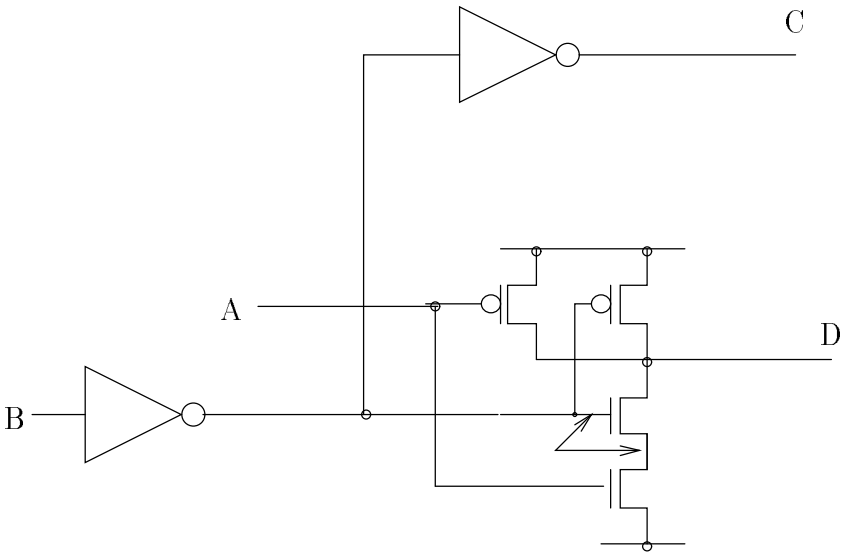


Figure 2: Bridge Fault Affecting other Inputs.

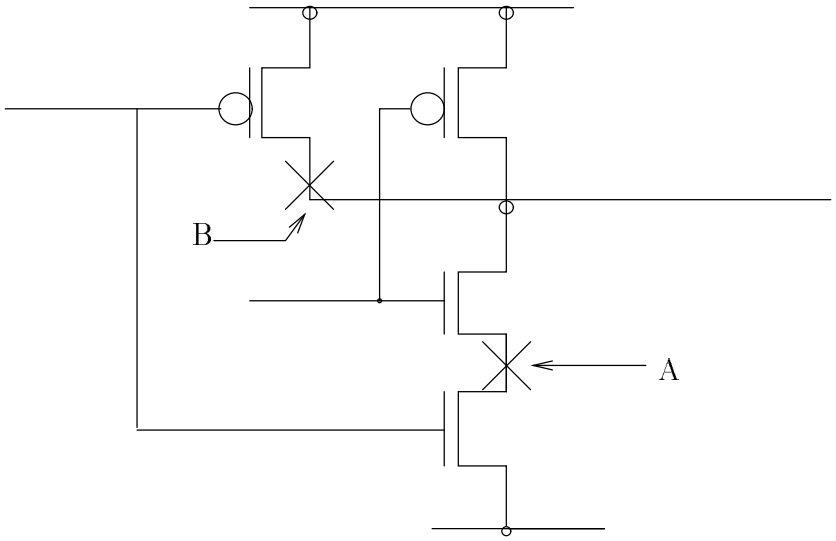


Figure 3: Non-gate-node bridge faults.

to a difficult-to-predict voltage making it difficult to ascertain the resulting behavior[3]. In experiments with MOSIS supplied chips with fabricated opens, the transistors with floating gates tended to be "weakly on"[MNN88]. More recent studies by Rodríguez-Montañés, et al. show that accurately modeling breaks in the input nodes requires knowledge of the capacitances to other nodes in the circuit[RMSC+91].

In order to determine which bridges, breaks, and delay faults are the most likely requires that the layout of the circuit and the known fabrication defects be taken into account. Since circuits are generally irregular and large, this should be done automatically in software. The Carafe fault extractor was developed for this purpose[Jee91]. The next section describes Carafe and possible future enhancements. Once Carafe has determined the likely bridge and break faults, a test pattern generation system needs to produce tests that detect the presence or absence of each fault. Section 3 describes the Nemesis ATPG system along with some of its possible future enhancements. Section 4 shows how Carafe, Nemesis, and other software are combined for ATPG of realistic faults. Section 5 describes how the physical design of the circuit can be changed to make it more testable.

2 The Carafe Fault Extractor

Carafe takes as input the layout of a CMOS circuit (in Magic or GDS II format) and a statistical description of the defects that occur during manufacture. The description of defect statistics consists of the relative defect density of each layer of the circuit (polysilicon, metal 1, field oxide, etc.) and the distribution of defect sizes for each layer.

Carafe performs a circuit extraction and presents a list of possible bridge and break faults[4]. The relative likelihood of each circuit-level fault is given based on the circuit's layout and defect statistics. For instance, assume that the size distribution of defects on the metal 1 and polysilicon layers are the same and that the metal 1 bridge defect density is 5 times greater than the polysilicon defect density. If nodes A and B are adjacent to each other in the polysilicon layer for 20 microns, and nodes A and C are adjacent to each other on the metal layer for 40 microns and the distance between A and C is the same as that between A and B, then a bridge or short between nodes A and C is 10 times as likely than one between nodes A and B (40*5 vs. 20*1). The significance of this is that a test that detects only the bridge between nodes A and C detects 10 times as many bad chips as a test that detects only the bridge between nodes A and B. Hence the defect coverage and the fault coverage of a circuit can be quite different even if all realistic faults are considered and defect coverage is a better indicator of the quality level of the product than fault coverage.

Carafe can either be run in batch mode or interactively. Batch mode allows Carafe to extract and record the fault list for large circuits, extract the fault list from several circuits, or perform multiple analyses on the same circuit with different defect statistics. The interactive

[3]We are assuming that all cells are complex gates or are composed of complex gates. Hence all inputs to a cell are applied to at least one nMOS and one pMOS transistor.

[4]Break faults will be incorporated into Carafe in the near future.

mode uses X-windows to display the circuit and any of the faults of interest. This allows the designer to view the more troublesome faults and perhaps redesign the circuit to eliminate them.

Several researchers have suggested enhancing Carafe to support faults that are not currently considered. Below we consider two enhancements: the extraction of resistive bridges and breaks, and the extraction of reliability faults. We then present the difficulties of making these enhancements.

2.1 Non-critical Faults

We define a *non-critical fault* as a circuit-fault that does not result in a complete short or open. Carafe currently determines the relative likelihood of faults that can be modeled either as a zero resistance short between two nodes or as a break that completely severs an electrical node into two subnodes. Many spot defects cause non-critical circuit-faults that cannot be modeled this way. The non-critical faults that we are aware of in CMOS circuits fall into one of four categories: resistive shorts, resistive opens, shorts in the gate oxide, and transistors that do not switch on or off completely.

Examples of resistive breaks are losses of conducting material in contacts or polysilicon regions that do not completely break the node, or oxide that was not completely removed from a via. Similarly there can be additional conductive spot defects that cause resistive shorts between nodes. Gate oxide shorts are caused by pinholes in the gate oxide region and result in non-linear resistance between the gate and the source, drain, or channel. The fourth category — transistors that do not switch on or off completely — can be caused by many types of defects. A transistor that is partially on when it should be off can be caused by missing polysilicon over the transistor channel region causing the channel to be too short over part of the gate; this results in off-current as shown in figure 4. Other circuit faults may also be manifested as a partially-on transistor. For example, a break in a node that leaves a single transistor gate floating has been shown to sometimes cause the transistor to be partially-on[MNN88], as can gate oxide pinholes[Syr87]. There may be spot defects that affect the diffusion doping or field oxide that may reduce the effective transistor channel width and thus cause reduce the strength of the transistor. An example of this is shown in Figure 5.

For non-critical faults to fit into the Carafe framework requires that there be some method of estimating the relative likelihood of the non-critical circuit faults. Studies of pinholes in the gate oxide have provided some information on how to estimate their relative likelihood. It appears that in most processes, the probability of a gate oxide pinhole occurring is proportional to the area of the transistor channel. For other processes there is evidence of a dependence on the length of the perimeter of the transistor channel region. It is reasonable to expect that the probability of a resistive short in a specific layer is primarily determined by how far apart the two nodes are and the extent that they are adjacent to each other; the probability of a resistive open is related to the length, thickness and layer of the conducting node. This is what is currently done with the critical faults modeled by Carafe, so

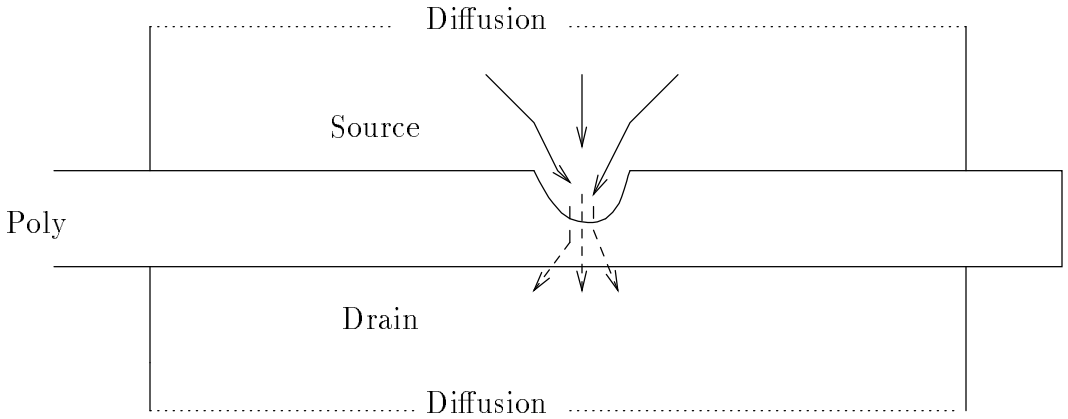


Figure 4: Missing Polysilicon Causing Transistor Never-off Circuit-Fault.

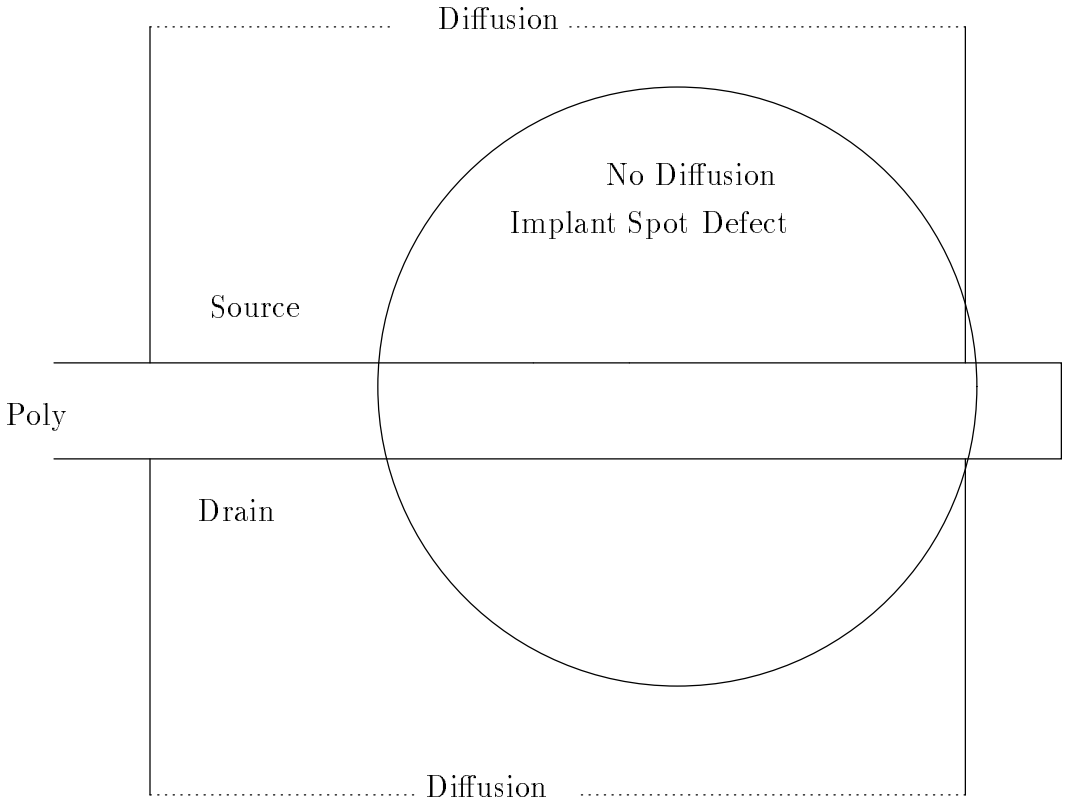


Figure 5: Missing Diffusion Causing Weak-Transistor Circuit-Fault.

the sensitive area approach used by Carafe should work for most occurrences of non-critical faults.

For the purposes of this report *fault strength* can be understood to be a measure of how close to critical the fault is. A resistive short that has high resistance has low fault strength and one that has very little resistance has high fault strength. Little is known of the relative probability of different values of fault strengths and how they are affected by the layout of the circuit.

Incorporating non-critical faults into Carafe and determining the resulting faulty behavior will require more sophisticated defect models, knowledge of the distribution of fault strengths, and more sophisticated circuit-fault to behavior-fault translation to reduce simulation time. Finally many non-critical faults do not result in changes in logic function and are detectable only as delay faults.

2.2 Relibility Fault Extraction

The Department of Defense has contracted with Sandia National Laboratories and the University of California at Santa Cruz to improve their "reliability fault modeling" procedure for digital sub-circuits. The purpose of reliability fault modeling is to accurately model the logic behavior of circuits if and when they fail in the field. The current procedures are very expensive and use an *ad hoc* fault model.

We expect Sandia and researchers at the University of New Mexico to provide us with the descriptions of defects that affect the long-term reliability of ICs. We can then modify Carafe to report likely reliability faults so that they can be extracted automatically just as manufacturing defects are now.

Many reliability faults start as a non-critical fault as discussed earlier. Examples include changes in the threshold of transistors due to trapped charge in a transistor's $Si - Si0_2$ interface, decreased resistance over time from gate oxide pinholes, and increased resistance in metal lines due to metal migration. These first become delay faults. If and when the short or break becomes critical the logic function of the IC changes to the value that Carafe now predicts for bridges and breaks. Before the circuit's function reaches it critical fault value, it may exhibit faulty logic functions other than its critical function.

If we decide to extract the reliability faults, we must determine whether to extract and determine the final critical-fault function, to extract and simulate the intermediate non-critical fault functions, to extract the delay faults, or do all of these. If we choose to do anything other than determine the final critical-fault function, then Carafe must extract the non-critical faults and we must develop more efficient procedures for determining their faulty behavior.

2.3 Problems

There are few challenges to enhancing Carafe to extract non-critical and reliability faults. However there are two problems relating to the accuracy and use of the resulting realistic

fault list:

1. **A lack of knowledge of causes of reliability and non-critical faults.** To accurately predict which faults may occur, one must know what defects cause these faults and what conditions must exist in the physical design to make the circuit susceptible to the faults. To accurately predict the relative likelihoods of the resulting faults, one must know the defect density, size, and strength distribution of the defects. Much of this is technology-specific and Carafe's technology file can be expanded to include these defect and fault types. The limitation of the accuracy of the resulting fault list depends heavily on the accuracy of the defect models and distribution.

2. **An inability to efficiently fault simulate the resulting fault list.** Even with critical bridges and breaks the resulting fault list is large and, as we discuss in Section 4, the resulting faults may be difficult to translate to changes in Boolean behavior. A continuous range of fault strengths for each bridge, break, and transistor fault complicates matters much more. Finally, since many non-critical and reliability faults may be more easily detected as delays, it may be necessary to simulate these in such a way as to automatically detect changes in delay and find all delay faults.

3 The Nemesis Automatic Test Pattern Generator

Nemesis takes as input a description of the circuit, a list of faults for which it must generate tests, a list of fault types, and a list of primitive bridge functions describing the logic change in function for each type of fault. Carafe has been modified to provide all of the input except the primative bridge functions, which are provided by the Bridger program. We discuss the Bridger program in greater depth in later sections.

Nemesis differs from most existing ATPG systems. In the past three decades, practical automatic test pattern generation (ATPG) systems, beginning with the Roth's D-algorithm, have pursued the problem using essentially the same structural-search paradigm [FS83, Goe81, Rot66, STS88]. Progress has been steady, but slow: the best systems of today do not work well on large circuits, sequential circuits, or models of failure other than the single stuck-at fault. These restrictions are unworkable in the long term as the size of integrated circuits increase, design methodologies fail to transform all sequential circuits into circuits that can be tested using combinational techniques, and evidence mounts that many IC manufacturing defects are not detectable as single stuck-at faults.

Nemesis is a successful ATPG system that uses a completely new approach called the Boolean satisfiability method [Lar92]. The new method generates a test pattern for a given fault in two steps: First, it constructs a formula representing all possible tests for the fault. Second, it applies a Boolean satisfiability algorithm to the resulting formula. This method is general and effective; it allows for the addition of any heuristic used by structural search methods, and it produces excellent results on the ISCAS-85 set of testing benchmarks collected by Brglez and Fujiwara[BF85].

Nemesis's separation of the formula extraction from the formula satisfaction provides great flexibility because the same satisfier can be used with many different extractors. Nemesis generates tests for defects that cause an increase in the quiescent power supply current because of bridges or stuck-on transistors, and it generates tests for bridge defects (feedback and non-feedback). In the process of modifying Nemesis to handle these additional faults, we have improved both our algorithmic test pattern generation and our fault simulation routines. For the larger benchmark circuits, our bridge fault simulation techniques offer a significant advancement over existing methods [AM85].

As we work on the integrated Carafe-Nemesis system, there are some improvements we are considering that affect only Nemesis (and not Nemesis's interface with other software in our system). Below we consider two enhancements to the Nemesis system: Improvements to the Satisfier (the back-end of Nemesis), and the use of Boolean satisfiability in sequential ATPG.

3.1 Improvements to the Satisfier

The first improvement to the Satisfier we are interested in is the addition of testability measures. Testability measures are estimates of the relative difficulties of controlling or observing 0's or 1's on given wires. Structural search test pattern generators have had much success with testability, and although we anticipate a less dramatic effect in a Boolean Satisfiability system (because of the existing propagation heuristics), we would like to investigate the benefit of adding testability measures to our system.

Having calculated testability measures for each wire, the obvious use of these measures in the Boolean satisfier is to use them to influence variable ordering. We hypothesize that testability measures will give us the largest payoff in the area of controllability (line justification), since Nemesis already has excellent heuristics with respect to observability (fault propagation).

The second improvement to the Satisfier that we are interested in is parallelization. We will have the option of using existing techniques for parallel satisfiability or developing a more specialized version that exploits our unique approach to satisfiability. We will also have the option of simultaneously starting orthogonal problems on different processors and devoting additional processors to the solution showing more progress.

3.2 Sequential ATPG

We have developed a preliminary ATPG system for sequential circuits with a reset state. Many test generators for synchronous circuits use a technique called time frame expansion, which unrolls the sequential behavior of the circuit. That is, if a test sequence takes N clock cycles, unrolling the circuit N times produces N copies of the combinational part of the circuit, cascaded by connecting the next state lines of the combinational circuit in time frame t-1 to the present state lines of the combinational circuit in time frame t. After time-frame expansion, a combinational ATPG system can be used to search for a test vector.

When a faulty sequential circuit is time frame expanded, each copy of the combinational part of the circuit contains the same fault. Thus, a combinational ATPG system that works on single-stuck-at faults can not be directly used, since the unrolled circuit is a multi-fault combinational circuit. The underlying combinational ATPG system needs to handle multiple faults. Since Nemesis has no difficulties dealing with multiple faults, we were able to quickly build a sequential version of Nemesis for preliminary investigation. Our preliminary results are promising, and we will now proceed with extraction of the state transition graph so we can do more sophisticated analysis (including identification of sequentially untestable faults).

4 Accurate Fault Grading and ATPG

A major goal of our research on accurate fault modeling was to provide a more accurate measure of the quality level for a test set. A test set's fault coverage is the percentage of logic changes (within the fault model) that the test set detects. A weakness of the fault coverage metric is that many faults in an IC may not be represented by the fault model. This can be corrected by using the fault extraction techniques of Carafe coupled with finding the logic behavior of each extracted circuit fault. The second weakness of using fault coverage is that all faults are implicitly considered equally important since they are given the same weight in the fault coverage figure. In reality the probability of occurrence for different faults was as high as 14 to 1 in a relatively small circuit[MFS84].

With a list of realistic faults and their relative likelihoods of occurrence, a much more accurate estimate of the quality of a test set can be made. This metric gives the percentage of defects that the test set detects, allowing the designer to be able to directly relate this realistic defect coverage to the probability that any chip passing the test set has no Boolean faults. McCluskey and Buelow's formula, shown as equation 1 relating test transparency, yield, and quality level can then be used to estimate the resulting quality level[McC85]. In equation 1, QL is the fraction of the parts that pass the test that is good, Y is the fraction of the manufactured parts that has no defects, and TT, the test transparency, is the fraction of all defects that is not detected by the test. This formula makes the assumption that defects are independent and hence does not take into account defect clustering but it can serve as a basis for comparison until there is a demand for even more accuracy.

$$QL = Y^{TT} \tag{1}$$

For more accurate estimates of quality we use defect coverage, instead of fault coverage. A test set's *defect coverage* is the percentage of fault causing defects (within the defect model) that the test set detects. Within the accuracy of the defect statistics used for the defect model, the test set's defect coverage is the percentage of faulty chips that is detected by the test set. Hence the defect coverage is 1-TT. The defect coverage can be obtained by first assigning each fault a weight based on its relative likelihood. A fault simulation is then run on each fault. The defect coverage, as a percentage, contributed by each fault is its weight

divided by the sum of the weights of all defects. The defect coverage is shown in equation 2 where k_f is equal to 1 if the fault is detected by the test set and 0 if it is not detected.

$$DC(t) = \frac{\sum_{j=1}^{\#faults} k_f(L_{fault_j})}{\sum_{j=1}^{\#faults} L_{fault_j}} \tag{2}$$

Since many of the faults that are extracted using the fault extraction procedure are not SSA faults, we have developed software to accurately model the logic-level behavior of the most commonly extracted faults (bridge faults), and have enhanced our fault simulator so that they can be simulated. Our next major goal is to accurately model the logic-level behavior of break faults and enhance the fault simulator accordingly.

4.1 Our System for Defect Grading and ATPG

Another major goal of our research is to develop procedures that generate tests that target the likely faults in the circuit. We modified the Nemesis ATPG system to generate tests for the extracted bridge faults and hope to incorporate breaks into it in the near future.

The testing procedure that we advocate consists of partitioning the circuit into two categories: interconnect and logic. The discussion in this report is restricted to the domain of fault modeling and ATPG for circuits designed using standard cells. The approach presented here can be extended to apply to gate arrays and other ASIC technologies, but is more easily implemented for standard cell designs.

Our strategy is to use circuit simulation for as small a part of the circuit as possible, translate this to a change in local logic function, then after it is in the Boolean domain, fault simulate and generate tests for the circuit using techniques that have been developed for SSA ATPG.

We first consider bridge faults in the interconnect. The voltage of the two nodes being bridged together is a function of the logic values on the inputs of the two gates whose outputs are bridged together. Figure 1 shows the resulting logic values for such a bridge between two NAND gates. In general only the gates whose outputs are bridged and the gates whose inputs are driven by the outputs require circuit simulation to determine the logic behavior of the fault[5]. The remainder of the circuit can be simulated in the Boolean domain. There is a maximum of n^2 bridge faults where n is the number of standard cells. For bridges within the logic, only a single cell needs to be simulated and this can be done as part of the cell characterization procedure. Each fault that is circuit simulated is likely to have several instances in the circuit — all bridge faults between the 2-input NAND cell and the 3-input NOR cell would result in the same voltages, thus reducing total simulation time considerably.

Alternative strategies often involve test pattern generation at the transistor level only, which will probably remain impractical. What we advocate exploits two levels of hierarchy for test pattern generation: We determine the logic behavior of realistic circuit-level faults

[5]If the threshold voltages of the gates whose inputs are the bridged nodes have been determined before the fault simulation, then only the two gates whose outputs were bridged need to be simulated.

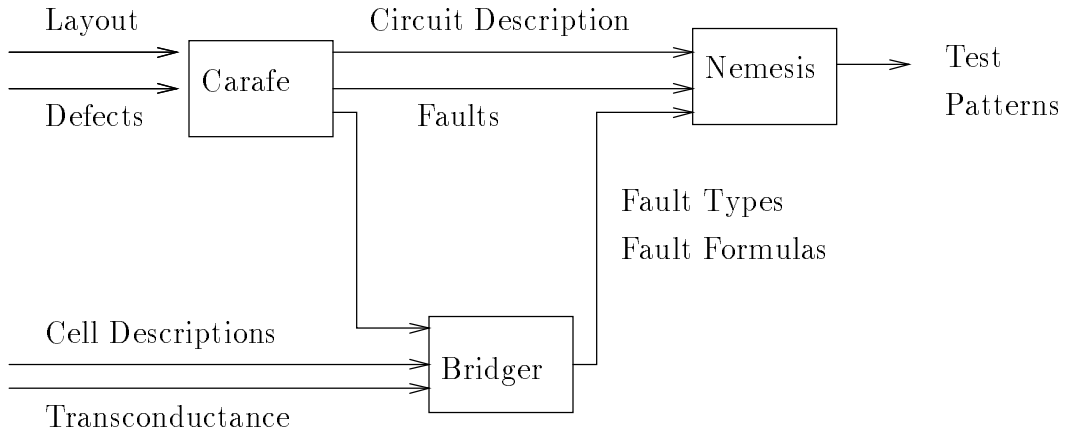


Figure 6: Existing Fault-Extraction/Test-Generation System.

extracted by Carafe by simulation at the circuit level, and we propagate errors and justify line values at the Boolean algebra level.

We have modified the Carafe fault extractor and the Nemesis automatic test pattern generation system so that the hierarchal nature of standard cell designs can be exploited as previously described. The modified Carafe recognizes the standard cells and extracts faults only in the interconnect. The faults within a standard cell have been pre-processed with Carafe beforehand and simulated. A new piece of software, Bridger, provides the circuit-fault (from the fault list generated by Carafe) to logic-fault translation that is necessary to translate Carafe's output to a local change in logic function. An example of this is the truth-table for the bridge in Figure 1. Carafe then sends the fault lists and circuit, and Bridger sends the truth tables, to Nemesis. Nemesis can then generate tests for the realistic faults. The Carafe-Bridger-Nemesis system is shown in Figure 6[FL91b].

The Bridger program is necessary for interconnect bridge faults because their behavior cannot be modeled as a simple wired–or or wired–and. More generally, the resultant voltage for a bridge between the outputs of two cells can be modeled by replacing the two cells with a single bridge-cell implementing the logic function of the bridged node (the logic function of the new cell is known as a primitive bridge function). Figure 7 illustrates this general approach. Nemesis can generate tests for any bridge with any primitive bridge function presented to it by Bridger.

4.2 I_{DDQ} Testing

Many defects do not cause a change in the logic function of the circuit. They are often detectable as either a delay fault or as an increase in the quiescent power supply current. We call the latter an I_{DDQ} *fault*. Only defects in circuits with little quiescent power supply current, such as static CMOS gates, can be detected using this technique.

It can be shown that for complex and primitive static gates, all source-drain, gate-drain, gate-source, and gate channel bridges are detected as I_{DDQ} faults by any test set that applies

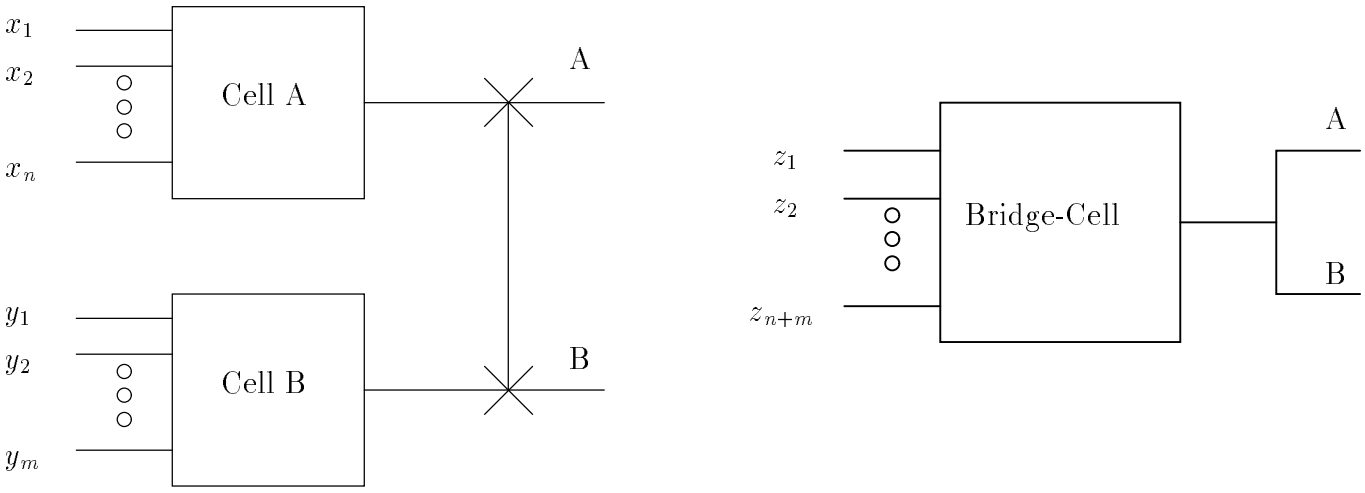


Figure 7: Modeling of Interconnect Bridge Faults.

a SSA test set to the appropriate gate-level description of the circuit as if the gate's output was a primary output. There is no need to propagate a logic error value to a primary output of the circuit. Only the inputs to the gate must be justified to the primary inputs by the ATPG software.

Theorem: Any short between the source, drain, and gate in any MOS transistor within a static CMOS gate produces excess quiescent current during at least one of the tests in any test set that sensitizes all the SSA faults of the appropriate logic level representation of the gate as described by Reddy, et al[RRK83].

Proof: Consider an arbitrary NMOS transistor T being driven by input signal s.

A stuck-at 0 test for signal s requires that there be a path of on-transistors from the source of T to ground and from the drain of T to the output, and the value of s is 1 (which turns on T)[RRK83]. Since the source and drain of T are connected to ground and the gate is at logic 1, excess current results for any short between the gate and either the source or drain.

A Stuck-at-1 test for signal s requires that there be a path of on transistors from the source of T to ground and from the drain of T to the output which is at logic 1. The value of s is 0 during this test. Since the source of T is connected to ground and the drain is connected to the output, which is connected to Vdd, excess current results if there is a short between the source and drain.

The case for PMOS transistors is analogous. □

It is well known that any SSA test set also excites all gate oxide pinhole shorts in full CMOS gates as I_{DDQ} faults.

Note that since propagation of errors are not necessary for I_{DDQ} testing, only the inputs to the gate must be justified to the primary inputs. A simple modification of existing ATPG SSA fault software to fault simulate and generate tests for all transistor shorts (between gate, source, or drain; and also gate oxide pinhole shorts) is to treat all gate outputs as primary outputs of the circuit[FL91a].

The Carafe-Nemesis software for standard cell designs has been modified for I_{DDQ} testing so that it fault simulates and generate tests for all transistor shorts and likely bridge faults between signal lines. Since the standard cells in the library are static gates it is unnecessary for Carafe to present a transistor stuck-on fault list — Nemesis uses the above Theorem to generate its tests. Carafe presents Nemesis with a list of bridges between signal lines and Nemesis generates tests for them by forcing different values on the two potentially bridged nodes.

4.3 Future Work in Fault Grading and ATPG

One area of improvement in our system is the accuracy of the circuit-fault to logic-fault translation. There are two requirements for accurate translation: the output voltage of the faulted nodes must be correct, and the logic thresholds of the inputs must be known. The Bridger software for realistic CMOS bridge faults is being made more accurate by using a circuit level simulator for the cases that require more accuracy.

Even with accurate voltages, an indeterminate range of voltages exists if one uses the same logic threshold for all CMOS gates. This is because each input to a standard cell is likely to have different logic thresholds due to the different gate-to-source voltages of two transistors in series when the gate voltage is the same. If logic values are computed from the voltages of bridged nodes without considering which inputs are being driven by the node, then the voltage range for indeterminate logic values must encompass the lowest and highest threshold values in the cell library. The alternative is to translate the faulty voltage to a logic value for each cell input. This is possible since each input is parameterized to within a few tenths of a volt[6].

The second most likely circuit-level fault after bridges is breaks. Breaks are expected to be finished in Carafe in November. The most difficult problem with breaks is modeling their behavior correctly. A relatively naive model would be to treat all nodes that cannot be initialized to a logic value as either one logic value or the other. Then all signal line breaks that separate the output of a gate from all inputs and signal line breaks that separate only one input from an output would be modeled as single stuck-at faults. Signal line breaks that separate the output of gate from multiple, but not all, inputs that it fans-out to would be treated as a multiple stuck-at fault. In this way Carafe can present a list of multiple stuck-at faults that are realizable by a single defect.

We are integrating each of these fault types into the Carafe-Nemesis ATPG package.

[6] This idea came to me from my SRC mentor, John Acken.

5 Physical Design for Testability

In general the design of the circuit (at the RTL, logic, or physical levels) can be changed to enhance testability in at least four ways.

1. **Design the circuit to have fewer faults.** If the circuit has fewer faults, fault simulation time, test pattern generation time, and the number of vectors to detect the faults are likely to be reduced.

2. **Make difficult to detect faults easier to detect.** Adding control and observation points to nodes that are difficult to control and observe respectively is a traditional DFT technique that makes difficult to test faults easier to test. Scan design makes faults that are inherently difficult to detect because of state in the circuit, less difficult to detect by removing that state.

3. **Make difficult-to-detect faults unlikely.** This technique is meaningful only if one is considering defect coverage instead of fault coverage. An example of this is shown in Figure 8. A break in one of the paths from the gate's output to ground is a difficult-to-detect fault for this circuit. This would cause the gate to have sequential behavior[Wad78]. The most likely cause of this happening is if a via (represented in Figure 8 as an "x") between the metal and diffusion regions is missing. One way to make this fault less likely is to provide a redundant connection between the two vias with diffusion. This can usually be run under the metal with no overhead or performance penalty. However it is still possible that the sequential fault will occur by an open in the diffusion path between the via and the transistor's source node, but this is much less likely than a missing via. Similar changes in layout have been discussed by Koeppe[Koe87] and Levitt and Abraham[LA90].

This list of DFT objectives is probably not exhaustive and some existing DFT techniques fall into multiple categories, but it serves to classify the P-DFT techniques that we outline later. We further divide potential improvements to the layout into those involving the physical design of the logic and those involving the interconnection between the logic.

P-DFT for logic is accomplished by laying out the logic blocks to enhance testability. Since the Carafe-Nemesis system extracts faults from and generates tests for circuits composed of standard cells, we will use examples from that technology. P-DFT for interconnect will be accomplished by placement, routing, design rules for routing, and selection between logic blocks with the same function. We do not assume a specific fault model in our analysis. Instead we base our evaluation on the testability of the circuit's realistic faults, those that are realizable due to a single defect[MFS84].

5.1 P-DFT for Logic

We first consider P-DFT of the logic blocks. It is more beneficial to modify the layout of a specific logic component to make it more testable for design environments where the same

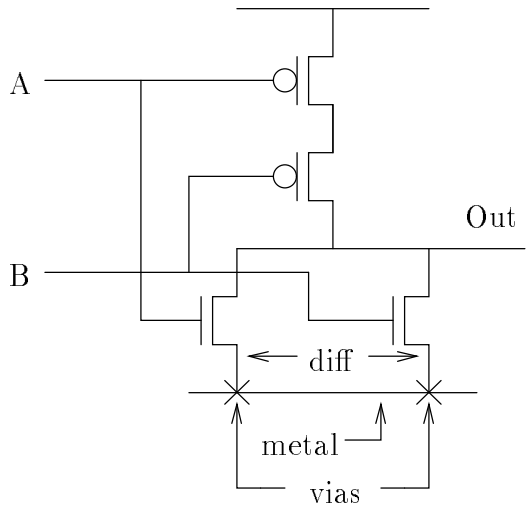


Figure 8: A CMOS NOR gate showing the metal-diff vias from the nMOS transistors to ground.

logic components are reusable, than it is in design environments where there is less logic reuse. Design environments with high logic reuse include standard cell, gate array, and some high-level synthesis environments.

The three objectives for P-DFT are to reduce the number of faults, make the difficult to detect faults easier to detect, and reduce the probability of difficult to detect faults. To work on improving any of these objectives requires that we have a measure for number of faults or difficulty to detect. Since a fault is a change in the behavior of the cell, the number of faults is the number of behaviors. If we consider only logic faults, then this would be the number of faulty truth tables. As an example consider the CMOSn standard cell whose logic diagram is shown in Figure 9. It has 43 bridge circuit-faults involving metal2, metal1, or poly, including shorts between these layers which result in 26 unique faulty functions. The maximum defect diameter used to find these bridge faults was 2.5 times the minimum line width of that layer. If there were fewer faulty functions the circuit would likely be easier to test.

Even greater savings in the number of faults are possible by considering delay faults since there are more potential delay faults than logic faults. A four-input complex gate has potentially $(2^4 * 2^4) - 2^4$ sequences of two vectors (excluding the same vector twice) to test all possible delay faults[7]. Reducing the number of potential delay faults could potentially reduce testing costs or defect levels considerably.

Equation 3 gives a simple metric for how difficult-to-detect a specific fault, f, in a one-output cell is. In the case of logical faults in combinational circuits "all possible tests" is 2^n where n is the number of inputs. "All possible tests" for delay faults may mean all possible

[7]If the circuit is glitch-free one need only consider the sequences in which the two vectors produce different output values. So a more reasonable estimate of the number of two pattern sequences would be $(2^4 * 2^4) - 2^4$ divided by 2.

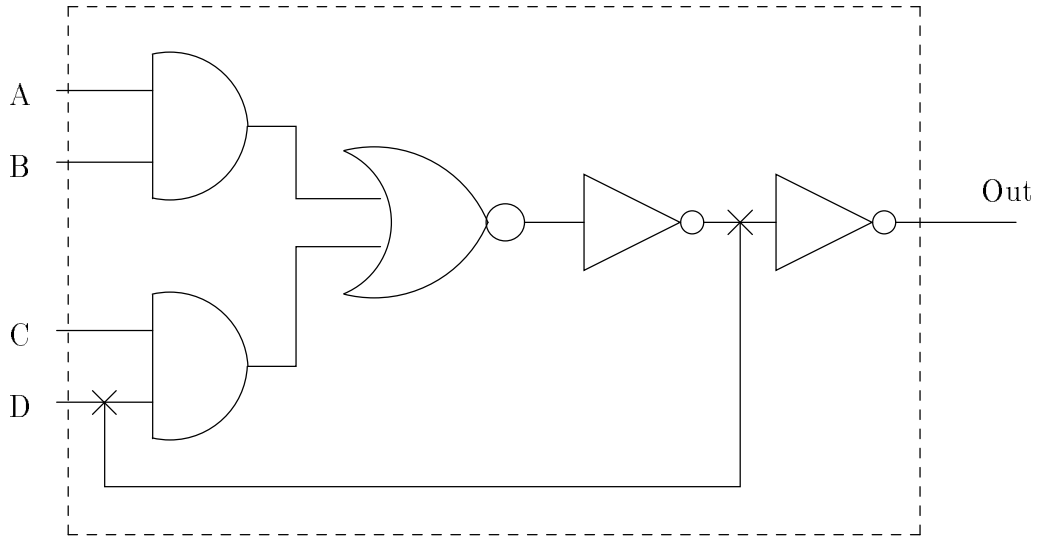


Figure 9: A bridge fault which can cause sequential behavior in a CMOS standard cell.

sequences of length two. Using Equation refdtd D_f for a input stuck-at 1 fault for a 3-input NAND gate is eight, whereas D_f for a input stuck-at 0 fault for a 3-input NAND gate is two.

$$D_f = \frac{\sum_{all\ possible\ tests}}{\sum_{all\ tests\ that\ detect\ f}} \tag{3}$$

One might rate fault types on a scale of difficulty to detect. For example, delay faults are generally more difficult to detect than logic faults and hence one might want to reduce the likelihood of a defect causing a delay fault even if it increases the likelihood of a logic fault by the same or greater amount. For the CMOSn standard cell that corresponds to the logic diagram of Figure 9 there are two bridge faults that cannot be detected as a logic fault and must be detected as a delay fault or I_{DDQ} fault. It may be possible to make these bridge faults to be expressed as a logic fault, thus making them easier to test. Lastly we can make the difficult to test faults less likely by routing the nodes away from each other so that they are less likely to happen, or by increasing the spacing of nodes that, if bridged, would be difficult to detect.

5.2 P-DFT for Placement and Routing

P-DFT can also be practiced for bridge faults involving the interconnect between cells. The design aspects that can be changed to affect these bridges are placement, routing, standard cell selection, and standard cell design. The first step for our research will be experimentation to determine what makes an interconnect bridge fault difficult to detect. Then we can add placement, routing, and logic choice rules to make the circuit more easily testable.

We first consider feedback bridge faults to show how changes in placement, routing, and

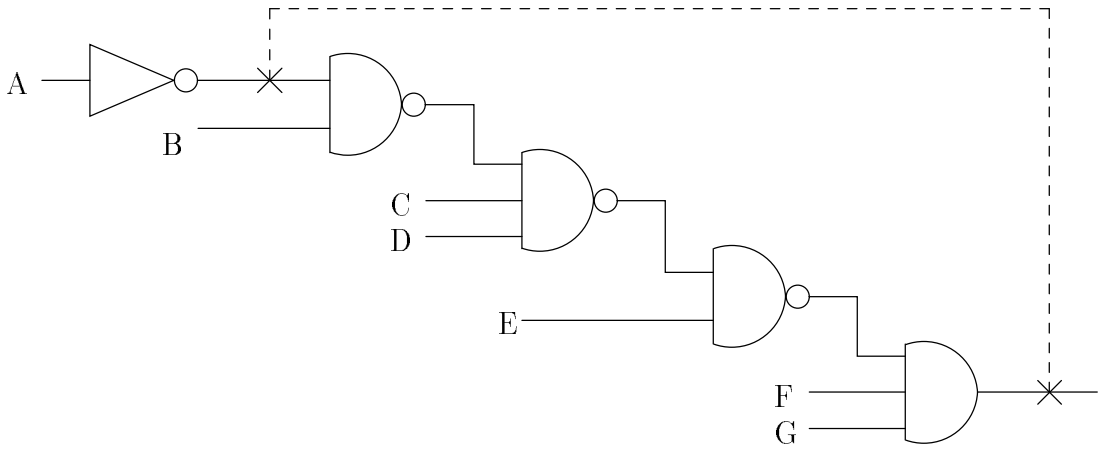


Figure 10: A potentially undetectable bridge fault.

cell selection may improve the testability of these circuits. Consider the subcircuit, which is assumed to be embedded in a much larger circuit, in Figure 10. If the drive of the AND gate is greater than the drive of the inverter, and there is no fanout in the path between these two gates then there is no test that guarantees the bridge fault shown will be detected. In this case the logic discrepancy can only occur on the output line of the inverter and the only propagation path available for the discrepancy is through the AND gate. However that will change the value of the AND's output which will propagate back to the inverter output, and back to the AND gate. This causes a logic value oscillating between 0 and 1 on the AND gate.

In the CMOSn library the AND gate is implemented as a NAND gate followed by a inverter. The inverter embedded in the AND gate is stronger than the standard inverter so the faulty circuit shown above implemented in this standard cell library would be considered untestable. We now consider the four ways that we can make this difficult to test fault less likely to occur.

1. Placement: You can make feedback bridge faults less likely by adding placement rules that place gates at the same depth in the circuit close together. Then the signal lines from gates towards the input of the circuit are less likely to be adjacent and hence less likely to bridge.

2. Routing: The routing of the circuit can be constrained to prevent potential feedback bridges by not routing them adjacent to one other. This is probably too restrictive so it can be relaxed to not allowing potential feedback bridges between nodes in which both (1) the node closer to the output has stronger drive and (2) there is no fanout in the nodes between the two potentially bridged nodes. This would prevent this undetectable fault from occurring.

3. Logic Selection: If the inverter has more drive than the AND gate then not only would there be no oscillation but the feedback bridge fault shown in Figure 10 is likely

to be fairly easily detectable. This is because the inverter, being nearer the inputs, is probably easy to control, and the AND gate, being nearer the output, is probably easier to observe. Physical design tools can be modified to select a cell with the appropriate drive strength based on the proximity of a difficult to detect fault.

We have presented three ways to change placement, routing, and logic selection to show how to make a specific fault type more easily testable. Other types of difficult to detect faults may be eliminated in similar ways. The first step is to use the Carafe-Nemesis system to determine which faults are difficult to generate tests for and see what set of characteristics they have in common. Then we can devise a set of possible P-DFT techniques that may be effective. Lastly we will implement these techniques and see how effective they are and how they affect the delay and cost of the circuit.

6 Summary

This report has presented the current state in fault modeling with emphasis placed on bridging faults. We then showed how the Carafe-Nemesis fault extraction and test pattern generation software will be used in future testing research.

References

[Ack83] J.M. Acken. Testing for bridging faults (shorts) in CMOS circuits. *Proceedings of Design Automation Conference*, pages 717–718, 1983.

[Ack88] John M. Acken. *Deriving Accurate Fault Models*. PhD thesis, Stanford University, Department of Electrical Engineering, September 1988.

[AM85] M. Abramovici and P.R. Menon. A practical approach to fault simulation and test generation for bridging faults. *IEEE Transactions on Computers*, C-34(7):658–663, July 1985.

[BF85] F. Brglez and H. Fujiwara. A neutral netlist of 10 combinational benchmark circuits and a target translator in fortran. In *Proceedings of the IEEE International Symposium on Circuits and Systems*, 1985.

[FL91a] F. Joel Ferguson and Tracy Larrabee. Test pattern generation for current testable faults in static CMOS circuits. In *Proceedings of the 1991 VLSI Test Symposium*, pages 297–302. IEEE, 1991.

[FL91b] F. Joel Ferguson and Tracy Larrabee. Test pattern generation for realistic bridge faults in CMOS ICs. In *Proceedings of International Test Conference*, pages 492–499. IEEE, 1991.

[FS83] H. Fujiwara and T. Shimono. On the acceleration of test-generation algorithms. *IEEE Transactions on Computers*, C-32(12):1137–1144, December 1983.

[Goe81] P. Goel. An implicit enumeration algorithm to generate tests for combinational logic circuits. *IEEE Transactions on Computers*, C-30:215–222, March 1981.

[Hei88] Dennis V. Heinbuch. *CMOS3 Cell Library*. Addison-Wesley Publishing Company, 1988.

[HS86] C.F. Hawkins and J.M. Soden. Reliability and electrical properties of gate oxide shorts in CMOS ICs. In *Proceedings of International Test Conference*, pages 443–451. IEEE, 1986.

[Jee91] Alvin Jee. Carafe: An inductive fault analysis tool for CMOS VLSI circuits. Technical Report UCSC-CRL-91-24, University of California at Santa Cruz, Computer Engineering Department, February 1991.

[Koe87] Siegmar Koeppe. Optimal layout to avoid CMOS stuck-open faults. In *Proceedings of Design Automation Conference*, pages 829–835. IEEE, 1987.

[LA90] Marc E. Levitt and Jacob A. Abraham. Physical design of testable VLSI: Techniques and experiments. *IEEE Journal of Solid-State Circuits*, 25(2):474–481, April 1990.

[Lar92] Tracy Larrabee. Test pattern generation using boolean satisfiability. *IEEE Transactions on Computer-Aided Design*, pages 4–15, January 1992.

[McC85] E.J. McCluskey. Built-in self-test techniques. *IEEE Design and Test of Computers*, pages 21–28, April 1985.

[Mei74] K.C.Y. Mei. Bridging and stuck-at faults. *IEEE Transactions on Computers*, C-23(7):720–727, July 1974.

[MFS84] W. Maly, F.J. Ferguson, and J. P. Shen. Systematic characterization of physical defects for fault analysis of MOS IC cells. In *Proceedings of International Test Conference*, pages 390–399. IEEE, 1984.

[MNN88] W. Maly, P.K. Nag, and P. Nigh. Testing oriented analysis of CMOS ICs with opens. In *Proceedings of International Conference on Computer-Aided Design*, pages 344–347. IEEE, 1988.

[MTCC87] W. Maly, M.E. Thomas, J.D. Chinn, and D.M. Campbell. Double-bridge test structure for the evaluation of type, size and density of spot defects. Technical Report CMUCAD-87-2, Carnegie Mellon University, SRC-CMU Center for Computer-Aided Design, Dept. of ECE, February 1987.

[PR88] A.K. Pramanick and S.M. Reddy. On the detection of delay faults. In *Proceedings of International Test Conference*, pages 845–856. IEEE, 1988.

[RMSC+91] R. Rodríguez-Montanés, J.A. Segura, V.H. Champac, J. Figueras, and J.A. Rubio. Current vs. logic testing of gate oxide short, floating gate, and bridging failures in CMOS. In *Proceedings of International Test Conference*, pages 510–519. IEEE, 1991.

[Rot66] J. P. Roth. Diagnosis of automata failures: A calculus and a method. *IBM Journal of Research and Development*, 10:278–291, 1966.

[RRK83] S.M. Reddy, M.K. Reddy, and J.G. Kuhl. On testable design for CMOS logic circuits. In *Proceedings of International Test Conference*, pages 435–445. IEEE, 1983.

[STS88] M.H. Schulz, E. Trischler, and T.M. Sarfert. SOCRATES: a highly efficient ATPG system. *IEEE Transactions on Computer-Aided Design*, CAD-7(1):126–137, January 1988.

[Syr87] Marek Syrzycki. Modelling of spot defects in MOS transistors. In *Proceedings of International Test Conference*, pages 148–157. IEEE, September 1987.

[TBG+83] C. Timoc, M. Buehler, T. Griswold, C. Pina, F. Stott, and L. Hess. Logical models of physical failures. In *Proceedings of International Test Conference*, pages 546–553. IEEE, October 1983.

[TLPM85] M.E. Turner, D.G. Leet, R.J. Prilik, and D.J. McLean. Testing CMOS VLSI: Concepts, and experimental results. In *Proceedings of International Test Conference*, pages 322–328. IEEE, 1985.

[Wad78] R.L. Wadsack. Fault modeling and logic simulation of CMOS and MOS integrated circuits. *Bell System Technical Journal*, 57(5):1449–1474, May-June 1978.

[WNS87] B.W. Woodhall, B.D. Newman, and A.G. Sammuli. Empirical results on undetected CMOS stuck-open failures. In *Proceedings of International Test Conference*, pages 166–170. IEEE, 1987.