Modeling and Simulation of Real Defects Using Fuzzy Logic

Amir Attarha
Dept. of EE
The Univ. of Texas at Dallas
Richardson, TX 75083
attarha@utdallas.edu

Mehrdad Nourani
Dept. of EE
The Univ. of Texas at Dallas
Richardson, TX 75083
nourani@utdallas.edu

Caro Lucas
Dept. of ECE
The Univ. of Tehran
Tehran 14399, IRAN
lucas@karun.ipm.ir

Abstract

Real defects (e.g. stuck-at or bridging faults) in the VLSI circuits cause intermediate voltages and can not be modeled as ideal shorts. In this paper we first show that the traditional zero-resistance model is not sufficient. Then, we present a resistive fault model for real defects and use fuzzy logic techniques for fault simulation and test pattern generation at the gate-level. Our method produces more realistic fault coverage compared to the conventional methods. The experimental results include the fault coverage and test pattern statistics for the ISCAS85 benchmarks.

1. INTRODUCTION

CMOS fabrication of digital integrated circuits includes defects that can not be represented using conventional idealistic stuck-at or bridging fault models. Unfortunately, such defects represent a significant fraction of faults in complex digital circuits [1] [2]. As transistor size shrinks, such resistive defects influence the fault detection even more [3] and thus it is vital to investigate their presence, effects, and detectability.

A fault occurs when two nodes are unintentionally connected together. We call faults (e.g. stuck-at or bridge) with zero resistance *ideal* faults. In reality, parasitic resistance (R), capacitance (C), and inductance (L) are always associated with the defects in the VLSI chips [4] [5]. The resistance value (specific or a statistical range), which is the most noticeable one, highly depends on the logic style, technology and the fabrication process. The faults with their associated resistances are called *real* faults in this paper.

Real stuck-at or bridging faults produce resistive paths between power supply and ground leading to intermediate voltages in the circuit nodes. The actual voltage values depend on the resistances of the networks that connect the signal to the power supply and ground. The interpretation and propagation of the intermediate voltages depend on many factors including the threshold voltages of the driver and driven gates, the nonlinear behavior of transistors and even asymmetry of the logic gates.

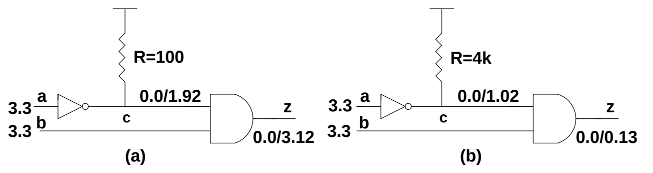


Figure 1: Voltage values for stuck-at-1 at point *c*

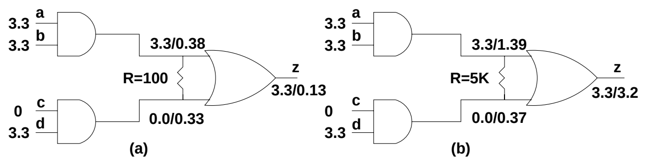


Figure 2: Voltage values for a bridging fault

1.1 Motivating Examples

Accurate modeling of real faults in the VLSI chips requires considering the parasitic R, C, and L associated with the faults. In this paper we consider only the resistance value which is the most influential one among the three in terms of affecting the node voltages and thus the fault detection.

• Example 1: Stuck-at Fault
Figure 1 shows SPICE simulation [6] results for a real stuck-at fault in a small circuit using a cell library with $Vdd = 3.3$ volt. Pattern $ab = 11$ can be applied to detect ideal s-a-1 at point c. However, real s-a-1 at point c may or may not be detected with this pattern. Figure 1(a) shows that if $R \approx 100$ Ohm, the fault-free and faulty voltage values on z correspond to logic 0 and 1, respectively; and thus the fault will be detected. However, if $R \approx 4$ K Ohm it won't be detected as shown in Figure 1(b). In actual testing, a test equipment that uses the same pattern to test the circuit, depending on the value of R may or may not see it.

• Example 2: Bridging Fault
Figure 2 shows SPICE simulation [6] results for a real (resistive) bridging fault using the same cell library as in the first example. Pattern $abcd = 1101$ can detect resistive fault when $R \approx 100$ Ohm but will fail if $R \approx 5$ K Ohm as shown in Figure 2 (a) and (b), respectively.

• Example 3: The Effect of Gate Asymmetry
Figure 3 shows internal transistors of a CMOS AND gate (NAND followed by a NOT) and the result of SPICE simulation [6] for a

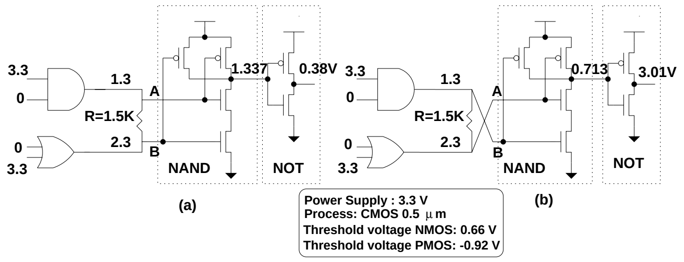


Figure 3: The effect of gate internal asymmetry

small circuit. In 3 (a), the inputs A and B are driven by a 2-input AND gate and a 2-input OR gate, respectively. The bridging fault R, between the outputs of the AND and the OR gates, produces two intermediate voltages at its two ends. Based on SPICE simulation these intermediate voltages force output of the NOT gate to 0.38 volt, that is logic ’0’. On the other hand, if the inputs of NAND gate are swapped, as shown in 3 (b), then, the output of circuit is 3.01 volt, that is logic ’1’. Because of the asymmetry of the NAND gate, with respect to its internal transistors, different orientations for a gate lead to different interpretations for the same input voltages after passing only two level of primitive gates.

These three examples clearly show that the conventional fault simulation is not sufficient and the fault simulation of real faults require accurate voltage analysis. Using transistor level simulators, such as SPICE, is not practical for large circuits. Moreover, these simulators often generate other information (e.g. timing behavior) which are not used in fault simulation. This motivated us to propose a fault simulator with high accuracy for voltage computation. Such simulator considers resistive faults and generates only the voltage levels for circuit nodes that are crucial in fault simulation process.

We acknowledge that some of the real defects (e.g. a very large resistive stuck-at fault) may not harm the functionality of a circuit. However, there are various reasons why detecting such faults is still important. They may create signal skew [7], cause excessive power consumption [8] or indicate problems to come in future, e.g. migration of metal to the surrounding areas over time and shortening the life and reliability of a chip [2].

1.2 Related Works

Most methods tried to improve the accuracy of their fault modeling by using an approximation method at the gate level such as a voting model [9] [10]. Although these methods are very fast, their accuracies are not acceptable, because they only analyze the bridge output voltages without carefully considering how the faults propagate [11]. The performance of the switch level tools such as SWITEST [12] or the analog simulators like SPICE [6] are not always acceptable, especially if large VLSI circuits have to be analyzed [13]. A different family of methods using mixed level or multi-level simulation techniques have been proposed in [14] [15]. These methods switch from logic simulation to transistor level simulation whenever an unconventional fault is encountered. These methods are relatively accurate but for large circuits they do not run efficiently as discussed in [11].

The above shortcomings motivated us to employ the fuzzy logic theory to model and simulate real faults. Fuzzy logic with the ability to model any nonlinear system provides a powerful tool to deal with uncertainties and complexities inherent in a practical problem [16]. It further enables us to utilize human experience in form of ad-hoc rules in the design or analysis process which eliminates the need to identify complicated mathematical representations. Such rules can be usually optimized using empirical data [21].

1.3 Contribution and Paper Organization

Our fault model assigns a non-zero resistance, randomly selected in a predefined range $[R_{min}, R_{max}]$, to the stuck-at and bridging faults in general. Ideal (zero resistance) faults will be a special case in our modeling, where $R_{min} = R_{max} = 0$. We first model logic components as fuzzy blocks by extracting the information of non-embedded logic gates from results reported by SPICE. This feature makes our method fully adaptable by new libraries and technologies. Then, we use fuzzy logic to develop an accurate (for voltage calculation) fault simulator to analyze real faults in digital circuits at the gate level for the purpose of fault grading. Our fault simulator will report a true fault coverage by considering the real faults and thus improves the yield factor when chips are actually tested by the test equipments.

Feltham and Maly [17] demonstrated that many defects in modern CMOS technologies cause changes in the circuit description that result in electrical shorts and implied that many failures can be modeled by bridging faults. What differentiates our model from [17] or similar approach such as [11] and [18] is in: a) considering resistive stuck-at faults in addition to bridging faults, b) using an analytical fuzzy-based analysis instead of lookup tables for accurate voltage computation, and c) generating test patterns using the resistance value of faults.

The rest of the paper is organized as follows. Our fault model is explained in Section 2. Section 3 describes how an individual (non-embedded) logic gate is modeled as a fuzzy block. Section 4 explains the fault simulation algorithm. In Section 5, we comment on how our simulator can be used to generate test patterns for real faults. The experimental results are discussed in Section 6. Finally, the concluding remarks are in Section 7.

2. FAULT MODEL

Our fault model assumes a single resistive bridging fault (R_{bridge}) exists in a circuit as shown in Figure 4. A stuck-at fault is a bridging fault, occurred between specific node and Vdd or Gnd through a resistance. This resistance is zero for ideal and takes a non-zero value for real faults.

In CMOS, each node is driven by a resistive path from the power supply or ground. To analyze the behavior of bridging faults in the circuit, the voltages of the two nodes of the bridge must be determined first. These voltages, then, should be propagated accurately across the circuit. These two issues are addressed next.

2.1 Voltage Calculation

As presented by [20], [2], and [1] the resistance of the realistic defects may vary between several Ohms to several Kilo Ohms depending strongly on layout details, technologies and fabrication process. For example, authors in [2] and [1] reported defect resistance between 200 Ohm to 30K Ohm for register cell structures and 0 Ohm to 5K Ohm for bridge defects, respectively. Empirically, however, the resistance of 0.5K to 2K provided satisfactory results in test of digital circuits[20]. In our work we allow user define a range (e.g. $[R_{min}, R_{max}]$) and the simulator will select a random resistance, i.e. R_{bridge} in this range.

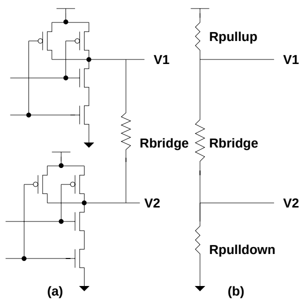


Figure 4: A resistive path for a bridging fault.

Connection of two nodes via a bridging defect makes a resistive path between power supply and ground as shown in Figure 4. Typical pull up and pull down resistors are often given in data sheets of cell libraries [4] and therefore two nodes of the fault can be analyzed by voltage division:

$$\begin{cases} V_1 = \frac{R_{bridge}+R_{pulldown}}{R_{pullup}+R_{bridge}+R_{pulldown}} \cdot V_{dd} \\ V_2 = \frac{R_{pulldown}}{R_{pullup}+R_{bridge}+R_{pulldown}} \cdot V_{dd} \end{cases} \quad (1)$$

2.2 Voltage Propagation

The second factor that determines the accuracy of a real fault model is the propagation of the voltages at the nodes on two sides of the fault (i.e. generalized as a bridge). This step traces the voltages accurately through the circuit using input voltages, thresholds of logic gates and their asymmetry. We have developed a fuzzy fault simulator, with SPICE precision for voltage computation, to carry out this step. The fuzzy fault simulator will be discussed extensively in the next section. Here, we just point out that to achieve a reasonable run time we can take advantage of logic voltage margins inherent in digital gates. Specifically, in CMOS technology, any voltage in the range of $[0, 0.3Vdd]$ is recognized as LOW, in the range of $[0.3Vdd, 0.7Vdd]$ is recognized as HIGH and between these two is considered MED (abnormal) [22].

Note that these margins at $0.3Vdd$ and $0.7Vdd$ are not crisp and may differ for each technology, library or even logic gate. Also, these margins may change based on the factors that can influence threshold voltage such as temperature. This vagueness is an important indication why fuzzy system can be used to approximate logic gates behavior [16].

3. MODELING LOGIC COMPONENTS

For each logic gate in the target library a fuzzy block is designed, which approximates the desired behavior in response to different level of voltages. This approximation should be accurate enough to reflect the behavior of real resistive faults and their effects when propagated through the circuit. We construct such a database for all logic gates used in the circuit through the following three steps that are quite standard in developing a fuzzy system [16].

3.1 Fuzzy System Development

■ **Step 1: Find the Input-Output Behavior**

We simulate all logic components in the library by SPICE with desired accuracy (e.g. 0.01 volt). Note that SPICE simulation is done once and the input-output data obtained will be used to construct the fuzzy block corresponding to each logic component. To build a database for our fuzzy blocks the whole range of input voltages (the universe of discourse in fuzzy terminology) have to be covered.

■ **Step 2: Initial Structure and Parameters**

There are basically two approaches to construct fuzzy systems from input-output pairs of data [16]. In the first approach, fuzzy IF-THEN rules are first generated from input-output pairs, and the fuzzy system is constructed from these rules according to certain choices of fuzzy inference engine, fuzzifier, and defuzzifier. In the second approach, the structure of fuzzy system is pre-designed with some free parameters. Then, these free parameters are optimized according to the input-output pairs. In this work, we adopt the second approach.

We select the first order Sugeno model [16] as the basic structure of the fuzzy blocks. The output of the Sugeno model is a linear function of input variables, therefore, the Sugeno fuzzy system can be viewed as a somewhat piece-wise linear function, where the change from one piece to the other is smooth rather than abrupt [21]. Based on this model the fuzzy system $f(X)$, $X = [x_1, x_2, ..., x_n]$ is of the following form:

$$f(X) = \frac{\sum_{l=1}^{M} \left[Z^l(X) . W^l(X) \right]}{\sum_{l=1}^{M} W^l(X)} \quad (2)$$

where M is the number of IF-THEN rules; and $Z^l(X)$ (output of the lth rule) and $W^l(X)$ (excitation weight of the lth rule) are defined as follows:

$$\begin{cases} Z^l(X) = \sum_{i=1}^{n} k_i^l x_i + c_i^l \\ W^l(X) = \prod_{i=1}^{n} exp\left(- \frac{x_i - \mu_i^l}{\sigma_i^l} \right)^2 \end{cases}$$

where the superscript l refers to the lth rule, n is the number of inputs, μ_i and σ_i denote average and standard deviation of the membership functions, respectively. Finally, k_i and c_i represent a factor and a constant associated with the polynomial defined in the first-order Sugeno model, respectively. Note that μ_i, σ_i , and k_i are free parameters, which need to be optimized (tuned) to complete the fuzzy system.

Each rule comprises IF-THEN condition and has the following form:

$$Rule^{(l)}: \ IF \ \ ((x_1 \ is \ A_1^l) \ \& \cdots \& \ (x_n \ is \ A_n^l)) \ \ THEN$$
$$Z^l(X) = \sum_{i=1}^{n} k_i^l x_i + c_i^l$$

where A_i^l's are fuzzy sets in the antecedent, and $Z^l(X)$ is a crisp first-order polynomial function in the consequent [16]. To initialize the system, we must first determine initial rules and initial values of μ_i and σ_i . These initial parameters are determined empirically using linear approximation. We partition the input universe of discourse to three spaces, "LOW", "MED" and "HIGH", where LOW, MED and HIGH refer to three Gaussian membership functions with initial $[\sigma_i, \mu_i]$ values of $LOW = [0.35, 0.60]$, $MED = [0.8, 1.8]$, and $HIGH = [0.75, 2.8]$, respectively.

Figure 5 shows the SPICE output and fuzzy output for a NOT gate after these initial settings, where the output behaves imprecisely in some ranges. To improve our model, an optimization technique (see Step 3) is used to determine the free parameters i.e. μ_i, σ_i, and k_i, more precisely.

■ **Step 3: Optimize the Free Parameters**

We use nonlinear least square method to optimize the free parameters. This method plays a prominent role in the framework of

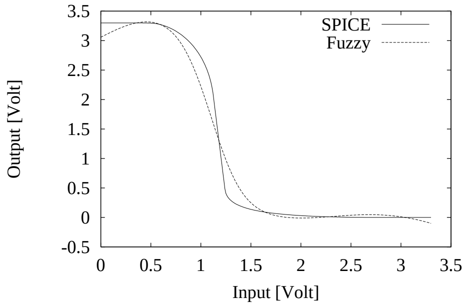
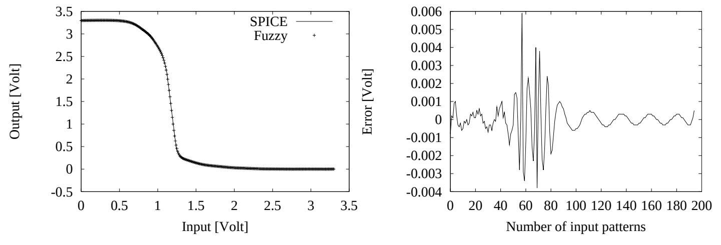
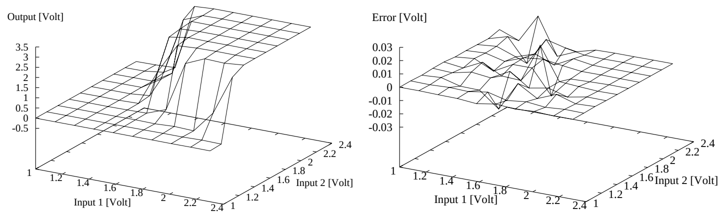


Figure 5: SPICE vs. fuzzy for NOT gate after initial setting



(a) Output (b) Absolute Error

Figure 6: SPICE vs. fuzzy for NOT gate



(a) Output (b) Absolute Error

Figure 7: SPICE vs. fuzzy for 2-input AND gate

soft computing, and the sum of squared errors is frequently chosen as the objective function to be minimized [21]. This method is commonly used in data fitting and regression [21] and is briefly described below.

Consider an n-input, single output model with m free parameters: $y = f(X,\Theta)$ where y is the model's scalar output, $X = [x_1,...,x_n]$ is the input vector of size n, and $\Theta = [\theta_1,\theta_2,...,\theta_m]^T$ is the parameter vector. In designing the fuzzy system, we focus on minimizing the error function $E(\Theta)$, that is the sum of squared error. Finding a parameter vector Θ^* that minimizes $E(\Theta)$ is of primary concern:

$$E(\Theta) = \sum_{p=1}^{m}\left(t_p - y_p\right)^2 = \sum_{p=1}^{m}\left(t_p - f(X_p,\Theta)\right)^2 = \sum_{p=1}^{m} r_p(\Theta)^2 = \mathbf{r}^{\mathbf{T}}(\Theta)\mathbf{r}(\Theta)$$

where t_p and y_p are the desired output (e.g. SPICE results) and the approximation result (e.g. by the fuzzy simulator) for the same input X_p, respectively; and $\mathbf{r}(\Theta) = [r_1(\theta),...,r_m(\theta)]$.

Since $E(\Theta)$ is nonlinear, to minimize it we use the iterative descend method [16], in which the next point Θ_{next} is determined by a step down from the current point Θ_{now} in a direction vector $g(\Theta)$:

$$\Theta_{next} = \Theta_{now} + \eta(\Theta)g(\Theta)$$

$g(\Theta)$ is the straight downhill direction and $\eta(\Theta)$ is a positive step size regulating to what extent to proceed in that direction. In this work, we utilized the nonlinear Levenberg-Marquart method [21] to determine $g(\Theta)$ and $\eta(\Theta)$.

3.2 Fuzzy Logic Versus SPICE

In our formulation, $\Theta = [\mu_i,\sigma_i,k_i]$ is the parameter vector. After optimizing Θ the fuzzy model for NOT gate shows very high accuracy compared to the SPICE, as shown in Figure 6. The cell is selected from a library using $0.5\mu m$ technology and $Vdd = 3.3$ Volt.

Figure 6(a) and (b) show the outputs of SPICE versus fuzzy simulation and absolute error for each of the 200 input patterns (voltages) in the range of $[0, 3.3V]$. Figure 7 (a) and (b) show the result of fuzzy simulator and absolute error for a two input AND gate, respectively. The behavior of AND gate is approximated accurately, such that the maximum error for all input combinations is less than 0.03 Volt as shown in Figure 7 (b) and mean square error is less than 0.04. Note that this figure only shows the result of 350 patterns to generate the input voltages between 1V to 2.5V (abnormal region) which is a critical part of analysis in real fault simulation.
In modeling a logic circuit using fuzzy logic, the small error observed in the intermediate levels (e.g. 0.03 V in above example) does not accumulate in the process and thus could be ignored, because the intermediate voltages reach to Vdd or Gnd after few levels anyway.

The whole practical point about our fuzzy simulator is that a real (stuck-at or bridging) fault that cause abnormal level of voltages in the circuit can be traced carefully toward the output(s). This is a fundamental necessity for real fault detection. It is worth mentioning that once we model all logic gates as fuzzy blocks the fuzzy simulator is quite fast.

4. FUZZY FAULT SIMULATION

Having all the logic gates as fuzzy blocks, we can carry out circuit simulation with high accuracy in terms of computing voltages in various nodes. Such pseudo analog simulation working at the gate level is the most important feature in our approach as it presents high precision (even comparable to the SPICE) to catch real faults.

Our fuzzy fault simulator operates at the gate level and at present is limited to the combinational circuits. The simulator works similar to a traditional single fault propagation scheme. First, the fault-free circuit is simulated for an input vector. Then, the fault is inserted and the faulty circuit is analyzed and the result is compared to the fault-free value. The computation of faulty values starts at the site of the fault and continues until all faulty values become identical to the fault-free values or the fault observed at one of the primary outputs [5].

Figure 8 details the fuzzy engine in fault simulation process. The resistance of the real fault (stuck-at or bridge) is selected randomly from the range predefined by user. In practice, such range is determined by empirical and statistical data and varies for different styles, technologies and even fabrication plants [23][2]. The voltage of two sides of the resistive fault is calculated using Equation 1. If the voltage is within normal range (i.e. $[0, 0.3Vdd]$ for

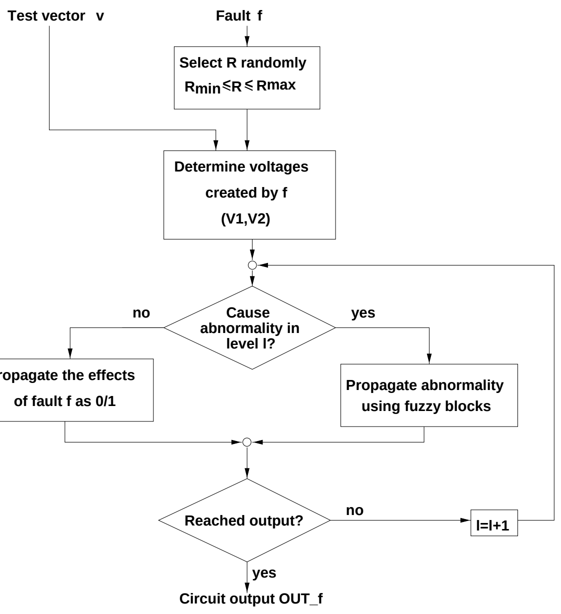

Test vector v
Fault f
Select R randomly
Rmin≤R≤Rmax
Determine voltages created by f (V1,V2)
no
Cause abnormality in level l?
yes
Propagate the effects of fault f as 0/1
Propagate abnormality using fuzzy blocks
Reached output?
no
l=l+1
yes
Circuit output OUT_f

Figure 8: The Fuzzy Engine in fault simulation

logic "0" and $[0.7Vdd, Vdd]$ for logic "1"), the simulator just propagates the effect of those faults through the circuits as logic "0" or "1". However, if the voltages are within the abnormal range, i.e. $[0.3Vdd, 0.7Vdd]$, there are two cases:

- Case 1: The abnormal voltage is changed to normal by the "controlling input" (e.g. 0 for AND and 1 for OR). Obviously, such controlling input masks the effect of abnormal voltage and thus we continue normal functional simulation as if no abnormality happened.

- Case 2: The abnormal voltage is not masked by other inputs. We, therefore, employ the fuzzy block behavior for those gates which see abnormality and trace the effect carefully through that level (l). This process is repeated until we reach the output. This mechanism allows us to optimize the running time of the fuzzy simulator. The fuzzy block evaluation is invoked only when an abnormality is identified. This is especially important since depending on the abnormal voltages, after a few levels (e.g. 2, 3, or 4) the voltages enter normal range and we can continue simulation with higher speed by not entering the fuzzy computations.

5. TEST PATTERN GENERATION

Although test pattern generation for resistive faults is not the focus of this paper, we would like to briefly comment on the key question of how test patterns can be generated for real faults. We believe our fuzzy engine can be also used to generate patterns, which overall have better chance to detect real faults and so the fault coverage can be enhanced. To do this, our basic strategy is to first use a conventional test pattern generation algorithm (e.g. PODEM [24]) to generate test vectors and then ask the fuzzy simulator to evaluate different choices for their potential in detecting real faults in a given resistance range. By doing so, our pattern selection mechanism is geared toward detecting the real faults. However, we may select more patterns to cover a wider range of resistances associated with real faults.

When we deal with the real stuck-at faults, we just ask PODEM to

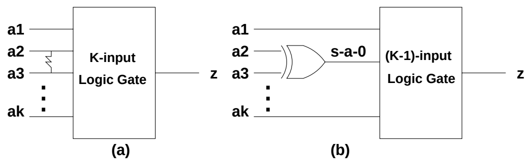


Figure 9: XOR replacement for bridging faults

Circuits	*Number of gates*	*Number of stuck-at faults*	*Number of bridging faults*
C432	162	602	311
C880	449	1435	325
C1355	562	1934	588
C1908	781	2207	634
C2670	1078	3397	1045
C3540	1773	4819	1655
C5315	2338	7060	2452
C7552	3499	9861	2787

Table 1: Benchmark circuits used for experiments

generate the patterns. For the bridging faults We limit ourselves to the faults between two inputs of a gate. For such faults, an appropriate test pattern forces two sides of bridge to take opposite logic values (0 and 1), because it activates the fault in a way that two voltages take different values and we can later use fuzzy simulator to propagate it. Figure 9 shows that by a simple trick we can use PODEM to generate the test pattern. To do this, we insert a 2-input XOR gate in the location of the bridging fault and ask PODEM to generate a pattern to detect stuck-at-0 at the output of XOR. To activate the fault, PODEM forces output of XOR to be one which means it selects pattern(s) to create 01 or 10 in the XOR inputs. That is exactly what fuzzy simulator wants to see (i.e. two different voltages on two sides of the fault) to proceed. Note carefully that appropriate (non-controlling) values for propagation for other inputs before and after XOR replacement remain unchanged.

After finding some candidates, the fuzzy simulator verifies if the test vector(s) generated by PODEM actually detects the real fault in the original circuit. Currently, our procedure takes a conservative strategy and selects more patterns than PODEM to catch wide range of resistive faults.

6. EXPERIMENTAL RESULTS

We implemented our method in C running on SPARC ULTRA 1 workstations. The running time of the fuzzy simulator for the ISCAS85 benchmarks varies from 0.6 second to 39.9 seconds. Table 1 summarizes the specification of the benchmarks. The bridging faults are assumed only among the inputs of gates. Table 2 and 3 show the fault simulation results by conventional (third column shown as $R = 0$) and fuzzy (fourth column shown as $R \neq 0$) simulators, for stuck-at and bridging faults, respectively. Number of patterns (N_{pat}) given in the second column of these two tables are based on PODEM. The fuzzy simulator reports lower fault coverage ($FC\%$) because it considers the real faults in the range of ($R \in [0.5K, 2K]$) and predicts accurately the situation that test equipment will experience in the actual testing. A conventional fault simulator considers only ideal faults ($R = 0$) and its report on fault coverage is simply too optimistic.
By using our fuzzy test pattern generation (FTPG) procedure detecting real faults will be improved with the cost of more time to

Circuits	N_{pat}	*FC%*	*FC%*	*FTPG*	
		$R = 0$	$R \neq 0$	N_{pat}	*FC%*
C432	77	99.4	78.3	108	85.7
C880	103	100	87.3	112	94.3
C1355	102	100	82.9	143	91.7
C1908	149	100	89.1	201	100
C2670	153	98.8	77.5	178	94.5
C3540	278	98.4	88.3	329	88.3
C5315	248	99.5	89.6	302	100
C7552	348	98.7	84.1	417	93.9

Table 2: Test results for stuck-at faults

Circuits	N_{pat}	*FC%*	*FC%*	*FTPG*	
		$R = 0$	$R \neq 0$	N_{pat}	*FC%*
C432	121	87.9	67.3	169	73.2
C880	149	99.3	75.1	287	87.9
C1355	217	95.3	65.5	293	71.9
C1908	193	96.4	78.3	281	89.5
C2670	183	88.6	74.9	401	83.1
C3540	292	93.3	73.7	483	81.2
C5315	312	98.9	79.9	378	87.2
C7552	373	91.4	72.7	545	79.8

Table 3: Test results for bridging fault

apply. This is reflected in the last two columns in Table 2 and 3. Our test pattern generation strategy uses 10 to 50 percent more patterns to be able to detect a wide range ($[0.5K - 2K]$) of resistive faults. Assuming that the statistical analysis of fabrication process allows us to narrow down this range to $[1K - 1.5K]$, we observed that the overhead for number of patterns is reduced to 4 to 18% while the fault coverage increases by 6% on the average for the bridging faults, compared to those reported in Table 3.

7. CONCLUSION

Detecting real defects in the VLSI circuits needs accurate analysis of the circuit by considering at least the resistance associated with those defects. We proposed a fuzzy based engine to accurately compute and propagate the voltage values through a gate level circuit for stuck-at and bridging faults. The fault coverage reported by our fuzzy simulator is realistic, often lower than the optimistic coverage reported by a conventional fault simulator, for the same set of test patterns. The realistic view of the fuzzy engine can be used to search for a more complete set of test patterns that have a better chance to detect real faults.

8. REFERENCES

[1] F. Hawkins, J. Soden, A. Righter and F. Ferguson, "Defect Classes – An Overdue Paradigm for CMOS IC Testing," *Proc. Int. Test Conf.*, pp. 413-425, Oct. 1994.

[2] R. Aitken, "Finding Defects With Fault Models," *Proc. Int. Test Conf.*, pp. 498-505, Oct. 1995.

[3] H. Vierhaus, W. Meyer and U. Glaser, "CMOS Bridges and Resistive Transistor Faults I_DDQ versus Delay Effects," *Proc. Int. Test Conf.*, pp. 83-91, Oct. 1993.

[4] N. Weste and K. Eshraghian, *Principles of CMOS VLSI Design*, Addison-Wesley, 1993.

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

[6] TI SPICE3 User's and reference manual, *1994 Texas Instrument Incorporation.*,

[7] S. Sparmann, D. Luzenburger, K. Cheng and S. Reddy, "Fast Identification of Robust Dependent Path Delay Faults," *Proc. of the 32nd Design Automation Conf.*, pp. 119-125, June 1995.

[8] M. Nourani, J. Carletta and C. Papachristou, "A Scheme for Integrated Controller-Datapath Fault Testing," *Proc. of the 34th Design Automation Conf.*, pp. 546-551, June 1997.

[9] M. Acken and S. Millman, "Fault Model Evolution for Diagnosis: Accuracy vs. Precision," *Proc. of IEEE Custom Integrated Circuits Conf.*, pp. 13.4.1-13.4.4, 1992.

[10] Chess and T. Larrbee, "Bridge Fault Simulation Strategies for CMOS Integrated Circuits," *Proc. of Design Automation Conf.*, pp. 1503-1507, 1993.

[11] Di and J. Jess, "An Efficient CMOS Bridging Fault Simulator: with Spice Accuracy," *IEEE Trans. on Computer Aided Design*, vol. 15, no. 9, pp. 1071-1080, Sept. 1996.

[12] J. Lee, C. Njinda and M. Breuer, "SWITEST: A Switch Level Test Generation System for CMOS Combinational Circuits," *Proc. of Design Automation Conf.*, pp. 26-29, June 1992.

[13] Mahlstedt and J. Alt, "Simulation of non-classical Faults on the Gate Level: The Fault Simulator COMSIM ," *Proc. of Int. Test Conf.*, pp. 883-892, 1993.

[14] Rearick and J. Patel, "Fast and Accurate CMOS Bridging Fault Simulation," *Proc. of Int. Test Conf.*, pp. 54-62, 1993.

[15] S. Greenstein and J. Patel, "E-PROOFS: A CMOS Bridging Fault Simulator," *Proc. of Int. Conf. Computer Aided Design*, 1992.

[16] X. Wang, *A Course in Fuzzy Systems and Control*, Prentice-Hall, 1997.

[17] D. Felthaam and W. Maly, "Physically Realistic Fault Models for Analog CMOS Neural Networks," *IEEE Journal of Solid-State Circuits*, vol. 26, pp. 1223-1229, Sept. 1991.

[18] V. Sar-Dessai and D. Walker, "Resistive Bridge Fault Modeling, Simulation and Test Generation," *Proc. Int. Test Conf.*, pp. 596-605, Oct. 1999.

[19] D. Lavo, B. Chess, T. Larrabee and F. Ferguson, "Diagnosing Realistic Bridging Faults with Single Stuck-at Information," *IEEE Trans. on Computer Aided Design*, vol. 17, no. 3, March 1998.

[20] Dalpasso, M. Favalli, P. Olivo and B. Ricco, "Fault Simulation of Parametric Bridging Faults in CMOS IC's," *IEEE Trans. on Computer Aided Design*, vol. 12 , no. 9, pp. 1403-1410, Sept. 1993.

[21] Jang, C. Sun, and E. Mizutani, *Neuro-Fuzzy and Soft Computing*, Prentice-Hall, 1997.

[22] J. Wakerly, *Digital Design Principles and Practices*, Prentice-Hall, 1990.

[23] A. Miller, "IDDQ Testing in Deep Submicron Integrated Circuits," *Proc. of Int. Test Conf.*, pp. 724-729, 1999.

[24] P. Goel and B. Rosales, "PODEM-X: An Automatic Test Generation System for VLSI Logic Structures," *Proc. of Design Automation Conf.*, pp. 260-268, June 1981.