# Application of Fuzzy Logic in Resistive Fault Modeling and Simulation

Mehrdad Nourani, *Member, IEEE*, Amir R. Attarha, and Caro Lucas

*Abstract*—**Real defects (e.g., resistive stuck at or bridging faults) in the very large-scale integration (VLSI) circuits cause intermediate voltages which cannot be modeled as ideal shorts. In this paper, we first show that the traditional zero-resistance model is not sufficient for fault simulation. 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 uses Takagi–Sugeno (TS) fuzzy system to accurately model digital VLSI circuits and produces much more realistic fault coverage compared to the conventional methods. The experimental results include the fault coverage and test-pattern statistics for the ISCAS85 benchmarks.**

*Index Terms*—**Bridging faults, fault simulation, fuzzy logic, Mamdani model, resistive faults, stuck at faults, Takagi–Sugeno (TS) model, test-pattern generation.**

## I. Introduction

CMOS fabrication of digital integrated circuits includes defects that cannot 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]; 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 very large-scale integration (VLSI) chips [3], [4]. 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.

Manuscript received September 18, 2000; revised October 31, 2001 and January 8, 2002.

M. Nourani and A. R. Attarha are with the Department of Electrical Engineering, The University of Texas at Dallas, Richardson, TX 75083-0688 USA (e-mail: nourani@utdallas.edu; attarha@utdallas.edu).

C. Lucas is with the Department of Electrical and Computer Engineering, University of Tehran, 14399 Tehran, Iran (e-mail: lucas@karun.ipm.ir).

Publisher Item Identifier 10.1109/TFUZZ.2002.800689.

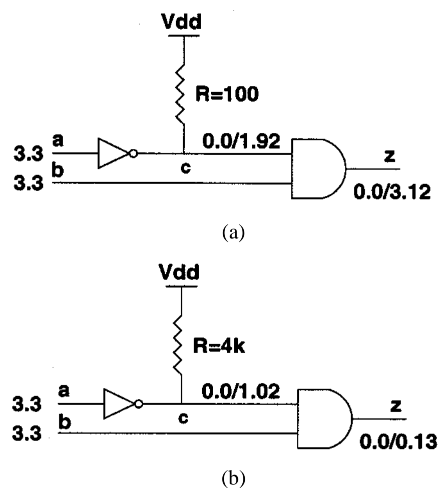



Fig. 1. Fault-free/faulty voltage values for stuck at-1 at point $c$. (a) $R = 100$ $\Omega$. (b) $R = 4$ K $\Omega$.

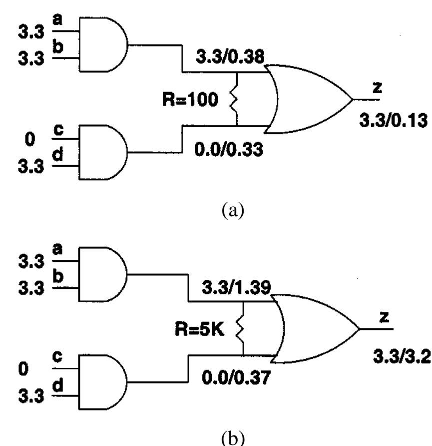


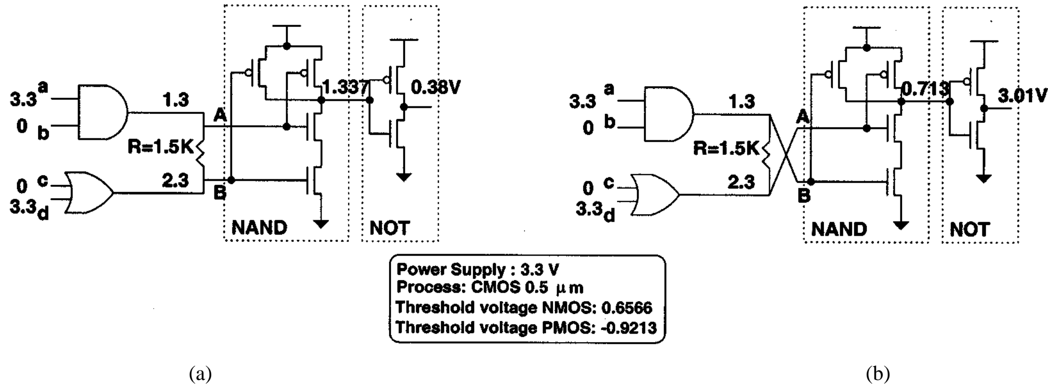

Fig. 2. Fault-free/faulty voltage values for a bridging fault. (a) $R = 100\ \Omega$. (b) $R = 5$ K $\Omega$.

### A. Motivating Examples

Accurate modeling of real faults in the VLSI chips requires considering the parasitic $R$, $C$, and $L$ associated with the faults [6]. 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.

*1) Example 1: Stuck At Fault:* Fig. 1 shows SPICE simulation [5] results for a real (resistive) stuck at fault in a small circuit using a cell library with $V_{dd} = 3.3$ V. Pattern $ab = 11$ can be applied to detect ideal s-a-1 at point $c$. However, depending on the resistance value, the real s-a-1 at point $c$ may or may not be detected with this pattern. Fig. 1(a) shows that if $R = 100\,\Omega$, the

1063-6706/02$17.00 © 2002 IEEE




(a) (b)

Fig. 3. The effect of gate internal asymmetry. (a) Orientation 1 for NAND. (b) Orientation 2 for NAND.

fault-free and faulty voltage values on $z$ are logically opposite and thus the fault is detected. However, if $R = 4$ K $\Omega$ it cannot be detected as shown in Fig. 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 catch it.

*2) Example 2: Bridging Fault:* The bridging fault is a model to represent fabrication defects that erroneously connect two points, none of which is necessarily $V_{dd}$ or $Gnd$ [7], [8]. Fig. 2 shows SPICE simulation [5] results for a real (resistive) bridging fault using the same cell library. Similar to the previous example, pattern $abcd = 1101$ can detect resistive fault when $R = 100\ \Omega$ but will fail if $R = 5$ K $\Omega$ as shown in Fig. 2(a) and (b), respectively.

*3) Example 3: The Effect of Gate Internal Asymmetry:* Fig. 3 shows internal transistors of an AND gate (NAND followed by a NOT) and the results of SPICE simulation [5] for a small circuit. In Fig. 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, $R$, which occurs 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 V, that is logic "0." On the other hand, if the inputs of the NAND gate are swapped, as shown in Fig. 3(b), then, the output of circuit becomes 3.01 V, that is logic "1." Because of the asymmetry of the NAND gate, with respect to its internal transistors, different orientations for a gate leads 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 requires accurate voltage analysis. Using a transistor level simulator 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 motivates us to propose a fault simulator with high accuracy for voltage computation. Our simulator considers the resistive nature of 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 [9] or cause excessive power consumption [10]. Moreover, they may indicate reliability issues, e.g., migration of metal to the surrounding areas over time and shortening the lifetime of the chip [2].

### B. 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 [11], [12]. 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 [13]. The performance of the switch-level tools such as SWITEST [14] or the analog simulators like SPICE [5] are not always acceptable, especially if large VLSI circuits have to be analyzed [15]. A different family of methods using mixed level or multilevel simulation techniques have been proposed in [16], [17]. These methods switch from functional 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 [13].

The above shortcomings motivated us to employ the fuzzy logic theory to model and simulate real faults. In [18], we presented an approach to model logic gates with limited simulation capability. 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 [19]. 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 [20].

### C. Contribution and Paper Organization

Our fault model assigns a nonzero resistance, randomly selected in a predefined range $[R_{\min}, R_{\max}]$, to the stuck at and bridging faults in general. Ideal (zero resistance) faults are 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 nonembedded logic gates from results reported by SPICE. This feature makes our method fully adaptable when new libraries, logic styles and technologies are used. 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 reports

true fault coverage by considering the real faults and thus improves the yield factor when chips are actually tested by a test equipment.

The rest of this paper is organized as follows. Our fault model is explained in Section II. Section III reviews the steps for developing a fuzzy system using two well-known models, i.e., Mamdani and Takagi–Sugeno (TS) fuzzy models. Section IV describes how an individual (nonembedded) logic gate is modeled accurately as a fuzzy block. Section V explains the fault simulation algorithm. In Section VI, we comment on how our simulator can be used to generate test patterns for real faults. The experimental results are discussed in Section VII. Finally, the concluding remarks are presented in Section VIII.

## II. Fault Model

Our basic fault model assumes a single resistive bridging fault ($R_{\mathrm{bridge}}$) exists in a circuit as shown in Fig. 4. A stuck at fault is a bridging fault, occurred between a specific node and $V_{dd}$ or $Gnd$, associated with a resistance. This resistance is 0 for ideal and has nonzero value range for real faults.

Feltham and Maly [21] 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 [21], or similar approaches such as [22] and [13], is in: 1) considering a predefined range of resistances and 2) accurate voltage analysis and propagation of their effects in the circuit using fuzzy logic.

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.

### A. Voltage Calculation

As presented by [23], [2], and [1] the resistance of the bridging faults may vary between several ohms to several kilo ohms depending strongly on the layout details, technology and fabrication process. However, the resistance of 0.5–2 K provided satisfactory results in test of digital circuits [23]. In our work, we allow the user to define a range (e.g., $[R_{\min}, R_{\max}]$) and the simulator will select a random resistance, i.e., $R_{\mathrm{bridge}}$ within this range.

Connection of two nodes via a bridging defect makes a resistive path between power supply and ground as shown in Fig. 4. Typical pullup and pulldown resistors of the driving gates are often given in data sheets of cell libraries [3], and therefore, two nodes of the bridging fault are analyzed by voltage division

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

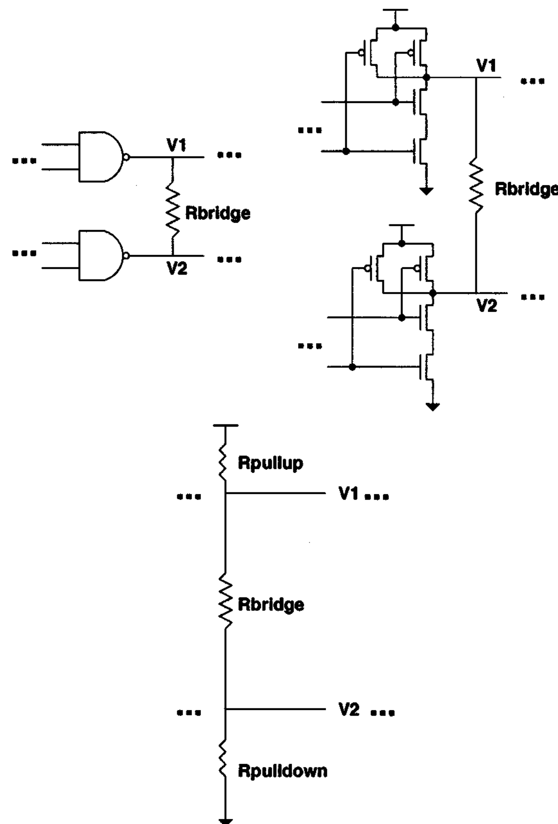



Fig. 4. A resistive path for a bridging fault.

### B. Voltage Propagation

The second factor that determines the accuracy of a real fault model is the propagation of the voltages at the two nodes of the fault (i.e., generalized as a bridge). In this step, the voltages are propagated accurately through the circuit, considering different threshold voltages of logic gates and their asymmetry. We have developed a fuzzy fault simulator, with SPICE precision for voltage propagation, to carry out this step. The fuzzy fault simulator is 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 $[0, 0.3V_{dd}]$ range is recognized as LOW, in $[0.7V_{dd}, V_{dd}]$ range is recognized as HIGH and between these two is considered MEDIUM (abnormal) as shown in Fig. 5 [24].

Note that these margins at 0.3 $V_{dd}$ and 0.7 $V_{dd}$ are not crisp and may slightly differ for each technology, cell 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 a fuzzy system can be used to approximate the behavior of logic gates [19].

## III. Developing a Fuzzy System

*Modeling* is one of the most popular applications of fuzzy systems and has been studied in various practical issues, such as fuzzy control in consumer products, industrial process control, etc. In this work, we intend to model logic gates as fuzzy

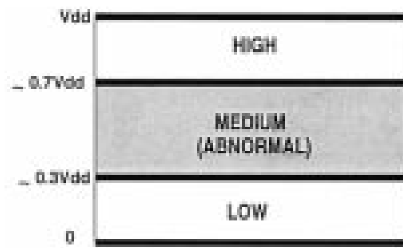



Fig. 5. Typical noise margins of CMOS gates.

blocks to exploit advantages of fuzzy systems in fault simulation of digital circuits. In general, selection of the fuzzy system configuration is a very early decision in the system modeling and significantly affects the system performance and accuracy. Consequently, the accuracy and efficiency of the model cannot be predicted at early stages.

### A. Modeling Choices and Practical Issues

Depending on the available information, different approaches can be employed for designing fuzzy systems. When the nonlinear function of a system is fully known, a mathematical model can be used instead of a fuzzy model to express its behavior. On the other hand, if the nonlinear function of the system is unknown but for any input the corresponding output can be determined, the *classic fuzzy modeling* method is often used. This method requires two steps. In the first step, we define a desired number of fuzzy sets which are: 1) *normal* (there is a point that its membership value is equal to one); 2) *consistent* (a unique membership value for one specific point); and 3) *complete* (for any point in the input space, there is a nonzero membership value). Then, in the second step, we construct the fuzzy IF–THEN rules and extract $f(x)$ from the rules considering an appropriate inference engine, fuzzifier and defuzzifier. Due to a wide range of applications and the artistic nature of modeling, most activities within these two steps are done in *ad-hoc* way. This includes optimization methods to achieve certain optimal aspects of fuzzy systems [25].

In what follows, we review three well-known fuzzy systems based on: 1) lookup table (LUT); 2) Mamdani; and 3) TS models. We start by discussing the practical pros and cons of approximating the input–output characteristics of the logic gates using the LUT model. Then, in the two subsections that follow, we use the Mamdani and TS fuzzy system types to model a single NOT gate as a test bench for evaluating the efficiency of these fuzzy system types for our application. We keep the requirements of two systems (i.e., the number of fuzzy sets and fuzzy rules) close together, to be able to have a reasonable comparison criteria.

More specifically, the Mamdani [26] and TS [27], [28] are two fuzzy system types recognized as successful strategies for modeling sophisticated behaviors. They are known as universal approximators, since they are able to approximate almost any continuous function with a desired level of accuracy [29], [30]. This hypothesis has been proved using the Stone–Weierstrass theorem [19]. However, depending upon data complexity, system behavior, and the desired accuracy one of these two types can be more efficient. In [31] and [32], the authors have developed different fuzzy applications using Mamdani and TS models to show which class of applications can be suitable for each. For example, the authors in [31] demonstrated that the minimal configuration of typical Mamdani and TS fuzzy systems strongly depends on the number and the location of the extrema of the function to be approximated.

### B. Lookp Table-Based Systems

When the analytical model of a system is unknown and only limited pairs of input and outputs are provided, a LUT mechanism can be used [25]. The LUT-based mechanism, widely used in many research areas, is an efficient representation that balances storage requirement versus data access time. Specifically, for fuzzy systems, a LUT-based approach can be implemented by performing the following four steps. 1) Analyze sufficient pairs of input–outputs and assign membership functions to the input–output pairs. The number and form of membership functions are generally selected *ad hoc* based on designer's experience and available data. 2) Determine the membership values in the input and output spaces based on each input–output pair. For instance, consider a system with two input ports, one output port and three membership functions assigned to each port. For each input–output pair, the corresponding values of membership functions are calculated. 3) Generate one rule from each input–output pair and resolve the conflicts. The number of input–output pairs is usually large and it is highly likely that there are conflicting rules, i.e., rules with the same *IF* parts but different *THEN* parts. To resolve the conflicts, a *degree* is assigned to each generated rule and thereby one rule with the maximum degree is opted from the conflicting group. There is no unique way to define the degree. In general, the degree has to reflect the value of a rule corresponding to a data pair. For example, the degree can be defined as the product of the values of membership functions corresponding to an input–output pair or its input–output rule counterpart. In this step, the fuzzy rule base is obtained that to large extent represents the main behavior of the system. 4) Select appropriate fuzzifier, defuzzifier, and inference engine to construct the fuzzy system. When chosen properly based on the limited number of input–output pairs the LUT method leads to an efficient fuzzy system [33].

In our application of modeling logic gates, all possible input–output pairs are available, e.g., from SPICE simulation. If high accuracy is desired, the gate analysis provides huge number of input and output pairs and their corresponding membership functions. Generating one rule for each input–output pair will lead to impractical number of rules. Assigning degrees to the rules reduces the number of rules but overall it remains big enough to significantly slow down the simulation. Creating an intelligent LUT-based model for logic gates to accurately approximate the input–output characteristics is certainly possible. However, due to the nature of logic gates and our definition of fixed membership functions, we found the classic fuzzy modeling method more straightforward for tuning and optimization and less demanding for memory and simulation time.

### C. Mamdani Fuzzy System

Fuzzy knowledge is expressed by the concept of fuzzy sets and linguistic variables that are often defined as membership

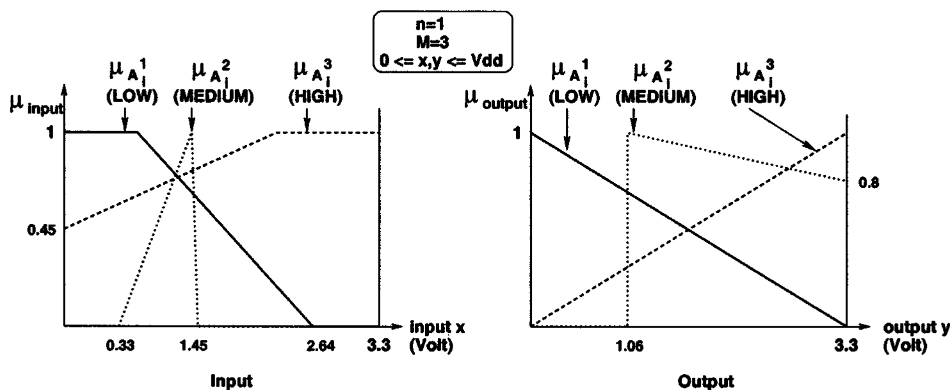



Fig. 6. Membership functions for a NOT gate.

functions. Additionally, a fuzzy rule base is also necessary for representing fuzzy knowledge. The fuzzy rule base comprises the rules defined in general as

$$\text{Rule}^{(l)}\text{: IF } (x_1 \text{ is } A_1^l) \,\&\, \cdots \,\&\, (x_n \text{ is } A_n^l) \qquad \text{THEN } (y \text{ is } B^l). \tag{2}$$

In (3), $n$ is the number of inputs of the fuzzy system and $A_i^l$ and $B^l$ are fuzzy sets. $x_i$ and $y$ are the input and output variables of the fuzzy system which taking their values in the universe of discourse $X$ and $Y$, respectively. We denote the membership functions of fuzzy sets $A_i^l$ and $B^l$ as $\mu_{A_i^l}$ and $\mu_{B^l}$, respectively, where

$$\begin{cases} \mu_{A_i^l}: X \rightarrow [0,\, 1] \\ \mu_{B^l}: Y \rightarrow [0,\, 1] \\ i = 1,\, 2,\, \ldots,\, n \\ l = 1,\, 2,\, \ldots,\, M. \end{cases} \tag{3}$$

In (3), $n$ is the number of inputs of the fuzzy system and $M$ is the number of IF–THEN rules. To complete the Mamdani fuzzy system, the minimum fuzzy inference engine is used in this work. The main advantage of this inference engine is in its computational simplicity. For a given input $x^\star = (x_1^\star, x_2^\star, \ldots, x_n^\star) \in X$, the output of fuzzy inference engine ($\mu_{B'}(y)$) is defined as follows:

$$\mu_{B'}(y) = \max_{\substack{1 \le l \le M \\ y \in Y}} \left[ \min_{y \in Y} \{ \mu_{A_1^l}(x_1^\star),\, \mu_{A_2^l}(x_2^\star),\, \ldots,\, \mu_{A_n^l}(x_n^\star),\, \mu_{B^l}(y) \} \right] \tag{4}$$

where the $\min$ operator selects the minimum value among the values of membership functions in the IF proposition of a given input $x^\star$ and the membership function of the THEN proposition of the universe of discourse of the output $Y$. For obtaining the final output of the fuzzy system, a defuzzifier is needed. The defuzzifier specifies a point in the universe of discourse of output, which best represents the fuzzy set at the output of the fuzzy system. We have used the mean of maxima (MoM) method [19] as defuzzifier in our Mamdani system.

*Modeling a NOT Gate:* To model a NOT gate as a fuzzy system, three linguistic variables LOW, MEDIUM, and HIGH are considered corresponding to the input space partitions of logic levels in CMOS technology. A membership function is defined for each region. The membership functions can be obtained empirically or by an optimization technique based on the behavior of the NOT gate in the different regions.

Examining the analog behavior of NOT gate shows that differential changes of the output with respect to the input depends on the region where input changes. When the input is in the MEDIUM region, any incremental changes in the input results in significant changes in the output. Therefore, the membership function needs to be sharp. In contrast to the MEDIUM region, the LOW and HIGH regions can be described in terms of static output for a given input. The membership functions of the NOT gate in the input and the output, shown in Fig. 6, are experimentally obtained. Using the interpretability of fuzzy system, each membership function is dedicated to a suitable linguistic variable, which can be tuned by changing the membership function parameters considering the interpretation of the desired behavior.

The rule set that configures the fuzzy system for a NOT gate is as follows:

$$\begin{cases} \text{Rule}^{(1)}\text{: IF the } \mathit{input} \text{ is LOW} \\ \qquad\qquad \text{THEN the } \mathit{output} \text{ is HIGH} \\ \text{Rule}^{(2)}\text{: IF the } \mathit{input} \text{ is MEDIUM} \\ \qquad\qquad \text{THEN the } \mathit{output} \text{ is MEDIUM} \\ \text{Rule}^{(3)}\text{: IF the } \mathit{input} \text{ is HIGH} \\ \qquad\qquad \text{THEN the } \mathit{output} \text{ is LOW.} \end{cases}$$

Fig. 7 illustrates the method. We use the above rules and membership functions to find the output of a NOT gate $y^\star$ when the input is $x_1^\star$. The $\min$ operator selects the minimum values among the membership functions. This selection is shown using thick solid lines [corresponding to $\mu_{B^1}(y)$, $\mu_{B^2}(y)$ and $\mu_{B^3}(y)$] in Fig. 7.

The $\max$ operator is used to aggregate the output of the fuzzy model. In our example, this is done by selecting the maximum among three curves [i.e., $\mu_{B^1}(y)$, $\mu_{B^2}(y)$ and $\mu_{B^3}(y)$] in a piecewise fashion. The result will be a curve made of line pieces similar to what obtained in Fig. 7. Finally, the MoM method is used as defuzzifier to provide the final output of fuzzy model of NOT as a voltage value. Graphically, this is the center of the horizontal piece corresponding to the maximum $\mu_{B'}(y)$ as shown

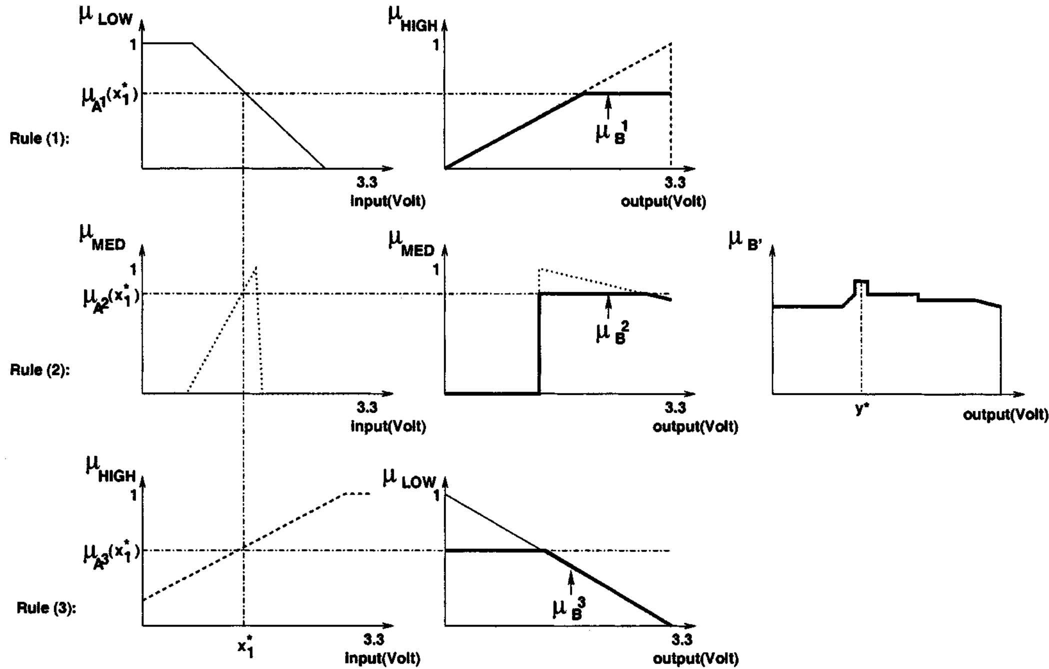



Fig. 7. Graphical representation of Mamdani's method for NOT gate.

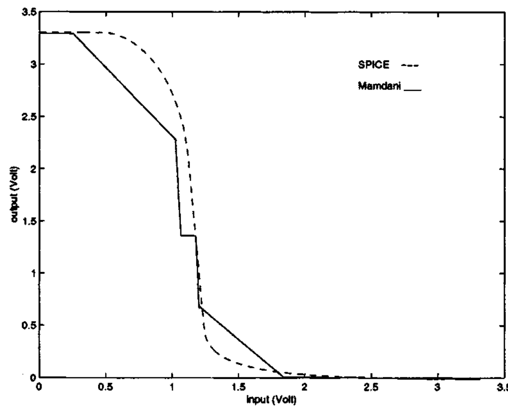



Fig. 8. Comparing output of Mamdani model and SPICE for NOT gate.

also in Fig. 7. Fig. 8 shows the output of Mamdani model in comparison with SPICE output for a NOT gate.

In general, the Mamdani fuzzy system is a modeling strategy that can be designed by formulating a qualitative knowledge about the system behavior. Due to the incompleteness of knowledge, the rules and its predicates need to be revised several times in order to tune and optimize the system. Considering these requirements, tuning the Mamdani fuzzy system to model a NOT gate was difficult and inaccurate as shown in Fig. 8. More importantly, the main relations representing the Mamdani model is neither continuous nor differentiable (due to the presence of MAX or MIN operator). Therefore, most efficient optimization techniques that use derivatives, e.g., gradient descent method, cannot be applied. Some researchers used the evolutionary optimization techniques for optimization. However, they are much more time consuming compared to the derivative-based methods [20]. This makes the Mamdani model less appealing for our fault simulation application.

### D. TS Fuzzy System

The output of the TS model is a linear function of input variables, therefore, the TS fuzzy system can be viewed as a somewhat piecewise linear function, where the change from one piece to another is smooth rather than abrupt [20]. According to this model, the fuzzy system $f(X)$, $X = [x_1, x_2, \ldots, x_n]$ is of the following form:

$$f(X) = \frac{\sum_{l=1}^{M} \left[Z^l(X) \cdot W^l(X)\right]}{\sum_{l=1}^{M} W^l(X)} \qquad (5)$$

where $M$ is the number of IF–THEN rules and $Z^l(X)$ (output of the $l$th rule) and $W^l(X)$ (excitation weight of the $l$th 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 $l$th rule, $n$ is the number of inputs, $\mu_i$ and $\sigma_i$ denote average and standard deviation of the membership functions, respectively. Finally, $k_i$ and $c_i$ represent a factor and a constant of the polynomial defined in the first-order TS model, respectively. Note that $\mu_i$, $\sigma_i$ , and $k_i$ are free parameters, which need to be optimized (tuned) to complete the fuzzy systems.

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

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

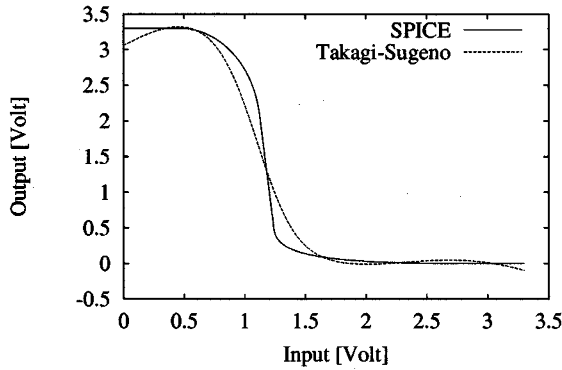



Fig. 9. Comparing SPICE and the TS fuzzy model of a NOT gate after initial setting.

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 [19].

*Modeling a NOT Gate:* To initialize the system, we must first determine initial rules and initial values of $\mu_i$ and $\sigma_i$. These initial parameters are determined empirically using linear approximation. Similar to Mamdani system, we partition the input universe of discourse to the three spaces, i.e., LOW, MEDIUM and HIGH. They refer to three Gaussian membership functions with initial $[\sigma_i, \mu_i]$ values of LOW $= [0.35, 0.60]$, MEDIUM $= [0.8, 1.8]$, and HIGH $= [0.75, 2.8]$, respectively.

Fig. 9 shows the SPICE output and fuzzy output for a NOT gate after these initial settings, where the output behaves imprecisely in some ranges. However, comparing to Fig. 8, we conclude that the behavior of NOT gate is obtained more accurately using TS fuzzy system type, with the similar system requirements such as the number of rules and membership functions for Mamdani and TS model.

Although the NOT gate modeled by TS is more accurate than Mamdani's model, we need to model the logical gates more accurately to address fault simulator precision. Therefore, we applied one optimization technique to determine the free parameters, i.e., $\mu_i$, $\sigma_i$, and $k_i$, more precisely. This is explained in the next section.

## IV. Modeling Logic Components as Fuzzy Blocks

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 [19].

### A. Find the Input–Output Behavior

We simulate all logic components in the library by SPICE with desired accuracy (e.g., 0.01 V). Note that SPICE simulation is done once and the input–output data obtained is 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) has to be covered.

### B. Design and Optimization of Fuzzy Gates

There are basically two approaches to construct fuzzy systems from input–output pairs of data [19]. 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 a fuzzy inference engine, a fuzzifier, and a defuzzifier. In the second approach, the structure of the fuzzy system is predesigned 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 TS model [19] as the basic structure of the fuzzy blocks. A nonlinear least square method is utilized to optimize the free parameters. This method plays a prominent role in the framework of soft computing and the sum of squared errors is frequently chosen as the objective function to be minimized [20]. This method is commonly used in data fitting and regression involving nonlinear models [20] and is briefly described in the following.

Consider an $n$-input, single-output model with $m$ free parameters: $y = f(X, \Theta)$ where $y$ is the model's scalar output, $X = [x_1, \ldots, x_n]$ is the input vector of size $n$, and $\Theta = [\theta_1, \theta_2, \ldots, \theta_m]^T$ is the parameter vector of size $m$. 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 $\Theta^*$ that minimizes $E(\Theta)$ is of primary concern:

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

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 $\mathbf{r}(\Theta) = [r_1(\theta), \ldots, r_m(\theta)]$.

Since $E(\Theta)$ is nonlinear, to minimize it we use the iterative descend method [19], in which the next point $\Theta_{\text{next}}$ is determined by a step down from the current point $\Theta_{\text{now}}$ in a direction vector $g(\Theta)$

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

$g(\Theta)$ 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 least square Levenberg–Marquart method [20] to determine $g(\Theta)$ and $\eta(\Theta)$.

### C. Fuzzy Logic Versus SPICE

In our formulation, $\Theta = [\mu_i, \sigma_i, k_i]$ is the parameter vector. After optimizing $\Theta$ the fuzzy model for NOT gate shows very high accuracy compared to the SPICE, as shown in Fig. 10. The cell is selected from a library using $0.5\mu$m technology and $V_{dd} = 3.3$ V.

Fig. 10(a) and (b) show the outputs of SPICE versus fuzzy simulation and absolute error for each of the 200 input patterns (voltages) in range of $[0, 3.3]$. Fig. 11(a) and (b) show the result of fuzzy simulator and absolute error for a two input AND gate, respectively. The behavior of the AND gate is approximated very accurately, such that the maximum error for all input combinations is less than 0.03 V as shown in Fig. 11(b) and mean square error is less than 0.04.

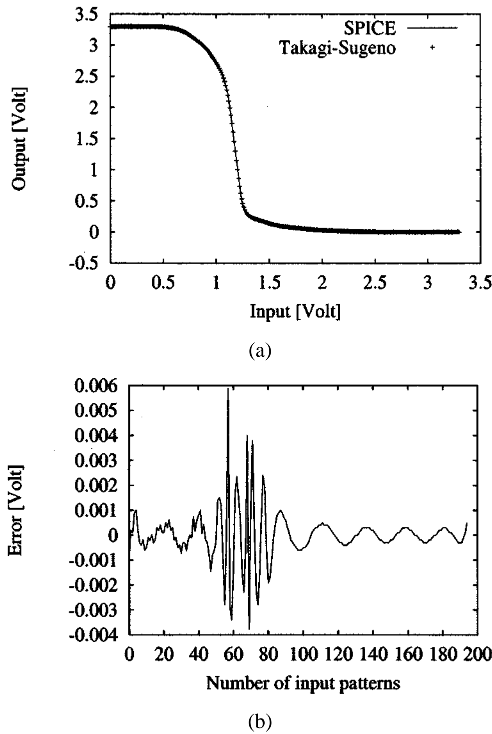



Fig. 10. Comparing SPICE and fuzzy simulators for a NOT gate.

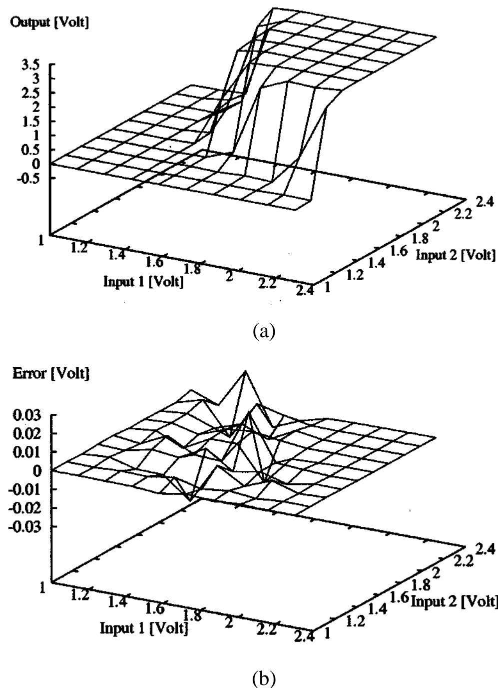



Fig. 11. Comparing SPICE and fuzzy simulators for a 2-input AND gate.

In modeling a logic circuit using fuzzy logic, the small error observed in the intermediate levels (e.g., 0.03 V in the previous example) is not accumulated in the process and thus could be ignored, because the intermediate voltages reach to $V_{dd}$ or $Gnd$ after few levels anyway. To show this, we compared SPICE and fuzzy simulator for a multilevel circuit shown in Fig. 12(a) for 350 set of random input voltages. The output of fuzzy simulator is still very close to the actual level reported by SPICE and the error is negligible.

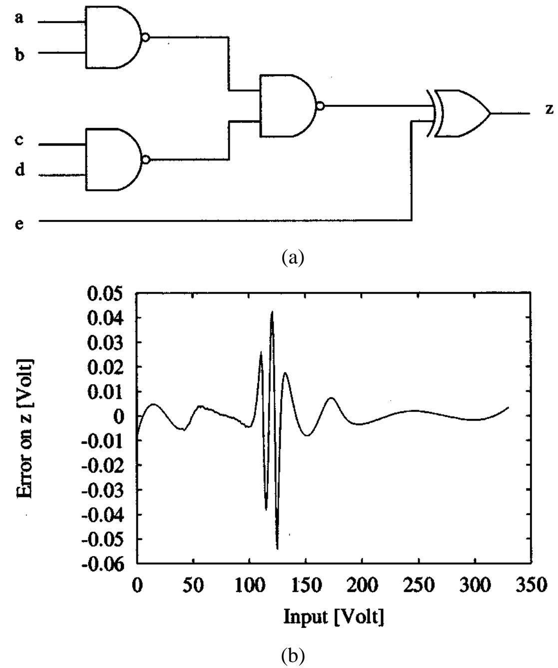


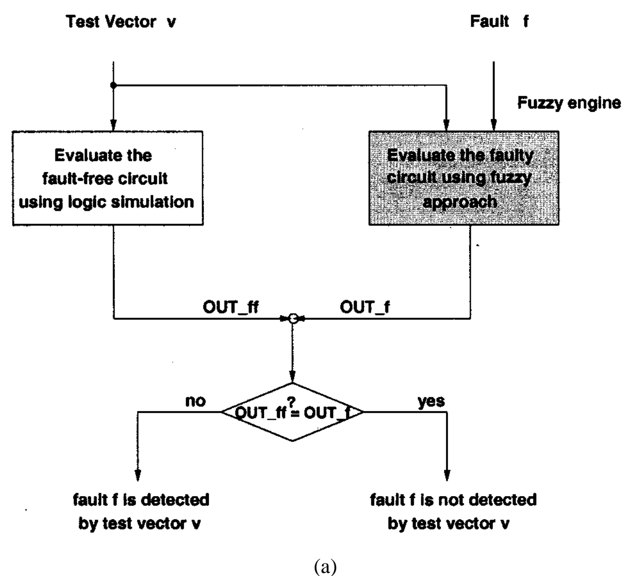

Fig. 12. Comparing SPICE and fuzzy simulators for a multilevel circuit.

The whole practical point about our fuzzy simulator is that a real (resistive stuck at or bridging) fault that causes abnormal level of voltages in the circuit can be traced carefully toward the output(s). This is a fundamental necessity for real fault detection.

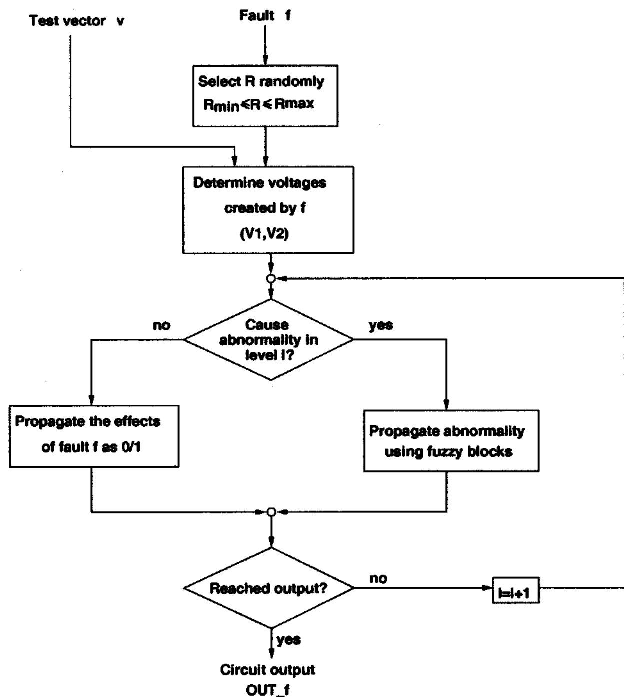
It is worth mentioning that once we model all logic gates as fuzzy blocks the fuzzy simulator runs quite fast. For example, simulating 350 patterns for the circuit shown in Fig. 12 is much faster than SPICE, i.e., 0.12 s compared to 2.70 s CPU time in a SPARC Ultra 1 for this circuit. Fig. 12 is a small circuit showing the proof of concept. Our experimentation on large circuits, to be discussed in Section VII, shows that the fuzzy simulator is very time-efficient in handling large circuits. This is due to the fact that in our application, there is no need to perform all fuzzy computations [e.g., (5)] at every point in the circuit. Only if the input of a gate is within the abnormal range, (5) needs to be evaluated. Otherwise, the regular logic operation is performed. This makes the fuzzy simulator much faster than SPICE for resistive fault simulation. Note also that accurate simulators such as SPICE use sophisticated transistor models and matrix calculations to provide accurate waveform (timing, voltage) information at every point. Such detailed information is not needed for fault simulation. Our simulator works at the gate level and invokes the fuzzy engine only when abnormal voltages are confronted. In Section VII, we will show empirical evidence that our fuzzy fault simulator runs quite efficiently on large (160–3500 gates) ISCAS85 benchmarks.

## V. 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 pseudoanalog simulation working at the gate level is the most important feature in our




(a)




(b)

Fig. 13. Fault simulation process.

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 corresponding 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 is detected [4].

Fig. 13(a) shows the core of our fuzzy simulator. The complete fuzzy simulation environment employs this core in an iterative process to apply all test vectors to check the possibility of detecting all faults. As shown in Fig. 13(a), the circuit under test is evaluated under "fault-free" and "single-fault $f$" assumption. If the two evaluations are different, the fault $f$ is tagged as detected. Otherwise, vector $v$ cannot detect fault $f$.

Fig. 13(b) details the fuzzy engine evaluator shaded in Fig. 13(a). The resistance of the real fault (stuck at or bridge) is selected randomly from the range predefined by user. In practice, such range is generated by empirical and statistical data and varies for different styles, technologies and even fabrication plants [34], [2]. The voltage of two sides of the resistive fault is calculated using (1). If the voltage is within normal range, i.e., LOW or HIGH; 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., MED, 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 that see abnormality and trace the effect carefully through that level. This process is repeated until we reach the output or the voltage gets to the normal ranges (LOW or HIGH). 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 (usually 2, 3, or 4) the voltages enter normal range and we can continue simulation with higher speed by not entering the fuzzy computations.

## VI. Test Pattern Generation for Real Faults

Although test-pattern generation 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 resistive faults. We believe our fuzzy engine can be also used to generate patterns, which overall have a better chance to detect real faults and so the fault coverage can be enhanced. Accordingly, our basic strategy is to first use a conventional test-pattern generation algorithm (e.g., PODEM [35]) to generate test vectors and then ask the fuzzy simulator to evaluate different choices, when they exist, for their potential in detecting a real fault with a resistance $R_f$ in a given 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.

The general scheme to generate patterns for real faults is summarized in Fig. 14. When we deal with the real stuck at faults, we just ask PODEM to generate the patterns (line 7). 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 nodes of the bridge to opposite logic values (0 and 1), to excite the fault. Fig. 15 shows that by a simple trick we can use PODEM to generate the test pattern. We insert a two-input XOR

```
FTPG ()
Input: gate level circuit and a fault list.
Output: test pattern(s) if exist.

01:repeat {
02:    select a fault f from the fault list.
03:    if f is a bridging fault {
04:        replace the fault with an XOR
05:        redefine f as s-a-0 at the XOR output
06:    }
07:    run PODEM to find test vector(s) for f
08:    if (found) {
09:        run fuzzy simulator to verify and select patterns to detect R_f ∈ [R_min, R_max]
10:        return (test vector(s))
11:    }
12:    else
13:        return (not-found)
14: } until (all faults are checked)
```

Fig. 14. Pseudocode of the fuzzy-based test-pattern generation (FTPG) procedure.

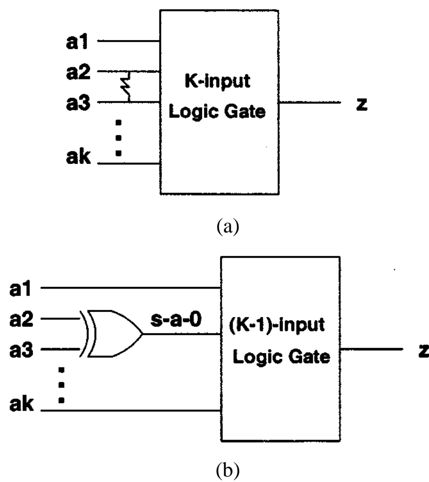



Fig. 15. XOR replacement for bridging faults. (a) Original gate and a fault. (b) After XOR replacement.

TABLE I
BENCHMARK CIRCUITS USED FOR EXPERIMENTS

| *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 |

gate in the location of the bridging fault (line 4) and ask PODEM to generate a pattern to detect stuck at-zero 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 at two nodes of the fault) to proceed. Note carefully that appropriate (noncontrolling) values for propagation for other gate 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 (line 9). Currently, our procedure takes a conservative strategy and selects more patterns than PODEM to catch a wide range of resistive faults.

## VII. 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 to 39.9 s. wall-clock time. Table I summarizes the specifications of the benchmarks considered in this work. The bridging faults are assumed only among the inputs of the gates. Tables II and III show the fault simulation results by conventional ($R_f = 0$) and fuzzy ($R_f \in [R_{\min}, R_{\max}]$) simulators, for stuck at and bridging faults, respectively. Subscript $\mathrm{set_1}$ refers to the patterns obtained by PODEM while $\mathrm{set_2}$ is the set of patterns produced by our FTPG algorithm. $N_{\mathrm{set_1}}$ and $N_{\mathrm{set_2}}$ are the number of patterns in $\mathrm{set_1}$ and $\mathrm{set_2}$, respectively. The corresponding fault coverage values are listed under $FC_{\mathrm{set_1}}$ and $FC_{\mathrm{set_2}}$ in these tables.

A conventional fault simulator considers only ideal faults ($R_f = 0$) and its report on fault coverage (third column in Tables II and III) is simply too optimistic. Such idealistic analysis by fault simulators fails to predict the true statistics when automatic test equipment (ATE) actually test the real chips. One advantage of our approach is in its realistic view during fault simulation. Our fuzzy simulator reports lower fault coverage (the fourth column) for the same set of patterns because it considers the real resistive faults and predicts accurately the situation that ATE will experience in actual testing.

Additionally, by using our fuzzy test pattern generation procedure (FTPG) detecting real faults is improved with the cost of more patterns, and thus more test time, to apply. This is reflected in the fifth and sixth columns in Tables II and III. The last two columns in these two tables show the percentage of increase in the number of patterns [i.e., $\Delta N = (N_{\mathrm{set_2}} - N_{\mathrm{set_1}})/(N_{\mathrm{set_1}})$] and fault coverage ($\Delta FC = (FC_{\mathrm{set_2}} - FC_{\mathrm{set_1}})/(FC_{\mathrm{set_1}})$), respectively.

TABLE II
TEST RESULTS FOR STUCK-AT FAULTS

| *Circuits* | *Conventional Approaches* ($R_f = 0$) | | *Fuzzy Logic Approach* ($R_f \in [0.5k, 2k]$) | | | | |
|---|---|---|---|---|---|---|---|
| | $N_{set_1}$ | $FC_{set_1}$ % | $FC_{set_1}$ % | $N_{set_2}$ | $FC_{set_2}$ % | $\Delta N\%$ | $\Delta FC$ % |
| C432 | 77 | 99.4 | 78.3 | 108 | 85.7 | +40.3 | +8.5 |
| C880 | 103 | 100 | 87.3 | 112 | 94.3 | +8.7 | +8.0 |
| C1355 | 102 | 100 | 82.9 | 143 | 91.7 | +40.2 | +10.6 |
| C1908 | 149 | 100 | 89.1 | 201 | 100 | +34.9 | +12.2 |
| C2670 | 153 | 98.8 | 77.5 | 178 | 94.5 | +16.3 | +21.9 |
| C3540 | 278 | 98.4 | 88.3 | 329 | 96.3 | +18.3 | +8.0 |
| C5315 | 248 | 99.5 | 89.6 | 302 | 100 | +21.8 | +11.6 |
| C7552 | 348 | 98.7 | 84.1 | 417 | 93.9 | +19.8 | +11.6 |

TABLE III
TEST RESULTS FOR BRIDGING FAULT

| *Circuits* | *Conventional Approaches* ($R_f = 0$) | | *Fuzzy Logic Approach* ($R_f \in [0.5k, 2k]$) | | | | |
|---|---|---|---|---|---|---|---|
| | $N_{set_1}$ | $FC_{set_1}$ % | $FC_{set_1}$ % | $N_{set_2}$ | $FC_{set_2}$ % | $\Delta N\%$ | $\Delta FC$ % |
| C432 | 121 | 87.9 | 67.3 | 169 | 73.2 | +39.7 | +8.8 |
| C880 | 149 | 99.3 | 75.1 | 287 | 87.9 | +92.6 | +17.0 |
| C1355 | 217 | 95.3 | 65.5 | 293 | 71.9 | +35.0 | +9.8 |
| C1908 | 193 | 96.4 | 78.3 | 281 | 89.5 | +45.6 | +14.3 |
| C2670 | 183 | 88.6 | 74.9 | 401 | 83.1 | +119.1 | +10.9 |
| C3540 | 292 | 93.3 | 73.7 | 483 | 81.2 | +65.4 | +10.2 |
| C5315 | 312 | 98.9 | 79.9 | 378 | 87.2 | +21.1 | +9.1 |
| C7552 | 373 | 91.4 | 72.7 | 545 | 79.8 | +46.1 | +9.8 |

For stuck at faults (Table II), our test pattern generation strategy uses up to 40% more patterns to be able to detect a wide range ($[1K–1.5K]$) of resistive faults. By applying more patterns, the fault coverage has improved 8–22%. For bridging faults (Table III), our approach uses 21%–120% more patterns but achieves 8.8%–17% more fault coverage. Obviously, applying more patterns prolongs the test time. However, from practical point of view this is well justified since we achieve high testability and avoid sending faulty chips to the market.

The results of the fault grading heavily depends on the nature of the circuits and it is impossible to draw a conclusion as to when our method generates less pattern overhead or more fault coverage. Combining all statistics in Tables II and III we conclude that for these eight benchmarks our method on the average uses 41.5% more patterns but achieves 11.4% more fault coverage. In test environments, this is a significant test improvement with reasonable overhead.

Practically, the statistical analysis of fabrication process often limits the fault resistance range [2] and therefore for methods such as ours there will be modest test time increase. To show the effect, in another set of experiments we limited the resistance range to $[1K–1.5K]$. We observed that the average overhead for the number of patterns is reduced to 18% (as opposed to 41.5%) for the eight benchmarks reported in this section while the fault coverage improvement remained around 11%.

## VIII. 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 system engine based on TS model 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.

## REFERENCES


[1] F. Hawkins, J. Soden, A. Righter, and F. Ferguson, "Defect classes—An overdue paradigm for CMOS IC testing," in *Proc. Int. Test Conf.*, Oct. 1994, pp. 413–425.

[2] R. Aitken, "Finding defects with fault models," in *Proc. Int. Test Conf.*, Oct. 1995, pp. 498–505.

[3] N. Weste and K. Eshraghian, *Principles of CMOS VLSI Design*. Reading, MA: Addison-Wesley, 1993.

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

[5] Texas Instrument, Inc. *TI SPICE3 user's and reference manual*, 1994.

[6] S. Shen, W. Maly, and F. Ferguson, "Inductive fault analysis of MOS integrated circuits," *IEEE Design Test Comput.*, vol. 2, pp. 13–26, Dec. 1985.

[7] P. Maxwell and R. Aitken, "Biased voting: A method for simulating CMOS bridging faults in the presence of variable gate logic thresholds," in *Proc. Int. Test Conf.*, 1993, pp. 63–72.

[8] D. Millman and J. Garvey, "An accurate bridging fault test pattern generator," in *Proc. Int. Test Conf.*, 1991, pp. 411–418.

[9] S. Sparmann, D. Luzenburger, K. Cheng, and S. Reddy, "Fast identification of robust dependent path delay faults," in *Proc. 32nd Design Automation Conf.*, June 1995, pp. 119–125.

[10] M. Nourani, J. Carletta, and C. Papachristou, "A scheme for integrated controller-datapath fault testing," in *Proc. 34th Design Automation Conf.*, June 1997, pp. 546–551.

[11] M. Acken and S. Millman, "Fault model evolution for diagnosis: Accuracy vs. precision," in *Proc. IEEE Custom Integrated Circuits Conf.*, 1992, pp. 13.4.1–13.4.4.

[12] B. Chess and T. Larrabee, "Bridge fault simulation strategies for CMOS integrated circuits," in *Proc. Design Automation Conf.*, 1993, pp. 1503–1507.

[13] C. Di and J. Jess, "An efficient CMOS bridging fault simulator: With spice accuracy," *IEEE Trans. Comput. Aided Design*, vol. 15, pp. 1071–1080, Sept. 1996.

[14] J. Lee, C. Njinda, and M. Breuer, "SWITEST: A switch level test generation system for CMOS combinational circuits," in *Proc. Design Automation Conf.*, June 1992, pp. 26–29.

[15] U. Mahlstedt and J. Alt, "Simulation of nonclassical faults on the gate level: The fault simulator COMSIM," in *Proc. Int. Test Conf.*, 1993, pp. 883–892.

[16] J. Rearick and J. Patel, "Accurate CMOS bridging fault simulation," in *Proc. Int. Test Conf.*, 1993, pp. 54–62.

[17] S. Greenstein and J. Patel, "E-PROOFS: A CMOS bridging fault simulator," in *Proc. Int. Conf. Computer Aided Design*, 1992.

[18] A. Attarha, M. Nourani, and C. Lucas, "Modeling and simulation of real defects using fuzzy logic," in *Proc. 37th Design Automation Conf.*, June 2000, pp. 631–636.

[19] X. Wang, *A Course in Fuzzy Systems and Control*. Upper Saddle River, NJ: Prentice-Hall, 1997.

[20] S. R. Jang, C. T. Sun, and E. Mizutani, *Neuro-Fuzzy and Soft Computing*. Upper Saddle River, NJ: Prentice-Hall, 1997.

[21] D. Felthaam and W. Maly, "Physically realistic fault models for analog CMOS neural networks," *IEEE J. Solid-State Circuits*, vol. 26, pp. 1223–1229, Sept. 1991.

[22] D. Lavo, B. Chess, T. Larrabee, and F. Ferguson, "Diagnosing realistic bridging faults with single stuck-at information," *IEEE Trans. Comput. Aided Design*, vol. 17, Mar. 1998.

[23] M. Dalpasso, M. Favalli, P. Olivo, and B. Ricco, "Fault simulation of parametric bridging faults in CMOS IC's," *IEEE Trans. Comput. Aided Design*, vol. 12, pp. 1403–1410, Sept. 1993.

[24] J. Wakerly, *Digital Design Principles and Practices*. Upper Saddle River, NJ: Prentice-Hall, 1990.

[25] X. Wang and J. M. Mendel, "Generating fuzzy rules by learning from examples," *IEEE Trans. Syst., Man, Cybern.*, vol. 22, pp. 1414–1427, Nov. 1992.

[26] E. Mamdani and S. Assilian, "An experiment in linguistic synthesis with a fuzzy logic controller," *Int. J. Man Mach. Studies*, vol. 7, no. 1, pp. 1–13, 1975.

[27] T. Takagi and M. Sugeno, "Fuzzy identification of systems and its applications to modeling and control," *IEEE Trans. Syst., Man, Cybern.*, vol. SMC-15, pp. 116–132, Feb. 1985.

[28] J. J. Buckley, "Sugeno type controllers are universal controllers," *Fuzzy Sets Syst.*, vol. 53, pp. 299–303, 1993.

[29] B. Kosko, "Fuzzy systems as universal approximators," in *Proc. IEEE Int. Conf. Fuzzy Syst.*, San Diego, CA, 1992, pp. 1153–1162.

[30] J. Kickerson and B. Kosko, "Fuzzy function approximation with ellipsoidal rules," *IEEE Trans. Syst., Man, Cybern.*, vol. 26, pp. 542–560, July 1996.

[31] H. Ying, Y. Ding, S. Li, and S. Shao, "Comparison of necessary conditions for typical Takagi–Sugeno and Mamdani fuzzy systems as universal approximators," *IEEE Trans. Syst., Man, Cybern.*, vol. 29, pp. 508–514, Sept. 1999.

[32] M. Figueriredo, F. Gomide, A. Rocha, and R. Yager, "Comparison of yager's level set method for fuzzy logic control with Mamdani's and Larsen's methods," *IEEE Trans. Fuzzy Syst.*, vol. 1, pp. 156–159, Apr. 1993.

[33] E. Cox, *The Fuzzy Systems Handbook*. New York: Academic, 1994.

[34] A. Miller, "IDDQ testing in deep submicron integrated circuits," in *Proc. Int. Test Conf.*, 1999, pp. 724–729.

[35] P. Goel and B. Rosales, "PODEM-X: An automatic test generation system for VLSI logic structures," in *Proc. Design Automation Conf.*, June 1981, pp. 260–268.

**Mehrdad Nourani** (S'93–M'95) received the B.Sc. and M.Sc. degrees in electrical engineering from the University of Tehran, Tehran, Iran, and the Ph.D. degree in computer engineering from Case Western Reserve University, Cleveland, OH, in 1986 and 1993, respectively.

During the academic year 1994, he was a Postdoctoral Fellow with the Department of Computer Engineering, Case Western Reserve University. He served in the Department of Electrical and Computer Engineering at the University of Tehran, from 1995 to 1998, and the Department of Electrical Engineering and Computer Science, Case Western Reserve University from 1998 to 1999. Since August 1999, he has been on the faculty of the University of Texas at Dallas, Richardson, TX, where he is currently an Assistant Professor of electrical engineering and a member of the Center for Integrated Circuits and Systems (CICS). His current research interests include design for testability, system-on-chip testing, signal integrity modeling and test, application specific processor architectures, high-level synthesis and low-power design methodologies.

Dr. Nourani has received the Texas Telecommunications Consortium Award (1999), the Clark Foundation Research Initiation Grant (2001), and the National Science Foundation Career Award (2002). He is a Member of the IEEE Computer Society and the ACM.

**Amir R. Attarha** received the B.Sc. and M.Sc. degrees in computer engineering from the University of Tehran, Tehran, Iran, in 1997 and 1999, respectively. He is currently working toward the Ph.D. degree in electrical engineering at the University of Texas at Dallas, Richardson.

His research interests include high-speed arithmetic unit design, interconnect testing, fault modeling, and memory testing.

**Caro Lucas** received the M.S. degree from the University of Tehran, Iran, and the Ph.D. degree from the University of California, Berkeley, in 1973 and 1976, respectively.

He is a Professor with the Department of Electrical and Computer Engineering, the University of Tehran, as well as a Researcher at the Intelligent Systems Research Faculty (ISRF), Institute for Studies in Theoretical Physics and Mathematics (IPM), Tehran, Iran. He was a Visiting Associate Professor at the University of Toronto, ON, Canada, (summer, 1989–1990), the University of California, Berkeley (1988–1989), an Assistant Professor at Garyounis University, Libya (1984–1985), the University of California, Los Angeles (1975–1976), a Senior Researcher at the International Center for Theoretical Physics and the International Center for Genetic Engineering and Biotechnology, both in Trieste, Italy, the Institute of Applied Mathematics, Chinese Academy of Sciences, Harbin Institute of Electrical Technology, a Research Associate at Manufacturing Research Corporation of Ontario, Canada, and a Research Assistant at the Electronic Research Laboratory, University of California, Berkeley. He is currently an Associate Editor of *Journal of Intelligent and Fuzzy Systems*. He holds a patent on a speaker independent Farsi isolated word neurorecognizer. His research interests include biological computing, computational intelligence, uncertain systems, intelligent control, neural networks, multiagent systems, data mining, business intelligence, financial modeling, and knowledge management. He has served as the Chairman of several international conferences, and was the founder of the ISRF. He has assisted in founding several new research organizations and engineering disciplines in Iran.

Dr. Lucas was the recipient of several research grants from the University of Tehran and ISRF. He has served as the Director of ISRF (1993–1997), Chairman of the Electrical and Computer Engineering Department at the University of Tehran (1986–1988), Managing Editor of the Memories of the Engineering Faculty, University of Tehran (1979–1991), Reviewer of Mathematical Reviews (since 1987), and Chairman of the IEEE, Iran Section (1990–1992).