X-Masking During Logic BIST and Its Impact on Defect Coverage

Yuyi Tang   Hans-Joachim Wunderlich

Institute of Computer Architecture and Computer Engineering, University of Stuttgart
Pfaffenwaldring 47, D-70569 Stuttgart, Germany

Harald Vranken   Friedrich Hapke   Michael Wittke

Philips Research Laboratories   Philips Semiconductors GmbH
Prof. Holstlaan 4, Building WAY 4.101   Design Technology Center
5656 AA Eindhoven, The Netherlands   Georg-Heyken-Str. 1, 21147 Hamburg, Germany

Piet Engelke   Ilia Polian   Bernd Becker

Institute for Computer Science, Albert-Ludwigs-University
Georges-Köhler-Allee 51, D-79110 Freiburg i. Br., Germany

Abstract

We present a technique for making a circuit ready for Logic BIST by masking unknown values at its outputs. In order to keep the area overhead low, some known bits in output responses are also allowed to be masked. These bits are selected based on a stuck-at n-detection based metric, such that the impact of masking on the defect coverage is minimal. An analysis based on a probabilistic model for resistive short defects indicates that the coverage loss for unmodeled defects is negligible for relatively low values of n.

Keywords: X-Masking, Logic BIST, Defect Coverage, Resistive Bridging Faults

1 Introduction

Built-in self test solves many of today's testing problems, including pin throughput issues, complexity of test programs and test application at speed, and enables in-field testing [1]. While BIST became industry standard for memories in the 1990s, [2], there are still some obstacles for its application to random logic. One class of circuits that are difficult to handle using Logic BIST (LBIST) are those that produce unknown values (X values) at the outputs. Sources of unknown values include tri-stated or floating buses, uninitialized flip-flops or latches, signals that cross clock domains in circuits with multiple clock domains, and X values coming from analog or memory blocks that are embedded in the random logic circuit If an unknown value is fed into a test response evaluator

(TRE), the signature can be affected. For the most popular TRE, the Multiple Input Signature Register (MISR), a single X value invalidates the whole signature.

This problem has been attacked from two directions. First, TREs that are less vulnerable to Xes have been proposed, including X-COMPACT by Intel [3] and Convolutional Compactor by Mentor Graphics [4]. The second solution puts no restriction on the type of TRE used. The unknown values that appear at the outputs of the circuit are *masked out* by additional logic, such that only known values are fed into the TRE [5, 6]. The technique proposed here is of the second type. The *X Masking Logic* (XML) is introduced between the circuit under test and the TRE. It consists of OR gates and synthesized control logic. The first input of each OR gate is connected to an output of the circuit under test, while the second input originates from the control logic. When the control logic produces a logic-1, the output of the OR gate is forced to logic-1, and hence the response of the circuit under test is masked. The control logic is a combinational function that uses as inputs the pattern counter and bit counter, which are generally part of the LBIST test control logic for controlling the number of applied patterns and the scan shift/capture cycles.

In principle, it is possible to mask out only the unknown values in the response and to leave unchanged all the other values. However, masking exactly the unknown bits would result in high area overhead of XML. Furthermore, this is not necessary, as the vast majority of faults is detected by many different patterns. Figure 1 shows the number of stuck-
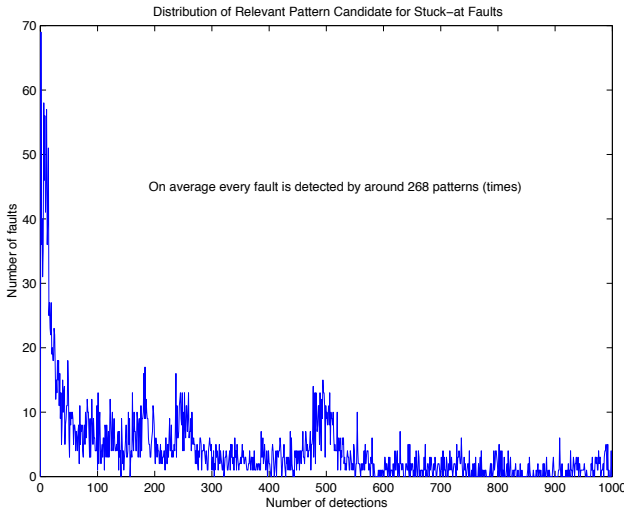
Figure 1: Number of detections for stuck-at faults of s5378 (1000 random patterns)

at fault detections per pattern for the ISCAS circuit s5378, which is also representative for other circuits. It indicates that not all unknown bits are actually required for detection. Hence, we allow also some of the known bits to be masked out, in a way that the stuck-at fault coverage is not compromised. However, the coverage of *unmodeled defects* might be affected by masking out known bits. To reduce the likelihood of coverage loss for unmodeled defects, we introduce more conservative requirements for allowing a known bit to be masked out. The requirements are based on n-detection [7, 8]. In general, introducing XML will lower the number of times a stuck-at fault is detected (even if each fault is still detected at least once). For a given parameter $n \geq 1$, the number of detections for a stuck-at fault must not decline below n due to masking. For instance, assume that a stuck-at fault is detected 5 times without masking of known bits, and let n be 3. Then, it is acceptable that the number of detections with XML drops to 4 or 3, but not below. Increasing n leads to a higher number of stuck-at fault detections (and hence hopefully to a better coverage of unmodeled defects) but also to more area overhead.

In this paper, we study the impact of masking on unmodeled defects for the proposed architecture. For this purpose, we consider resistive bridging faults (RBF) [9, 10] as surrogates of unmodeled defects. The RBF model [11, 12, 13] takes into account several non-trivial electrical properties of resistive defects, such as pattern dependency. Using the simulator from [13], we compute the RBF coverage with and without masking of known bits. Note that the information on RBF coverage is not available to the XML synthesis proce-

dure, which is guided by stuck-at detection information only. For different values of n we obtain different implementations of XML which trade off unmodeled defect coverage vs. area overhead. It turns out that the difference in RBF coverage with and without XML are not significant, and for $n \geq 5$ it practically disappears.

The most advanced X-masking solution proposed so far [6] is based on LFSR reseeding. For a given set of responses, an LFSR generates control signals for masking. Similarly to our method, the technique from [6] accepts masking of some of the known bits as long as the stuck-at fault coverage is not sacrificed. The LFSR seeds are stored on-chip. However, the issue of unmodeled defects is not dealt with in [6]. In contrast, we use n-detection information and study the trade-off between unmodeled defect coverage and the size of the logic. Furthermore, it turns out that the proposed XML requires less area than the LFSR-based architecture from [6], although we use a higher probability of X appearance.

The remainder of the paper is structured as follows: In Section 2, the X Masking Logic (XML) is introduced and its synthesis is explained. Essential information on the resistive bridging fault (RBF) model is summarized in Section 3. Experimental setup is described and the results are reported in Section 4. Section 5 concludes the paper.

2 X Elimination Logic

2.1 Problem formulation

Let the circuit under test (CUT) have p outputs, and let the test set consist of q patterns. Let the responses of the CUT be $(r_{11}, r_{12}, \ldots r_{1p})$, $(r_{21}, r_{22}, \ldots r_{2p})$, $(r_{q1}, r_{q2}, \ldots r_{qp})$, where $r_{ij} \in \{0, 1, X\}$ is the value that appears at the jth output of the CUT as a response to the ith test pattern in absence of any fault. We are looking for a function $XML : \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{B}$ such that $XML(i, j) = 1$ iff $r_{ij} = X$ (i. e. all unknown values are masked). Furthermore, some r_{ij} that are important for preserving the fault coverage (called *relevant bits*) must *not* be masked ($XML(i, j) = 0$ must hold for these bits). Selection of relevant bits will be explained in Section 2.3. For values of (i, j), for which $r_{ij} \neq X$ and which are not among the relevant bits, XML is allowed to assume either 0 or 1. This degree of freedom is utilized for minimizing the XML logic, as described next.

2.2 Implementation

We describe the implementation of XML for deterministic logic BIST (DLBIST) based on bit flipping [14, 15]. The generalization to other architectures is straight-forward.
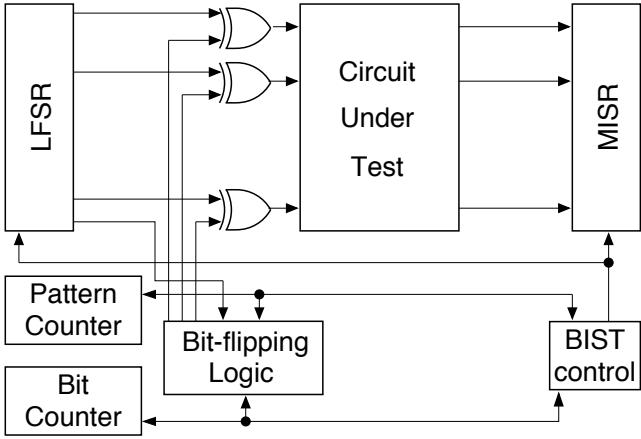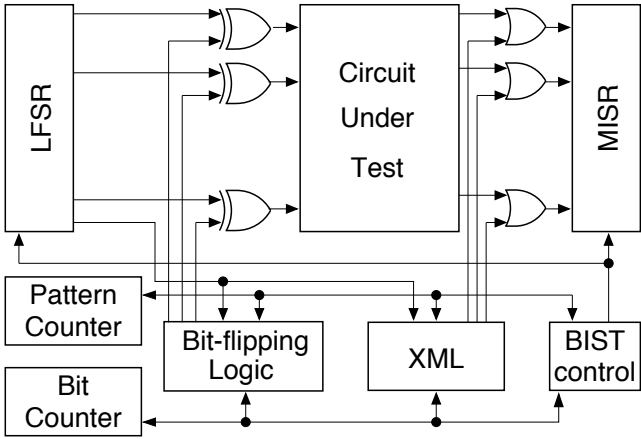
Figure 2: DLBIST without XML

Figure 3: DLBIST with XML

Figure 2 shows the DLBIST architecture without XML logic. An LFSR is used as the source of random patterns. In order to achieve the desired fault coverage, some of the bits produced by the LFSR are inverted, which is controlled by *bit-flipping logic* (BFL, referred to in [16] as bit-fixing logic). BFL is a combinational block that takes the LFSR state, the pattern number (from the pattern counter) and the bit number (from the bit counter) and selects the LFSR outputs to be inverted by driving a logic-1 at the inputs of the corresponding XOR gates. The responses of the CUT are fed into a MISR.[1]

The DLBIST architecture with XML is shown in Figure 3.

---

[1]The original DLBIST was proposed for sequential circuits with (multiple) scan chains. The method proposed here works for combinational, sequential and scan circuits. When we mention 'outputs', this means primary outputs for combinational and sequential circuits and scan_out signals for scan circuits.

Similarly to BFL, XML is a combinational logic block that has the LFSR state, the pattern number and the bit number as inputs. XML provides control signals to the OR gates between the CUT and the MISR. A bit is masked iff XML generates a logic-1 at the corresponding OR gate.

The problem to synthesize the XML can be formulated as an instance of logic synthesis with don't cares [17]. The value at jth output of the CUT when the ith test pattern is applied is uniquely determined by the triple (LFSR state, pattern number, bit number), i. e. a state of (LFSR, pattern counter, bit counter). With the notation of Section 2.1, the logic synthesis instance is composed as follows: the ON set consists of (LFSR, pattern counter, bit counter) state triples that correspond to (i, j) with $r_{ij} = X$. The OFF set includes all those triples that correspond to *relevant bits* (the description of how the relevant bits are selected follows in the next section). All other triples constitute the DC (don't care) set.

Once the ON and OFF sets are known, logic synthesis can be run. In general, compact ON and OFF sets will lead to smaller logic, because a logic synthesis tool has more degrees of freedom. While the ON set is given by the X values in the responses, there are several alternative OFF sets, depending on which bits are selected as relevant. Thus, both the number of relevant bits and the number of patterns they belong to should be minimized.

2.3 Selection of relevant bits

For the sake of simplicity, we call a value at an output j of the circuit when a test pattern i is applied a bit (so for p outputs and q patterns there are pq bits). A subset of these pq bits has to be selected as relevant bits that are excluded from masking. Remember that a triple (LFSR state, pattern number, bit number) corresponds to a bit. The triples corresponding to relevant bits are included into the OFF set of the logic synthesis problem formulated above. If more bits are selected as relevant, the number of fault detections, but also the area overhead are both growing. As an additional constraint, there is a parameter n which is defined as the minimal number of detections that must be preserved when known bits are masked out. Obviously, a higher value of n requires more bits to be selected as relevant.

The selection algorithm uses the *fault isolation table* to select relevant bits. The fault isolation table contains for each stuck-at fault f all bits for which it is detected when no XML logic is introduced (the number of such bits is denoted as N_f). A bit is said to *detect* a fault if the fault is detected at the output of the circuit for the test pattern that corresponds to the bit. For each fault f, the number of detections D_f must be guaranteed to be at least $\min\{N_f, n\}$.[2]

---

[2]Note that if n bits detecting a fault have been selected as relevant, the

3

Procedure select_rel_bits
Input: Fault isolation table FIT; parameter n
Output: Compact set RB of relevant bits
      that fulfills coverage requirements
(1) $RB := \emptyset$;
(2) **while** (FIT not empty) **begin**
(3) $f :=$ fault from FIT with
        lowest number of detections;
(4) $RB := RB \cup$
        select_bits_for_fault$(f, \min\{N_f, n\} - D_f)$;
        // Select bits that ensure sufficient D_f
(5)   **for each** fault g from FIT **begin**
(6)   Determine D_g with relevant bits selected so far;
(7)   **if** $(D_g \geq \min\{N_g, n\})$
(8)   **then** exclude g from FIT;
(9)   **end for**
(10) **end while**
(11) **return** RB;
end select_rel_bits;

Figure 4: Algorithm for selecting relevant bits

Procedure select_bits_for_fault
Input: Fault f, number M of bits to select
Output: M bits $b_1, b_2, \ldots, b_M$
(1) set_of_bits $SB := \emptyset$;
(2) **while** $(|SB| < M)$ **begin**
(3)   Select a pattern P with at least 1 bit detecting f
      (according to cost function CF1 – see text);
(4) $SB := SB \cup$ bits of P that detect f;
(5) **end while**
    // Now, SB may contain more than M bits
(6) Sort SB according to cost function CF2 (see text);
(7) **return** First M elements of SB;
end select_bits_for_fault;

Figure 5: Procedure for selecting relevant bits for a single fault (bit-based)

The algorithm select_rel_bits is shown in Figure 4. It constructs the set RB of relevant bits such that each fault f is detected by at least $\min\{N_f, n\}$ bits from RB. This is done iteratively. In each iteration, (Lines 2 – 10), a fault is picked and several bits are selected as relevant, such that the fault is detected by a sufficient number of bits (D_f = number of detections of the fault f). The selected bits might also detect other faults. This is checked in Line 6. All faults g whose number of detections D_g is greater or equal than the required number $\min\{N_g, n\}$ are excluded from the fault isolation table (Line 7). Note that the fault f from Line 3 is always

---

actual number of detections will typically be higher, because the XML could (but is not guaranteed to) leave other bits detecting this fault (but not selected as relevant) unmasked.

among these faults. The algorithm stops when the fault isolation table is empty (Line 2).

The sub-routine select_bits_for_fault (called in Line 4 of Procedure select_rel_bits) has to select $M := \min\{N_f, n\} - D_f$ relevant bits that detect the fault f (where D_f is the number of detections of f by bits selected for other faults treated before f). The pseudo-code of Procedure select_bits_for_fault is shown in Figure 5. The goal is to select bits from as few different patterns as possible. First, a suitable pattern is selected according to cost function CF1. CF1 assigns lower cost to patterns already taken for some other faults and to patterns that detect a high number of faults. Also, patterns with a low number of unknown bits are preferred by CF1, because this helps to decouple unknown bits (ON set) and relevant bits (OFF set). Bits detecting f are collected (Lines 3 and 4). If there are less than M bits, then bits from an additional pattern are added (Line 2). In the end of the first stage, there is a pool of at least M patterns, from which exactly M patterns are selected according to the cost function CF2 (Line 6). CF2 prefers a bit position that corresponds to circuit output j and pattern i such that the number of Xes for pattern i and other circuit outputs and for output j and other patterns are minimal. (Again, this is done in order to decouple the ON-set from the OFF-set). The selected bits are added to RB in Line 4 of Procedure select_rel_bits.

For comparison purposes, we implemented an alternative version of Procedure select_bits_for_fault. For a given n, it selects all bits from at least n patterns in which at least one bit detects the fault. If there are less than n such patterns then all the bits from all the patterns are selected. If the number of such patterns exceeds n, selection is made based on the cost function CF1 mentioned above. We refer to this relevant bits selection method as 'pattern-based', while we call the method outlined above 'bit-based'. Note that the pattern-based approach typically results in more bits selected as relevant as the bit-based method for the same value of n.

3   Resistive Bridging Fault Model

In this section, we provide a brief overview of the resistive bridging fault (RBF) model, which is used as a surrogate of unmodeled defects in this paper. The material here is restricted to concepts necessary for understanding the analysis in this paper; [13] gives an in-depth consideration.

The main difficulty when dealing with resistive faults is that, unlike for the non-resistive case, there is an unknown value to be taken into account, the resistance. This is because it cannot be known in advance which particle will cause the short defect corresponding to the bridge (parameters like its shape, size, conductivity, exact location on the die, evapo-
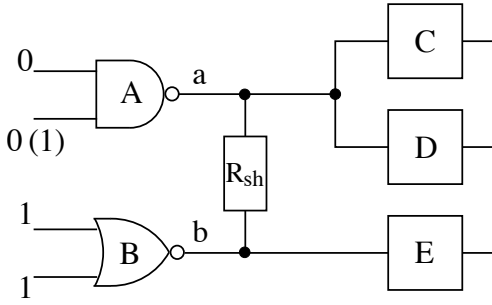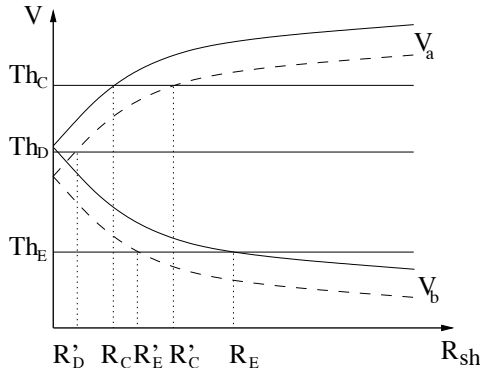
Figure 6: Example circuit

Figure 7: R_{sh}-V-diagram

ration behavior and electromigration can influence the resistance of the short defect). A short defect may be detected by a test pattern for one resistance value, and the short between the same nodes may not be detected by the same pattern for another resistance. This fundamentally changes the meaning of standard testing concepts, like redundancy, fault coverage, and so forth.

In order to handle this ambiguity, Renovell et al. [11, 12] introduced the concept of *Analogue Detectability Interval* (ADI) and probabilistic fault coverage. In the following, we will illustrate the model by means of an example.

Consider the circuit in Figure 6. The lines *a* and *b* are bridged, with *a* (*b*) being the output of a NAND2 (NOR2) gate. Let us first assume that the applied pattern is 0011. In CMOS, two *p* transistors from the pull-up network of the gate A (connected in parallel) drive the node *a*, and two *n* transistors (also in parallel) from the pull-down network of the gate B drive the node *b*. Thus, in absence of the bridge there will be a 1 on *a* and a 0 on *b*. The voltages on *a* and *b* in presence of the bridge, V_a resp. V_b, depend on the bridge resistance R_{sh}. For $R_{sh} = 0\Omega$, there will be some intermediate voltage identical for both lines. For $R_{sh} = \infty$, V_a will equal V_{DD} and V_b will equal 0V, as if the bridge were

not present. A possible voltage distribution for intermediate values of R_{sh} (those between 0Ω and ∞) is depicted by solid curves in Figure 7. The abscissa corresponds to different values of R_{sh}, the ordinate shows which voltages are assumed on the lines *a* and *b* if the bridge has such a resistance. With increasing R_{sh}, V_a and V_b diverge, with V_a approaching V_{DD} and V_b approaching 0.

The gates succeeding the bridge (gates C and D are successors of *a* and gate E is successor of *b*) will interpret these voltages as a logical value of 1 or a logical value of 0, depending on their *input threshold*. In accordance to previous works, we assume an exact-defined threshold voltage Th, which however may be different for different gate types. Thus, we rule out that some voltage is not recognized as a logical value; any voltage above Th is interpreted as the logical value of 1, and any below as the logical value of 0.[3] Moreover, we also neglect that for different manufactured ICs, the threshold of the same gate may vary.

In Figure 7, the thresholds for the gates C, D, E are shown as horizontal lines labeled by Th_C, Th_D and Th_E, respectively. Consider the gate C. Given a resistance R_{sh}, this gate will either interpret the value on *a* as logic-0 (for $R_{sh} < R_C$) or as logic-1 (for $R_{sh} > R_C$): for a bridge with low resistance, the value 0 on the line *b* has larger impact on the voltage on *a* than for a highly-resistive bridge. Hence, the RBF is detected on the output of the gate C iff $R_{sh} \in [0, R_C]$. For the gate D, the threshold Th_D is below the curve. This means that for any R_{sh} the gate D will recognize the voltage on *a* as logical value of 1. The fault is not detectable for any value of R_{sh}. For the gate E, the solid curve V_b is relevant. E interprets the voltage on *b* as faulty logical value (1) only for $R_{sh} \in [0, R_E]$.

Overall, the fault can be detected at the output of C iff $R_{sh} \in [0, R_C]$, at the output of D iff $R_{sh} \in \emptyset$ (i. e. for no value of R_{sh}) and at the output of E iff $R_{sh} \in [0, R_E]$. The fault effect is visible at one of the outputs iff $R_{sh} \in [0, R_C] \cup \emptyset \cup [0, R_E] = [0, R_E]$. The interval $[0, R_E]$ (in which the fault is detected at (at least) one output) is called *Analogue Detectability Interval (ADI)* of the pattern 0011. In contrast to fault simulation for 'classical' fault models (which determine for a fault whether it has been detected or not), RBF simulation determines for a fault and a test pattern the ADI, i. e. *for which values of bridge resistance* the fault has been detected. If the ADI is empty, then the fault has not been detected for any R_{sh}.

Now imagine that there is a logical value of 1 on the second input of the NAND gate (pattern applied is 0111).

---

[3]In their study of (non-resistive) bridging faults in an AMD design, Ma et al. [18] reported that disregarding potentially ambiguous intermediate voltages in the vicinity of the threshold had an impact on fault coverage which was below 0.007%.

Then, only one p transistor will pull up the voltage on the line a to the power supply. This results in logic-1 being driven with less strength on a. With logic-0 driven on b with the same strength as before (two parallel n transistors), the voltage characteristic for V_a and V_b in the R_{sh}-V-diagram will be described by curves situated underneath the original ones (one possibility is shown by the dashed curves). This results in new detection conditions: $R_{sh} \in [0, R'_C]$ at the output of C, $R_{sh} \in [0, R'_D]$ at the output of D (note that this interval has been empty for the pattern 0011), and $R_{sh} \in [0, R'_E]$ at the output of E. The ADI for 0111 is $R_{sh} \in [0, R'_C] \cup [0, R'_d] \cup [0, R'_E] = [0, R'_C]$. So, a RBF with $R_{sh} \in [R'_C, R_E]$ is detected by the pattern 0011 but not by 0111, although the logic values on the lines a and b in the fault-free circuit are identical for these two patterns (pattern-dependency).

C-ADI of a test set (C stands for 'covered') is defined as the union of the ADIs of individual test patterns. G-ADI (G means 'global') is the C-ADI of the exhaustive test set. Hence, C-ADI includes all the bridge resistances for which the fault has been detected by at least one test pattern, while G-ADI consists of all values of R_{sh} for which the fault is *detectable*. If C-ADI of a test set equals G-ADI, then this test set is as effective in detecting RBF as the exhaustive test set. A bridging fault with resistance not in G-ADI is redundant.

The *global fault coverage* G-FC [12, 13] is defined as

$$G\text{-FC}(f) = 100\% \cdot \left(\int_{C\text{-ADI}} \rho(r)dr \right) / \left(\int_{G\text{-ADI}} \rho(r)dr \right),$$

where $\rho(r)$ is the probability density function of the short resistance r obtained from manufacturing data. Thus, G-FC relates C-ADI to G-ADI, weighted by the likelihood of different values of R_{sh}.

4 Experimental Results

We applied the X Masking Logic (XML) synthesis approach to ISCAS 85 [19] and combinational parts of ISCAS 89 [20] circuits. As these circuits don't have tri-state buses or multiple clock domains, they don't produce X values at the outputs. consequently, we assumed a scenario when a preceding block induces unknown values at the circuit's inputs. We used the test sets for stuck-at faults generated by a commercial tool and randomly injected X values at 1% of inputs.

Logic synthesis has been performed using a BDD-based tool developed at the University of Stuttgart in cooperation with Philips. For selecting relevant bits, we employed both the bit-based and the pattern-based approach (explained in Section 2.3) with different values of n.

Procedure Monte_Carlo_Evaluation
Input: Input Pattern Set IP with Xes; set X_{Base}
   of output bits with Xes; for K XMLs, sets
   $X_1, X_2, \ldots, X_K$ of bits masked out
Output: Average RBF coverage $RBFC^{\emptyset}_{Base}$ of the
   base scenario; for K XMLs, average RBF
   coverages $RBFC^{\emptyset}_1, RBFC^{\emptyset}_2, \ldots, RBFC^{\emptyset}_K$
(1) $RBFC^{\emptyset}_{Base} := RBFC^{\emptyset}_1 := RBFC^{\emptyset}_2$
     $:= \cdots := RBFC^{\emptyset}_K := 0;$
(2)   **for** ($i := 1$ **to** 100) **begin**
(3) $IP_i := IP$ with Xes randomly assigned to 0s / 1s;
(4) $RBFC^{\emptyset}_{Base} := RBFC^{\emptyset}_{Base} + RBFSim(IP_i, X_{Base});$
(5)   **for** ($j := 1$ **to** K)
(6) $RBFC^{\emptyset}_j := RBFC^{\emptyset}_j + RBFSim(IP_i, X_j);$
(7)   **end for**
(8)   **return** $RBFC^{\emptyset}_{Base}, RBFC^{\emptyset}_1, RBFC^{\emptyset}_2, \ldots, RBFC^{\emptyset}_K$
end Monte_Carlo_Evaluation;

Figure 8: Monte-Carlo estimation of unmodeled defect coverage

4.1 Experimental setup

In order to estimate the impact of XML on the coverage of unmodeled defects, we simulated resistive bridging faults (RBF, see Section 3) in the circuits with and without XML. The fault set consisted of 10,000 randomly selected non-feedback faults, where available. For calculating the global fault coverage G-FC, we employed the density function ρ derived from one used in [21] (which is based on the data in [9] and assigns lower probability to higher values of bridge resistance).

The RBF model cannot handle unknown values at circuit inputs in a meaningful way.[4] Hence, we perform a Monte-Carlo simulation of the circuit with and without XML. The X values in the test set IP are set randomly, resulting in a test set IP_1. Resistive bridging fault simulation is performed with test set IP_1 without unknown values. The simulation is repeated 100 times with test sets $IP_1, IP_2, \ldots, IP_{100}$. (All known bits in IP are preserved in every IP_i, and the Xes are set randomly.) The average RBF coverage over $IP_1, IP_2, \ldots$ is determined then.

Fault detections at some of the output bits should not be accounted for. In absence of an XML, the output bits which are Xes don't contribute to detection. We refer to the test setting without an XML as to the *base scenario*, and we denote

---

[4]Remember that in the circuit from Figure 6, the pattern 0011 detects the fault for the bridge resistance $R_{sh} \in [0, R_E]$, where the maximal faulty effect is propagated through the gate E. The pattern 0111 detects the fault in the resistance interval $[0, R'_C]$, and the maximal faulty effect is propagated through the gate C. So, it is hard to say what the detection condition of the pattern 0x11 is.

the output bits with unknown values as X_{Base}. If an XML is present, then no detection is possible at the masked bits. Several different architectures of XML are synthesized, using bit-based and pattern-based approach and different values of n, and the XML area overhead is determined for these architectures. Let the number of these architectures be K, and let X_i be the set of bits masked by the ith XML, $1 \leq i \leq j$. (Note that $X_{Base} \in X_i$ always holds).

In order to account for masking, we modified the RBF simulator from [13] such that fault detections by some patterns at some outputs are excluded from consideration. Procedure $RBFSim(IP', X')$ simulates the test set IP' (which is not allowed to have X values) not accounting for the detections at the bits specified by X'.

The exact flow of the experiment is shown in Figure 8. For each of 100 test sets IP_i (which have been obtained from the original test set IP by randomly assigning the Xes, Line 3), we perform a total of $K + 1$ simulation runs. The first run (Line 4) determines RBF coverage $RBFC_{Base}$ for the base scenario (i. e. when the bits with unknown values X_{Base} at the outputs do not contribute to fault detection). The same is repeated for every of the K XML architectures, resulting in RBF coverages $RBFC_1, RBFC_2, \ldots, RBFC_K$ (Lines $5 - 7$). Note that $RBFC_{Base}$ is always greater or equal to any $RBFC_j$. The difference $RBFC_{Base} - RBFC_j$ is the indicator of the coverage loss for unmodeled defects due to masking out known values by the jth XML. The averaged RBF values (indicated by superscript $^\emptyset$) are the output of the experiment (Line 8).

For instance, consider the circuit from Figure 6 and the test set IP consisting of one input pattern 00XX. There is an X value at the output of the gate E. In the first run of the Monte-Carlo simulation, the Xes in the input pattern are assigned randomly, resulting in, e. g., $IP_1 = 0011$. The detection at the output of the gate E is not accounted for because of the X value on this output ($X_{Base} = \{E\}$). Hence, in the base scenario the fault is detected in the interval $[0, R_C]$ (rather than $[0, R_E]$). Suppose that there is one XML architecture ($K = 1$) that masks out the output of the gate E (because it has an unknown value) and the output of the gate C ($X_1 = \{E, C\}$). The pattern 0011 is simulated once again, but now neither detections at the output of E nor at the output of E are counted. As the fault is not detected at the output of D for any bridge resistance, it is not detected at all. The global coverage for the base scenario is $RBFC_{Base} = 100\% \cdot \left(\int_0^{R_C} \rho(r)dr \right) / \left(\int_0 R_E \rho(r)dr \right)$, and the coverage for XML is $RBFC_1 = 100\% \cdot 0 / \left(\int_0 R_E \rho(r)dr \right) = 0\%$. Then, the Xes in the pattern 00XX are again randomly assigned and G-FC is calculated for the base scenario and the XML architecture. After this has been iterated 100 times,

the value $RBFC_{Base}^\emptyset$ for the base scenario and the value $RBFC_1^\emptyset$ for XML are obtained by averaging 100 individual results, respectively.

4.2 Results

Table 1 summarizes the results for the bit-based relevant bit selection procedure, while Table 2 contains the results when the pattern-based approach has been used. The first three columns contain the circuit name, the number of patterns in the test set and the number of outputs of a circuit. The number 'Bits' of bits masked out in the base scenario (which is the number of X values at the output) and 'FC', the average global fault coverage G-FC for the base scenario, follow. The remainder of the table contains the data on XML architecture. For various values of n, the size of synthesized logic in gate equivalents ('LS'), the number of bits masked out ('Bits'), and the average global fault coverage G-FC ('FC') are reported. For three of the circuits (c3540, c6288 and c7552), G-ADI required for calculating G-FC was not available. For these circuits, G-ADI in the denominator is over-approximated by $[0, R_{\max}]$, where $R_{\max}$ is the maximal bridge resistance for which a faulty effect can be produced. Note that by over-approximating the denominator the fault coverage may be below its real value. However, the base scenario and all XML measurements are affected by this to the same extent, so comparing them is still meaningful.

From the table, it can be seen that the logic size does grow with n, however much slower than n. The RBF coverage loss is not dramatic even for $n = 1$, but for $n = 3$ the difference to the base scenario is very small for most circuits. Note that the area overhead for $n = 3$ and $n = 1$ is quite similar in most cases.

We repeated the experiment with 3% of input values (instead of 1%) set to X. In order to obtain patterns with relatively large and relatively small fractions of unknown values, we distributed Xes as follows: we defined a random variable y that assumes values between 0 and 6 (with uniform probability). For a pattern, we first assign a random value between 0 and 6 to y. Then, we set $y\%$ of the positions in the pattern to X.

Results (only for the bit-based method) are reported in Table 3. The structure of Table 3 is identical to Table 1. It can be seen that the coverage drop is quite severe for $n = 1$ for some of the circuits. In particular, for c0499 and c1355 the loss is a double-digit number. In such cases, higher values of n are required in order not to loose too much of the unmodeled defect coverage.

The results suggest that for low fractions of unknown values the XML synthesis procedure based on stuck-at fault detection is quite effective. Even if no n-detection properties

Circ	Pat	Outs	Base		$n=1$			$n=3$			$n=5$			$n=10$		
			Bits	FC	LS	Bits	FC	LS	Bits	FC	LS	Bits	FC	LS	Bits	FC
c0432	43	7	28	95.72	23	55	95.69	26	31	95.72	28	30	95.72	30	28	95.72
c0499	52	32	63	99.29	37	556	99.29	53	361	99.29	62	310	99.29	75	176	99.29
c0880	33	26	20	96.63	25	88	96.37	29	39	96.61	33	30	96.62	35	23	96.63
c1355	85	32	128	99.58	62	1277	99.22	86	894	99.39	101	669	99.44	132	521	99.51
c1908	116	25	183	99.44	122	881	99.41	168	656	99.42	187	547	99.44	219	394	99.44
c2670	65	140	160	97.89	128	2021	97.73	181	1103	97.88	199	746	97.89	243	420	97.89
c3540	126	22	205	96.94	157	588	96.91	171	383	96.94	205	322	96.94	212	278	96.94
c5315	72	123	406	99.27	228	2657	98.92	365	1485	99.13	428	1107	99.19	494	770	99.27
c6288	21	32	143	90.38	51	188	90.25	53	151	90.38	56	143	90.38	56	143	90.38
c7552	96	108	602	98.81	358	3561	98.66	468	2056	98.79	528	1633	98.80	582	1100	98.81
cs00298	25	20	8	97.48	10	37	97.45	11	18	97.48	11	12	97.48	12	11	97.48
cs00344	16	26	11	95.68	13	37	94.60	14	18	95.66	14	14	95.66	15	11	95.68
cs00400	28	27	17	98.28	26	75	98.19	27	43	98.27	28	37	98.24	30	29	98.28
cs00444	28	27	9	97.82	13	66	97.76	16	35	97.82	18	34	97.82	20	17	97.82
cs00526	55	27	23	98.35	35	180	98.29	45	96	98.34	48	77	98.35	49	54	98.35
cs00713	31	42	15	98.68	20	135	98.57	27	69	98.67	29	48	98.67	31	25	98.68
cs05378	121	228	2024	98.97	455	8292	98.84	608	5262	98.94	716	4410	98.95	863	3320	98.96

Table 2: Experimental results, pattern-based relevant bit selection (1% Xes at the inputs)

are taken into account ($n = 1$), the RBF coverage loss is small. For small $n > 1$, the coverage loss becomes negligible. But for a higher percentage of Xes, preserving n-detection is essential in maintaining the coverage of unmodeled defects.

4.3 Comparison with earlier work

Table 4 compares our results with those of [6]. [6] reports results for $p = 0.05\%$, 0.1% and $p = 0.2\%$, where p is the percentage of the output values set to X randomly. Note that we set the *input values* to X with a probability larger than 0.2% and thus end up with more Xes at the outputs, which are also correlated in a realistic way. For each p, the number S of seeds and the number P of stages in the LFSR is quoted in [6]. We compute logic size in gate equivalent according to the formula

$$GE = 6 \cdot P + 2 + S \cdot P/4$$

We count a flip-flop as 6 gate equivalents (GE): two gates for the RS circuit, 3 gates for the multiplexer, and one gate for edge handling. We assume that there are two XOR gates to implement feedback, and we count an XOR gate as one GE, which is an under-approximation. Hence, the LFSR totals $6 \cdot 6 + 2$ GE. Note that the LFSR is *not* used for random pattern generation; it is a resource present exclusively for the purpose of masking Xes. $S \cdot P$ bits have to be stored on-chip (reseeding information); we assume a PLA implementation and count one bit as 1/4 GE.

Column 2 of Table 4 ('Prop') contains the size of XML generated by our approach. We quote the results obtained using the bit-based method for relevant bit selection and $n = 1$, because it corresponds to the goal of [6] (to ensure that every stuck-at fault is detected at least once without considering unmodeled defects or n-detection). The percentage p of Xes among the output bits is shown in the third column (it corresponds to p from [6] and is obtained from the data of Table 1 as (100% · 'Base Bits') / ('Pat' · 'Outs') for the respective circuits). The remaining columns contain the values of S and P from [6] and the size of the logic in GE estimated using the formula above. It can be seen that our solution requires less area overhead despite a higher value of p.

5 Conclusions

Blocks that produce unknown values at their outputs are hard to deal with in a BIST environment, as the signature may be corrupted by the unknown values. Masking the X values at the outputs of such modules allows the use of arbitrary test response evaluators, including those vulnerable to X values. Since most faults are detected by many patterns, some known bits can also be masked without loss of stuck-at fault coverage.

We proposed a method to synthesize X Masking Logic (XML) that works for combinational, sequential, scan and partial scan circuits. It can be integrated into any BIST architecture. While previous works concentrated on sustaining the stuck-at coverage after masking, we are using more

Circ	Proposed		[6], $p = 0.05\%$			[6], $p = 0.1\%$			[6], $p = 0.2\%$		
	GE	p	S	P	GE	S	P	GE	S	P	GE
s298	9	1.6%	1	6	39.5	2	6	41	2	14	93
s344	12	2.6%	1	6	39.5	1	8	52	1	20	127
s400	25	2.2%	1	9	58.25	1	19	120.75	1	19	120.75
s444	13	1.2%	1	10	64.5	1	8	52	4	10	72
s526	33	1.5%	1	20	127	8	11	90	4	30	212
s713	18	1.2%	1	20	127	4	30	212	9	26	216.5
s5378	338	7.3%	55	19	377.25	59	27	562.25	103	30	954.5

Table 4: Result comparison to [6]

conservative metrics based on n-detection, in order to preserve the coverage of unmodeled defects. To the best of our knowledge, this is the first study that considers the effects of X-masking on unmodeled defects. We estimated the coverage of unmodeled defects using a sophisticated resistive bridging fault model, which accounts for pattern dependency and the Byzantine General Problem. By varying n, there is a trade-off between the size of the synthesized XML and the coverage of unmodeled defects. Relatively small values of n were sufficient to achieve practically the same coverage as with no masking logic, as long as the fraction of Xes to be masked was relatively low. For a higher percentage of Xes, sacrificing the n-detection properties of the test set for the sake of minimizing XML results in a significant drop in coverage of unmodeled defects. In such cases, XML architectures synthesized using a high value of n should be used.

Acknowledgment

6   References

[1] H.-J. Wunderlich. BIST for systems-on-a-chip. *INTEGRA-TION, the VLSI Jour.*, 26(12):55–78, December 1998.

[2] A. J. van der Goor. *Testing Semiconductors Memories, Theory and Practice*. ComTex Publishing, 1998.

[3] S. Mitra and K.S. Kim. X-Compact: An efficient response compaction technique for test cost reduction. In *Int'l Test Conf.*, pages 311–320, 2002.

[4] J. Rajski, C. Wang, J. Tyszer, and S.M. Reddy. Convolutional compaction of test responses. In *Int'l Test Conf.*, pages 745–754, 2003.

[5] I. Pomeranz, S. Kundu, and S.M. Reddy. On output response compression in the presence compression in the response of unknown output values. In *Design Automation Conf.*, pages 255–258, 2002.

[6] M. Naruse, I. Pomeranz, S.M. Reddy, and S. Kundu. On-chip compression of output responses with unknown values using lfsr reseeding. In *Int'l Test Conf.*, pages 1060–1068, 2003.

[7] S.C. Ma, P. Franco, and E.J. McCluskey. An experimental chip to evaluate test techniques experimental results. In *Int'l Test Conf.*, pages 663–672, 1995.

[8] S.M. Reddy, I. Pomeranz, and S. Kajihara. Compact test sets for high defect coverage. *IEEE Trans. on CAD*, 16:923–930, 1997.

[9] R. Rodríguez-Montañés, E.M.J.G. Bruls, and J. Figueras. Bridging defects resistance measurements in a CMOS process. In *Int'l Test Conf.*, pages 892–899, 1992.

[10] M. Renovell, P. Huc, and Y. Bertrand. CMOS bridge fault modeling. In *VLSI Test Symp.*, pages 392–397, 1994.

[11] M. Renovell, P. Huc, and Y. Bertrand. The concept of resistance interval: A new parametric model for resistive bridging fault. In *VLSI Test Symp.*, pages 184–189, 1995.

[12] M. Renovell, F. Azaïs, and Y. Bertrand. Detection of defects using fault model oriented test sequences. *Jour. of Electronic Testing: Theory and Applications*, 14:13–22, 1999.

[13] P. Engelke, I. Polian, M. Renovell, and B. Becker. Simulating resistive bridging and stuck-at faults. In *Int'l Test Conf.*, pages 1051–1059, 2003.

[14] H.-J. Wunderlich and G. Kiefer. Bit-flipping BIST. In *Int'l Conf. on CAD*, pages 337–343, 1996.

[15] G. Kiefer, H. Vranken, E.J. Marinissen, and H.-J. Wunderlich. Application of deterministic logic bist on industrial circuits. In *Int'l Test Conf.*, pages 105–114, 2000.

[16] N.A. Touba and E.J. McCluskey. Altering a pseudo-random bit sequence for scan based bist. In *Int'l Test Conf.*, pages 649–658, 1996.

[17] R.K. Brayton, R. Rudell, A.L. Sangiovanni-Vincentelli, and A.R. Wang. MIS: A multiple - level logic optimization system. *IEEE Trans. on Comp.*, 6(6):1062–1081, 1987.

[18] S. Ma, I. Shaik, and R. Scott-Fetherston. A comparison of bridging fault simulation methods. In *Int'l Test Conf.*, pages 587–595, 1999.

[19] F. Brglez and H. Fujiwara. A neutral netlist of 10 combinational circuits and a target translator in fortran. In *Int'l Symp. Circ. and Systems, Special Sess. on ATPG and Fault Simulation*, pages 663–698, 1985.

[20] F. Brglez, D. Bryan, and K. Kozminski. Combinational profiles of sequential benchmark circuits. In *Int'l Symp. Circ. and Systems*, pages 1929–1934, 1989.

[21] C. Lee and D. M. H. Walker. PROBE: A PPSFP simulator for resistive bridging faults. In *VLSI Test Symp.*, pages 105–110, 2000.

Circ	Pat	Outs	Base		n = 1			n = 3			n = 5			n = 10			n = 15			n = 20		
			Bits	FC	LS	Bits	FC	LS	Bits	FC	LS	Bits	FC	LS	Bits	FC	LS	Bits	FC	LS	Bits	FC
c0432	43	7	28	95.72	21	66	95.58	24	50	95.68	25	41	95.71	26	31	95.72	28	29	95.72	28	29	95.72
c0499	52	32	63	99.29	35	586	99.29	51	347	99.29	58	293	99.29	73	165	99.29	85	117	99.29	99	84	99.29
c0880	33	26	20	96.63	23	110	96.35	30	36	96.62	32	30	96.62	34	23	96.63	35	23	96.63	36	21	96.63
c1355	85	32	128	99.58	60	1371	99.26	83	979	99.40	97	832	99.40	129	535	99.47	142	431	99.52	166	341	99.53
c1908	116	25	183	99.44	110	1215	99.26	139	920	99.40	157	646	99.43	197	531	99.44	225	440	99.44	232	344	99.44
c2670	65	140	160	97.89	112	2690	97.67	156	1360	97.85	185	1021	97.88	226	682	97.89	247	490	97.89	262	344	97.89
c3540	126	22	205	96.94	118	1080	96.64	149	580	96.92	169	487	96.92	191	403	96.93	214	287	96.94	225	269	96.94
c5315	72	123	406	99.27	206	3430	98.69	330	2021	99.15	369	1502	99.10	451	1103	99.24	467	1096	99.23	496	979	99.20
c6288	21	32	143	90.38	49	211	90.23	52	159	90.36	54	144	90.38	56	143	90.38	56	143	90.38	56	143	90.38
c7552	96	108	602	98.81	336	4359	98.52	437	3044	98.77	493	2225	98.80	550	1713	98.80	595	1283	98.80	610	1040	98.80
cs00298	25	20	8	97.48	9	36	97.45	11	13	97.48	11	15	97.48	12	10	97.48	13	8	97.48	13	8	97.48
cs00344	16	26	11	95.68	12	65	95.34	13	22	95.64	14	20	95.68	15	11	95.68	15	11	95.68	15	11	95.68
cs00400	28	27	17	98.28	25	117	97.91	27	52	98.27	27	45	98.27	28	29	98.28	32	22	98.28	33	17	98.28
cs00444	28	27	9	97.82	13	97	97.71	15	50	97.82	16	40	97.82	20	16	97.82	21	14	97.82	22	10	97.82
cs00526	55	27	23	98.35	33	281	98.09	41	166	98.32	44	97	98.34	48	58	98.35	50	43	98.35	51	37	98.35
cs00713	31	42	15	98.68	18	154	98.58	24	84	98.66	27	60	98.68	31	24	98.68	31	23	98.68	32	16	98.68
cs05378	121	228	2024	98.97	338	10383	98.63	470	7810	98.87	555	6556	98.91	703	5170	98.96	801	4448	98.97	883	4032	98.97

Table 1: Experimental results, bit-based relevant bit selection (1% Xes at the inputs)

Circ	Pat	Outs	Base		n = 1			n = 3			n = 5			n = 10			n = 15			n = 20		
			Bits	FC	LS	Bits	FC	LS	Bits	FC	LS	Bits	FC	LS	Bits	FC	LS	Bits	FC	LS	Bits	FC
c0432	43	7	61	94.01	33	122	93.20	41	88	93.86	43	78	94.00	45	70	94.01	46	66	94.01	47	63	94.01
c0499	52	32	314	94.53	75	1222	43.27	111	1023	85.57	123	912	88.20	183	703	88.59	231	525	94.10	275	405	94.52
c0880	33	26	63	96.34	51	240	95.63	64	110	96.25	67	117	96.28	81	81	96.34	83	71	96.34	83	66	96.34
c1355	85	32	542	95.10	98	2245	70.56	156	2110	73.89	203	1859	76.22	258	1581	81.80	304	1287	88.06	411	1047	89.24
c1908	116	25	360	99.10	136	1757	98.61	208	1261	99.01	249	1132	99.08	300	888	99.10	333	776	99.10	356	674	99.10
c2670	65	140	456	93.98	207	4477	92.24	286	3084	93.80	358	2176	93.92	440	1453	93.97	507	1205	93.98	541	774	93.98
c3540	126	22	433	96.18	219	1503	95.06	273	1120	95.80	314	930	96.08	381	735	96.15	414	640	96.17	419	564	96.17
c5315	72	123	925	98.81	305	5090	96.74	502	3820	98.14	623	3153	98.22	762	2238	98.51	846	1976	98.67	875	1678	98.73
c6288	21	32	326	88.25	72	427	85.74	77	344	88.12	82	333	88.25	89	327	88.25	89	326	88.25	89	326	88.25
c7552	96	108	1574	97.57	532	6801	95.14	792	5125	97.28	880	4866	97.45	1004	3984	97.50	1071	3325	97.53	1152	2409	97.53
cs00298	25	20	29	96.94	29	133	96.35	33	76	96.90	38	45	96.93	43	39	96.94	44	33	96.94	46	30	96.94
cs00344	16	26	20	95.69	21	71	94.75	26	33	95.67	29	27	95.69	30	20	95.69	30	20	95.69	30	20	95.69
cs00400	28	27	49	97.58	38	253	95.97	53	155	97.36	57	123	97.46	65	60	97.58	70	54	97.58	74	50	97.58
cs00444	28	27	41	97.01	36	191	95.60	43	89	96.88	48	86	96.87	53	49	97.00	54	45	97.01	54	42	97.01
cs00526	55	27	94	98.19	77	493	97.72	99	300	98.08	104	235	98.14	109	218	98.17	132	146	98.19	134	129	98.19
cs00713	31	42	87	98.31	57	447	97.74	84	273	98.19	99	174	98.30	105	126	98.31	107	123	98.31	111	99	98.31
cs05378	121	228	2926	98.54	518	16112	97.64	792	13659	98.35	997	11497	98.38	1299	8700	98.50	1484	7034	98.53	1658	6331	98.53

Table 3: Experimental results, bit-based relevant bit selection (3% Xes at the inputs)