Achieving 100% Cell-Aware Coverage by Design*

Zeye (Dexter) Liu, Ben Niewenhuis, Soumya Mittal and R. D. (Shawn) Blanton

Department of Electrical and Computer Engineering
Carnegie Mellon University, Pittsburgh, PA, 15213
http://www.ece.cmu.edu/∼actl/

***Abstract*—A comprehensive investigation of new integrated circuit design and fabrication technologies is crucial for yielding reliable parts. Prior work proposed a novel logic characterization vehicle called the Carnegie Mellon Logic Characterization Vehicle (CM-LCV), and an implementation flow that ensures a test chip to be product-like with near optimal testability and diagnosability. This work describes an enhanced implementation methodology for CM-LCV that not only guarantees 100% intra-cell defect testability for all standard cells but also reflects the user-specified design characteristics. Experiments comparing intra-cell defect testability between a CM-LCV and various benchmark circuits demonstrate the efficacy of this approach. Specifically, the CM-LCV achieves 92.4% overall input pattern fault coverage and 100% cell-aware fault coverage using an optimal, minimal test set.**

I. Introduction

Fast yield ramping requires a design and fabrication methodology that enables technology learning to be accomplished using actual silicon structures and product-like sub-circuits. There are many types of test structures, ranging from the passive (via arrays, comb drives, etc.) to large, active circuits (e.g., ring oscillators, SRAM, etc.). A logic characterization vehicle (LCV) is intended to have all the logic characteristics of an actual customer product. Design and fabrication of LCVs are now common practice for fabless companies (e.g., NVidia, QUALCOMM and Broadcom), foundries (e.g., TSMC and Global Foundries), and integrated device manufacturers (e.g., Intel and Samsung). Traditional product-like test chips are created by adapting existing product designs, and are not ideal for collecting fabrication feedback due to non-optimal testability and diagnosability [1], [2]. Poor testability and diagnosability is especially problematic for yield learning, where every missed or improperly diagnosed failure can lead to more test chips being fabricated, more time and effort spent on yield learning, and ultimately the possibility of lower product yield.

Given the drawbacks of traditional test chip approaches, the Carnegie Mellon Logic Characterization Vehicle (CM-LCV) was introduced [2]–[4]. The CM-LCV is a two-dimensional array of functional unit blocks (FUBs). This architecture leverages C-testability theory [5] to guarantee controllability and observability for all faults within the two-dimensional array with a minimal test set. Prior work describes a methodology for measuring the standard-cell characteristics of design and a design flow that reflects those characteristics into a CM-LCV [2], [3]. Work in [4] describes a BIST scheme that applies all input patterns to each FUB in the CM-LCV and achieves 100% single-stuck line (SSL) [6] fault detection for a reference CM-LCV design with an 86.9% reduction in test time.

Although prior work addresses various aspects of the CM-LCV (FUB testability, diagnosability, etc.), investigation into the testability of intra-cell defects inside the standard cells has not been explicitly considered. Past research has shown that the quality of test patterns can be improved significantly by explicitly targeting intra-cell defects. Specifically, works in [7]–[9] examined the impact of defects within the cell, which was later commercialized as cell-aware testing [10], [11]. Work in [1], [12] proposes a more general input-pattern (IP) fault model that ensures the detection of every irredundant intra-cell defect if applied at the standard-cell level.

Building on our prior work, we describe a matrix-based formulation that not only captures the standard-cell characteristics of the CM-LCV but also its IP fault and cell-aware testability. Using this formulation, various FUB implementations are identified that ensure high IP fault testability and 100% cell-aware testability using a optimally-provable, minimal test set. Several experiments support the claims made concerning fault coverage for intra-cell defects and test-set size.

The rest of this paper describes the details of the proposed approach. Specifically, Section 2 discusses the IP fault model and the testability of various benchmark designs which are meant to represent a conventional LCV. Section 3 describes the implementation methodology, focusing mainly on the design and implementation of a CM-LCV with near optimal testability for intra-cell defects. Section 4 describes the experiments used to evaluate this implementation methodology. Finally, conclusions and future work are provided in Section 5.

II. Intra-cell defect detection

Defects inevitably occur during the integrated circuit fabrication process. To detect these defects, fault models have been proposed and used for generating and quantifying the completeness or quality of test patterns. Many fault models and test metrics are described in the literature (e.g., single-stuck line, transition, bridge, N-detect, etc.) [6]. However, a major assumption is that defects can be modeled as a modification of the structure or behavior of the cell interconnect. Prior work has demonstrated that the SSL fault model is insufficient to

*This work is sponsored by the National Science Foundation under contract no. 1527606 and the Semiconductor Research Corporation under contract no. 2237.001.

Inputs (AB)	Output (Z)	
	Expected	Faulty
00	0	1
01	0	1
10	0	1
11	1	0

Table 1: An example illustrating the IP fault model for a 2-input AND gate [12].

guarantee that all detectable intra-cell defects are tested [10], [13]–[15]. Detailed analysis has shown that the majority of test escapes can be attributed to insufficient detection of intra-cell defects [16]. To be specific, among one million tested parts, an average of 24.8 devices/mm^2 escaped the SSL test pattern set but were caught by the intra-cell test pattern set. Further examination determined that one missing input condition for a multiplexer cell was the root cause for a significant fraction of the test escapes. Because the SSL fault model implicitly assumes that all defects map to single lines being permanently stuck-at 0 or 1, the ATPG tool is not required to cover all of the input conditions needed to detect intra-cell defects.

Being aware of the drawbacks of conventional fault models, the work in [10], [11] focus on intra-cell defects. By analyzing the physical layout of standard cells, the cell-aware test approach identifies input conditions for each standard cell that, if tested, would result in improved intra-cell defect testability. The IP fault model described in [12] is a general approach for capturing the changes in circuit module functionality, where modules can be arbitrarily defined as gates, sub-circuits, etc.

Table 1 shows the truth table for a 2-input AND gate with inputs A and B, and output Z. The rows of the truth table form the input patterns of the gate and are exhaustively enumerated ip_0=00, ip_1=01, ip_2=10 and ip_3=11. Every input pattern has an expected response and an example faulty response, listed in the second and third columns, respectively. An IP fault changes the module's response to an input pattern from the expected value to some faulty value. For example, if an intra-cell defect leads to an input-output functionality change for the first input pattern, it is denoted as f_0=(00→0/1), where the first value after the arrow indicates the expected value, and the second value is the faulty value. It is possible that different intra-cell defects could exhibit the same input-output functionality. Therefore, one IP fault can model more than one intra-cell defect. Since a detectable intra-cell defect must change a standard cell's functionality, the set of all possible IP faults for a standard cell subsumes all detectable functional intra-cell defects[1]. In this case, the test set of all possible IP faults for the standard cell is equivalent to what is required by the gate-exhaustive approach [1]. Work in [1] demonstrates the effectiveness of this test set in detecting defective chips compared to test sets generated using conventional fault models.

Both the cell-aware and IP fault models can be used to generate test patterns that detect intra-cell defects. Note that the cell-aware test approach uses layout-aware fault induction to determine which cell input patterns expose faulty behavior at the cell boundary; thus the cell-aware fault model is an intelligently selected subset of all IP faults. Fault induction via a cell-aware like methodology is not suitable for test chips as all possible defect mechanisms may not be known for a new technology node. Thus, for test chips, the full IP fault model is preferred because it guarantees that all detectable intra-cell defects are tested.

[1] It should be noted that some defects that affect cell behavior (i.e., timing and function) may require a sequence of input patterns to be applied.

Circuit	No. of redundant IP faults	No. of IP faults	IP fault cover-age	No. of redundant IP fault classes	
				Circuit	Library
s13207	1231	8164	84.9%	3	1489
s15850	1821	10958	83.4%	3	1481
s35932	1728	19038	90.9%	2	1630
s38417	2499	33724	92.6%	48	1440
s38584	4921	42434	88.4%	11	1439
B17	39931	142388	71.9%	13	1281
B20	16565	53262	68.9%	16	1330
B21	16761	54336	69.1%	16	1334
B22	23827	79174	69.9%	16	1308
B18	100048	365182	72.5%	16	1262
B19	201555	732262	72.4%	14	1258
[10]	N/A	N/A	66%	N/A	N/A

Table 2: IP fault coverage analysis for various circuit examples.

Given the ability of the IP fault model to cover all detectable intra-cell defects, IP fault coverage analysis is performed on various benchmark circuits to examine their intra-cell testability. Five ISCAS89 [17] and six ITC99 [18] benchmark circuits are synthesized using a commercial standard-cell library that contains 154 different standard cells, and a commercial ATPG tool [19] is used to target all cell-level IP faults. Table 2 provides detailed IP fault coverage information for the five ISCAS89 and the six ITC99 benchmark circuits. The last row in Table 2 shows the average IP fault coverage for ten industrial designs reported in [10]. The IP fault coverages range from 66% to 92.6%.

However, the last two columns of Table 2 indicate that there are certain IP fault classes that are completely redundant. Here, an IP fault class is defined as F=(c, f_k), where c is a standard cell and f_k is an IP fault. For example, consider the IP fault class consisting of IP fault f_0=(00→0/1) for a two input AND cell with single drive strength (denoted AND2×1). Each instance of the AND2×1 cell in a design has an IP fault f_0 that is a member of this specific IP fault class. If all of these individual IP faults are redundant for this design, then the fault class F=(AND2×1, f_0) is redundant. The column labeled "circuit" (col. 5) of Table 2 shows the number of redundant IP fault classes when considering only the standard cells used in each circuit (and not the entire standard-cell library). Note that redundant fault classes are a significant disadvantage in test chips because any systematic defect that is only detectable by one or more redundant fault classes is guaranteed to remain undiscovered.

Furthermore, since intra-cell defects are closely related to the cell layout [10], [11], standard cells that have the same logical functionality but different drive strengths (and thus

physical layouts) are susceptible to different intra-cell defects. Therefore it is imperative that every IP fault class for every standard cell within a test chip be testable for comprehensive yield learning. There are 154 standard cells and 1,692 total IP fault classes for the standard-cell library used in this table. The last column (col. 6) shows the number of undetected IP fault classes in each circuit with respect to all 1,692 IP fault classes for the standard-cell library. Table 2 reveals that conventional benchmark designs not only fail to detect all IP fault classes for the standard cells that they use, they also fail to cover a significant number of IP fault classes in the overall standard-cell library.

It is critical that a product-like test chip guarantees maximal IP fault coverage on all standard cells used in the design. As previously described, the CM-LCV has optimal testability on the FUB level. Since each FUB in the CM-LCV implements a bijective function, any change in the FUB's truth table is guaranteed to propagate to the boundary of an arbitrarily large CM-LCV. This means that, if an intra-cell defect is detected at the FUB boundary, it is guaranteed to be detected at the primary output of the CM-LCV (assuming only one activated defect). In other words, the CM-LCV approach leverages C-testability theory to guarantee that each FUB is exhaustively tested. Thus the problem of intra-cell defect testability can be simplified to finding FUB implementations that, when exhaustively tested, detect all IP faults for the standard cells that are used.

III. Implementation

Prior work describes an implementation flow for imposing standard-cell usage characteristics onto the CM-LCV [2], [3]. This flow begins with a standard-cell library and culminates in a FUB template, that is, a collection of FUB implementations that together reflect either a design family or a specified standard-cell usage. This current work extends this flow to ensure detection of all IP fault classes for the given standard-cell library.

Fig. 1 illustrates this design flow. It begins with a standard-cell library for a given technology. In the first step the logical functions are extracted from the standard-cell library S to create a logic library L. Note that a typical standard-cell library contains numerous logic functions implemented as cells with varying drive strengths; for example, there may exist a two-input NAND with nominal drive strength (denoted typically as NAND2×1) and a two-input NAND with twice the drive strength (e.g. NAND2×2). Both of these cells would be considered equivalent to the two-input NAND logic function in this extraction process.

The logic library is then used to repeatedly synthesize a given FUB function, resulting in a collection of FUB implementations. Each FUB implementation produced is characterized, resulting in two vectors **C** and **R**:

- The vector **C** tracks how many times each logic function is used in a given FUB. Thus vector component **C**$[l]$ is an integer count of how many times logic function l is used in a FUB implementation.

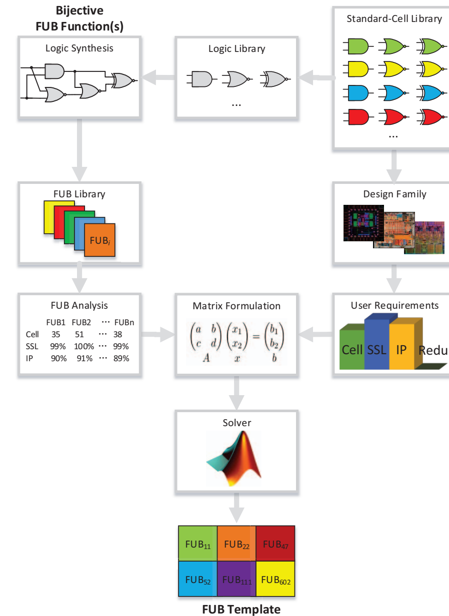


Figure 1: The CM-LCV implementation flow.

- The vector **R** represents the number of redundant IP faults for each IP fault class. Thus **R**$[(l,\ f_k)]$ is the number of redundant IP faults (f_k) for logic function l in the given FUB implementation.

These two vectors are used to construct a boolean vector **FC** representing the IP fault class coverage for a given FUB. IP fault class coverage can be constructed using either the *strong* or the *weak* criterion:

- *Strong* - Each IP fault class is considered covered (**FC**$[(l,\ f_k)]$=1) if and only if its logic function is used in the FUB (**C**$[l] > 0$) and there are no redundant faults for that fault class (**R**$[(l,\ f_k)] = 0$). Otherwise **FC**$[(l,\ f_k)]$=0.
- *Weak* - Each IP fault class is considered covered (**FC**$[(l,\ f_k)]$=1) if and only if its logic function is used in the FUB (**C**$[l] > 0$) and at least one of its member IP faults in the FUB are detected (**C**$[l] >$ **R**$[(l,\ f_k)]$). Otherwise **FC**$[(l,\ f_k)]$=0.

The logic function counts **C**, redundant faults **R**, and the IP fault class coverages **FC**$_{strong}$ and **FC**$_{weak}$ are collected for all of the FUB implementations to create a matrix formulation used by various solvers. The flow of Fig. 1 shows that targeted characteristics can be derived from former production chips and/or specified by a user. The fundamental requirement for the FUB template is that each fault class is irredundant. In other words, if an intra-cell defect is only detectable by a given IP fault class, there must be a chance to detect the defect by ensuring the class is not redundant within the LCV. Based on this prerequisite, this method also provides two additional enhanced features to impose the user specified design characteristics onto the LCV. The first feature is high overall IP fault coverage. In the test chip, fault coverage can be understood as the possibility of detecting an actual defect. For example, if the IP fault coverage for a test chip is 90%,

there is a 90% possibility to detect that defect if it randomly occurred within any standard cell. It is clear that a high IP fault coverage is desirable for a test chip; otherwise it is necessary to increase the volume of chips manufactured to achieve the same probability of detecting a defect. Equation 3.1 shows the mathematical expression for achieving high IP fault coverage:

$$\begin{aligned} &Minimize\ (j(Rx) + (\frac{Rx}{TFx})) \\ &Subject\ to \quad FCx \geq d \\ &\qquad x \geq 0 \end{aligned} \tag{3.1}$$

where: R is a matrix that has the number of redundant IP faults for each FUB implementation; TF is a matrix that contains the number of total IP faults in each FUB implementation; FC is a constraint matrix that represents the FUB IP fault testability by using either the *strong* or *weak* criterion; d is a matrix that guarantees each IP fault class is testable; j is a parameter for tuning the cost function; x is the solution that represents the number of FUB implementations selected for the FUB template.

Besides high fault coverage, it is important to mimic the logic-design characteristics for a product-like test chip. Since the physical layout of standard cells used in a product design and a FUB template are identical and because physical layout influences intra-cell defects [10], [11], this work considers standard-cell usage as the most important characteristic for ensuring intra-cell defect detection. Equation 3.2 gives the mathematical expression for creating the product-like test chip with similar standard-cell usage:

$$\begin{aligned} &Minimize\ (j(Rx) + \|Cx - b\|^2) \\ &Subject\ to \quad Dx \geq T \\ &\qquad x \geq 0 \end{aligned} \tag{3.2}$$

where: C is a matrix that has the logic function counts for each FUB implementation; b is a matrix that represents the target product or user-specified design characteristics; D is a matrix that extracts, for each IP fault class, the number of gates that has a corresponding testable IP fault instance within the FUB implementation; T is a constraint matrix that ensures that a certain number of IP faults for a given class are testable within the FUB template; j, R and x have the same meaning used for Equation 3.1.

The first term in the cost functions of Equations 3.1 and 3.2 is identical, and drives the solver to minimize the total number of redundant faults within the FUB template. The second term of Equations 3.1 and 3.2 is what makes the two formulations different. Specifically, the second term of Equation 3.1 drives the increase in the IP fault coverage, while the second term of Equation 3.2 minimizes the mismatch in cell usage between the FUB template and the targeted cell usage distribution. The constraint condition in Equations 3.1 and 3.2 guarantees that every standard-cell function will be instantiated and every IP fault class is testable in the FUB template produced. Either formulation can be used to produce a FUB template; which formulation to use simply depends on the user's desired objective for the CM-LCV.

IV. Experiment

This work continues to use the 6-input bijective function used in prior work [3]. A logic library is extracted from the commercial standard-cell library used to implement the various benchmark circuits in Section 2. This logic library was used with the implementation flow described in Section 3 to create multiple FUB templates. Note that, while the FUB implementations produced by this flow are defined using the logic functions from the logic library (and not the standard cells), it is straightforward to convert them by swapping out the logic functions for standard cells with the corresponding logic function.

Table 3 provides a detailed comparison of SSL and IP fault coverage for two FUB templates, families of ISCAS89 and ITC99 benchmark designs, and industrial designs taken from [10]. Here, the five ISCAS89 and six ITC99 benchmark circuits (see Table 2) are collectively evaluated in the first and second rows, respectively. The last column of Table 3 lists the number and percentage of redundant IP fault classes with respect to the total 1,692 IP fault classes for the standard-cell library. The reported number of redundant IP fault classes decreases for the ISCAS89 and ITC99 design families compared to Table 2 as different benchmark circuits cover different IP fault classes. In the third row, only the average IP fault coverage of ten industrial designs is reported in [10]. The last two rows report on two FUB templates created by solving Equation 3.1 using the "BARON" computational system [20] with both the *strong* and *weak* constraint criterion. The SSL fault coverages of the two FUB templates virtually match the two families of benchmark designs. Note that, because synthesis implicitly optimizes SSL fault testability, the FUB template achieves this high SSL fault coverage without an explicit SSL fault coverage constraint. More importantly, the overall IP fault coverages of the two FUB templates are higher than both the benchmark families and the industrial designs, and, furthermore, the FUB templates do not contain any redundant IP fault classes.

In addition to achieving high IP fault coverage, the implementation flow described in Section 3 also provides the ability to reflect design characteristics. Here, as in [3], the frequency of standard cells in the design is the primary characteristic targeted. The IP fault coverages reported in Table 3 for FUB templates resulting from either the *strong* or *weak* criterion do not differ significantly. Using the *weak* criterion increases the flexibility for building the product-like test chip, since more FUB cells satisfy the *weak* criterion. Fig. 2a compares the logic function usage between the ISCAS89 family of designs and the FUB templates that are created using the implementation flow with a convex solver [21]. Three different FUB templates result from trading off IP fault coverage and cell usage mismatch using the parameter j in Equation 3.2. The *x*-axis of Fig. 2a shows the eighteen different logic functions in the library. The *y*-axis of Fig. 2a is the usage for each function, that is, the total number of cells in the design that implement the corresponding function. Table 4 shows IP

Circuit	SSL fault				IP fault				No. of redundant IP fault classes
	No. of redundant faults	Total no. of faults	Coverage	No. of tests	No. of redundant faults	Total no. of faults	Coverage	No. of tests	
ISCAS89	676	128618	99.5%	584	12200	114318	89.3%	1336	1374 (81%)
ITC99	11794	1222173	99.0%	5026	398687	1426604	72.0%	10583	1192 (70%)
[10]	N/A	N/A	N/A	N/A	N/A	N/A	66%	N/A	N/A
FUB$_{strong}$	490	77060	99.4%	64	4390	57870	92.4%	64	0 (0%)
FUB$_{weak}$	470	56750	99.2%	64	3720	42950	91.3%	64	0 (0%)

Table 3: SSL and IP fault coverage comparison for the FUB template against various benchmark and industrial designs.

fault coverage and usage mismatch for three different FUB templates. Usage mismatch is the discrepancy between the logic-function counts measured from the design family and the FUB template, and is defined as $\frac{\sum|\Delta_i|}{K}$, where $|\Delta_i|$ is the absolute difference between the number of cells of function type i in the design(s) and the FUB template, and K is the total number of instances of all function types in the design(s). The last column in Table 4 shows the value of j for creating each FUB template. By varying j from 1,000 to 3,500, it is possible to achieve a 10% increase in IP fault coverage at the cost of a 7% increase in cell usage mismatch.

In addition, the user can specify not only the logic function usage but also the minimum number of detected IP faults for each IP fault class. In Fig. 2b a function-usage distribution is arbitrarily specified, as represented by the first set of bars. Given that for the ideal case an IP fault class should be detected as many times as its corresponding cell function is used in the design, the same distribution is used as the specification for the minimum number of detections for *each* IP fault class. The remaining bars represent the cell usage distributions for the two FUB templates that result from the convex [21] and the BARON [20] solvers. Fig. 2b indicates that the cell usage for the two FUB histograms is higher than the user-specified cell usage. This is due to the fact that the current FUB implementations cannot achieve 100% IP fault coverage for some IP fault classes. Thus the solver includes additional FUBs to meet the requirement on the number of detectable IP faults for each IP fault class. Table 5 summarizes IP fault analysis and cell-usage mismatch for the two different FUB templates of Fig. 2b. Compared to the result generated by BARON, the convex solver creates a FUB template with higher IP fault coverage, lower cell-usage mismatch, and a 10× reduction in run time.

As previously described in Section 2, cell-aware tests are generated by examining the cell layout. However, physical layout information is not included in the commercial standard-cell library used for the previous experiments. Thus a second standard-cell library with available physical layout information is adopted for this experiment.

Circuit	No. of redun-dant IP faults	No. of IP faults	IP fault coverage	Cell-usage mis-match	Value of j
FUB$_{i89a}$	8484	72254	88.3%	22.0%	3500
FUB$_{i89b}$	15054	89486	83.3%	18.5%	1500
FUB$_{i89c}$	21179	98750	78.5%	14.8%	1000

Table 4: Fault analysis and cell mismatch for FUB templates when targeting cell usage.

FUB	No. of re-dundant IP faults	No. of IP faults	IP fault coverage	Cell-usage mismatch	Run time
Convex	47432	450178	89.5%	31.4%	372.5s
BARON	61523	465030	86.8%	33.9%	4000s

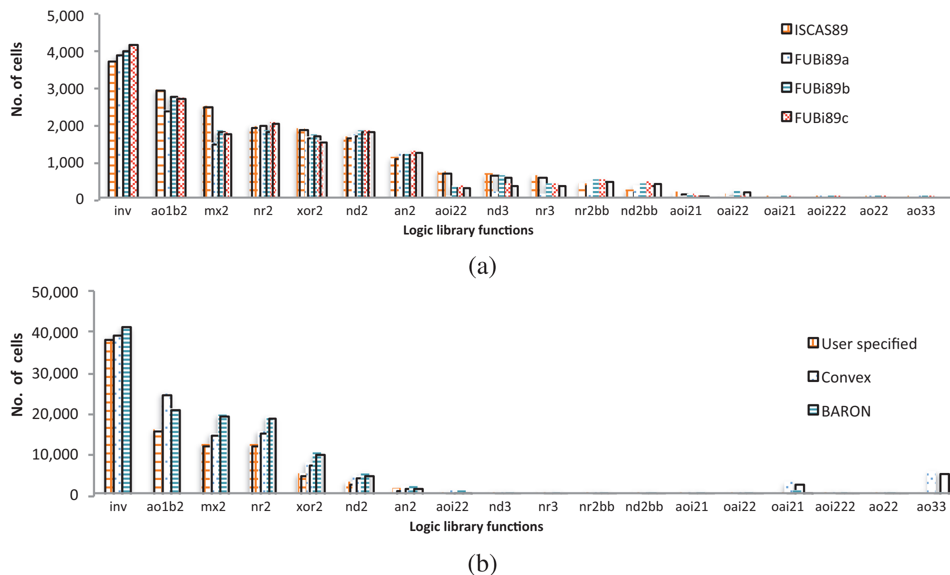
Table 5: Fault analysis and cell mismatch for FUB templates when targeting IP fault coverage.

SLIDER, a defect-simulation framework described in [22], is used to extract intra-cell defect behavior. In the following experiment, only the IP test patterns required to detect the defects induced by SLIDER for the corresponding standard cells are considered in the IP fault coverage analysis.

Table 6 shows the intra-cell fault coverage comparison between the ISCAS89 and ITC99 design families and three FUB templates generated while targeting only those IP faults identified by SLIDER as necessary for defect detection. To be specific, 560 out of a total 1,692 IP fault classes are required to detect all of the SLIDER-injected defects. The first two rows show the SLIDER intra-cell fault coverage for the ISCAS89 and ITC99 design families is increased since some input patterns for standard cells are not required for defect detection. The remaining parts of Table 6 shows three alternative versions of FUB templates with 100% SLIDER-induced fault coverage. Specifically, the FUB$_{slider_a}$ is created using the complete FUB cell library with 42,802 FUB cells, the FUB$_{slider_b}$ uses the same FUB library excluding those used in FUB$_{slider_a}$, while FUB$_{slider_c}$ is created using the same FUB library excluding those used by both FUB$_{slider_a}$ and FUB$_{slider_b}$. The 100% SLIDER intra-cell fault coverage of the FUB template indicates that not only is every SLIDER-induced fault for the corresponding standard cell testable, but also no redundant fault is introduced into the FUB template. It should be noted that while the number of SLIDER-induced faults for the template is much smaller than the benchmark families, this is not at all a concern since the FUB template can

Circuit	No. of redundant SLIDER-induced faults	No. of SLIDER-induced faults	SLIDER fault coverage
ISCAS89	2079	60835	96.6%
ITC99	54677	548642	90.0%
FUB$_{slider_a}$	0	6230	100%
FUB$_{slider_b}$	0	3950	100%
FUB$_{slider_c}$	0	5700	100%

Table 6: Comparison of intra-cell defect coverage between three FUB templates and two design families.



(a)

(b)

Figure 2: Standard-cell histograms for (a) the ISCAS89 design family and (b) the user-specified characteristics compared with two FUB templates created using the implementation flow of Section 3.

be repeated an arbitrary number of times while maintaining 100% coverage.

V. Conclusion

This work presents an implementation methodology for the CM-LCV that targets maximal IP fault coverage for all cells in a standard-cell library. High IP fault coverage is an appealing objective in test chip design because full coverage of IP faults ensures the detection of all detectable intra-cell defects. The CM-LCV implementation methodology results in a *FUB template*, which is a collection of FUB implementations that can be replicated and seamlessly interconnected to create an LCV array of any desired size. FUB templates created using this methodology achieve up to 92.4% overall IP fault coverage with a small, constant-size test set. Furthermore, these FUB templates guarantee that all IP faults are tested at least once for every cell in the standard-cell library. Meanwhile, the proposed implementation flow is able to reflect design characteristics provided by a user with minimal cell usage mismatch. Finally, and importantly, the FUB templates created by the described implementation flow can also easily achieve 100% coverage of induced intra-cell (i.e., cell-aware) defects. However, this work mainly focuses on static intra-cell defects; any timing related defects (e.g., transition fault, path-delay fault, etc.) and the testability of sequential elements (e.g., flip-flops) will be addressed in our future work.

References

[1] K.Y. Cho, S. Mitra, and E. McCluskey, “Gate Exhaustive Testing,” *IEEE International Test Conference*, Nov. 2005.

[2] R. D. Blanton, B. Niewenhuis, and C. Taylor, “Logic Characterization Vehicle Design for Maximal Information Extraction for Yield Learning,” *IEEE International Test Conference*, Oct. 2014.

[3] R. D. Blanton, B. Niewenhuis, and Z. Liu, “Design Reflection for Optimal Test-Chip Implementation,” *IEEE International Test Conference*, Oct. 2015.

[4] B. Niewenhuis and R. D. Blanton, “Efficient Built-in Self Test of Regular Logic Characterization Vehicles,” *IEEE VLSI Test Symposium*, 2015.

[5] A. D. Friedman, “Easily Testable Iterative Systems,” *IEEE Transactions on Computers*, vol. C-22, no. 12, pp. 1061–1064, Dec. 1973.

[6] M. Bushnell and V. Agrawal, *Essentials of Electronic Testing For Digital, Memory and Mixed-signal VLSI Circuits*. Springer, 2000.

[7] J. Khare, W. Maly, and N. Tiday, “Fault Characterization of Standard Cell Libraries Using Inductive Contamination Analysis (ICA),” *IEEE VLSI Test Symposium*, pp. 405–413, May 1996.

[8] A.Jee and J. F. Ferguson, “Carafe: An Inductive Fault Analysis Tool for CMOS VLSI Circuits,” *VLSI Test Symposium*, April 1993.

[9] B. Chess, F. J. Ferguson, and T. Larrabee, “Testing CMOS Logic Gates for Realistic Shorts,” *IEEE International Test Conference*, Oct 1994.

[10] F.Hapke, J.Schloeffel, H.Hashempour, and S.Eichenberger, “Gate-Exhaustive and Cell-aware Pattern Sets for Industrial Designs,” *IEEE VLSI Design, Automation and Test*, April 2011.

[11] F. Hapke, et al., “Cell-Aware Test,” *IEEE Transactions on Computer-Aided Design of Integrated Circuit and Systems*, vol. 33, no. 9, pp. 1396–1409, Sep. 2014.

[12] R. D. Blanton and J. Hayes, “Properties of The Input Pattern Fault Model,” *IEEE International Conference on Computer Design*, Oct. 1997.

[13] S.C.Ma, P.Franco, and E. McCluskey, “An Experimental Chip To Evaluate Test Techniques Experiment Results,” *IEEE International Test Conference*, Oct. 1995.

[14] R. D. Blanton, K. N. Dwarakanath, and A. B. Shah, “Analyzing The Effectiveness of Multiple-detect Test Sets,” *IEEE International Test Conference*, Oct. 2003.

[15] Benware, et al., “Impact of Multiple-detect Test Patterns on Product Quality,” *IEEE International Test Conference*, Oct. 2003.

[16] S. Eichenberger, et al., “Towards a World Without Test Escapes,” *IEEE International Test Conference*, Oct. 2008.

[17] F. Brglez, D. Bryan, and K. Kozminski, “Combinational Profiles of Sequential Benchmark Circuits,” *International Symposium on Circuits and Systems*, 1989.

[18] S. Davidson, “ITC’99 Benchmark Circuits-Preliminary Results,,” *IEEE International Test Conference*, pp. 1125–1130, 1999.

[19] “Encounter Test and Diagnostics,” Product Version 15.11, Cadence Design Systems Inc., San Jose, CA, 2015.

[20] Sahinidis, V. Nikolaos, and M. Tawarmalani, “BARON: Global Optimization of Mixed-integer Nonlinear Programs.” User’s manual, 2005.

[21] M. Grant and S. Boyd, “CVX:Matlab Software for Disciplined Convex Programming,” Version 2.0, beta.http://cvxr.com/cvx, Sept. 2013.

[22] W. C. Tam and R. D. Blanton, “SLIDER: Simulation of Layout-Injected Defects for Electrical Responses,” *IEEE Transactions on Computer-Aided Design*, vol. 31, no. 6, pp. 918–929, June 2012.