Exploiting the correlation with traditional fault models to speed-up cell-aware fault simulation

Exploiting the correlation with traditional fault models to speed-up cell-aware fault simulation

Reza Khoshzaban*, Iacopo Guglielminetti†, Michelangelo Grosso†, Matteo Sonza Reorda*, Riccardo Cantoro*
* Politecnico di Torino, DAUIN — Turin, Italy † STMicroelectronics — Turin, Italy

***Abstract*—A fault list analysis methodology is proposed to determine how many Cell-Aware Test (CAT) defects can be detected by test patterns generated targeting the other fault models, including stuck-at faults (SAFs), transition-delay faults (TDFs), and small-delay defects (SDDs). Our analysis reveals that a proper ordering in fault simulation can accelerate the CAT fault simulation process. We evaluated our approach on a RISC-V core, synthesized using an industrial technology library. We demonstrated an innovative method to optimize CAT fault simulation by means of preliminary static fault list analysis, resulting in fault simulation runtime reduction of up to 80% for static and 35% for dynamic CAT faults.**

I. Introduction

Ensuring high-quality in integrated circuit testing is vital, with traditional fault models—stuck-at faults (SAFs), transition-delay faults (TDFs), and small-delay defects (SDDs)—commonly used by fault simulation and ATPG tools. These models target faults at standard cell boundaries but are limited in detecting intra-cell defects, prompting the adoption of Cell-Aware Testing (CAT) [1]–[6]. CAT improves defect coverage by modeling electrical-level defects but incurs significant fault simulation and ATPG runtime due to larger fault lists and complex detection conditions [7]–[9].

Efforts to optimize CAT include characterization flow improvements, such as machine learning-based model prediction [10], [11], exclusion of undetectable fault/pattern pairs [12], and graph-based detection analysis [13]. These affect only model generation where our methodology instead targets the fault simulation phase.

Other work includes reusing traditional test patterns [14], applying software-based self-test to detect static CAT faults [15], [16], and optimizing ATPG through matrix manipulation [17]. While the first two reuse existing patterns and the last improves ATPG, none reduces fault simulation runtime. [18] proposes multi-conditional fault collapsing for CAT, and [19] reduces small-delay fault lists via gross-delay tests. These approaches reduce fault counts, but our method uses traditional fault models to lower CAT fault simulation runtime. Until now, the correlation between CAT faults and traditional models remained underexplored for simulation optimization. This paper explores that correlation. We show that a CAT fault can be mapped to correlated faults of varying complexity. Through static fault list analysis and prioritizing faster-to-simulate models, we reduce fault simulation computational complexity. We call this flow Fault Analysis for Cell-aware Test (FACT), which serves as a pre/post-processing layer. Experimental results on a RISC-V processor show that up to 85% of static CAT and 40% of dynamic CAT faults correlate with traditional fault models, enabling relative runtime reduction.

TABLE I: CAT static and dynamic detection matrices for full-adder cell

Static Detection Matrix										
A	B	CI	CO	S	$D1$	$D2$	$D3$	$D4$	$D5$	...
0	0	1	0	1	0	0	1	2	3	...
0	1	0	0	1	3	1	1	2	1	...
1	0	1	1	0	0	2	0	2	0	...
1	1	0	1	0	0	1	0	0	3	...
Dynamic Detection Matrix										
A	B	CI	CO	S	$D12$	$D22$	$D32$	$D42$	$D52$	...
0	0	R	0	R	0	2	0	0	3	...
0	R	0	0	R	0	0	0	1	1	...
R	0	0	0	R	2	0	1	0	0	...
0	0	F	0	F	1	0	0	0	3	...

II. Background

A. Cell-Aware Test

Test patterns for traditional fault models (e.g., SAFs, TDFs, SDDs) model defects at standard cell boundaries. In contrast, CAT targets intra-cell defects characterized at the electrical level, which traditional models may miss.

CAT requires electrical-level characterization of standard cells, producing defect matrices that define excitation and observation conditions. This starts with extracting structural and electrical data from transistor-level descriptions. Defect models are built from physical layout analysis, capturing mechanisms like resistive bridges, parasitic effects, and shorts/opens. Simulations emulate these defects to assess their impact on logic behavior. Static faults are identified via voltage deviations under input stimuli; if none are seen in one cycle, dynamic analysis checks transitions. Equivalent defects are grouped to reduce redundancy.

Depending on the EDA vendor, CAT supports a user-defined fault model (UDFM) or Cell Test Model (CTM). We use CTMs, which include static and dynamic defect detection matrices. Defects are represented by *prime* entries corresponding to defect classes. Table I shows examples for a full-adder cell with CO and S outputs. Static CAT faults D1–D5 are linked to input conditions; values '1' and '2' indicate detection via CO or S, and '3' via either port. The matrix supports one-timeframe (1tf) tests. Dynamic Detection Matrix is for two-timeframe (2tf) tests, with 'R' and 'F' denoting rising/falling transitions and '0' and '1' steady values.

B. Fault Collapsing

Fault collapsing reduces the number of faults during ATPG and fault simulation by grouping similar faults into *equivalence classes* or *dominance hierarchies* [20]–[22].

1) Fault Equivalence: Two faults f_1 and f_2 are equivalent if the test vector sets detecting them are identical ($T_1 = T_2$). They can then be collapsed into a single representative fault.

2) Fault Dominance: Fault f_2 dominates f_1 if any test vector for f_1 also detects f_2 ($T_2 \subseteq T_1$), making f_1 redundant.

3) Similarity between faults: Let T_i and T_j be input vectors detecting faults f_i and f_j with $T_i, T_j \neq \emptyset$. The similarity level is $SL_{i,j} = \frac{|T_i \cap T_j|}{|T_i|}$. If $T_i = T_j$, then $SL_{i,j} = 1$; if $T_i \subset T_j$, then $SL_{i,j} = 1$, $SL_{j,i} < 1$; otherwise, both < 1.

We use equivalence, dominance, and similarity to optimize fault lists. We refer to these concepts collectively as *correlation*.

III. The FACT Flow

The initial CAT fault list is derived from individual cell models, which include static and dynamic detection matrices. Our approach links CAT faults to multiple fault models, allowing prioritization of faster-to-simulate faults. We consider SAFs, TDFs, and SDDs in our experiments, but the method can be extended to other fault models.

To determine the correlation between CAT and traditional fault models, we analyze detection matrices (e.g., Table IIa). Each CAT fault is evaluated based on conditions that trigger traditional faults. *Correlation* occurs when all input conditions detecting the traditional fault also detect the CAT fault and *conditional correlation* occurs when only a subset does.

For example, in Table IIc, static CAT fault D1 correlates with SAF A_0 as both are detected by $[A = 1, B = 1]$. D1 conditionally correlates with Z_1 since the vectors detecting Z_1 do not cover all the conditions to detect D1 (e.g., $[A = 1, B = 1]$). D1 does not correlate with B_1, which is only detected by $[A = 1, B = 0]$. The FACT approach is based first on identifying correlations, and then performing a fault simulation/pruning phase.

A. Correlation identification from Detection Matrices

Detection matrices map faults to logic states (0 or 1). To establish correlations between CAT faults and traditional fault models, we first identify vectors capable of detecting a traditional fault on a given input signal by analyzing the logic gate's truth table.

As an example, Table IIb presents correlation conditions for a 2-input AND gate with inputs (A, B) and output (Z). The SAF column set lists all possible SAF conditions using a detection matrix where '0' indicates the SAF is not detected, and '1' means it is detected by the corresponding vector.

Once correlation conditions per vector are identified, we analyze how each static CAT fault aligns with SAF conditions. Table IIc shows this for fault D1, where '0' means no correlation and '1' indicates an equivalent detection condition. Asterisks (*) mark conditions where the CAT fault is only conditionally detected. Green-highlighted rows indicate CAT fault detection conditions. If all '1's of a SAF align with green cells, the fault is correlated (e.g., A_0 with D1). Conditional correlation occurs when a SAF has detection conditions not matching D1. For example, Z_1 has such a condition not aligned with D1. Grayed-out columns represent SAFs with no correlation to the CAT fault (no '1's in green-highlighted rows) and are excluded from the process.

TABLE II: Static CAT fault correlation identification for 2-input AND gate

(a) Static CAT detection matrix from the CTM

Vector			Static CAT			
A	B	Z	D1	D2	D3	...
0	0	0	1	0	1	...
0	1	0	1	1	1	...
1	0	0	0	0	1	...
1	1	1	1	1	0	...

(b) Static condition correlation matrix

Vector			SAF					
A	B	Z	A_0	A_1	B_0	B_1	Z_0	Z_1
0	0	0	0	0	0	0	0	1
0	1	0	0	1	0	0	0	1
1	0	0	0	0	0	1	0	1
1	1	1	1	0	1	0	1	0

(c) Condition correlations for D1

CAT fault	SAF					
D1	A_0	A_1	B_0	B_1	Z_0	Z_1*
1	0	0	0	0	0	1
1	0	1	0	0	0	1
0	0	0	0	1	0	1
1	1	0	1	0	1	0

To identify the correlation of dynamic CAT faults with TDFs and SDDs, we employ a methodology similar to that used for static CAT faults. From the CTMs, we use detection matrices to identify correlations between detection conditions of dynamic CAT faults and TDFs/SDDs, and use timing information to distinguish between them, as the matrices alone do not indicate whether timing constraints apply.

B. Fault Analysis and fault simulation/pruning

Algorithm 1 CAT fault Detection Process

Input: CAT fault list D, fault lists $F_1, F_2, \ldots, F_N$
foreach *CAT fault* $d \in D$ **do**
  mark d as not detected;
foreach *fault list* F_i **do**
  perform **fault simulation** on F_i;
  $R_i \leftarrow$ simulation results for F_i
  foreach *CAT fault* $d \in D$ **do**
    if d *is correlated to any detected fault in* R_i **then**
      mark d as detected;

The proposed FACT flow is summarized in Algorithm 1, which uses a loop-based approach to simulate fault lists based on runtime efficiency. The input consists of D, the list of CAT faults from the CTM, encompassing both static and dynamic faults, and $F_1, F_2, \ldots, F_N$, the ordered fault lists for different models (e.g., SAFs, TDFs, SDDs). These lists are processed

sequentially in a prioritized manner, starting with faster-to-simulate models. i is the index of the current fault list being simulated.

The algorithm begins by marking all CAT faults $d \in D$ as undetected. It then iterates over each fault list F_i, performing fault simulation on the entire list. The simulation results are mapped to the CAT fault list by identifying which CAT faults are correlated to detected faults. Any correlated CAT fault is then marked as detected. This process continues through all fault lists until all simulation results have been evaluated. The fault model priority is determined by runtime efficiency, with simpler models (e.g., SAFs) processed first and more complex models (e.g., SDDs) later.

C. Implementation of the FACT Flow

The implementation of the *FACT flow* consists of two primary phases: *CTM analysis* and *fault list processing and simulation*. The first phase extracts CAT fault information from CTMs using logic operations and table manipulations, while the second optimizes fault simulation by prioritizing efficiently detectable faults.

The first stage processes CTMs to identify prime faults and their fault conditions using the methodology in Section III-A, generating an initial fault list categorized by fault model.

In this paper we consider SAFs and *1-timeframe Conditional SAFs (1CS)* for static CAT, and TDFs, SDDs, Conditional TDFs (CTDFs), Conditional SDDs (CSDDs), and *2-timeframe Conditional SAFs (2CSs)* for dynamic CATs as an example where more fault models can be introduced if they are deemed necessary. *Conditional fault models* require specific test vectors for detection. A fault f_c is *conditional* if detectable only by a subset of test vectors $V_c \subset V_d$ where V_c represents the test vectors detecting f_c and V_d represents the full test vector set. For example, if fault Z_0 is only observable with vector v_1, it is classified as conditional Z_0.

In the second phase, FACT generates fault lists per model using the set of correlations and maps faults to circuit locations. Next, it performs fault simulation in a structured order, starting with models that are less expensive to simulate. As an example, SAFs and then 1CS for static faults and TDFs, SDDs, and then their conditional variants (CTDF, CSDD, 2CS) for dynamic faults.

As each simulation step completes, the tool maps detected faults to correlated CAT faults, refines fault lists by removing covered faults, and updates coverage data. The process iterates through remaining models until all have been simulated or maximum coverage is reached.

After simulating the final fault list for static and dynamic CAT faults, any remaining CAT faults are marked as *Not Detected*, meaning further effort is needed to detect them. The tool then calculates cumulative fault coverage at each step and reports the final CAT fault coverage.

IV. Experimental Results

To evaluate the effectiveness of the proposed FACT flow, we applied it to the RISC-V CV32E40P processor—a 32-bit, in-order core with a four-stage pipeline, maintained by OpenHW

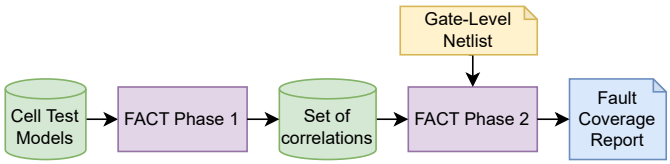

Fig. 1: FACT Flow overview

[23]. The circuit was synthesized using a proprietary 130-nm HCMOS technology optimized for power by STMicroelectronics. We used Synopsys Design Compiler for logic synthesis and Synopsys CMGen for CAT characterization. Experiments were run on an Intel Xeon CPU E5-2680 v3. We are assuming that all the faults are detected at each step of the process to assess the maximum optimization obtainable by the proposed approach.

The computational effort of applying this methodology is negligible. FACT uses lightweight Python scripts between test stages, adding virtually no overhead. Overall execution time is dominated by fault simulation runtime.

Tables IIIa and IIIb present the results of applying the FACT flow to the CV32E40P core. The column *Fault Model* refers to the fault type at each simulation step. *cum FC%* represents cumulative fault coverage before the step. The *Detected Faults* column shows the number of faults detected per step. *Active Faults* indicate faults still active before the step. Active faults are those not included in previous fault lists or undetected. The remaining columns show step-by-step fault list reduction. After each step, the number of surviving faults per fault model is recorded, and cumulative coverage is updated.

Table IIIa shows that 84% of static CAT faults are mapped to SAFs, which can be covered by SAF tests alone.

To demonstrate the effectiveness of the flow, Table IV reports the computational effort for static CAT fault simulation on the CV32E40P core with and without FACT. Without FACT, all static CAT faults are considered as 1CS, resulting in high simulation time. Using FACT, 84% of static CAT faults are correlated to SAFs. This reduces the effort for simulating 1CS faults by nearly 84% and lowers overall runtime by about 80%. The optimization stems from correlating each single SAF with multiple CAT faults that are mapped to multiple 1CS faults, which reduces the number of faults that need to be simulated.

A similar trend is observed for dynamic CAT faults. As shown in Table IIIb, about 40% of dynamic CAT faults map to TDF and SDD.

While FACT flow optimizes CAT fault detection by leveraging traditional fault models, CAT remains indispensable for identifying defects that are not adequately covered by classical models. Certain electrical-level defects, particularly those involving parametric variations, bridging effects, or resistive opens, may not be effectively detected through traditional fault models alone. Rather than replacing CAT, the proposed methodology enhances its efficiency by reducing redundant fault simulations.

TABLE III: FACT results on CV32E40P assuming all faults are detected at each step. In the first step, active faults indicate total counts; in subsequent steps, they show values before execution. Detected Faults indicates the number of faults detected after each step.

(a) Static CAT faults on CV32E40P

Fault Model	cum FC (%)	Detected Faults	Active Faults	Active SAF	Active 1CS
Step 1 (SAF)	0.0	4,514,628	4,713,283	4,514,628	1,186,811
Step 2 (1CS)	84.1	1,038,656	749,372	0	1,038,656
	100.0	0	0	0	0

(b) Dynamic CAT faults on CV32E40P

Fault Model	cum FC (%)	Detected Faults	Active Faults	Active TDF	Active SDD	Active 2CS	Active CTDF	Active CSDD
Step 1 (TDF)	0.0	1,513,153	3,344,732	1,513,153	197,864	632,504	2,274,285	1,513,040
Step 2 (SDD)	35.0	160,313	2,172,871	0	160,313	361,832	1,834,267	1,167,404
Step 3 (2CS)	39.5	361,714	2,023,248	0	0	361,714	1,829,175	1,125,044
Step 4 (CTDF)	49.8	1,606,474	1,677,981	0	0	0	1,606,474	1,009,023
Step 5 (CSDD)	85.5	606,102	485,196	0	0	0	0	606,102
	100.0	0	0	0	0	0	0	0

TABLE IV: Fault simulation runtime (times in hours and minutes).

Methodology	**FACT**	**SAF**	**1CS**	**Static CAT**
Without FACT	N/A	N/A	60h 10m	60h 10m
With FACT	0h 2m	1h 30m	10h 41m	12h 13m
Reduction (%)	N/A	N/A	82.3%	79.7%

V. Conclusions

We presented a methodology to reduce CAT fault simulation runtime by exploiting correlations between CAT defects and traditional fault models (SAFs, TDFs, SDDs). By prioritizing faster-to-simulate faults and mapping them to CAT defects, our approach significantly reduces simulation effort while maintaining coverage.

FACT achieves up to 80% reduction for static and 35% for dynamic CAT fault simulation. While CAT remains essential for detecting certain electrical-level defects, our method enhances its efficiency by eliminating redundant simulations.

Future work includes expanding to other fault models, such as bridging faults, and refining model prioritization strategies, as well as application of the FACT methodology for ATPG.

Acknowledgment

This publication is part of the project PNRR-NGEU which has received funding from the MUR – DR 117/2023.

References

[1] F. Hapke *et al.*, "Defect-oriented cell-aware atpg and fault simulation for industrial cell libraries and designs," in *International Test Conference*, 2009.

[2] F. Hapke *et al.*, "Cell-aware test," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 33, no. 9, 2014.

[3] F. Hapke *et al.*, "Defect-oriented cell-internal testing," in *IEEE International Test Conference*, 2010.

[4] F. Hapke *et al.*, "Cell-aware production test results from a 32-nm notebook processor," in *IEEE International Test Conference*, 2012.

[5] F. Hapke *et al.*, "Cell-aware analysis for small-delay effects and production test results from different fault models," in *IEEE International Test Conference*, 2011.

[6] A. D. Singh, "Cell aware and stuck-open tests," in *21th IEEE European Test Symposium (ETS)*, 2016.

[7] F. Hapke *et al.*, "Introduction to the defect-oriented cell-aware test methodology for significant reduction of dppm rates," in *17th IEEE European Test Symposium (ETS)*, 2012.

[8] W. Howell *et al.*, "Dppm reduction methods and new defect oriented test methods applied to advanced finfet technologies," in *IEEE International Test Conference (ITC)*, 2018.

[9] F. Hapke *et al.*, "Defect-oriented test: Effectiveness in high volume manufacturing," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 40, no. 3, 2021.

[10] P. d'Hondt *et al.*, "A learning-based methodology for accelerating cell-aware model generation," in *Design, Automation Test in Europe Conference Exhibition (DATE)*, 2021.

[11] P. D'Hondt *et al.*, "A comprehensive learning-based flow for cell-aware model generation," in *IEEE International Test Conference (ITC)*, 2022.

[12] F. Lorenzelli *et al.*, "Speeding up cell-aware library characterization by preceding simulation with structural analysis," in *IEEE European Test Symposium (ETS)*, 2021.

[13] G. Mongelli *et al.*, "A graph-based methodology for speeding up cell-aware model generation," in *IEEE 30th International Symposium on On-Line Testing and Robust System Design (IOLTS)*, 2024.

[14] N. Mirabella *et al.*, "A comparative overview of atpg flows targeting traditional and cell-aware fault models," in *29th IEEE International Conference on Electronics, Circuits and Systems (ICECS)*, 2022.

[15] R. Cantoro *et al.*, "Assessing the effectiveness of software-based self-test programs for static cell-aware test," in *IEEE European Test Symposium (ETS)*, 2024.

[16] P. Bernardi *et al.*, "Recent trends and perspectives on defect-oriented testing," in *2022 IEEE 28th International Symposium on On-Line Testing and Robust System Design (IOLTS)*, 2022.

[17] Z. Gao *et al.*, "Optimization of cell-aware atpg results by manipulating library cells' defect detection matrices," in *IEEE International Test Conference in Asia (ITC-Asia)*, 2019.

[18] R. Krenz-Bååth *et al.*, "Fault collapsing of multi-conditional faults," in *IEEE 16th International Symposium on Design and Diagnostics of Electronic Circuits Systems (DDECS)*, 2013.

[19] A. Jain *et al.*, "Optimized timing aware atpg for at-speed test of cell internal faults," in *IEEE 8th International Test Conference India (ITC India)*, 2024.

[20] I. Pomeranz *et al.*, "On fault equivalence, fault dominance, and incompletely specified test sets," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 24, no. 8, 2005.

[21] I. Pomeranz *et al.*, "Equivalence, dominance, and similarity relations between fault pairs and a fault pair collapsing process for fault diagnosis," *IEEE Transactions on Computers*, vol. 59, no. 2, 2010.

[22] I. Pomeranz *et al.*, "Level of similarity: A metric for fault collapsing," in *Proceedings Design, Automation and Test in Europe Conference and Exhibition*, vol. 1, 2004.

[23] M. Gautschi *et al.*, *Near-Threshold RISC-V Core With DSP Extensions for Scalable IoT Endpoint Devices*, https://github.com/openhwgroup/cv32e40p, 2017.