Received 1 August 2023, accepted 13 September 2023, date of publication 18 September 2023, date of current version 2 October 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3316511

RESEARCH ARTICLE

A Toolchain to Quantify Burn-In Stress Effectiveness on Large Automotive System-on-Chips

FRANCESCO ANGIONE[1], (Graduate Student Member, IEEE), DAVIDE APPELLO[2], PAOLO BERNARDI[1], (Senior Member, IEEE), ANDREA CALABRESE[1], (Member, IEEE), STEFANO QUER[1], (Member, IEEE), MATTEO SONZA REORDA[1], (Fellow, IEEE), VINCENZO TANCORRE[2], AND ROBERTO UGIOLI[2]

[1]Department of Control and Computer Engineering (DAUIN), Politecnico di Torino, 10129 Turin, Italy
[2]STMicroelectronics, 20864 Agrate Brianza, Italy

Corresponding author: Francesco Angione (francesco.angione@polito.it)

ABSTRACT Complexity and performance of Automotive System-on-Chips have exponentially grown in the last decade, also according to technology advancements. Unfortunately, this trend directly and profoundly impacts modern Electronic Design Automation tools, which must handle very large amounts of logic gates. The consequence is an exponential increase in computation times, potentially leading to significant production delays. In the context of Burn-In, to reduce the computing time, the stress specification is often relaxed due to the difficulty of grading extensive pattern sets, and it may result in the insurgence of unstressed circuit zones. As a matter of fact, current Electronic Design Automation software tools provide limited capabilities to effectively quantify stress effectiveness, investigate per-pattern set coverage loss, and compute layout-aware stress metrics. This article proposes a toolchain to overcome the limitations mentioned above. We propose a complete software flow to evaluate Burn-In stress patterns through standard toggle coverage and activity effectively. Together with these standard metrics, this article illustrates how to complement traditional measurement with layout-aware toggle coverage metrics. By exploiting parallel programming paradigms and machine learning algorithms, the proposed toolchain drastically reduces computation time for evaluating traditional stress metrics, and it offers new analysis metrics to test engineers conceiving the Burn-in stress patterns. In addition, the toolchain offers some commodities to superimpose the generated stress from different patterns and visualize it over the SoC layout through a heatmap, providing great benefits to test engineers in charge of composing Burn-In recipes. We validated our toolchain on two industrial devices from STMicroelectronics belonging to the SPC58 and SPC56 families, which include around 20 million and 2.7 million gates, respectively.

INDEX TERMS Automotive SoCs burn-in, simulation analysis, parallel applications, density aware metrics, toggle activity, stress-test evaluation, EDA tool, stress coverage loss troubleshooting, stress plot, burn-in stress effectiveness analysis.

I. INTRODUCTION

State-of-the-art System-on-Chips (SoCs) integrate several technologies, include heterogeneous devices, and contain many embedded memories. They include a massive number of logic gates. Because the probability of a physical defect in a SoC is related to its complexity and size in logic gates [1], testing modern devices is always more challenging.

In the automotive scenario, devices should also respect the safety requirements dictated by the standard ISO-26262 [2], every defect potentially causing a harmful failure should be avoided. Therefore, for automotive SoCs, the standard

The associate editor coordinating the review of this manuscript and approving it for publication was Thomas Canhao Xu.

manufacturing test flow, usually encompassing only wafer and package tests, is enhanced with a Burn-In stress step, followed by a final package test, and eventually a System-Level-Test phase [3], [4], [5]. In this article we essentially concentrate on the Burn-In phase. Still, we show that our analysis flow can also provide useful feedback about final test procedures and SLT.

Burn-In (BI) is a stress phase designed to remove the infant mortality of SoCs [6] manufactured with some weakness, such as thinner metal oxide or metallization. The BI phase provides both external and internal stress to a device. External overexertion (such as thermal stress) is generated by a climatic chamber or at the socket level. Its main objective is to age the device material [7]. Internal stress (such as electrical stress) is produced by scan-based approaches [8], Built-In Self-Test (BIST) modules [9], or functional test programs [10], [11]. Its main target is to force device gates to produce high internal activity and possibly exacerbate the insurgence of latent defects not screened by the wafer and packaged test procedures, which could be later captured by the successive test steps. The final test is performed by automotive chip manufacturers just before the market. Some companies execute this step by re-running all previous tests after the BI stress phase; others also perform an additional step called System-Level Tests (SLTs) to cope with system interaction issues [12], [13], [14]. The SLT procedures usually use advanced functional (often holistic) test strategies, such as booting an operating system [13].

Nowadays, grading the quality of BI procedures is getting extremely important [3], [15]. Unfortunately, grading activities are affected by the size and complexity of modern SoCs. Moreover, modern devices present new defect types [16], [17] requiring enhanced coverage metrics not fully supported by current Electronic Design Automation (EDA) tools. For example, the most frequently used metric, i.e., the so-called toggle coverage [18], represents the number of gates that make at least one transition. Nonetheless, there is an increasing interest in also evaluating the average number of times and how uniformly a signal is stressed [19]. To perform these advanced tasks, it is necessary to simulate the stress patterns and post-process the resulting simulation dump [20]; this approach may become prohibitively expensive for large devices. Furthermore, the SoC layouts should be considered as gate density may vary from area to area, affecting the accuracy of the stress measures [18], [21].

In summary, reaching an excellent stress quality during BI for automotive SoCs is a significant achievement [16]. From an industrial perspective, a trade-off between stress metrics and computation time is often required. With the increasing complexity of SoCs, the specifications on the stress metrics are often relaxed to keep the computation time reasonable. Moreover, to speed up the development time of BI stress patterns, approaches from other manufacturing areas are reused, e.g., functional programs used for verification purposes are also used for BI [19]. However, they are not primarily intended to stress the SoC effectively and may not exacerbate all latent defects. Grading the stress abilities of these procedures borrowed from other steps in device production is often left to designers' expertise due to the lack of proper measurement tools.

A problem related to the complexity and the deep submicron era is that different activation methods exist for effectively stressing a gate [9]. Moreover, the gates distribution is not uniform, which makes it critical to distribute the stress over the SoC layout uniformly. With the current state-of-the-art stress metrics, a more dense area has the same weight in terms of stress coverage compared to a less dense area. Consequently, the applied stress per area in the SoC can vary depending on the layout region and may affect the insurgence of latent defects by delaying their appearance. All the issues mentioned above could lead to ineffective BI stress for automotive devices. Consequently, only a subset of all the latent defects are captured during the Final and System Level Tests. Therefore, latent defects may arise during the SoC operational life, creating in-field returns for manufacturers and dangerous situations for customers.

This article illustrates the components of a toolchain designed to conceive the efficiency of computing BI stress and better quantify its effectiveness. The toolchain encompasses several software analysis tools orchestrated to evaluate stress patterns more flexibly, efficiently, and effectively. Moreover, state-of-the-art EDA tools do not consider advanced stress metrics and lack of layout-aware capabilities. Overall, the toolchain includes tools to:

- Process the files generated during the logic simulation phase to quickly and effectively measure the stress ability of the stress pattern.
- Compute many kinds of stress metrics, ranging from the common toggle activity to a layout-aware toggle activity.
- Evaluate a single or multiple stress pattern sets to highlight their individual, collective, and progressive coverage abilities.
- Grade different stress patterns, highlighting their different abilities and improving them by using coverage [22] and specific area-related information.
- Display the stress activity over the plot of the chip surface to increase the awareness and confidence of the designers in the testing process.

We resort to thread-based parallel computation strategies and machine-learning techniques to perform these steps in a reasonable time. The main result of the above efforts is a toolchain capable of evaluating stress metrics on extremely complex automotive SoCs and considering a variety of pattern types, spanning from ATPG to Functional patterns, and able to grading BIST executions.

As stated by Polian et al. [3], grading SLT applications through fault simulations is becoming increasingly prohibitive due to the rising SoC complexity. Therefore, the toggle activity could be a clever trade-off between accuracy and computation time for grading SLT applications. However,

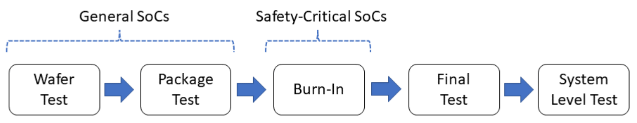


FIGURE 1. **The manufacturing test flow of automotive SoCs.**

SLT-oriented grading is a by-product of the toolchain, which was mainly conceived for BI purposes.

We gather experimental results on two automotive SoC belonging to the SPC58 and SPC56 families of ST Microelectronics, respectively. The SoC from the SPC58 family, designed with different integration strategies and a 40 nm technology node, is ASIL-D compliant and includes about 20 million gates. We apply and evaluate different stress approaches, such as logic BIST, memory BIST, scan-based, and functional stress techniques. Our graphical output shows the main weaknesses of the stress pattern generation process on a colored layout heatmap. The toolchain effectively highlights the unstressed zones and enables the development of additional stress patterns to increase the stress level.

The SoC from the SPC56 family is designed with an older 90 nm technology node, and it includes 2,7 million gates. In the final part of the experimental results, this case of study is used to demonstrate the use of the toolchain for a different SoC family and technology node. By relying on an IEEE standard for simulation dumps and an open format specification for layouts widely integrated within EDA tools, the toolchain does not lose generality when porting it to different SoC design flow. Therefore, porting the toolchain to a different SoC implies only running a preprocessing phase in which we automatically extract layout information. In the experimental results section, functional stress patterns of the SPC56 SoC are evaluated to validate the portability.

We ran all experiments on the medium-size server, and our toolchain was able to complete the entire analysis in a few hours. In contrast, the same task was previously carried out on extremely powerful workstations and required weeks and sometimes even months.

The article is organized as follows. Section II provides some background on the manufacturing process for automotive SoCs and the related evaluation metrics. Moreover, to better clarify the contribution of the article, we present a comparison with the state-of-the-art EDA tools for BI metrics and the proposed toolchain. The comparison is made from the analysis perspective of stress metrics for BI. Section III first illustrates the most critical units of the proposed toolchain; then, it describes every tool in detail. Section IV shows our results on an industrial SoC, and Section V shows how to apply the toolchain to a different industrial SoC. Finally, Section VI draws some conclusions.

II. BACKGROUND

In this section, we include a general description of the manufacturing test process (Section II-A), emphasize the BI phase (Section II-B), illustrate its evaluation metrics (Section II-C), and briefly summarize state-of-the-art EDA tools for BI metrics (Section II-D).

A. MANUFACTURING TEST FLOW

Industrial manufacturing test flows are well-known and precisely defined [3], [16].

Their main target is to discard all faulty chips, and they usually consist of the phases represented in Figure 1:

- Wafer test: Performed at the wafer level, it checks the primary electrical functionalities of the device, and it executes structural tests.
- Package test: Executed after the packaging, it tests and measures the essential electrical characteristics of the pins.
- Burn-In (BI): Latent defects are exacerbated to be captured by later test steps and reduce infant mortality, BI provides external stress to the device in terms of voltage and temperature, aiming at aging the device. Moreover, it provides internal electrical stress using structural and functional patterns.
- Final Test (FT): Run after the BI phase, it captures manufactured devices with some weaknesses. It executes a mix of structural and functional tests.
- System-Level Test (SLT): Performed before shipping, it focuses on hardware, software, and module interactions using advanced functional test programs, such as booting an Operating System [3].

B. BURN-IN PRINCIPLES

The distribution of the failure rate [6] is often described using the bathtub curve of Figure 2. The curve defines three distinct working periods: An infant mortality period (with a decreasing failure rate), a regular (or ''useful'') life (with a low and relatively constant failure rate), and a wear-out period (that exhibits an increasing failure rate).

The BI process [16] is designed to produce a more uniform behavior, i.e., it exacerbates latent faulty SoCs. A BI tester exacerbates latent defects deriving from thin metal oxide or metallization. The climate chamber of a BI tester can have many BI Boards (BIBs), each possibly hosting hundreds of packaged SoCs. Consequently, a single BI tester can simultaneously provide intense external stress

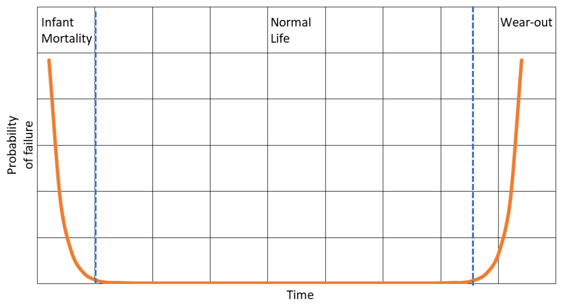


FIGURE 2. **The typical bathtub curve for automotive devices.**

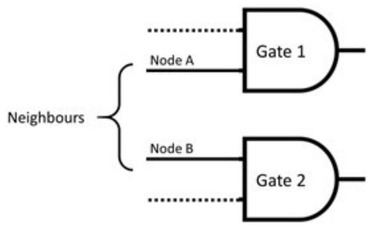


FIGURE 3. Neighbors gates.

to many SoCs. Each BI tester achieves substantial external stress through a temperature and a voltage higher than the nominal ones. These two factors are crucial to aging SoCs and accelerating infant mortality, forcing weaker SoCs to fail in later manufacturing test steps. Another critical feature of a BI tester is the possibility of generating internal overexertion by employing logical electrical stimulation. Internal stress is produced by acting on the SoC scan inputs, providing suitable patterns generated by Automatic Test Patter Generator (ATPG), and using internal logic generators such as logic BIST and functional stress procedures. For young technology, the BI phase can last up to 12 hours. For mature technology, it may last a few hours.

C. STRESS METRICS

Specific metrics are adopted to evaluate the stress capabilities of test patterns [7], [23], [24]. Unfortunately, with new technologies, adequate activity can be obtained only with complex methods, e.g., static electrical stress approaches (which share similarities with *IDDQ* [25] patterns). In this article, we consider the following stress metrics:

- Single point stress metrics. These focus on single-gate events. A gate is considered covered when it executes both transitions (i.e., from 0 to 1 and vice versa) during the simulation. Commonly named *toggle coverage*, it can be further extended with statistics (i.e., the number of times an event occurs on a given gate, also known as *toggle activity*) or timing-related measures (i.e., average toggling frequency of a gate along the simulation time).
- Multiple point stress metrics. These *neighborhood-oriented* metrics focus on close gate pairs and analyze their behavior for a specific amount of time.

As a final consideration, in modern automotive SoC, it is essential to consider the physical layout. As the physical gates distribution is not uniform, the stress can be unbalanced, and latent faults may not be stressed. Therefore, to reach a uniform and specific coverage, *Layout-aware* metrics weight the stress uniformly between dense and less dense sections of the SoC [18]. As represented in Figure 3, it is possible to analyze the physical placement of the gates and detect neighbor gates, i.e., gates whose distance is lower than a given threshold.

Neighbor gates can be further stressed by forcing their inputs to opposite values for a sufficiently long time (t in Figure 4).

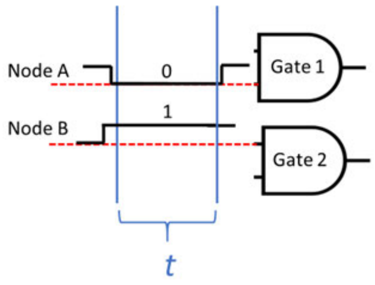


FIGURE 4. Neighbor gates keeping opposite values for some time equal to *t*.

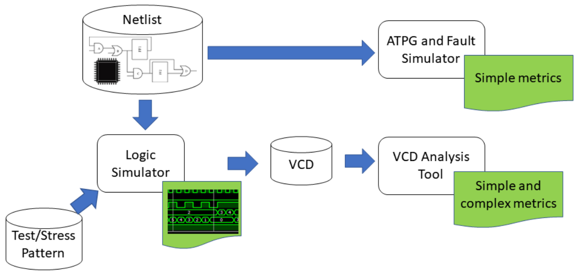


FIGURE 5. Possible evaluation flows for stress metrics.

Effective stress can be generated when the logical configurations "0–1" and "1–0" are maintained on the gate pair by "blocking" well-tailored values for a given time. This measure is called *neighborhood-oriented* static stress metric [18] and it exacerbates latent defects with electromagnetic stress [21], [26], [27], [28].

D. STATE-OF-THE-ART TOOLS FOR BI METRICS

Overexertion must stimulate as many device nodes as possible [7], [23], [24], and evaluating BI metrics is crucial for grading the quality of stress patterns. Figure 5 illustrates two possible high-level workflows for evaluating BI stress metrics: An ATPG and a VCD (Value Change Dump) based approach.

ATPG-based methods (the upper path of Figure 5) focus on toggle coverage metrics based on the fault simulation of the netlist. Unfortunately, they cannot focus on more specific coverage metrics for the BI phase, as stated in Table 1. Moreover, since a single core may execute a single fault simulation, ATPG-based methods may require prohibitive computational times for large devices. VCD-based methods (the lower path of Figure 5) rely on the standard VCD format to evaluate the BI metrics. They often run ad-hoc tools in a post-processing phase without the necessity of performing fault simulation [29].

Table 1 compares the proposed toolchain with some main state-of-the-art EDA tools. The table reports the capabilities of the different tools in terms of the following: adopted metrics (single point, multi-point, layout-aware), the use of parallel techniques to improve time efficiency (notice that process-based techniques are more expensive than

TABLE 1. **The table compares our toolchain with state-of-the-art EDA tools. The Comparison is made only from the analysis perspective of stress metrics for BI.**

Tool	Metrics			Parallelization	Superimpose capability	Coverage-loss Troubleshoot
	Single-point	Multi-point	Layout-aware			
TestMax *Synopsys*	Toggle coverage	N.A.	N.A.	Process based	Yes	Per module/gate coverage
Tessent *Siemens*	Toggle coverage	N.A.	N.A.	Process/thread based	Yes	Per module/gate coverage
Modus *Cadence*	Toggle coverage	N.A.	N.A.	Process/thread based	Yes	Per module/gate coverage
Proposed toolchain	Toggle Coverage Toggle Activity + Avg.	Neighborhood static stress	Yes	Thread based	Yes	Per module/gate, layout heatmap

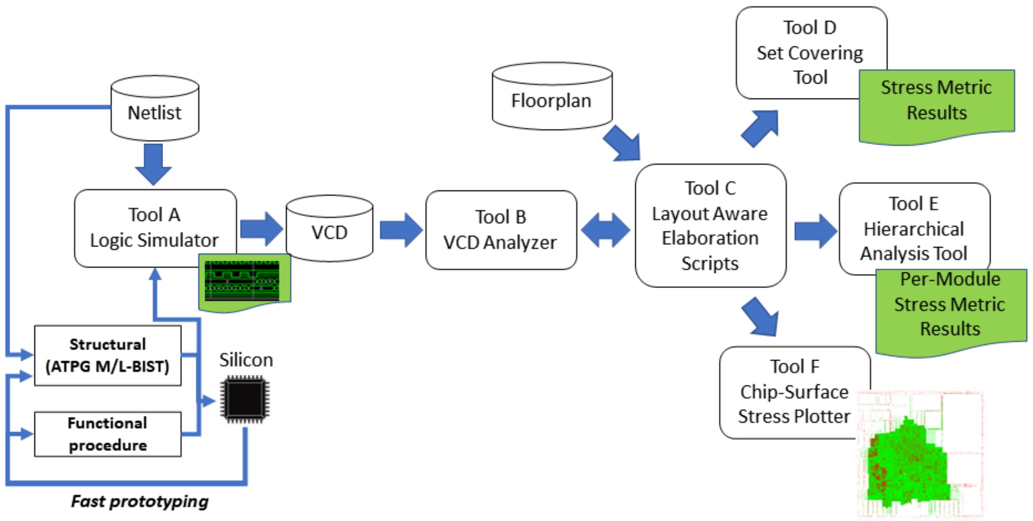


FIGURE 6. **The picture represents a high-level view of the proposed toolchain. The stress pattern, either structural or functional, is validated on the silicon implementation of the SoC. Afterwards, a logic simulation based on the given stress pattern is performed to provide a VCD file. This file is then analyzed to provide a single or multi-point stress coverage for the stress pattern. The Layout-Aware elaboration links the stress pattern to the SoC layout by weighting the stress metrics with the gate density. There are several options for the final step, from plotting the stress over a layout heatmap to superimposing different stress patterns or presenting stress coverage metrics for each module.**

thread-based parallelization), the capability to superimpose different stress patterns, and the troubleshooting capability for coverage-loss (coarse or fine coverage, per module or gate coverage, or by plotting the stress over a layout heatmap). Overall, it could drastically reduce development time and improve the quality of the stress patterns.

III. THE PROPOSED TOOLCHAIN

As the complexity of automotive devices is dramatically increasing, so are the requirements on memory and CPU time from tools to compute metrics for grading stress patterns. Therefore, a common trade-off is to relax the constraints of the final coverage and relief of the tools from time-consuming analysis and simulations.

Moreover, the increasing complexity of automotive devices is also impacting the stress metrics, requiring more advanced metrics [18]. More in detail, the distribution of gates among the SoC floorplan could play a crucial role in distributing uniformly the stress, as well as the mutual interaction between neighbor gates. The latter considerations are not handled by commercial tools in the metrics, as presented in Table 1.

The aim of the proposed toolchain, represented in Figure 6, is to abate the computing cost for handling the increased complexity of automotive devices by exploiting thread-based parallel techniques instead of process-based parallel techniques and to overcome the limitation of current EDA tools in terms of stress metrics analyzed per pattern.

Moreover, it provides test engineers with a visualization of the stress over the layout of the SoC, as well as capabilities to troubleshoot coverage loss by providing fine and coarse stress coverage per gate or module.

As described in Figure 6, the toolchain analyzes a VCD file produced by a commercial logic simulator from a functional test program or a structural pattern. In general, test patterns can be developed on the existing silicon version of the device to reduce the prototyping time [30]. Therefore, after the VCD file generation, the toolchain performs the following steps:

- Tool B: VCD analysis and computation of stress metrics.
- Tool C: Layout-aware elaboration for connecting the VCD analysis to the physical device.
- Tool D: Superimposition of the stress results for different stress patterns to provide incremental coverage.
- Tool E: Hierarchical netlist analysis for computing per-module/gate stress metrics.
- Tool F: Visualization of the stress with a heatmap created from the physical layout of the device.

An automated flow for overcoming the limitation of current commercial EDA tools is the major progress concerning the state of the art that this article is proposing. The concrete benefit is a conjunction of human time cost saving derived from the quick execution times achieved by parallel computing and the clear indications that the results return, either in percentages or visually. In the following sections, we describe every tool in detail from both the theoretical and implementation point of view.

A. TOOL A: THE LOGIC SIMULATOR

During the BI phase, both structural and functional tests are executed. Consequently, the netlist-based logic simulation phase of the SoC is based on a commercial tool capable of generating a VCD file for different stress approaches and then analyzed by the proposed toolchain.

In the testing, verification, and validation steps, the logic simulation phase is one of the most time-consuming phases. Although, during the years, attempts to parallelize logic simulation have been published [31]. These strategies are based on netlist analysis, which would make the approach unfeasible today due to the rising complexity of modern SoC because they are based on time-consuming formal methods. Therefore, a logic simulation is commonly a single-thread process reproducing the device behavior correctly. Occasionally, a second thread can be used to write the VCD file, providing a slight speedup.

Functional stress programs are developed as small programs in terms of executed instructions because, in this case, the simulated netlist needs to behave exactly as the actual implementation of the device, e.g., the power-up operations. Under those circumstances, a logic simulation based on functional patterns cannot be reduced in terms of execution time. Consequently, the functional programs are based on a defined sequence of instructions to be executed for reaching a specific state, and they cannot be further boosted except by changing the hardware on which the logic simulator runs. Nonetheless, a slight speedup can be achieved by preloading the memories before the actual logic simulation starts.

Meanwhile, for logic simulations based on structural patterns, the most straightforward solution is to simulate the whole shifting phase of the test vectors inside the scan chain exhaustively to evaluate the effectiveness of a scan pattern in terms of stressing capabilities. This method can replicate what happens at the hardware level; it could be defined as an "exhaustive" approach because it measures every activity produced by the device. Exhaustive methods can succeed in reproducing the device behavior in a reasonable time if part of the device is stimulated, making a limited number of modules work simultaneously. Conversely, when a scan chain is used, exhaustive methods are not recommended. The shift phase can require the simulation of millions of clock cycles while simulating the entire SoC. Such a combination of duration and activation abilities leads to a very long simulation time (estimated in months for the case study described below).

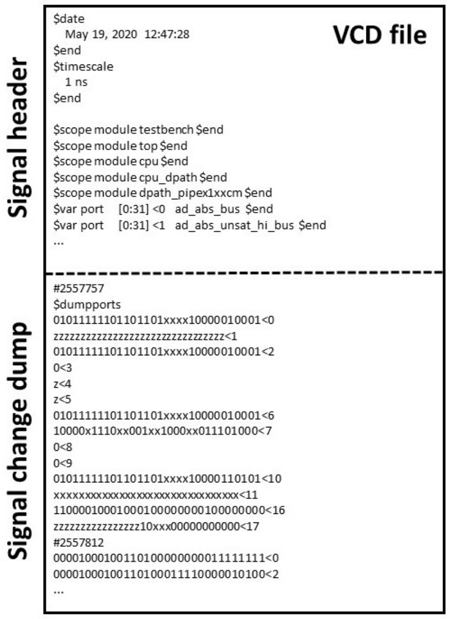
To overcome problems related to scan-chain-based stress and to find a compromise between computational effort and accuracy during the simulation phase, ATPG engines use a "deductive" approach. It consists in loading the corresponding value in each flip-flop of the scan chain in parallel. Such a parallel load approach is based on the following simplified assumption: given a set of test patterns to be applied to the SoC, only the final configuration of the scan chain after the entire shift of every single pattern is considered.

Theoretically, it is true that if applying two consecutive final configurations to the combinational part cause transitions, then the same transitions will show up during the shift operations. It means that:

$$t(deductive) \subseteq t(exhaustive) \tag{1}$$

where t stands for transitions caused by the method within round brackets. This approach is called "deductive" as the recorded transitions constitute a subset of the transitions that take place when applying the exhaustive approach, and a lower bound on the possible transition coverage is deduced. When we simulate two consecutive final configurations, as in the deductive approach, if a gate toggles, we can demonstrate that this gate is guaranteed to toggle in the exhaustive approach. Hence, the gate is guaranteed to toggle. On the other hand, if the gate does not toggle according to the deductive approach, we cannot guarantee that it does not toggle. In conclusion, the results obtained through the deductive approach are approximated but conservative. More in detail, the deductive approach is implemented as an additional simulated set of *System Verilog* tasks. *System Verilog* tasks load a given structural stress pattern in all the scan-chain registers in parallel.

The article [18] illustrates in more detail how the deductive approach works. Even though the deductive approach provides an approximation, its use is valuable not only for its efficiency but also because it is conservative and can guide the test engineer to develop good patterns, leaving the shift phase out of consideration. It is also appropriate for a high switching frequency, driven by a PLL, during the apply phase but less significant during the low switching frequency of the shift phase.


VCD file
Signal header
Signal change dump

```
$date
   May 19, 2020  12:47:28
$end
$timescale
   1 ns
$end

$scope module testbench $end
$scope module top $end
$scope module cpu $end
$scope module cpu_dpath $end
$scope module dpath_pipex1xxcm $end
$var port    [0:31] <0   ad_abs_bus  $end
$var port    [0:31] <1   ad_abs_unsat_hi_bus  $end
...
--------------------------------------------------
#2557757
$dumpports
01011111101101101xxxx10000010001<0
zzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzz<1
01011111101101101xxxx10000010001<2
0<3
z<4
z<5
01011111101101101xxxx10000010001<6
10000x1110xx001xx1000xx011101000<7
0<8
0<9
01011111101101101xxxx10000110101<10
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx<11
11000010001000100000000100000000<16
zzzzzzzzzzzzzzzz10xxx00000000000<17
#2557812
00001000100110100000000011111111<0
00001000100110100011110000010100<2
...
```

FIGURE 7. The VCD file format.

B. TOOL B: THE VCD FILE ANALYZER

The VCD File analyzer is a crucial tool within the toolchain, and it is based on the parallel analysis of typical data format [29].

1) THE VCD FILE FORMAT

A general Value Change Dump (VCD) file produced by the logic simulation is divided into three main sections, as depicted in Figure 7:

- **Signal header**. Each gate or bus is located within a hierarchical structure, and an ID identifies it; the main difference is that gates hold one logic value, while buses hold many, hence the need for a *sub-ID* to identify each of those values. We identify as signals both gates and buses. This section also provides the initial value for all signals. Every signal is guaranteed to appear in it, together with the initial time of the recorded simulation.
- **Signal change dump**. Each change list is introduced by the current time of the simulation; the list contains only the signals that changed state at the previously declared time. The signal initialization and the change lists share the same format, with the difference that the first is ensured to contain all the signals previously declared. The list goes on until the end of the file.

2) THE VCD FILE ANALYZER

The VCD tends to be very large (typically between hundreds of GBytes and few TBytes) due to the high number of changes. Because of this, the signal change dump analysis is the most critical in terms of computation time. As described in Section III, we consider three types of analysis: the full and statistical single-point stress metrics, and the multiple-point stress metric. To speed up the process, we implement a parallel pipeline consisting of 5 stages, each managed by one or more concurrent threads, as shown in Figure 8:

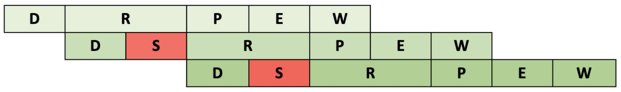


FIGURE 8. The different stages of the pipeline: [D]iscovery, [R]eading, [P]arsing, [E]laboration, [W]rite-Back. In a realistic scenario, some [S]talls might be present.

- The **Discovery** stage divides the file into logical sections, dividing the file for the next stage.
- The **Reading** stage is in charge of reading the logical sections and loading them from the disk to the main memory.
- The **Parsing** stage uses the Reading stage results to transform the text data into logical data; this stage may also update the signal list in the main memory.
- The **Elaboration** stage performs calculations based on the data provided by the Parsing stage, updating the signal list in the main memory.
- The **Write-Back** stage stores the analysis results on disk.

In Figure 8, we have a visual representation of the pipeline stages. In a pipeline, the elapsed time strongly depends on the slowest stage. As the pipeline may need to be more perfectly balanced, stalls may appear when waiting for stages that take longer than others. For example, reading from a disk is the slowest operation; because of this, the Parsing, Elaboration, and Write-Back stages are delayed. In this case, we put more effort into reducing the reading stage overhead as much as possible.

Using Equation 2 we can describe the elapsed time T_{proc} based on the knowledge of the slowest stage $T_{slowest}$ and the number of times N that we make a pipeline call; the other stages, as they are executed in parallel with each other (and with the slowest stage), so the elapsed time for each stage T_s only appears once.

$$T_{proc} = T_{slowest} \cdot N + \sum_{s}^{\{all\ stages \setminus slowest\}} T_s \tag{2}$$

Communication between each stage occurs through thread-safe queues with limited, predefined size; as a stage fills in a queue with its results, the next one empties it until the end of the computation occurs. Please notice that this is a general idea, during implementations for each type of analysis slightly differs. In particular:

- In the **full single-point metric** (toggle analysis), the Parsing and Elaboration stages are merged, as the overhead of data passing would be much higher than the duration of each operation. Moreover, since the Write-Back stage is only needed at the end of the elaboration, it starts after every other stage is finished.
- In the **statistical single-point stress metric**, we have all the stages in the pipeline. Since the results may be

huge on disk, the Write-Back stage works parallel with the others. We have two versions for the output file: One similar to the standard file, with the difference that we only save the toggle information, and one using the widely used SQLite [32].

- In the **multiple-point stress metrics**, the Parse and the Elaboration stage may read and write from the same queue. We designed a mechanism that needs subsequent circuit states to be read. Parallelism in the Parsing stage may lead to an issue where two elements in the queue may not be subsequent. We solved this issue by providing a unique increasing identifier for each element in a queue; in this case, the queue is a priority queue, where the element identifier is the key, and elements are removed in ascending order; if we extract two elements with no consecutive identifiers, those elements are put back in the queue.

Low-level optimizations are also a critical factor in speeding up the process:

- We created a low-level buffered reader that performs file reading and it provides essential tools for parsing. A buffered reader reduces the number of file readings by loading file chunks of approximately 8 MBytes; the choice of chunk size is based on experimental data on our system.
- We minimize the number of dynamic allocations as much as possible, resorting to fixed-size structures allocated before the signal change analysis. Queues are also pre-allocated using their maximum size available.
- We use an $O(1)$ access table for accessing the signals through the ID and sub-ID keys, as signals are frequently accessed and updated by different threads.
- We use data structures optimized for the needed analysis; this comes at the expense of their flexibility. However, as future devices may be larger, we want to reduce memory usage and computation time as much as possible.

Considering the large spectrum of manipulation that the stress information should endure, and as we would like all tools to be compatible, we designed an intermediate human-readable text-based file format to pass information among the various components of the proposed toolchain. This intermediate formatted source allows the toolchain to be improved with less effort, possibly introducing intermediate steps if needed, only sacrificing a small amount of disk memory. Tools are also built to read and write files that must be coherent with the format specified; we divide the values by spaces, while each record is on a separate line, meanwhile comments starts with "#"; comments usually include traceability information such as the metric coverage.

C. TOOL C: LAYOUT-AWARE ELABORATION SCRIPTS

In this phase of the proposed toolchain, a set of scripts is included to provide layout-awareness capabilities. This toolchain elaboration step is exploited to link the stress evaluation with the actual SoC physical characteristics.

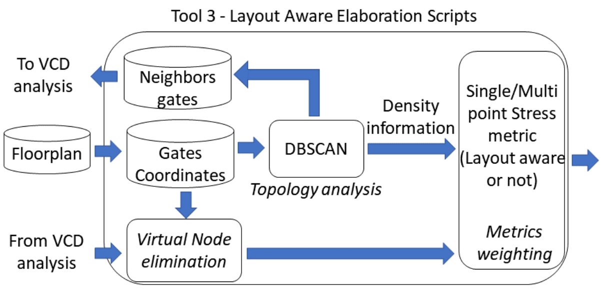


FIGURE 9. Layout-aware elaboration substeps.

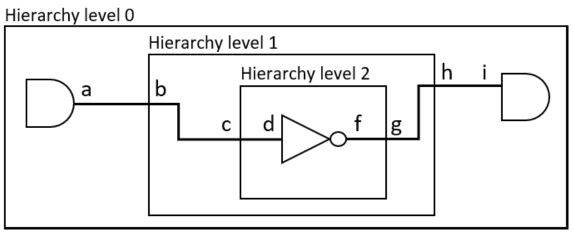


FIGURE 10. An example of the filtering method result.

The substeps are summarized in Figure 9, and three main components can be individuated:

- Virtual node elimination: it eliminates virtual nodes and signals that do not have a physical implementation.
- Topology analysis: it analyzes the floorplan of a given SoC to generate helpful information on the neighbor gates used for multi-point stress metric analysis in the VCD file analyzer.
- Metrics Weighting: the stress metrics can be weighted using a bi-dimensional density by exploiting layout information.

In the following subsection, every component of the Layout-aware elaboration scripts is described in detail.

1) VIRTUAL NODE ELIMINATION

A significant issue in the stress coverage evaluation process comes from the pure VCD analysis of the simulation dump. As a matter of fact, by looking at the VCD header part, the number of signals saved in the dump includes many replications. As shown in Figure 10, a circuit connection traversing hierarchies is memorized several times in the VCD.

More in detail, the path to and from the not gate ports includes physical circuits points "a" to "d" and "f" to "i", but the VCD dump also includes points "b", "c", "g", and "f", which are not corresponding to any real circuit point but are inherited as simulation artifacts. Including extra points leads to longer computations and affects the stress metric value because an unexcited gate may reflect in many VCD signals, thus polluting the final stress metric value.

In order to eliminate VCD over information and not affect stress metrics, a tool is used to filter useless VCD signals. Such a method is similar to a collapsing strategy, but it is not based on the netlist analysis.

The signal pruning is obtained by matching the VCD information with the list of real SoC gates, i.e., gates with a physical implementation in the layout. This operation is based on two steps:

1) The coherent loading of a list of gates extracted from the layout.
2) The linear search of VCD signals in the layout gates.

An example of the operation performed in this step can be seen in Algorithm 1.

Algorithm 1 Virtual Node Elimination Pseudo Code

```
Input: VCD file.
Require: Layout information.
1: Read and save physical gates into a hash table.
2: for signal in VCD do
3:     if signal in the hash table then
4:         Preserve the signal.
5:     end if
6: end for
```

In terms of complexity, the method leads to an overall $O(n)$ algorithmic complexity.

2) TOPOLOGY ANALYSIS

When dealing with a complex System-on-Chip, it is fundamental first to understand its topology, i.e., how gates are physically placed across the layout to gather valuable insights that can help understand the meaning of the computed stress metrics and devise tests to cover all the parts of the SoC adequately and uniformly.

Modern SoCs do not show a uniform distribution of gates on the layout front-end [18]. Traditional stress metrics are usually "gate-based", i.e., they consider the behavior of a gate or a set of gates regardless of how the SoC is structured; they are also "unweighted", i.e., they consider each gate to yield the same contribution to the metric. The stress per unit of the area varies across the layout, and it may lead to different aging scenarios depending on the density of gates in a given area. Therefore, knowledge of the device topology is crucial to assessing the quality and uniformity of the stress.

For the reasons above, we provide layout-aware elaboration steps to enhance the computed stress metrics.

More in detail, the purposes of crossing information extracted from the VCD analysis with layout data permits to:

1) Introduce gates aggregation when measuring multi-point stress metrics.
2) Reach a high level of accuracy by weighting the stress measurement according to the SoC density.
3) Generate SoC stress heatmap plot by providing information about gate coordinates.

The core aspect of the topology analysis is based on a heuristic method capable of generating density information starting from the layout front-end.

It is essential to mention that exhaustive methods exist for computing the bidimensional density. However, when we use exhaustive methods, the computing time grows with the dimension of gates, and it explodes for larger SoC. For the sake of this work, to overcome computing time limitations, Machine Learning approaches are used. In particular, the *Density-based spatial clustering of applications with noise* (DBSCAN) [33] approach is used. The idea of DBSCAN is to generate a set of clustering, i.e., grouping data in the same group with some similarities.

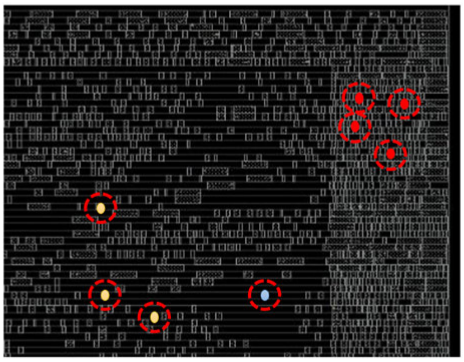


FIGURE 11. An example of the clustering method (DBSCAN) on layout front-end.

DBSCAN is a density-based clustering algorithm. Given a set of points, it groups closely packed points (points with many nearby neighbors), highlighting outliers in the low-density region.

With a fixed inter-gate distance, all the logic gates of the SoC can be organized into tuples of neighbors for performing multiple-point analysis. In the toolchain flow, a gate pair located at a distance smaller than a selected threshold on the layout is considered of interest and extracted to feed the VCD analysis tools for the multi-point analysis.

In particular, the computation time costs to extract a couple of gates close enough to each other are traded off with accuracy; a classification method that implements clustering is proposed to abate the timing costs while guaranteeing sufficient accuracy in the selection.

Figure 11 illustrates with an example how the algorithm works on a generic front-end layout: The classification process divides SoC gates into core points (in red), border points (in yellow), and noise points (in blue), and it analyzes the neighborhood within a fixed inter-gate distance (red dashed circle). Once the clusters are identified over the SoC surface, the list of couples (or tuples) can be performed in a little computation time. Furthermore, the method is helpful for successive steps of the flow, particularly for weighting the stress activity based on the SoC density, which can vary from one region to another in the SoC layout.

3) METRICS WEIGHTING

As stated in the previous subsection regarding the topology analysis, the stress per unit of area differs across the layout in modern SoC. Depending on the density of gates in a given area, it may lead to different aging scenarios. Therefore, knowledge of the SoC topology is crucial to assessing the quality and uniformity of the stress and enhancing the stress

Signal Name	Rising transition-Coverage			Falling transition-Coverage		
	C_1	C_2	C_3	C_1	C_2	C_3
a	1	0	0	1	0	0
b	0	1	0	0	1	0
c	1	0	1	1	0	1
d	1	1	1	1	1	1
e	0	0	1	0	1	1
f	0	0	1	0	1	0

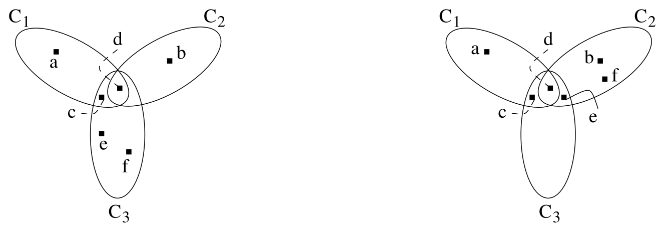


FIGURE 12. **Rising and falling transition set cover: From file to set interacts.**

metric with density awareness as proposed in [18]. This substep of the toolchain provides formulas and considerations presented in [18].

Consequently, a layout-aware stress metric is provided for the single and multi-point stress metric to weigh the stress over denser areas of the SoC instead of considering all the gates equal among them.

D. TOOL D: THE SET COVERING TOOL

During the BI, different stress approaches of various natures can be used. For example, a BI phase can run scan-based patterns, logic or memory BISTs procedures, and functional programs. Distinguishing which coverage is granted by which method (i.e., which pattern covers a specific section of the SoC) is extremely important, as it allows an understanding of which approaches provide more remarkable improvements.

Therefore, the proposed toolchain includes a tool for implementing a set covering analysis. This tool receives the stress results collected by adopting patterns of different natures, generating all possible set interactions, and providing insights about the stress percentage for each stress approach.

In particular, the set tool provides the following:

- A confusion matrix of toggle coverages among different stress approaches.
- The list of unique toggle coverages for each stress approach.
- The number of times that a signal toggles.
- The possibility to merge different results, using the different coverages as a subsequent superimposition of stress approaches.
- The possibility to identify raising or falling transitions of a given gate for each stress approach.

More in detail, for inputs and outputs of gates, we store all the raising and falling signal transitions in the table.

As signal names are unique and immutable and tend to be referenced by many files, we use a technique called "string pooling" or "string interning" to avoid having equal strings in memory as strings would be memory hungry. Signals in different files share references to their names instead of having more instances of the same string. However, reading the input files is the slowest part of the process, string pooling is enabled when moving the signals to an appropriate data structure; in this way, we avoid slowing down the file reading stage. In the beginning, the application reads all covering sets directly from the file and stores them in hash tables using the identifiers of the signal as a hash key. Later, as we identify the sequence of signals, we move them to a dynamic array.

Figure 12 show an example of the entire process. The upper table shows the original set covering representation, which stores each rising and falling transition covering for each signal. The bottom Venn diagrams show the set interpretation and highlighting per-pattern subsets.

Suppose to have N signals (with N potentially very large) and M set covering (with M limited by the number of different covers), the application has an $O(N \cdot M^2)$ time complexity and memory complexity. As the value of M is usually limited to a few units, the time required by the entire process is restricted to a few tens of seconds in the worst case. Moreover, using a single bit for each signal value reduces the memory usage to a few GBytes.

E. TOOL E: THE HIERARCHICAL ANALYSIS TOOL

The "Divide et impera" approach in digital design has increased the overall complexity of SoCs, allowing teams to focus more on a design of a single entity instead of the whole SoC. Following this approach, today's SoCs comprise several subunits with a variable number of other nested subunits up to the leaves, i.e., the logic gates. Therefore, considering metrics on SoCs with millions of gates, it becomes evident that coarse metrics on the overall device have less meaning that thorough computed metrics on modules below the average coverage of the whole SoC.

TABLE 2. A simplified view of the coverage file.

Signal Name	...	Coverage	...
a/b/c	...	1.0	...
a/b/d	...	0.5	...
a/b/f/g	...	0.5	...
a/h	...	1.0	...

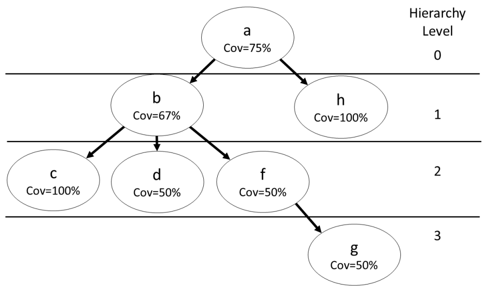


FIGURE 13. A high-level view of the tree data structure containing the coverage for each node.

The stress analysis frequently has to concentrate on some specific module, e.g., the ones showing low coverage. In order to support this analysis, the flow includes a selection process implemented as a tool that extracts critical modules below a given threshold and a per-module stress coverage.

The hierarchical analysis tool analyzes a stress pattern, or a set of them, in order to produce a module-based coverage file where the module and sub-modules have their coverage within the design hierarchy.

The hierarchical analysis starts from the standard human readable text-based input file of coverage. Table 2 presents an example of a coverage file, where signal names contain the hierarchy with the associated stress coverage.

The tool is independent of the SoC and analyzes the input file sequentially to avoid non-deterministic access to data structures. Consequently, it analyzes only the coverage file recreating the hierarchy by decomposing the path. In other words, using as an example the design represented in Table 2, the top-level unit *a* is decomposed into a subunit called *b* and *h*; *b* is further decomposed into its children, i.e., *c*, *d*, *f*, and the leaf *g*.

Internally, it works on parsing strings in such a way as to create a tree of modules and sub-modules, starting from the top entity; for each node of the tree, it calculates and saves the list of signals and their coverages in the internal data structure. The tool creates the tree-coverage structure depicted in Figure 13.

An important aspect to mention is that the coverage of a general parents node in the tree hierarchy follows the recursive formula:

$$Cov(node) = \frac{\sum_{i=0}^{\#children(node)-1} Cov(i)}{n} \tag{3}$$

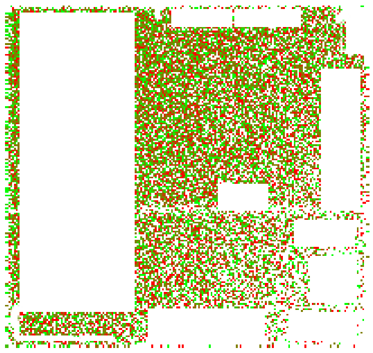


FIGURE 14. An example of stress heatmap over a generic SoC layout.

where n is the number of leaves in the sub-tree, if the number of children of the node is one, then the node coverage is returned. The formula can generate, given a module, its related stress coverage by computing the average on all the children nodes.

More in detail, it generates warnings on those modules in the hierarchy that does not reach the acceptable level of coverage, given as an input parameter. Moreover, it produces a file containing every module with its associated level in the hierarchy and the related stress coverage. Instead of searching for a not satisfied coverage module, the tool also accepts a module name (a string) to extract the stress coverage and generate a coverage file within the module hierarchy.

A hierarchical decomposition of SoC modules eventually allows for test engineers to focus on the module, which would more likely give a more significant step up into the coverage than modules that have already reached an acceptable level.

F. TOOL F: THE CHIP-SURFACE STRESS PLOTTER

Although an essential aspect of the stress approach is a quantitative information provided by the hierarchical analysis tool and the set covering tool, a qualitative visualization of stress plays a crucial role. Qualitative visualization of stress over the SoC layout allows locating stress pattern weaknesses and easily highlighting the coverage abilities of different stress patterns.

The plot tool uses as input a pure outcome of the VCD analysis or the superimposition of different stress approaches from the set covering tool to generate a heatmap of the stress over the SoC layout.

Figure 14 presents an example of such a heatmap over a generic SoC.

All signals are mapped into pixels that summarize the stress applied to the device gates, while white zones are embedded memories that are not part of the BI grading. Gate stress coverage resides between 0% (red pixel) and 100% (green pixel), with values in the middle represented by a color gradient.

As it can be seen from Figure 14, there exist portion of red that correspond to non-stressed modules, portions of green (stressed modules) in different shades depending on the

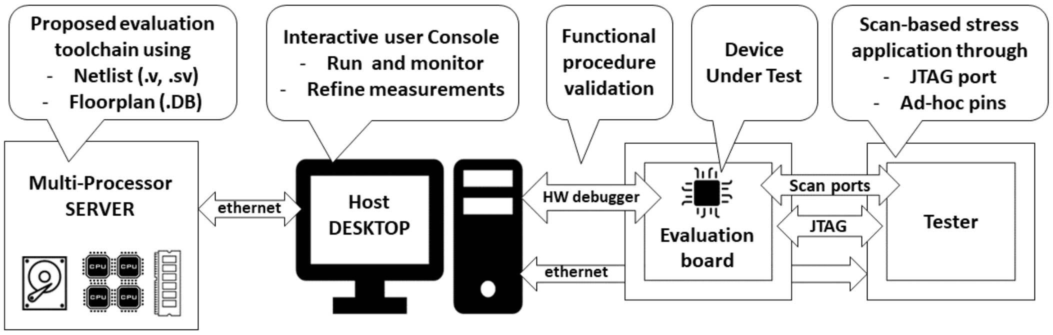


FIGURE 15. Experimental setup.

resolution of the image, and islands of white that correspond to memories, analog and power modules (outside the scope of BI).

Since images of arbitrary resolutions can be generated, it is important to speed up the drawing process depending on the level of detail we need in parallel. GPUs are usually the way to go for image elaboration, as their parallel computation capabilities vastly surpass the ones of the CPU. However, to guarantee compatibility with every device, we perform computations on CPU resorting to the widely used SDL2 libraries [34] for saving the image into a standard format. SDL2 libraries provide APIs for image creation and manipulation on the CPU and GPU. It also provides methods to color the image pixel-by-pixel. Since every pixel is independent, this tool assigns each pixel to a different thread. In this case, we use OpenMP [35], a widely used library for easing parallel patterns implementation to perform image generation.

IV. EXPERIMENTAL RESULTS

Each element in the proposed toolchain is validated on a target industrial automotive SoC from the 40nm SPC58 family produced by ST Microelectronics. Moreover, an additional SoC of the 90nm family SPC56, also manufactured by STMicroelectronics, is considered to prove the portability of the flow from one chip design to another. The following sections describe the SoC analyzed, our experimental setting, and discuss benefits and costs of each phase of our toolchain.

A. THE CASE STUDY: AN AUTOMOTIVE SoC

The target device is a 40 nm Automotive SoC [36] belonging to the SPC58 family manufactured by ST Microelectronics and compliant with the standard ISO26262 ASIL-D. In the following, it is referred to as DUT. The DUT has a multicore architecture with three 32-bit cores using the PowerPC Variable-Length Encoding (VLE) instruction set. It has 6 Mbyte of Flash memory and 128 Kbyte of general-purpose SRAM. It contains about 20 million logic gates in the logic parts and about 700 k flip-flops. Therefore, it constitutes a medium-high complexity case study for the proposed toolchain.

As far as the stress flow during the BI phase is concerned, the DUT is stimulated by:

- A configurable scan chain stimulates the DUT from regular pins using scan-based patterns.
- A logic and memory Built-In Self-Test (BIST) activated from inside or outside the device.
- Some functional programs are executed by the DUT.

The RTL and gate-level description of the DUT, as well as the layout, are available and accessible to extract useful information for the toolchain. Furthermore, the manufactured DUT is available to speed up the development process of functional and structural stress approaches [30].

B. EXPERIMENTAL SETUP

Figure 15 illustrates our simulation and analysis setup designed to evaluate BI stress effectively and mitigate the computational costs of the evaluation. The experimental setup includes two different phases on different platforms:

- A developing phase. In this phase, we use the manufactured SoC to run and verify quickly structural and functional stress patterns. A medium-performance desktop executes the local stress pattern validation phase by communicating with the evaluation board and an ad-hoc developed tester [30], [37]. This desktop is equipped with a quad-core Intel Core i7 running at 2.8 GHz, and 16 GBytes of main memory.
- An analysis phase. In this stage, we evaluate the structural and functional stress patterns. We run the toolchain, from the simulation phase to the extraction of final BI metrics, on a high-performance multi-processor server. The server has an Intel Xeon Gold 6238R processor with 112 CPUs. These processors have 64 bits architecture, allow two hardware threads per CPU, and run at 2.2 GHz. Moreover, the system has 256 GByes of RAM, a storage system of 8 TB, and a network disk of an additional 10 TBytes. The operating system orchestrating the server is CentOS Linux 7.

We use the manufactured SoC during the developing phase to boost the validation phase with stress patterns coming from different sources, such as ATPG (using single or multiple scan ports), LBIST engines (using compressed

TABLE 3. The logic simulation phase: CPU times and size of the VCD files generated. The symbol "*" means the time is estimated. "NA" means that the value is not available.

Acronym	Stress Pattern	Execution time [ms]	Exhaustive simulation [hours]	Deductive simulation [minutes]	VCD file size [GBytes]
32 ATPG	32 ATPG scan patterns	2.5	5,376*	45	8.4
12 ATPG+	12 ATPG scan patterns (additional)	2.0	4,122*	39	4.0
1024 PR	1,024 Pseudo-Random scan patterns	7.0	172,032*	567	251
LBIST	Logic BIST	3.0	14	NA	182
MBIST	Memory BIST	52	240	NA	3,300
FUNCT	Functional (RTOS boot)	2.1	8.5	NA	184

test patterns), firmware controlled MBIST engine and pure functional programs. The validation step is supported by a tester, which applies scan-oriented stress, whereas a hardware debugger supports the development of functional procedures.

For example, validating a stress pattern requires a few seconds (functional stress pattern) to minutes (structural stress pattern), depending on the type of pattern to be executed on the manufactured SoC. More in detail for the time-consuming structural stress patterns, during this phase, they may be applied by resorting to a low-cost microcontroller [37] or a more complex tester based on a Zynq Ultrascale+ MPSoC ZCU104 evaluation board by Xilinx [30]. The application time of structural stress patterns is higher than functional stress patterns due to a large amount of provided data to the high number of scan cells (around 700k); the low-cost electrical connections impact the maximum application frequency of structural patterns.

Downstream of the developing phase, the evaluation process starts on the server, where it must guarantee enough resources to:

- Store huge files on disk, up to Terabytes, generated by the initial simulation and then post-processed to collect refined results.
- Keep in the main memory all required information, up to tens of Gigabytes while running the post-process phase.
- Distribute parallel tasks to CPUs.

C. TOOL A: THE LOGIC SIMULATOR

The logic simulator is the first step in the proposed toolchain. It is based on a commercial logic simulator capable of generating a VCD file, which stores all the signal events during the logic simulation.

As mentioned, during the BI phase, the DUT is exposed to structural and functional stress patterns. Therefore, the logic simulator phase can apply structural and functional stress patterns to the DUT and dumps the related VCD file. The simulation of a structural or a functional stress pattern is substantially different.

In the functional case, the simulation exactly reproduces the behavior of the functional test program. Therefore,

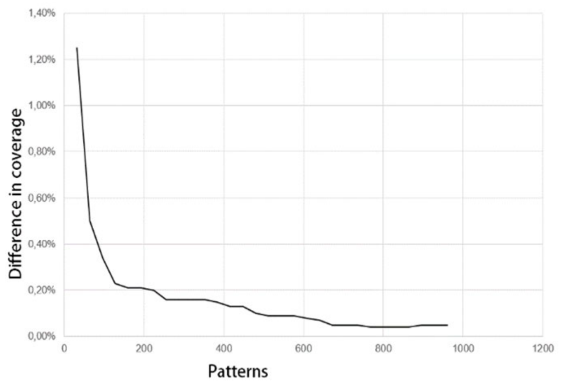


FIGURE 16. Coverage difference between exhaustive and deductive structural simulation for OpenRisc 1200.

a functional test program should be as short as possible from the perspective of execution time.

Regarding the structural patterns logic simulation, as already proved in [18], the exhaustive simulation of a complete shift of the scan chain may require an extremely high number of clock cycles. Exhaustive simulations have been executed on the open-source benchmark OpenRisc 1200, which is small enough to allow the exhaustive simulations to be performed. When 200 patterns are applied, the full results show that the difference in terms of stress coverage between the two strategies amounts to less than 0.2% as Figure 16 shows, whereas the difference in terms of time is, on average, around 1700x more for the complete simulation, and it depends on the number of applied stress vectors. Therefore, the deductive simulation has the tradeoff of a slight decrease in the final coverage but strongly affects the computing time.

Following this strategy, Table 3 shows the simulation time for all stress patterns considered in the given case study. Case in point, the simulation of a 32 ATPG stress vector lasts about 45 minutes using our deductive approach, whereas the estimated time required to run the exhaustive approach would be about 5,376 hours (i.e., seven months). These data prove the validity of the deductive approach, which, as expected, can provide precise and conservative results with reasonable computational effort.

TABLE 4. Profiled execution for the VCD Analysis for the single-point stress metric.

Input VCD	File size [GBytes]	Execution time	Out File size [GBytes]	Rt Memory Usage [GBytes]
32ATPG	8.5	51 s	3.2	11.4
12ATPG+	4.0	18 s	3.2	11.4
1024PR	251	38 m 50 s	3.2	11.4
LBIST	182	21 m 52 s	3.2	11.4
MBIST	3300	10 h 37 m 27 s	3.2	11.4
FUNCT	184	28 m 58 s	3.2	11.4

TABLE 5. Profiled execution for the VCD Analysis for the multi-point stress metric.

Input VCD	File size [GBytes]	Execution time	Out File size [MBytes]	Rt Memory Usage [GBytes]
32ATPG	8.5	4 m 19 s	307	11.4
12ATPG+	4	3 m 5 s	307	11.4
1024PR	251	4 h 47 m 45 s	307	11.4
LBIST	182	4 h 36 m 44 s	307	11.4
MBIST	3300	63 h 1 m 40 s	307	11.4
FUNCT	184	42 h 51 m 4 s	307	11.4

In Table 3, simulation times for the deductive approach regarding Logic and Memory BISTs and functional test programs are not calculated since they are not based on shifting patterns along the entire scan chain.

D. TOOL B: THE VCD FILE ANALYZER

The VCD analysis performs a transformation of the original VCD file from a time-based view to a signal-based view. Experimental results presented in [20] and [29] are updated with VCD files ranging from a few GBytes to TBytes. The execution time bottleneck, as stated in [29], is always the file reading from the disk.

Table 4 shows single-point analysis elaboration times for each stress pattern; the size of input and output files is reported. Main memory occupation is constant between the stress patterns, as each signal contains only trivially copyable elements [38] regarding the analysis; thus, memory occupation only depends on the signal names and the number of signals themselves. On the other hand, the output file size is always constant and directly proportional to the number of signals in the VCD file.

On the other hand, in Table 5, elaboration times for multi-point analysis of different stress patterns are presented, with input and output file sizes and runtime memory usage.

As Table 5 shows, the runtime memory usage is constant across different stress patterns due to the internal data structure containing the signals. The output file size is reduced since the couples are saved as an incremental, unique index. Regarding the execution time, experimental results in Table 5 depict the advantages of resorting to parallel programming for analyzing multi-point metrics in a reasonable amount of time.

TABLE 6. Profiled execution of virtual node elimination script.

Stress Pattern	Input File Size [GBytes]	Output File Size [GBytes]	Execution Time [s]	Runtime Memory Usage GBytes]
1024PR	3.2	2.3	365	16.9
MBIST	3.2	2.3	365	17.0
FUNCT	3.2	2.3	401	17.0

E. TOOL C: LAYOUT-AWARE ELABORATION SCRIPTS

In this phase of the proposed toolchain, a set of scripts provides layout-awareness capabilities to the entire toolchain. Consequently, the following experimental results depend on the analysis of the DUT layout. In the following subsection, experimental results on the Layout-aware elaboration scripts are presented in detail.

1) VIRTUAL NODE ELIMINATION

The layout information is extracted, in advance, from the physical design of the DUT, generating a file that contains the physical positions of logic gates within the layout.

The aforementioned preliminary step allows refining the stress metric produced by the VCD analysis tool in such a way as to correctly consider only physical, implemented logic gates.

As time and memory usage depend mainly on the output size of the VCD elaboration the result, this step has a fixed cost (proportional to the number of signals present in the input file) for each analyzed DUT as Table 6 shows.

2) TOPOLOGY ANALYSIS

The topology analysis on the layout of the DUT is performed as a preliminary step of the toolchain to generate all the required data.

The DUT layout used as a case study can be seen in Figure 17, where an important concept can be highlighted: typically, an SoC does not show a uniform distribution of gates. The differences in the gate density distribution of the various parts of an SoC are further highlighted in the Figure 17; a brighter shade of green describes parts with a higher gate density, while a darker shade of green indicates low gate density.

Following the aforementioned consideration, in order to extract the neighbor gates of a given gate for being used in the toolchain, there exist two different approaches:

- The exhaustive method is based on elaborating the list of all gates. Each gate is analyzed by comparing its Euclidean distance with its layout physical position.
- The heuristic method is based on the DBSCAN algorithm to extract the neighbors for a given logic gate by using as internal metric the Euclidian distance with a fixed internode distance of $6\mu m$.

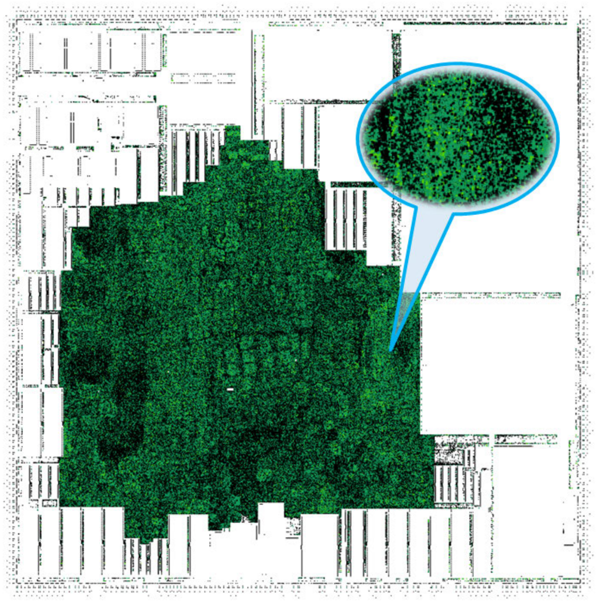


FIGURE 17. Density-colored heatmap for the DUT.

TABLE 7. Comparing the execution time of the exhaustive and the heuristic approach.

Analysis approach	Execution Time
Exhaustive	20 days
Heuristic (DBSCAN)	654.13 s

Table 7 compares the execution time of the Exhaustive method and the heuristic one.

As seen from Table 7, the execution time of the exhaustive method is 1000 orders of magnitude more than the heuristic one. This substantial reduction in the execution time allows for analyzing more complex DUT.

Regarding the exhaustive method, the runtime memory consumption is stable, and it is less than 4 GBytes due to the elaboration of a single gate at the time. An important aspect to mention, rather than the execution time of the heuristic method is the runtime memory consumption as Figure 18 depicts. Due to the nature of the heuristic method, it can analyze more than one gate in parallel. Therefore, its memory consumption is unstable, as seen from Figure 18. The initial ramp-up is due to the file reading where the physical positions of gates are stored. On the other hand, the peak in memory consumption is when the DBSCAN starts its computation. It creates for each gate a cluster, thus the peak in the memory usage, and it gradually merges neighborhood clusters until a stable configuration.

Whereas a heuristic method is used, it dramatically relieves the execution time, but it impacts the accuracy of the analysis compared to the exhaustive method. Therefore, it is fair to report the accuracy of the analysis by utilizing a confusion matrix shown in Table 8.

It can be observed that the heuristic method returns a larger set of close nodes, i.e., about 30 million pairs. On the contrary, the heuristic method returns 75 million close pairs and misses

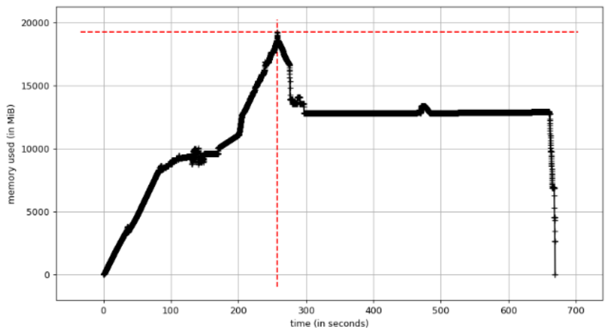


FIGURE 18. Runtime memory consumption of the heuristic method.

TABLE 8. Confusion matrix showing the accuracy of the heuristic method compared to the exhaustive.

Exhaustive \ Heuristic	Close	Far
Close	30 M	1 M
Far	75 M	200 K

almost 1 million far pairs. Overall, the exhaustive and the heuristic methods discard coherently 200 thousands of pairs. Therefore, the accuracy of the heuristic method is 95,9%, which is acceptable for the other analysis.

3) METRICS WEIGHTING

Downstream the topology and VCD analyses, the single or multi-point metric can be enhanced with layout awareness capabilities. As for the evaluation of the stress metrics, experiments have been performed to show how layout awareness affects the metrics and their computational costs regarding time and memory consumption.

Table 9 details results starting from the single point metric, the toggle activity, which is enhanced with layout awareness coverage, i.e., weighting the metric based on the density surrounding a given logic gate.

As seen from Table 9, the required execution time and runtime memory usage remains constant across different stress patterns. The reason behind the constant values of memory and execution time is the nature of the input file containing the list of signals and their toggle activity. Those files have the same number of signals. Thus the same size and the script introducing layout awareness in the simple stress metric is independent of the stress pattern.

On the other hand, Table 10 reports the results on the multiple-point metric enhanced with the layout awareness.

Table 10 confirms the same results seen in Table 9, i.e., the execution time and the runtime memory usage are constant across different stress patterns, and they only depend on the input file size and its nature.

Furthermore, in order to prove the effectiveness of layout awareness metrics, Figure 19 visually represents how the weighted and unweighted activity metrics evolve concerning the number of applied structural patterns in the OpenRISC 1200 [18].

TABLE 9. Single point stress metric coverage.

Stress pattern	Toggle coverage [%]	Layout-aware coverage [%]	Execution time [s]	Rt Memory Usage [GBytes]
32ATPG	82.49	82.5	473.69	31
12ATPG+	83.21	83.14	492.75	31
1024PR	91.53	90.32	508.43	31
LBIST	85.49	87.5	478.88	31
MBIST	10.08	7.9	510.60	31
FUNCT	10.59	8.56	612.38	31

TABLE 10. Multiple point stress metric coverage.

Stress pattern	Neighbors coverage [%]	Layout-aware coverage [%]	Execution time [s]	Rt Memory Usage [GBytes]
32ATPG	80.47	80.48	328.42	15
12ATPG+	79.87	79.84	331.10	15
1024PR	87.04	87.04	278.38	15
LBIST	83.96	83.96	335.18	15
MBIST	3.11	2.78	289.64	15
FUNCT	28.58	28.56	286.61	15

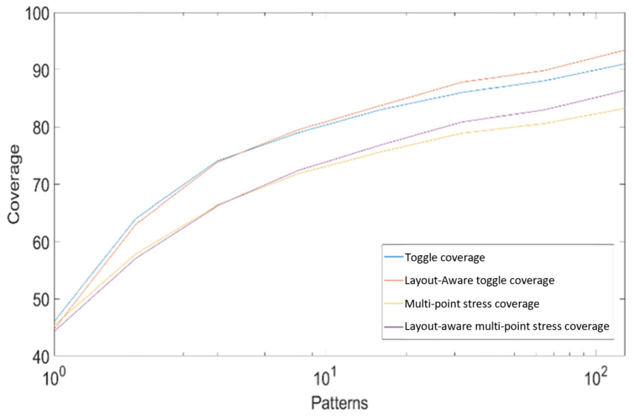


FIGURE 19. Evolution of BI metrics.

When just a few patterns are used, the layout-aware metric is lower than the unaware one, whereas when more than eight patterns are applied, the layout-aware metric tends to have higher values. Consequently, a much more significant part of the denser areas of the DUT is being covered.

This kind of behavior captures the way the patterns stimulate the different parts of the DUT. Indeed, with just a few patterns applied, the stimulation is nicely "spread" across the DUT. On the contrary, when many patterns are applied, the activity concentrates on the denser parts of the DUT. In this way, the contribution to the layout-aware metric tends to increase. Eventually, the layout-aware metric values exceed those provided by the unaware ones.

F. TOOL D: THE SET COVERING TOOL

The time and memory costs required by the set covering tool mainly depend on the number of signals analyzed and the number of input files compared and merged. File reading is the slowest step of this stage, whereas analyzing the sets require fewer resources than the file reading step.

TABLE 11. Profiled execution of the set covering tool.

Number of Input files	Execution Time [s]	Average Memory Usage [GBytes]
4	459	26.1
6	500	37.1
8	544	48.2
10	676	59.3

Table 11 shows the memory and time consumption trend over an increasing number of files of a fixed size of 2.3GB, performing all the available operations on the tool, which includes comparing and merging capabilities.

Memory increases proportionally with the number of files involved. However, the execution time increases only by a small amount. As more and more files get added, profiling shows that moving them to the proper data structure is the most costly operation, with the operation of saving the results. On the other hand, the output file size is always the same as the input file.

G. TOOL E: THE HIERARCHICAL ANALYSIS TOOL

In order to analyze the stress pattern and its strength in terms of coverage in the entities and sub-entities composing the DUT, a hierarchical analysis is performed.

Table 12 presents a profiled execution of the tool for a different number of entities for which the coverage is extracted. Moreover, runtime memory usage and execution time for different execution is presented, as well as the size of the output file. Table 12 depicts a constant execution time and memory consumption independently from the stress pattern. Therefore, its execution time, output file size, and memory usage is directly proportional to the number of signals in the input file, hence, in the DUT.

The high memory consumption is due to the internal data structure of the tool holding all the information for each entity within the DUT.

H. TOOL F: THE CHIP-SURFACE STRESS PLOTTER

A plotter tool is used for troubleshooting, visually, the eventual coverage loss of a stress pattern or showing weaknesses due to the superimposition of different stress patterns. A stress-colored heatmap is produced by exploiting the physical placement of logic gates in the DUT layout.

Table 13 shows the runtime memory usage and execution time and how they are affected by the resolution of the image. Regarding the input files, they are most of the same size (around 2.3 GBytes for the analyzed stress pattern and 2 GBytes for the file containing the physical placement of gates as coordinates).

TABLE 12. Profiled execution of Hierarchical analysis tool.

Extracted entities	Output File Size [MBytes]	Execution Time [s]	Rt Memory usage [GBytes]
2	172	1,140	36.7
4	172	1,119	36.7
6	172	1,245	36.7
8	172	1,124	36.7

TABLE 13. Profiled execution of chip-surface stress plotter.

Resolution [pixels]	Average File Size [MBytes]	Execution Time [s]	Rt Memory Usage [GBytes]
1000x1000	4	535	3.9
2000x2000	16	537	3.9
4000x4000	62	559	3.8
8000x8000	245	532	4.8

Most of the time cost to build images is the file reading, which, again, it strongly depends on the number of signals and the number of gates in the DUT.

Upscaling or downscaling the image between resolution values of 1,000 $\times$ 1,000 to 10,000 $\times$ 10,000 does not impact performance as much as the file reading, due to the intrinsic parallelism of the drawing operation, despite the image generation being performed on the CPU rather than on GPU.

For example, Figure 22 contains output images representing the stress coverage from different stress patterns.

In addition, Figure 22 highlights unstressed and stressed regions for different patterns. Figure 22a depicts a non-stressed region better strained by the pattern in Figure 22b. Similarly, the pattern in Figure 22d focuses on a module not well stressed by the pattern represented in Figure 22c. On the other hand, the same pattern does not activate enough memory ports and some functional units, i.e., the upper and lower left zooms in Figure 22d. Finally, Figure 22e and Figure 22f show how to adequately stress memory ports and functional units, respectively.

I. WRAPPING UP THE FLOW

The BI for safety-critical devices includes different patterns, ranging from structural to functional. Therefore, as represented in Table 3, single patterns are superimposed and analyzed to understand their weaknesses and ability. This activity allows the generation of the stress-colored heatmap plot for the overall BI phase displayed in Figure 20. From these images, it is straightforward to distinguish the zones where the stress level is adequate from those needing additional patterns (such as the area inside the dotted red circle).

As pinpointed from Figure 20 and confirmed from the hierarchical analysis tool (Tool E), an unstressed zone exists that does not reach an acceptable level of stress coverage.

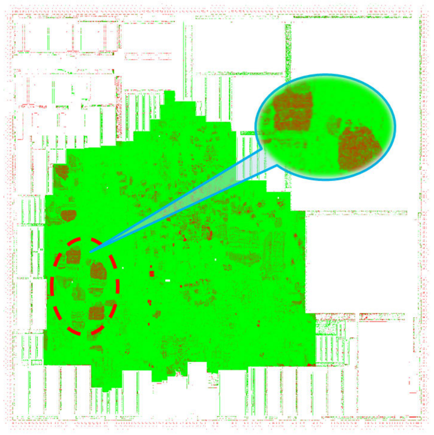


FIGURE 20. Visualization of the overall stress provided by the superimposition of all stress patterns.

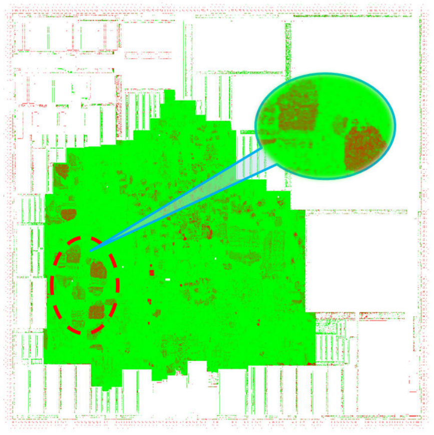


FIGURE 21. Visualization of the overall stress provided by the superimposition of all stress patterns, plus the additional functional stress pattern targeting the identified unstressed module.

Therefore, we develop an additional functional stress pattern that can provide additional stress as represented in Figure 21.

As shown from Figure 21, the ad-hoc developed functional pattern stresses the region of interest, effectively increasing the stress coverage.

V. THE TOOLCHAIN PORTABILITY

This section focuses on the portability of our toolchain on a different SoC. To provide the effectiveness of the proposed toolchain on a different SoC from a different family, in conjunction with STMicroelectronics, we singled out an SoC with a higher technology node (90nm vs. 40nm of the other SoC used as a benchmark) without using the new SoC as a benchmark for the designed toolchain but rather to prove the easiness of porting the toolchain to a different Soc, from a different family, and with a different technology node.

A. THE CHARACTERISTICS OF THE NEW SoC

The target device is a 90 nm Automotive SoC [39] belonging to the SPC56 family, manufactured by ST Microelectronics,

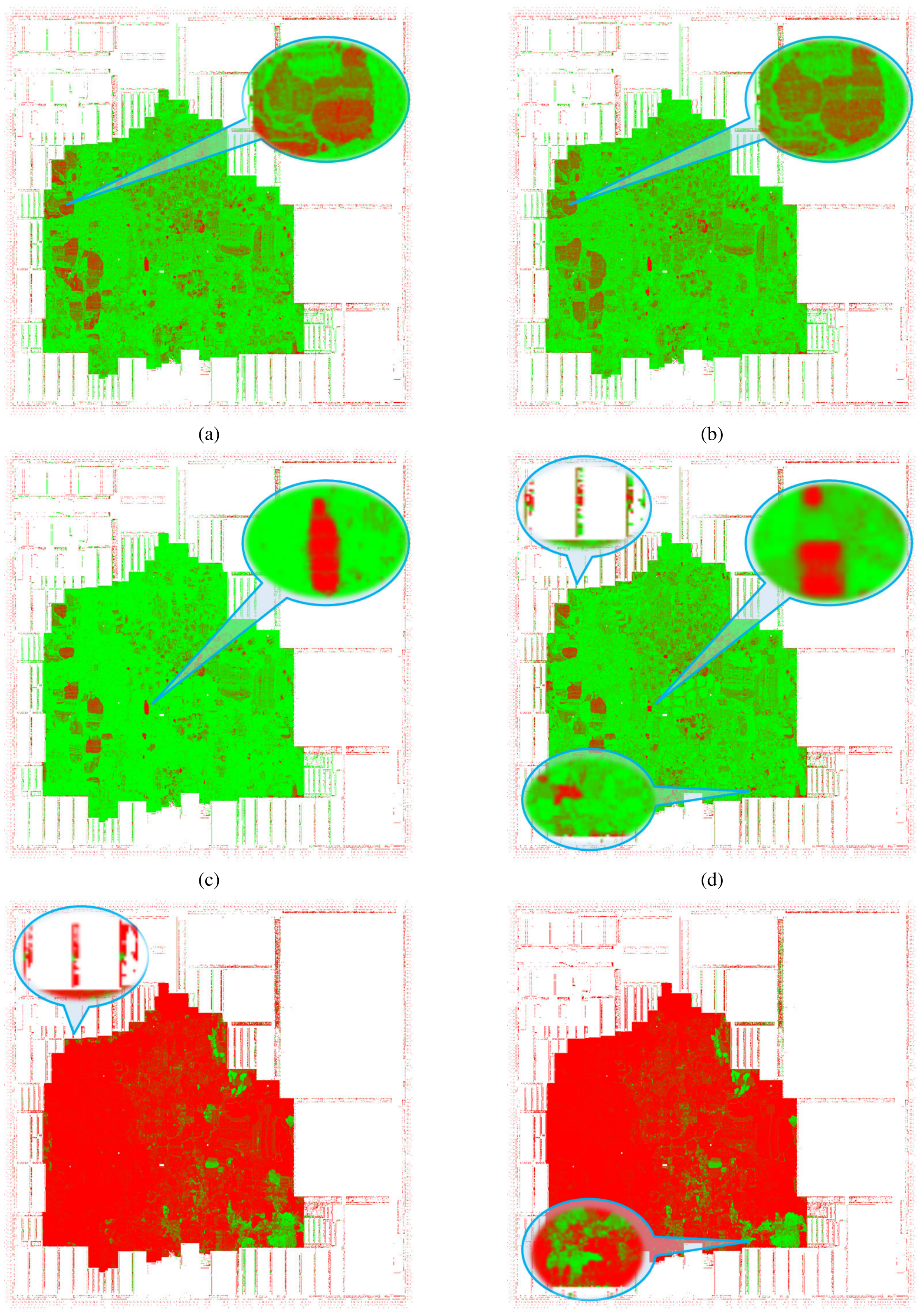

(a)
(b)
(c)
(d)
(e)
(f)

FIGURE 22. Stress-colored heatmaps in terms of toggle coverage with detailed zoom on some regions. Figure 22a: 32 scan based ATPG patterns, 22b: 12 Selective ATPG patterns; 22c: 1024 Scan based pseudo-random patterns; 22d: LBIST patterns; 22e: MBIST patterns; 22f: Functional pattern.

TABLE 14. Profiled execution for the VCD Analysis for the single-point stress metric.

Input VCD	File size [GBytes]	Execution time	Out File size [GBytes]	Rt Memory Usage [GBytes]
FUNCT1	35	162.32 s	0.351	2.4
FUNCT2	12	22.49 s	0.351	5.3
FUNCT3	28	42.63 s	0.351	13.3
FUNCT4	110	125.34 s	0.351	18.4

TABLE 15. Comparing the execution time of the exhaustive and the heuristic approach.

Analysis approach	Execution Time
Exhaustive	2.7 days
Heuristic (DBSCAN)	88.3 s

and compliant with the standard ISO26262 ASIL-D. This device has a single-core 32-bit architecture using the PowerPC Variable-Length Encoding (VLE) instruction set. It includes 4 Mbyte of flash memory and 192 Kbyte of general-purpose SRAM. It contains about 2.7 million logic gates in the logic parts and about 86 k flip-flops. Therefore, it constitutes a medium-complexity case study to show the portability of our toolchain on a different device architecture. We refer to this device as DUT2.

Regarding stress patterns, the DUT2 is only stimulated by four functional programs. The RTL and gate-level description of the DUT2 and the layout are available and accessible to extract helpful information for the toolchain.

B. THE NEW EXPERIMENTAL RESULTS

Running the toolchain to a different SoC requires a pre-processing phase. This step automatically extracts valuable layout information from the floorplan. The floorplan is generated by a physical design EDA tool. Physical design EDA tools can save the layout in a Design Exchange Format (DEF)/Library Exchange Format (LEF) [40]. Those formats are open specifications for describing the physical layout of an integrated circuit widely adopted by EDA tools.

More in detail, we first use the floorplan to generate a file containing the placements of all gates. Then, we use this file to analyze: the neighbor gates for the multi-point analysis, the virtual node elimination for linking the logic simulation to the layout, and the density information for weighting the stress metrics.

The VCD file generated from the Logic simulation is based on an IEEE standard [41]; consequently, the VCD file analyzer does not need any adjustment. Therefore, since the fundamental input files are based on two widely used standards and formats, the toolchain can be easily integrated within every design flow without losing generality.

As reported in Table 14, the analysis time strictly depends on the input file size, which depends on the number of signals stored during the simulation. Overall, we have a linear dependency on the execution time concerning the file size generated.

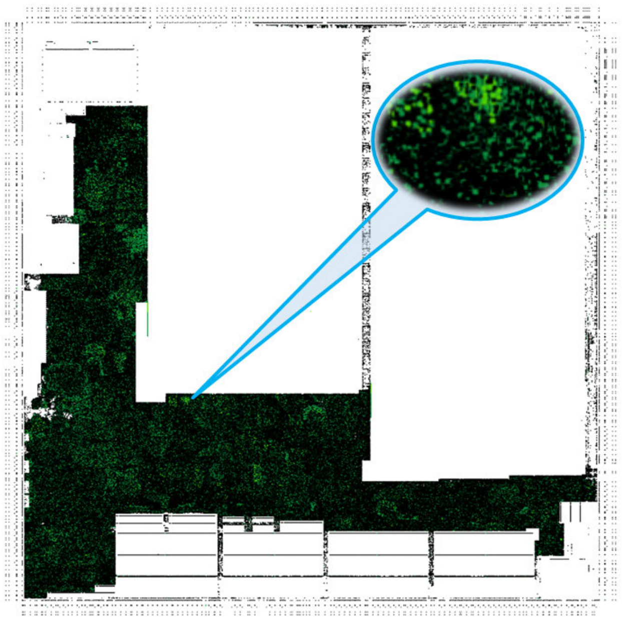


FIGURE 23. Density-colored heatmap for the DUT2.

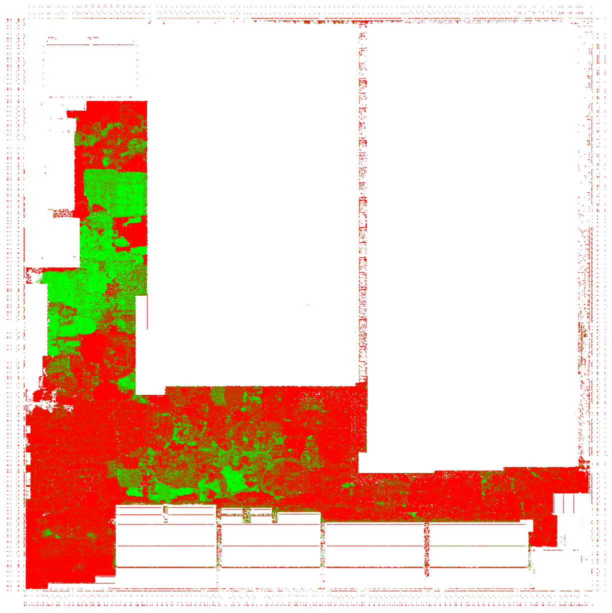


FIGURE 24. Visualization of the overall stress for the DUT2.

Regarding the topology analysis, it is worth mentioning that since the case study has a minor complexity compared to the one presented before, as Table 15 shows, it is feasible to execute an exhaustive topology analysis.

Different technology nodes (in this case 90 nm) and numbers of gates lead to a different gate density distribution, as Figure 23 shows.

The same concept can be applied to the chip-surface stress plotter. Higher technology nodes have a higher distance between gates. Therefore, by maintaining the exact resolution of the previous case study, the street-colored layout heatmap is represented in Figure 24.

Compared to the case study analyzed before, memory usage and execution time are lower due to the reduced complexity in terms of gates.

The DUT2 has a higher technology node (90nm); consequently, the SoC layout is strongly different, i.e., the intra-gates distance is bigger, and it has less dense areas (as well as fewer gates). The technology node impacts all the layout-aware tools of the toolchain. Regarding the VCD, since we rely on a commercial logic simulator, it dumps the VCD according to the IEEE standard. However, the reduced complexity in terms of logic gates affects only the number of signals in the VCD, reflecting a reduced file dimension.

VI. CONCLUSION

SoCs continuously grow in size and complexity and require more efficient and scalable strategies to be designed, tested, and verified. Unfortunately, recent statistics show that most SoC designs miss their time-to-market deadlines since their complexity grow much faster than many testing and verification strategies [42].

One of the most critical phases in the automotive design flow is the so-called BI phase and its evaluation [8]. Burn-in is the process used to stress SoCs before their final test [3].

A crucial aspect when assessing the BI phase is that a stress pattern, or a set of stress patterns, must stimulate the whole device uniformly to an acceptable level. Achieving acceptable stress in the BI phase is essential to exacerbate latent defects that may arise during later manufacturing test steps to increase the quality of SoCs. However, stress patterns are usually retrieved from other manufacturing process steps and sometimes even from the early steps, such as the verification phase during the design. Although this approach may speed up the development of BI stress patterns generation, it may result in unstressed gates due to the nature of the pattern, or not considering all the possible working modes, or non-uniform stress, of a module within the DUT [11].

This article illustrates a flexible, efficient, and scalable toolchain to effectively evaluate the BI stress efficiency in integrated circuits.

To summarize, the main characteristics of the proposed toolchain are the following:

- To overcome the limitations of simple toggle coverage by generating extended statistics on the average toggle events and the total toggle events.
- A layout-aware stress metric based on topology information (such as gate neighbors) is provided to overcome problems derived from uniformity and variation in the device density.
- To superimpose different stress patterns to understand each pattern's strengths and weaknesses.
- To improve the designer's confidence by providing visualization tools over a stress heatmap over the layout of the devices. Visually, moving from one plot to another, even with the human eye, some approaches stimulate, in green, the part of the device better than others where it is red. Moreover, it provides per-module/gate stress coverages.
- To improve the scalability of large devices by drastically reducing the computation effort, in terms of time and memory, through parallel techniques, clever software optimizations, and machine learning techniques. This way, the computing time drastically shrinks from days or weeks to just a few hours.
- To enhance the portability for different SoC by extracting layout information.

Nevertheless, even using the incremental superimposition of different stress patterns, the stress layout heatmap may still have red stains. For example, on the boundaries of the SoC, there are analog modules and pads which any stress pattern could not stimulate. Regarding the Sea of Gates, red stains are due to the presence of bypassed or unstimulated gates of the interconnection between components, the presence of timer modules and sparse logic, which could be safely activated only by specific functional procedures.

All main steps of the toolchain have been evaluated using an industrial SoC from the SPC58 family of ST Microelectronics. Experimental results show that the proposed approach significantly reduces the analysis time, and at the same time, it improves the confidence and accuracy of BI metrics. In addition, it increases the designer's confidence in developing new stress pattern, troubleshooting eventual coverage-loss problems, and enhancing productivity as a side effect. In addition, the portability of the proposed toolchain is validated on a different SoC family, the SPC56 family, manufactured by ST Microelectronics.

ACKNOWLEDGMENT

The authors would like to thank Lorenzo Cardone, Stefano Littardi, Walter Ruggeri, and Giovambattista Gallo, for writing initial prototypes of their tools, cooperating in the experimental analysis, and giving them valuable hints and suggestions.

REFERENCES

[1] A. Vassighi, O. Semenov, M. Sachdev, A. Keshavarzi, and C. Hawkins, "CMOS IC technology scaling and its impact on burn-in," *IEEE Trans. Device Mater. Rel.*, vol. 4, no. 2, pp. 208–221, Jun. 2004.

[2] *Road Vehicles—Functional Safety*, Standard ISO 26262-[1-10], 2011.

[3] I. Polian, J. Anders, S. Becker, P. Bernardi, K. Chakrabarty, N. ElHamawy, M. Sauer, A. Singh, M. S. Reorda, and S. Wagner, "Exploring the mysteries of system-level test," in *Proc. IEEE 29th Asian Test Symp. (ATS)*, Nov. 2020, pp. 1–6.

[4] H. H. Chen, "Beyond structural test, the rising need for system-level test," in *Proc. Int. Symp. VLSI Design, Autom. Test (VLSI-DAT)*, Apr. 2018, pp. 1–4.

[5] P. Varma, "System chip test: Are we there yet?" in *Proc. Int. Test Conf.*, 1998, p. 1144.

[6] T. M. Mak, "Infant mortality—The lesser known reliability issue," in *Proc. 13th IEEE Int. On-Line Test. Symp. (IOLTS)*, Jul. 2007, p. 122.

[7] M. F. Zakaria, Z. A. Kassim, M. P.-L. Ooi, and S. Demidenko, "Reducing burn-in time through high-voltage stress test and Weibull statistical analysis," *IEEE Design Test Comput.*, vol. 23, no. 2, pp. 88–98, Mar. 2006.

[8] A. Benso, A. Bosio, S. Carlo, G. Natale, and P. Prinetto, "ATPG for dynamic burn-in test in full-scan circuits," in *Proc. 15th Asian Test Symp.*, Nov. 2006, pp. 75–82.

[9] D. Appello, C. Bugeja, G. Pollaccia, P. Bernardi, R. Cantoro, M. Restifo, E. Sanchez, and F. Venini, "An optimized test during burn-in for automotive SoC," *IEEE Design Test.*, vol. 35, no. 3, pp. 46–53, Jun. 2018.

[10] F. Almeida, P. Bernardi, D. Calabrese, M. Restifo, M. S. Reorda, D. Appello, G. Pollaccia, V. Tancorre, R. Ugioli, and G. Zoppi, "Effective screening of automotive SoCs by combining burn-in and system level test," in *Proc. IEEE 22nd Int. Symp. Design Diag. Electron. Circuits Syst. (DDECS)*, Apr. 2019, pp. 1–6.

[11] F. Angione, P. Bernardi, G. Filipponi, M. S. Reorda, D. Appello, V. Tancorre, and R. Ugioli, "An optimized burn-in stress flow targeting interconnections logic to embedded memories in automotive systems-on-chip," in *Proc. IEEE Eur. Test Symp. (ETS)*, May 2022, pp. 1–6.

[12] S. Biswas and B. Cory, "An industrial study of system-level test," *IEEE Design Test Comput.*, vol. 29, no. 1, pp. 19–27, Feb. 2012.

[13] D. Appello, H. H. Chen, M. Sauer, I. Polian, P. Bernardi, and M. S. Reorda, "System-level test: State of the art and challenges," in *Proc. IEEE 27th Int. Symp. On-Line Test. Robust Syst. Design (IOLTS)*, Jun. 2021, pp. 1–7.

[14] P. Bernardi, M. Restifo, M. S. Reorda, D. Appello, C. Bertani, and D. Petrali, "Applicative system level test introduction to increase confidence on screening quality," in *Proc. 23rd Int. Symp. Design Diag. Electron. Circuits Syst. (DDECS)*, Apr. 2020, pp. 1–6.

[15] P. Bernardi, A. Bosio, G. Di Natale, A. Guerriero, E. Sanchez, and F. Venini, "Improving stress quality for SoC using faster-than-at-speed execution of functional programs," in *VLSI-SoC: System-on-Chip in the Nanoscale Era—Design, Verification and Reliability* (IFIP Advances in Information and Communication Technology), vol. 508, T. Hollstein, J. Raik, S. Kostin, A. Tšertov, I. O'Connor, and R. Reis, Eds. Tallinn, Estonia: Springer, Sep. 2016, pp. 130–151. [Online]. Available: https://hal.inria.fr/hal-01675205

[16] C. He and Y. Yu, "Wafer level stress: Enabling zero defect quality for automotive microcontrollers without package burn-in," in *Proc. IEEE Int. Test Conf. (ITC)*, Nov. 2020, pp. 1–10.

[17] S.-J. Lee, S.-G. Lee, B.-S. Suh, H. Shin, N.-I. Lee, H.-K. Kang, and G. Suh, "New insight into stress induced voiding mechanism in Cu interconnects," in *Proc. IEEE Int. Interconnect Technol. Conf.*, Jun. 2005, pp. 108–110.

[18] W. Ruggeri, P. Bernardi, S. Littardi, M. S. Reorda, D. Appello, C. Bertani, G. Pollaccia, V. Tancorre, and R. Ugioli, "Innovative methods for burn-in related stress metrics computation," in *Proc. 16th Int. Conf. Design Technol. Integr. Syst. Nanosc. Era (DTIS)*, Jun. 2021, pp. 1–6.

[19] D. Appello, P. Bernardi, R. Cagliesi, M. Giancarlini, M. Grosso, E. Sanchez, and M. S. Reorda, "Automatic functional stress pattern generation for SoC reliability characterization," in *Proc. 14th IEEE Eur. Test Symp.*, May 2009, pp. 93–98.

[20] D. Appello, P. Bernardi, A. Calabrese, S. Littardi, G. Pollaccia, S. Quer, V. Tancorre, and R. Ugioli, "Accelerated analysis of simulation dumps through parallelization on multicore architectures," in *Proc. 24th Int. Symp. Design Diag. Electron. Circuits Syst. (DDECS)*, Apr. 2021, pp. 69–74.

[21] D. Vogel, S. Rzepka, B. Michel, and A. Gollhardt, "Local stress measurement on metal lines and dielectrics of BEoL pattern by stress relief technique," in *Proc. Semiconductor Conf. Dresden*, Sep. 2011, pp. 1–3.

[22] P. Maxwell, I. Hartanto, and L. Bentz, "Comparing functional and structural tests," in *Proc. Int. Test Conf.*, 2000, pp. 400–407.

[23] D. Appello, P. Bernardi, G. Giacopelli, A. Motta, A. Pagani, G. Pollaccia, C. Rabbi, M. Restifo, P. Ruberg, E. Sanchez, C. M. Villa, and F. Venini, "A comprehensive methodology for stress procedures evaluation and comparison for burn-in of automotive SoC," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, Mar. 2017, pp. 646–649.

[24] X. Guo, W. Burleson, and M. Stan, "Modeling and experimental demonstration of accelerated self-healing techniques," in *Proc. 51st ACM/EDAC/IEEE Design Autom. Conf. (DAC)*, Jun. 2014, pp. 1–6.

[25] R. Kawahara, O. Nakayama, and T. Kurasawa, "The effectiveness of IDDQ and high voltage stress for burn-in elimination [CMOS production]," in *IEEE Int. Workshop IDDQ Test. Dig. Papers*, Oct. 1996, pp. 9–13.

[26] Y. Wang, H. Luo, K. He, R. Luo, H. Yang, and Y. Xie, "Temperature-aware NBTI modeling and the impact of standby leakage reduction techniques on circuit performance degradation," *IEEE Trans. Depend. Sec. Comput.*, vol. 8, no. 5, pp. 756–769, Sep. 2011.

[27] Z. Tokei, P. Roussel, M. Stucchi, J. Versluijs, I. Ciofi, L. Carbonell, G. P. Beyer, A. Cockburn, M. Agustin, and K. Shah, "Impact of LER on BEOL dielectric reliability: A quantitative model and experimental validation," in *Proc. IEEE Int. Interconnect Technol. Conf.*, Jun. 2009, pp. 228–230.

[28] K. Croes, D. Kocaay, I. Ciofi, J. Bömmels, and Z. Tökei, "Impact of process variability on BEOL TDDB lifetime model assessment," in *Proc. IEEE Int. Rel. Phys. Symp.*, Apr. 2015, pp. BD.5.1–BD.5.5.

[29] D. Appello, P. Bernardi, A. Calabrese, G. Pollaccia, S. Quer, V. Tancorre, and R. Ugioli, "Parallel multithread analysis of extremely large simulation traces," *IEEE Access*, vol. 10, pp. 56440–56457, 2022.

[30] F. Angione et al., "Test, reliability and functional safety trends for automotive system-on-chip," in *Proc. IEEE Eur. Test Symp. (ETS)*, May 2022, pp. 1–10.

[31] M. L. Bailey, J. V. Briner, and R. D. Chamberlain, "Parallel logic simulation of VLSI systems," *ACM Comput. Surv.*, vol. 26, no. 3, pp. 255–294, Sep. 1994, doi: 10.1145/185403.185424.

[32] (2022). *SQLite*. Accessed: Nov. 21, 2022. [Online]. Available: https://sqlite.org

[33] M. Hahsler, M. Piekenbrock, and D. Doran, "dbscan: Fast density-based clustering with R," *J. Stat. Softw.*, vol. 91, no. 1, pp. 1–30, 2019.

[34] (2022). *SDL Libraries*. Accessed: Nov. 21, 2022. [Online]. Available: https://www.libsdl.org

[35] (2022). *OpenMP*. Accessed: Nov. 21, 2022. [Online]. Available: https://www.openmp.org

[36] STMicroelectronics. *SPC58NN84C3, 32-Bit Power Architecture MCU for High Performance Applications*. Accessed: Sep. 22, 2023. [Online]. Available: https://www.st.com/en/automotive-microcontrollers/spc58nn84c3.html

[37] F. Angione, D. Appello, P. Bernardi, C. Bertani, G. Gallo, S. Littardi, G. Pollaccia, W. Ruggeri, M. S. Reorda, V. Tancorre, and R. Ugioli, "A low-cost burn-in tester architecture to supply effective electrical stress," *IEEE Trans. Comput.*, vol. 72, no. 5, pp. 1447–1459, May 2023.

[38] (2022). *C++ Named Requirements: Trivially Copyable*. Accessed: Nov. 21, 2022. [Online]. Available: https://en.cppreference.com/w/cpp/named_req/TriviallyCopyable

[39] *STMicroelectronics. SPC564A80, 32-Bit Power Architecture MCU for Automotive Powertrain Applications*. Accessed: Sep. 22, 2023. [Online]. Available: https://www.st.com/resource/en/datasheet/spc564a80l7.pdf

[40] Cadence. *LEF/DEF 5.8 Language Reference*. Accessed: Sep. 22, 2023. [Online]. Available: https://web.archive.org/web/20210208081801/http://coriolis.lip6.fr/doc/lefdef/lefdefref/lefdefref.pdf

[41] *IEEE Standard for Verilog Hardware Description Language*, IEEE Standard 1364-2005 (Revision of IEEE Standard 1364-2001), 2006, pp. 1–590.

[42] R. Srivastava, N. Mudgil, G. Gupta, and H. Mondal, "SoC time to market improvement through device driver reuse: An industrial experience," in *Proc. Int. Symp. Electron. Syst. Design (ISED)*, Dec. 2012, pp. 56–61.

FRANCESCO ANGIONE (Graduate Student Member, IEEE) received the M.Sc. degree in computer and embedded systems engineering from Politecnico di Torino, in 2020, where he is currently pursuing the Ph.D. degree in system-level-test techniques for automotive SoCs with the CAD and Reliability Group. His main interests include real-time operating systems, computer architectures, and their dependability.

DAVIDE APPELLO received the degree in electronics engineering from Università di Pavia, Pavia, Italy. He is currently the Product-Engineer Director for the automotive digital products with STMicroelectronics, Agrate Brianza, Italy, where he is involved in testability and testing. He is active within TTTC and TTEP groups of the IEEE.

PAOLO BERNARDI (Senior Member, IEEE) received the M.S. and Ph.D. degrees in computer science, in 2002 and 2006, respectively. He is currently an Associate Professor with Politecnico di Torino, where he is involved with the Electronic CAD and Reliability Research Group. His current interests include system-on-chip tests and reliability, especially in the direction of high-quality automotive devices. He is the General Chair of the Test Technology Educational Program (TTEP) and the Program Chair of the Automotive Reliability and Test (ART) Workshop held in conjunction with the International Test Conference. He was recently acting as the Topic Chair for the European Test Symposium (ETS), the Design and Diagnosis of Electronic Circuits Symposium (DDECS), and the International On-Line Test Symposium (IOLTS).

ANDREA CALABRESE (Member, IEEE) received the degree in computer science from Politecnico di Torino, in 2020, where he is currently pursuing the Ph.D. degree. His research interests include software optimization and parallel algorithms.

STEFANO QUER (Member, IEEE) received the M.S. degree in electronic engineering from Politecnico di Torino, Torino, Italy, in 1991, and the Ph.D. degree in computer engineering from the Ministry of University and Scientific and Technological Research in Rome, in 1996. He has been a Visiting Faculty with the Department of Electronic Engineering and Computer Science, University of California at Berkeley, Berkeley, CA, USA. He has been an Intern with the Advanced Technology Group, Synopsys Inc., Mountain View, CA, USA, and the Alpha Development Group, Compaq Computer Corporation, Shrewsbury, MA, USA. He has also been a Compaq Computer Corporation Consultant. He is currently a Professor with the Department of Control and Computer Engineering, Politecnico di Torino. His main research interests include systems and tools for CAD for VLSI, formal methods for hardware and software systems, and embedded systems. Other activities focus on the development of sequential and concurrent algorithms and on optimization techniques able to achieve acceptable solutions with limited resources.

MATTEO SONZA REORDA (Fellow, IEEE) received the master's degree in electronics and the Ph.D. degree in computer engineering from Politecnico di Torino, Italy, in 1986 and 1990, respectively. He is currently a Full Professor with the Department of Control and Computer Engineering (DAUIN), Politecnico di Torino. He has published more than 400 papers in the area of test and fault-tolerant design of reliable circuits and systems. He is involved in numerous research projects with companies and other research centers worldwide. He received several best paper awards at major international conferences.

VINCENZO TANCORRE received the B.S. degree in electrical engineering from Politecnico di Bari. He is currently a Design-to-Test Engineer with the Testing Technology Center, STMicroelectronics. His research interest includes test-related process monitoring using diagnostic solutions for memories and unstructured logic.

ROBERTO UGIOLI received the degree in electronic engineering from Politecnico di Torino, in 1990. He has 30 years of experience in design for testability and silicon validation of ASIC devices. Since 2018, he has been with the STM Automotive Division, STMicroelectronics, as a Senior Hardware Product Engineer in charge of system level test equipment for volume production tests of ADAS devices.

Open Access funding provided by 'Politecnico di Torino' within the CRUI CARE Agreement