Automating the Generation of Programs Maximizing the Repeatable Constant Switching Activity in Microprocessor Units via MaxSAT

Nikolaos I. Deligiannis[†], Tobias Faller[*], Riccardo Cantoro[†], Tobias Paxian[*], Bernd Becker[*], Matteo Sonza Reorda[†]

***Abstract*—Throughout device testing, one key parameter to be considered is the switching activity (SWA) of the circuit under test (CUT). To avoid unwanted scenarios due to excessive power consumption during test, in most cases the SWA of the CUTs must be retained to a minimal value when the test stimulus is applied. However, there are specific cases where the opposite, namely the SWA maximization within the CUT, or a certain sub-module of it, can be proven beneficial. For example, during dynamic Burn-In testing we aim at maximizing the internal stress by applying suitable stimuli. This can be done in a functional manner by following the Software-Based Self-Test paradigm. However, generating such suitable programs represents a costly and arduous task for the test engineers. We consider the case where the CUT is a pipelined processor core and we aim to maximize the SWA of certain core sub-modules. We present a comprehensive methodology based on formal methods, able to automatically generate the best two-instruction stress-inducing sequence for the targeted processor module. The generated stimulus is composed of a short, arbitrarily-long repeatable sequence of a pair of assembly instructions, thus guaranteeing the maximum possible constant switching activity. The proposed method was applied to the OpenRISC 1200 and the RI5CY (PULP) processor cores demonstrating its effectiveness when compared to other methods. We show that the time for generating the best repeatable instruction sequence is limited in most cases, while the generated sequence can always achieve a significantly higher repeatable and constant SWA than other solutions.**

***Index Terms*—Switching Activity Maximization, Formal Methods, Microprocessor, Burn-In, Stress Test**

I. Introduction

With the continuous technological growth characterizing our times, semiconductor companies and manufacturers are in an ever-growing need for robust and reliable circuits. The test engineers are tasked with the non-trivial identification and development of the appropriate test procedures able to meet the imposed reliability standards while also maintaining a viable cost. The whole set of test steps that is usually employed by the semiconductor industry at the end of the manufacturing process may include Burn-In test (BI) [1]. In the domain of safety-critical applications, where the safety evaluation process and criteria have to obey the respective standards (e.g., ISO 26262 for automotive, DO 254 for avionics, IEC 62304 for medical equipment), BI testing is always present.

During BI the device under test (DUT) is operating at high temperature, power and frequency conditions as it is subject to different type s of external and **internal** stress in order to artificially age it. In this way any weak component evolves into an observable defect that can be detected (and the corresponding device thrown away). BI testing contributes to the reliability of the final product since it is the prime countermeasure against the phenomenon better known as *Infant Mortality* [2], where a significant amount of defective products is observed in their early operational life stages due to latent defects. Another testing method that is typically employed by the semiconductor industry during the devices' development cycle is Highly Accelerated Life Testing (HALT) [3]. HALT is not a standard pass-or-fail testing method but a *test-to-fail* method (i.e., the product is tested until failure). In this way, crucial information about the weaknesses and the limitations of the devices' can be derived in a short period of time and taken into consideration by the respective device designers [4]. In other words, all defects are analyzed until the root cause of the failure is found. It is a stress-test procedure that, similar to BI, uses environmental/external (e.g, extreme temperatures, rapid thermal transitions, repetitive shock vibrations) and **operational**/internal (e.g., stress inducing stimulus) stress to expose such weak points (if any) in a product's design. Furthermore, another stress inducing stress test procedure that is commonly employed by semiconductor manufacturers is High-Temperature Operation Life (HTOL) testing [5]. The goal of HTOL testing (also called Extended BI-testing in the literature) is to determine the intrinsic reliability of the devices under test (namely, to age the device such that a short experiment will allow the lifetime of the IC to be predicted). In order for the DUTs to age in such a manner, they are stressed at an elevated temperature, high voltage and dynamic operation for a predefined period of time. Although the test duration time is typically defined by the test engineers, an industry good practice calls for 1,000 hours of test time.

In all of the aforementioned stress test procedures, stress inducing stimuli can be proven beneficial for a more efficient stressing, and thus aging, of the DUTs. However, up until recently, the most commonly applied BI procedure was *static BI*, during which the DUTs are stressed at a fixed and elevated temperature for an extended period of time without any application of functional stimulus during the test. A major

N. I. Deligiannis, R. Cantoro and M. Sonza Reorda are with the Department of Control and Computer Engineering, Politecnico di Torino, Turin, Italy.
T. Faller, T. Paxian and B. Becker are with the Institute of Computer Science, University of Freiburg, Freiburg, Germany.

drawback of this approach is that nets of the circuit are not exercised [1]. Furthermore, as the devices' feature size continues to scale down and their structural and architectural complexity increases, so does the complexity and the cost of the BI test, making it unaffordable. Moreover, tuning the key parameters, such as the test duration, temperature and voltage, is becoming more and more difficult. In fact, for each new technology a non-negligible time period is required in order for the technology to mature and for the appropriate BI-test parameters to be identified (e.g., temperature and voltage), while at the same time, process variations introduce a notable amount of uncertainty. Any error or miss-tuning of the BI parameters can have catastrophic consequences since under the wrong conditions, the test can damage the DUTs and thus, result in a yield loss. For example, the rise of the junction temperature of the devices during BI testing can result in an increase in the leakage currents, which can cause thermal runaway. Regarding the test application time, all these forms of accelerated aging methods are very time consuming since their duration can be in the order of tens or hundreds of hours (especially for new technologies) and thus, they can become a bottleneck for the whole manufacturing process.

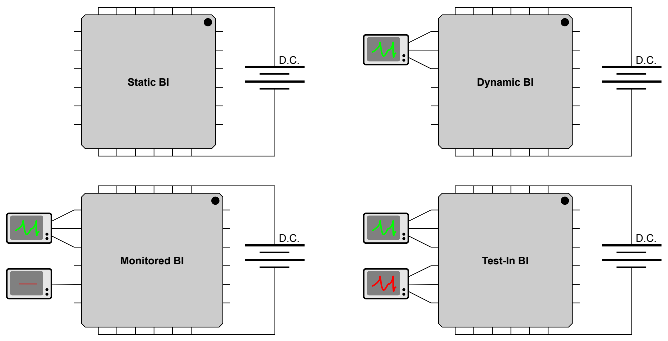

Fig. 1. Types of Burn-In testing

To amend these obstacles, BI is evolving into new forms [6] where the stress is generated and induced to the DUTs with less dangerous and more controllable actions by also resorting to internal stress [7]. In specific, as depicted in Fig. 1, there are several types of BI that rely on the application of internal stress on the DUTs spanning from simple stimuli application to the DUT inputs (Dynamic BI) up to output capturing (Monitored BI) and full evaluation of the DUT responses (Test-In BI). It has been proven that this approach can drastically reduce the test application times without affecting the reliability and the quality of the test [8]. If the DUTs are equipped with Design for Testability (DfT) infrastructures (e.g., scan chains), internal stress can be easily generated by relying on them. While this is true, in such an approach the DUTs would operate in test mode and thus, they would age in a way that would be different than in operational mode. Hence, this approach to the problem can possibly introduce unnecessary test escapes or even overstress the DUTs and potentially cause yield loss. On the other hand, by using functional stimuli to stress the DUTs, it is not possible to cause damage on the devices unless there exists a fatal design flaw since the circuits are executing code in a manner they are destined to do so. Furthermore, it has been shown [9], that creating proper temperature gradients between different parts in a circuit (or cores in a SoC) may enable the detection of defects that could hardly be targeted in other ways. Evidently, we can conclude that it is important to devise strategies able to generate purely functional test stimuli that maximize the switching activity (SWA) while the DUT works in normal mode [10, 11].

Inducing stress in DUTs in an internal manner, i.e., maximizing the switching activity either in the whole circuit or in some parts of it, can be also proven beneficial in other test steps and not solely in BI-testing. It has been observed that with the increasing of temperature, the delays on circuits also increase, so that high temperature aggravates the impact of delay defects in the circuits, easing their detection. Although this is a known problem, the most common approach to alleviate this issue is based on the generation of stimuli based on DfT infrastructures. Thus, as mentioned earlier the tests may include non-functional patterns that are not representative during an operational scenario. In [12], the researchers propose a temperature-aware software-based self-test (SBST) method in order to self-heat processors to a high temperature range and then to effectively test for delay faults in the circuit. Similarly, in [13] an instruction-based self-test approach is presented based on integer linear programming, in order to test a processor circuit for delay faults after increasing its temperature to the maximum operational levels. More recently, the adoption of System Level Test (SLT) to detect defects that could escape all the traditional test steps enables the search for functional stimuli able to produce particularly stressful conditions. Once again, this is possible by maximizing the internal gate switching activity and/or by creating temperature gradients between different modules within the DUT [14, 15].

In this paper we define the problem of repeatable constant SWA maximization, focusing on the case where the DUT is a fully pipelined processor. We propose an algorithm based on formal methods that automatically generates ***repeatable*** pairs of stress-inducing assembly instructions. These pairs are characterized by the property that they leave the processor in the same state they started from, hence being repeatable. Moreover, they induce at each clock cycle a constant amount of stress. The proposed method requires the gate-level description of the processor along with the respective technology library and a list of constraints. The generated pairs of instructions can be used to construct a test program by concatenating an arbitrarily long number of pairs. The resulting test program is guaranteed to induce the highest possible switching activity within a targeted processor module in a ***constant*** (hence, each instruction when executed forces the same number of nets to perform a logic switch) and **repeatable** manner (hence, for an arbitrarily long time period). With respect to other solutions, the generated sequences of stress-inducing assembly instructions are not only repeatable but guarantee that the induced constant switching in the processor unit is the maximum possible (unlike for example in [16]). Hence, we can use the generated sequences to construct test programs in order to force the DUT temperature always in a controlled manner and can be used e.g., to achieve accelerated aging of the devices during BI-testing. We consider the general case where the mission profile is not known by the manufacturer. Later in the text, we discuss the case where a mission profile is available and how this affects the stimuli generation process. Note that this may differ with respect to other approaches where the goal of the stress program application is to toggle each net of the DUT at least once [17]. Also, switching every net of the DUT at least once is a goal that is already considered (in an implicit manner) in the general case of generating software test libraries for stuck-at faults. Furthermore, in industry, it is standard during BI-testing to incorporate a test program composed of a variety of different routines each for a specific purpose of the stress test. Hence, since the routines are typically interleaved with one-another our approach can be proven beneficial since such a stress sequence could cover a considerable amount of internal circuit net switching. Also, modern BI-testing routines are often composed of the application of both external and internal stress. By generating functional sequences that are able to stress the DUT in a constant and repeatable manner the tuning of the external stress parameters (e.g., the temperature), becomes an easier task for the test engineers since a stable thermal gradient is enforced via the application of such sequences to the DUT.

With respect to our previous work [11], in this paper we provide a comprehensive and in-depth explanation about the practical motivations for the stress test and the importance of functional stimuli being available in the test procedures. We also presented various use-cases in-line with the industry standards that can benefit from such test routines. We provide an extensive comparison with the state-of-the-art and with related works in the area, while underlining the differences with our proposed method. We define more accurately the problem and greater emphasis is given on the explanation of our approach and the employed stress metric. Also, more experimental results are included regarding additional functional units that we used as stress targets (e.g., the load and store unit) as well as another processor of the RISC-V family.

The rest of the paper is organized as follows; in Section II we provide some background by elaborating on previous works in the area of the SWA maximization in ICs. In Section III we present our proposed method. In Section IV we report about the experimental setup and the results we gathered. Lastly, in section V we draw some conclusions.

II. Related Works

In this section, we discuss previous works around the subject of the SWA maximization in two categories distinguishing between combinational and sequential circuits.

A. Combinational Circuits

The reliability of integrated circuits is strongly linked with the problem of the identification of the maximum current consumption during the devices' testing phases. The research community has studied the problem extensively in the past. For instance, in [18] the authors address the problem of the maximum current estimation in MOS ICs. They state that unrestricted voltage drops in the circuits power and ground lines can lead a system to dysfunction and cause a degradation in the switching speed. Such current estimates play a crucial role in the accurate timing analysis of the circuit and can further be used to enhance the reliability of the circuit's power and ground buses. In [18] the authors also propose a heuristic approach that can be applied to small combinational IC blocks in order to identify their maximum static and transient current by dividing the circuit into groups of combinational interconnections of logic gates and acting on each group independently. In [19], the authors approach the same problem, namely the estimation of the maximum current in CMOS ICs, by modeling it as an optimization problem. In that sense, they search for stimuli (i.e., test patterns) that maximize the circuit's current value waveform. They further state that identification of such estimates plays a vital role in the reliability and the performance of the circuits since such knowledge can be used to enhance the circuit's design, rendering it resilient towards soft errors and overheating scenarios. In [20] the authors propose algorithms for probabilistically estimating the average switching activity in combinational circuits of moderate size. They link the switching activity of a circuit with estimates that concern the circuits power and heat dissipation.

The authors of [21], further highlight the importance that accurate power consumption estimates have for the performance and the reliability of VLSI chips. More importantly, they state that in order to obtain such accurate estimates, one has to identify a pair of **two** consecutive input patterns (test vectors) which induces as many logical switches within the device as possible. That is, to maximize the circuit's switching capacitance. Moreover, they state that the complexity of such an exhaustive search for the identification of the appropriate pattern pair on a combinational circuit with n primary inputs is $\mathcal{O}(4^n)$. Their approach in order to compute such power estimates is based on an automatic test generation technique, while also a Monte-Carlo methodology is described. Although the proposed method is effective, it is applied to relatively small-sized combinational circuits. Likewise, in [22], the authors approach the problem of power dissipation in combinational CMOS ICs by relying on formal methods. They model the circuit power dissipation as a Boolean function of the circuit's primary inputs. They as well link the power estimation problem to the identification of the appropriate pair of two consecutive test vectors that maximizes the gate switching of the device. Their solution allows a reduction of the problem to a weighted max-satisfiability (MaxSAT) problem. But, once again, the method was applied to small combinational circuits due to the high complexity imposed in order to obtain the objective function of the circuit and to optimize it.

Overall, the maximization of the SWA of a circuit can be proven beneficial from a designer's perspective, since it allows for the extraction of important information that enables the enhancement of the overall reliability of the devices by fine-tuning certain design parameters. However, the SWA maximization can be advantageous in the context of device testing as well. During the multiple test steps that are adopted during the device manufacturing, it may happen that faults do not manifest themselves during testing and escape the whole set of test steps. These latent defects are the prime suspects behind the Infant Mortality behavior which is resolved via BI. In

[23] the authors present a probabilistic approach (i.e., random excitation of internal nodes) to generate input patterns that maximize the SWA of a combinational block to further achieve the maximization of power dissipation during BI testing. In [24] the author relies on a genetic algorithm to develop a method for the maximization of a combinational circuit's SWA in order to also maximize the circuit's heat dissipation during BI testing.

Having acknowledged the importance and the benefits that stem from the SWA maximization on ICs and the acquisition of switching information of a circuit in general, the researchers focused deeper into the SWA maximization problem and proposed various methodologies to effectively solve it. In [25] the authors present a technique to extract estimates about the maximum switching activity of a combinational circuit's nets while also considering delays within the circuit. Although the method is applied solely to combinational circuits the authors claim that such an approach can be also employed in sequential circuits by isolating the combinational blocks and applying the method in a divide-and-conquer fashion. In [26] a methodology based on ATPG tools is presented in order to identify the pair of test vectors that maximize the weighted switching activity of a combinational IC. This is achieved by modifying the gate-level description of the circuit appropriately and then by generating vectors while targeting a selected set of faults on the modified netlist while considering a variable delay. In [27] the authors present a simulation-based approach for the identification of the appropriate combination of two test vectors for combinational ICs that maximize the switching activity and thus the power consumption of the circuit. Although highly effective, the generated pairs of test-vectors for a combinational IC block with large number of inputs cannot be proven to be the absolute best due to the exhaustive simulations that are needed for an exponentially growing number of vectors. In [28], another approach based on formal methods is presented targeting combinational circuits. The authors present an integer linear-programming approach that utilizes SAT solvers as an underlying technology in order to effectively compute the maximum weighted switching activity of combinational ICs that can be used to derive information about the circuits' power dissipation.

In general, the subject of the SWA maximization on combinational circuits has been extensively studied in the past. A variety of solutions have been proposed although they are typically applied to moderate sized circuits.

B. Sequential Circuits

Regarding sequential circuits, and specifically processors (or processor cores), the authors of [29] propose a method based on formal methods for the generation of patterns that maximize the SWA on various parts of the processor in a uniform manner by isolating the functional units of the core and encoding them in Conjunctive Normal Forms (CNFs). By considering each unit separately they define in a flexible manner simpler functional constraints (i.e., transitions maximization) per unit and evaluate the satisfiability of each formula separately. This enables the construction of functional stimuli that can be used during BI to assist the aging process of various parts of the circuit.

In [10] the authors present a methodology based on evolutionary techniques, that builds stressful assembly programs for a target processor by extracting characteristics (i.e., code segments) that were found to be causing high switching activity when executed. These segments in turn are used to compose a final stress program that gets refined during the evolution process and is able to induce high stress within the functional units of the core. In [16] we present another method based on evolutionary techniques, which is able to generate from the ground-up (i.e., without dependencies to pre-existing code) assembly programs that induce high stress amounts within specific units of the processor in a repeatable manner. Although experimental results prove the effectiveness of the method, there is no guarantee that the generated programs are the best possible ones.

In [30] the authors, while targeting a 32-bit processor intended for mission-critical usage, present a comprehensive methodology and propose metrics for the comparison and the evaluation of stress procedures that are applied to the circuit during BI. The processor, is used in two variants in their case study. On the one hand, the core is equipped with DfT infrastructures (e.g., scan) while on the other hand it is not. Their experimental results demonstrate that while the processor with scan is being stressed, a uniform elevation of its temperature is observed inside the various modules. On the other hand, when functional stimuli (i.e., stress programs) are applied at-speed to the processor, a much higher thermal activity is observed in the respective processor unit. The authors conclude that a purely functional code applied at-speed to the DUT has a notably better performance in terms of induced stress and thus, render it more suitable than DfT-based approaches such as scan.

To summarize, the SWA maximization problem has been approached with a variety of methodologies in the past both for combinational and sequential circuits. In the present paper, while covering the case where the DUTs are sub-modules of a pipelined microprocessor, we aim to generate stimuli that maximize the repeatable constant switching activity within the targeted module of the core. In this case, the generated functional stimuli correspond to assembly programs. We propose an algorithm that optimally solves the problem by relying on formal methods. As mentioned in Section I, the generated sequences of instructions are guaranteed to be the best in terms of induced switching activity.

III. Proposed Method

Problem Definition

Various methodologies have been proposed in the past for the maximization of the SWA focusing primarily on combinational circuits and assuming the full control of their inputs i.e., the value of every input can be freely selected. Our goal is to propose an algorithm that given the gate-level description of a pipelined processor as input, generates functional stimuli to effectively stress any module within it. More precisely, we aim to identify a pair of two instructions (or vectors) (I_1, I_2) that are able to maximize the repeatable constant switching activity of a certain processor module (i.e., the processor module for which the test engineer is required to generate stress stimulus) when executed in sequence. Specifically, the sequence induces the maximum possible constant switching activity within the module. This means that each instruction of the two-vector pair is able to maximize the number of logical switches performed from the nets of the targeted module from High to Low (HL) and from Low to High (LH) manner when they are applied in sequence. Low corresponds to the logic value 0 and High to the logic value 1. More specifically, that means that if the application of I_1 forces n nets to toggle, then I_2 forces the same n nets to make the inverse transition.

Note that we wish to be able to maximize the constant switching activity i.e., induce the same stress amounts in the processor unit for an arbitrary long time period (in clock cycles); hence we require the generated sequence of instructions to be a repeatable one. Let's assume that for a specific processor module T the optimal sequence $\widetilde{s}$ of two instructions has been found. We further assume that the whole processor has been initialized properly, either via the activation of its *RESET* signal or via the execution of a synchronization sequence. The initialization step is vital, since the underlying solver is ignorant about the architectural constraints of the pipeline (e.g., a boot address must be set before starting the execution of code in the program counter bits). Thus, if the initialization step is not done, in an attempt to satisfy the formula the solver may end up with an assignment, which although satisfies the CNF, forces the processor to a non-functional state. Hence, after the initialization phase, the core is driven to a well defined and legal functional state s^a. The first of the two stress inducing instructions of the sequence $\widetilde{s}$, drives the processor to a new state s^b. Lastly, the second instruction drives the processor to the same initial state s^a in order for the generated sequence $\widetilde{s}$ to be a repeatable one. By satisfying the aforementioned equivalence that guarantees a repeatable sequence to be generated, we can maintain a maximum gate switching activity within the processor module T for an arbitrarily long period of time. This sequence of state transitions for the processor can be interpreted as a finite state machine as depicted below in Fig. 2.

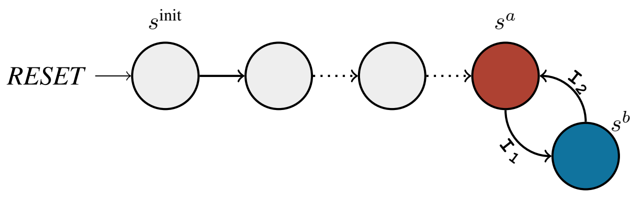

Fig. 2. FSM representation of the SWA maximization process for a processor module

Assuming that the optimal stress-inducing pair of instructions ($\widetilde{s}$) has been generated, it can be used to stress the processor module T (e.g., to create a hot-spot within the core) in a constant manner. This can be achieved by repeating the sequence $\widetilde{s}$ for as many times as required (e.g., $\widetilde{s}, \widetilde{s}, \widetilde{s}, \ldots, \widetilde{s}$). After the repetition of the generated pair of instructions for a sufficiently large number of times, an unconditional jump instruction can be issued to transfer the code execution back to the start of the stress segment. The loop can be forcibly stopped e.g., by issuing an interrupt call.

Stress Evaluation

In order to measure the effectiveness of the generated instruction sequences in terms of stress induced to the target processor module we use the average induced stress percent metric. Given that the generic target processor module T consists of m nets and the generated sequence $\widetilde{s}_n$ is composed of n instructions, we calculate the average induced stress percentage as:

$$\overline{stress_{\%}} = \frac{\sum_{i=1}^{m}[HL(i) + LH(i)]}{n \times m} \times 100 \tag{1}$$

The numerator of the fraction represents the total amount of HL and LH transitions that were performed by every net out of the m existing in the target processor module T. The denominator of the fraction represents the maximum achievable value, corresponding to the case where every net of the module makes a transition (either from HL or LH) when every instruction of the sequence $\widetilde{s}_n$ is executed. For example, assuming a sequence of 2 instructions, then the maximum achievable value would be $2 \times m$, where the first instruction forces all nets to switch and the second instruction forces the same m nets to perform the inverse transitions. When considering a pipelined processor, in a first phase we assume that every instruction of the sequence is executed by the target module over 1 clock cycle, unless it is stated otherwise. We will relax this assumption later. Obviously, the denominator value of the fraction should be interpreted as a theoretical maximum, which can never be achieved in practice, e.g., due to the presence of uncontrollable lines within the module T [31]. Most importantly, it is clearly not given that all nets can be toggled in the same clock period in a functional manner.

For example, let us consider the two combinational circuits depicted below in Figure 3: a 1-bit Full-Adder (FA) at the top and a 3-bit message odd parity checker below. Further, let's assume that they are initialized in such a manner that all nets hold the logic value *0*. For the case of the FA one optimal sequence that maximizes the consecutive gate switching is $\widetilde{s}_{FA} :=$ <I_1,I_2> = <101,000>. This sequence forces 3 out of 5 nets to toggle and there exists no pair of vectors that when applied to the FA circuit's inputs can force a higher number of HL and LH transitions than 3. On the other hand, considering the parity checker, there exists a sequence that forces all nets to toggle e.g., $\widetilde{s}^{odd}_{parity} :=$ <I_1,I_2> = <010,000>.

Previously, we have generalized the problem for the generation of a stress inducing sequence composed of N instructions. Let us now consider a processor module T and a sequence of N=*2* instructions $\widetilde{s} = (I_1, I_2)$ and assume that it induces the maximum possible repeatable and constant SWA within the processor module T ($SWA^{max}_{\widetilde{s}\to T}$) when executed, i.e.,

$$s^a \xrightarrow[\text{clk}\uparrow]{\mathrm{I}_1} s^b \xrightarrow[\text{clk}\uparrow]{\mathrm{I}_2} s^a \tag{2}$$

The term $SWA^{max}_{\widetilde{s}\to T}$ can be broken down as the sum of $SW_{I_1\to I_2}$ that represents the total number of nets switching

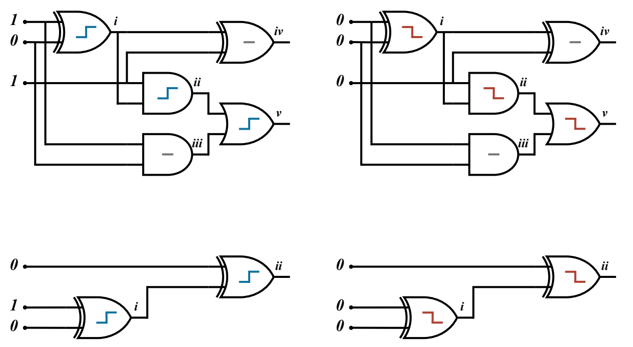

Fig. 3. 1-bit full adder (top) and 3-bit odd parity checker (bottom) maximum switching for sequences $\widetilde{s}_{FA}$ and $\widetilde{s}^{odd}_{parity}$, respectively

when I_2 is executed after I_1 plus the number $SW_{I_2 \rightarrow I_1}$, which in turn represents the total number of nets switching when I_1 is executed after I_2. Since the sequence $\widetilde{s}$ is repeatable and induces a constant SWA (as mentioned earlier), a symmetric switching is implied, i.e., if the first transition forces a certain net of the module to switch, then the second transition will force the same net to switch again (as depicted also in Fig. 3). Thus it holds that $SW_{I_1 \rightarrow I_2} = SW_{I_2 \rightarrow I_1}$. In other words, maximizing the SWA can be translated to finding the right pair of instructions that induces the highest possible constant gate switching when SW_{I_2} is processed after SW_{I_1} (or vice-versa). The sequence $\widetilde{s}$ that maximizes the SWA can be repeated for a generic number of times N, yielding a total of $2 \times N$ instructions (e.g., $N = 2 : I_1, I_2, I_1, I_2$).

Run-time Analysis

The identification of the instruction pair that leads to the repeatable constant maximization of the SWA of a processor module is a non trivial task, with an intricacy that depends on the size and characteristics of the sub-module we are targeting.

For example, let us consider as a target the 32-bit adder of the processor's arithmetic and logic unit. In this case, the search space of the problem can be shrunk significantly, since we know beforehand which instructions have to be considered in the search, namely, the `add` instructions of the processor's ISA. Thus, the problem can be reduced to the identification of the appropriate pairs of operands that maximize the SWA of the adder unit.

On the other hand, assuming that the stress target is the processor's instruction decode unit then the algorithm should not only consider potential operands like in the case of the 32-bit adder, but a combination of instructions and operands in order to generate a stress effective instruction sequence. In fact, the search space should include the majority of the set of instructions supported by the processor's ISA. So, for the case of the decoding unit, the task is harder since the search space is considerably larger than in the case of the adder.

For example, assuming that there is only one integer addition assembly instruction in the ISA (to simplify) then the search space (S_{adder}) can be described as a set given by the cartesian product:

$$\begin{aligned} S_{adder} &= S_{instruction_1} \times S_{instruction_2} \\ S_{instruction_1} &= I_{N-bit} \\ S_{instruction_2} &= I_{N-bit} \end{aligned}$$

where I_{N-bit} is the set of all N-bit integers (e.g., if the processor supports 32-bits registers then this would be the set of all 32-bit integers). In the case of the decoder being the stress target, then the search space ($S_{decoder}$) can be described as a set given by the cartesian product:

$$\begin{aligned} S_{decoder} &= S_{instruction_1} \times S_{instruction_2} \\ S_{instruction_1} &= S_{ISA} \times S_{op} \\ S_{instruction_2} &= S_{ISA} \times S_{op} \end{aligned}$$

where S_{ISA} is the set of all instructions supported in the processor's ISA and S_{op} is the set of all operands that are supported by every instruction of the ISA. It is clear that the size of the set $S_{decoder}$ is much greater than the size of the set S_{adder}.

Thus, as mentioned earlier, one can safely assume that the scalability of the method depends on the complexity of the targeted processor module. Also, the structure of the targeted module has an impact on the overall run-time of the method. This point will be addressed later in the paper.

Formal Methods Approach

Our approach to optimally solve the problem is based on formal methods. Specifically, we model the problem as a weighted maximum satisfiability (MaxSAT) problem. The reason for this selection is the fact that the problem of the maximization of the SWA of a processor's module can be seen as an optimization problem. The core idea is to enforce constraints that encode differences (transitions) between two consecutive clock cycles and thus, maximize the SWA over those two cycles. Then, the corresponding MaxSAT solver evaluates these constraints in order to determine the satisfiability of the CNF formula. Lastly, if the formula is satisfiable we extract the two input vectors that stress the maximum by examining the assignment found by the solver.

It is known that the problem of ATPG can be reduced to a SAT problem [32]. Based on the same principles, the first step towards the reduction of the SWA maximization problem to the SAT domain is to obtain the Boolean formula of the circuit in a CNF. This is achieved by first unrolling the circuit in time. During circuit unrolling the circuit is duplicated for a specified amount of times and then the circuit instances are appended to each other. More precisely, the pseudo primary inputs (PPIs) of the n^{th} instance of the circuit are driven by the pseudo primary outputs (PPOs) of the *n-1*th instance of the circuit and so on. Every instance of the circuit in the unrolled format represents a state of the processor and corresponds to a clock cycle. These instances are called *timeframes* (TFs). Specifically, in our case, we steer clear of the large complexity imposed by the traditional sequential ATPG methods by knowing *a priori* the unrolling depth of the processor circuit, which is typically a small value. After circuit unrolling, the circuit is encoded

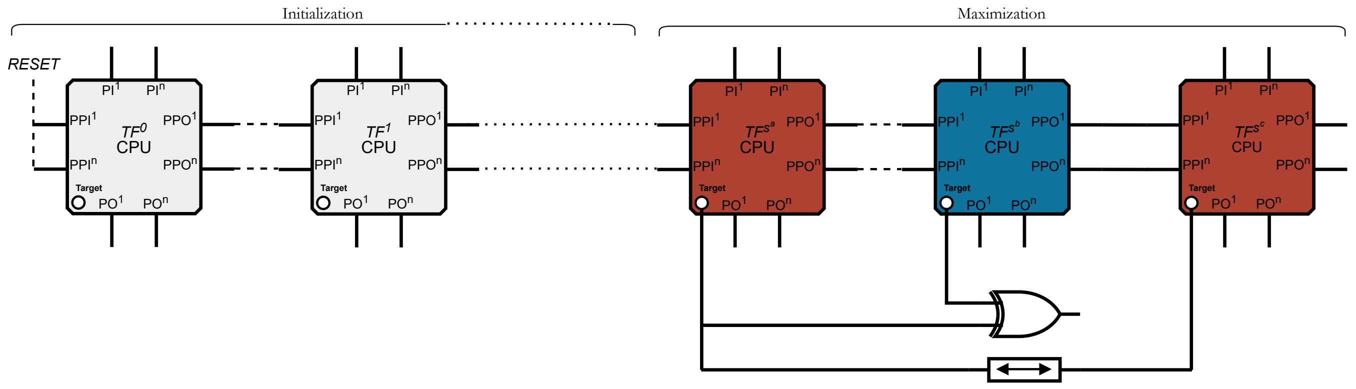

Fig. 4. MaxSAT Model

and its CNF formula is obtained via symbolic simulation in combination with the Tseitin transformation [33].

One crucial parameter to the accuracy and the effectiveness of the method is the unrolling depth, i.e., the total number of times the circuit has to be replicated. As shown in Fig. 4 (which can be correlated with Fig. 2) the unrolling depth is the sum of the initialization phase timeframes plus the maximization phase timeframes. We have previously explained that we begin from a valid and well defined initial state i.e, there are no Don't Care values in the pipeline and the processor state is a functional one. In our case, we assert the processor *RESET* signal and unroll the circuit for the minimum number of times that is required for the effects of the signal to take effect and propagate to the whole pipeline. This number is architecture specific, i.e., for a different processor a different number of initialization timeframes may be required and this information can be derived either from the specification of the architecture (if available) or by inspecting the RTL description of the DUT. Following the initialization phase, we must define the duration of the maximization phase i.e., the 3 states that we have previously mentioned in Eq. (2). For most of the processor units the duration (in terms of clock cycles) for each instruction is 1 clock cycle. Yet, there are specific cases (e.g., for the multiplication unit) where the instruction remains active for more than 1 clock cycle. Specifically, the exact number of timeframes required for the maximization phase can be derived from the Eq. (3) below:

$$TF_{max} = (2 \times duration) + 1 \tag{3}$$

As mentioned earlier, the parameter *duration* is architecture specific and represents the number of clock cycles that the instruction requires to be executed in the module of the pipeline that we intend to stress. Hence, we need twice the duty cycle duration of the module in order to fully account for the states s^a and s^b, respectively. Lastly, we need 1 extra timeframe in order to force the equivalence of the resulting state after the execution of the second instruction of the sequence with the initial state, i.e., $s^c \equiv s^a$.

Up until this point, we have enforced the constraint for the activation of the *RESET* signal in the very first timeframe of the unrolled circuit. In general, the constraints are encoded as clauses and are appended to the circuits CNF formula. In the MaxSAT terminology 2 types of clauses exist. The so called *hard* clauses, which must all be satisfied in order for the CNF to be evaluated as satisfiable and the *soft* clauses, whose satisfiability does not affect the satisfiability of the CNF formula. Each soft clause is also correlated with an integer weight value and so, the goal of the MaxSAT solver is to find an assignment that satisfies all the hard clauses of the CNF formula while maximizing the sum of weights for the satisfied soft clauses at the same time. In our work, we use a uniform weight distribution and thus, each soft clause is associated with the same integer value as a weight.

In Figure 4 we have an abstract representation of the proposed MaxSAT model. The initialization phase consists of the encoding of a hard clause on the core's *RESET* signal that corresponds to the signal activation. For every one of the following timeframes, clauses are inserted for the *RESET* signal in order to render it inactive and thus, to prohibit the solver to use it for the sensitization of lines in the targeted module. Although it is true that the activation of the *RESET* signal in certain cases can cause a significant amount of concurrent transitions, such behavior is abnormal and thus it must be prohibited in order to enforce a functional scenario within the pipeline. Moreover, we have previously introduced the idea of repeatable instruction sequences that maximize the constant SWA of a given module. In order to guarantee the repeatability, every net of the module must have the same value in TF^{s^a} and in TF^{s^c}. For this reason, every literal that encodes a net of the targeted processor module during TF^{s^a} ($l^{s^a}_{net_i}$), along with every literal that encodes a net of the targeted module during TF^{s^c} ($l^{s^c}_{net_i}$) are linked together by inserting the following logic implications as hard clauses to the CNF formula, which guarantee their equivalence during these two states:

$$\omega^{hard}: \; l^{s^a}_{net_i} \longleftrightarrow l^{s^c}_{net_i} \equiv (\neg l^{s^a}_{net_i} \vee l^{s^c}_{net_i}) \wedge (\neg l^{s^c}_{net_i} \vee l^{s^a}_{net_i})$$

Hence, given that the solver finds a satisfying assignment for the CNF formula, it is guaranteed that all of the nets of the module will hold exactly the same logic values during TF^{s^a} and TF^{s^c} and thus, the generated sequence is indeed repeatable. Besides the states' equivalence we also have to encode the corresponding switching of the module's nets between the timeframes TF^{s^a} and TF^{s^c}. This switching corresponds to soft clauses i.e., we request from the solver to satisfy as many switching transitions as possible. For every net *i* of the targeted processor module we encode a XOR gate on the CNF as hard clauses, whose inputs are the corresponding literals for the net *i* in TF^{s^a} and TF^{s^b}, respectively. In order for the switching requirement to take effect we further encode soft clauses ($l^{\oplus}_{\text{diff}_i}$) that corresponds to the output of the XOR gate and require it to be 1 as shown below:

$$\left.\begin{array}{lll} \omega^{hard}: & (\neg l^{s^a}_{\text{net}_i} \vee \neg l^{s^b}_{\text{net}_i} \vee l^{\oplus}_{\text{diff}_i}) & \wedge \\ & (l^{s^a}_{\text{net}_i} \vee l^{s^b}_{\text{net}_i} \vee l^{\oplus}_{\text{diff}_i}) & \wedge \\ & (l^{s^a}_{\text{net}_i} \vee \neg l^{s^b}_{\text{net}_i} \vee \neg l^{\oplus}_{\text{diff}_i}) & \wedge \\ & (\neg l^{s^a}_{\text{net}_i} \vee l^{s^b}_{\text{net}_i} \vee \neg l^{\oplus}_{\text{diff}_i}) & \wedge \end{array}\right\} l^{s^a}_{\text{net}_i} \oplus l^{s^b}_{\text{net}_i} \text{ in CNF}$$

$$\omega^{soft}: \ (l^{\oplus}_{\text{diff}_i})$$

The first four clauses correspond to the Tseitin transformation of the XOR ($l^{s^a}_{\text{net}_i} \oplus l^{s^b}_{\text{net}_i} = l^{\oplus}_{\text{diff}_i}$) gate in order to be represented as a CNF. The very last unit clause requests that the output of the XOR gate must be evaluated as 1. Thus, if satisfied it corresponds to a difference between the two literals that encode net *i* in TF^{s^a} and in TF^{s^b} i.e., the corresponding net performed a HL or a LH transition. Due to the state equivalence enforced by the aforementioned hard clauses, it is redundant to enforce another set of switching constraints for the target module's nets between TF^{s^b} and TF^{s^c} since it is already implied due to the equivalences enforced for TF^{s^c} and TF^{s^a}.

Assuming that the solver managed to satisfy the CNF, we can now extract the input vectors in the corresponding timeframes and compose the stress inducing sequence. In order to do so, we need to identify the literals that are used to encode the PIs of the module and extract the *0/1* logic values from the found model. For instance, considering the FA circuit of Fig. 3, we can extract the first vector $v_1 :=$ <1,1,0> from the first timeframe and the second vector $v_2 :=$ <0,0,0> from the second timeframe by examining the assignments of the corresponding input literals. In the general case, we can further extract and disassemble the *N*-bit instructions from the respective pipeline instruction register and use it in combination with the previously decoded vectors in order to compose an assembly program that effectively stresses the target processor module. In the general case, one can always use as a probing point the instruction bus of the processor and disassemble the N literals back to N-bit instructions (while always respecting the bit order).

Validity Checking

The constraints presented up until now, are the base constraints that can be generalized and applied no matter the targeted processor module. They are the core constraints that guarantee the SWA maximization and the repeatability attribute of the generated instruction sequence. But in almost all cases, further constraints have to be devised in order to circumscribe the search space in a more accurate manner. Most importantly, it has to be guaranteed that there are no violations within the pipeline (e.g., the generation of an illegal

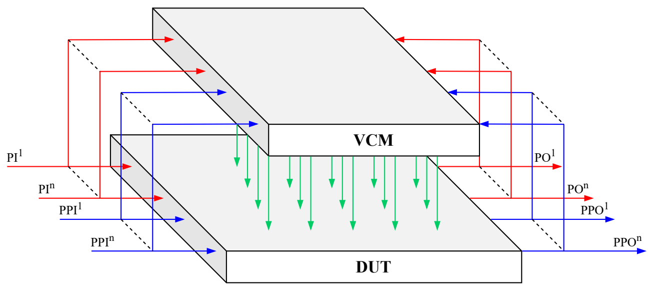

Fig. 5. Interaction of Validity Checker Module with the Device Under Test

instruction, a load or a store from the instruction memory, the generation of a misaligned address, etc.) in order for the generated sequences of instructions to represent valid and reachable states.

For these reasons, we employ the idea of a validity checking module (VCM). The VCM, as depicted in Fig. 5, is able to enforce constraints on the circuit under stress and has full access to its inputs (PIs/PPIs), outputs (POs/PPOs) and nets. It can be implemented either by directly acting on the encoded circuit's CNF, in terms of propositional logic, or it can be described in a hardware description language, synthesized and parsed along with the gate-level description of the processor [34]. Similar approaches have been also presented in the past [35, 36]. In our case all the VCMs, for every processor module considered as a stress target, were implemented in propositional logic, similarly to what is shown in Figure 6. They are primarily used in order to enforce valid instructions to be generated by the solver, to prohibit misaligned effective address generation, to prohibit the generation of exceptions and to disable the interrupts for the core. The constraints specifying allowed instructions are created manually from the processor's specification. Furthermore, the use of the VCM renders the consideration of a mission profile feasible during the stimuli generation process (given that the latter is available). In the context of processor testing, this may mean that a specific sub-set of instructions of the ISA are considered only; the VCM can be used to prevent the usage of the rest of the instructions and only allow the mission profile-mandated ones during the stimuli generation process.

Optimization: Literal D-Chaining

In this subsection we present an optimization that was introduced to the algorithm in order to improve the run-time of the method by "assisting" the solver to converge faster to the solution.

During the generation of the switching constraints (soft clauses), extra clauses are being added for each gate in the CNF that link the switching of the gate's inputs with it's output as shown in Figure 7. Specifically, we add backward implication D-chains [37] for every net of the targeted processor module. For example, let us assume a gate of the targeted module that consists of *n* inputs, with their respective literals being $d_1, d_2, ..., d_n$. By adding the following implication:

$$d_{out} \rightarrow (d_1 \vee d_2, \vee ... \vee d_n)$$

```
auto reset = GetDriver(reset_pin);
auto if_miss = GetDriver(if_miss_pin);
auto id_inst = GetDrivers(id_inst_pins);

// Reset and fetch miss constraints
// from 0 to max_tf timeframes.
AddConstraint(reset, 0, ONE);
for (int tf { 1 }; tf < tf_max; tf++)
{
 AddConstraint(GetLiteral(reset, tf), ZERO);
 AddConstraint(GetLiteral(if_miss, tf), ZERO);
}

// Constraining decoded instructions
// from decode start to max_tf timeframes.
for (int tf { tf_start }; tf < tf_max; tf++)
{
 vector<Literal> instructions {
  MatchLiterals(id_inst, tf, // Inst. 1
   "XXXXXXXXXXXXXXXXXXXX␣XXXXX␣0110111"),
  MatchLiterals(id_inst, tf, // Inst. 2
   "XXXXXXXXXXXXXXXXXXXX␣XXXXX␣0010111"),
  MatchLiterals(id_inst, tf, // Inst. 3
   "XXXXXXX␣XXXXX␣XXXXX␣000␣XXXXX␣1100011"),
  // ... (for the rest of instructions)
 };
 AddConstraint(AnyOf(instructions), ONE);
}
```

Fig. 6. Simplified excerpt of the C++ VCM

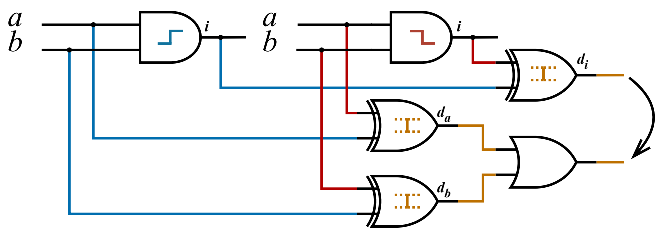

Fig. 7. Difference literals added to an AND-Gate

we state that the respective gate's output can only toggle if **at least one** of the inputs also toggles. Although the generation of the D-chains requires extra effort by extending the CNF formula, it has been shown to drastically reduce the solver runtime complexity.

Algorithm summary

Figure 8 depicts the proposed algorithm in a compact pseudo-code format. The algorithm takes as input a triplet, namely the synthesized, gate-level description of the processor, the targeted processor module (i.e., the unit we aim to generate stressful stimuli for), and lastly the unrolling depth. The unrolling depth parameter may vary and strongly depends on the targeted processor but also on the specific module within the processor, as we have previously explained in the text. For example, assuming that the targeted processor module requires 2 clock cycles for the *RESET* signal propagation and 2 clock cycles to finish the processing of an instruction, then the unrolling depth for this use case shall be $TF_{rst}+TF_{max} = 2+(2\times 2)+1 = 7$ timeframes in total.

```
input  : A triplet $(G,M,ud)$ where
         $G$ is the gate-level description of the processor
         $T$ is the target module we intend to stress
         $ud$ is the unrolling depth i.e.,
         the number of timeframes to be used
output : Two instructions $I_1, I_2$ maximizing the SWA of $T$
1  CNF ← UnrollAndEncodeCircuit($G$,$ud$)
   // Initialization of the Whole Processor
2  ActivateResetAt(CNF, $TF^0$)
   // Maximization Clauses and State Equivalence
3  foreach net $n_i \in T$ do
4      AddDifferences  (CNF, $n_i$, $TF_{s^a}$, $TF_{s^b_{max}}$, $weight$)
5      AddImplications (CNF, $n_i$, $TF_{s^a}$, $TF_{s^c_{max}}$)
6      AddImplications (CNF, $n_i$, $TF_{s^c_{max}}$, $TF_{s^a}$)
7      AddD-Chains     (CNF, $n_i$)
8  end
   // VCM Implementation for the Target Module
9  ConstraintsFromVCM(CNF,$T$)
10 MaxSolve(CNF)
   // Extract Operands and Disassemble them to Instructions
11 $I_1$ ← ExtractOperandsFrom(CNF, $T$, $TF_{s^a}$)
12 $I_2$ ← ExtractOperandsFrom(CNF, $T$, $TF_{s^b_{max}}$)
13 return $I_1$, $I_2$
```

Fig. 8. Instruction sequence generation routine

IV. Experimental Setup and Results

Phaeton [35] is used as an underlying framework to model the SWA maximization problem considered in this paper. Originally, Phaeton was designed to identify sensitizable paths and generate test pairs to exercise these paths using SAT-solving. Phaeton supports a large number of models and sensitization conditions and provides a generic interface that can be used by different applications. Due to a number of elaborated speed-up techniques, Phaeton scales to industrial circuits, as demonstrated in experimental evaluations on numerous differing applications [38]. We implemented a prototypical tool inside the Phaeton framework (accounting for approximately 2,000 lines of C++ code) for the algorithm described in Fig. 8.

The experiments were performed on a machine using an Intel i9-9900 processor running at 3,10GHz. For every sequence generated for the targeted modules within the considered processor cores, a logic simulation environment was set up using QuestaSIM by Mentor Graphics. Specifically, during the simulation of the generated stress inducing sequences, the targeted processor module was isolated and a toggling check was performed for **every** clock cycle. The reported results from every processor module were then aggregated and post-processed in order to calculate the stress induced via Equation (1).

The first processor that we used is OpenRISC 1200 (OR1200) [39]. OR1200 is a 32-bit scalar RISC processor using the Harvard micro-architecture. It is mainly intended for embedded, portable and networking applications. The processor's RT-Level Verilog description was synthesized using the Silvaco 45nm Open Cell Library [40] using Design Compiler by Synopsys.

The second processor we considered is the RISC-V (RV) processor RI5CY [41]. RI5CY is a 4-stage in-order 32-bit RISC-V processor core. The ISA of RI5CY was extended to support multiple additional instructions including hardware

TABLE I
EXPERIMENTAL RESULTS

Processor	**Stress Program Generation Approach**	**Average Induced Stress**			**CPU Generation Time**		
		Adder	Decoding Unit	Load Store Unit	Adder	Decoding Unit	Load Store Unit
OR1200	Formal Techniques (MaxSAT)	82%	91%	65%	3^{sec}	15^{min}	8^{sec}
	Stuck-At Test Program	24%	44%	57%		-	
RI5CY	Formal Techniques (MaxSAT)	76%	36%	34%	5^{sec}	14^{min}	7^{sec}
	Stuck-At Test Program	73%	30%	32%		-	

loops, post-increment load and store instructions and additional ALU instructions that are not part of the standard RV ISA. RI5CY has become a popular core for a huge variety of applications and especially for IoT designs. The processor's RT-Level SystemVerilog description was once again synthesized using the Silvaco 45nm Open Cell Library using Design Compiler by Synopsys.

For both cores, the units that we targeted to generate stress inducing stimuli for are:

(i) the 32-bit Adder of the processors' ALU
(ii) the Instruction Decode unit
(iii) the Load and Store unit.

In order to have a mean of comparison with our results, we use for both cores hand-written test programs that reach a high stuck-at fault coverage. These test programs are cross-compiled and simulated in the same manner as the generated sequences are and their effects on the targeted processor units in terms of induced switching activity are investigated clock cycle per clock cycle. The results are then post-processed in pairs of two consecutive instructions in order to effectively compare with the instruction pair of the MaxSAT-generated sequences. The highest stress inducing pairs are considered in the comparison presented in Table I.

Experimental results show that for both processor cases, for all considered functional units, the proposed method generated sequences that outperform the respective segments found in the functional stuck-at test program. Most importantly, there is a notable difference in the case of the Decoding Unit and the Load and Store Unit of the RI5CY core. One can see that in fact the maximum sustainable switching activity is lower than that of the respective units in the OR1200 processor. Although, one cannot directly compare between two completely different (structurally) circuits this deviation is fully justified by the complexity imposed by the respective units in the case of the RI5CY processor. It is safe to assume (as can be seen also in Fig. 3) that as the complexity of the module increases, the number of possible concurrent logical switches within a circuit decreases i.e., they are related in an inversely proportional manner. Regarding the two units, in the case of the RI5CY processor it is true, that they cover a significantly larger set of instructions than these of the OR1200 processor. Lastly, it can be seen that there exists no clock cycle, during the application of the stuck-at test program on the RI5CY core, in which a higher number of concurrent toggling was found than the number of concurrent toggling induced by any of the generated instructions.

Comparing with Other Methods

TABLE II
SUPPLEMENTARY COMPARISONS

Generation Approach	**Average Induced Stress**			
	Adder	Multiplier	Decoding Unit	Load Store Unit
MaxSAT	82%	62%	91%	65%
Evo	61%	55%	63%	58%
Test Program	24%	6%	44%	57%

In order to provide the reader with a further comparison, we have also developed stress programs for the OR1200 processor (considering the same modules) via μgp [42], which is an evolutionary optimizer that was initially developed to produce assembly programs maximizing a given fitness function for a variety of processors. The implemented evolutionary algorithm is similar to the one described in [16]. We also considered the special case of the 32-bit multiplier unit. The results are reported in Table II below.

We can clearly see that although the evolutionary-based stress program generation yields sequences of instructions that are better in terms of induced stress in the considered processor units than the test programs, they are sub-optimal solutions and thus, the proposed method outperforms them in all cases. For the case of the 32-bit multiplier though, the required CPU time was quite large (approximately 85 hours), and considerably greater than for the rest of the considered units. It is well known that the arithmetic multiplier circuits represent an arduous task for formal methods [43, 44]. In order to overcome the complexity imposed by the multiplier circuit, a heuristic sampling approach was employed. Specifically, instead of generating a switching constraint for every net of the multiplier unit (as dictated by the algorithm presented in Figure 8), we sampled a portion of high fan-out nets of the circuit and enforced switching constraints only on them. This implies that since the circuit has not been fully considered (in terms of nets being constrained for maximization), the generated solution is still a sub-optimal one, although better than the one produced with other approaches.

V. CONCLUSIONS

In many IC test scenarios the SWA of the CUT must be kept to a minimum to avoid potentially unwanted and catastrophic conditions such as overheating and over-stressing of the DUTs. However, there are specific cases in which the maximization of the SWA of the DUTs (or of certain sub-modules) is required, and represents a major challenge. For instance, during BI test the repeatable constant SWA maximization of the CUT via functional stimuli can assist in the maximization of the overall stress and thus, to screen out early failures in a more efficient manner.

In this paper, we propose an algorithm based on formal methods, able to effectively tackle the case of the SWA maximization when the DUT is a fully pipelined processor and we aim to optimally maximize the repeatable constant SWA of certain sub-modules of the core by generating appropriate assembly programs. We have shown that under the two-instruction sequence assumption, our method generates the optimal sequence in terms of induced repeatable and constant switching activity to the targeted processor module.

Further work is currently being done to consider different and more sophisticated stress metrics (e.g., taking into account the layout of the circuit).

References

[1] R. Vollertsen. "Burn-In". In: *IEEE International Integrated Reliability Workshop*. 1993. DOI: 10.1109/irws.1999.830588.

[2] T. M. Mak. "Infant mortality - The Lesser Known Reliability Issue". In: *IOLTS 2007 13th IEEE International On-Line Testing Symposium*. 2007. DOI: 10.1109/IOLTS.2007.40.

[3] C. Ascarrunz. "HALT: Bridging the Gap Between Theory and Practice". In: *IEEE International Test Conference*. 1994. DOI: 10.1109/test.1994.527998.

[4] C. Laplante. "Improving Reliability Throughout the Product Life Cycle". In: *Annual Reliability and Maintainability Symposium*. 2018. DOI: 10.1109/RAM.2018.8463120.

[5] B. Peng et al. "IC HTOL Test Stress Condition Optimization". In: *IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems*. 2004. DOI: 10.1109/DFTVS.2004.1347849.

[6] C. He. "Advanced Burn-In - An Optimized Product Stress and Test Flow for Automotive Microcontrollers". In: *International Test Conference*. 2019. DOI: 10.1109/ITC44170.2019.9000147.

[7] Y Han Ng, Y. Hock Low, and S. Demidenko. "Improving Efficiency of IC Burn-In Testing". In: *2008 IEEE Instrumentation and Measurement Technology Conference*. 2008. DOI: 10.1109/IMTC.2008.4547315.

[8] M. Fairuz Zakaria et al. "Reducing Burn-In Time Through High-Voltage Stress Test and Weibull Statistical Analysis". In: *IEEE Design and Test of Computers* 23.2 (2006). DOI: 10.1109/MDT.2006.50.

[9] N. Aghaee, Z. Peng, and P. Eles. "Temperature-Gradient-Based Burn-In and Test Scheduling for 3-D Stacked ICs". In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 23.12 (2015). DOI: 10.1109/TVLSI.2014.2380477.

[10] R. Cantoro et al. "On the Maximization of the Sustained Switching Activity in a Processor". In: *21st IEEE International On-Line Testing Symposium, IOLTS 2015*. 2015. DOI: 10.1109/IOLTS.2015.7229826.

[11] N. I. Deligiannis et al. "Effective SAT-based Solutions for Generating Functional Sequences Maximizing the Sustained Switching Activity in a Pipelined Processor". In: *2021 IEEE 30th Asian Test Symposium (ATS)*. 2021. DOI: 10.1109/ATS52891.2021.00025.

[12] Y. Zhang et al. "Temperature-Aware Software-Based Self-Testing for Delay Faults". In: *Design, Automation and Test in Europe, DATE*. 2015. DOI: 10.7873/date.2015.0744.

[13] N. Hage et al. "Instruction-based self-test for delay faults maximizing operating temperature". In: *2017 IEEE 23rd International Symposium on On-Line Testing and Robust System Design, IOLTS 2017*. 2017. DOI: 10.1109/IOLTS.2017.8046231.

[14] I. Polian et al. "Exploring the Mysteries of System-Level Test". In: *Asian Test Symposium*. 2020. DOI: 10.1109/ATS49688.2020.9301557.

[15] D. Appello et al. "System-Level Test: State of the Art and Challenges". In: *IEEE International Symposium on On-Line Testing and Robust System Design (IOLTS)*. 2021. DOI: 10.1109/IOLTS52814.2021.9486708.

[16] N. I. Deligiannis, R. Cantoro, and M. Sonza Reorda. "Maximizing the Switching Activity of Different Modules Within a Processor Core via Evolutionary Techniques". In: *2021 Euromicro Digital System Design (DSD)*. 2021. DOI: 10.1109/dsd53832.2021.00086.

[17] Ruggeri, W. and others. "Innovative methods for Burn-In related Stress Metrics Computation". In: *International Conference on Design Technology of Integrated Systems in Nanoscale Era (DTIS)*. 2021. DOI: 10.1109/DTIS53253.2021.9505067.

[18] S. Chowdhury and J. Sabir Barkatullah. "Estimation of Maximum Currents in MOS IC Logic Circuits". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 9.6 (1990). DOI: 10.1109/43.55194.

[19] H. Kriplani, F. Najm, and I. Hajj. "Maximum Current Estimation in CMOS Circuits". In: *Proceedings - Design Automation Conference*. 1992. DOI: 10.1109/dac.1992.227873.

[20] J. Monteiro et al. "Estimation of Average Switching Activity in Combinational Logic Circuits Using Symbolic Simulation". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 16.1 (1997). DOI: 10.1109/43.559336.

[21] Chuan Yu Wang and Kaushik Roy. "Maximum Power Estimation for CMOS Circuits Using Deterministic and Statistical Approaches". In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 6.1 (1998). DOI: 10.1109/92.661255.

[22] S. Devadas, J. White, and K. Keutzer. "Estimation of Power Dissipation in CMOS Combinational Circuits Using Boolean Function Manipulation". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 11.3 (1992). DOI: 10.1109/43.124424.

[23] Kuo Chan Huang et al. "Maximization of Power Dissipation Under Random Excitation for Burn-In Testing".

In: *International Test Conference 1998*. 1998. DOI: 10.1109/TEST.1998.743200.

[24] A. Sagahyroon. "Maximizing Heat Dissipation for Burn-In Testing". In: *IEEE CCECE2002 Canadian Conference on Electrical and Computer Engineering*. 2002. DOI: 10.1109/CCECE.2002.1015257.

[25] F. Najm and M. Zhang. "Extreme Delay Sensitivity and the Worst-Case Switching Activity in VLSI Circuits". In: *32nd Annual ACM/IEEE Design Automation Conference*. 1995. DOI: 10.1145/217474.217600.

[26] S. Manich and J. Figueras. "Maximizing the Weighted Switching Activity in Combinational CMOS Circuits Under the Variable Delay Model". In: *Proceedings of the 1997 European Conference on Design and Test, EDTC 1997*. 1997. DOI: 10.1109/edtc.1997.582422.

[27] W. Qing, Q. Qinru, and M. Pedram. "Estimation of Peak Power Dissipation in VLSI Circuits Using the Limiting Distributions of Extreme Order Statistics". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 20.8 (2001). DOI: 10.1109/43.936376.

[28] F. A. Aloul and A. Sagahyroon. "Estimation of the Weighted Maximum Switching Activity in Combinational CMOS Circuits". In: *2006 IEEE International Symposium on Circuits and Systems*. 2006. DOI: 10.1109/ISCAS.2006.1693238.

[29] F. A. Aloul and A Sagahyroon. "Using SAT Techniques in Dynamic Burn-In Vector Generation". In: *Melecon 2010 - 2010 15th IEEE Mediterranean Electrotechnical Conference*. 2010. DOI: 10.1109/MELCON.2010.5476223.

[30] D. Appello et al. "A Comprehensive Methodology for Stress Procedures Evaluation and Comparison for Burn-In of Automotive SoC". In: *IEEE Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017*. 2017. DOI: 10.23919/DATE.2017.7927068.

[31] N. I. Deligiannis et al. "New Techniques for the Automatic Identification of Uncontrollable Lines in a CPU Core". In: *VLSI Test Symposium (VTS), 2021*. 2021. DOI: 10.1109/VTS50974.2021.9441040.

[32] T. Larrabee. "Test pattern generation using Boolean satisfiability". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 11.1 (1992). DOI: 10.1109/43.108614.

[33] Tseitin, G. S. "On the Complexity of Derivation in Propositional Calculus". In: *Automation of Reasoning: 2: Classical Papers on Computational Logic 1967–1970*. Berlin, Heidelberg: Springer Berlin Heidelberg, 1983, 466–483. ISBN: 978-3-642-81955-1. DOI: 10.1007/978-3-642-81955-1_28.

[34] T. Faller et al. "Towards SAT-Based SBST Generation for RISC-V Cores". In: *Latin American Test Symposium (LATS)*. 2021. DOI: 10.1109/LATS53581.2021.9651819.

[35] M. Sauer et al. "PHAETON: A SAT-Based Framework for Timing-Aware Path Sensitization". In: *IEEE Transactions on Computers* 65.6 (2016). DOI: 10.1109/TC.2015.2458869.

[36] S. Gurumurthy et al. "Automatic Generation of Instructions to Robustly Test Delay Defects in Processors". In: *12th IEEE European Test Symposium (ETS)*. 2007. DOI: 10.1109/ETS.2007.13.

[37] P. Raiola et al. "Evaluating the Effectiveness of D-chains in SAT-based ATPG and Diagnostic TPG". In: *Journal of Electronic Testing: Theory and Applications (JETTA)* 33 (2017). DOI: 10.1007/s10836-017-5693-6.

[38] A. Riefert et al. "A Flexible Framework for the Automatic Generation of SBST Programs". In: *IEEE Transactions on Very Large Scale Integration (VLSI) System* 24.10 (2016). ISSN: 1063-8210, 1557-9999. DOI: 10.1109/TVLSI.2016.2538800.

[39] *OpenRISC*. https://openrisc.io. [Online; accessed 24-Feb-2022].

[40] *Silvaco 45nm Open Cell Library*. https://si2.org/open-cell-library. [Online; accessed 24-Feb-2022].

[41] *PULP*. https://pulp-platform.org/. [Online; accessed 24-Feb-2022].

[42] Ernesto Sanchez et al. *Evolutionary Optimization: the μGP toolkit*. Berlin, New York: Springer, 2011. ISBN: 9780387094250.

[43] A. Mahzoon et al. "PolyCleaner: clean your polynomials before backward rewriting to verify million-gate multipliers". In: *International Conference on Computer-Aided Design*. 2018. DOI: 10.1145/3240765.3240837.

[44] D. Kaufmann et al. "Verifying Large Multipliers by Combining SAT and Computer Algebra". In: *2019 Formal Methods in Computer Aided Design (FMCAD)*. 2019. DOI: 10.23919/FMCAD.2019.8894250.

Nikolaos Ioannis Deligiannis received the MSc degree in Computer Science and Engineering from the Department of Computer Science and Engineering of University of Ioannina, Greece, in 2019. He was a research assistant in the Department of Control and Computer Engineering of Politecnico di Torino where he is currently a Ph.D. student. His research interests include testing of processors using formal methods and fault tolerance. He is a member of the IEEE.

Tobias Faller is a Ph.D. student, supervised by Prof. Bernd Becker at the chair of computer architecture at the University of Freiburg in Germany. His research focuses on software-based self-test generation for RISC-V using formal methods. Tobias received his Bachelor and Master of Science in Embedded Systems Engineering from the University of Freiburg. He is a member of the IEEE.

Riccardo Cantoro received the MSc degree and the Ph.D. in computer engineering from Politecnico di Torino, Italy, in 2013 and 2017, respectively. He is currently an Assistant Professor with the Department of Computer Engineering of the same university. His research interests include software-based functional testing of SoCs and memories, and machine learning applied to test and diagnosis. He is a member of the IEEE.

Tobias Paxian is a Ph.D. student in computer science and works currently in the chair of computer architecture at the University of Freiburg in Germany. Prof. Armin Biere and Prof. Bernd Becker are his supervisors. His research focuses on formal verification, encodings for optimization problems in CNF, SAT and MaxSAT solving. Tobias received his Bachelor and Master of Science from the University of Freiburg, where he developed a new encoding which is used in the later developed MaxSAT solver Pacose. He is a member of the IEEE.

Matteo Sonza Reorda received the MSc degree in electronics and the Ph.D. degree in Computer Engineering from Politecnico di Torino, Italy, in 1986 and 1990, respectively, where he is currently a Full Professor with the Department of Control and Computer Engineering. He published more than 400 papers in the area of test and fault tolerant design of reliable circuits and systems, receiving several Best Paper Awards at major international conferences. He is involved in numerous research projects with companies and other research centers worldwide. He is a Fellow of the IEEE.

Bernd Becker received the Diploma and Ph.D. degree in mathematics and computer science from the University of Saarland, Saarbrücken, Germany, in 1979 and 1982, respectively. From 1995 to 2021, he has been a Full Professor and the Chair of Computer Architecture with the Faculty of Engineering, University of Freiburg, Freiburg im Breisgau, Germany. Prior to joining the University of Freiburg in 1995, he was with J.W. Goethe-University Frankfurt as an Associate Professor for complexity theory and efficient algorithms. His research interests include design, test, and verification methods for embedded systems, and nanoelectronic circuitry. He is a Director of the Centre for Security and Society, University of Freiburg, and a member of Academia Europaea. He is a Fellow of the IEEE.