Effective SAT-based Solutions for Generating Functional Sequences Maximizing the Sustained Switching Activity in a Pipelined Processor

Effective SAT-based Solutions for Generating Functional Sequences Maximizing the Sustained Switching Activity in a Pipelined Processor

Nikolaos I. Deligiannis†, Riccardo Cantoro†, Tobias Faller*, Tobias Paxian*, Bernd Becker*, Matteo Sonza Reorda†
† Politecnico di Torino, DAUIN - Torino, Italy * University of Freiburg - Freiburg, Germany

***Abstract*—During device testing, one of the aspects to be considered is the minimization of the switching activity of the circuit under test in order to steer clear of introducing problems due to device overheating. Nevertheless, there are also certain scenarios during which the maximization of switching activity of the circuit under test (CUT) or of certain parts of it could be proven beneficial e.g., during Burn-In (BI), where internal stress is often produced by applying suitable stimuli. This can be done in a functional manner based on Software-based Self-Test in order to avoid possible damages to the CUT and/or any kind of yield loss. However, the generation of suitable test programs for this task represents a non-trivial task. In this paper we consider a scenario where the circuitry to be stressed is a pipelined processor. We present a methodology, based on formal techniques, able to automatically generate the best functional stress stimuli, i.e., a short and repeatable sequence of assembly instructions, which is guaranteed to induce the maximum switching activity within a given target processor module over a pre-defined time period. For the purposes of our experiments we used the OpenRISC 1200. The gathered experimental results demonstrate the effectiveness of the developed method. In particular, we show that the time for generating the best instruction sequence is limited in most cases, while the generated sequence can always achieve a significantly higher sustained toggling activity than any other solution.**

I. Introduction

When considering the different steps composing the whole test flow adopted by semiconductor companies to screen their devices, we can identify a few of them, where the availability of stress stimuli may be important. One of these steps is *Burn-In* (BI), when the Circuit Under Test (CUT) is subject to different types of external and internal stress in order to artificially age it, so that any weak component evolves into an observable defect and can be detected. While in the past BI stress was mainly based on *external* solutions (e.g., applying high temperature and voltage to the CUT), the recent trend goes rather in the direction of creating *internal* stress, e.g., maximizing the switching activity [1]. While the internal stress can be easily induced by relying on Design for Testability (DfT) mechanisms, such as scan, in these solutions the CUT works in test mode, thus ages in a way which may be different than in operational mode, and thus possibly introducing unnecessary test escapes or yield loss. As a result, it becomes crucial to devise strategies, able to generate functional test stimuli able to maximize the stress, i.e., the switching activity while the CUT works in normal mode [2].

Maximizing the switching activity, either in the whole CUT or in some parts of it, may turn to be effective even in other test steps. For example, it has been speculated that testing for delay faults while the circuit temperature is at the top of the allowed range may enable the detection of a higher percentage of defects [3] [4]. This result may be achieved by carefully written functional stimuli able to maximize the internal stress.

More recently, the adoption of System Level Test (SLT) to detect defects that could escape all the traditional test steps led to the search for functional stimuli able to produce particularly stressful conditions. Once again, this can be achieved by maximizing the internal activity and creating temperature gradients between different modules within the CUT while resorting to functional stimuli [5].

To summarize, different steps in a typical test flow currently adopted by semiconductor companies may benefit from effective solutions able to generate functional programs maximizing the switching activity either in the whole CUT or in a specific target module.

This paper addresses the issue of the automatic generation of the sequence of instructions whose execution maximizes the switching activity in a given module of a CPU. The idea behind our approach is that the above task can be achieved by generating a few instructions, assuming that they can be executed repeatedly, thus creating a high and sustained switching activity in the target CPU module. In this way, we can use the generated programs not only to stress the target module, but also to control the CUT temperature, driving it to the maximum value allowed by functional conditions. For this purpose we propose a solution based on formal techniques, which guarantees that the generated program achieves the absolute maximum switching activity.

Results gathered on an OpenRISC 1200 (OR1200) pipelined processor show the effectiveness and feasibility of the method. The method is relatively easy to be adopted since it does not require in-depth knowledge of the processor's architecture but rather a basic/conceptual understanding of it. Also, the method generates stress programs from the ground up and does not require any dependency on pre-existing programs. Furthermore, the generated test programs are relatively short in length, typically composed of 2 instructions that can be indefinitely repeated. Finally, by construction, the generated programs induce the maximum switching activity in the target

module, and comparisons with other programs (e.g., test applications) show that the switching activity achieved by our method is by far higher.

The rest of the paper is organized as follows: Section II reports about previous work in the area and provides some background concepts. In Section III we better state the addressed problem and outline the proposed method. In Section IV we describe the experimental setup we created to assess the validity of the method along with the respective results. Finally, in Section V we draw conclusions and provide insight on our future work.

II. Background & Motivation

The significance of power consumption of integrated circuits (ICs) during device design and testing is a well known problem, strongly linked with the attribute of reliability. Researchers have thoroughly studied the problem in the past. In [6], [7], the authors present algorithms for accurate estimations of maximum currents in combinational MOS ICs while explaining how such information can be proven beneficial to the reliability of the design. For example, the design of a reliable power and ground bus inside a circuit requires an accurate estimation of the respective currents.

In [8] the authors highlight the role that an accurate power estimation has in the reliability of a CMOS IC. They state that the problem of accurate power consumption estimation nests the problem of identifying two consecutive test vectors which maximize the switching activity (SWA) of the IC, which is an NP-Complete problem with a complexity of $\mathcal{O}(4^n)$, where n is the total number of primary inputs for the IC. They introduce a methodology aiming to identify pairs of consecutive input vectors (to instigate high stress amounts in the circuit), based on automatic test generation algorithms, in order to effectively calculate a power estimate for a combinational IC.

In [9] the authors present a method for the estimation of the maximum power dissipation of a combinational circuit by describing the power dissipation as a Boolean function of the circuit's primary inputs. They relate the problem of the power dissipation estimation to maximizing gate output SWA while introducing techniques for transforming the problem into a weighted max-satisfiability (MaxSAT) problem and present strategies to effectively solve it. The methods were applied to relatively small combinational circuits due to the high complexity imposed in order to (i) obtain the objective function of the circuit and to (ii) optimize the objective function.

In [10], [11] the authors propose methods aiming to maximize the SWA of combinational ICs in order to estimate the power consumption while considering a variety of delay models.

While the importance of a circuit's SWA maximization is important and plays a critical role during the design phase of the devices, it is advantageous in the context of device testing as well. During the multiple test "layers" that are introduced in-between each step of the device manufacturing, it may happen that faults do not manifest themselves during testing and even escape from the final system level test. They are the prime suspects behind the Infant Mortality behavior which is typically resolved via Burn-In (BI), where the devices are being exercised in elevated temperature and power conditions. In [12] the authors present a probabilistic approach (random excitation) to maximize the SWA of a combinational circuit to further achieve the maximization of power dissipation during BI testing. In [13] the author employs a genetic algorithm to develop a technique for the maximization of a combinational circuit's SWA in order to also maximize the circuit's heat dissipation during BI testing.

The subject of SWA maximization in combinational circuitry has been thoroughly and extensively studied; moving to sequential circuits, and more specifically to processors, the authors of [2] present an evolutionary technique that aims to maximize the sustained SWA of a processor by extracting characteristics, i.e., high stress-inducing sequences of instructions from suites of pre-existing test programs for the target processor. In [14] the authors, while targeting a 32-bit processor, present a comprehensive methodology and propose metrics for the comparison and the evaluation of stress procedures that are applied on the circuit during BI. On one hand, the circuitry is equipped with design for testability 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 circuit. On the other hand, when functional stimuli (stress programs) are applied at-speed to the processor, a significantly higher thermal activity is observed.

Knowing that it is crucial to maintain a minimal SWA within a device during test, we can clearly see that there are certain occasions where the maximization of the SWA can be proven beneficial to the overall reliability of the device. In this paper, while focusing on processors, we provide an algorithm for the sustained maximization of the SWA of certain processor modules by generating the appropriate functional stimuli i.e., sequences of instructions.

III. Problem Definition & Proposed Approach

Preliminaries

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, e.g., via DfT. Our goal in this paper is to propose an algorithm that takes the gate-level description of a pipelined processor as input and generates functional stimuli to effectively stress any module within it. This means identifying two instructions that are able (when executed in sequence) to induce the maximum stress in the target module, meaning that the pair of instructions is guaranteed to produce the highest possible gate activity, from High to Low (HL) and Low to High (LH), within the target processor module.

Furthermore, we are interested in the maximization of the *sustained* SWA in the target processor module, which means that we also require the generated sequence of instructions

to be *repeatable*. We assume that the processor, after its proper initialization to remove potential X (Don't Care) values e.g., via the activation of the asynchronous $RESET$ signal, is functionally driven to a well defined and legal state σ^a. Then, the first of the two stress-inducing instructions drives it to a new state σ^b_{max}: this transition triggers the maximum amount of nets switching within the module. Finally, the second instruction drives the processor to a state σ^c_{max} for which it must hold that $\sigma^c_{max} \equiv \sigma^a$. For each targeted module specific constraints are considered in order to ensure that the starting state of the stress segment (i.e., σ^a) is a valid one. The aforementioned sequence of states can be summarized as in Figure 1.

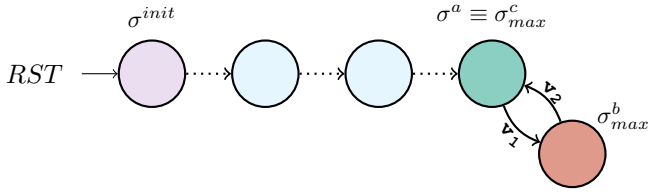

Fig. 1. Representation of a processor's module sustained SWA maximization

By guaranteeing that the generated instruction sequence is repeatable, we can maintain a sustained high gate activity within the target module for an arbitrarily long period of time. Assuming that the optimal repeatable pair of instructions ($\widetilde{s}$) has been generated by the algorithm, they can be used to stress the target module (e.g., to create a hot-spot within the processor). This can be achieved by repeating the sequence $\widetilde{s}$ for a high, yet discrete amount of times (e.g., $\widetilde{s}$,$\widetilde{s}$,$\widetilde{s}$,...,$\widetilde{s}$). At the end of the sequence of repeated instructions we can introduce an unconditional jump instruction to transfer the code execution back to the start of the stress sequence. In this way, the sequence can be applied an arbitrarily large amount of times until it is stopped (e.g., via an interrupt).

SWA Metric and Optimal Sequence Length

The metric we use to measure the amount of created stress is the average induced stress percentage. Given that the generic target processor module consists of m nets and the assembly stress program is composed of n instructions, we calculate the average induced stress percentage as:

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

The numerator of the fraction represents the total amount of HL and LH transitions that were performed by every net of the target module, while the denominator represents the maximum, when every net of the processor makes a transition when each instruction of the stress program is executed. This value can be negatively affected by the presence of uncontrollable lines in the target module [15]. Moreover, it is clearly not given that all nets can be toggled in the same clock period. Thus, the 100% can be interpreted as a theoretical maximum.

Let us consider a sequence of instructions $\widetilde{s}$ = (I_1, I_2) that induce the maximum SWA ($SWA^{max}_{\widetilde{s}}$) within a processor module T when executed (i.e., $I_1 \rightarrow I_2 \rightarrow I_1$). The $SWA^{max}_{\widetilde{s}}$ corresponds to the sum of the number SW_{I_1,I_2} representing the number of nets changing their values when I_2 is executed after I_1 plus the number SW_{I_2,I_1} when I_1 is executed after I_2. If the first transition forces a certain net to switch, then the second transition will force the same net to switch ($SW_{I_1,I_2} = SW_{I_2,I_1}$). Thus, maximizing the SWA can be translated to finding the pair of instructions that induces the highest gate switching when I_2 is processed after I_1 (or vice-versa). The sequence $\widetilde{s}$ that maximizes the SWA can be repeated for a generic number of times N, for a total of $2 \times N$ instructions (e.g., $N = 2 : I_1, I_2, I_1, I_2$). There is no other sequence of $2 \times N$ instructions that induces a higher SWA in the module.

Proof: Let's assume a second sequence of instructions $\widetilde{s'}$ = (I'_1, I'_2, I'_3, I'_4) for the module T that induces a higher $SWA^{max}_{\widetilde{s'}}$ value than $SWA^{max}_{\widetilde{s}}$. It holds that:

$$SWA^{max}_{\widetilde{s'}} = SW_{I'_1,I'_2} + SW_{I'_2,I'_3} + SW_{I'_3,I'_4} \quad (1)$$

$$SWA^{max}_{\widetilde{s}} = 3 \times SW_{I_1,I_2} \quad (2)$$

This implies that a term exists in (1) that has a higher value than any term of (2). However, we showed that sequence $\widetilde{s}$ is composed of the two instructions that maximize the number of nets switching within T. Thus, no term of (1) can have a higher value than any term of (2). Hence, the sequence $\widetilde{s}$ is the one maximizing the SWA value for $N = 2$. The same reasoning can be repeated for higher values of N.

Approach using Formal Techniques

In order to effectively solve the problem, we rely on formal techniques and specifically to the problem of the max-satisfiability (MaxSAT) since the maximization of the SWA of a processor's module can be seen as an optimization problem i.e., to sensitize as many nets of a circuit as possible during a short time window. The core idea is to force constraints that maximize the SWA over two consecutive clock cycles, and then let the solver determine the satisfiability of the corresponding conjunctive normal form (CNF) formula in order to finally obtain the two vectors that can stress the module to the maximum. For example, let us assume that we intend to stress the adder of the arithmetic and logic unit. In this scenario, the two input vectors that we request correspond to the four operands (two for every addition) that will be added. So, assuming two clock cycles for our scenario, one for every addition instruction, we have three valid states for the circuit:

$$\sigma^1 \xrightarrow[\text{clk}^{\uparrow}]{\text{instr}_1^{\text{add}}} \sigma^2 \xrightarrow[\text{clk}^{\uparrow}]{\text{instr}_2^{\text{add}}} \sigma^3$$

But since it is required that the generated stress maximization sequence is repeatable, as previously shown in Figure 1, this further implies for the aforementioned states that $\sigma^3 \equiv \sigma^1$.

In order to achieve such a scenario i.e., transform the problem of the circuit's sustained SWA maximization into a MaxSAT problem, we must first obtain its Boolean formula in a CNF. This is achieved by first unrolling the circuit in time. During circuit unrolling the circuit is duplicated a specified amount of times and then the segments are connected to

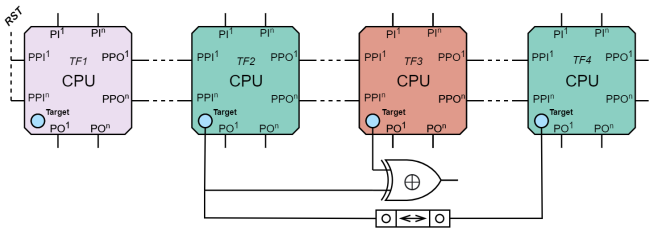

Fig. 2. MaxSAT model

each other. 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, and so on. Each instance on the now unrolled circuit represents a state of the processor corresponding to a clock cycle and is called a *timeframe*. For the purposes of the proper initialization of the circuit, we use extra timeframes (the very first ones), during which we drive the circuit to a well defined state. This is done to remove potential X (Don't Care) values and can be achieved either by activating the $RESET$ signal on that timeframe or by forcing the circuit to execute a given initialization sequence. For example, for the aforementioned scenario regarding the maximization of the SWA of the adder, we would need to unroll the circuit at least $1+3$ times (once for every state). Lastly, by encoding the unrolled circuit we obtain the Boolean formula in a CNF (e.g., via symbolic simulation).

Once the CNF is generated we can further encode constraints as clauses to the formula. For instance, we can force specific logic values on the circuit's nets at certain timeframes and we can require certain properties for nets at particular timeframes in order to avoid some undesirable scenarios e.g., to prohibit the SAT-solver to act on the $RESET$ signal in order to sensitize a net. For partial weighted MaxSAT an additional set of soft clauses, each one correlated with an integer weight, is added. The optimization goal of MaxSAT is then to maximize the sum of weights for the satisfied soft clauses, whereas the original CNF, containing so called hard clauses, has to be satisfied.

Figure 2 (which can be correlated with Figure 1) illustrates the MaxSAT model we propose for the generation of stress inducing sequences for a certain processor module. The first timeframes are used to initialize the whole processor. Initially, we force a constraint in the CNF formula, requiring the $RESET$ signal to be activated. Since the initialization phase takes more than 1 clock cycle, the first timeframe (TF1) can be interpreted as the last timeframe of the initialization phase. In the following three states, we focus on the module of the processor we intend to stress. As previously explained, the first (TF2) and the last state (TF4) of the module must be equivalent in order to guarantee repeatability. For that reason, every literal that encodes a net of the targeted module during TF2 ($l_{tf_2}^{net_i}$), along with every literal that encodes a net of the targeted module during TF4 ($l_{tf_4}^{net_i}$) are linked together by forcing the following logic implications as hard clauses to the CNF formula, which dictates their equivalence during the these two states:

$$\omega^{hard}: \quad l_{tf_2}^{net_i} \longleftrightarrow l_{tf_4}^{net_i} \equiv (\neg l_{tf_2}^{net_i} \vee l_{tf_4}^{net_i}) \wedge (\neg l_{tf_4}^{net_i} \vee l_{tf_2}^{net_i})$$

Hence, the two states are now equivalent i.e., it is guaranteed that all the nets of the module will hold exactly the same logic values during TF2 and TF4.

Since we have encoded the equivalence between the aforementioned states for every net of the targeted module, we now develop soft clauses regarding the maximization of the switching of the target module's net between TF2 and TF3 i.e., we request to the solver to maximize as many of these clauses during TF2 and TF3. For every net i of the target module we encode an XOR gate on the CNF, whose inputs are the corresponding literals for the net i in TF2 and TF3, respectively. Then, we force a difference by requiring that the output of every XOR gate generated during this process to be 1. Note, that due to the equivalence between TF2 and TF4, it is redundant to further add soft clauses for the switching maximization during TF3 and TF4, since that is already implied through the states' equivalence.

Finally, after the successful solving of the CNF formula we extract the input vectors by transforming the relevant input literals i.e., the ones that encode the PIs of the module, to 0/1 logic. For example, assuming that the targeted processor module is the adder, we can extract the (A,B), (C,D) pairs of operands from TF2 and TF3 respectively along with the instruction register of the execute stage of the processor (which nests the adder module) in order to reconstruct a valid stress program. This can be achieved by extracting and disassembling N-bit instruction that the instruction register holds in order to find the assembly instruction to be used along with the respective registers. Finally, we can extract the (A,B) (C,D) operands from TF2 and TF3 respectively.

Validity Checking

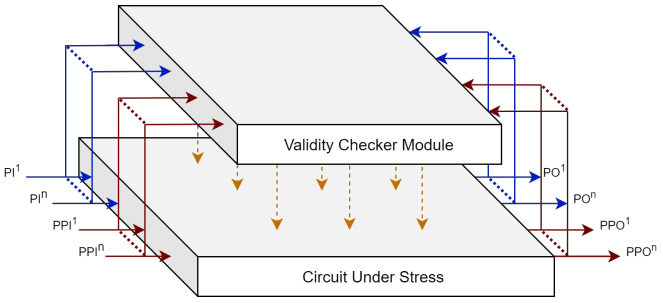

Fig. 3. Validity Checker Module interaction with the Circuit Under Stress

While the basic constraints for the problem can be generalized to almost every module of a processor, in most cases further actions must be performed in order to ensure that there are no violations and unwanted behaviours within the pipeline. Furthermore, we must also guarantee that the generated sequences of instructions represent valid and reachable states. For instance, the SAT solver could generate an invalid instruction and thus, trigger an exception at a certain point or assign incorrect values to certain register bits. For such reasons, we develop via propositional logic (acting on the CNF

TABLE I
EXPERIMENTAL RESULTS

Stress Program Generation Approach	Average Induced Stress			CPU Generation Time		
	Adder	Multiplier	Decoding Unit	Adder	Multiplier	Decoding Unit
Formal Techniques (MaxSAT)	81.92%	62.15%	90.56%	3^{sec}	85^{hrs}	15^{min}
Evolutionary Algorithm (EA)	61.34%	54.77%	62.57%	48^{hrs}	50^{hrs}	73^{hrs}
Stuck-At Test Program	24.00%	6.34%	44.43%		-	

formula) suitable validity checking modules (VCMs), which are specific to the processor module we aim to stress. A similar approach is presented in [16], [17]. Figure 3 illustrates the interaction of the VCM and the circuit under stress. The VCM has access to the PIs/PPIs and POs/PPOs of the target module and is able to access the internal state of the circuit.

For example, assuming that we aim to stress the decoding unit of a processor, the stimuli we are looking for are not just pairs of operands, but pairs of arbitrary instructions. Thus, we ensure via the VCM that the appropriate instruction syntax will be followed for every instruction of the processor's ISA by forcing constraints on the instruction register bits and developing mechanisms (via propositional logic) that exclude scenarios from the search space that would cause an exception to be triggered e.g., invalid instruction or misaligned memory accesses.

The Algorithm

The proposed approach is presented in a pseudo-code format in Fig. 4. Note that the unrolling depth may vary depending on the target processor module we intend to stress. For the example behavior where every instruction of the processor requires 1 clock cycle to move from one pipeline stage to the next, 4 timeframes would suffice. But in some cases e.g., for multiplier circuits within the ALU, it is possible that > 1 clock cycles are required for the multiplication instruction to be executed. For instance, assuming a certain processor whose multiplier module requires 2 clock cycles per multiplication instruction we would need at least $1 + 2 + 2 + 2 = 7$ timeframes to model our problem (twice for every state). The algorithm accounts for such scenarios, given that the circuit has been unrolled a sufficient amount of times, since it allows for explicit formulation and insertion of constraints at any timeframe of the CNF formula.

```
input  : A triplet (G,M,ud) where
         G is the gate-level description of the processor
         M 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 M
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 ∈ M do
4  |  AddDifferences (CNF, n_i, TF_σ^a, TF_σ^b_max)
5  |  AddImplications(CNF, n_i, TF_σ^a, TF_σ^c_max)
6  |  AddImplications(CNF, n_i, TF_σ^c_max, TF_σ^a)
7  end
   // VCM Implementation for the Target Module
8  ConstraintsFromVCM(CNF,M)
9  MaxSolve(CNF)
   // Extract Operands and Disassemble them to Instructions
10 I_1 ← ExtractOperandsFrom(CNF, M, TF_σ^a)
11 I_2 ← ExtractOperandsFrom(CNF, M, TF_σ^b_max)
12 return I_1, I_2
```

Fig. 4. Instruction sequence generation routine

IV. EXPERIMENTAL SETUP AND RESULTS

Our experiments were performed on a machine using an Intel i9-9900 CPU running at 3.10GHz. The generated stress programs were logically simulated via QuestaSIM by Mentor Graphics.

The processor used in our experiments is the OpenRISC 1200 (OR1200) [18]. The OR1200 is a 32-bit scalar RISC processor with Harvard micro-architecture. It is composed of 5 integer pipeline stages. The RT-level description of the core was synthesized using the Silvaco 45nm Open Cell Library [19]. In this paper, we focus on the sustained maximization of the SWA of the $adder$, the $multiplier$ and the $decoding\ unit$ within the OR1200.

Phaeton [17] 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 [20].

Our implementation of the proposed method was developed using Phaeton. It accounts for approximately 2,000 lines of `C++` code. Additionally, we wrote a tool in `python` (approximately 500 lines of code) that takes as input the values generated by the Phaeton application and produces assembly programs. Finally, the programs were logically simulated to validate them and obtain statistics regarding the induced SWA. In order to provide the reader with a reasonable comparison, we have also developed stress programs for the same modules via μgp3 [21], 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 [2]. Furthermore, we also measured the SWA induced by a test program for the OR1200 that reaches 85% stuck-at fault coverage [22]. The results of our experiments are reported in Table I.

From Table I we can see that the test programs generated by the proposed MaxSAT technique induce higher amounts of stress on all the units that we have considered. There is also a notable difference in the CPU generation time since we can see that for both the $adder$ and the $decoding\ unit$ the proposed method significantly outperforms the generation times of the evolutionary algorithm approach.

It can be seen though, that in the case of the $multiplier$ the required CPU time is significantly higher than for the other modules. It is well known that handling of arithmetic multiplier circuits represents an arduous task for formal techniques [23], [24]. For the purpose of our work, we employed a sampling approach in order to ease the large complexity imposed when targeting the multiplier as a target for SWA maximization. Namely, instead of creating a maximization soft clause for every net of the circuit (as dictated by the algorithm in Fig. 4), we sampled a portion of the nets of the circuit and forced the switching constraints only on them. This implies, that since the circuit has not been fully considered for maximization, the solution is a sub-optimal one, although better than the one produced with other techniques.

V. Conclusions

While in most scenarios the minimization of the circuit's switching activity is crucial during the device testing to avoid effects such as overheating, there are cases (e.g., during BI) where the goal is the maximization of the switching activity (of the whole CUT or certain sub-modules). During BI the maximization of the system's sustained switching activity via functional stimuli could aid to maximize the stress and thus to screen out early failures.

We proposed an algorithm, based on formal techniques, that takes the gate-level description of a pipelined processor as input and generates a sequence of assembly instructions able to optimally stress any module within it by maximizing the switching activity. We focused on the $adder$, the $multiplier$ and the $decoding\ unit$ and demonstrated the effectiveness of the generated stress programs while comparing with other methods. Work is currently conducted to optimally account also for the case of the $multiplier$ (for which we are currently not able to generate the absolute best sequence due to the special characteristics of the module) while also extending the method to other sub-modules and processors.

References

[1] C. He, "Advanced Burn-In - An Optimized Product Stress and Test Flow for Automotive Microcontrollers," in *2019 IEEE International Test Conference (ITC)*. Washington, DC, USA: IEEE, Nov. 2019.

This work was supported in part by the German Federal Ministry of Education and Research (BMBF) within the project Scale4Edge under contract no. 16ME0132.

[2] R. Cantoro *et al.*, "On the maximization of the sustained switching activity in a processor," in *2015 IEEE 21st International On-Line Testing Symposium (IOLTS)*. Halkidiki, Greece: IEEE, Jul. 2015.

[3] Y. Zhang *et al.*, "Temperature-Aware Software-Based Self-Testing for Delay Faults," in *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2015*. Grenoble, France: IEEE Conference Publications, 2015.

[4] 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)*. Thessaloniki, Greece: IEEE, Jul. 2017.

[5] I. Polian *et al.*, "Exploring the Mysteries of System-Level Test," in *Asian Test Symposium (ATS), 2020*. Virtual Conference: IEEE, Nov. 2020.

[6] S. Chowdhury and J. Barkatullah, "Estimation of maximum currents in MOS IC logic circuits," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 9, no. 6, Jun. 1990.

[7] H. Kriplani *et al.*, "Maximum current estimation in CMOS circuits," in *1992 Proceedings 29th ACM/IEEE Design Automation Conference*. Anaheim, CA, USA: IEEE Comput. Soc. Press, 1992.

[8] Chuan-Yu Wang and K. Roy, "Maximum power estimation for CMOS circuits using deterministic and statistical approaches," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 6, no. 1, Mar. 1998.

[9] S. Devadas *et al.*, "Estimation of power dissipation in CMOS combinational circuits using Boolean function manipulation," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 11, no. 3, Mar. 1992.

[10] C.-Y. Wang *et al.*, "Maximum power estimation for CMOS circuits under arbitrary delay model," in *1996 IEEE International Symposium on Circuits and Systems. Circuits and Systems Connecting the World. ISCAS 96*, vol. 4. Atlanta, GA, USA: IEEE, 1996.

[11] S. Manich and J. Figueras, "Maximizing the weighted switching activity in combinational CMOS circuits under the variable delay model," in *Proceedings European Design and Test Conference. ED & TC 97*. Paris, France: IEEE Comput. Soc. Press, 1997.

[12] Kuo Chan Huang *et al.*, "Maximization of power dissipation under random excitation for burn-in testing," in *Proceedings International Test Conference 1998*. Washington, DC, USA: Int. Test Conference, 1998.

[13] A. Sagahyroon, "Maximizing heat dissipation for burn-in testing," in *IEEE CCECE2002. Canadian Conference on Electrical and Computer Engineering. Conference Proceedings*, vol. 1. Winnipeg, Man., Canada: IEEE, 2002.

[14] D. Appello *et al.*, "A comprehensive methodology for stress procedures evaluation and comparison for Burn-In of automotive SoC," in *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017*. Lausanne, Switzerland: IEEE, Mar. 2017.

[15] N. I. Deligiannis *et al.*, "New Techniques for the Automatic Identification of Uncontrollable Lines in a CPU Core," in *VLSI Test Symposium (VTS), 2021*. Virtual Conference: IEEE, Apr. 2021.

[16] S. Gurumurthy *et al.*, "Automatic Generation of Instructions to Robustly Test Delay Defects in Processors," in *12th IEEE European Test Symposium (ETS'07)*, May 2007.

[17] M. Sauer *et al.*, "PHAETON: A SAT-Based Framework for Timing-Aware Path Sensitization," *IEEE Transactions on Computers*, vol. 65, no. 6, Jun. 2016.

[18] "OpenRISC," https://openrisc.io, [Online; accessed 03-Jun-2021].

[19] "Silvaco 45nm Open Cell Library," https://si2.org/open-cell-library, [Online; accessed 03-Jun-2021].

[20] A. Riefert *et al.*, "A Flexible Framework for the Automatic Generation of SBST Programs," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 24, no. 10, Oct. 2016.

[21] E. Sanchez *et al.*, *Evolutionary Optimization: the μGP toolkit*, 2011th ed. Berlin ; New York: Springer, Apr. 2011.

[22] R. Cantoro *et al.*, "An analysis of test solutions for COTS-based systems in space applications," in *2018 IFIP/IEEE International Conference on Very Large Scale Integration (VLSI-SoC)*, Oct. 2018.

[23] A. Mahzoon *et al.*, "PolyCleaner: clean your polynomials before backward rewriting to verify million-gate multipliers," in *Proceedings of the International Conference on Computer-Aided Design*. San Diego California: ACM, Nov. 2018.

[24] D. Kaufmann *et al.*, "Verifying Large Multipliers by Combining SAT and Computer Algebra," in *2019 Formal Methods in Computer Aided Design (FMCAD)*. San Jose, CA, USA: IEEE, Oct. 2019.