Automating the Generation of Programs Maximizing the Sustained Switching Activity in Microprocessor units via Evolutionary Techniques

Automating the Generation of Programs Maximizing the Sustained Switching Activity in Microprocessor Units via Evolutionary Techniques

Nikolaos I. Deligiannis, Riccardo Cantoro, Matteo Sonza Reorda
Politecnico di Torino, Dip. Automatica e Informatica. Torino, Italy
{nikolaos.deligiannis|riccardo.cantoro|matteo.sonzareorda}@polito.it

***Abstract*—During device testing, an important parameter to be considered by the test engineers is the switching activity (SWA) of the circuit under test (CUT). It is well known that the SWA must be kept to a minimum in order to avoid catastrophic scenarios on the CUTs, e.g., unacceptable peak power consumption or over-stressing that can lead to an artificial yield loss. However, there are scenarios, where the inverse, namely the switching activity maximization, can be proven beneficial. For example, this happens during *Burn-In* testing, when we aim at exercising the circuit under extreme operating conditions in terms of temperature and temperature gradients to accelerate aging phenomena in a controlled manner. In some cases, this is achieved by applying to the CUT stimuli in a functional manner, based on the Software-Based Self-Test (SBST) paradigm. Though, the generation of such appropriate programs for this task represents a challenge for the test engineers. In this paper we consider the scenario where the modules to be stressed are the units of a pipelined processor. We present a method, based on an evolutionary approach, that is able to automatically generate maximum stress programs, i.e., sequences of instructions achieving a high switching activity in the target module. With respect to other approaches, the generated sequences are built from the ground-up, are short and repeatable, thus guaranteeing their easy usability to stress a module (and increase its temperature). The processor we used for our experiments is the Open RISC 1200. Results demonstrate that the generated stimuli from the proposed method are effective in achieving a high value of sustained switching activity within the considered processor modules while also achieves a better results when comparing with other methods (e.g., stuck-at oriented test programs).**

***Index Terms*—Switching Activity Maximization, Evolutionary Techniques, Microprocessor, Internal Stress, Burn-In Test**

I. Introduction

As the semiconductor industry is in an ever-growing need for robust and reliable circuitry, efficient test solutions are crucial to achieve and guarantee such figures in an electronic system. Testing of electronic devices is performed in several different steps. The test engineers are tasked with the arduous task of identifying and applying such test solutions in order to achieve the target reliability standards, while also maintaining a viable cost. The whole set of test steps that is usually adopted by the semiconductor companies at the end of the manufacturing process may include Burn-In test (BI) [1], especially in the domain of the safety-critical applications. In the latter, the safety evaluation process and criteria are mandated and partly described by the respective standards (e.g., ISO 26262 for automotive, DO 254 for avionics, IEC 63304 for medical equipment) and BI testing is always present for devices meant for such applications.

During BI test, the circuit under test (CUT) is exercised in elevated temperature, power and frequency conditions as it is subject to different types of external and **internal** stress in order to artificially age it. Thus, any weak component evolves into an observable defect that can be detected. In this way, the phenomenon known as *Infant Mortality* [2], where a significant amount of defective products is observed (due to latent defects) in the early life stages of the devices, is countered. Another *test-to-fail* method (i.e., the product is tested until failure) commonly employed by semiconductor industries during the devices' development cycle is Highly Accelerated Life Testing (HALT) [3, 4]. Similarly to BI, HALT is a stress-test procedure that uses environmental (external) and **operational** (internal) stress to expose weak points in a product's design. Lastly, an additional stress inducing test procedure similar to BI that makes use of external and internal stress is the High Temperature Operation Life (HTOL) test. It is used to determine the effects of bias and temperature stress conditions on solid-state devices over time while it simulates the devices' operating condition in an accelerated manner and is primarily employed for device reliability evaluation [5].

It is rather clear that in all of the aforementioned cases stress inducing stimuli can be proven beneficial for a more efficient stressing of the CUTs (in an internal manner). Yet up until recently, the most commonly applied BI procedure was *static BI*, during which the CUTs are stressed at constant and elevated temperature for an extended period of time without any application of stimulus during the test. However, a drawback of this approach is that nets of the circuit are not exercised [1]. Moreover, as the devices' feature size continues to scale down and their complexity increases, so does the complexity and cost of the BI test, making it unaffordable. Also, the tuning of the key parameters such as duration, temperature and voltage is becoming more and more difficult since it requires a long characterization phase for each new technology, while at the same time, process variations introduce a significant amount of uncertainty. Any error in the BI parameters can be catastrophic since under the wrong conditions, the test can damage the CUTs and thus, result in a yield loss. Regarding the test application time, BI is very time consuming since it

requires many hours and thus, it can become a bottleneck for the whole manufacturing process.

To amend these obstacles, BI is evolving to new forms (e.g., dynamic BI, monitored BI, Test-In BI), where the stress is generated with less dangerous and more controllable actions by also resorting to internal stress [6, 7]. Internal stress can be easily induced by relying on Design for Testability (DfT) infrastructures in the CUT, such as scan chains. However, in these solutions the CUT works in test mode, thus ages in a way which may be different than in operational mode. Hence, such an approach can possibly introduce unnecessary test escapes or even overstress the CUT and possibly cause yield loss. As a conclusion, it is important to devise strategies, able to generate purely functional test stimuli, i.e., specially developed programs able to maximize the switching activity (SWA) while the CUT works in normal mode [8, 9].

Stressing a CUT in an internal manner, i.e., maximizing the switching activity either in the whole circuit or in some parts of it, may be effective even in other test steps. For instance, it has been speculated that testing for delay faults while the circuit temperature is at the top of the allowed range may allow the detection of a higher percentage of defects [10, 11]. More recently, the adoption of System Level Test (SLT) to detect defects that could escape all the traditional test steps leads to the search for functional stimuli able to produce particularly stressful conditions. Once again, this can be achieved by maximizing the internal nodal switching activity and/or creating temperature gradients between different modules within the CUT [12]. Overall, 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.

In this paper, we formalize the problem of the SWA maximization for the case where the CUT is a fully pipelined processor and propose a method based on evolutionary techniques to automatically generate such stress-inducing sequences of instructions (i.e., assembly programs) for various modules of the core. With respect to other solutions, the generated stress programs do not only maximize the switching activity of the targets but can also be repeated an arbitrary number of times. In this way, it is guaranteed that a high value can be achieved for an arbitrary long time period. Hence, we can use the generated test programs to control the CUT temperature. Furthermore, the proposed method generates programs in an automated manner from the ground up, i.e., there are no dependencies with pre-existing software that must be available for the targeted core, as described in [8]. Results gathered on the OpenRISC 1200 (OR1200) processor show that the method can produce high-quality stress programs with an acceptable computational effort. It is also relatively easy to be adopted since it does not require an in-depth knowledge of the processor's architecture and its specifics, like in other approaches, but a knowledge of the processor's Instruction Set Architecture (ISA), only.

The rest of the paper is organized as follows. In Section II we present previous works and the state-of-the-art, while in Section III we formalize the problem and present the base concept of the method. In section IV we describe the proposed method and in Section V we elaborate on its implementation. Lastly, in Section VI we present the experimental setup along with the gathered results. In Section VII we draw some conclusions.

II. Previous Works And Motivation

The significance of the maximum current consumption in integrated circuits (ICs) during the device design and testing phases is a well known problem, strongly linked with the attribute of reliability. Researchers have thoroughly studied the problem in the past. In [13] the authors address the problem of maximum current estimation in MOS ICs. They state that such estimates are crucial for accurate timing analysis and for reliable design of power and ground buses. They propose a heuristic combinational-based approach in order to identify the maximum transient current. In [14] the authors further elaborate on the reliability problems that stem from excessive power and ground currents in CMOS ICs and propose another maximum current estimation method by modeling the problem as an optimization task by trying to obtain stimuli (i.e., test patterns) that maximize the CUT's current waveform.

In [15] 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 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 of the IC. They propose a methodology aiming to identify such stress-inducing consecutive input vectors in order to provoke a high amount of logical switches within the circuit. Their approach is based on automatic test generation algorithms and try to effectively calculate the power estimate of a combinational block in a CMOS IC. In [16] the authors present another 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 also relate the problem of the power dissipation estimation to maximizing gate output SWA while introducing techniques for reducing the problem into a weighted max-satisfiabilty problem and propose strategies to effectively solve it. The methods were applied to relatively small combinational circuits due to the high complexity imposed to obtain the objective function of the circuit and optimize it. In [17, 18] the authors propose methods aiming to maximize the SWA of combinational blocks in order to estimate the circuit's power consumption while considering a variety of delay models.

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 vital information that can enhance the overall reliability of the devices. However, the SWA maximization can be advantageous in the context of device testing as well. During the multiple test steps that are introduced in-between the device manufacturing steps, it may happen that faults do not manifest themselves during testing and escape the whole set of test steps. Such latent

defects are the prime suspects behind the Infant Mortality behavior which is resolved via BI. In [19] the authors present a probabilistic approach (i.e., random excitation of internal nodes) to maximize the SWA of a combinational block to further achieve the maximization of power dissipation during BI testing. In [20] 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 [21] 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 [22] 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 (i.e., stress programs) are applied at-speed to the processor, a significantly higher thermal activity is observed. In [23] we present a method based on formal methods that is able to identify the optimal sequence of two instructions that induce the highest possible switching activity within a targeted processor module. Although this method produces the best-possible result, it requires an in-depth knowledge of the processor's architecture along with its ISA. Also, for the case of certain processor modules (e.g., the multiplier) the method suffers from high complexity times due to the architectural nature of the unit.

In [10] the authors mention that when a delay test is performed on a thermally stressed CUT, it is likely to capture more timing related faults and propose an instruction based self-test (IBST) technique to elevate the temperature of a chip near to functional temperature and test it for delay faults at that temperature. As a case study they use an out-of-order superscalar processor. Lastly, in [11] the authors propose a temperature-aware software-based self-test method, based on automatic test instruction generation, that self-heats the CUT, which is a processor, to a high temperature range and test it for transition delay faults.

To summarize, the SWA maximization problem has been considered with a variety of methodologies in the past targeting both combinational and sequential circuits. We present the case where the CUT is a sub-module of a pipelined processor: in this case the stimuli to be generated correspond to programs. We effectively solve the problem by relying on evolutionary techniques, which can be more easily integrated in an industrial environment and have more general applicability with respect to formal techniques. The proposed methodology generates repeatable stress-inducing sequences of instructions from the ground-up (i.e., we do not require pre-existing programs like in [21]).

III. Problem Definition and Formalization

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. In this paper we propose a method, based on evolutionary techniques for the automatic generation of assembly programs, i.e., sequences of instructions able to maximize the switching activity of a certain module (or sub-module). More precisely, we aim to identify a pair of two instructions (I_1, I_2) that are able (when executed in sequence) to induce high stress in the target module. That is, for the pair to maximize the logical switches performed from the nets of the targeted module from High to Low (HL) and from Low to High (LH).

Since we want to be able to maximize the switching activity for an arbitrary long period of time (clock cycles) in a sustained manner, we further require for the generated sequence of instructions to be *repeatable*. We assume that the processor, after its proper initialization to remove any X (Don't Care) values, e.g., via the activation of the asynchronous reset signal or via the execution of an initialization instruction sequence, is functionally driven to a well defined and legal state s^{α}. Then, the first of the two generated stress-inducing instructions drives it to a new state s^b. Finally, the second instruction drives the processor to a state s^c for which it must hold that $s^c \equiv s^a$ in order for the generated sequence to be a repeatable one. By satisfying the aforementioned equivalence that guarantees a repeatable sequence, we can maintain a sustained high nodal activity within the targeted processor module for an arbitrarily long period of time. The aforementioned sequence of state transitions can be interpreted as a finite state machine (FSM) as depicted below in Figure 1.

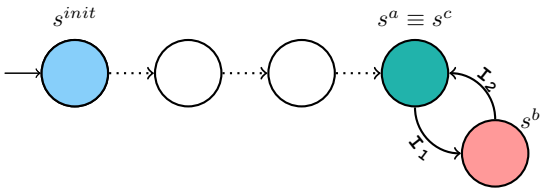

Fig. 1. FSM representation of the SWA maximization of a processor module

Assuming that such a high stress-inducing pair of instructions $(\widetilde{s})$ has been generated, it can be used to stress the target processor module (e.g., to create a hot-spot within the core). Then, a sustained SWA maximization within the module can be achieved by repeating sequence $(\widetilde{s})$ for as many times as required (e.g., $\widetilde{s}, \widetilde{s}, \widetilde{s}, ..., \widetilde{s}$). At the end of the sequence of the repeated instructions, we can introduce an unconditional jump instruction to transfer the code execution back to the start of the stress sequence. Thus, we create an endless loop allowing us to apply the sequence for a large amount of times until it is stopped, e.g., triggering an interrupt.

In general, given a sequence $\widetilde{s}_n$ of *n* instructions and a processor module composed of *m* nets serving as our target, we

aim at maximizing the value of the logical switches performed by these nets by applying the sequence $\widetilde{s}_n$. This can be interpreted as an optimization problem and thus, we can define our objective function, i.e., the function whose value we aim to maximize as:

$$SW = \frac{1}{n \times m} \sum_{i=0}^{m} t_{i,\widetilde{s_n}} \quad (1)$$

$t_{i,\widetilde{s_n}}$: the number of logical switches performed by net i while $\widetilde{s}_n$ was executed

Equation (1) represents the normalized switching activity induced by the sequence $\widetilde{s}_n$. The total nodal switching activity for the module is given by the sum and is normalized to the $[0.0, 1.0]$ range by dividing it with the theoretical maximum given by $(n \times m)$; meaning that all *m* nets of the considered processor module performed a transition when each instruction of the sequence $\widetilde{s}_n$ was executed. This figure represents the theoretical maximum switching value because in practice this value can be limited by the presence of uncontrollable lines inside the module [24]. Since we now have a well defined objective function we can define the optimization problem of the maximization of the switching activity for a specific module of the processor as:

$$\max_{\widetilde{s}_n}\{SW\}$$

Since we have generalized the problem for the generation of a stress inducing sequence of *N* instructions, we are going to elaborate on the ideal sequence length. Let us consider a sequence of 2 instructions $\widetilde{s} = (I_1, I_2)$ and assume that they induce the maximum possible SWA within the target processor module *T* ($SWA^{max}_{\widetilde{s}\to T}$) when executed (i.e., $I_1 \to I_2 \to I_1$). The $SWA^{max}_{\widetilde{s}\to T}$can be broken down as the sum of $SW_{I_1\to I_2}$ that represents the number of nets changing their values when I_2 is executed after I_1 plus the number $SW_{I_2\to I_1}$ when I_1 is executed after I_2. 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, thus it holds that $SW_{I_1\to I_2} = SW_{I_2\to I_1}$. So, maximizing the SWA can be translated to finding the right pair of instructions that induces the highest 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., $N = 2 : I_1, I_2, I_1, I_2$). There is no other sequence of $2 \times N$ instructions that induces a higher SWA in the module.

Proof. Let's assume that a different sequence of instructions $\widetilde{s'} = (I_1^{'}, I_2^{'}, I_3^{'}, I_4^{'})$ exists for the module T that in fact induces a higher $SWA^{max}_{\widetilde{s'}\to T}$ than $SWA^{max}_{\widetilde{s}\to T}$. It holds that:

$$SWA^{max}_{\widetilde{s'}} = SW_{I_1'\to I_2'} + SW_{I_2'\to I_3'} + SW_{I_3'\to I_4'} \quad (2)$$

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

But this implies that a term exists in (2) that has a higher value than any term of (3). However, we showed that the sequence $\widetilde{s}$ is composed of the two instructions that maximize the number of nets switching within the target *T* and so, no term of (2) can have a higher value than any term of (3). ∴ Hence, the sequence $\widetilde{s} = (I_1, I_2)$ is the one maximizing the SWA value for *N = 2*. The same reasoning can be reapeated for higher values of *N*. □

While the goal is the same for every processor module, namely, increasing the module's switching activity over a period of time by forcing the execution of a suitable chunk of instructions, the complexity of the approach, i.e., the search space of the method differs and is specific to every module. Let us assume for example that we want to stress the circuit of the adder of the processor, located in the core's arithmetic and logic unit. In this case, the method's search space can be simplified significantly since for the generated instruction pair we know *a priori* which are the instructions that must be used. These are the add instructions of the processor's ISA and thus, the problem gets reduced to the problem of identifying the appropriate operands that maximize the SWA of the adder during the transition from the first to the second instruction and from the second to the first since the sequence will be also a repeatable one and thus the final functional state in which the circuit is driven after the execution of the second instruction should be equal to the initial one.

On the other hand, assuming that we now want to stress the processor's instruction decode unit then the algorithm should not only consider operands, like in the case of the adder, but a combination of instructions and operands in order to effectively stress it. In fact, the search space should include the whole set of instructions. It is clear that for this case the complexity of the task is higher, since the search space is much larger than in the previous case. One can safely assume that the method's complexity is proportional to the complexity of the tasks performed by the targeted processor module. Furthermore, the search space has to be carefully circumscribed to take into account constraints (e.g., aligned memory addresses) that if not considered could potentially result in the generation of a non-functional sequence. This point will be further discussed later in the paper.

IV. Proposed Method: An Evolutionary Perspective

For the purposes of our task, namely the generation of a sequence of two instructions that maximize the switching activity within a target processor module we developed an algorithm based on the evolutionary paradigm. Given a well defined optimization problem, the evolutionary algorithms, which are efficient heuristic search methods, generate and propose solutions in a manner inspired by nature i.e., the Darwinian evolution. The algorithm initially generates valid solutions to the problem, which are called *individuals*, in a random manner. The group of the generated individuals composes a *generation*. Then, each of the generated individuals is assigned a *fitness* value. The fitness is a function which takes as input an individual and returns a value according to how "fit" or "good" this individual is as a potential solution to the problem we are considering. Then, the whole generation is ranked/sorted according to the corresponding fitness values in order to distinguish between the "good" and the

"bad" solutions. Following the ranking, *selection* takes place. During selection the *k*-best individuals of the generation i.e., the *k* individuals with the best fitness values are chosen to become parent individuals in order to produce *offsprings* (new individuals), which will be part of the next generation. The generation of the new individuals is a result of the application of *genetic operators* on the parent individuals. One of the most common genetic operators is *crossover* [25, 26]. Crossover, which is also called recombination, is an operator inspired by biology that is used to combine information from at least 2 parents in order to create new individuals. Specifically, the newly generated individuals consist of the splicing of the parents characteristics. Before the newly generated individuals are added to the population, the *mutation* [27, 28] takes place which is another genetic operator. The mutation operator, which is analogous to the biological mutation, is used to maintain diversity within the population. It is a probabilistic alternation on the characteristics of the newly generated individuals.

Assuming that we aim to stress a processor module *T*, the proposed algorithm takes as input a (i) set of population settings and (ii) a set of constraints, which is specific for the module *T*. The first set contains crucial information to tune the evolutionary algorithm to the user's and the problem's needs (e.g., population size, number of genetic operators etc.). The latter set contains information about the structure and the syntactical correctness of the individuals and lists potential constraints that the algorithm has to consider in order to generate valid individuals during the generation process. In the context of our work, the target module *T* is a sub-model of a processor and the individuals are stress inducing **assembly programs** for the target *T*. Given the gate-level description of the processor we can cross-compile and evaluate the generated individuals i.e., calculate the number of HL and LH transitions that were performed by every net of the processor's module *T* during the execution of the stress sequence.

Figure 2 depicts the stress program generation algorithm. The algorithm takes as input the set of population settings *S* and the set of population constraints *C* that were previously discussed and also an integer number *Max* that is used as a termination criterion for the algorithm. It represents the maximum allowed steady states for the algorithm. If the evolution process does not yield an individual that is assigned a fitness value greater than the current best one for *Max* generations then the algorithm is forcibly terminated.

```
input  : A triplet (S,C,Max) where
         S is the set of population settings
         C is the set of population constraints
         Max is the maximum allowed steady generations
output : The best generated .asm program P for target T
1  best_old    := null
2  best_new    := null
3  steady_cc   := 0
4  population:= InitializePopulation(S,C)
5  do
6      FitnessEvaluation(population)
7      RankPopulation(population)
8      children ← Crossover(population)
9      Mutation(children)
10     population = population + children
11     best_new = FindMaxFitnessValue (population)
12     if best_old == best_new  then
           // best fitness didn't change
13         steady_cc = steady_cc + 1
14     end
15     else
           // new best fitness
16         steady_cc = 0
17     end
18     best_old = best_new
19 while (steady_cc ≠ Max)
20 return best_old
```

Fig. 2. Stress Program Generation Routine

Fitness Function

The function that we used to characterize the generated individuals is based on Section III. In order to evaluate the switching the generated pairs of instructions induce in the target processor module we launch a logic simulation during which for every clock cycle of the processor we dump tog-files i.e, files that contain information about the toggling activity of every net of the module. Assuming that the processor module consists of *m* nets and the stress sequence is composed of *n* instructions, the fitness function for an individual *X* is calculated as:

$$\textit{Fitness}(X) := \frac{\sum_{i=1}^{m}[HL(i) + LH(i)]}{n \times m}$$

and it holds:

$$\textit{Fitness}(X)\ \varepsilon\ [\mathbf{0.0}, \mathbf{1.0}]$$

The terms $HL(i)$ and $LH(i)$ are functions that take as input the index *i* of a certain net of the target module and return the total number of HL and LH transitions respectively performed by the net when executing the generated instruction sequence.

V. Algorithm Implementation

For the purposes of our work we rely on a software tool called μGP [29] that is used as an underlying engine for the development of our algorithm. It is a general, multipurpose evolutionary optimizer whose initial purpose was the generation of assembly programs for a variety of microprocessors. The optimizer is general and thus it can be adapted to any kind of optimization problem, given that the latter is properly defined. It takes as input a user defined set of rules and parameters (e.g., sets *S, C* of Figure 2) and based on them generates initially random solutions to the problem, which are continuously refined and evaluated during the evolution process. The optimizer is independent on the target problem at hand and thus, it needs an external evaluator to assign a fitness value to each of the generated individuals. The aforementioned environment is depicted in Figure 3.

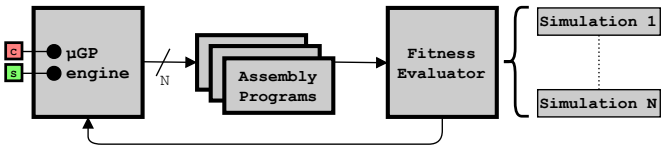

Fig. 3. μGP and Fitness evaluator interaction

A. Population Settings (Set S)

The first of the two sets given to the evolutionary tool as input corresponds to the population settings. It is a collection of parameters that are linked directly to the genetic algorithm such as the population size, the number of genetic operators to be applied on each iteration, elitism and so on. A detailed description of the basic parameters along with their respective values on our experiments are reported in Table I below.

TABLE I
POPULATION SETTINGS

Parameter	Context	Value
ν	Initial size of the population	320
μ	Maximum size of the population	200
λ	Number of genetic operators per every step	120
σ	Strength of mutation operators	0.9
α	Inertia of the self adapting parameters	0.9
$maxAge$	After this limit the individual is forcibly killed	15
$elite$	Number of best individuals that do not get killed	2

The number of individuals to be generated on the initial population during the first iteration is defined by the parameter ν. On every evolutionary step forward λ individuals are being generated by the application of genetic operators and mutation operations. The impact the latter has on the individuals is defined by the parameter σ. After the ranking of the individuals an elimination phase on the population follows in order to comply with the size limit defined by the parameter μ. Also, if a certain individual is within the μ limit, and is not substituted by any of his successors, he gets forcibly eliminated after $maxAge$ generations. The parameter α defines the rate of change of the internal evolutionary core parameters. It is set to a high value in order to prohibit random changes on the self-adapting parameters of the tool. Lastly, we protect from the forced elimination after the $maxAge$ generations $elite$ individuals. These individuals are kept alive until new, improved individuals replace them.

B. Constraints Settings (Set C)

The set of constraints given as the second parameter to the tool regards the set of rules and formats the tool has to consider in order to generate valid individuals. In other words, to ensure that the tool will generate syntactically correct assembly programs under certain constraints. Although the population settings remains unchanged in our experiments, the set of constraints has to be tailored according to the processor module that we aim to stress. In this paper, we are focused on the SWA maximization of 4 modules within the considered pipelined processor (described in the next Section):

- the 32-bit adder
- the 32-bit multiplier
- the instruction decode unit
- the load and store unit (LSU).

The structure of the generated individuals according to all sets *C* is displayed at Figure 4. Initially, registers that will be used by the generated instructions are loaded with their respective values in two steps. First the upper 16-bits of the register) and then the bottom 16-bits. Then if any assumptions are defined in the set *C* they are applied during the next step. For example, a specific write (store instruction) to a designated memory address. We assume full access to the processor's memory. Then, the stress sequence begins with the repetition of the generated sequence of two instructions. In this way, we implicitly guarantee that the sequence of instructions generated by the tool will be repeatable and thus, it will induce high stress amounts in the unit in a sustained manner.

```
_setup:
 movhi REG_A, VALUE_A
 ori   REG_A, VALUE_A
          .
          .
          .
 movhi REG_N, VALUE_N
 ori   REG_N, VALUE_N

_assumptions:
  // e.g. mem access

_stress:

    instruction_1
    instruction_2

    instruction_1
    instruction_2
          .
          .
          .
    instruction_1
    instruction_2

__exit:
        halt
```

Fig. 4. Generated programs' structure

Regarding the adder and the multiplier modules, as mentioned earlier, the problem's search space can be significantly reduced by focusing on the identification of the appropriate pair of operand values for the two addition and multiplication instructions (respectively) that maximize the SWA in the module. Thus, besides mandating the structure of the individuals, the set *C* is used to limit the operands range with respect to the processor's 32-bit register capacity i.e, $\{min, max\} := \{0,\ 2^{32}-1\}$. No further assumptions are considered for the case of these modules.

Regarding the decoding unit module, the search space is significantly larger than for the case of the adder and the multiplier. The set *C* has to contain information about every

TABLE II
EXPERIMENTAL RESULTS

Generation Method	Fitness (%)				CPU Generation Time ≈ hours			
	Adder	Multiplier	Decoding Unit	LSU				
Evolutionary Algorithm	61.34%	54.77%	62.57%	58.00%	12	12	19	15
Stuck-At Test Program (85.00% FC)	24.00%	6.34%	44.43%	57.00%	–	–	–	
	237	2640	880	456				
	Module's Size (# nets)							

possible instruction that is supported by the processor's ISA in order for the appropriate sequence to be identified by the algorithm. Note, that control transfer instructions (e.g., *jumps*) are not considered. It is up to the user to be aware of the ISA of the processor in order to accurately describe all the instructions to be considered and their limitations (if any). For instance, for the register-immediate instructions, the immediate must respect the 16-bit representation and not exceed it; for the register-register instructions the register *$zero* must not be used as a destination register; in general, care must be taken in order to avoid the generation of exceptions.

Lastly, regarding the load and store unit, although there are only a handful of load/store instructions, we have to also consider the interaction with the memory. Initially, the memory of the processor is empty and thus in this case the only switching within the module would stem from the generation of the effective address. For instance, if we consider a load instruction, any kind of switching in the unit responsible for bytes or half sing and zero extensions would not be possible since the fetched memory content would be zero. For this reason, we are also allowing the tool, during the execution of the *_assumptions* label, to store data inside some memory address and thus, we are giving the flexibility to use the data stored in each address if it is indeed found that the switching is higher when the pre-stored memory content is used as stimuli in the *_stress* section. Lastly, the effective address must respect the address bus width and thus the range of the addresses must be also constrained.

VI. RESULTS AND EXPERIMENTAL SETUP

Our experiments were performed on a machine using 2 AMD EPYC 7301 CPUs running at 2.70 GHz. According to Figure 3, the evolutionary core generates a batch of individuals (120) in every step. The evaluation of each individual is an independent task and thus the external fitness evaluator supports heavy parallelization. So, we used 25 threads for the evaluator: in each generation we support 25 concurrent executions of the fitness evaluation tool. For the purposes of logic simulation we used QuestaSIM by Mentor Graphics.

The processor we used as a use case for the application of our algorithm is OpenRISC 1200 (OR1200). The OR1200 is a 32-bit scalar RISC processor of the Harvard micro-architecture. It is mainly intended for embedded, portable and networking applications. The Verilog RT-Level [30] description of the core was synthesized using the Silvaco 45nm Open Cell Library [31] using Design Compiler by Synopsys.

Approximately 500 lines of code were written primarily in `bash`, `python3` and `tcl`, which account for linking the evolutionary core with the external fitness evaluator, for executing the parallel logic simulations, for monitoring and for extracting data from each experiment.

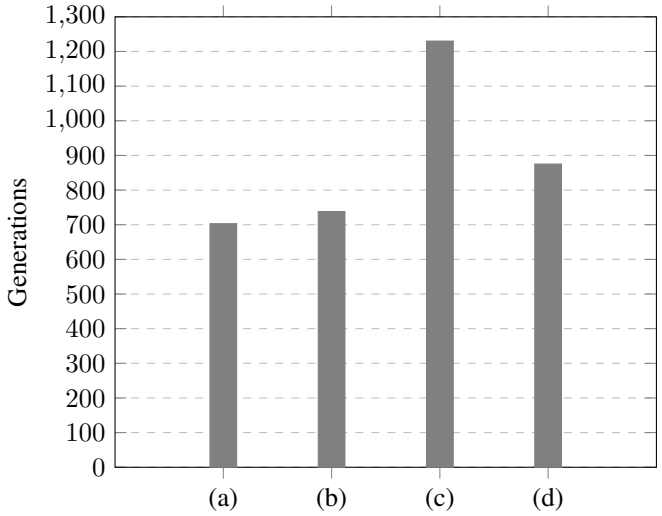

Fig. 5. Total number of generations for the converging of the algorithm for the (a) adder (b) multiplier (c) decoding unit and (d) load and store unit

Figure 5 shows the total number of generations required for the algorithm to converge for each of the targeted processor modules. As we previously explained, for the cases of the adder and the multiplier, we can see that the algorithm required approximately the same number of generations (and CPU time) to converge. Indeed, this seems reasonable considering that the search space was smaller than for the case of the other modules. As expected, the highest run-time was observed for the case of the decoding unit, where we had to consider not only the operands to be used for the module, but also the appropriate combination of ISA instructions. Furthermore, for every one of the target processor modules, the number of maximum steady state generations (*Max* variable in Figure 2) has been set to 200. The reason for selecting such a high value is to minimize the probability of premature convergence of the algorithm that can lead to the generation of solutions (i.e., individuals) that would drastically differ from one-another in consecutive executions of the algorithm. This has been experimentally proven, since after multiple iterations of the method, the algorithm yielded the same best individuals after converging for every processor unit we considered.

In Table II we present our results for the case study of the OR1200 processor and its modules. In order to have a mean of comparison with our results we also further present the stress induced by pairs of instructions derived from a Stuck-At test program written for the same processor that reaches ≈ 85% of

fault coverage. The test program duration is $\approx$ 33,000 clock cycles and thus, in order to sufficiently compare with the short and repeatable sequences we generated we run an extensive logic simulation of the program while classifying for every considered module the toggling for every clock cycle. Then, we post processed all of the generated files and calculated the maximum switching induced in any window of 2 consecutive instructions. We present in the table the highest fitness value found.

VII. Conclusions

In several IC test scenarios the SWA of the CUT must be kept to a minimum to avoid potentially catastrophic conditions such as overheating and over-stressing of the devices. However, there are specific cases in which the maximization of the SWA of the CUTs (or of certain sub-modules) is a major challenge aiming at enhancing the reliability of the overall design. For instance, during BI-test the sustained 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 the evolutionary paradigm able to effectively tackle the case when the CUT is a fully pipelined processor (hence, the generated stimuli are programs) and we aim to maximize the sustained SWA of some sub-modules of the CPU core. We emphasize the fact that the proposed method can be easily tailored to the user's needs since it only requires an understanding of the processor's ISA and not an in-depth knowledge of the CUT's architecture and design specifics (like in [23]) (which may represent a time-consuming task since such information must be communicated to the test engineers from the respective designers). Furthermore, the proposed method does not have any kind of dependencies (e.g., pre-existing code [21]) since it identifies and generates stress inducing sequences from the ground-up. The CPU times required for the convergence of the algorithm are proportional to the available computational resources since the method does support parallelization and thus it has acceptable run-times. We further demonstrate the effectiveness of the generated repeatable stress segments while comparing with segments from a stuck-at test program while focusing on the circuits of the adder, the multiplier, the decoding unit and the load and store unit of the OR1200 microprocessor.

References

[1] R.-P. 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: *Proceedings - 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] "Improving Reliability Throughout the Product Life Cycle". In: *Proceedings - Annual Reliability and Maintainability Symposium*. Vol. 2018-January. 2018. DOI: 10.1109/RAM.2018.8463120.

[5] Brian 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] Mohd 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). ISSN: 07407475. DOI: 10.1109/MDT.2006.50.

[7] Chen He. "Advanced burn-in - An optimized product stress and test flow for automotive microcontrollers". In: *Proceedings - International Test Conference*. Vol. 2019-November. 2019. DOI: 10.1109/ITC44170.2019.9000147.

[8] R. Cantoro et al. "On the maximization of the sustained switching activity in a processor". In: *Proceedings of the 21st IEEE International On-Line Testing Symposium, IOLTS 2015*. 2015. DOI: 10.1109/IOLTS.2015.7229826.

[9] 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, pp. 535–540. DOI: 10.1109/dsd53832.2021.00086.

[10] Nihar 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.

[11] Ying Zhang et al. "Temperature-aware software-based self-testing for delay faults". In: *Proceedings -Design, Automation and Test in Europe, DATE*. Vol. 2015-April. 2015. DOI: 10.7873/date.2015.0744.

[12] Ilia Polian et al. "Exploring the Mysteries of System-Level Test". In: *Proceedings of the Asian Test Symposium*. Vol. 2020-November. 2020. DOI: 10.1109/ATS49688.2020.9301557.

[13] S. Chowdhury and Javed 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). ISSN: 19374151. DOI: 10.1109/43.55194.

[14] Harish Kriplani, Farid Najm, and Ibrahim Hajj. "Maximum current estimation in CMOS circuits". In: *Proceedings - Design Automation Conference*. 1992. DOI: 10.1109/dac.1992.227873.

[15] 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). ISSN: 10638210. DOI: 10.1109/92.661255.

[16] Srinivas Devadas, Jacob White, and Kurt 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). ISSN: 19374151. DOI: 10.1109/43.124424.

[17] 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.

[18] Chuan Yu Wang, Tan Li Chou, and Kaushik Roy. "Maximum power estimation for CMOS circuits under arbitrary delay model". In: *Proceedings - IEEE International Symposium on Circuits and Systems*. Vol. 4. 1996. DOI: 10.1109/iscas.1996.542136.

[19] 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. ISBN: 9780780350939. DOI: 10.1109/TEST.1998.743200. (Visited on 05/29/2021).

[20] A.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. ISBN: 9780780375147. DOI: 10.1109/CCECE.2002.1015257. (Visited on 05/29/2021).

[21] 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, July 2015. ISBN: 9781467379052. DOI: 10.1109/IOLTS.2015.7229826. (Visited on 05/29/2021).

[22] 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. ISBN: 9783981537086. DOI: 10.23919/DATE.2017.7927068. (Visited on 05/29/2021).

[23] Nikolaos 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, pp. 73–78. DOI: 10.1109/ATS52891.2021.00025.

[24] Nikolaos I. Deligiannis et al. "New techniques for the automatic identification of uncontrollable lines in a CPU Core". In: *Proceedings of the IEEE VLSI Test Symposium*. Vol. 2021-April. 2021. DOI: 10.1109/VTS50974.2021.9441040.

[25] Esra'a Alkafaween and Ahmad B.A. Hassanat. "On enhancing genetic algorithms using new crossovers". In: *International Journal of Computer Applications in Technology* 55.3 (2017). ISSN: 0952-8091. DOI: 10.1504/ijcat.2017.10005868.

[26] Luca Manzoni, Luca Mariot, and Eva Tuba. "Balanced crossover operators in Genetic Algorithms". In: *Swarm and Evolutionary Computation* 54 (May 2020), p. 100646. ISSN: 2210-6502. DOI: 10.1016/J.SWEVO.2020.100646. arXiv: 1904.10494.

[27] Nitasha Soni and Tapas Kumar. "Study of Various Mutation Operators in Genetic Algorithms". In: *International Journal of Computer Science and Information Technologies (IJCSIT)* 5.3 (2014).

[28] Siew Mooi Lim et al. "Crossover and mutation operators of genetic algorithms". In: *International Journal of Machine Learning and Computing* 7.1 (2017). ISSN: 20103700. DOI: 10.18178/ijmlc.2017.7.1.611.

[29] Ernesto Sanchez, Massimiliano Schillaci, and Giovanni Squillero. *Evolutionary Optimization: the μGP toolkit*. English. 2011th edition. Berlin ; New York: Springer, Apr. 2011. ISBN: 9780387094250.

[30] *OpenRISC*. https://openrisc.io.

[31] *Silvaco 45nm Open Cell Library*. https://wiki.cse.buffalo.edu/services/content/nangate-open-cell-library.

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.

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 a researcher 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.

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.