Automating the Generation of Functional Stress Inducing Stimuli for Burn-In Testing

Automating the Generation of Functional Stress Inducing Stimuli for Burn-In Testing

Nikolaos I. Deligiannis†, Tobias Faller*, Zhou Chenghan†, Riccardo Cantoro†,
Bernd Becker*, Matteo Sonza Reorda†

† Politecnico di Torino, Department of Control and Computer Engineering (DAUIN) - Turin, Italy
* University of Freiburg, Department of Computer Science - Freiburg, Germany

***Abstract*—In the domain of high reliability applications, Burn-In testing (BI) is always present since it is one of the prime countermeasures against the infant mortality phenomenon. Traditional static BI testing proves to be inefficient for modern circuit designs. As the devices' feature size scales down and their structural and architectural complexity increases, so does the complexity and cost of the BI test. Different BI methods are employed by the industry where stimuli are also applied to the devices under test (DUTs) in order to effectively stress and stimulate all nets of the design. One known industry practice resorts to Design for Testability (DfT) infrastructures (e.g., scan) and is based on the application of test vectors at low frequency to excite the DUT as much as possible with the goal of switching each net of the design at least once. In this paper we consider the case where the layout of the circuit is known and propose two novel methods able to automatically produce functional stimuli to switch pairs of neighboring nodes (i.e., nodes that are placed within a specified distance in the DUT) in short periods of time. This solution has been shown to be able to trigger some latent defects in a circuit better than other methods. As a case study, we target functional units within a RISC-V processor (RI5CY). We show that the functional stimuli generated by the exact method described in the paper are able to achieve optimal results (i.e., the maximum functional switching of neighboring pairs), thus maximizing the chance that their at-speed application can activate weak points in the circuit.**

***Index Terms*—Burn-In Test, Safety, Functional Test, Microprocessor, RISC-V**

I. Introduction

With the continuous growth of technology that characterizes our times, the semiconductor industry has an ever-growing need for producing robust and reliable circuits. This issue becomes even more evident in the domain of the safety-critical applications (e.g., automotive, robotics, avionics, biomedical). In that case, the manufacturer must also guarantee that a system is safe. Hence, the test engineers are tasked with the non-trivial identification and development of the most appropriate test procedures, able to meet the imposed reliability standards while also maintaining a viable cost. The whole set of test routines that is usually adopted by the industry at the end of the manufacturing phase may include Burn In (BI) test. In the domain of the safety-critical applications BI testing is always present.

BI test greatly contributes to the reliability of the final product since it is the main countermeasure against the Infant-Mortality [1] phenomenon. During BI testing the DUT is exercised in elevated thermal and voltage load as it is subjected to different types of external and internal stress in order to artificially age it. Up until recently, the most commonly applied BI procedure was *static* BI, during which the DUTs are exposed to a fixed and elevated temperature for an extended period of time without any application or stimulus during the test. This is achieved by placing the circuits into a climatic chamber that is able to heat the DUTs according to their specification limits. A drawback of static BI is that the circuit is not exercised. As the circuits' feature size continues to scale down and their structural and architectural complexity increases, so does the complexity and the cost of the BI test, rendering it unaffordable. BI test can be very time consuming, since its duration can be in the order of hours (especially for new technologies) and thus it can become a bottleneck for the whole manufacturing process.

To overcome these obstacles, new forms of BI testing are employed by the industry [2]. *Dynamic* stress is also applied to the DUT in a controllable manner. This form of stress is produced by activating the available functionalities of the DUT [3] and when induced in combination to its static counterpart, it can effectively accelerate aging in a controlled manner and thus detect the highest percentage of potential latent defects. A common approach to stress-inducing stimuli generation is based on DfT infrastructures, such as scan. Test patterns are uploaded to the tester machine and then, in an iterative fashion, they are fed to the DUTs through the scan chains at low speed with the goal of switching every net of the design at least once [4]. Regarding functional stimuli used as a stress factor during BI testing [5, 6], it has been proven that the at-speed application of stress-inducing load can in fact cause higher heat gradients in the DUT than a scan-based approach [7]. Another benefit of the purely functional stimulus is the fact that it is not possible to cause damage on the devices unless there exists a fatal design flaw. Furthermore, it has also been proven that functional stimuli can be beneficial for detecting delay faults when the DUT is under high thermal load [8].

In this paper, inspired by the concept of the multi-point stress metric proposed in [9], we propose two novel methodologies (formal methods-based and evolutionary-based) to automate the process of functional stimuli generation, aiming at the maximization of the metric for a functional unit of a scalar, single-issue, pipelined processor. The two methods can be seen as a trade-off between their implementation effort and the optimality of the generated solutions. As benchmark, the two methods are compared with a stuck-at software-based self-test (SBST) [10] program that achieves 95% of functional

stuck-at fault coverage on the whole processor. We targeted two functional units within the processor, namely the adder and the decoder. For the former, both methods identified the optimal stress sequence, while for the decoder the formal method (i.e., the method that yields the optimal results) was proven to be more efficient than the evolutionary solution, while the latter converged to a result close to the case of the SBST program.

The rest of the paper is organized as follows; in Section II we define the problem and elaborate on the employed stress metric; in Section III we present the first of the two methods, based on the evolutionary paradigm and in Section IV we present the second method, based on formal methods. Lastly, in Section V we present the results we gathered on the RISC-V processor we used as a case study and in Section VI we draw some conclusions.

II. Background

Given the gate-level description of a processor and assuming that the layout *L* of the target processor's sub-module *M* is available, then it is possible to know the pairs of nodes which are neighboring, i.e., they are placed within a minimum specified distance from one another. Given a list that contains pairs, with each pair consisting of **two** neighboring nodes, our goal is to generate functional stimuli (i.e., snippets of assembly code) that can maximize the total number of pairs switching within the targeted unit. In other words, for every pair of two nodes, we aim to generate a sequence of instructions that is able to induce both transitions to each pair starting from an initial state. Furthermore, given that during BI test the resources (e.g., tester memory) are limited, we consider the case where the generated instruction sequence length is minimal. This means, the transitions of the nodes must happen over a short period of clock cycles.

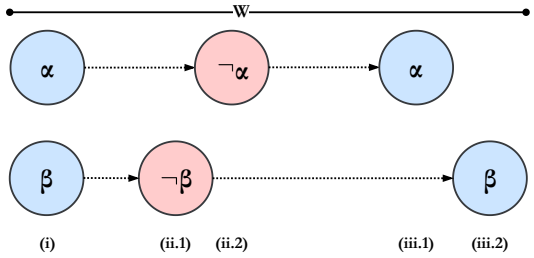

Figure 1. Abstract concept of the pair switching maximization within the DUT

Figure 1 depicts the concept of multi-point switching (MPS) for a pair of nodes in the DUT. We start from a well-defined initial state (i) where both nodes are assigned an initial logic value $\alpha, \beta \in \mathbb{B}$. These values may correspond to those produced by the activation of the system's reset signal or after the execution of an initialization sequence. After the execution of a specific instruction, or combination of instructions (ii.1) the node β performs a transition and is assigned the opposite value from the initial state. The same happens later for the node α (ii.2). It is of course possible that the transition for both nodes can occur in the same clock cycle, i.e., via the execution of the same instruction. Lastly, for both nodes (iii.1, iii.2) another switch is induced that forces the nodes to the opposite logic value than the one they previously reached, i.e., they are assigned the initial values. Also, as mentioned previously, the aforementioned sequence of events must take place in a small time window (W). Our goal is to generate sequences of instructions that induce the maximum number of transitions, as shown in fig. 1, for every pair of nodes in the DUT. Clearly, it is not given that this switching pattern will be possible for every pair of nodes, due for example to uncontrollable lines or to the inability of both nodes of a pair to perform both transitions from their initial states as shown in fig. 1. Hence, we define the stress efficiency (SE) metric for a generated instruction sequence (seq) of a given pair (α, β) of nodes as:

$$SE(seq) := \sum_{i \in \{\alpha, \beta\}} T(i, seq) \mid_{init} \tag{1}$$

where *T* is a function that computes the number of states (transitions) reached from the initial *init* state of node *i* during the application of the sequence *seq*. Thus, the range of values that can be held by the function *T* used to compute eq. (1) is $\{0, 1, 2, 3, 4\}$. For every pair of nodes, we are interested in generating functional sequences for which the SE function gives the maximum value (4), meaning that both nodes were forced to perform both transitions. Note that our metric differs from the MPS metric introduced in [9]. Namely, we do not only consider the case where the nodes of each pair hold opposite initial values. Instead, we generalize by considering whichever initial value for both nodes. By considering eq. (1) as an objective function, we can formally define our goal as an optimization problem:

$$\forall\ pair \in L : \max_{seq_{pair}} \{SE(seq_{pair})\}|_{\#seq \leq W}$$

In this paper we present two methods for solving this problem. The first method is based on an evolutionary algorithm. It provides an elegant way of describing and solving the problem while abstracting from implementation details and architectural information of the DUT. Due to the intrinsic characteristics of evolutionary algorithms, there is no guarantee that the generated result is the optimal. The second method is based on formal methods. Although this method, when compared to the former, does require a better understanding of the DUT's architecture, it does guarantee that the generated result is optimal in terms of stress efficiency, but also in terms of optimal sequence length for the achieved stress efficiency.

III. Method A: Evolutionary Algorithm

Given a well defined problem, the evolutionary algorithms generate solutions in a manner inspired by nature. The algorithm initially generates solutions to the problem i.e., *individuals* in a random manner. The generated individuals compose a *generation*. Every individual of the population is assigned a *fitness* value. The fitness is a function which takes as input the individual and returns a value according to how "good" this individual is with respect to the problem consideration.

After the assignment of the fitness values a ranking of the population takes place in order to distinguish between

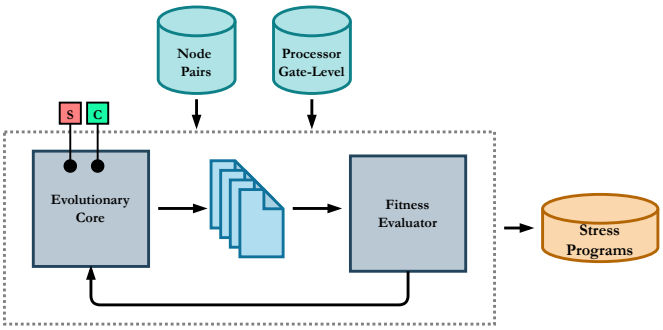

Figure 2. Concept of Method A

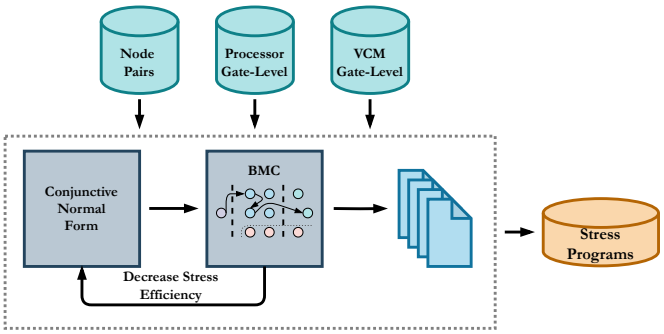

Figure 3. Concept of Method B

the "good" and the "bad" solutions. The individuals with the better fitness values are undergoing a selection process to become parent individuals and to produce offsprings, which will be part of the next generation. The generation of new individuals is primarily the result of the application of the *genetic operators* on the parent individuals. One of the most common genetic operator is the cross-over, during which, the new individuals are generated from the splicing of the parents characteristics. Finally, the *mutation* procedure takes place, which is a probabilistic alternation on the characteristics of the individuals.

The proposed algorithm, depicted in Figure 2 takes the gate-level description of the core along with the layout-derived pairs of two nodes for a functional unit (our stress target) as input. For **every** node pair, an instance of the evolutionary algorithm is launched with a goal of generating a stress-efficient individual for that particular pair. The evolution process is composed by two components:

The evolutionary core is responsible of orchestrating the whole generation process. It is tuned by two user-defined sets: the settings (S) and constraints (C). The former regards the population settings and 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, and so on. The latter regards the set of rules and formats the tool has to consider in order to generate valid individuals. For example, it mandates that the generated instructions are compliant to the processor's instruction set architecture (ISA).

The fitness evaluator is responsible for assessing a batch of individuals (generated by the evolutionary core in each iteration) by assigning to them a fitness value. The implemented fitness function is eq. (1). In order to compute this metric, a logic simulation of each individual on the gate-level description of the core is launched. After each individual is assigned a fitness value, the evaluator reports the ranking of the individuals back to the evolutionary core and the generation process begins anew.

The optimization process is halted when the optimal individual has been generated for the current node pair i.e., the test program for which the fitness function is assigned a maximal value for a certain amount continuous generations (steady-state generations) i.e., the best fitness value of the population was not further improved. An advantage of this method is that it does not require an in-depth knowledge of the underlying architecture but solely a good understanding of the processor's ISA. Furthermore, the algorithm supports parallelization. Namely, during the fitness computation phase for a batch of generated individuals for a specific pair, their evaluation can be done in a concurrent manner. Also, the whole optimization process that is invoked for each pair of nodes can in fact be done in parallel manner since there are no dependencies from one optimization process to another.

IV. Method B: Formal Method

The second method is using Bounded Model Checking (BMC) as an underlying tool. This approach runs an optimization loop for each targeted node pair that maximizes the number of transitions in the pre-defined number of timeframes W by repeatedly solving BMC problems that are constrained to enforce a maximum SE. The maximum SE is decreased until either a solution is found or a SE of 0, being equivalent to the pair being uncontrollable, is found.

The proposed algorithm (shown in Figure 3) takes the gate-level description of the core, the layout derived pairs of nodes for a functional unit and also the gate-level description of a Validity Checker Module (VCM) as input. The processor and VCM gate-level descriptions are transformed into a Conjunctive Normal Form (CNF) for which a valid assignment is sought via a BMC solver. The VCM is specific to the functional unit. It is used to enforce a valid behavior, i.e., functional constraints during the stimuli generation process. The concept of a VCM that is employed in the BMC process has been introduced in [11, 12] and allows specifying functional constraints in a hardware description language. The VCM is connected to the processor via its inputs and observes it. Based on those inputs a validation result is signaled via its so-called constraint outputs. These outputs are constrained to a constant logic value of 1 during the BMC process which forces the BMC solver to apply the functional constraints encoded in the VCM and excluding all non-valid behavior according to the VCM.

The optimization loop starts by constructing a CNF consisting of the processor and VCM gate-level circuit. It starts by assuming the maximum possible SE of 4. If a solution is found it is returned and the optimization loop ends. Should there be a timeout the loop is exited and the pair marked as unknown as no conclusion about the maximum reachable

number of transitions can be made. In the case that the BMC solver determines that there can be no solution that satisfies the number of transitions the SE is decreased by one and the loop is repeated.

After the solver yields a solution for each optimization problem performed for every pair, the values assigned to the bits of the instruction register of the decode stage are decoded to binary values in order to generate the instructions that compose the stress inducing sequence.

V. Results

Method A was implemented using μGP [13] as an underlying framework. The external fitness evaluator module was written in `Python` and for the logic simulation uses `QuestaSim` by Siemens. Method B was implemented using a rewritten version of PHAETON [14] framework. We implemented a prototypical tool inside the framework (accounting for approximately 1,500 lines of `C++` code) for the algorithm described in fig. 3. Our experiments for both methods were performed on a system using two Intel(R) Xeon(R) Gold 6238R CPUs (56 cores, 112 threads) running at 2.20 GHz with 256 GB of RAM.

The processor we used in our experiments is RI5CY [15]. RI5CY is a 4-stage in-order 32-bit RISC-V processor core. The processor's RTL SystemVerilog description was synthesized using the Silvaco 45nm Open Cell Library [16] using `Design Compiler` by Synopsys. Since we did not have the actual layout of the processor available, instead we generated an artificial layout-derived mapping of the nets of the functional unit to be stressed. Without loss of generality, for every unit, we grouped the internal nets and divided them into unique pairs by using a uniform distribution.

As stress target modules within the RI5CY core we used:

- the 32-bit adder, consisting of 538 internal nets that were grouped into $\frac{538}{2}$ = 269 node pairs
- the decoding unit, consisting of 616 internal nets that were grouped into $\frac{616}{2}$ = 308 node pairs.

These functional units were strategically selected in order to showcase the efficiency of the proposed methods. The size of the search space for the optimal sequence required to stress a pair of the modules can be approximated by the Cartesian product:

$$S_{\text{seq}} \leq \prod_{i=1}^{W} S_{\text{instruction}_i}$$

The reader should note that the set $S_{\text{instruction}_i}$ differs according to the functional unit we target. It holds that $S^{\text{adder}}_{\text{instruction}_i} \ll S^{\text{decoder}}_{\text{instruction}_i}$ since the decoder is responsible of handling every instruction of the processor's ISA, which means that the search space for the case of the decoder is much larger than the search space for the case of the adder.

The results of our experiments are shown in Table I. The table shows, for both methods, the percentage of pair of nodes that achieve a given stress efficiency (from 0 to 4). All sequences generated by Method A had a fixed length of W = 10, whereas for the case of the sequences generated by Method B they had a length $W \leq 10$. The ideal window size W is dependent on the underlying micro-architecture. In our case, we experimentally verified that the optimal stress efficiency achieved by both methods does not increase if the window size is bigger than the aforementioned value. The results of both methods were compared with those produced by an SBST program achieving 95% of functional stuck-at fault coverage on the whole processor with a duration of approximately 130k clock cycles. In order to accurately compare our results with the SBST program we performed a coverage profiling logic simulation. Namely, we calculated the number of transitions for all pairs for both units every *W=10* clock cycles. The chunk of instructions achieving the maximum value of eq. (1) is used for comparing with our results.

For the case of the adder we can see that both methods converged to the same results by achieving to force 84.75% of the unit's pairs to all 4 combinations of values in the target time window. Both methods were found to slightly outperform the stuck-at SBST program. For the case of the decoder, Method B outperformed Method A and the SBST program by achieving a notable stress efficiency of 82.14% while the rest achieved 72.07% and 77.60% of optimal switching, respectively. Furthermore, the pair switching is also achieved in a notably shorter period of time than the case of the SBST. For instance, if we concatenate the generated sequences by method A for a given unit into a test program the final size would be $10\times$ *total_pairs* in terms of instructions whereas for method B this would be $\leq 10\times$ *total_pairs*. It is also clear that Method B dominates Method A in terms of CPU runtime. For both test generation procedures Method B converged faster by orders of magnitude than Method A, most notably for the case of the decoder. The justification is the following. Firstly, the way the two methods approach the problem and generate solutions is different. Method A, i.e., the evolutionary algorithm, starts from a completely random (yet valid) set of sequences that are refined in every iteration. Thus, the initial solutions may in fact be far off from the optimal point and hence longer times are required for the algorithm to converge. On the other hand, Method B, i.e., the BMC-based algorithm, starts by searching a solution sensitizing both nodes at the same time. As most node pairs are shown to be fully sensitizable, it is rare that multiple BMC problems need to be solved and a fast reasoning is achieved. Additionally, the BMC algorithm increases the number of timeframes gradually starting by searching for short, easy to compute sequences aiding in reducing the runtime. In theory, through the k-induction and craig interpolation implemented in the BMC solver, unsensitizable node pairs can be found before reaching the maximum depth W, which increases the convergence speed of the optimization loop.

As a reference, we performed a further comparison with DfT-based stimuli. After converting the processor to its scan equivalent, we launched an ATPG process, using `TestMAX` by Synopsys, and considered the generated vectors. By using a test-bench written in `SystemVerilog` we applied the patterns and computed the maximum stress efficiency they induced to the DUTs in a manner identical to the SBST program (i.e., by using a window of W = 10 capture cycles). For the case of the adder, we can see that with scan it is possible to optimally stress pairs of nodes that no functional

Table I
RESULTS

Stimulus	Method	Adder Runtime	Adder Stress Efficiency 0	1	2	3	4	Decoder Runtime	Decoder Stress Efficiency 0	1	2	3	4
Functional	A	6 h	0.74 %	0.00 %	14.58 %	0.00 %	84.75 %	171 h	0.97 %	0.32 %	23.70 %	2.92 %	72.07 %
	B	8 min	0.74 %	0.00 %	14.58 %	0.00 %	84.75 %	15 min	0.65 %	0.00 %	12.34 %	4.87 %	82.14 %
	SBST	–	1.12 %	0.00 %	14.87 %	0.00 %	84.01 %	–	0.32 %	0.32 %	19.16 %	2.60 %	77.60 %
Non Functional	SCAN	–	0.37 %	0.00 %	1.86 %	1.86 %	95.91 %	–	0.65 %	0.00 %	13.27 %	3.56 %	82.52 %

method managed to toggle. The same can be seen for the case of the decoder. This is expected, since as it has been showcased in [7], DfT approaches enable a better stress distribution since they simultaneously exercise many cells in the circuit. Yet, for both cases, we can see that the stress efficiency of the proposed methods is not that far off than the case of scan-induced stress. On the other side, functional stimuli can be applied at-speed to the DUTs (in contrast to scan), thus can be better for exciting possible weak point in the circuit. Moreover, functional stimuli are guaranteed not to stimulate the circuit differently than in the operational mode (thus avoiding any form of overtesting).

VI. CONCLUSIONS

Stress inducing stimuli are required as an internal stress factor during BI test for modern circuit designs in order to effectively and homogeneously sensitize their internal parts and thus detect potential latent defects. The generation of functional stress-inducing stimuli represents a major challenge, since it requires a lot of manual effort from the perspective of the test engineer.

In this paper we considered the case where the layout of the design is known and presented two novel methods that enable the automatic generation of functional stress-inducing code snippets able to force pairs of neighboring nodes to toggle and hold all possible combinations of values. We applied the two methods to generate such stimuli for functional units of a pipelined processor. The generated results show the efficiency of the proposed methods as well as the difference between them in terms of optimality of the stress efficiency and computational cost. Further work is currently being done to consider more functional units and different processor designs.

ACKNOWLEDGMENT

This work was supported in part by the German Federal Ministry of Education and Research (BMBF) within the project Scale4Edge under contract no. 16ME0132 and by the Italian ICSC National Research Centre for High Performance Computing, Big Data and Quantum Computing within the NextGenerationEU program.

REFERENCES

[1] T. M. Mak. “Infant mortality - The Lesser Known Reliability Issue”. In: *13th IEEE International On-Line Testing Symposium*. 2007.

[2] C. He. “Advanced Burn-In - An Optimized Product Stress and Test Flow for Automotive Microcontrollers”. In: *International Test Conference*. 2019.

[3] D. Appello et al. “An Optimized Test During Burn-In for Automotive SoC”. In: *IEEE Design & Test* 35 (2018).

[4] C. He and Y. Yu. “Wafer Level Stress: Enabling Zero Defect Quality for Automotive Microcontrollers without Package Burn-In”. In: *2020 IEEE International Test Conference*. 2020.

[5] N. I. Deligiannis et al. “Maximizing the Switching Activity of Different Modules Within a Processor Core via Evolutionary Techniques”. In: *2021 Euromicro Digital System Design*. 2021.

[6] 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*. 2021.

[7] 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*. 2017.

[8] Y. Zhang et al. “BMC-Based Temperature-Aware SBST for Worst-Case Delay Fault Testing Under High Temperature”. In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 30 (2022).

[9] W. Ruggeri et al. “Innovative methods for Burn-In related Stress Metrics Computation”. In: *2021 16th International Conference on Design & Technology of Integrated Systems in Nanoscale Era*. 2021.

[10] M. Psarakis et al. “Microprocessor Software-Based Self-Testing”. In: *IEEE Design & Test* 27 (2010).

[11] T. Faller et al. “Towards SAT-Based SBST Generation for RISC-V Cores”. In: *Latin American Test Symposium*. 2021.

[12] S. Gurumurthy et al. “Automatic Generation of Instructions to Robustly Test Delay Defects in Processors”. In: *12th IEEE European Test Symposium*. 2007.

[13] *microGP3*. //https://github.com/squillero/microgp3. [Online; accessed 21-Dec-2022].

[14] A. Riefert et al. “A Flexible Framework for the Automatic Generation of SBST Programs”. In: *IEEE Transactions on Very Large Scale Integration Systems* 24 (2016).

[15] *PULP*. https://pulp-platform.org/. [Online; accessed 21-Dec-2022].

[16] *Silvaco 45nm Open Cell Library*. https://si2.org/open-cell-library. [Online; accessed 21-Dec-2022].