EMBER: A Cycle-based Framework for Early-Stage Reliability Assessment in Parametric RTL Designs

Alessandro Veronesi*, Leticia Maria Bolzani Pöhls*, Michele Favalli†, Milos Krstic*‡, Davide Bertozzi§
*IHP – Leibniz-Institut für innovative Mikroelektronik, Frankfurt (Oder), Germany;
‡Universität Potsdam, Potsdam, Germany; Email: {veronesi, poehls, krstic}@ihp-microelectronics.com
†Università degli Studi di Ferrara, Ferrara, Italia; Email: michele.favalli@unife.it
§University of Manchester, Manchester, United Kingdom; Email: davide.bertozzi@manchester.ac.uk

***Abstract*—Modern trends towards higher architectural complexity and smaller technology nodes do not align with the requirement for reliability that many application domains expose. While system robustness remains one of the most critical aspects for mission-critical application domains, the currently available tools allow designers to accurately investigate the reliability only at the late design stages, limiting the effectiveness of their intervention in addressing architectural vulnerabilities.**

In this context, this paper presents a novel framework for early-stage reliability assessment that enables fast evaluations to guide the RTL design process, without compromising analysis quality or relying on imprecise high-level fault injection models. In the presented analysis, the paper shows how the proposed framework leads to a significant time saving when targeting highly parametric hardware designs, achieving up to 79x compile time reduction, and up to 37.6x simulation time reduction when compared to other state-of-the-art approaches.

***Index Terms*—Fault Injection Simulations, RTL Simulations, Reliability, Mission-Critical Systems, CAD**

I. Introduction

The complexity of modern embedded systems is increasing due to the demanding computational power related to emerging applications' requirements [1], [2]. In more detail, emerging applications demand high performance with strict constraints in terms of area, power consumption, and timing, increasingly foster the adoption of advanced technology nodes, which also present higher fault susceptibility [3]. Therefore, the computational integrity of those systems is currently jeopardized by possible in-field faults that can compromise application's reliability, preventing their adoption in mission-critical domains.

As traditional hardening techniques, based on redundancy, are revealing cost-ineffective due to the design complexity, we can observe a surging need for tailored hardening techniques, which provide strong reliability enhancements with minimal introduced overhead [4]. This trend is particularly evident in the field of highly parallel architectures, where the performance-reliability-cost trade-off can be explored just by manipulating the design configuration space, without introducing any additional hardware overhead [5]. Therefore, we can identify a rising momentum for methodologies and workflows that are not confining reliability analysis exclusively to a evaluation phase, but that foreshadow robustness assessment since the early design stages [6].

As analytical methodologies struggle to capture the complexity of the fault propagation through the digital circuit and its implications on the final application, simulation-based fault injection remains the most valuable asset that designers are provided to assess the system robustness since the initial phase of the hardware design [6]–[8]. However, widely adopted event-driven simulators struggle in keeping pace with the growth of the Design Under Test (DUT) complexity, exposing prohibitive simulation time and finally, preventing the maturation of reliability-driven design methodologies.

In this panorama, cycle-based simulations offer an appealing opportunity. This simulation paradigm presents a fast evaluation time compared to traditional event-driven approaches, while maintaining cycle accuracy in the simulation, fundamental for consistency with most commonly adopted fault models (e.g., Stuck-At Faults (SAFs), Single Event Upsets (SEUs), Multi-bit Upsets, etc.). Additionally, because of the elevated simulation speed it offers, designers may avoid reliance on highly abstract frameworks, which are currently the only available option to investigate hardware/software interactions, but whose applicability for reliability assessments is limited to "what-if" scenarios [7], [8].

In this paper, we present the Extensible Microarchitectural Backend for Early-stage Reliability analysis (EMBER), a novel C++ RTL modeling and simulation framework with a built-in cycle-based simulation kernel and native support for fault injection through the adoption of saboteurs, designed to be easily extensible for representing different fault models associated to manufacturing deviations at time zero as well as to time-dependent deviations during lifetime[1]. The proposed framework was evaluated assuming as case study the reliability assessment of Deep-Learning Accelerators (DLAs) with respect to SEUs in their pipeline. In fact, those highly parallel components offer a competitive challenge for reliability assessment, due to the complex propagation of silent errors and to the prohibitive simulation time required when adopting event-driven approaches. The obtained results show that EMBER offers a simulation compile time reduction of 79x and a runtime

The work presented in this report has received funding from the European Union's Horizon Europe research and innovation program under grant No. 101092598 (COREnext).

[1]Available at link: https://github.com/IHP-Neuromorphic/EMBER

reduction of up to 37.6x compared to state-of-the-art simulators, when considering the adopted benchmark, which consists of 5 different convolutions and 18 different DLA configurations. Thus, compared to other state-of-the-art approaches in this domain [9], [10], EMBER offers significant improvements in the time required for compiling and running the simulations. For this reason, EMBER represents a better candidate for driving future reliability-aware hardware/software co-design strategies.

This paper was structured as follows: after a brief presentation on the background theoretical knowledge in Section II, we present the proposed framework in Section III. Following, the selected case study and its discussion are presented in Sections IV and V respectively. Last, we will draw our conclusions in Section VI.

II. Background

Simulation-based fault injection is a commonly adopted experimental technique for performing reliability assessment during the DUT simulation. State-of-the-art approaches for simulation-based fault injection can be divided into two main groups: script-based, and tool-based [11]. *Script-based* approaches implement the fault injection support during the simulation by manipulating the DUT signals. This is usually achieved via tool commands implemented in `tcl` simulation scripts written by the user. *Tool-based* approaches are similar to script-based, with the difference that the fault injection support is embedded with the tool itself by its designers and provides the user with simplified control GUIs or APIs. Nevertheless, this easy adoption comes at the cost of lower controllability over the fault insertion process and of limited support w.r.t. the available fault models.

As first solution to cope with the demanding evaluation time traditional simulation-based fault injection exposes, *emulation-based* fault injection has been proposed [12]. This approach mainly consists of running the fault injection experiment on an FPGA-synthesized DUT. The fault injection is achieved via the adoption of synthesizable *saboteurs*, which are dedicated hardware components customized for this purpose. While this approach allows fast evaluations (i.e., the evaluation time is the actual execution time of the synthesized design), it requires both the design and the saboteurs to be entirely synthesizable. Additionally, as saboteurs are synthesized components, their control requires dedicated control input ports, finally increasing the overall design complexity. In more detail, this limits the variety of the injected fault models and the applicability of the approach to the late design stages.

A different approach to reduce the fault injection campaign duration that has recently become popular, consists in the adoption of High-Level Models (HLMs) for fault injection simulations. This approach majorly differs from traditional simulation-based fault injection due to the adoption of microarchitectural HLMs, tipically Gem5 for processors [7] or dataflow descriptions for DLAs [8], instead of RTL or gate-level models. In this case, the fault injection is achieved via the usage of highly abstracted saboteurs. The adoption of HLMs for hardware simulations allows evaluations that

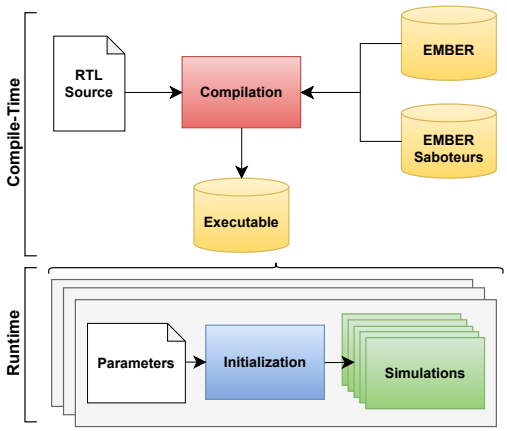

Fig. 1: EMBER workflow, from RTL design to parallel fault simulations for different hardware configurations.

are usually orders of magnitude faster than event-driven RTL simulations, and also enables reliability assessments even before RTL availability. However, this further introduces additional layers of abstraction from the final design, and therefore limits their applicability to *"what-if"* scenarios. Additionally, because of the high discrepancy between the HLM and the final design, the saboteurs adopted for those models require demanding validation against more mature designs.

Last, fault injection frameworks leveraging cycle-based simulation tools have been proposed [9], [10]. This strongly reduces the evaluation time, while not trading for the design accuracy as HLM approaches. However, those works only present tool-based customizations, thus exposing typical limitations w.r.t. the variety of possible fault models and their controllability. At the same time, none of the above mentioned approaches has been developed including features which are typically required by the RTL designer at the preliminary stages [13]. Therefore, to the best of our knowledge, this is the first work presenting a complete framework that combines the possibility of performing design exploration with capability of conduct reliability assessment, extensible fault models as well as easy hardware/software co-simulation capability.

III. The EMBER Framework

The proposed framework for early-stage reliability assessment of emerging applications is presented in this Section. The key aspects related to EMBER with respect to traditional workflows are the following:

- Adoption of cycle-based RTL simulation in contrast to mainstream event-driven simulation;
- Introduction of a dedicated step for initialization of design parameters, after design compilation, instead of compilation-time parameters resolution.
- Use of simulation-friendly built-in *saboteurs* for fault injection, which differs from tool customization-based approaches [9], [10].

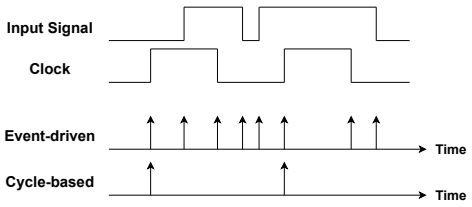

Fig. 2: Simulated events for event-driven and cycle-based approaches.

Thus, in order to support the aforementioned characteristics, EMBER was developed as a C++ hardware description framework, with built-in support for fault injection as well as features for architectural design-space exploration. Figure 1 depicts an overview of the EMBER's workflow including the framework's main software components. In the next paragraphs, we will present the principles of cycle-based simulations, the EMBER approach to the design compilation of the simulation binary and the framework's primitives for hardware modules and saboteurs design.

A. Cycle-based Simulations

While most of the hardware simulation engines rely on event-driven simulations, EMBER adopts a cycle-based approach to hardware simulation, which differs from the event-driven paradigm for the way the simulation processes are scheduled. In fact, event-driven simulations are composed of simulation processes that describe different hardware components' functionalities. In Verilog and VHDL simulation processes are usually obtained after sequential processes descriptions (i.e., `always` and `process` blocks in Verilog, and VHDL, respectively) or from dataflow-like descriptions, such as signal assignments. Every simulation process is then associated with a sensitivity list, which corresponds to a set of signals that usually play the role of inputs for the associated outputs of the process. Thus, when assuming event-driven simulations, once a signal commutes, it generates an associated event that calls for the *events scheduler*. The events scheduler then executes all processes having the triggered event in their sensitivity list, calculating the next value for the process output signals, which are then updated at the next delta cycle.

Differently, cycle-based simulation processes are exclusively sensitive to the clock signal. In fact, hardware components' connectivity is resolved before starting the simulation and triggered processes are pre-calculated during the compilation phase. An example of triggered events for event-driven and cycle-based simulations is presented in Figure 2. Thus, cycle-based approach enables a pre-scheduling of the simulation execution, resulting in lighter runtime simulation engines, both because: (a) it contains fewer events to process; (b) the highly predictable simulation allows to remove the event scheduler.

While granting cycle-accuracy, the major drawback is the insensitivity of this paradigm to events that happen between two clock cycles, discouraging its deployment for post-synthesis and post-layout timed-annotated simulations. Nevertheless, cycle-based simulations are very commonly adopted for functional verification, and can also be deployed for fault injection campaigns [9], [10], since many fault models are either insensitive to time, or are injected at the beginning of the clock cycle to reflect worst-case scenarios.

B. Design Parameters Initialization Step

Modern hardware designs are highly parametric. In more detail, the configuration space of a design can vary from the number of bits per signal to the number of parallel ALUs for SIMD architectures, to the choice among possible implementations of a multiplier unit. The exploration among different possible architectural configurations is considered a crucial aspect during the design phase and can lead to designs that strongly differ in performance and cost. Additionally, the design parallelism also has a strong impact on the fault propagation. This can be particularly appreciated in SIMD architectures, as it has been observed in [5]. Other relevant aspect is related to the fact that most of the hardware description languages, such as Verilog or VHDL, support design parametrization via pre-processor directives and compile-time polymorphism only. Thus, simulation recompilation is required for the evaluation of different design configurations. Differently, we propose the introduction of a dedicated step for *Design Parameters Initialization* after compilation, and before the simulation run. This characteristic allows not only faster and more interactive exploration of designs' trade-offs, but also easy integrations in domain-space exploration automations, while preserving the cycle-accuracy of RTL simulations.

C. EMBER Modules and Saboteurs Descriptions

Different from other cycle-based frameworks [9], [10] which are focused on tool customizations, EMBER adopts saboteurs for the support of fault injection. An example of D-type Flip-Flop including a saboteur for SEUs is presented in Figure 3. Designing hardware modules and saboteurs with EMBER is achieved by inheriting C++ abstract classes and by implementing their abstract methods. In more detail, EMBER components are described as C++ classes implementing the `ember::IModule` interface. This interface requires the designer to describe the component's ports through dedicated objects and to implement the component behavior via the `update()` (i.e., for describing sequential elements), the `eval()` (i.e., for describing combinational elements), and the `reset()` (i.e., for behavior during the reset phase) methods. The `getSaboteurs()` method is designed to recursively return the saboteurs objects' references through the whole design hierarchy. The class constructor is used for implementing the *design parameters initialization* step. In addition, the `ember::ISaboteur` abstract class is instead used for implementing fault injection features through the `genFaultMask()`, `clearFaultMask()` and `applyFaultMask()` methods. Note that this class is additionally templated according to the desired fault model, but the implementation is left entirely to the EMBER module designers. This allows for extensibility w.r.t. the support for multiple fault models, while replacing the need for dedicated

```
class FF_typeD :  public ember::IModule,
            public ember::ISaboteur<ember::seu> {
 protected:
    bool seu_mask;
 public:
    FF_typeD(const char* name) :
        IModule(name), ISaboteur<ember::seu>(),
        D(), Q(), Qn() {}

    // IO Ports declaration
    ember::inPort<bool> D;
    ember::outPort<bool> Q, Qn;

    // Methods - EMBER Module
    void update() {
        Q.write(D.read()); Qn.write(!D.read());
    }
    void eval() { // Do Nothing }
    void reset() {
        Q.write(false); Qn.write(true);
    }
    ember::ISaboteur* getSaboteurs<ember::seu>() {
        return this;
    }

    // Methods - EMBER Saboteur
    void genFaultMask() { seu_mask = true; }
    void clearFaultMask() { seu_mask = false; }
    void applyFault() {
        Q.write(Q.read() ^ seu_mask);
        Q.write(!(Q.read() ^ seu_mask));
        clearFaultMask();
    }
};
```

Fig. 3: Example EMBER module description for a D-type FlipFlop supporting SEUs injection.

control input signals with simple class methods. Therefore, supporting additional fault models and saboteurs results as easy as inheriting and extending the related abstract class. Resuming, those implementation decisions also bring additional consequences to the workflow:

- By exploiting runtime polymorphism, the designer can explore different hardware configurations and dynamically instantiate sub-modules during class construction via constructor parameters, enabling with a single simulation binary to explore large configuration spaces.
- The implementation of a saboteur behavior is delegated and tailored to a specific hardware module, potentially allowing highly accurate representations of the components' failure at the system level.

Currently, EMBER supports the injection of SEUs and SAFs saboteurs not only for Flip-Flops, but also for dual-port Static Random Access Memories (SRAMs).

IV. Case Study: SEUs in Deep-Learning Accelerators

In order to evaluate EMBER potential as a framework for early-state design exploration and to validate its capability for reliability assessment, a DLA including a pipeline based on the NVDLA architecture was adopted as case study. Figure 4 depicts the studied DLA block diagram.

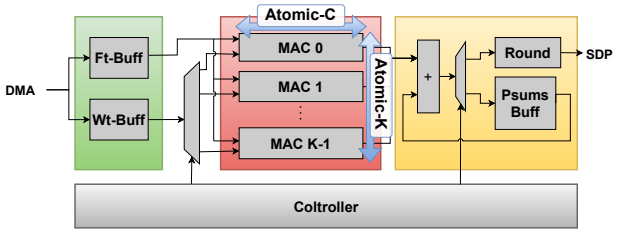

Fig. 4: Block diagram of the evaluated DLA pipeline.

The DLA execution starts with the fetch of all features and weights data from a external memory to the internal buffers, named *Ft-Buff*, and *Wt-Buff* for input features and weights data, respectively. Following, the weight matrix is sequentially loaded from the *Wt-Buff* to the MAC array, while several input feature vectors are loaded sequentially, thus executing a sequence of Vector-to-Matrix Multiplication operations. The partial results, *Psums*, are then stored to the accumulation buffer, also referred to as *Psums-Buff*, until the operation is finished, and finally delivered to the Single-Data Processor (SDP) for post-processing. Note that the pipeline can be configured for different buffer sizes, computing precisions as well as number of input and output elements of the MAC array, which are referred to as *Atomic-C*, and *Atomic-K*, respectively.

Thus, in order to demonstrate the fault injection capability and performance of the proposed framework, the impact of SEUs on the behavior of different DLA configurations was analyzed. Table I summarizes all DLA configurations adopted during the experiments. In more detail, several fault injection campaigns injecting SEUs on the whole architecture have been performed. It is important to highlight that fault injection capability was achieved through the use of dedicated saboteurs able to implement bit-flips in FFs, which represents a approach commonly adopted [9], [10], [12]. Figure 3 show a saboteur example used for performing DLA's reliability assessment.

V. Obtained Results and Discussion

Assuming the cased study previously described, the evaluation of the proposed framework performance have been performed with respect to two state-of-the-art approaches: cycle-based simulation via Verilator and event-driven simulation via Siemens®QuestaSim. The comparison has been performed on a machine running an Intel®Xeon®w7-2475X equipped with a 64 GB DRAM. The performance evaluation of the proposed framework includes the analysis of two aspects, the compilation and the simulation time for every DLA's configuration presented in Table I, totalizing 18 different

HW Parameter	Value(s)
Atomic-C	8, 16, 32
Atomic-K	8, 16, 32
Wt-Buff	512 kB
Ft-Buff	64 kB
Psums-Buff	64 kB
Precisions	`int8, int16`

TABLE I: Evaluated DLA Configurations: a total of 18 configurations were analyzed.

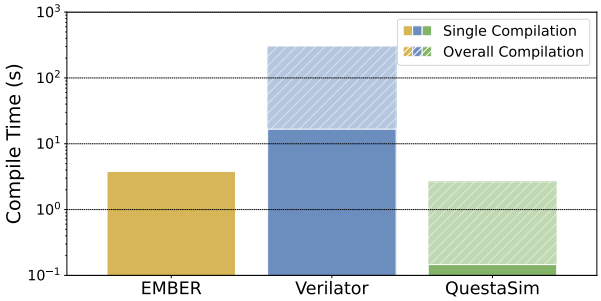

Fig. 5: Compile time for all the evaluated approaches for both single and multiple DUTs.

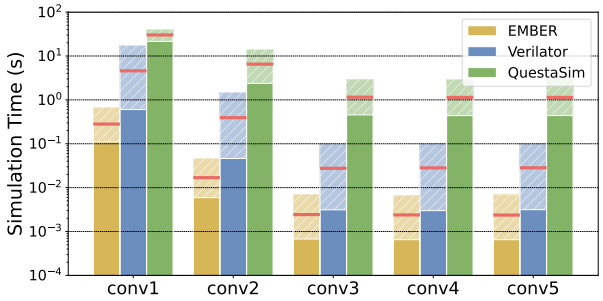

Fig. 6: Simulation time related to all evaluated convolutional benchmarks assuming all DLA configurations.

hardware configurations. Since the adopted DLA accelerates convolutions, 5 different convolutional benchmarks have been adopted. Table II summarizes the adopted convolutional benchmarks, extracted from an AlexNet model, trained for CIFAR10. As a final notice, the simulation time has been assessed on its single-core performance. This choice is motivated by the fact that multi-core acceleration in fault simulations is achieved via job-level parallelization, where the fault injection campaign is split into smaller parallel campaigns, instead of parallelizing the single simulation run.

A. Compilation Time

Figure 5 depicts the compilation time assuming all evaluated DLA configurations for all three considered approaches. To maintain a fair comparison, we evaluated the compile time on its single-core performance, without optimization or debug flags activated. As can be easily observed, while the compilation time of EMBER and Verilator remains comparable (i.e., 3.8s and 16.7s, respectively), the compilation time of QuestaSim is significantly smaller (i.e., 145.9ms). This behavior is caused by the strongly different compilation approach adopted by QuestaSim. While QuestaSim performs translation of the Verilog code to pre-compiled simulation primitives in order to generate simulation processes, a different strategy is adopted by EMBER and Verilator. Latter require a full *Ahead-of-Time* compilation, which involves the invocation of both the C++ compiler and the linker to convert the high-level language source code into an executable binary. When compared to Verilator, EMBER results slightly faster. This is mostly caused by the Verilog to C++ conversion that Verilator performs.

Nevertheless, when approaching a wide hardware configuration space, the situation significantly changes. In fact, while QuestaSim and Verilator avoid recompiling every file of the evaluated design, still the compilation time linearly grows with the number of assumed configurations, as can be observed from the hatched bars of Figure 5. In more detail, Figure 5 reports the total compilation time for all adopted configurations. It is important to highlight that differently from Verilator and QuestaSim, EMBER does not require recompilation. This is because EMBER design remains a parametric description, even after compilation, allowing all hardware parameters (e.g., memory sizes, signal bitwidth, etc.) to be specified at runtime.

Finally, it is important to mention that overall, EMBER does not offer an advantage compared to QuestaSim in terms of compilation time, since the latter remains 69% faster (i.e., it requires 2.6s in total) due to the fast compilation time for a single DLA configuration. However, compared to other cycle-based approaches, EMBER is 79x faster, since Verilator requires 300.6s in total for compiling all DLA configurations.

B. Simulation Time

To properly assess the simulation time required by all three different evaluated approaches, We run all the benchmarks assuming all possible hardware configurations. The simulation time results are presented in Figure 6, where for each benchmark, we highlighted the minimum, maximum, and average simulation time with respect to different hardware configurations. As expected, the event-driven approach results in the slowest one, with an overall simulation time that varies from 438.1 ms to 40.4 s for the *conv4* and for the *conv1* benchmarks, respectively.

The cycle-based approach exposes significant advantages, as can be observed from the performance achieved by Verilator. In fact, compared to the QuestaSim approach, the verilated designs are generally one to two orders of magnitude faster, exposing a simulation time reduction that varies from 2.3x to 147.8x for the *conv1* and for the *conv4* benchmarks, respectively.

Nevertheless, EMBER further improves the simulation time beyond Verilator performance, accelerating the benchmarks of up to one more order of magnitude. The reason behind this performance boost is primarily the C++ hardware description itself. In fact, the proposed framework allows the use of C++ built-in data-types to model hardware signals. This results in a better use of the processor's resources when performing the arithmetic operations. Instead, Verilator uses only the same data-types that Verilog provides, limiting the vision on the datapath operations as binary descriptions and allowing

	Feature Tensor (BxCxHxW)	Weight Tensor (KxCx$\tilde{H}$x$\tilde{W}$)	Stride, Padding, Dilation
conv1	(1x3x32x32)	(6x3x11x11)	1, 2, 1
conv2	(1x6x12x12)	(192x6x11x11)	1, 2, 1
conv3	(1x192x5x5)	(384x192x3x3)	1, 1, 1
conv4	(1x384x5x5)	(256x384x3x3)	1, 1, 1
conv5	(1x256x5x5)	(256x256x3x3)	1, 1, 1

TABLE II: Adopted convolutional benchmarks.

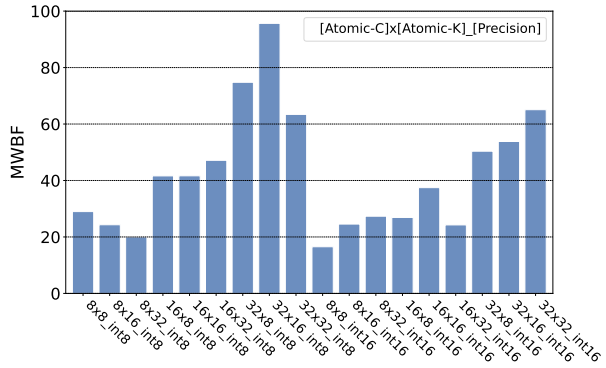

Fig. 7: Mean Work Between Failure as number of fault-free convolutions between failures for different DLA configurations.

their logical interpretation only in rare opportunities. Overall, EMBER guarantees a performance boost that results on average 12.6x faster than Verilator and 437.6x faster than QuestaSim. With a simulation time boost that ranges from 4.6x to 37.6x when compared to Verilator, and a boost that ranges from 48.7x to 813.2x when compared to QuestaSim.

C. Reliability Assessment

To demonstrate the use of EMBER as a powerful framework for performing reliability assessment of SIMD architectures during the design stage, SEU injection campaigns over all hardware configurations have been performed.

The reliability assessment consisted of 1000 fault injection campaigns introducing single SEU assuming all hardware configurations, which represents 18000 simulations for every benchmark. For analysis's simplicity, we assumed all configurations to run at 250 MHz, considering a 40nm technology node. We present the results for the only *conv3* benchmark, as it resulted the most sensitive against SEUs.

Figure 7 presents the computed Mean Work Between Failure (MWBF) for different DLA configurations. In more detail, in order to understand the impact of SEU, the MWBF was calculated according to the following equation.

$$MWBF = CPS \div (BER_{ff} \cdot N_{ff} \cdot PVF) \quad (1)$$

where CPS are the DLA's Convolutions-Per-Second while running the *conv3* benchmark; BER_{ff} is the Bit-Error-Rate of a single FF (e.g., 600 FIT/MB for a 40nm technology [14]), and N_{ff} is the number of FFs in each design; and last, PVF is the Program Vulnerability Factor observed from the SEU injection campaign[2]. As reported from our previous work [5], although the evaluated DLAs configurations are buffer dominated, resulting in almost constant N_{ff} (i.e., memories occupy more than 99% of the total amount of SEU locations), the PVF may still strongly vary across configurations.

First, we observe how `int16` configurations expose higher fault vulnerabilities compared to `int8`, as they require larger internal pipelines, mostly storing more sensitive sign bits. Second, we can observe how larger configurations (i.e., higher values of *Atomic-C*, and *Atomic-K*) expose shorter execution time, thus resulting in higher CPS. Last, as also observed in [5], growing values of *Atomic-K* increase the error propagation probability. Therefore, architectures with an equivalent number of total MACs (i.e., calculated as *Atomic-C* $\times$ *Atomic-K*) present higher MWBF for smaller *Atomic-K* values. Consequently, the optimal DLA configuration may significantly vary given reliability, cost, and performance requirements the overall system presents. In fact, for the selected benchmark, the MWBF varies from 20.0 to 95.7 in the case of `int8` pipelines, and from 16.5 to 65.1 in case of `int16` pipelines.

[2]To be noticed, $FIT = BER_{ff} \cdot N_{ff} \cdot PVF$. From [8]

VI. Conclusions

This paper presented a new framework that enables reliability assessment at early-stage design phase, making the development of highly reliable SIMD architecture possible. Differently from other state-of-the-art tools, EMBER compilation approach results strongly scalable against the size of the design configuration space. At the same time, for the considered benchmark, EMBER exposes a simulation time reduction that resulted up to 37.6x faster than Verilator and up to 813.2x faster than QuestaSim. Therefore, the proposed framework simplifies the integration of fault injection campaigns and consequently allows to perform reliability assessment of different DUT configurations at the early stages, enabling trade-off explorations from preliminary design phases.

References

[1] Z. Zhang and J. Li, "A review of artificial intelligence in embedded systems," *Micromachines*, vol. 14, 2023.
[2] C. Galagain, M. Poreba, and F. Goulette, "Is semantic slam ready for embedded systems? a comparative survey," 2025.
[3] A. Zimpeck *et al.*, *Mitigating Process Variability and Soft Errors at Circuit-Level for FinFETs*. Springer, 2021.
[4] P. M. Aviles *et al.*, "Hardening architectures for multiprocessor system-on-chip," *IEEE Transactions on Nuclear Science*, vol. 71, 2024.
[5] A. Veronesi *et al.*, "Cross-layer reliability analysis of nvdla accelerators: Exploring the configuration space," in *IEEE European Test Symposium (ETS)*, 2024.
[6] S. Sudhakara Hegde *et al.*, "Early reliability assessment of ai-based automotive systems," *ACM Transactions on the Internet of Things*, vol. 1, 2025.
[7] M. Kaliorakis *et al.*, "Differential fault injection on microarchitectural simulators," in *IEEE International Symposium on Workload Characterization (IISWC)*, 2015.
[8] A. Veronesi *et al.*, "Exploring software models for the resilience analysis of deep learning accelerators: the nvdla case study," in *International Symposium on Design and Diagnostics of Electronic Circuits and Systems (DDECS)*, 2022.
[9] S. Nema *et al.*, "Eris: Fault injection and tracking framework for reliability analysis of open-source hardware," in *IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2022.
[10] E. Kaja *et al.*, "Extending verilator to enable fault simulation," in *Workshop on Methods and Description Languages for the Modeling and Verification of Circuits and Systems (MBMV)*, 2021.
[11] H. Ziade *et al.*, "A survey on fault injection techniques," *Int. Arab J. Inf. Technol.*, vol. 1, pp. 171–186, 01 2004.
[12] C. Lopez-Ongil *et al.*, "Autonomous fault emulation: A new fpga-based acceleration system for hardness evaluation," *IEEE Transactions on Nuclear Science*, vol. 54, 2007.
[13] A. D. Pimentel, "Exploring exploration: A tutorial introduction to embedded systems design space exploration," *IEEE Design & Test*, vol. 34, 2017.
[14] S. Jagannathan *et al.*, "Frequency Dependence of Alpha-Particle Induced Soft Error Rates of Flip-Flops in 40-nm CMOS Technology," *IEEE Trans. Nuclear Science*, vol. 59, pp. 2796–2802, 2012.