A Reliability Evaluation Flow for Assessing the Impact of Permanent Hardware Faults on Integer Arithmetic Circuits

IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 24 December 2024, accepted 23 January 2025, date of publication 27 January 2025, date of current version 21 February 2025.

Digital Object Identifier 10.1109/ACCESS.2025.3534274

RESEARCH ARTICLE

A Reliability Evaluation Flow for Assessing the Impact of Permanent Hardware Faults on Integer Arithmetic Circuits

NIKOLAOS I. DELIGIANNIS[1], (Member, IEEE), JUAN-DAVID GUERRERO-BALAGUERA[1], (Member, IEEE), RICCARDO CANTORO[1], (Member, IEEE), SERAG E. D. HABIB[2], (Life Senior Member, IEEE), AND MATTEO SONZA REORDA[1], (Fellow, IEEE)

[1]Department of Control and Computer Engineering (DAUIN), Politecnico di Torino, 10138 Turin, Italy
[2]Department of Electronics and Electrical Communications Engineering, Cairo University, Giza 12261, Egypt

Corresponding author: Nikolaos I. Deligiannis (nikolaos.deligiannis@polito.it)

This work was supported by the National Resilience and Recovery Plan (PNRR) through the National Center for High Performance Computing (HPC), Big Data and Quantum Computing.

ABSTRACT Arithmetic circuits are fundamental building blocks in modern digital computers, allowing for precise mathematical operations and driving the digital age. They are essential components in almost every digital device, from basic CPUs to advanced accelerators in AI applications. In particular, in safety-critical fields like automotive, avionics, and medical equipment, the flawless operation of these circuits is paramount to ensure the correct system operation. Unfortunately, silicon devices manufactured with cutting-edge technologies are more likely to be affected by faults. Their effects can eventually produce computational errors and lead to catastrophic consequences. Since arithmetic modules play an important role in AI-oriented circuits, it is crucial to study the degree of severity of the fault-induced errors affecting them, both to estimate the achieved level of safety and to support the development of effective fault mitigation mechanisms. The analysis can also be used to guide designers in the selection of the most suitable arithmetic module for a given application. Also, the fault evaluation process is an essential part of safety methodologies like *failure mode effects and criticality analysis* (FMECA). This work proposes a method to analyze the inherent reliability of arithmetic modules, and uses as case studies four multiplier circuits: two variants of 32-bit Dadda multipliers (`Circuit A` and `Circuit B`) and two variants of 8-bit Booth multipliers (`Circuit C` and `Circuit D`). The proposed flow allows us to comprehensively assess their reliability and evaluate the impact and severity of permanent faults affecting the selected circuits. The analysis is based on extensive fault injection campaigns using both random values and traces from DNN workloads as input operands. We evaluated the impact of faults on every circuit by calculating and comparing different metrics, such as the mean absolute error (MAE), the mean relative error (MRE), the mean square error (MSE), the mean operations between errors (MOBE), the bit error rate (BER), and the fault activation and propagation rate (FAPR). The results allow for assessing the reliability of each multiplier and its suitability for a given application scenario. We also show that the considered workload significantly impacts the fault severity for all multipliers. As a summary, the combination of fault injection campaigns with application-specific benchmarks proposed in this work plays a vital role in accurately forming a reliability verdict for a circuit design when adopting different evaluation metrics in terms of error analysis, error rate, and fault severity evaluations.

INDEX TERMS Reliability, safety, stuck-at faults, fault injection, arithmetic circuits, Dadda multipliers, artificial intelligence.

The associate editor coordinating the review of this manuscript and approving it for publication was Fabian Khateb.

I. INTRODUCTION

Reliability is a paramount concern in the realm of modern digital systems. As our reliance on digital technology

continues to expand, the accurate and dependable operation of arithmetic circuits has become integral to our daily lives since even minor errors can have profound consequences. Hence, the evaluation of arithmetic circuit reliability is a major concern, especially when dealing with safety-critical systems (e.g., in automotive, healthcare, or aerospace applications [1], [2], [3], [4], [5]).

Arithmetic operations (e.g., multiplication, addition, division) are extensively utilized in different areas such as signal processing, communications, scientific computing, and machine learning (ML). These arithmetic operations are commonly implemented in hardware, leading to fundamental processing units used in a wide spectrum of devices, spanning from CPUs to specialized accelerators like Graphics Processing Units (GPUs) or Tensor Processing Units (TPUs) [6], [7], [8]. Indeed, almost every computational device nowadays incorporates arithmetic circuits (e.g., multipliers and adders) to execute basic operations, complex algorithms, and mathematical functions (e.g., linear algebra operations) [6], [8]. Nowadays, the rise of artificial intelligence (AI) has stimulated the blossoming of specialized hardware accelerators that implement parallel dot-product and matrix-multiplication operations, accelerating convolutions and linear layers in deep neural network (DNN) workloads [7], [8].

Recent advancements in the semiconductor industry, like transistor technology scaling, in combination with chiplet technologies, allow the incorporation of many components and functionalities in a single system-on-chip (SoC). For example, modern AI accelerators can include hundreds to thousands of arithmetic circuits (mainly multipliers and adders) embedded in a single chip. This remarkable parallelism allows different systems to achieve exceptional computational performance (in terms of giga or tera operations per second). Unfortunately, the outstanding performance of modern computational devices can be overshadowed by reliability issues associated with the vulnerabilities of modern semiconductor technologies, that make them increasingly prone to faults [9], [10].

These faults may belong to two categories: permanent and transient faults. this work focuses only on the first category. Permanent faults might exist in a circuit due to several reasons, such as *(i)* undetected manufacturing defects originated during chip production, or *(ii)* sudden damages arising in the device during in-field operations caused by accelerated aging or premature degradation, possibly exacerbated by harsh operational conditions (e.g., high temperatures) [1], [9]. These faults can affect the arithmetic units of a system (e.g., an AI accelerator) that in turn may induce errors at the application level and consequently produce serious application consequences (e.g, mistaken DNN predictions causing wrong decisions in an autonomous system). In fact, recent studies have revealed that faults in cutting-edge scale integration nodes (7nm and beyond) can produce Silent Data Errors (SDEs) that propagate through the application execution, jeopardizing the overall reliability of the systems. Recently, some works ([11], [12], [13]) highlighted the relevant impact of faults affecting devices used in data centers, and not only in safety-critical applications.

In this regard, the reliability evaluation of arithmetic circuits with respect to faults is a paramount aspect mandated by the respective standards (e.g., ISO-26262 for the automotive, DO-254 for avionics, or ISO-14971 for medical devices). In effect, system manufacturers can ensure that their products meet the required levels of reliability and safety by applying methodologies like *failure mode effects and criticality analysis* (FMECA) [14]. This failure analysis provides a systematic methodology for identifying failure modes and their severity consequences in the criticality assessment of individual components in the whole system in the case of failures. This analysis also serves to devise remedies or practices that contribute to increasing or maintaining the overall dependability and accuracy of the system, thereby minimizing the risks associated with any system failures.

In this regard, it is crucial to evaluate the effect of faults in arithmetic circuits to quantify their severity and consequently provide realistic information to perform FMECA analysis. Furthermore, detailed fault evaluations can be used to select the most suitable arithmetic module for a given application, considering aspects like area, speed, power consumption, and reliability. Thus, based on this analysis, it is up to the designer to identify the module version that best suits the application standards.

This work proposes a general fault evaluation flow in combination with a set of evaluation metrics to assess the impact of permanent hardware faults in arithmetic circuits, using data traces from representative DNN workloads.

In particular, our work targets integer multiplier circuits as a study case since these arithmetic modules are crucial units in most modern hardware accelerators. For example, the convolution multiply accumulation (CMAC) core of the NVDLA accelerator incorporates 1,024 fp16/int16 multipliers, representing almost 80% of the total computational units of the accelerator [15], [16]. Although this work uses as test cases some multiplier circuits, it is worth noting that the proposed fault evaluation flow can be adapted to any arithmetic circuit. In order to exemplify the usage of our methodology four different multiplier circuits were selected: Two variants of the 32-bit Dadda multiplier (`Circuit A` and `Circuit B`), and two variants of the 8-bit Booth multiplier (`Circuit C` and `Circuit D`). The first two multiplier circuits correspond to representative circuits commonly used in general-purpose platforms (i.e., GPUs and CPUs) to accelerate different application domains, including machine learning algorithms such as DNNs. The second set of circuits corresponds to multipliers, which are representative circuits used in specialized AI accelerators, such as Tensor Core Units or Tensor Processing Units. It is worth noting that this work significantly extends the method, study cases, and results of a previous publication in the area [17].

Although the proposed approach can be easily extended to transient faults, this work concentrates on the study of permanent hardware faults. The experimental evaluations are based on multiple fault injection campaigns that assess the impact of faults on every circuit. The evaluations considered traces from four different workloads corresponding to three DNN models for image classification tasks (*LeNet5*, *ResNet18*, and *ResNet50*) and a random set of input operands following a uniform distribution. It is important to underline that the DNN traces were used to generate realistic input data required to characterize the selected circuits. The fault injection results were analyzed in order to evaluate the overall reliability of the selected multiplier circuits by calculating and comparing the mean error distance (MED), mean relative error distance (MRED), mean square error (MSE), mean operations between errors (MOBE), bit error rate (BER), fault activation and propagation rate (FAPR). Furthermore, this work proposes the fault severity bins (FSBs) and the fault severity levels (FSLs) as evaluation strategies that permit quantifying and grouping the faults in the circuits according to their severity in terms of error distributions and worst error distance.

In addition to experimentally identifying the more resilient multiplier architecture, the experiments conducted on the selected circuits showed that every multiplier exhibits different error distributions in all evaluated benchmarks. In particular, faults in 32-bit multiplier circuits have significantly larger error distances than for 8-bit multipliers.

The MOBE and BER evaluation metrics show that permanent faults manifest as intermittent error corruptions during the circuit operation. In fact, this work demonstrates that such intermittent effects are particularly associated with the application workload characteristics.

The fault severity analysis allows for the association of portions of faults with different severity levels in terms of MED and WED. This approach also allows the quantification of the number of faults associated with different severity intervals, ranging from safe faults with no impact up to faults that induce large error magnitudes.

Finally, based on the experimental evaluation results, this work suggests two potential software-level error correction solutions that contribute to mitigating the large error magnitudes induced by permanent faults on 32-bit multipliers when executing 8-bit DNN workloads.

Although the proposed fault evaluation flow focuses on arithmetic circuits, it is important to highlight that the evaluation strategy also provides circuit characterization with respect to faults and workloads. Thus, the proposed error rate metrics (e.g., MOBE and BER) can be further exploited to assess the reliability of complex AI accelerators or the DNN workloads using high-level error abstractions. Nonetheless, such further evaluations fall out of the scope of the present work.

In conclusion, the combination of fault injection campaigns with application-specific benchmarks proposed in this work plays a vital role in accurately forming a reliability verdict for a circuit design when adopting different evaluation metrics in terms of error analysis, error rate, and fault severity evaluations. In summary, the main contributions of this work are listed as follows:

- A comprehensive methodology for evaluating the reliability of arithmetic circuits, involving three main phases: *i)* circuit and benchmarks preparation, *ii)* exhaustive fault injection campaigns and *ii)* thorough statistical evaluation to characterize fault effects.
- A collection of evaluation metrics that allow the reliability assessment of different arithmetic circuits in terms of error magnitudes (i.e., MED, MRED, MSE), fault severity (i.e., WED and FAPR), and error rate (i.e., MOBE and BER).
- A detailed reliability evaluation using the proposed strategy on four multiplier circuits (two 32-bit multipliers and two 8-bit multipliers) with respect to permanent faults utilizing data from different DNN benchmarks (ResNet18, ResNet50 and LeNet5).
- A comprehensive set of experimental results reporting the impact of faults in terms of error magnitudes and error rate (e.g., MED, FAPR, MOBE, and BER). The provided results demonstrate that the reliability of every multiplier circuit heavily depends on the application workload, the size of the multiplier circuit (bit precision 32-bit or 8-bit), and its architecture implementation.
- The experimental evaluation of two potential error correction solutions for software-level implementation, aiming to mitigate the effects of faults on 32-bit multipliers circuits in the context of general-purpose devices (e.g., GPUs or CPUs). The results indicate that the proposed approaches enhance the reliability of 32-bit multipliers when executing 8-bit workloads by reducing the MED by about 5 orders of magnitude, increasing the MOBE to more than 10^8 operations, and reducing the BER by approx 0.8 times with respect to uncorrected scenarios.
- An open-source and fully automated environment specially developed to implement the fault injection campaigns on the arithmetic circuits regardless of the hardware description or the technology library.[1]

The rest of the paper is organized as follows. Section II-B presents some relevant works about the evaluation of multipliers circuits in terms of accuracy and error resilience, whereas section II-C provides some insights about the architecture of multipliers circuits. Section III presents a detailed description of the proposed flow to assess the reliability of multiplier circuits through fault injection campaigns. Section IV reports the characteristics of each multiplier variant that has been considered as a case study. Section V reports the experimental results and the evaluation of the impact produced by faults on the evaluated circuits. Section VI proposes certain software-based reliability enhancements based on the observations of our experimental results. Lastly,

[1] github.com/NikosDelijohn/finjenv

Section VII draws some conclusions and provides insights into future research directions.

II. BACKGROUND

This section provides an introduction to hardware faults in modern silicon devices as well as an overview of related works regarding the reliability evaluation of arithmetic circuits, emphasizing the evaluation strategy, the target arithmetic hardware, and possible limitations. In addition, this section presents some insights about the reliability evaluation of multiplier circuits, followed by a comprehensive description of the main architectural details of different integer multipliers.

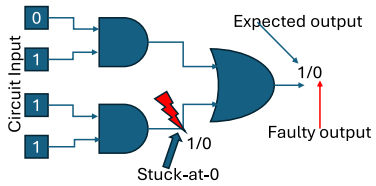

FIGURE 1. Representation of a stuck-at-0 fault affecting a basic combinational circuit.

A. HARDWARE FAULTS AND RELIABILITY

The advancements in CMOS technologies have enabled the creation of smaller, faster, and more energy-efficient electronic devices (e.g., AI accelerators, GPUs, or CPUs) used in a wide spectrum of applications (e.g., AI domains). These advancements mainly involve the incorporation of high transistor densities on a single chip pushing the technology scaling ruled by the Moore's law [1]. However, the continuous miniaturization of these technologies (e.g., 7nm and below) has also led to reliability concerns as they are more susceptible to faults caused by aging, overstress, environmental harshness, or potential manufacturing defects. In fact, technology scaling highly contributes to an increased probability of hardware defect occurrence (e.g., induced by terrestrial radiation effects [18]). Several studies have demonstrated that the failure rate grows as technology scales [1], [9], [19]. This accelerated failure rate of modern silicon technologies can seriously reduce the lifespan of devices and affect the reliability of a vast amount of applications nowadays.

The relationship between the increased failure rate and the technology scaling can be attributed to changes in manufacturing methods used to improve the performance of modern devices as the transistors shrink. This has resulted in the introduction of new materials, transistor architectures, and a shift towards 3D chip design [1]. Unfortunately, all these changes can produce silicon defects corresponding to permanent faults that can appear during the operative life of the device, ultimately leading to serious problems such as system failures [11], [20], [21]. A tangible sign of the occurrence of permanent faults in modern devices has been found in field tests/reports from large-scale server operations from Google [12], [20], Facebook [11], and ORNL [22]. In fact, authors in [12] discovered that CPUs in data centers exhibited wrong computation results that caused the failure of the system or silent data corruption on the application workloads.

For all the above reasons, being able to estimate the impact of permanent faults in current circuits is crucial. Fault simulation is a widely used strategy that permits mimicking the behavior of a physical defect through circuit simulation at different hardware abstraction levels, such as gate or RT levels. The stuck-at-fault model is the most common approach used to describe permanent faults affecting individual signals in a circuit netlist by forcing them to 0 or 1 values. Figure 1 depicts a simple example of an stuck-at-0 fault inside a basic combinational circuit. Fault simulation campaigns of permanent faults are essential for devising testing strategies like Built-in Self-Test (BIST) or Software-Based Self-Test (SBST). Likewise, fault simulations are crucial for evaluating the impact of permanent faults on different digital circuits (e.g., arithmetic circuits) to characterize the fault behaviors or to devise possible fault-tolerance mechanisms during the circuit's design phases.

B. RELATED WORKS

Considerable research effort has been put on the topic of the accurate reliability evaluation of integrated circuits that resulted in the introduction of rigorous mathematical tools [23], [24], [25].

In [26], the authors evaluate the reliability of various majority gates full adder circuits while comparing them against XOR-based full-adders at the gate level. The study reveals that the majority of gate adders demonstrate superior robustness. While the number of gates is a reliability factor, the specific implementation and interconnection of these gates play a crucial role in determining the reliability of the circuit.

In [27], the authors present a comprehensive approach to understanding and predicting errors in arithmetic operations, particularly focusing on addition. They introduce a novel error model to analyze the behavior of a common arithmetic unit, emphasizing the importance of enhancing the reliability of arithmetic operations in computing systems, especially in safety-critical applications where undetected errors can lead to significant issues.

Regarding the general area of arithmetic circuits, in [28], the authors present a novel approach for evaluating the reliability of combinational arithmetic circuits at the transistor level. They develop a framework for calculating output probabilities of basic logic primitives and propose an efficient algorithm for computing the overall circuit reliability. They demonstrate how transistor-level reliability analysis can inform design choices by providing insights into achieving high reliability without significant hardware expansion.

In [29], the authors perform a comprehensive comparative analysis of various approximate multipliers, considering both

error and circuit characteristics and applying these findings to a practical image processing application to assess real-world performance.

Regarding arithmetic circuits based on the approximate computing paradigm, in [30], the authors present a detailed overview of the test and the reliability of such circuits. They discuss the intricate balance between the achieved system reliability and the efficiency gains from using approximate hardware, with a focus on strategies like fault classification, error detection, and selective hardening techniques.

In [31], the authors propose two methods for evaluating the reliability of approximate arithmetic circuits using signal reliability analysis. The methods incorporate subtraction and division correlation coefficients to address the correlation issue caused by fanout reconvergence. These techniques are validated against Monte Carlo simulations. The methods are more efficient and accurate compared to existing reliability analysis approaches. Additionally, the paper provides detailed experimental results showing the methods' scalability and reduced memory overhead compared to prior methods.

In [32], the authors present an efficient method for evaluating the reliability of approximate arithmetic circuits (AACs) under varying input signal probabilities. By addressing the correlation issues caused by reconvergent fan-outs, they develop a new model to estimate the reliability of AACs and propose two heuristic search algorithms to determine the lowest possible reliability, forming a reliability boundary. The method is validated on several AAC designs, including adders and multipliers, and demonstrates higher accuracy and efficiency compared to prior models. The drawbacks, as mentioned in the paper, include higher memory overhead due to the storage of intermediate data like signal probabilities and correlation coefficients.

To the best of our knowledge, this is the first work to propose a reliability evaluation flow in combination with evaluation metrics specifically designed to assess the impact of permanent hardware faults on arithmetic circuits. Our proposed work seeks to provide different metrics for assessing the reliability and contribute to the *failure mode effects and criticality analysis* (FMECA) of these units, which can be incorporated into different computational devices in safety-critical systems. Unlike most existing studies focusing on approximate circuits, our approach quantifies the effects of permanent hardware faults accurately at the gate level of an arithmetic circuit, producing statistical information regarding the fault severity and error rates.

C. MULTIPLIER CIRCUITS

The multiplication operation is ubiquitous in electronic systems nowadays, from simple consumer gadgets to high-end computational systems used for scientific computation. Indeed, multiplication is one of the most critical operations in any arithmetic circuit or accelerator since it significantly impacts the circuit's speed, area, power expenditure, and reliability. These main aspects of multiplier circuits are strictly related to the complex nature of multiplication in hardware, which also implies the extensive number of logic gates required to perform the operation. In fact, multiple strategies are adopted to implement efficient multiplier circuits to accomplish strict design constraints such as performance, power budget limitations, area overhead, and speed. Nonetheless, when considering safety-critical systems, reliability is a paramount aspect that every design, including the multipliers, must accomplish. Therefore, evaluating their vulnerabilities regarding hardware defects is important to provide enough information to the designer when selecting an adequate multiplier architecture for a target application.

In general, a multiplier circuit is composed of two main building blocks: *(i)* a partial product generation block and *(ii)* an array of adders that performs the "sum of products". How each of these blocks is constructed or organized defines the specific architecture that directly impacts the physical and operational characteristics of the final circuit (e.g., area, power, and delay). Figure 2 depicts the basic algorithm of an array multiplication circuit. This architecture comprises a significant amount of half-adders and full-adders that highly impact by the carry propagation delay of the multiplier, especially in the critical path from $z_{0,0}$ to p_{2n-1} [33].

In this regard, there are multiple works in the literature that propose architectural variations that are able to tackle the delay limitations and the area and power features of the final circuit. Nonetheless, none of such works have considered the impact on the reliability of the multiplication operation when considering variations of the architecture in the underlying computational hardware. In this work, we study Two well-known multiplier architectures: Booth, and Dadda multipliers. In the following subsections, we provide further details about each of them.

$$
\begin{array}{llllll}
 & & a_{n-1} & a_{n-2} & \cdots & a_0 \\
 & \times & b_{n-1} & b_{n-2} & \cdots & b_0 \\
\hline
 & & z_{0,n-1} & z_{0,n-2} & \cdots & z_{0,0} \\
 & z_{1,n-1} & z_{1,n-2} & \cdots & z_{1,0} & \\
 & \iddots & \iddots & \iddots & & \\
+ & z_{n-1,n-1}\cdots & z_{n-1,0} & & & \\
\hline
p_{2n-1} & p_{2n-2} & \cdots & & p_1 & p_0
\end{array}
$$

FIGURE 2. Multiplication of two n-bit integer operands (A, B), using the classical multiplication algorithm.

1) WALLACE AND DADDA MULTIPLIER ARCHITECTURES

Wallace [34] and Dadda [35] multipliers are designed to deliver high speed computations. In fact, the Dadda multiplier is a variation of the Wallace multiplier architecture with reduced amount of gates. These multipliers focus their optimization on the delay reduction of the "sum of products" block of the multiplier. Their main idea is to carry out any HA or FA operation in a given bit position as soon as its operands are ready, irrespective of their partial product (row) position.

In fact, these multiplier architectures try to run as many HA or FA operations in parallel as possible. Consequently, they gain in terms of speed. The multiplier operation is split into several phases. For each phase, the ready FA or HA operations are carried out, and thus, the tree depth is reduced as we move from phase i to phase $i+1$ until the number of tree rows is reduced to two rows. These two rows are then passed to a fast n-bit adder (e.g., carry-look-ahead adder) to get the final multiplication result. The detailed algorithms of Wallace or Dadda tree multipliers include several tweaks to ensure that the final tree is reduced uniformly to two rows across as many bit positions as possible.

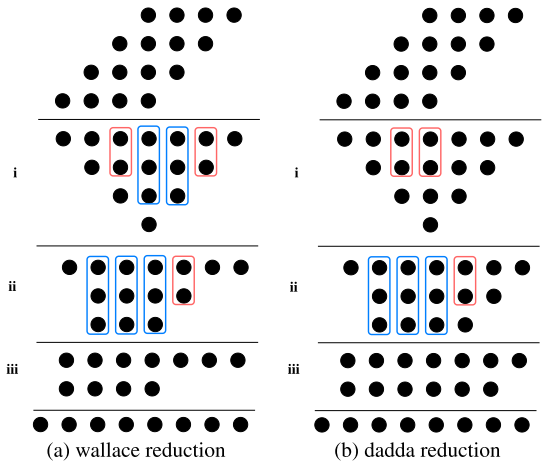

FIGURE 3. Summand reduction using Wallace and Dadda strategies for d=3,2 (see [33], [35]). Dots represent $z_{i,j}$ summands, and red/blue boxes represent HAs/FAs, respectively.

In Figure 3, we illustrate an example of these reduction strategies for 4-bit operand multiplication. The algorithm is carried out in the following phase sequence:

i. The $z_{i,j}$ summands are re-organized into a tree structure and divided into HA/FA groups.
ii. The tree depth is recursively trimmed by the usage of HAs and FAs according to the algorithm until it reaches a depth of 2.
iii. The final 2 rows are passed to a fast n-bit adder to get the final multiplication result.

FAs are alternatively called 3:2 compressors as they compress their three input into two-bit outputs. Similarly, HA are called 2:2 compressors. The research community further dived into the reduction process (step ii), and besides the HAs/FAs, further compressor schemes have been proposed [36], which resulted in even faster and more efficient multiplier circuits.

2) BOOTH MULTIPLIER ARCHITECTURE

The Booth multiplier architecture can perform both unsigned and two's complement signed multiplications with slight variations, without changing the complete circuit architecture. One of the main advantages of the Booth multiplier is that it optimizes the partial product generation stage instead of changing the sum of product reduction as Wallace or Dadda multipliers do. Typically, the Booth multiplier reduces the number of partial products by a factor of two, reducing the number of stages required to perform the sum of products [37].

Figure 4 illustrates the typical architecture of an 8-bit Booth multiplier that uses radix-4 encoding for one of the inputs (i.e., Multiplier). The Booth encoder takes 3-bits from the multiplier input to generate one multiplicative factor $Enc_i = \{0, \pm 1, \pm 2\}$. In total, for an 8-bit multiplier, four Booth encoders generate four partial products. The partial products generation uses a special Booth decoder unit that implements the product between the input M and the encoder Enc_i generating the partial product $PP_i = Enc_i M$.

The "sum of products" takes the partial products and implements an adder tree structure to calculate the final result. It must be noted that partial product accumulation can also be implemented by following the Wallace or Dadda reductions.

We selected four multiplier architectures spanning two important bit widths (8 and 32 bit widths) and two important, yet orthogonal, directions for enhancing the performance of binary multipliers (Dadda and Booth multiplication algorithms). A vast myriad of multiplier architectures is available in the literature, including approaches that combine the Dadda and Booth algorithms to achieve a higher performance level for binary multipliers. Our focus is on exact multipliers only. Also, we avoided prospective approaches that are not ripe yet for industrial use or over complex approaches. Although we used four examples, we believe that we covered the vast multiplier design space in a reasonable and feasible way. Our approach for assessing binary multipliers' reliability can be easily applied to any other multiplier architecture.

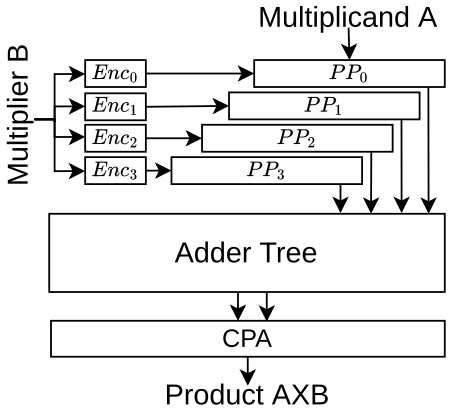

FIGURE 4. Booth multiplier architecture.

III. RELIABILITY EVALUATION OF MULTIPLIERS CIRCUITS

This section describes the proposed procedure to quantitatively assess the impact of permanent (stuck-at) faults on different multiplier circuits. Also, we introduce the evaluation metrics used to estimate the fault criticality of any arithmetic

circuit. The evaluation strategy comprises the following main stages: *(i)* circuit and dataset preparation, *(ii)* fault injection campaign, and *(iii)* data post-processing. In the following sections, we introduce further details about every stage of the evaluation methodology.

A. CIRCUIT AND INPUT DATA PREPARATION

This is a crucial stage in the reliability evaluation of the arithmetic circuits since it uses the real hardware implementation of the target circuit in order to define *(1)* the set of faults to be considered and *(2)* the input stimuli to be used during the reliability evaluation. In an ideal case, the reliability evaluation should consider the evaluation of all possible faults in the circuit in combination with all possible input combinations of operands that could eventually be applied to the target arithmetic circuit. In the first case, some faults inside the circuit are considered functionally equivalent [38], [39], [40], producing the same effects on the circuit. Thus, the redundant faults can be removed or collapsed into a subset of faults to be used during the fault injection campaigns, directly reducing the overall evaluation effort.

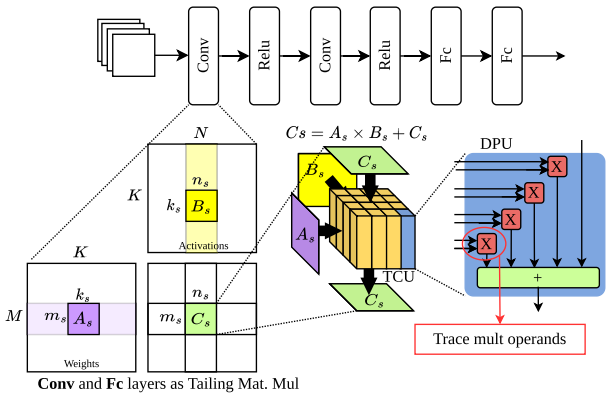

FIGURE 5. CNN implementation using a TCU accelerator of shape $m_S \times n_S \times k_S = 16 \times 16 \times 16$. One of the multipliers of one DPU was profiled in order to get the operand traces during the inference of a neural network.

On the other hand, using exhaustive input stimuli combinations to feed the circuit during the fault injection campaigns does not provide substantial benefits to the reliability evaluations. First, the number of possible operations may result in an unmanageable set of possible operand values. For example, in a 32-bit arithmetic circuit, there are 2^{64} or $\approx 1.8 \times 10^{19}$ possible input operand combinations, which would require an extremely high simulation time even when only considering the circuit without faults. In contrast, when the circuit is included in the context of an application like DNN model computation, then the operands are bounded between certain ranges, limiting the number of combinations but increasing the number of times the hardware executes a particular operation. Therefore, using representative sampling traces from a target application provides better insights into the reliability of the circuit under realistic workloads.

Thus, considering these crucial aspects, we used the following sequence of steps to generate both the list of faults and the set of evaluation stimuli. First, we assume that the target circuits are available as an RTL description in a given Hardware Description Language (HDL) representation. Then, the circuit is synthesized into its gate-level netlist using a target technology library through an EDA tool. In addition, the synthesis stage also provides preliminary information about the circuit, such as the number of cells, area, power, and delay.

After the gate-level synthesis, the circuit netlist is analyzed to identify the list of faults to be used during the fault injection campaigns. In this work, we performed the evaluations using the stuck-at-fault model. However, considering other fault models, the same steps can also be applied. The fault list is generated by resorting to the fault manager of a commercial Automatic Test Pattern Generation EDA tool. The fault manager performs a structural fault analysis of the target circuit and fault-collapsing procedures (to remove equivalent faults), keeping only the prime faults [41]. The fault-collapsing procedure is significant since it produces only the fundamental set of faults, reducing the total number of fault injection campaigns.

Finally, the input stimuli selection for the fault injection experiments is composed of quantized 8-bit integers from a set of convolutional neural networks and a set of data randomly generated as pairs of 8-bit integers. In the first case, the stimuli selections were conducted by extracting traces of a single multiplier from a Tensor Core Unit (TCU) like accelerator [8], [42] while executing a DNN workload. For the purpose of this paper, we employed a single TCU core composed of 256 dot product units (DPUs) arranged in a bi-dimensional shape of 16 × 16. The TCU comprises 4,096 multipliers, that is, 16 multipliers per DPU. In addition, the DNN computations were mapped to the TCU core by transforming convolutional and linear layers into tiling matrix multiplications of size 16 × 16× 16 as depicted in the Figure 5. It is important to highlight that the multiplication traces were obtained using typical 8-bit quantized DNN models while executing the inference of the 20% of the evaluation dataset using the described TCU core.

The data tracing procedure may result in a huge volume of data, ranging from thousands to hundreds of millions of possible operations. However, the parameters of the DNN and the intermediate features of a DNN make the data uniformly distributed around zero. This means that a significant number of multiplications correspond to the same operands (i.e., pairs of weights and activations). In order to reduce the number of stimuli to be applied during the fault evaluations, we can filter the traced data, extracting only the unique pairs of operands rather than considering the number of times they are used during the DNN computation. This subset of operands is then sampled uniformly using only 10,000 unique samples (i.e., operand pairs). This sampling procedure significantly reduces the computational cost required during the fault

injection, which still represents more than 99% of the data used during the computation of a DNN algorithm.

B. FAULT INJECTION CAMPAIGNS

The next step in our evaluation flow is the simulation-based fault injection procedure. This procedure allows us to assess the impact of every fault on the circuit's results when applying a given input stimulus. At this stage, the gate-level description of the circuit under evaluation is wrapped in a testbench (TB) that applies the input stimulus and captures the circuit responses, creating a file used in the post-processing stages. The Fault injection campaign is orchestrated via a commercial simulation tool through simulation commands (i.e., `force`) that control the status of the internal circuit signals according to the selected fault model.

Algorithm 1 Fault Injection Campaigns routine for CUT

```
Results ← ∅
Errors ← ∅
for all stuck-at-fault sa in fault-list do
    FM_responses ← ∅
    GM_responses ← ∅
    inject_fault(sa)
    for all operand pair (a, b) in stimulus source do
        FM_responses = FM_responses ∪ CUT (a, b)
        GM_responses = GM_responses ∪ a × b
    end for
    remove_fault(sa)
end for
statistics = error_analysis(GM_responses , FM_responses )
return statistics
```

Algorithm 1 describes the fault injection procedure used in this study. First, the TB is responsible for importing the fault list and an input stimulus source. Subsequently, the TB generates two instances of the circuit under test, one for the Golden Machine (GM) and one for the Faulty Machine (FM). These two circuits are used to form a miter circuit, which keeps track of the different responses of the two circuits. In an iterative fashion, the FM is injected with one stuck-at-fault at a time. Then, the TB feeds the operand pairs to both the FM and GM models while logging their responses. When all operands have been processed, the FM and GM responses are dumped into an output file that is later used to assess the impact of each injected stuck-at-fault.

C. FAULT INJECTION POST-PROCESSING

Finally, the reports obtained from the fault injections are used to assess the effect of faults in the target circuit and quantify their severity. Inspired by previous studies in the field of approximate circuits [43], [44], [45], we used the Error Distance (ED), the Mean Error Distance (MED), the Mean Relative Error Distance (MRED), the Mean Square Error (MSE), and the Error Rate (ER) as evaluation metrics that grade the impact produced by defective arithmetic circuits.

The Error Distance (ED) describes the arithmetic difference between the results of the FM and GM circuits. Therefore, the ED is calculated as $ED = |R^{GM} - R^{FM}|$, where R^{GM} and R^{FM} represent the result of an operation produced by the GM and FM circuits during the fault injection campaigns, respectively.

The MED describes the average error induced by a fault in the computation results of the evaluated circuit. MED is computed according to Equation (1), where N is the total number of operand pairs in the input stimulus sequence, and ED_i is the error distance between the FM and GM responses for the input operand pair i.

$$\mathrm{MED} = \frac{1}{N} \times \sum_{i=1}^{N} \mathrm{ED}_i \tag{1}$$

The Relative ED (RED) shows the relative difference of a fault-induced error with respect to the fault-free result. The RED is calculated as $RED = \frac{ED}{R^{GM}}$. This metric allows us to assess the severity of the fault considering the magnitude of the expected results, such that the higher the severity of the fault, the higher the RED value. For example, a single fault may produce the same ED for two different operations in the same arithmetic circuit. Nonetheless, the fault will produce more critical effects on the operation that yields a smaller magnitude result. In addition, we can calculate the mean RED per fault according to Equation (2), where N is the total number of operand pairs of the input stimulus source, RED_i the relative error distance between the FM and GM responses for the input operand pair i.

$$\mathrm{MRED} = \frac{1}{N} \times \sum_{i=1}^{N} \mathrm{RED}_i \tag{2}$$

The Mean Squared Error (MSE) is another metric typically used to quantify the accuracy of approximate circuits [43], [44], [45]. However, MSE can be used to quantify the severity of faults affecting a given arithmetic circuit. In that case, a higher MSE magnitude indicates higher fault severity on the evaluated component. The MSE that represents the effects of a fault on a circuit can be defined as follows:

$$\mathrm{MSE} = \frac{1}{N} \times \sum_{i=1}^{N} \mathrm{ED}_i{}^2 \tag{3}$$

Although these metrics are widely used to assess the probability of arithmetic errors in different contexts, it is also important to underline that a circuit may be affected by hundreds or thousands of faults. Hence, each fault may have a different impact on the circuit behavior. This means that some faults can produce tolerable or accepted errors, while others can severely corrupt the circuit's operation. Thus, the severity of faults on the complete circuit cannot be described as a scalar evaluation metric only. In this paper, we propose additional evaluation metrics that can be used to perform an FMECA analysis on the arithmetic circuits.

Following this approach, we propose a *criticality evaluation analysis*, based on statistical quantiles, that allows us to identify each fault's severity or group of faults. Hereafter, we refer to this severity evaluation as *Fault Severity Bins* (FSBs). The FSB considers the MED per fault and groups them into five statistical quartiles defined as follows:

Q_0: The minimum value of the distribution.
Q_1: The value under which the 25% of the data are located.
Q_2: The value under which the 50% of the data are located.
Q_3: The value under which the 75% of the data are located.
Q_4: The maximum value of the distribution.

The FMECA procedure will identify the most critical faults, i.e., those belonging to the highest bins (Q_3 and Q_4). The Q_0 bin reports faults that do not produce any erroneous results, which can thus be considered as *safe faults* according to the definition of some safety standards, such as ISO 26262 [46].

On the other hand, the fault severity analysis can be complemented by considering the worst error distance produced by a given fault on the circuit's behavior. In fact, faults do not necessarily produce errors in the complete dynamic range of the circuit outputs. Instead, the errors produced by a fault might be associated with its location inside the circuit, which also determines how the fault propagates through the circuit structure, affecting only portions of the final result.

In order to study such fault effects, we define the Worst Error Distance (WED) of a fault as:

$$\text{WED} = \max_{i\in[1,N]} \{\text{ED}_i\}$$

where N is the total number of operand pairs of the input stimulus set, and ED_i is the error distance between the FM and GM responses for the computed operation i. This metric can be used to assign faults with similar error effects to a common severity group. We define as many severity levels β_n as the number of bits Nbit in the output port of the arithmetic circuit under analysis. The severity levels are mutually exclusive, which means that a fault assigned to a group can not be part of another severity level. Equation (4) presents the definition of a severity level β_n in terms of the bit n and WED. It is worth noting that the β_0 represents the lowest and $\beta_{N\text{bit}-1}$ the highest severity levels, respectively. This permits associating the maximum possible error induced by a fault with a single bit flip observed at the output result. For example, if a fault induces a $WED = 2,000$, the fault is assigned to the severity level $\beta_{11} = 2^{10} < 2,000 \leq 2^{11}$ which approximates the worst error as a bit-flip in the 11-th bit of the output result.

$$\beta_n = 2^{n-1} < \text{WED} \leq 2^n \tag{4}$$

Lastly, complementary evaluation metrics are used to evaluate the impact of the faults in terms of the error rate. First, we define the Fault Activation and Propagation Rate (FAPR) as a scalar metric that represents the fraction of faults in the evaluated circuit that generate at least one corrupted result at the circuit's output. Equation (5) describes the FAPR calculation, where K is the total number of faults in the circuit, WED_i is the worst error distance for the i-th fault, and $\delta(WED_i \neq 0)$ is the indicator function, that is equal to 1 when $WED_i \neq 0$ and 0 when $WED_i = 0$.

$$\text{FAPR} = \frac{1}{K} \times \sum_{i=1}^{K} \delta(\text{WED}_i \neq 0) \tag{5}$$

We also define the Mean Operations Between Errors (MOBEs) as an evaluation metric that indicates the number of correct operations produced by the faulty circuit before one corrupted result appears. It is important to mention that MOBEs consider the number of times every operand is used during the whole application execution. MOBE can be mathematically described by the Equation (6), where M represents the length of the tracing report; ED_{ij} corresponds to the error distance induced by the i-th fault on the j-th multiplication (from the fault injection results); W_j corresponds to the number of times the j-th operand pair is used during the CNN computation, and $\delta(ED_{ij} \neq 0)$ is the indicator function that is equal to 1 when $ED_{ij} \neq 0$ and 0 when $ED_{ij} = 0$.

$$\text{MOBE} = \frac{K \cdot M}{\sum_{i=1}^{K} \sum_{j=1}^{N} W_j \cdot \delta(\text{ED}_{ij} \neq 0)} \tag{6}$$

Similarly, the bit error rate (BER) measures the fraction of bits that are corrupted when considering the complete workload execution in the presence of faults. Equation (7) illustrates the BER calculations from the fault injection results. In the equation, $d(R_{ij}^{\text{GM}}, R_{ij}^{\text{FM}})$ represents the Hamming distance between the GM and FM responses for the i-th fault in the j-th operation, whereas Nbit represents the number of bits of the output port of the evaluated circuit.

$$\text{BER} = \frac{\sum_{i=1}^{K} \sum_{j=1}^{N} d(R_{ij}^{\text{GM}}, R_{ij}^{\text{FM}}) \cdot W_j}{K \cdot M \cdot N\text{bit}} \tag{7}$$

IV. EXPERIMENTAL SETUP

The first two circuits correspond to different implementations of a signed 32-bit Dadda multiplier. Namely, *Circuit A* is a variant that uses 7 to 3 compressors and a carry look-ahead adder for the final addition, whereas *Circuit B* is a variant that uses 3 to 2 compressors and a Han-Carlson adder, respectively [36]. The RT-level description of these circuits has been developed by members of the Aoki laboratory at Tohoku University.

The other two multipliers correspond to two variants of an 8-bit Booth multiplier architecture. The *Circuit C* implements the classical Booth multiplier circuit [47], whereas *Circuit D* is a multiplier taken from the publicly available NVDLA accelerator [48]. Moreover, the NVDLA multiplier is a mixed precision architecture that can be configured to operate as one 16-bit multiplier or two 8-bit multipliers. It is worth noting that the NVDLA multiplier does not include hardware to perform the last sum of the final result (i.e., the CPA adder). This is because the multiplier belongs to a DPU unit that merges the sum of several multipliers in a common adder tree unit. As the evaluation of full DPU units is out of the scope of this work, we implemented the functional behavior of the last CPA adder in the test-bench to calculate the final multiplication value.

All circuits were synthesized using the OCL Nangate 45nm technology library [49] via Synopsys Design Compiler. Table 1 shows the details of the logic synthesis for every

TABLE 1. Circuits' details.

Circuit	Area	Cells			# Stuck-At	
		Comb.	Seq.	Buf./Inv.	Collapsed	Total Faults
A	4,912.22	3,299	0	708	12,456	30,226
B	5,721.92	2,729	0	250	10,512	33,464
C	1,060.54	1,202	2	267	2,920	6,030
D	1,569.67	947	0	128	5,743	9,594

circuit. In detail, we report the area in terms of the number of cells (combinational and sequential) and I/O buffers. Interestingly, `Circuit A` reports less cell area than `Circuit B` while having a higher number of cells, which naturally results in more stuck-at faults. On the other hand, the NVDLA multiplier (`Circuit D`) is only 45% larger than the basic Booth multiplier (`Circuit C`), even though it implements the functionality of two 8-bit multipliers in parallel. Nevertheless, `Circuit D` has almost double the amount of permanent faults with respect to `Circuit C`.

For the input stimuli selection, we used three CNN models for image classification applications: LeNet5, ResNet18, and ResNet50. All models used 8-bit Post Train Quantization using PyTorch [50]. The LeNet5 model performs image classification among ten different classes from the MNIST dataset, whereas both ResNet models classify images in the CIFAR10 dataset. We extracted the multiplication traces from one of the multipliers in a 16 × 16× 16 TCU accelerator while executing the inference of all CNNs using the 20% of the validation dataset (i.e., 20,000 images from MNIST and 2,000 images for CIFAR10).

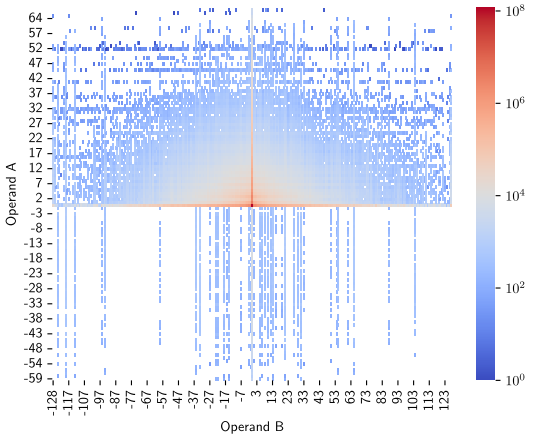

FIGURE 6. Distribution of operands for ResNet50 multiplications. OperandA corresponds to the neuron activation values, and OperandB corresponds to the values of ResNet50 weights. The operands combinations around zero in the range [−32, 32] are the most frequently used, exceeding in more than 10^6 other operands combinations.

The length of the traces significantly increases as the DNN models grow in size (i.e., in terms of the number of layers). Thus, for the LeNet5 model, we obtained 5×10^6 different multiplications operations, whereas, for the ResNet18 and resNet50 models, the number of traced multiplications reached 104×10^6 and 2×10^8 multiplications, respectively. As mentioned in the previous section, the CNN computations are typically concentrated into a small set of unique operand combinations. Figure 6 reports the traced distribution of operands in the ResNet50 model. At first, we observed that the `OperandB` covers the complete dynamic range of the signed int 8-bit representation. This is because this multiplier input operand is always fed with the weight values of the neural network. This is explained by the per-channel symmetric quantization employed on the neural network benchmarks used in this paper [50], [51], [52]. On the other hand, the dynamic range of the `Operand A` is restricted to the range $[-64, 64]$, describing the intermediate feature maps of the neural network traced during the inference stage. It is worth noting that most of the data of the `Operan A` is distributed mainly in the range $[0, 48]$. In addition, we observed that in neural network computations, most operations are concentrated around zero for both operands. In fact, the most recurrently used operand combinations (in more than 10^6 times) lay in the ranges $[-32, 32]$ for `operand B` and $[0, 32]$ for `operand A`. This similar data distribution pattern was found for the other neural network benchmarks used in this paper.

In order to soften the complexity of the fault injection campaigns, we selected the 10,000 unique most frequent operand pair combinations as the input stimulus during the fault injection campaigns. In addition, we generated a supplementary set of evaluation stimuli composed of 10,000 operand pairs randomly generated according to a uniform distribution. This last set of stimuli is used to compare the evaluation differences between data taken from realistic workloads and evaluations based on artificial data generation for this kind of evaluation.

The fault injection procedure described in the previous section was implemented in `finjenv`, an automated fault injection environment we developed for quantitative reliability assessment of arithmetic circuits [53]. The environment used for performing the fault injection campaigns is based on QuestaSIM by Siemens EDA. A fault injection campaign was conducted for every target circuit using the selected stimulus from the CNNs, injecting one stuck-at-fault at a time. Subsequently, a complete post-processing analysis allowed to quantify the degree of severity of each fault for every multiplier under different CNN application workloads.

All of our experiments were performed in a system using 2 Intel(R) Xeon(R) Gold 6238R processors with 256 GB of RAM. The tracing procedure for all CNNs required approximately 12 hours. The fault injection campaigns required about 48 hours for all the circuits and sets of selected stimuli.

V. EXPERIMENTAL RESULTS

This section reports the experimental and quantitative evaluation of the impact of faults on four different multiplier circuits when considering three representative DNN workloads.

A. FAULT-INDUCED ERROR ANALYSIS

The first set of analyses evaluates the impact of fault in terms of arithmetic errors affecting the result of every multiplier circuit.

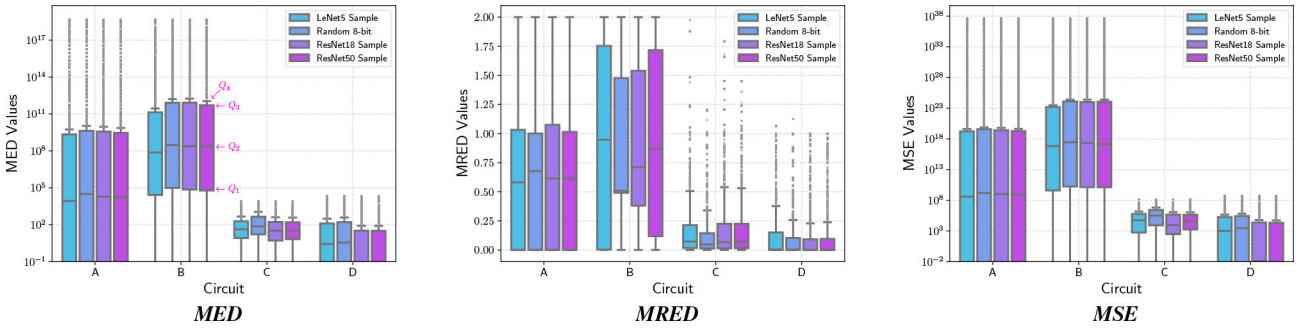

FIGURE 7. Distribution of errors induced by faults on every multiplier circuit.

Figure 7 illustrates the distribution of MED, MRED, and MSE through their quartiles in the form of a box-and-whiskers plot for each circuit and application workload data [54]. The upper and bottom edges of the boxes represent the Q_3 and Q_1 quartiles, respectively. The line in the center of the box represents the median or Q_2. and the upper and bottom whiskers represent the maximum (Q_4) and the minimum (Q_0) without outliers.

In general, the results demonstrate that 32-bit multipliers are prone to generate high error magnitudes both in terms of MED and MSE, whereas the 8-bit multipliers induce errors whose size is several orders of magnitudes lower than their 32-bit counterparts. The reason for this noticeable difference is associated with the number of bits used by every multiplier, such that the higher the number of bits at the output of the multiplier, the higher the error magnitude possibly induced by any fault in the circuit. Consequently, a fault on a 32-bit multiplier, commonly present in general-purpose platforms (e.g., GPU), potentially induces a dominant error magnitude that overruns application execution during the computation of 8-bit workloads, making the effects of faults more critical -in terms of error magnitudes- than using dedicated 8-bit hardware only.

Going into detail, the results show that faults in both of the 32-bit multipliers can induce zero as minimum MED and 10^{19} as maximum MED regardless of the application data. However, when looking at the Q_1, Q_2, and Q_3 quartiles of the error distributions, we observe that faults affecting the `Circuit B` induce about 10^4 higher MED when compared with faults affecting the `Circuit A`. Interestingly, in the case of the `Circuit A`, 25% of the operations (Q_1) were not affected by faults, which led the data under Q_1 almost the same as the minimum MED (Q_0). It must be noted that for both multipliers, the ResNet models and the Random 8-bit data generate one order of magnitude higher MED than the LeNet5 CNN.

When it comes to the 8-bit multiplier circuits, the faults can eventually generate, on average, up to $\approx 10^4$ in terms of MED magnitudes in both of the cases. In the case of `Circuit C`, the faults produce errors almost two orders of magnitude larger than in the case of faults in `Circuit D`. Furthermore, the results also show that different applications in the same multiplier can be affected differently. For example, the faults on the `Circuit C` generate similar MED effects when using data from CNN workloads; on the contrary, when using random 8-bit data, faults on the same circuit induce an almost one order of magnitude higher MED. This indicates that using stimuli artificially generated for this multiplier leads to pessimistic results in comparison with real CNN workloads.

On the other hand, faults on the `Circuit D` produce noticeable differences when performing multiplications on data coming from different application sources. In particular, the results indicate that 50% of the operations of LeNet5 and Random 8-bit benchmarks are corrupted by faults that induce up to $\approx 10^1$ MED. In contrast, the same percentage of operations in the ResNet models exhibit a MED lower than 1. Thus, when using the MED as an evaluation metric of faults in multiplier circuits, the results show that different hardware implementations cause faults to produce significant and myriad error magnitudes regardless of the application benchmarks. In addition, the results seem to indicate that such error magnitudes generated by faults are strictly associated with the final implementation of the multiplier circuit, and in some cases, the hardware implementation also impacts the applications differently.

Further evaluations regarding the MRED allow for assessing the fault severity in relation to the expected output results. In this case, the higher the MRED, the higher the impact of the fault. In general, the results show that 32-bit multipliers have higher MRED, which also means that faults affecting these circuits induce error distances way far from the expected output results. On the other hand, while the `Circuit A` exhibits an $MRED \leq 1$ for $\approx$75% of the operations in all of the evaluated benchmarks, the `Circuit B` generates up to 1.75 MRED for the same amount of data for the LeNet and ResNet50 benchmarks and up to 1.5 MRED for the other benchmarks. These results indicate that the faults in Circuit B are way more aggressive than in Circuit A and generate errors that are up to 150% larger than the expected ground truth values. The same evaluation for the 8-bit multipliers shows that for both circuits, the MRAD does not exceed 0.25 for the 75% of the data for all benchmarks, even though there are some rare cases in which the errors are significantly large (i.e., $MRED > 0.5$). It is important to point out that the impact

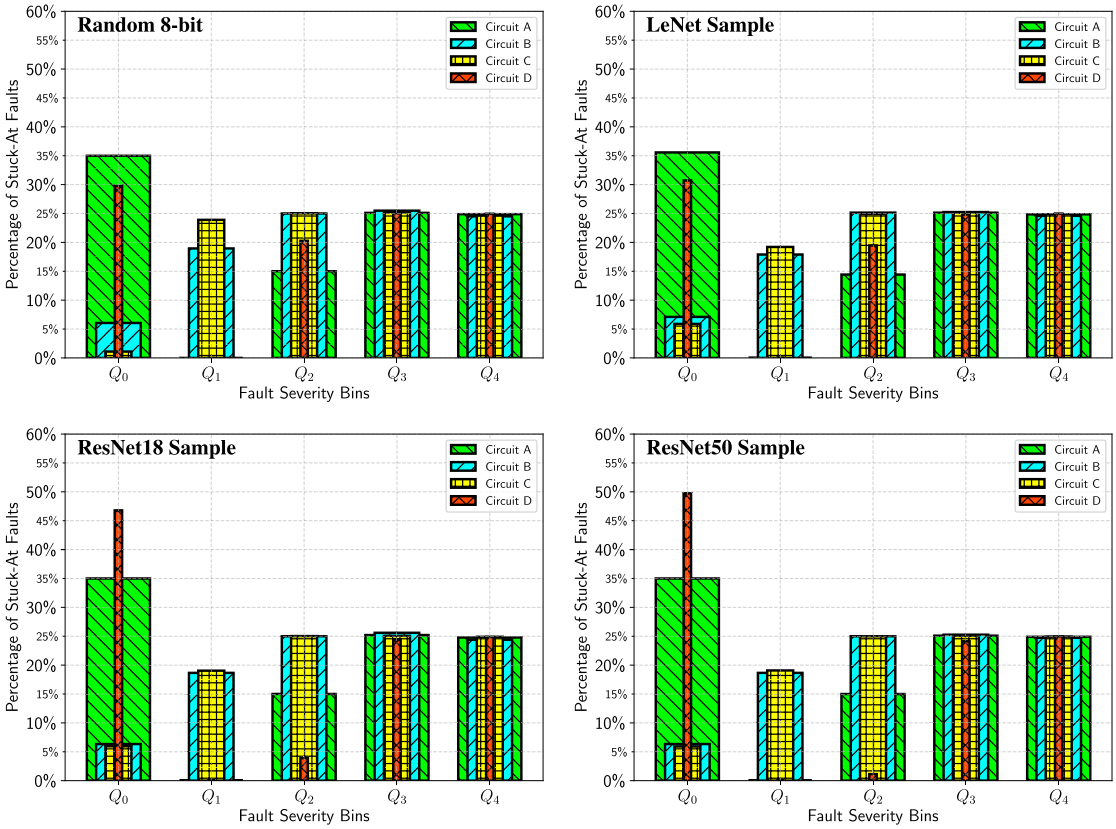

FIGURE 8. Fault severity bins of mean error distance (MED).

of faults on every benchmark changes when considering different multiplier circuits and different benchmarks. For example, the CNN benchmarks in Circuit C have similar error effects. However, in the case of Circuit D, the LeNet5 benchmark has a higher MRED than the other evaluated benchmarks.

These error evaluations provide a general view of the fault effects on different multiplier circuits while considering various representative CNN benchmarks. In these cases, the results indicate that, in general, using 32-bit multipliers to perform 8-bit computations will result in higher errors when permanent faults affect these circuits in comparison with faults affecting 8-bit hardware computation. Furthermore, the results show that the error magnitudes induced by faults on every multiplier may affect every application in different manners. Consequently, the fault injection campaigns described in this work are crucial to characterize the fault effects on the considered circuits correctly.

B. FAULT SEVERITY EVALUATION

In addition to scalar (i.e., based on single value metrics) evaluations of the fault effects on the studied circuits, it is crucial to discriminate the faults according to their error severity for all evaluated benchmarks and circuits. In this sub-section, we present the evaluation results considering two different strategies: *(i)* Fault Severity Bin (FSB) evaluation in terms of the MED, and *(ii)* Fault Severity Levels (FSL) in terms of the WED. The first evaluation allows us to compare all the circuits independently on their precision (i.e., 32-bits or 8-bits) by focusing the analysis on the percentage of faults associated with the quartiles of the error distributions. The second evaluation approach ranks the faults in the circuit according to several fine-grain FSLs regarding the WED.

Figure 8 presents the results of the fault injection campaigns in terms of the *Fault Severity Bins* (FSBs). We report an individual plot for each evaluated benchmark. For all experiments, the Q_0 bin contains the stuck-at faults which do not impact the outcome of the multiplication (i.e., the safe faults). Each other bin contains the faults for which the MED value belongs to the interval $(Q_{i-1}, Q_i]$. The reported results indicate that the faults may behave differently in terms of the multiplier circuit and the evaluated benchmark. The percentage of safe faults for both 32-bit multipliers remains constant regardless of the application benchmark, where 35% of the faults in Circuit A and 6% of the faults in Circuit B do not produce any observable effect at the output of the circuit. Nonetheless, this characteristic does not apply in the case of the 8-bit multipliers, where the safe faults change according to the multiplier circuit and the benchmark. For example, the safe faults in Circuit C ($\approx$5%) are consistent for all the CNN benchmarks, but in the case of the random uniform benchmark, the number of safe faults is significantly

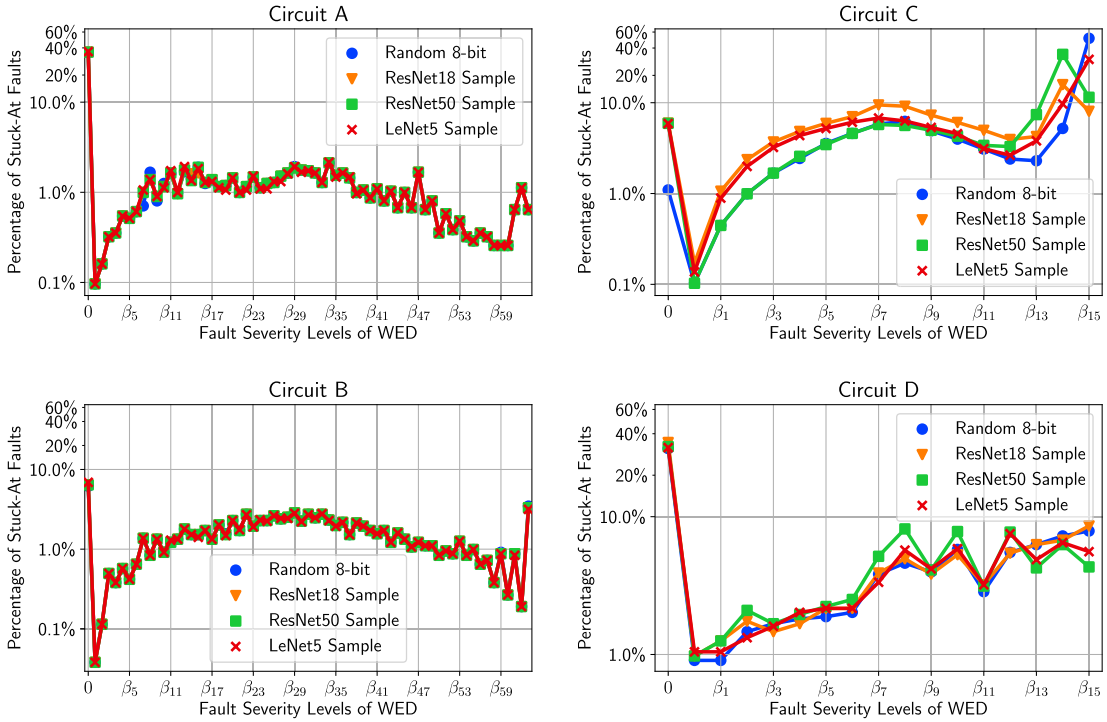

FIGURE 9. Percentage of faults in the multiplier generating a Worst Error Distance (WED) bounded in the range $\beta_n = 2^{n-1} < WED \leq 2^n$. The results indicate that around 50% of the faults, regardless of the circuit, always generate a WED greater than 2^m, where m represents the number of bits of the input operands.

lower, dropping to $< 1\%$. In contrast, the portion of safe faults in Circuit D changes among the evaluated benchmarks. For example, about 30% of the faults in circuit D are considered safe faults for the Random Uniform and LeNet5 benchmarks. However, such percentage increases to 46% and 50% for the ResNet benchmarks.

Interestingly, 50% of the faults in all multipliers and all evaluated benchmarks always lead to highly severe error effects (i.e., Q_3, and Q_4). This portion of faults is equally distributed into the Q_3 and Q_4 bins, where 25% of the faults generate up to 75% of the highest MED at the output of the circuits, whereas the other 25% of faults induce the maximum error in all circuits for all benchmarks. Notably, less than 1% of the faults in Circuit A and Circuit D generate effects associated with Q_1 across all the evaluated benchmarks. In contrast, almost 20% of the faults in circuits B and C lay in the Q_1 severity bins for the CNN benchmarks; however, in the case of circuit C, the number of faults increases to 25% when evaluating the Random 8-bit benchmark. On the other hand, in the case of the severity bin Q_2, the percentage of faults in circuits A (15%), B (25%), and C (25%) remain the same regardless of the evaluated benchmarks. Nonetheless, for the same severity bin, the percentage of faults in Circuit D varies significantly across the benchmarks. For example, when evaluating the Random 8-bit and LeNet5 benchmarks, 20% of the faults in the circuit D generate errors associated with bin Q_2, but in the case of the ResNet benchmarks, the percentage of faults drops to less than 5% for the same severity bin Q_2.

In general, the *FSB* results indicate that for all the multipliers and benchmarks, 50% of the faults are potentially critical since they produce the most severe MED, and are thus grouped in the Q_3 and Q_4 bins. Nonetheless, the other 50% of faults can produce diverse type of effects that clearly depend on both the hardware implementation and the evaluation benchmark.

We performed a fine-grain evaluation analysis considering FSLs regarding the WED of faults on the evaluated circuits and benchmarks. Figure 9 reports the percentage of faults on every circuit that lay on each FSL according to their WED on every evaluated benchmark. We report an individual plot for each evaluated circuit. For each benchmark, the sum of the faults in all FSLs corresponds to 100% of the total faults on every circuit. The label 0 in the horizontal axis indicates that the faults in that category do not produce any error; in other words, they are considered safe faults. From the plots, we observe that the fault severity in the case of 32-bit multipliers is almost the same across all the evaluated benchmarks. In fact, the percentage of faults in both Circuit A and B are almost uniformly distributed among the FSLs from β_8 to β_{47}. On average, 1% of the faults belong to each FSL. However, while the percentage of faults belonging to the higher FSLs in Circuit A decreases down to 0.5%, in the case of Circuit B, it always remains higher than 0.5%, and for the

most severe group, β_{63} the percentage of faults reaches almost 5% for this last circuit. Additionally, the percentage of faults is distributed in the same manner for both multipliers in the low-FSLs (i.e., β_0 to β_7).

We observe from all plots in Figure 9 that the relationship between WED severity levels and the percentage of stuck-at faults does not follow a monotonic trend. This can be explained by the fact that most stuck-at faults do not necessarily induce higher ED values. Instead, the distribution of stuck-at fault percentages is architecture-specific and influenced by the executed workload.

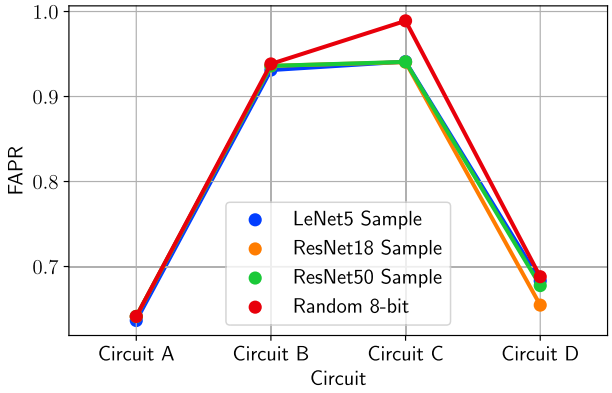

FIGURE 10. Fault Activation and Propagation Rate (FAPR). This plot represents the ratio between the number of faults in the multipliers corrupting at least one operation divided by the total number of faults in the circuit.

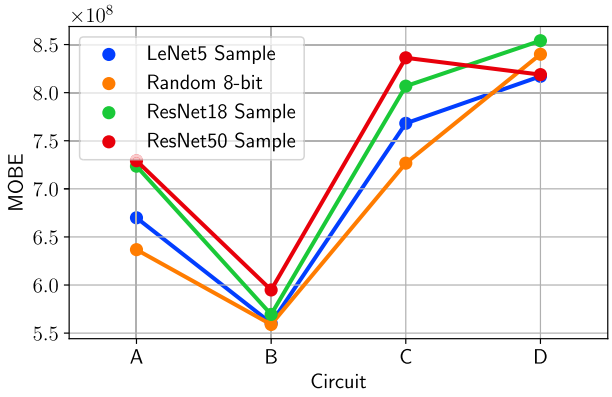

FIGURE 11. Mean Operations Between Errors (MOBE) in 10^9 multiplications.

On the other hand, when using 8-bit multipliers, we observed that the percentage of faults contributing to the different severity levels changes across the evaluation benchmarks, as well as between multipliers. For example, in the case of Circuit C, the percentage of faults belonging to the FSLs under β_7 is significantly lower for ResNet50 and Random 8-bit than for the other evaluated benchmarks. This indicates that when using Circuit C to compute ResNet18 and LeNet5, more faults generate WED below 2^7 than in the case of ResNet50 and Random 8-bit benchmarks. For the same circuit, the ResNet18 benchmark shows a significantly higher percentage of faults inducing WED in all the FSLs below β_{12} than all the other benchmarks. In addition, more than 10% of the faults seriously impact all the CNN benchmarks by inducing WED belonging to the β_{14}, which means that such a number of faults can generate WED about $\pm 2^{14}$.

Finally, Circuit D shows that the higher the FSL, the higher the percentage of faults per FSL. More in detail, the percentage of faults in the lowest FSLs linearly grows from 1% of faults in the FSL β_1 to 5% of the faults laying in the β_6 FSL. On the other hand, the percentage of faults along the FSLs above β_7 oscillates between 5% to 9% of the faults per FSL. Interestingly, the application benchmarks seem to impact how the fault generates different WED according the workload data. For example, when using ResNet50, the percentage of faults laying on some FSL (i.e., β_2, β_7, β_8, and β_{10}) is particularly higher than the other evaluated benchmarks. In contrast, all the other benchmarks exhibit similar behavior across all FSLs.

C. ERROR RATE RELATED METRICS

Figure 10 shows the fault activation and propagation rate for all evaluated circuits and benchmarks. The results indicate that all evaluated benchmarks activate and propagate the effects of about 63% of the faults in Circuit A. On the contrary, the same benchmarks can activate and propagate 94% of the faults in circuit B. Therefore, these results show that Circuit B is more likely to be affected by faults in the context of CNN benchmarks than Circuit A. On the other hand, every benchmark has different capabilities of activating and propagating the effects of faults when using 8-bit multipliers. In general, all the benchmarks executed on Circuit C can activate and propagate 30% more faults than Circuit D. In addition, the CNN benchmarks have a lower fault activation rate than the random 8-bit benchmark.

Figure 11 introduces the mean operations between errors for all evaluated circuits and benchmarks. The plot shows the MOBE when considering a workload execution of 10^9 multiplications. Interestingly, the results show that for most of the multipliers, large CNN workloads execute, on average, more correct operations before one operation is corrupted by faults on the circuits. The reasons behind these results are connected with the fact that in the CNNs benchmarks, pairs of operands are recurrently used during the workload computations. Thus, when such operands are safely computed by the multipliers, even in the presence of faults, the application is more resilient to such faults. Nonetheless, it is important to underline that this is a characteristic of CNN computations, but it might be different for other workloads, as in the case of the random 8-bit benchmark, which always exhibits lower MOBE.

In terms of multiplier circuits, the MOBE indicates that Circuit B is the least reliable of all evaluated circuits, while Circuit D seems to be the most reliable, with more correct operations computed before an operation is corrupted. The main reason Circuit D performs a higher amount of operation correctly than the others is due to the reconfigurable

architecture that allows it to compute in parallel two 8-bit multiplications. Therefore, a fault affecting one side of the circuit can only corrupt half of the operations; only faults in the configuration flip-flops and in the multiplexers that define the operative mode of the multiplier can corrupt both of the 8-bit multiplications. It is worth noting that the MOBE demonstrates that despite the fact that a single permanent fault can persist during the whole operational workload, it does not necessarily corrupt all the operations performed in the circuit. This means that a permanent fault in the multiplier circuit can be modeled as an intermittent error at the circuit output. Nonetheless, it is crucial to keep in mind that such intermittent effects of faults strictly depend on the hardware implementation of the arithmetic circuit and the specific workload under analysis.

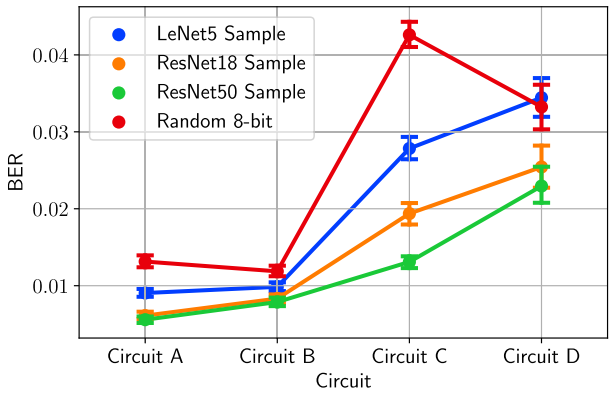

FIGURE 12. Bit Error Rates produced by faults on every multiplier circuit considering different Workloads.

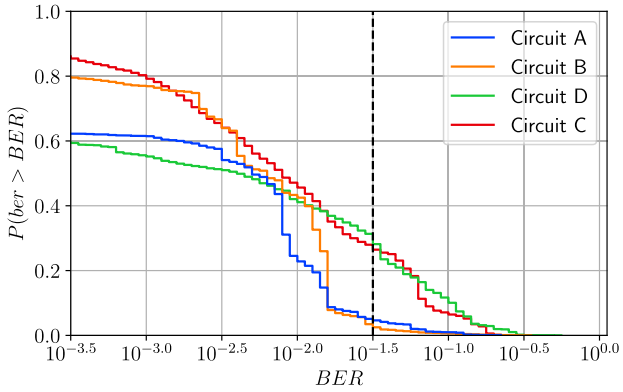

FIGURE 13. Empirical cumulative distribution functions of BER produced by faults on every multiplier circuit.

Finally, we analyze the results of the fault injection campaigns in terms of the bit error rate (BER). This is a well-known metric typically used to evaluate fault effects at the application level, describing generally transient phenomena. However, in this work, we demonstrate that it can also be applied to describe the intermittent effects of permanent faults on the multiplier circuits. Figure 12 introduces the bit error rate as a result of permanent fault effects on the evaluated circuits for all application benchmarks. A higher BER in the plot indicates a higher impact of faults in the circuit. As in the case of the MOBE evaluations, the BER significantly depends on the workload and the number of operations it performs. In this case, the large benchmarks we evaluated (i.e., ResNet CNNs) have the lowest BER in comparison with LeNet5 and the random 8-bit benchmarks for all the circuits. In addition, Figure 13 reports the survival function obtained by calculating the complementary empirical distribution function (CEDF) of the observed BER for all faults on each multiplier circuit. It is worth noting that the CEDF is a common function used in reliability engineering evaluations [55]. In the context of our evaluation, the survival function indicates the probability that a fault on a given multiplier is capable of inducing a bit error rate higher than a given BER.

The results indicate that, in general, faults in the multiplier circuits have a decreasing probability of generating high BERs. Nonetheless, in the case of the 32-bit multipliers, such probability decreases significantly more than in the case of 8-bit multipliers for cases in which the $BER > 10^{-2}$. For example, the vertical black line shows that a fault affecting Circuit D or C has around 30% probability of producing BER higher than $10^{-1.5}$, whereas for the same scenario, for circuits A and B, the probability drops down under 10%.

It is important to underline that a high BER probability does not necessarily imply that the circuit is less reliable or always induces high critical results. In fact, when looking at the FAPR and MOBE, Circuit D is the one that always performs better. This means Circuit D has fewer faults that could potentially produce an error. In addition, the MOBE indicates that Circuit D provides significantly the highest amount of correct operations before an erroneous result appears. Consequently, a high probability reported by the survival function regarding high BER values indicates that when a fault corrupts a given operation result, there is a high probability that the fault effect is observed in more than one bit per operation. In addition, the fault severity evaluation in terms of MED and WED revealed that the percentage of fault-inducing high error magnitude is lower than in the case of other circuits.

In conclusion, the reliability evaluation of an arithmetic circuit must include several evaluation viewpoints in order to rank the potential criticality stemming from the inclusion of the circuit in a safety-critical system. Furthermore, The results point out that it is crucial to consider the workload characteristics in order to gather realistic effects of faults affecting arithmetic circuits. In fact, it is not enough to rely only on random benchmarks without including information about the frequency at which every operation is executed along the time. On the other hand, the BER provides a general idea about the impact of faults on a given circuit under certain workload execution. However, BER has to be combined with other metrics such as FAPR, MOBE, and fault severity assessments to determine a more realistic reliability assessment of the evaluated circuits.

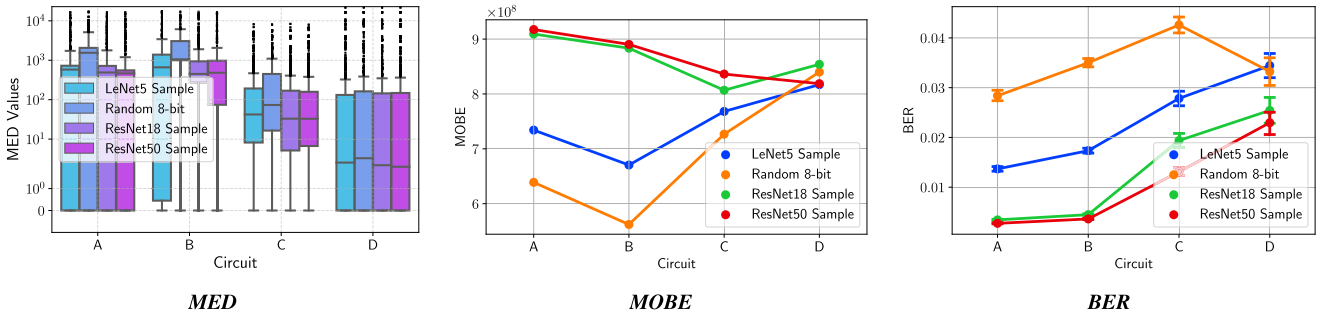

FIGURE 14. Reliability evaluation after applying the Fault-Aware Pruning (FAP) as error correction on the evaluated circuits. The results show reliability improvements in terms of MED, MOBE, and BER when compared with uncorrected results reported in Figure 7, Figure 11 and Figure 12.

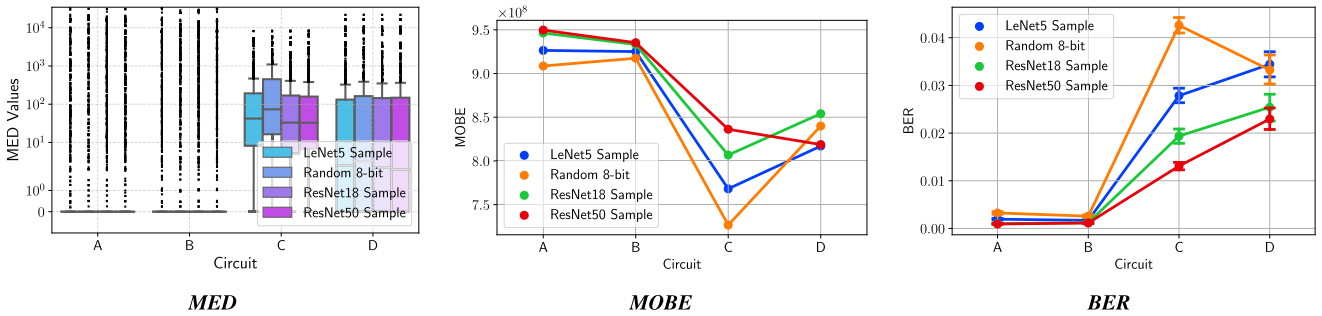

FIGURE 15. Reliability evaluation after applying the sign extension from 16 to 64/32 bits as error correction on the evaluated circuits. The results show noteworthy reliability improvements in terms of MED, MOBE, and BER when compared with uncorrected results reported in Figure 7, Figure 11 and Figure 12.

Hence, our proposed fault evaluation flow, together with the proposed metrics, can be used to select the best multiplier in terms of reliability. Thus, when considering several arithmetic circuits affected by faults, the ones with higher MOBE values and lowest BER can be considered more reliable than the others. On the other hand, the fault severity metrics (e.g., Worst Error Distance, or WED) seek to classify the number of faults according to the error magnitude observed at the output of the circuit. In this case, the circuit with the lowest number of faults inducing high error magnitude can be considered more reliable than others. Hence, these guidelines can help the designer to select a given arithmetic circuit to enhance the reliability of the system. Consequently, when we applied our evaluation methodology to four multipliers, we obtained that for 32-bit multipliers, CircuitA performed better than CircuitB, and for 8-bit multipliers, CircuitD performed better than CircuitC.

Moreover, the proposed evaluation strategy can be used to identify the hardware structures that induce the highest fault severity on the performance of the multipliers. This can be exploited for implementing hardware-based selective hardening on such identified hardware structures only or to implement self-checking and self-repair mechanisms that inform any hardware malfunction, preventing the occurrence of a catastrophic result. Alternatively, the proposed fault evaluation flow can be used to identify the fault effects that can be most harmful to an AI application. For example, when a fault induces an operation result that falls out of the bounds of the maximum expected computation outcome of the application, the designer can include additional (HW/SW) mechanisms to detect and mitigate such fault effects. In the following section we study two potential software-based error mitigation strategies derived from the experimental evaluation on the target circuits. In this regard, our proposed evaluation flow is a crucial step in designing and selecting reliable AI hardware for real-world applications.

VI. RELIABILITY ENHANCEMENTS

Given the reliability evaluation results, we observed that the 32-bit multipliers used to execute 8-bit workloads produce significantly larger error magnitudes under the presence of some faults. Most of these fault-induced errors exceed the maximum error margin of an 8×8 bit multiplication result for our 8-bit workload (i.e., DNN workloads). In addition, general-purpose computational devices (e.g., GPUs) include 32-bit hardware units that can be used to perform highly computationally intensive applications such as DNNs. In such cases, the 32-bit multiplier processing 8-bit workloads produces a 64-bit output result. This result contains the actual multiplication values in the 16-bit LSB, so the MSB contains the sign-bit extension. Consequently, any fault inducing an error in the most significant bits (further than the 16 LSB) typically creates a large magnitude of errors. In these cases, we can leverage the extra bits of the hardware units to detect

and counteract the fault effects treating the multiplier circuits. We propose two main solutions that can be implemented at the software level: *i)* Fault-aware pruning (FAP) and *ii)* sign extension from 16 to 64 bits.

The first hardening strategy took inspiration from previous works published by [56], [57], and [58], which skip or drop to zero the results of faulty multiplier circuits in AI accelerators. This strategy leverages the typical data distribution in a DNN application, which concentrates around zero; therefore, forcing erroneous results to zero prevents significant shifts in the inner computation of the DNN, improving the fault resilience of these circuits in the context of DNN applications. Consequently, we evaluated the effectiveness of the FAP approach when adopted at the software level by dropping to zero those multiplication results that fall out of the expected boundaries of the application. It is worth noting that the evaluation of the fault countermeasure strategies focuses on the evaluation metrics of the arithmetic circuit proposed in this paper; we did not assess or provide further evaluation of the DNN outputs, which fall out of the scope of this work. However, considering that the proposed solution uses software instructions already available in the target device (e.g., GPU, CPU, etc.) to implement FAP, there is execution overhead that also may slightly impact the energy consumption in the host device. For example, if we consider a hypothetical scenario of a 32-bit multiplier in a GPU device, the FAP implementation will require two additional operations, a comparison, and a zeroed register operation. According to the data reported by [59], the average energy consumption of an integer multiplication in a GPU device is about 165.6pJ, and the energy consumption of logical and comparison operations corresponds to 28.3pJ and 91pJ, respectively. Thus in this particular scenario, the energy consumption overhead of the FAP strategy per every multiplication can reach about 119pJ extra, representing around 70% of the consumed energy by the multiplication only.

Figure 14 reports the main results in terms of MED, MOBE, and BER when applying the FAP fault countermeasure on 32-bit multipliers. In general, the results show considerable reliability improvements in terms of MED, MOBE, and BER when compared with uncorrected results reported in Figure 7, Figure 11 and Figure 12. In detail, the MED metric reports a significant error reduction for 32-bit multipliers, circuits A and B, in approximately 5 orders of magnitude compared to the uncorrected results presented in the Figure 7. Moreover, the MOBE metric indicates a significant improvement when computing DNN benchmarks. In fact, the number of operations correctly executed increased by approximately 2.5×10^8 operations for the ResNet models and 1×10^8 operations for LeNet5 in comparison with the uncorrected results depicted in the Figure 11. It must be noted that in the case of uniformly distributed data (i.e., Random 8-bit), there is negligible improvement in terms of MOBE, which remains almost the same as the uncorrected cases.

When considering the BER metric, we observed that the FAP increased the BER by 2.5 times for the Random 8-bit benchmark and by 0.9 times for the LeNet in comparison with the uncorrected results. Nonetheless, in the case of the ResNet models, there is a significant improvement in terms of BER by approximately 0.7 times compared with the uncorrected results reported in Figure 12. In conclusion, the FAP strategy has a positive impact on the reliability of 32-bit multiplier circuits since it is possible to correct a significant amount of operations $> 10^8$ operations when executing 8-bit DNN workloads. In addition, the FAP effectively reduces the error distance between the expected and faulty results, which positively contributed to increasing the reliability of the DNN workloads, as demonstrated by different authors in the literature [56], [57], [58]. On the other hand, FAP may, in some cases, significantly reduce the BER produced by faults on 32-bit multipliers while executing large DNN workloads.

The second strategy applies sign extension from 16 to 64/32 bits, which overwrites any error produced in the output representation of the 32-bit multiplier further than the 16 LSB of the result. This sign extension can be performed right after the execution of every multiplication and can be implemented at the software level by resorting to shift operations or by using data type casting in two stages. For example, casting from int32/64 to int16 will drop all MSB to zero, then a casting procedure from int16 to int32/64 will extend the sign and recover from an error that may propagate through the workload accumulator. The energy consumption per multiplication required by this software-based fault countermeasure can be calculated as in the previous example for FAP. Let's assume a GPU device scenario where two-shift operations are required: one shift left and one arithmetic shift right. In such a case, we assume that the shift operation consumes equivalent energy as an integer adder (91.3pJ). Therefore, the energy consumption overhead per multiplication corresponds to 182.6pJ, which is equivalent to 1.1 times the consumed energy by the multiplication only.

Figure 15 shows the results after applying the error correction based on sign extension from 16 to 64/32 bit. It is worth noting that the results show noteworthy reliability improvements in terms of MED, MOBE, and BER when compared with uncorrected results reported in Figure 7, Figure 11 and Figure 12. More in detail, the results demonstrate a significant reduction in terms of the MED, where 75% of the evaluated data for all benchmarks has an error close to Zero for both Circuit A and B. Similarly, the MOBE raised above 9×10^8 operations regardless of the evaluation benchmark. These results significantly surpassed the MOBE achieved by 8-bit multipliers. In addition, the proposed strategy allows a significant reduction of the BER for all application benchmarks executed on the 32-bit multiplier circuits. These results demonstrate that applying a sign extension from 16 to 32/64 bits at the result of the multiplication significantly improves the reliability of

the circuit but also has a positive impact on the executed application since there is a significant reduction of the error that can potentially be propagated.

Finally, we demonstrate that the proposed evaluation metrics allow us to effectively assess the reliability of arithmetic circuits but also provide a quantitative way of measuring the effectiveness of hardening strategies in the reliability of hardware used for DNN computations. In addition, we show that in the context of general-purpose platforms that use 32-bit hardware, it is possible to adopt software countermeasures that significantly improve the reliability of the hardware in the presence of faults while executing 8-bit DNN workloads.

VII. CONCLUSION

Arithmetic circuits are integral to any digital system; their significance encompasses various applications, from traditional computing tasks to advanced fields, such as AI and cryptography. Unfortunately, the outstanding advancement in semiconductor technologies makes electronic devices, including arithmetic circuits, increasingly prone to faults. Hence, the evaluation of arithmetic circuit reliability is of major importance in modern digital systems, especially in safety-critical applications, such as automotive, aerospace, and healthcare.

This paper presents a quantitative approach to the reliability evaluation of different multiplier circuits with respect to permanent faults under deep learning contexts. The proposed evaluation flow combines various evaluation metrics to measure the impact of faults, such as mean error distance (MED), mean relative error distance (MRED), mean square error (MSE), worse error distance (WED), and bit error rate (BER). In addition, this work proposes a novel evaluation strategy that quantifies and ranks the percentage of faults in the circuit according to their severity level in terms of MED and WED. Also, The mean operations between errors (MOBE) was introduced as an evaluation metric that quantifies the resilience of the circuit in terms of the average number of correctly computed operations between errors.

The experimental evaluations resort to four different integer multiplier circuits: two 32-bit Dadda multipliers and two 8-bit Booth multipliers architectures. Every multiplier circuit was evaluated using four application benchmarks: (1) Three representative Convolutional Neural Networks (CNNs) (LeNet5, ResNet18, and ResNet50) and (2) a random uniform sample of 8-bit operands. The CNN benchmarks were obtained by profiling one multiplier of a Tensor Core accelerator during the inference of an evaluation dataset. Then, an exhaustive stuck-at-fault injection campaign was carried out to study the impact of every fault on the operation of the evaluated circuits by using the traced information from the previous step.

The experimental results indicate that 32-bit multipliers used for computing 8-bit workloads exhibit significantly higher MED, MRED, and MSE figures than in the case of 8-bit circuits. On the other hand, the results indicate that the evaluation workload plays a crucial role in the evaluated circuits' reliability in terms of error distribution and error rate metrics. In conclusion, the reliability evaluation of an arithmetic circuit must include multiple evaluation viewpoints to rank criticality in a safety-critical system, also taking into account the specific reliability target parameters of each application scenario. Workload characteristics should be considered to gather realistic effects of faults affecting arithmetic circuits.

In addition, the experimental results showed that in the case of 32-bit multiplier circuits used in general-purpose computational devices (i.e., GPUs), it is feasible to adopt software-level solutions that contribute to reducing the impact of faults in those circuits as well as to enhance the reliability in terms of MOBE, MED and BER metrics. In fact, the experiments show that the proposed solutions (FAP and sign-extension from 16bit to 64/32 bit) enhance the reliability of 32-bit multipliers in the DNN contexts, by reducing the MED by about 5 order of magnitude, increasing the MOBE to more than 10^8 operations and reducing the BER by approx 0.8 times with respect to unhardened scenarios.

We highlight that the proposed method for reliability evaluation is general and can be adapted to deal with other arithmetic modules and other fault models. Albeit the reliability evaluation has been performed on 32 and 8-bit multipliers, it can be performed on any arithmetic circuit of arbitrary precision. Of course, when considering a higher precision circuit, e.g., a 64-bit multiplier, the fault injection campaigns will be longer as the fault list is bigger than the case of, e.g., an 8-bit circuit. However, it is important to note that the design trend for hardware used in AI-related applications is to employ lower-precision circuits.

Finally, future activities derived from the current work may involve the application of the proposed fault evaluation metrics (e.g., fault severity level, MOBE, and BER) to effectively conceive accurate error modeling of Silent Data Errors (SDEs) in modern computing systems to study the potential impact of faults at the application and system levels. Such levels of evaluation provide more realistic cross-layer fault evaluations that delve into the development of hardware or software hardening strategies capable of counteracting critical effects produced by an eventual defective arithmetic unit. Furthermore, in our future work we plan on considering other arithmetic units (e.g., adders, shifters, accumulators) as case studies.

REFERENCES

[1] I. Hill, P. Chanawala, R. Singh, S. A. Sheikholeslam, and A. Ivanov, "CMOS reliability from past to future: A survey of requirements, trends, and prediction methods," *IEEE Trans. Device Mater. Rel.*, vol. 22, no. 1, pp. 1–18, Mar. 2022.

[2] N. Gerlin, E. Kaja, F. Vargas, L. Lu, A. Breitenreiter, J. Chen, M. Ulbricht, M. Gomez, A. Tahiraga, S. Prebeck, E. Jentzsch, M. Krstić, and W. Ecker, "Bits, flips and RISCs," in *Proc. 26th Int. Symp. Design Diag. Electron. Circuits Syst. (DDECS)*, May 2023, pp. 140–149.

[3] A. Lee. (2022). *Train Spotting: Startup Gets on Track With AI and NVIDIA Jetson to Ensure Safety, Cost Savings for Railways*. [Online]. Available: https://resources.nvidia.com/en-us-jetson-success/rail-vision-startup-uses?lx=XRDs_y

[4] R. Mariani. (2023). *Driving Toward a Safer Future: NVIDIA Achieves Safety Milestones With DRIVE Hyperion Autonomous Vehicle Platform*. [Online]. Available: https://blogs.nvidia.com/blog/2023/04/20/nvidia-drive-safety-milestones/

[5] A. Lee. (2022). *AI at the Point of Care: Startup's Portable Scanner Diagnoses Brain Stroke in Minutes*. [Online]. Available: https://resources.nvidia.com/en-us-jetson-success/emvision-portable-br?lx=XRDs_y

[6] B. Peccerillo, M. Mannino, A. Mondelli, and S. Bartolini, "A survey on hardware accelerators: Taxonomy, trends, challenges, and perspectives," *J. Syst. Archit.*, vol. 129, Aug. 2022, Art. no. 102561.

[7] N. P. Jouppi et al., "In-datacenter performance analysis of a tensor processing unit," in *Proc. ACM/IEEE 44th Annu. Int. Symp. Comput. Archit. (ISCA)*, Jun. 2017, pp. 1–12.

[8] M. A. Raihan, N. Goli, and T. M. Aamodt, "Modeling deep learning accelerator enabled GPUs," in *Proc. IEEE Int. Symp. Perform. Anal. Syst. Softw. (ISPASS)*, Mar. 2019, pp. 79–92.

[9] *The International Roadmap for Devices and Systems: 2022*, IEEE, 2022.

[10] A. J. Strojwas, K. Doong, and D. Ciplickas, "Yield and reliability challenges at 7nm and below," in *Proc. 26th Int. Conf. 'Mixed Design Integr. Circuits Syst.' (MIXDES)*, Jun. 2019, pp. 52–55.

[11] H. Dattatraya Dixit, S. Pendharkar, M. Beadon, C. Mason, T. Chakravarthy, B. Muthiah, and S. Sankar, "Silent data corruptions at scale," 2021, *arXiv:2102.11245*.

[12] P. H. Hochschild, P. Turner, J. C. Mogul, R. Govindaraju, P. Ranganathan, D. E. Culler, and A. M. Vahdat, "Cores that don't count," in *Proc. Workshop Hot Topics Operating Syst.*, 2021, pp. 9–16.

[13] A. Singh, S. Chakravarty, G. Papadimitriou, and D. Gizopoulos, "Silent data errors: Sources, detection, and modeling," in *Proc. IEEE 41st VLSI Test Symp. (VTS)*, Apr. 2023, pp. 1–12.

[14] International Electrotecnical Commission, *Failure Modes and Effects Analysis (FMEA and FMECA)*, Standard IEC 60812:2018, 2018.

[15] NVIDIA Corporation. (2018). *NVIDIA Deep Learning Accelerator (NVDLA)*. [Online]. Available: http://nvdla.org/hw/v1/ias/unit_description.html

[16] B. Keller, R. Venkatesan, S. Dai, S. G. Tell, B. Zimmer, W. J. Dally, C. T. Gray, and B. Khailany, "A 17–95.6 TOPS/W deep learning inference accelerator with per-vector scaled 4-bit quantization for transformers in 5 nm," in *Proc. IEEE Symp. VLSI Technol. Circuits (VLSI Technol. Circuits)*, Jun. 2022, pp. 16–17.

[17] N. I. Deligiannis, R. Cantoro, M. Sonza Reorda, and S. E. D. Habib, "Evaluating the reliability of integer multipliers with respect to permanent faults," in *Proc. 27th Int. Symp. Design Diag. Electron. Circuits Syst. (DDECS)*, Apr. 2024, pp. 124–129.

[18] J.-W. Han, M. Meyyappan, and J. Kim, "Single event hard error due to terrestrial radiation," in *Proc. IEEE Int. Rel. Phys. Symp. (IRPS)*, Mar. 2021, pp. 1–6.

[19] M. White, "Scaled CMOS technology reliability users guide," NASA, Washington, DC, USA, Tech. Rep. 102197, 2010, Accessed: Apr. 1, 2024. [Online]. Available: https://nepp.nasa.gov/files/18209/09_102_5_JPL_White_Scaled%20CMOS%20Technology%20Reliability%20Users%20Guide%201_10.pdf

[20] D. F. Bacon, "Detection and prevention of silent data corruption in an exabyte-scale database system," in *Proc. 18th IEEE Workshop Silicon Errors Log. Syst. Effects (SELSE)*, Jun. 2022.

[21] R. Bonderson. (2021). *Training in Turmoil: Silent Data Corruption in Systems at Scale*. Accessed: Apr. 1, 2024. [Online]. Available: https://research.google/pubs/training-in-turmoil-silent-data-corruption-in-systems-at-scale/

[22] D. Fiala, "Detection and correction of silent data corruption for large-scale high-performance computing," in *Proc. IEEE Int. Symp. Parallel Distrib. Process. Workshops Phd Forum*, May 2011, pp. 2069–2072.

[23] S. Krishnaswamy, G. F. Viamontes, I. L. Markov, and J. P. Hayes, "Accurate reliability evaluation and enhancement via probabilistic transfer matrices," in *Proc. Design, Autom. Test Eur.*, vol. 1, Mar. 2005, pp. 282–287.

[24] J. Han, E. Taylor, J. Gao, and J. Fortes, "Faults, error bounds and reliability of nanoelectronic circuits," in *Proc. IEEE Int. Conf. Appl.-Specific Syst., Archit. Processors (ASAP)*, Jul. 2005, pp. 247–253.

[25] J. Han, H. Chen, E. Boykin, and J. Fortes, "Reliability evaluation of logic circuits using probabilistic gate models," *Microelectron. Rel.*, vol. 51, pp. 468–476, 2011.

[26] W. Ibrahim, V. Beiu, and M. Hussein Sulieman, "On the reliability of majority gates full adders," *IEEE Trans. Nanotechnol.*, vol. 7, no. 1, pp. 56–67, Jan. 2008.

[27] P. Raab, S. Krämer, and J. Mottok, "Error model and the reliability of arithmetic operations," in *Proc. Eurocon*, Jul. 2013, pp. 630–637.

[28] B. Srinivasu and K. Sridharan, "A transistor-level probabilistic approach for reliability analysis of arithmetic circuits with applications to emerging technologies," *IEEE Trans. Rel.*, vol. 66, no. 2, pp. 440–457, Jun. 2017.

[29] H. Jiang, C. Liu, N. Maheshwari, F. Lombardi, and J. Han, "A comparative evaluation of approximate multipliers," in *Proc. IEEE/ACM Int. Symp. Nanosc. Architectures (NANOARCH)*, Jul. 2016, pp. 191–196.

[30] M. Traiola, B. Deveautour, A. Bosio, P. Girard, and A. Virazel, *Test and Reliability of Approximate Hardware*. New York, NY, USA: Springer, 2022, pp. 233–266.

[31] Z. Wang, G. Zhang, J. Ye, and J. Jiang, "Reliability evaluation of approximate arithmetic circuits based on signal probability," in *Proc. IEEE Int. Test Conf. Asia (ITC-Asia)*, Aug. 2021, pp. 1–6.

[32] Z. Wang, G. Zhang, P. Liu, J. Ye, and J. Jiang, "Accurate reliability boundary evaluation of approximate arithmetic circuit," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 30, no. 10, pp. 1507–1518, Oct. 2022.

[33] K. Abbas, *Handbook of Digital CMOS Technology, Circuits, and Systems*. New York, NY, USA: Springer, 2020, pp. 439–469.

[34] C. S. Wallace, "A suggestion for a fast multiplier," *IEEE Trans. Electron. Comput.*, vol. EC-13, no. 1, pp. 14–17, Feb. 1964.

[35] L. Dadda, "Some schemes for parallel multipliers," in *Alta Frequenza*, vol. 34. Milan, Italy: Associazione Elettrotecnica Ed, 1965.

[36] S. Asif and Y. Kong, "Design of an algorithmic Wallace multiplier using high speed counters," in *Proc. 10th Int. Conf. Comput. Eng. Syst. (ICCES)*, Dec. 2015, pp. 133–138.

[37] B. Dinakar, K. B. Prasad, N. Thyauarajan, M. S. Ohileshwari, and G. B. K. Prasad, "8-bit modified booth multiplier using 20 nm FinFET technology," in *Proc. IEEE Int. Students' Conf. Electr., Electron. Comput. Sci. (SCEECS)*, Feb. 2022, pp. 1–6.

[38] V. D. Agrawal, A. Prasad, and M. V. Atre, "Fault collapsing via functional dominance," in *Proc. Int. Test Conf. (ITC)*, 2003, pp. 1–7.

[39] I. Pomeranz and S. M. Reddy, "Equivalence, dominance, and similarity relations between fault pairs and a fault pair collapsing process for fault diagnosis," *IEEE Trans. Comput.*, vol. 59, no. 2, pp. 150–158, Feb. 2010.

[40] T. Grigoryan, H. Malkhasyan, G. Mushyan, and V. Vardanian, "Fault collapsing for digital circuits based on relations between stuck-at faults," in *Proc. Comput. Sci. Inf. Technol. (CSIT)*, Sep. 2015, pp. 15–18.

[41] Synopsys. *TestMAX ATPG*. Accessed: Jun. 3, 2024. [Online]. Available: https://www.synopsys.com/implementation-and-signoff/test-automation/testmax-atpg.html

[42] R. L. Sierra, J. D. Guerrero-Balaguera, J. E. R. Condia, and M. S. Reorda, "Analyzing the impact of different real number formats on the structural reliability of TCUs in GPUs," in *Proc. IFIP/IEEE 31st Int. Conf. Very Large Scale Integr. (VLSI-SoC)*, Oct. 2023, pp. 1–6.

[43] H. Jiang, F. J. H. Santiago, H. Mo, L. Liu, and J. Han, "Approximate arithmetic circuits: A survey, characterization, and recent applications," *Proc. IEEE*, vol. 108, no. 12, pp. 2108–2135, Dec. 2020.

[44] F. J. H. Santiago, H. Jiang, H. Amrouch, A. Gerstlauer, L. Liu, and J. Han, "Characterizing approximate adders and multipliers for mitigating aging and temperature degradations," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 69, no. 11, pp. 4558–4571, Nov. 2022.

[45] J. Liang, J. Han, and F. Lombardi, "New metrics for the reliability of approximate and probabilistic adders," *IEEE Trans. Comput.*, vol. 62, no. 9, pp. 1760–1771, Sep. 2013.

[46] J. Dongarra, H. Meuer, and E. Strohmaier. (2015). *ISO26262 Standard*. [Online]. Available: https://www.iso.org/obp/ui/#iso: std: iso:26262:-1: ed-1: v1: en

[47] K. Gurusankar. *Booth Multiplier in Verilog*. Accessed: Jun. 10, 2024. [Online]. Available: https://github.com/Guru227/Booth-Multiplier-in-iverilog

[48] NVIDIA Corporation. *NVDLA—Open Source Hardware V1.0*. Accessed: Jun. 10, 2024. [Online]. Available: https://github.com/nvdla/hw/tree/nvdlav1

[49] *Silvaco 45 nm Open Cell Library*. Accessed: Dec. 5, 2023. [Online]. Available: https://si2.org/open-cell-library

[50] S. Subramanian, M. Saroufim, and J. Zhang. (2022). *Practical Quantization in PyTorch*. [Online]. Available: https://pytorch.org/blog/quantization-in-practice/

[51] R. Krishnamoorthi, "Quantizing deep convolutional networks for efficient inference: A whitepaper," 2018, *arXiv:1806.08342*.
[52] H. Wu, P. Judd, X. Zhang, M. Isaev, and P. Micikevicius, "Integer quantization for deep learning inference: Principles and empirical evaluation," 2020, *arXiv:2004.09602*.
[53] N. Deligiannis. (May 2024). *finjenv: Fault Injection Environment for Arithmetic Circuits*. [Online]. Available: https://github.com/NikosDelijohn/finjenv
[54] M. Hubert and E. Vandervieren, "An adjusted boxplot for skewed distributions," *Comput. Statist. Data Anal.*, vol. 52, no. 12, pp. 5186–5201, Aug. 2008.
[55] C. Anderson-Bergman, "An efficient implementation of the EMICM algorithm for the interval censored NPMLE," *J. Comput. Graph. Statist.*, vol. 26, no. 2, pp. 463–467, Apr. 2017.
[56] J. J. Zhang, K. Basu, and S. Garg, "Fault-tolerant systolic array based accelerators for deep neural network execution," *IEEE Des. Test*, vol. 36, no. 5, pp. 44–53, Oct. 2019.
[57] M. Abdullah Hanif and M. Shafique, "SalvageDNN: Salvaging deep neural network accelerators with permanent faults through saliency-driven fault-aware mapping," *Philos. Trans. Roy. Soc. A, Math., Phys. Eng. Sci.*, vol. 378, no. 2164, Feb. 2020, Art. no. 20190164.
[58] M. Sadi and U. Guin, "Test and yield loss reduction of AI and deep learning accelerators," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 41, no. 1, pp. 104–115, Jan. 2022.
[59] J. Lucas and B. Juurlink, "ALUPower: Data dependent power consumption in GPUs," in *Proc. IEEE 24th Int. Symp. Modeling, Anal. Simulation Comput. Telecommun. Syst. (MASCOTS)*, Sep. 2016, pp. 95–104.

NIKOLAOS I. DELIGIANNIS (Member, IEEE) received the M.Sc. degree in computer science and engineering from the Department of Computer Science and Engineering, University of Ioannina, Greece, in 2019. He is currently pursuing the Ph.D. degree with the Department of Control and Computer Engineering, Politecnico di Torino. He was a Research Assistant with the Department of Control and Computer Engineering, Politecnico di Torino. His research interests include testing processors using formal methods and fault tolerance.

JUAN-DAVID GUERRERO-BALAGUERA (Member, IEEE) received the B.Sc. and M.Sc. degrees in electronics from Universidad Pedagógica y Tecnológica de Colombia (UPTC), Sogamoso, Colombia. He is currently pursuing the Ph.D. degree with the Department of Control and Computer Engineering, Politecnico di Torino, Turin, Italy. His research interests include digital design, fault-tolerant AI hardware, functional tests, artificial intelligence, and parallel architectures.

RICCARDO CANTORO (Member, IEEE) received the M.Sc. and Ph.D. degrees in computer engineering from Politecnico di Torino, Italy, in 2013 and 2017, respectively. He is currently an Assistant Professor with the Department of Computer Engineering, Politecnico di Torino. His research interests include software-based functional testing of SoCs and memories, and machine learning applied to test and diagnosis.

SERAG E. D. HABIB (Life Senior Member, IEEE) received the B.Sc. degree in electronics engineering from Cairo University, Egypt, in 1968, the M.Sc. degree in solid state physics from AUC, Egypt, in 1971, and the M.Sc. and Ph.D. degrees in electrical engineering from the University of Toronto, Canada, in 1973 and 1978, respectively. He is currently a Professor Emeritus at the Electronics Department, Cairo University. He has authored and co-authored one patent, three books, and over 100 papers. He shared in many funded research projects in Egypt, Kuwait, and Wales. His research interests include VLSI design and semiconductor devices. He received several Egyptian awards, including the Appreciation Award from Cairo University, in 2016.

MATTEO SONZA REORDA (Fellow, IEEE) received the M.Sc. degree in electronics and the Ph.D. degree in computer engineering from Politecnico di Torino, Italy, in 1986 and 1990, respectively. He is currently a Full Professor with the Department of Control and Computer Engineering (DAUIN), Politecnico di Torino. He has published more than 400 papers in the area of test and fault-tolerant design of reliable circuits and systems. He is involved in numerous research projects with companies and other research centers worldwide. He received several best paper awards at major international conferences.