Evaluating the Reliability of Integer Multipliers With Respect to Permanent Faults

Evaluating the Reliability of Integer Multipliers With Respect to Permanent Faults

Nikolaos I. Deligiannis†, Riccardo Cantoro†, Matteo Sonza Reorda†, S. E. D. Habib*
† Politecnico di Torino, Department of Control and Computer Engineering (DAUIN) - Turin, Italy
* Cairo University, Department of Electronics and Electrical Communications Engineering - Giza, Egypt

***Abstract*—Arithmetic circuits form the foundation of modern digital computation, enabling us to conduct precise mathematical operations and drive the digital age. They are integral components in nearly every digital circuit, such as processors' arithmetic and logic units. Especially in safety-critical domains like automotive and aviation, the flawless operation of these circuits is of paramount importance. This paper presents a case study involving two variants of Dadda multipliers and assesses their intrinsic reliability when affected by permanent hardware faults. We conducted extensive fault injection campaigns on the circuit models under various datasets, presenting the aggregated statistical errors in the form of the mean absolute error (MAE) for each case. Specifically, we performed fault injection campaigns in which the operands are sourced from trained quantized weights of a convolutional neural network, as well as randomly generated sets of integers. The results not only reveal differences between the two circuits but also show significant variations when different datasets are used in the fault injection campaigns.**

***Index Terms*—Reliability, Stuck-At Fault, Fault Injection Campaign, Arithmetic Circuits, Dadda Multipliers**

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.

Multiplication is a fundamental operation extensively utilized in different circuits used also in safety-critical domains. As an example, multipliers are part of CPUs and GPUs used in several crucial applications, such as in-flight control computers that govern critical aspects of aircraft operation and safety. In the area of autonomous vehicles, multipliers are key components employed in various systems (e.g., arithmetic processors, tensor processing units) and used for the execution of machine learning applications. For instance, multipliers enable the operation of the mathematical convolution (in convolutional neural networks) for local feature extraction from the input data and for down-sampling (pooling) to shrink the size of the input data representation.

The presence of a fault in a critical component (e.g., a multiplier within a tensor core executing computer vision algorithms for an autonomous vehicle) could potentially produce a system failure that could have catastrophic consequences for the surrounding environment or endanger human life. Hence, in such scenarios, the manufacturers must further provide strict reliability guarantees, as mandated by the respective standards. The safety standards mandate first of all certain high fault coverage thresholds to be met during the testing phases. Additionally, each system to be used in a safety-critical environment must undergo *failure mode effects and criticality analysis* (FMECA) [1]. FMECA is an inductive analytical process used to chart the probability of failure modes against the severity of their consequences. It proposes a methodology to assess the criticality of the system overall and identify failure modes of the system, such as an erroneous output under the presence of a fault. In this way, the overall reliability of the system can be identified.

In summary, the evaluation of arithmetic circuit reliability is of major importance in modern digital systems. Through adherence to safety standards, fault coverage testing, and the application of methodologies like FMECA, manufacturers can ensure that their systems meet the required levels of reliability and safety. By addressing potential failure modes and their consequences, these practices contribute to the overall dependability and accuracy of circuits, thereby minimizing the risks associated with system failures.

A wide variety of multiplier architectures have been proposed [2] differing in terms of area, speed, power, and reliability. It is up to the designer to identify the circuit version that best suits the application criteria. In this paper we describe an approach to characterize the reliability of an arithmetic module, considering two variants of integer multiplier circuits as a test case. We present the results of an extensive reliability analysis performed on the two circuit variants. We consider the case of permanent hardware faults and present the results of fault injection campaigns performed on both circuits under various workloads. Most importantly, in the set of stimuli used for our fault injection experiments, we also include stimuli taken from a processor trace from the execution of a convolutional neural network application. We also evaluate the overall reliability of the two circuits by considering aspects such as circuit area and commercial ATPG results. Besides experimentally identifying the more resilient multiplier architecture, we show that the presence of application-specific stimuli plays a vital role in accurately forming a reliability verdict for a design.

The rest of the paper is organized as follows. In section II-A we present some relevant works in the area, whereas in section II-B we provide some information about the architecture of our two multipliers, which we use as circuits under test (CUTs). In section III we present our fault injection campaign flow, and

in section IV we present the experimental results we gathered. Lastly, in section V we draw some conclusions and provide insights on our future work.

II. Background

In this section, we will first provide some information on relevant topics in the general area of the reliability of arithmetic circuits and will then provide some information about the Dadda multipliers, which we use as CUTs in our experiments.

A. 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 [3, 4, 5].

In [6] the authors evaluate the reliability of various majority gates full adder circuits while comparing against XOR-based full-adders at the gate level. The study reveals that majority gates 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 [7] 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 [8] 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 [9] 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.

Lastly, regarding arithmetic circuits based on the approximate computing paradigm, in [10] 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.

B. Multiplication

Multiplication is one of the most critical operations in an arithmetic circuit, since it significantly impacts the circuit's speed and power dissipation. This is due to the complex nature of multiplication algorithms and the extensive number of logic gates required to perform the operation. Efficient multiplication is crucial in various digital systems, such as processors, signal processors, and other computational devices, where speed and power efficiency are paramount.

In the general case of two n bit numbers, their multiplication is performed as:

$$
\begin{array}{llllllll}
 & & & & \times & a_{n-1} & a_{n-2} \cdots & a_0 \\
 & & & & & b_{n-1} & b_{n-2} \cdots & b_0 \\
\hline
 & & & & & z_{0,n-1} & z_{0,n-2} \cdots & z_{0,0} \\
 & & & & z_{1,n-1} & z_{1,n-2} \cdots & z_{1,0} & \\
 & & & \iddots & \iddots & \iddots & \iddots & \\
+ & & z_{n-1,n-1} & z_{n-1,n-2} & \cdots & z_{n-1,0} & & \\
\hline
p_{2n-1} & p_{2n-2} & & p_{2n-3} & & \cdots & p_1 & p_0
\end{array}
$$

However, in this "brute-force" approach, the summands ($z_{i,j}$) that are generated by the shifting of the multiplicand are equal to the number of bits (n). Hence, an inefficiently large amount of full-adders (FAs) and half-adders (HAs) must be used, introducing more area to the circuit. Furthermore, the usage of more FA and HA primitives would further cascade the delay of the circuit [11].

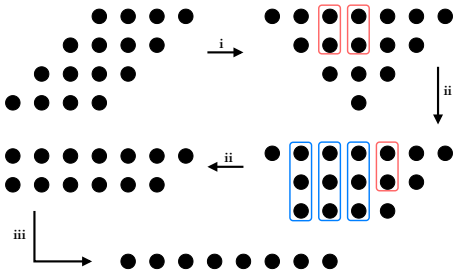


Figure 1: Summand reduction using Dadda strategy for d=3,2 (see [12, 11]). Dots represent $z_{i,j}$ summands and red/blue boxes represent HAs/FAs, respectively.

Wallace [13] and Dadda [12] tree multipliers are high speed multipliers. The main idea behind these multipliers 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.

In Figure 1, we illustrate an example of the Dadda multiplication algorithm for 4-bit operands. The algorithm is carried 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 Dadda 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 [14], which resulted in even faster and more efficient multiplier circuits. In this paper, we will use the 32-bit Dadda multipliers as a reliability case study without loss of generality. That is, the same evaluation methods can be applied to any kind of digital combinational multiplier circuit.

III. Experimental Setup

Our goal is to perform a fair comparison between the two versions of a signed multiplier circuit in terms of intrinsic reliability. The two signed combinational multiplier circuits at the RT-level, initially designed by members of the Aoki laboratory at Tohoku University, were synthesized using a technology library to obtain their gate-level representations. This process is facilitated through the use of commercial EDA tools. After the gate-level model is produced along with the respective circuit information (number of cells, area, etc.) we proceed to the next step in our flow, which is to perform automatic test pattern generation (ATPG) for permanent stuck-at faults (once again resorting to commercial EDA tools). As the circuits are combinational, the ATPG process is anticipated to achieve high coverage percentages. Upon completing this process, we export the results (i.e., fault coverage and number of patterns), including the collapsed fault lists for the circuits generated by the fault manager. Subsequently, we move on to the final step, which involves conducting fault injection campaigns to comprehensively assess the resilience of the circuits under each permanent fault but also under each input stimulus source.

Fault Injection Campaigns

The effort and time of a fault injection campaign strongly depend on the total number of faults that have to be injected in the CUT. Typically, due to the huge number of possible error configurations in circuits, a random selection of a subset of potential faults is usual in practical experiments. In our case however, given that we consider circuits that are typically used as a part of a system (e.g., the ALU of a processor) exhaustive fault injection campaigns can be performed since the total number of collapsed stuck-at faults is reasonable.

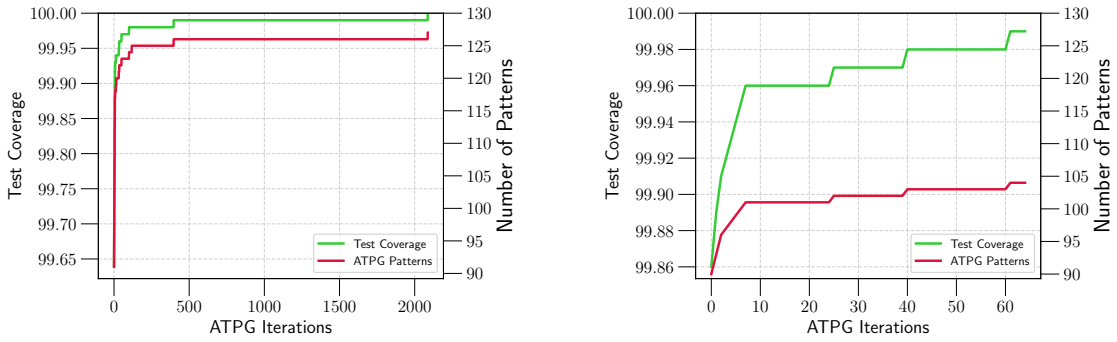
Another important aspect of the fault injection campaigns is the stimulus source for the CUT. Specifically, for the case of the multiplier circuits, the stimuli specify the pairs of integer operands that will be fed to the circuit while a stuck-at fault is present at a time. There are two kinds of stimuli that we have considered for this work, namely, quantized 8-bit integers from a convolutional neural network and randomly generated pairs of 8/16/32-bit integers. In greater detail, the former set was identified and extracted by the execution trace of a convolutional neural network on a RISC-V processor. A testbench housing the processor core and a memory loaded with the cross-compiled version of the trained network was used for dumping at each clock cycle the executed instruction and its operands. The operand pairs from the classification phase of the network were identified and stored as our first stimulus source for our fault injection campaigns.

```
1  Results ← ∅
2  foreach stimulus source D do
3      Errors ← ∅
4      foreach stuck-at fault sa in fault list do
5          FM_responses ← ∅
6          GM_responses ← ∅
7          inject_fault(sa)
8          foreach operand pair (a, b) in D do
9              FM_responses = FM_responses ∪ CUT (a, b)
10             GM_responses = GM_responses ∪ a × b
11         end
12         remove_fault(sa)
13         Errors = Errors ∪ mae(FM_responses, GM_responses)
14     end
15     Results = Results ∪ { (Errors, D) }
16 end
17 return Results
```

Figure 2: Fault Injection Campaigns routine for CUT.

The second input stimulus source is not application-specific and was artificially generated. We considered three distinct scenarios, specifically involving 8x8-bit operands, 16x16-bit operands, and 32x32-bit operands, respectively. Each set of random operands within these three cases comprises 20,000 numbers, forming a total of 10,000 pairs. These pairs serve as the second stimulus source for the CUTs during the fault injection campaigns. The chosen stimuli sample sizes were computed with a 95% confidence level, incorporating a margin of error of approximately 1%.

The fault injection campaign is orchestrated via a testbench circuit. The testbench is responsible for importing the fault list and an input stimulus source, and in an iterative fashion, it injects one stuck-at fault at a time into the CUT. Under the presence of the fault, the CUT becomes the *faulty machine* (FM). Then, the testbench feeds the operand pairs to the FM while logging its responses. In parallel to that, the testbench circuit performs the fault-free computation acting as the *golden machine* (GM). When all operands have been processed, the FM and GM responses are post-processed to compute the Mean Absolute Error (MAE) of **each** injected stuck-at fault. The MAE is computed as $\frac{1}{N} \times \sum_{i=1}^{N} |X_i^{\text{GM}} - X_i^{\text{FM}}|$, where *N* is the total number of operand pairs of the input stimulus source, X_i^{GM} the response of the golden machine for the operand pair i and X_i^{FM} the response of the faulty machine for the



(a) Circuit A

(b) Circuit B

Figure 3: ATPG results.

operand pair *i*. The fault injection campaigns are summarized in pseudo-code format in Figure 2.

IV. Experimental Results

To provide an example of the application of the proposed method, we considered two different implementations of a 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, and *Circuit B* is a variant that uses 3 to 2 compressors and a Han-Carlson adder, respectively [14]. All of our experiments were performed in a system using 2 Intel(R) Xeon(R) Gold 6238R processors with 256 GB of RAM. The convolutional neural network from which the first input stimulus source of 8-bit quantized integers was derived was LeNet [15], trained on the MNIST dataset of handwritten digits and executed on the RI5CY [16] processor. Both circuits were synthesized using the Nangate 45nm technology library [17] via Synopsys Design Compiler. The first half of Table I shows the details of the logic synthesis for both circuits. Circuit A has more cells than circuit B and occupies a smaller area.

Table I: Circuits' details.

Circuit	Area	Cells			# Stuck-At
		Comb.	Seq.	Buf./Inv.	
A	4,912.22	3,299	0	708	12,456
B	5,721.92	2,729	0	250	10,512

A. ATPG

The next step in our flow is to run an ATPG, resorting to TestMAX by Synopsys. This step primarily generates the two collapsed stuck-at fault lists of the circuits needed for our subsequent fault injection campaigns. Given the combinational nature of both CUTs, the ATPG process is expected to yield high fault coverage percentages and thus can be considered trivial. Albeit trivial, it further acts as an extra validation step to our synthesis flow, e.g., by showing no untested faults in the circuits. The rightmost part of Table I shows both circuits' collapsed, stuck-at fault list sizes. As expected, given that circuit A is composed of more cells, its fault list is more extensive than the fault list of circuit B, making it statistically more fault-prone. The results of the ATPG are shown in Figure 3. As expected, we achieve $> 99.98\%$ of fault coverage for both circuits. A difference can be observed in the number of patterns needed for the two circuits, with circuit A requiring 127 patterns and circuit B 104, respectively. Regarding circuit B, one stuck-at fault was not detected during the ATPG (no matter the effort) due to a blockage in its propagation path. Thus, the fault was marked as *not observable*. Performing ATPG alone doesn't provide insights into the impact and criticality of each stuck-at fault. Therefore, we move on to the next step, which involves fault injection campaigns.

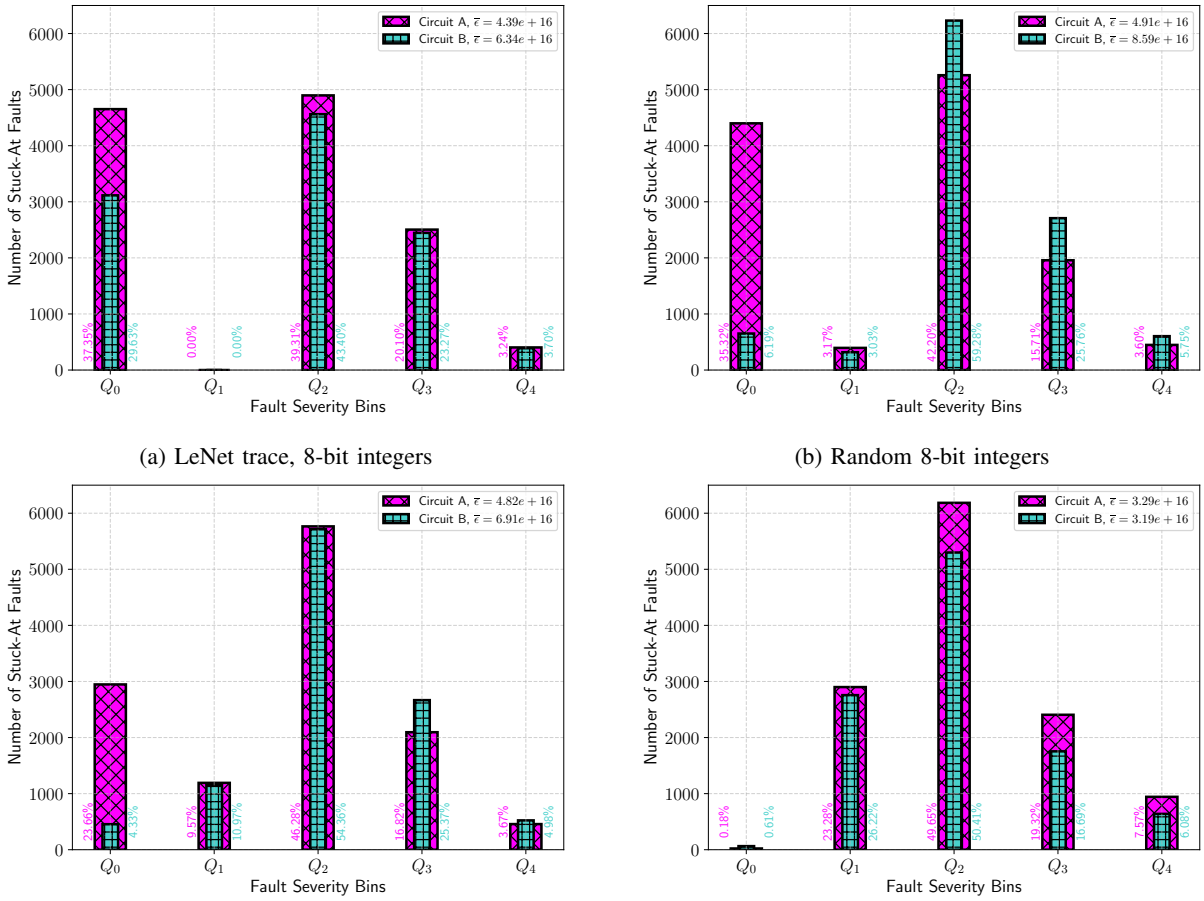
B. Fault Injection Campaigns

The primary goal of the fault injection campaigns was to assess the impact that possible permanent faults may produce on the results produced by the CUT.

The environment used for performing the fault injection campaigns is based on QuestaSIM by Siemens EDA. In Figure 4, we compare the behavior of the two circuits for each input stimulus source. A fault injection campaign was conducted for each source, during which every stuck-at fault was injected. Subsequently, MAE values were computed based on the results of each campaign. The computed MAE values for each circuit and each input source were also binned into statistical quartiles (that we call *Fault Severity Bins*). The quartiles are defined as:

Table II: Average MAE per circuit and input source.

Circuit	Average MAE	Input source
A	4.3929e+16	LeNet 8-bit
B	6.3396e+16	
A	4.9119e+16	Random 8-bit
B	8.5936e+16	
A	4.8181e+16	Random 16-bit
B	6.9117e+16	
A	3.2906e+16	Random 32-bit
B	3.1933e+16	



(a) LeNet trace, 8-bit integers

(b) Random 8-bit integers

(c) Random 16-bit integers

(d) Random 32-bit integers

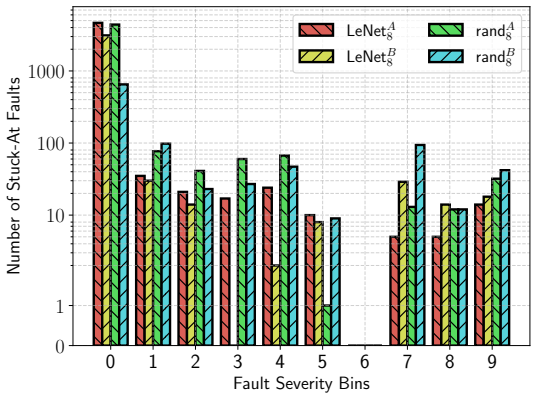
Figure 4: Fault Injection campaigns' results per dataset.

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.

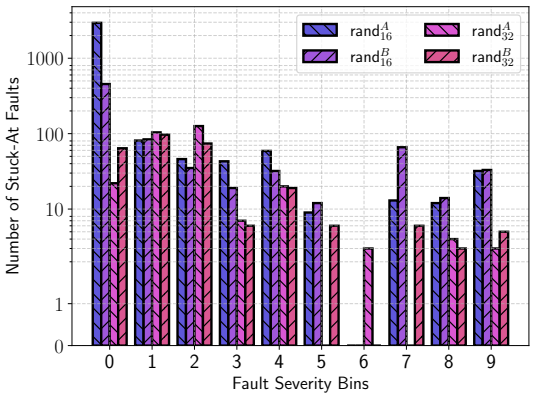
For all experiments, the Q_0 bin contains the stuck-at faults, which do not impact the outcome of the multiplication. Each other bin contains the faults for which the MAE value belongs to the interval $(Q_{i-1}, Qi]$. Moreover, Table II provides the average MAE value (over all faults) for each circuit and input stimulus source. These same values are also incorporated into the legends of the corresponding plots in Figure 4. Faults belonging to the highest bins (i.e., Q_3 and Q_4) are those the FMECA procedure will identify as most critical.

1) LeNet Trace Source: Regarding the input stimulus source derived from the execution of LeNet (Figure 4a), we observe that circuit A is more resilient than B since, in its case, a larger number of faults exist with minimum severity. In particular, 37.35% and 29.63% of the total stuck-at faults were found to produce an MAE of 0 for circuits A and B, respectively. When comparing the two circuits against faults in the Q_2 and Q_3 bins, we observe an almost similar trend with small differences in the absolute number of faults in each bin. Lastly, for the faults with the highest severity (i.e., the stuck-at faults in the Q_4 bin), the percentages are quite similar, with circuit A having 3.24% and circuit B having 3.70% of faults in this bin. Overall, Circuit A's average error is lower than Circuit B's, rendering it more resilient.

2) Randomly Generated Sources: Regarding the randomly generated input source (Figures 4b to 4d), we can initially make a general observation: as the operand size increases, the number of faults producing 0 MAE values decreases in an inversely-proportional manner. This phenomenon can be explained by considering a fault in the multiplier part responsible for processing the upper 24 or 16 bits of the operands when the actual operands are merely 8 or 16, respectively. In such cases, the fault is highly likely not to produce any effect. This probability of fault masking is attributable to the zero-padding of operands to a 32-bit length. As the actual operands are smaller, they are extended with zeros to meet the 32-bit requirement. The multiplication with zero during this padding process substantially augments the likelihood of nullifying the fault effect. Regarding the resilience of the circuits, we can see that Circuit A once again has a greater resilience than



(a) LeNet trace and randomly generated set of 8-bit operands.



(b) Randomly generated sets of 16/32-bit operands.

Figure 5: Accumulated MAE plots. Legend entries' subscripts indicate the input source and superscripts the CUT.

Circuit B.

Emphasizing the significance of application-specific data is crucial. Notably, when examining Figure 4a and Figure 4b, discernible differences are evident for both circuits since the number of faults producing potentially critical failures changes significantly. Once again, the percentage of faults producing the largest error (rightmost bin) is larger for random datasets than for the LeNet ones. In other words, the variations in error distribution are closely tied to the specific stimulus source under consideration. In this specific case, we could thus say that using random values, we get pessimistic results in terms of safety. This disparity can be ascribed to the nature of the application-specific input source, which consists of smaller values compared to its randomly generated 8-bit counterpart. This discrepancy likely arises from the presence of zeros in the trained weights of the neural network, aimed at eliminating redundant connections within the hidden layers of the network. Therefore, forming a conclusive judgment based solely on randomly applied patterns during our reliability evaluation would lack precision and accuracy.

To further validate this behavior, we increased the data binning granularity and plotted in a semi-log scale the MAE for each randomly generated stimulus source (Figure 5b) against the MAE values for the LeNet derived one (Figure 5a) for the two CUTs. We again see that with the randomly generated input source data, a higher percentage of faults produce more critical effects.

V. Conclusions

The evaluation of arithmetic circuit reliability is of major importance in modern digital systems. In this paper, we consider the case of two integer multiplication circuits, and we report results coming from a thorough reliability evaluation under various stimuli sources by performing exhaustive stuck-at fault injection campaigns.

The experimental results show that the presence of application-specific stimulus plays a significant role in the evaluation of the reliability of a CUT and that a randomly generated input stimulus source may not be accurate enough to model a real application scenario. In future work, we plan to further investigate this observed difference by considering different application-specific stimuli and different arithmetic circuits as our CUTs. We also plan to extend our work to consider other types of fault models.

References

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

[2] B. Parhami. *Computer Arithmetic: Algorithms and Hardware Designs*. Oxford University Press, 2000.

[3] S. Krishnaswamy et al. "Accurate Reliability Evaluation and Enhancement via Probabilistic Transfer Matrices". In: *Design, Automation & Test in Europe (DATE)*. 2005.

[4] Jie H. et al. "Faults, error bounds and reliability of nanoelectronic circuits". In: *IEEE International Conference on Application-Specific Systems, Architecture Processors (ASAP)*. 2005.

[5] H. Jie et al. "Reliability Evaluation of Logic Circuits using Probabilistic Gate Models". In: *Microelectronics Reliability* 51 (2011), pp. 468–476.

[6] W. Ibrahim et al. "On the Reliability of Majority Gates Full Adders". In: *IEEE Transactions on Nanotechnology* 7 (2008), pp. 56–67.

[7] P. Raab et al. "Error Model And the Reliability of Arithmetic Operations". In: *Eurocon*. 2013.

[8] B. Srinivasu et al. "A Transistor-Level Probabilistic Approach for Reliability Analysis of Arithmetic Circuits With Applications to Emerging Technologies". In: *IEEE Transactions on Reliability* 66 (2017), pp. 440–457.

[9] H. Jiang et al. "A Comparative Evaluation of Approximate Multipliers". In: *IEEE/ACM International Symposium on Nanoscale Architectures (NANOARCH)*. 2016.

[10] M. Traiola et al. "Test and Reliability of Approximate Hardware". In: *Approximate Computing*. Springer International Publishing, 2022, pp. 233–266.

[11] K. Abbas. "Arithmetic". In: *Handbook of Digital CMOS Technology, Circuits, and Systems*. Springer International Publishing, 2020, pp. 439–469.

[12] L. Dadda. "Some Schemes for Parallel Multipliers". In: *Alta Frequenza* 34 (1965).

[13] C. S. Wallace. "A Suggestion for a Fast Multiplier". In: *IEEE Transactions on Electronic Computers* EC-13 (1964), pp. 14–17.

[14] S. Asif et al. "Design of an Algorithmic Wallace multiplier using High Speed Counters". In: *Tenth International Conference on Computer Engineering & Systems (ICCES)*. 2015.

[15] LeCun, Y. *LeNet-5*. http://yann.lecun.com/exdb/lenet/.

[16] *PULP*. https://pulp-platform.org/.

[17] *Silvaco 45nm Open Cell Library*. https://si2.org/open-cell-library.