Guided Fault Injection Strategy for Rapid Critical Bit Detection in Radiation-Prone SRAM-FPGA

Trishna Rajkumar
KTH Royal Institute of Technology
Stockholm, Sweden
trishna@kth.se

Johnny Öberg
KTH Royal Institute of Technology
Stockholm, Sweden
johnnyob@kth.se

Abstract—**Fault injection test is vital for assessing the reliability of SRAM-FPGAs used in radiative environments. Considering the scale and complexity of modern FPGAs, exhaustive fault injection is tedious and computationally expensive. A common approach to optimising the injection campaign involves targeting a subset of the configuration memory containing essential and critical bits crucial for the system's functionality. Identifying *Essential bits* in an FPGA design is often feasible through manufacturer documentation. However, detecting *Critical bits* requires complex reverse engineering to map the correspondence between the configuration bits and the FPGA modules. This task requires substantial amount of details about the logic layout and the bitstream, which is not easily available due to their proprietary nature. In some cases, manual floorplanning becomes necessary, which could impact the performance of the application. Given these limitations, we examine the potential of Monte Carlo Tree Search in guiding the fault injection process to identify critical bits with minimal injections. The key benefit of this approach is its ability to harness the spatial relations among the configuration bits without relying on reverse engineering or offline campaign planning. Evaluation results demonstrate that the proposed approach achieves a 99% coverage using 18% fewer injections than traditional methods. Notably, 95% of the critical bits were detected in under 50% injections, achieving at least 2X higher sensitivity to critical bits with a minimal overhead of 0.04%.**

Index Terms—**Emulation-based Fault injection, Critical bits, Monte Carlo Tree Search, Single Event Upset**

I. Introduction

With the continued scaling of CMOS process, SRAM-FPGAs have emerged as the high-performance computing platform for applications ranging from underground accelerators to safety-critical space systems. However, as the feature size shrinks, the SRAM cells that store the configuration data of the FPGA become increasingly susceptible to radiation effects. This is due to the lowering of the charge threshold needed to corrupt the SRAM cells. One of the most common radiation error in modern SRAM-FPGAs is the Single Event Upset (SEU), wherein a single charged particle changes the logical state of a configuration bit. Although SEU can be mitigated through error correction codes, redundancy, and memory scrubbing techniques, modern SRAM-FPGAs are not completely radiation-proof. In order to measure the radiation susceptibility of FPGA resources and to validate the mitigation approaches, the FPGA is subjected to fault injection experiments. Fault injection is used to measure the probability and impact of SEU, to identify vulnerable areas for error propagation, and is commonly employed to ensure compliance with reliability requirements in safety-critical applications.

During fault injection, SEU is emulated by changing the state of the configuration memory, that is, a random bit is inverted, the resulting effect is analysed and the process is repeated for all the configuration bits. While this technique is quite effective, it often suffers from lengthy simulation times and scalability challenges owing to the large volume of the configuration bits. Considering that the size of the memory in modern FPGA designs could vary from a few megabits to over a gigabit, exhaustive fault injection can be computationally unmanageable and prohibitively time-consuming, especially for complex systems with a large number of fault scenarios. A common optimisation technique for minimising injections involves filtering the configuration bits to identify the critical subset responsible for system failure. This is based on the key insight that only a fraction of the available configuration bits are *essential* for proper functionality and only a subset of these essential bits, referred to as *Critical bits*, cause system failures when affected by SEU. The volume of the critical bits depends on the application but typically they constitute 5-10 % of the essential bits [1]. Therefore, targeting only these bits could improve the test duration by orders of magnitude.

Although exhaustive injection of essential bits has been effective in reliability assessment [2]–[5], recent advancements have sought to direct injections at critical bits using statistical approaches [6]–[8] and by reversing the FPGA architecture [2], [3], [9]. The former method reduces the sample size of the essential bits by considering only statistically significant injections whereas the latter relies on identifying the critical bits by mapping the FPGA modules to their corresponding configuration bits. Some of the notable fault injection tools like ACME [2] and BAFFI [9] employ reverse-engineering to target specific modules in the design and inject faults in lower hierarchial cells like lookup tables (LUT), registers etc. While reversing the FPGA layout has shown to be beneficial for optimising fault injection [3], the main challenge here is that it requires extensive knowledge of the fabric layout which is often scarce due to the proprietary nature of the FPGA. This entails significant engineering effort requiring the designer to intervene in the CAD flow (manual floorplanning), work at a lower abstraction level, and make assumptions about the underlying structure adding to the complexity of the design

TABLE I: Frame Address Format (Xilinx Ultrascale+)

Die	Block	Row	Column	Minor	Word	Bit
bit [41:40]	[38:36]	[35:30]	[29:20]	[19:12]	[11:5]	[4:0]

process. Moreover, this approach tends to be architecture-specific, limiting its portability across different FPGA families.

Considering these challenges, we investigate the potential of utilizing Monte Carlo Tree Search (MCTS) to optimise the injection methodology (reduce the number of injections) without reversing the FPGA layout. The goal of the proposed approach is to streamline the injection process by efficiently searching the fault space to identify the location of critical configuration bits. This is achieved by exploiting the spatial relations in the configuration memory through a combination of random sampling and learned exploration of the memory. Evaluations show that the MCTS based injection campaign requires 18% fewer injections than random campaign to identify 99% of critical bits while also being at least twice as fast as manually planned campaign. In fact, 95% of the critical bits could already be detected in under 50% of the injections required by the conventional approach. The MCTS strategy does not require any domain- or FPGA-specific information except the address format of the configuration memory made available by the FPGA vendors. Therefore, it is not only more efficient than the traditional fault injection but also can be generalised to FPGAs of different architecture. *To the best of our knowledge, this is the first guided injection methodology to accelerate critical bit detection through dynamic interaction with the configuration memory.*

II. Background and Related work

This section presents a brief overview of the relevant concepts that underpin the proposed injection approach.

A. Configuration Frame Address

The configuration memory is organized into frames, which serve as the smallest addressable unit. Each frame address indicates the level of granularity, specifying the block, row and column within the FPGA fabric to which the configuration bit belongs. Table I presents an illustration of the frame address format. Typically, the block indicates if the bit represents the configurable logic block (CLB) or the BRAM content, the row indicates the clock regions and the column represents the type of the resource (CLB, BRAM, DSP, clock or other I/Os). While the specific organization details may vary across FPGA families, the fundamental concept of frames remains consistent.

B. Fault Injection Campaign

Fault injection campaign can be broadly classified into three categories: radiation-, emulation- or simulation-based. In this work, we employ the commonly used emulation-based fault injection owing to its higher controllability, observability and cost-effectiveness. In emulation-based injection, faults are injected in the configuration memory either in a random or sequential order. Random campaign involves exhaustive injection of the memory at random frame addresses whereas in sequential injection, the campaign is planned offline and faults are injected in a systematic predefined order based on the distribution of the configuration bits.

C. Fault Model

We employ the widely used *bit-flip* model to simulate SEU in the configuration memory. According to this fault model, an upset results in a memory bit being set to either a '0' or '1' value until it is rewritten. In our experiments, we invert a single memory bit at a time, ensuring the memory is cleared of any existing faults prior to each injection.

D. System Failure

We monitored the serial interface for errors and identified the following failure modes: uncorrectable errors like CRC error and multi-bit errors, system timeout, data corruption, communication errors, and mitigation failure which warranted a system reconfiguration. These modes are consolidated under a common *Failure type* and the corresponding bits constitute the group of *Critical bits*.

E. Related Work

As efforts to optimise fault injection methodology continues to grow, recent work have primarily focused on pruning the volume of configuration bits to reduce the campaign time. An important development in this regard is Xilinx's Essential Bit technology that has been leveraged by numerous work [2], [5], [9]–[11] to obtain reliability estimates while minimising the number of injections. Aranda et al. [2] deconstruct the FPGA layout and approximate a linear relationship between the essential bits and the slice co-ordinates of the Pblocks which is then utilised by their tool ACME to direct injections at specific regions of the chip. The opensource tool BAFFI [9] can inject faults into fine-grained components like the LUTs and registers by mapping the essential bits with the hierarchical netlist using bitstream analysis. To reduce the set of essential bits, Tuzov et al. [3] map the combinational logic of the design to their corresponding configuration bit through a combination of static and dynamic analysis of LUTs. They conclude that while static analysis is worth the cost of reverse engineering, the dynamic analysis could result in significant overhead depending on the workload.

III. Monte Carlo Tree Search

Monte Carlo Tree Search (MCTS) is a best-first search algorithm commonly used in decision-making problems with large search spaces and incomplete information. The core principle of MCTS revolves around constructing a search tree based on the exploration and exploitation of the samples in the search space. The algorithm comprises four phases namely Selection, Expansion, Roll-out simulation, and Backpropagation as shown in Figure 1. The algorithm starts by constructing a search tree, where each node represents a state of the problem. The tree expands by exploring the search space iteratively. At each iteration, MCTS selects a node based on a selection strategy, typically balancing exploration (visiting unexplored nodes) and exploitation (choosing nodes with promising outcomes). Once

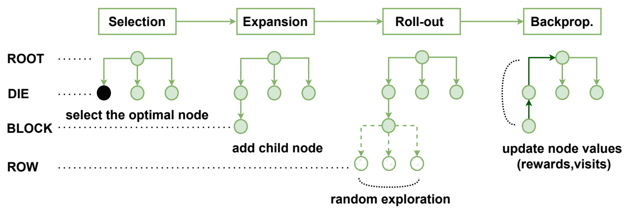


Fig. 1: Construction of Monte Carlo Tree

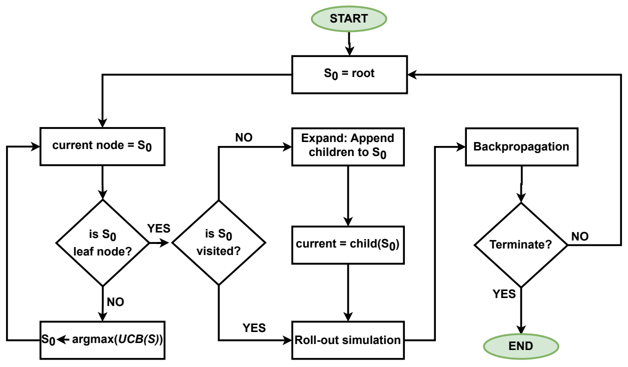


Fig. 2: MCTS workflow for fault injection

a node is selected, the algorithm performs a simulation from that node by making random moves until a terminal state or a predefined depth is reached. The outcome of the simulation is then backpropagated upto the root node, updating the statistics (visits, accumulated rewards) of all the visited nodes. Iterating through these four phases, the algorithm gradually converges towards identifying the most promising actions.

IV. Fault Injection using MCTS

A. Tree Structure

To adapt MCTS for fault injection, we model the tree to represent the spatial hierarchy of the FPGA fabric as defined by the frame address. The hierarchy consists of 4 levels namely die, block, row and column as explained in Section II-A and each node level in the tree is assigned to one of the hierarchical units as illustrated in Figure 1. The root node corresponds to the entire FPGA fabric and it branches out to the children nodes represented by the die address, which in turn branches down to block nodes. Each block node has children representing different rows, and the row nodes further branch out to children representing the columns.

B. Workflow

The MCTS workflow for guiding fault injection is illustrated in Figure 2. The tree search starts from the root node and the tree is traversed down to the leaf node by selecting the nodes with the highest *Upper Confidence Bound* (UCB) [12],

$$UCB = \bar{x} + c\sqrt{\frac{\log(V_p)}{V_n}} \tag{1}$$

where $\bar{x}$ is the mean node value which is the average reward obtained by the node, V_p is the number of visits of the parent node and V_n number of visits of the current node. The exploration coefficient c strikes a balance between the first term $\bar{x}$, promoting the exploitation of higher-reward choices, and the second term, fostering exploration of less-visited regions. When the algorithm reaches the optimal leaf node in the tree, the memory subregion is explored by initiating the roll-out simulation phase (Algorithm 1). During this phase, fault is injected into a random bit in the subregion and the status of the FPGA is analysed by a *Monitor* module. If the injected fault causes failure, the bit is considered critical and the corresponding leaf node is rewarded. The simulation phase is terminated after n injections and the total reward is backpropagated up the root node to guide the subsequent iterations.

Algorithm 1: Roll-out simulation phase

```
Input : No. of simulations $n$, Optimal leaf node $\mathcal{L}_l$,
        Reward function $\mathcal{R}$
        List of Essential bit addresses $A_{eb}$
   ▷ find optimal subregion based on UCB
1  $\mathcal{S}_l \leftarrow \{\mathcal{L}_l.die, \mathcal{L}_l.block, \mathcal{L}_l.row, \mathcal{L}_l.col\}$
   ▷ sampling of subregion
2  for $i \leftarrow 0$ to $n$ do
3      $\mathcal{M} \leftarrow$ random $A_{eb}$ belonging to $\mathcal{S}_l$
4      $InjectFault(\mathcal{M})$
5      $error \leftarrow Monitor()$
6      if $error = Critical$ then
7          $\mathcal{L}_l.reward \leftarrow \mathcal{R}$
8          $\mathcal{M}.critical \leftarrow True$
9      end
10     $Clear\_ConfigMem(\mathcal{M})$
11     $A_{eb}.delete(\mathcal{M})$
12 end
```

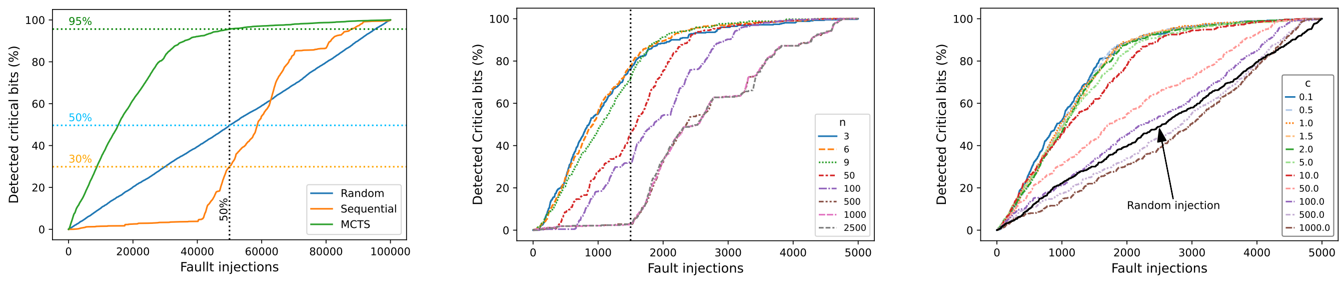
V. Evaluation

A. Experimental set-up

Our experimental setup comprises a Xilinx Ultrascale+ ZCU104 FPGA test platform and an Intel Xeon-based host PC connected via UART interface. An outline of the set-up is shown in Figure 4. The host PC functions as the supervisor for the fault injection campaign and consists of three components: the monitor, the MCTS module, and an address suite. The monitor distinguishes various errors, classifying faults as critical or noncritical based on its severity. This information is utilised by the MCTS module to determine the next location for fault injection, with the address suite providing frame addresses of the configuration bit. Fault injection is facilitated by the Xilinx Soft Error Mitigation (SEM) IP [13] which provides read and write access to the configuration registers through the internal configuration access port (ICAP). To manage the scale of the fault injection experiment, we selected 100,000 random essential bits from the design. These bits were extracted by following the guidelines detailed in [13].

B. Performance

To investigate the effectiveness of MCTS in optimising the injection campaign, we performed a comparison with the



(a) Coverage of critical bits ($n = 3, c = 0.5$)

(b) Impact of Roll-out simulations n ($c = 0.5$)

(c) Impact of Exploration coefficient c ($n = 3$)

Fig. 3: Evaluation of MCTS-based fault injection: Comparison with the traditional methods (a); Impact of Roll-out simulations (b) and Exploration coefficient (c) on MCTS detection performance

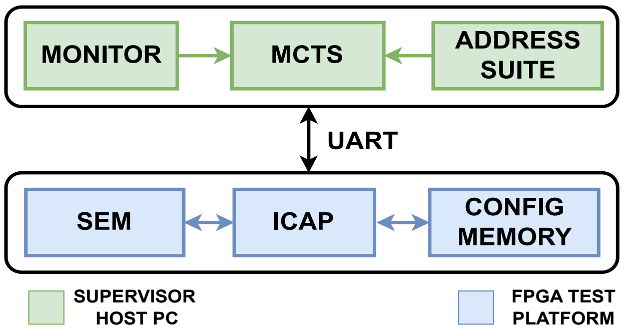


Fig. 4: Fault injection experiment set-up

random and sequential injection campaign explained in Section II-B. The traditional random campaign served as the baseline for comparison, while the sequential campaign allowed us to evaluate whether the MCTS approach could detect spatial relationships among the critical bits. To evaluate the quality of the injection campaign, we utilized the metrics of injection sensitivity and injection speedup as defined in (2),

$$Sensitivity = \frac{C_n}{j} \times 100 \qquad Speedup = \frac{j}{j_r} \tag{2}$$

where C_n is the detected critical bits, j is the number of injections required to detect the critical bits and j_r is the number of injections required by the random campaign to identify the same amount of critical bits.

Through exhaustive injection, a total of 9255 critical bits were identified in the set of 100k essential bits. The sensitivity and speedup of different fault campaigns are reported in Table II. The MCTS injection campaign achieved a 99% critical bit detection rate using 18% fewer injections than the random campaign. Notably, 95% of these critical bits were identified in less than half the number of random injections while the sequential campaign could detect just over 30% of the bits (marked by the dotted vertical line in Figure 3a). The MCTS campaign exhibits an average sensitivity of 23.5%, more than a 2-fold increase compared to the random and sequential campaigns with sensitivities of 9.3% and 8.3%, respectively. This implies that on an average MCTS requires twice as lesser injections to detect critical bits enabling faster turnaround times in the design and testing phase which is highly beneficial for validating large-scale FPGA designs. Though the sensitivity shows a declining trend as fewer critical bits remain to be detected, the MCTS approach still maintains a higher sensitivity compared to the random campaign. When less than 1% of critical bits remain to be detected, the MCTS performance becomes comparable to the random campaign. It could be attributed to either the absence of any relationship among the remaining bits or the limited availability of critical bits for learning. As the MCTS approach relies on sampling to approximate the behavior of the system, if there are insufficient critical bits available in the exploration phase, the sampled subset may not be adequate to derive any insights. Despite the diminishing availability of learning data, the MCTS approach still demonstrates superior injection performance compared to both random and sequential injections.

Figure 3a depicts the growth of detected critical bits for the three injection campaigns. Considering the steeper detection line of the sequential injection in relation to the random campaign, it can be deduced that there exists spatial relationships among certain critical bits within the same block/row/column. In the 5%-80% range of critical bit detection, the MCTS approach and the sequential campaign exhibit similar detection trajectories, indicating that the MCTS approach is able to exploit this correlation among the critical bits. Importantly, it eliminates the initial detection delay of 40k injections encountered in the sequential campaign, highlighting the advantages of exploration and exploitation strategies over offline campaign planning.

Overhead: The injection infrastructure used in our experiments (without MCTS) requires an average 3.3 seconds to inject fault in the configuration memory. This includes the time required (i) to access the memory through ICAP, (ii) injection sequence of the SEM controller (iii) analysis time of the *Monitor* module, (iv) memory reconfiguration time and (v) UART communication time between the FPGA and the host PC. Integrating the MCTS module in this infrastructure introduces an average overhead of 0.04% as corroborated by the findings in the Table III which reports the time required for the different MCTS phases. The overhead is dominated

TABLE II: Injection performance of MCTS, Sequential and Random campaign

Detected critical bits	MCTS			Sequential			Random		
	Injections ($j1$)	Speedup($j3/j1$)	Sensitivity(%)	Injections ($j2$)	Speedup($j3/j2$)	Sensitivity(%)	Injections ($j3$)	Speedup	Sensitivity(%)
15%	4237	3.6	32.8	44998	0.34	3.1	15279	1	9.1
30%	8761	3.4	31.7	50077	0.5	5.5	29796	1	9.3
45%	13804	3.2	30.2	57439	0.7	7.2	44504	1	9.4
60%	19276	3.0	28.8	62484	0.9	8.8	59202	1	9.4
75%	25777	2.9	26.9	66206	1.1	10.5	74961	1	9.3
90%	36265	2.4	22.9	81466	1.1	10.2	89982	1	9.3
95%	48679	1.9	18.1	87253	1.1	10.1	94761	1	9.3
99%	81188	1.2	11.3	91630	1.1	9.9	98894	1	9.3
100%	99551	1.0	9.3	99999	0.9	9.2	99993	1	9.3

TABLE III: MCTS overhead for a single injection cycle

	Selection + Expansion (μs)	Simulation (ms)	Backpropagation (μs)
Avg	35.04	1.4	6.4
Max	152.5	6.0	7.3
Min	5.5	0.5	1.6

by the simulation phase and the worst case is around 0.18% which occurs when the simulation paths are depleted in a node (explained further in Section V-C1).

Considering the MCTS campaign could identify a large majority of the critical bits with just half the number of random injections, what would typically require 87 hours in the random campaign (injection rate =$0.3\,\mathrm{bit\,s^{-1}}$) to identify at least 95% critical bits is now reduced to a mere 44 hours using the MCTS approach - a substantial improvement for applications that need faster design cycles. In comparison to the reverse engineering approach of Tuzov et al. [3] where the essential bits was reduced by upto 19%, accelerating the injection campaign by 1.16 times, the proposed approach can detect 99% of the critical bits at the same rate (19% reduction, 1.2 speedup) indicating that there is potential to improve injection methodologies without resorting to reverse engineering.

C. *Impact of Tuning parameters*

The tuning parameters namely the exploration coefficient c and the number of simulations n in the roll-out phase play an important role in injection performance and strategy discovery. In order to study the impact of these parameters, we injected fault in 5000 random essential bits uncovering a total of 422 critical bits. The findings of this experiment are reported in Figure 3b and 3c.

1) Roll-out simulation n: Too few simulations in the roll-out phase limits strategy discovery while too many inhibits the exploitation of the learned strategy. Moreover, conducting fewer simulations in each phase warrants more iterations of the tree search causing excessive overhead. Although this overhead is relatively small, conducting limited explorations in this phase can lead to wasted effort if no useful insights can be derived from them. An additional consideration in our case is that the critical bits are not uniformly distributed across the fabric. This means that, different leaf nodes have varying number of critical bits and consequently differing number of simulation paths. When n exceeds the available paths for a leaf node, an exception is raised and the exploration is terminated. Although subsequent exploration of such node is limited by the UCB policy by degrading the node value, complete elimination of revisits to such node requires an adaptive reward function that can balance the node's potential with its path depletion. Currently we use a fixed reward, and defining such a dynamic reward function remains an area for future investigation. The contrast in the detection rate for higher n can be observed by comparing $n = 3$ and $n = 1000$ in Figure 3b. In the former case over 75% of the critical bits are detected in less than 1500 injections (marked by the vertical line) whereas for the higher n, the algorithm is still in its learning phase having detected only 4% of the bits. Though the detection rate (slope of the detection curve) of higher n follows that of optimal n values after the initial learning offset, it is recommended to limit $n < 10$.

2) Exploration coefficient c: The selection of the exploration coefficient is influenced by the volume of the configuration bits and the availability of domain-specific knowledge. In the absence of such knowledge, a higher exploration coefficient might be beneficial. Conversely, scenarios involving a smaller fault space may suffice with a lower coefficient value. In our case, we found that a coefficient value of $c < 1$ provided optimal results. Figure 3c illustrates the effect of the coefficient on detecting critical bits, revealing a decline in detection performance for values $c > 2$ and an inflection point around $c = 10$ beyond which the detection trajectory starts resembling that of random campaign. At the juncture $c > 10$, the exploration phase dominates the search process, diminishing the distinction between high and low potential nodes, and thereby impeding the exploitation process.

D. *Comparison to Statistical Fault Injection*

Statistical Fault Injection (SFI) is used to reduce the number of injections by determining a representative sample of the fault space. Based on the framework proposed by Leveugle et al. [14], the sample size s for a fault injection campaign is determined using (3),

$$s = \frac{N}{1 + e^2 \times \frac{(N-1)}{z^2 \times p \times (1-p)}} \tag{3}$$

where N is the population size, e is the margin of error, p is the proportion of the population having a given characteristic (pro-

TABLE IV: Comparison of detected critical bits (C_n) in the samples (s) generated from SFI ($e = 1\%, N = 100k$)

	z	1.64 (90% conf.)	1.96 (95% conf.)	2.57 (99% conf.)	3.29 (99.9% conf.)
	s	6301	8763	14173	21298
Classical SFI	C_n	605	848	1345	1968
MCTS		1972	2756	4270	5970
	s	1068	1515	2578	4151
Dynamic SFI	C_n	108	147	230	400
MCTS		404	609	951	1330

portion of critical bits in our case) with a confidence interval given by the $z - score$.

We examined the efficacy of SFI and MCTS across four different confidence intervals [90%, 95%, 99%, and 99.9%] for an error margin of 1%, considering both the classical [14] and the dynamic [15] SFI methodology. The results of this experiment are outlined in Table IV. In dynamic SFI, the p value undergoes iterative updates as opposed to the fixed value of the classical method (typically $p = 0.5$). Though the dynamic approach yields a smaller sample size, the random selection of samples results in the injection sensitivity remaining the same as the classical SFI ($\approx 10\%$). In contrast, by prioritising the selection of samples that exhibit a higher likelihood of being critical, MCTS demonstrates 30% and 37% average sensitivity in the sample size generated by the classical and dynamic SFI respectively. The key difference in their approach is that SFI defines *how many* faults must be injected whereas MCTS focuses on *where* to inject faults. This distinction means that, while both approaches execute the same number of injections, MCTS is able to identify more critical bits owing to its exploration-exploitation phase. Our findings in Table IV underscores this difference, with MCTS consistently detecting at least 3X more critical bits than SFI across all confidence intervals.

VI. Conclusion

We proposed a novel fault injection methodology to identify critical bits in the FPGA configuration memory while minimising the number of injections. Our approach, based on Monte Carlo Tree Search (MCTS), effectively leverages spatial correlations to identify critical bits using 50 % fewer injections compared to traditional techniques. Our evaluations show that MCTS offers speedup akin to reverse engineering-based approach, but without the associated complexity and engineering costs associated with reversing the layout. Moreover, as there are no interventions in the standard design flow, the implementation of the proposed methodology is quite straightforward. Additionally, we demonstrate that our approach detects 3X more critical bits than the statistical fault injection scheme, offering an efficient strategy for assessing FPGA reliability in hazardous radiation environments.

References

[1] AMD. "Device Reliability Report." (2023), [Online]. Available: https://docs.xilinx.com/r/en-US/ug116/Failure-Rate-Summary.

[2] L. A. Aranda *et al.*, "ACME: A Tool to Improve Configuration Memory Fault Injection in SRAM-based FPGAs," *IEEE Access*, vol. 7, pp. 128 153–128 161, 2019.

[3] I. Tuzov *et al.*, "Reversing FPGA architectures for Speeding up Fault Injection: Does it pay?" In *2022 18th European Dependable Computing Conference (EDCC)*, 2022, pp. 81–88.

[4] C. Fibich *et al.*, "Device-and Temperature Dependency of Systematic Fault Injection Results in Artix-7 and Ice40 FPGAs," in *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, IEEE, 2021, pp. 1600–1605.

[5] S. Di Carlo *et al.*, "A Fault Injection Methodology and Infrastructure For Fast Single Event Upsets Emulation on Xilinx SRAM-based FPGAs," in *2014 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)*, 2014, pp. 159–164.

[6] A. Ruospo *et al.*, "Assessing Convolutional Neural Networks Reliability through Statistical Fault Injections," in *2023 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, IEEE, 2023, pp. 1–6.

[7] C. Thurlow *et al.*, "TURTLE: A Low-cost Fault Injection Platform for SRAM-based FPGAs," in *2019 International Conference on ReConFigurable Computing and FPGAs (ReConFig)*, IEEE, 2019, pp. 1–8.

[8] T. Tanaka *et al.*, "Impact of Neutron-Induced SEU in FPGA CRAM on Image-Based Lane Tracking for Autonomous Driving: From Bit Upset to SEFI and Erroneous Behavior," *IEEE Transactions on Nuclear Science*, vol. 69, no. 1, pp. 35–42, 2022.

[9] I. Tuzov *et al.*, "BAFFI: A Bit-accurate Fault Injector for Improved Dependability Assessment of FPGA Prototypes," in *2023 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, IEEE, 2023, pp. 1–6.

[10] A. Ramos *et al.*, "Characterizing a RISC-V SRAM-based FPGA Implementation against Single Event Upsets using Fault Injection," *Microelectronics Reliability*, vol. 78, pp. 205–211, 2017.

[11] W. Yang *et al.*, "Reliability Assessment on 16 nm Ultrascale+ MPSOC using Fault Injection and Fault Tree Analysis," *Microelectronics Reliability*, vol. 120, p. 114 122, 2021.

[12] L. Kocsis and C. Szepesvári, "Bandit based Monte-Carlo Planning," in *European conference on machine learning*, Springer, 2006, pp. 282–293.

[13] AMD, "UltraScale Architecture Soft Error Mitigation Controller v3.1 LogiCORE IP Product Guide," 2021.

[14] R. Leveugle *et al.*, "Statistical fault injection: Quantified error and Confidence," in *2009 Design, Automation & Test in Europe Conference & Exhibition*, IEEE, 2009, pp. 502–506.

[15] I. Tuzov *et al.*, "Accurate Robustness Assessment of HDL Models through Iterative Statistical Fault Injection," in *2018 14th European Dependable Computing Conference (EDCC)*, IEEE, 2018, pp. 1–8.