DeBaTE-FI: A Debugger-Based Fault Injector Infrastructure for IoT Soft Error Reliability Assessment

Alex Hanneman*, Jonas Gava†, Vitor Bandeira†, Rafael Garibotti‡, Ricardo Reis†, Luciano Ost*

* Wolfson School, Loughborough University, United Kingdom – {*a.d.hanneman, l.ost*}*@lboro.ac.uk*

† PGMicro/PPGC/UFRGS, Universidade Federal do Rio Grande do Sul, Porto Alegre, Brazil
{*jonas.gava, vvbandeira, reis*}*@inf.ufrgs.br*

‡ Vector Trading, Chicago, US – *rgaribotti@vectrading.com*

***Abstract*—This work presents DeBaTE-FI as a scalable tool to aid in assessing the susceptibility of embedded IoT systems to soft errors. DeBaTE-FI includes a method that allows precise targeting of the injection points, equivalent to common simulation methods, while providing significant improvement in the accuracy of the results, particularly concerning the propagation of errors in the control flow. A comparison against an instruction-based fault injection method is made, showing a doubling in the estimated rate of crashes/timeout outcomes using DeBaTE-FI when compared to an identical simulated campaign. We present improvements in the performance of the platform following the simulation campaigns and evaluate the system's scalability.**

***Index Terms*—Soft Error, Fault Injection, Low-Power Processors, Reliability, Resource-Constrained Devices.**

I. Introduction

The increasing adoption of resource-constrained systems into areas previously dominated by high-performance systems, such as the deployment of Deep Neural Networks (DNNs) driven by concepts such as Edge IoT is resulting in a desire to entwine sensors and the analysis of their outputs more closely. This migration of use cases poses a number of fundamental challenges; the applications deployed to the resource-constrained systems must be heavily adapted and optimised for their host system. The adaptation is already a complex task, with much literature discussing techniques that can be used to port high-performance neural networks to embedded systems that typically have severe restrictions on power consumption, memory availability and hardware to efficiently perform certain operations, such as floating-point arithmetic. In addition to these demands, the resource-constrained system may be deployed into a safety-critical system, where protection against the effects of Single Event Upsets (SEUs), also known as soft errors, may need to be provided. In order to minimise the effect of the SEU mitigation strategy on the performance of the system, an accurate assessment of the susceptibility of the system to SEUs must be made; such an assessment can then be used to adjust the mitigation strategy to strike an acceptable balance between reliability and performance.

To achieve such a goal, reliability engineers must be able to narrow down less suitable design alternatives, as illustrated in Figure 1, using progressively more accurate techniques before ultimately exposing the most promising system configuration to radiation tests, which are highly costly. In this context, authors are using statistical models [1], fault injection (FI) considering either virtual platforms [2], or low-level simulation [3] or board-oriented approaches (see works in Table I) to assess the influence of soft errors on DNN models.

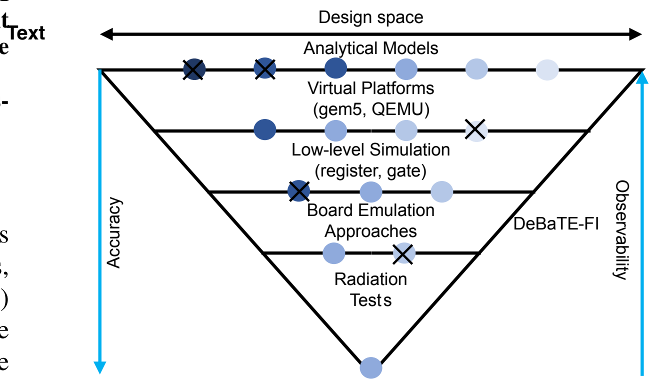


Fig. 1. Soft error resilience assessment approaches and DebaTE-FI's location.

While high-level approaches provide fast design space exploration and flexibility, fault injection campaigns conducted using development boards are time-consuming but likely to provide more accurate results since they better represent the masking effect of micro-architectural layers and inter-layer boundaries [4]–[6]. Due to the vast design space alternatives of DNNs (e.g., distinct quantisation, precision setups), we claim that early cross-layer soft error reliability analysis using FI simulation and board-oriented approaches must be jointly adopted before going down to actual radiation experiments.

In this regard, the main *contribution* of this paper is twofold: first, the development of DeBaTE-FI, a scalable debugger-based FI approach that allows fault injection campaigns to be conducted using multiple development boards operating in parallel; second, analysis of the soft error consistency of a just-in-time fault injection simulator against fault injection campaigns conducted with the proposed board debugger-based FI approach, considering the execution of a real CNN inference model in a microprocessor presented in commercial smart sensor systems [7] and drone platforms [8] [9].

This work was partially funded by: UK EPSRC (EP/R513088/1)

TABLE I
RELATED WORKS IN BOARD-ORIENTED FAULT INJECTION EMULATION APPROACHES

Work	Injection Method	Target Devices	Scalability	Accessibility	Targeting
Mosdorf et al [10] (2011)	Debugger-based emulation	Arm7 architecture	Single	Software visible registers	Random
Fidalgo et al [11] (2011)	FPGA-based emulation	RTL Descriptions	Single	Software visible registers	Targetable
Trindade et al [12] (2020)	Debugger-based emulation	Arm Cortex-M4	Single	Software visible registers	Random
Oliveira et al. [13] (2017)	Debugger-based emulation	Intel Xeon Phi	Single	Software visible registers	Random
Maistri et al. [14] (2022)	FPGA-based emulation	RTL descriptions	Single	Gate-level, set at synthesis	Cycle targetable
This work (2023)	**Debugger-based emulation**	**OpenOCD supported MCUs**	**Multi-board**	**Software visible registers**	**Predetermined targeting**

II. RELATED WORKS IN FAULT INJECTION

Though static analysis techniques exist for assessing SEU susceptibility of neural network models [1], the most adopted approach is fault injection (FI) simulation. Early FI simulation is key to understand the impact of soft errors on DNN models and is the basis for exploring and deploying fault-tolerant models [2]. In general, fault injection techniques can be divided into three branches, often performed at different stages of a system's development: 1) utilising high-level simulators [6], [15]–[17] or commercial low-level RTL/gate simulators; 2) emulating the system architecture, usually using a Field Programmable Gate Array (FPGA) to implement the target neural network solution, where bit flips can be inserted into the model [4], [14], [18]; 3) using debug engine or tools, such as GDB, to accomplish the fault injection on physical integrated circuits either available in development or FPGA boards [12].

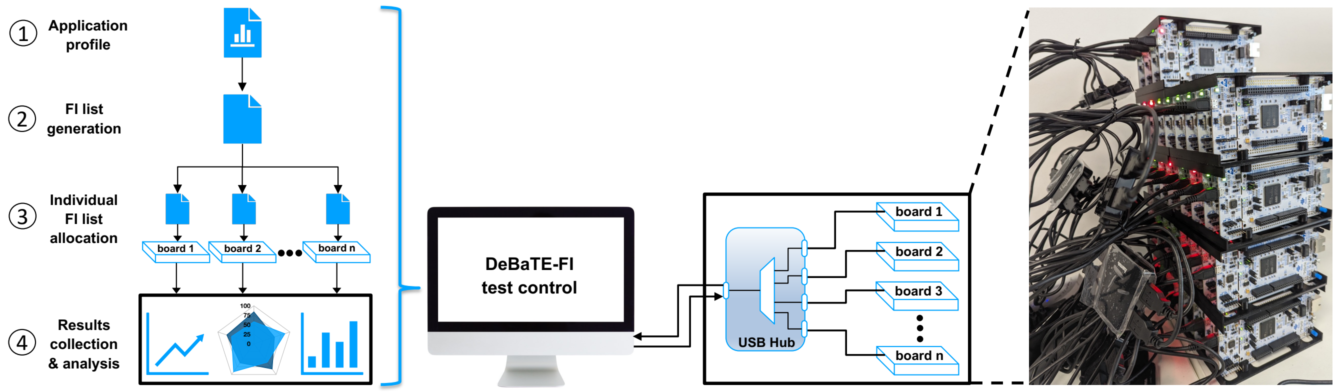
Simulation methods run the target DNN models using a model of the processor. Depending on the level of abstraction, the accuracy and time required to run the simulation can vary significantly. At one end of the scale, RTL simulations take significant amounts of time to run [4], [6]; however, offer high levels of accuracy as they simulate the full system; at the other lie instruction accurate simulators which can offer high performance due to the high level of abstraction used. Cycle-accurate simulators sit in a middle ground but still operate with a high degree of abstraction. FI simulations can be performed early in the design process; however, when using commercial off-the-shelf (COTS) hardware, the choice of models is limited to what the manufacturer supplies. Emulation in FPGAs also requires the RTL description of the target DNN model in a format where changes can be made to the design to allow the addition of hardware to inject the faults [4], [14], [18].

Table I provides examples of the third branch, which includes the proposed DeBaTE-FI. Except for [14], remaining works differ from FPGA-based emulation in that they utilise COTS hardware to provide the micro-architectural model, while this restricts the location where faults can be injected they do not require the full RTL description, which are rarely available to users. The underlying approaches can be used to gain more accurate insights of the influence of soft errors on software-based DNN inference models' behaviour before exposing them to real radiation tests [6], [19]–[21]. Radiation trials require specialist facilities to generate the high radiation environment; these are both scarce and costly and so using these facilities for iterative development may be impractical.

Different from reviewed works, the proposed DeBaTE-FI has been validated with multiple boards, and it includes a function lifespan FI technique, which reduces the FI spectrum by limiting the insertion time to those small intervals where the target DNN function/layer is active. This is also the first work to assess soft error analysis confidence of a JIT-based fault injection simulator w.r.t. a debugger-based approach, considering a real CNN inference model as case-study.

III. DEBUGGER BASED FAULT INJECTION METHOD

Figure 2 shows the DeBaTE-FI soft error assessment flow (a) and the hardware architecture (b). A minimal level of modification is needed to adapt a program for use with DeBaTE-FI; the most significant is the addition of two functions used to identify the start and end point of the injection zone, FIM_START() and FIM_EXIT(). These function names must be explicitly specified to work with the DeBaTE-FI program. Additionally, a version of the application must be made which is compatible with a simulation tool to extract the instruction trace information of each function call, shown as **(1)** application profile in Figure 2(a). This process typically involves removing the initialisation functions and using MCU-specific peripherals. Note that the method of providing the program with data may also require changing and the program must remain identical between the FIM_START() and FIM_EXIT() calls. The simulation version of the program is profiled considering the region from the call of FIM_START() to FIM_EXIT(). If the test region follows a broadly deterministic execution profile, then the profiling need only be performed once. The information obtained is then used to generate an appropriate fault list **(2)**, which shall cover the majority of



(a) DeBaTE-FI Soft Error Assessment Flow

(b) DeBaTE-FI Hardware Architecture

Fig. 2. DeBaTE-FI Architecture.

possible faults on the target system configuration. After that, the FI list is further divided into multiple individual fault lists, which are then allocated to each available board **(3)**. Then, the binaries are executed in parallel, and the test control uses debugging capabilities to emulate the occurrence of bit-flips into the memory or processor registers. Finally **(4)**, relevant information (e.g, application output, core dump file, etc) is collected and scripts are used to generate graphics/reports to further assist software engineers in the results' analysis.

The debugging capabilities of many embedded systems prohibit both a direct approach to a planned injection target or the possibility of determining the randomness of the injection locations after the fact. The former approach would require the ability to not only set breakpoints but to allow set breakpoints to be skipped a pre-determined number of times by the hardware in order that the processor halts only on the desired function call; the latter method would require the timing of the debugging chain to be tightly controlled to achieve a random distribution; however, the primary issue is the lack of ability to determine the exact location in the overall program execution that makes a determination of the actual randomness of the sampling difficult if not impossible to achieve.

Instead, the debugger software must navigate to a predetermined injection point; our approach performs this navigation in two stages. First, the system guides the Program Under Test (PUT) onto the specific call of the targeted function. Once the intended function call is reached, a breakpoint is then set on the particular instruction being targeted; this breakpoint is passed over until the instruction is called a set number of times; once this occurs, the fault is injected at a specified point in one of the General Purpose (GP) registers; (r0-r12), Link Register (LR); Stack Pointer (SP); Program Counter (PC); or Floating Point (FP) registers (d0-d15). The targeting algorithm is described in Figure 3.

The DeBaTE-FI test control is readily scalable to the number of boards available; once an injection scenario has been allocated to a device, there is no requirement for interaction with the centralised test control or other test instances until the injection process completes and a new scenario is requested.

```
targFunction ← looked-up function address
targOffset ← instruction distance from function start
targCall ← targeted function call
targRegister ← targeted register
targBit ← targeted bit of register
targPCCount ← targeted instance of instruction
lineAddress ← targFunction + targOffset
RUNTOBREAKPOINT(targFunction, targCall)
RUNTOBREAKPOINT(lineAddress, targPCCount)
targRegisterValue ← value of targRegister
targRegisterValue XOR bit mask of targBit
set targeted register to targRegister

function RUNTOBREAKPOINT(address, skipCalls)
    set breakpoint at address
    for skipCalls to 0 do
        resume program
        while program status ≠ HALTED do
            wait
        end while
    end for
    remove breakpoint
    return
end function
```

Fig. 3. DeBaTE-FI Targeting algorithm

IV. Comparison versus Simulation Fault Injection

In this work, we compare the results of fault injection campaigns performed using the presented debugger FI technique and an open-source simulation-based fault injection tool [15].

A. *Methodology*

A test application was created using based on the reference CMSIS-NN implementation of the CIFAR-10 image classification network [22]; this implementation of the neural network was adapted for deployment on an STM32 Nucleo L4R5 development board based on an Arm Cortex-M4 MCU, this implementation allowed for the image to be sent to the board, and the sending of the output classifications over a serial connection to the test controller (a Raspberry Pi 4B). An alternate version of the program was required for use in the simulation tool, which removed from the compiled binary the device-specific configuration (clock and peripheral initialisation), and passed the input data via a command line argument rather than over the serial version; both versions of the program were identical within the FI region bounded by FIM_START() and FIM_EXIT(). Using the gem5 virtual platform simulator, the simulation version of the program was profiled considering the fault injection region; from the data collected during profiling, a fault list of 1124 fault scenarios was generated; for initial testing, these were divided into three groups, two of 544 tests and one of 36, the group of 36 target only the stack pointer due to an error in generating the injection scenarios which omitted the stack pointer for the first two groups. For each scenario of the groups, a different input image from the CIFAR-10 dataset was used; the same set of 544 images was used for each group, so each image was used for at least two fault scenarios.

Each of the fault scenarios consisted of the following information: the symbol name that contains the instruction; the number of times to skip that symbol to reach the desired instance; the address offset of the target instruction from the top of the symbol; the number of times to skip that instruction before injecting the fault; the register to inject into; and the specific bit to inject into. The symbol name and offset were used due to the differences between the simulation and board versions of the program.

B. Results

Unlike the simulator, DeBaTE-FI is unable to determine from an examination of the processor alone whether the processor has suffered an unexpected termination; instead, an application crash is detected when a set timeout period elapses, given that the time-out period for the board test was set at a minimum of 10x the expected execution time, the cases classified as UT by the simulator are absorbed into the timeout classification for better comparison.

Figure 4 shows the summary comparison of the two methods, for both the majority of the injections result in a masked outcome; however, although the number of masked outcomes is smaller for the board results, this is somewhat unexpected as a more accurate model is typically expected to show a stronger inter-layer masking effect. The reduced number of masked outcomes for the board is potentially explained by a significant improvement in the modelling of the susceptibility of the architecture and micro-architecture layers to bit-flip induced time outs and crashes, shown by the more than 100% increase in the number of time outs recorded by the board tests over the simulations.

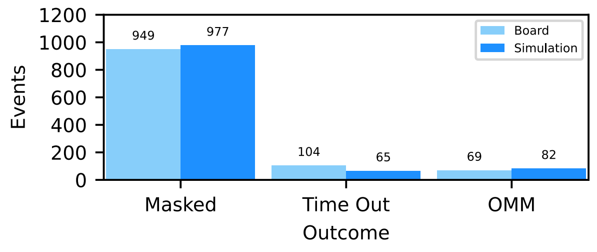


Fig. 4. Overall outcome comparison.

The simulation results do show a higher rate of OMM outcomes, this is partially attributable to the lower rate of timeouts; it is likely some of the faults that manifested as a timeout on the board resulted in an OMM in the simulation. This conclusion is supported by the rates of masking broken down by register in Figure 5, which shows that the number of masked outcomes between the board and simulation methods across the tested registers is fairly consistent, suggesting some transfer of outcomes from OMM to Timeouts.

Examining the breakdown of the number of timeout outcomes by the register causing them, Figure 6, supports the conclusion that the board simulation offers a significant improvement in assessing the susceptibility of the system to crashes and timeouts.

While the simulations and board experiments largely concur on the effect of injecting into the program counter (PC), the simulator predicts only three timeouts caused by injections into other registers, while DeBaTE-FI trials result in 41. Notably, after the PC, the register with the most timeout outcomes is r7. The r7 register, when compiling the program as a Thumb2

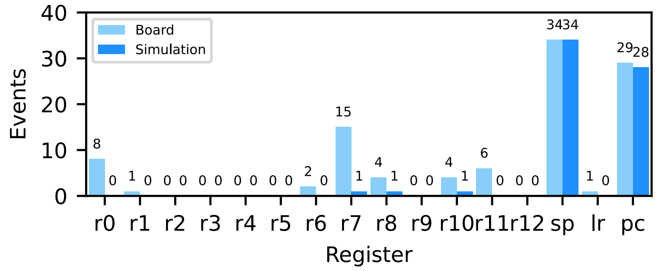


Fig. 6. Timeout outcome occurrence by the injected register.

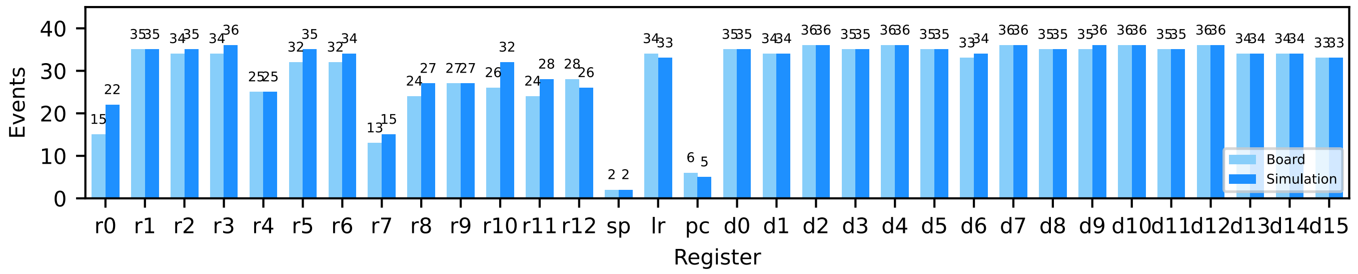


Fig. 5. Masked outcome occurrence by the injected register.

program, is utilised as the frame pointer, which holds the value of the stack pointer before the present function is called. A mechanism by which corruption of r7 results in a crash or fault is apparent from the DeBaTE-FI results but not from the simulation results. Likely as the frame pointer is used to set the stack pointer when the function returns, a corrupted frame pointer can easily lead to a corrupted stack pointer, in turn causing an attempt to access unexpected parts of the address space. Figure 7 indicates that both methods produce identical results for injections into the stack pointer itself.

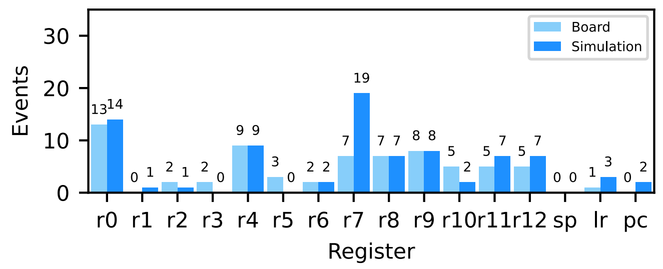


Fig. 7. Output Mismatch outcome occurrence by the injected register.

The differences between the susceptibility of the registers to OMMs between the DeBaTE-FI trials and the simulation results occur mainly where the DeBaTE-FI results indicated an excess of timeouts; other than those areas of disagreement; the two methods generally produce similar conclusions on the registers, particularly vulnerable to SEUs, as well as the overall rate of OMMs. Based on obtained results we can conclude that these errors are acceptable and are not a hindrance to evaluating soft error at early design phases.

The primary limitation of the DeBaTE-FI method over simulation approaches is the required runtime. To enable the accurate targeting of precise points in the application's execution, the test application must perform large amounts of stepping operations, as well as the setting and skipping of breakpoints; this presents significant performance problems. The tests conducted here required 19 days to perform 1124 test scenarios running in parallel on three development boards, however, further development has yielded significant improvements in this regard.

V. Scalability

One of the aims of the DeBaTE-FI platform is to provide a scalable solution; that is, the rate at which fault injection scenarios can be performed scales with the amount of target board connected. The CIFAR-10 campaign used three boards with a Raspberry Pi 4B as the test controller, with this configuration, the number of boards that could be connected to the test application was limited to approximately 8; after this point, it appears the USB controller is unable to connect further boards. While using additional test controllers would be possible, this presents an undesirable increase in the cost-effectiveness of the platform. Switching to the Rock 4C SBC for the test controller improved the number of boards that could be connected; this configuration was validated up to 30 boards (Table II).

TABLE II
Performance scaling with Rock 4C test controller.

Boards	t_{100}/s	t_{60}/s	$T_{100}s^{-1}b^{-1}$	$T_{60}s^{-1}b^{-1}$
10	24,890	13,994	$4.02 \cdot 10^{-4}$	$4.29 \cdot 10^{-4}$
20	21,471	11,550	$2.33 \cdot 10^{-4}$	$2.6 \cdot 10^{-4}$
30	21,437	10,718	$1.55 \cdot 10^{-4}$	$1.87 \cdot 10^{-4}$

A drone control algorithm was utilised as a benchmark application to assess the performance scaling of the DeBaTE-FI platform; as for the CIFAR-10 model, the application was profiled, and a fault list was generated with 100 injection scenarios. STM32F767ZI Nucleo development board were used for these tests, containing Arm Cortex M7 processors. From each test, two measurements are obtained t_{100} and t_{60}, the time taken to finish all 100 and 60 scenarios, respectively. t_{60} gives a better comparison of the performance with all boards running in parallel, as after completing 60 tests, some of the test cases will have nearly exhausted their input queues and reducing the benefits of the parallel operation. From t_{100} and t_{60} we derive metrics to compare the rate at which tests are completed, normalised for the number of boards used (b), T_{100} and T_{60} calculated as shown in Equations (1) and (2).

$$T_{100} = 100/t_{100}/b \tag{1}$$

$$T_{60} = 60/t_{60}/b \tag{2}$$

While the Rock 4C was able to connect 30 boards to the test application, the performance improvement expected did not occur. A reduction in total test time of an hour occurred moving from 10 to 20 boards, shown in Table II, yet this represents only a 14% decrease in execution time for a 100% increase in target boards. There is no significant change moving from 20 to 30 target boards; the increase in boards is compensated for by a decrease in both T_{100} and T_{60}. CPU utilisation on the Rock 4C showed a 100% load across all cores, suggesting a CPU bottleneck. Utilising a desktop PC resulted in an unexpected decrease in performance; however, switching to running the test control for each target board within a python Multiprocessing process, rather than a threading thread, resulted in significant performance improvements, shown in Table III.

TABLE III
Performance scaling with PC test controller, all tests used 36 target boards. Case 1 uses the python threading library; case 2 uses Multiprocessing Processes.

Case	T_{100}/s	T_{60}/s	$T_{100}s^{-1}b^{-1}$	$T_{60}s^{-1}b^{-1}$
1	22,985	9,589	$1.21 \cdot 10^{-4}$	$1.74 \cdot 10^{-4}$
2	747	311	$3.72 \cdot 10^{-3}$	$5.36 \cdot 10^{-3}$

Figure 8 shows the change in speedup and efficiency for an increasing quantity of boards from one to 35; these metrics are commonly used to assess the scalability of algorithms over multiple cores; and are used similarly here. The t_{60} times for the multi-board tests are used as the parallel execution times, while the t_{60} time of the single board run acts as

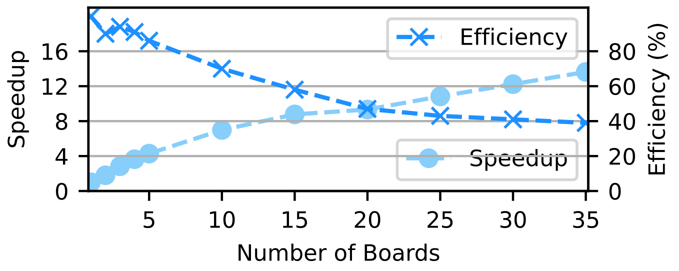


Fig. 8. Plot of the Speedup and Efficiency for varying numbers of boards

the serial execution time. The results in Figure 8 show that there is an initial somewhat sharp fall in the efficiency of the platform, falling from 86% to 47% between 5 and 20 boards, subsequently though the fall in efficiency stabilises dropping from 43% to 39% between 25 and 35 boards. The speedup from 2-4 boards remains fairly constant at around 90%; the much higher rate of decrease in efficiency above five boards is possibly due to the limited number of cores in the test controller system (4-core, 4-threads). The test programs' architecture is highly multithreaded, with much inter-process communication required between the OpenOCD session for each device and that device's test control process within the test program. Further investigation is needed to confirm this; however, at this stage, the DeBaTE-FI platform can perform comprehensive fault injection campaigns on many applications and algorithms targeted at IoT devices.

VI. Conclusion

This work has presented DeBaTE-FI as a tool to provide an embedded systems developer to assess the susceptibility of a system to the effects of SEUs; we have shown that DeBaTE-FI offers significant advantages over typical simulation methods, primarily in modelling vulnerability of the control flow aspects of the application. We have further demonstrated how the platform has been developed and tested to show the ability to scale in performance with the number of target boards available. Further development will aim to expand the range of systems where this method can be used, as well as to demonstrate the use of DeBaTE-FI to assess the effectiveness of software-based fault mitigation strategies. Future work also includes assessing the soft error consistency of DeBaTE-FI against results obtained from neutron radiation campaigns.

References

[1] H. Huang, X. Xue, C. Liu, Y. Wang, T. Luo, L. Cheng, H. Li, and X. Li, "Statistical Modeling of Soft Error Influence on Neural Networks," 2022. [Online]. Available: http://arxiv.org/abs/2210.05876

[2] G. Abich, J. Gava, R. Garibotti, R. Reis, and L. Ost, "Applying Lightweight Soft Error Mitigation Techniques to Embedded Mixed Precision Deep Neural Networks," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 68, no. 11, pp. 4772–4782, November 2021.

[3] S. Nema, J. Kirschner, D. Adak, S. Agarwal, B. Feinberg, A. F. Rodrigues, M. J. Marinella, and A. Awad, "Eris: Fault Injection and Tracking Framework for Reliability Analysis of Open-Source Hardware," in *International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2022, pp. 210–220.

[4] H. Cho, S. Mirkhani, C.-Y. Cher, J. A. Abraham, and S. Mitra, "Quantitative evaluation of soft error injection techniques for robust system design," in *ACM/IEEE Design Automation Conference (DAC)*, 2013, pp. 711–720.

[5] J. Wei, A. Thomas, G. Li, and K. Pattabiraman, "Quantifying the Accuracy of High-Level Fault Injection Techniques for Hardware Faults," in *IEEE/IFIP International Conference on Dependable Systems and Networks (DNS)*, 2014, pp. 375–382.

[6] A. Chatzidimitriou, P. Bodmann, G. Papadimitriou, D. Gizopoulos, and P. Rech, "Demystifying Soft Error Assessment Strategies on ARM CPUs: Microarchitectural Fault Injection vs. Neutron Beam Experiments," in *IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2019, pp. 26–38.

[7] STMicroelectronics, "STEVAL-STLKT01V1," 2023. [Online]. Available: https://www.st.com/en/evaluation-tools/steval-stlkt01v1.html

[8] OctavoSystems, "OSD32MP15x System-in-Package," 2023. [Online]. Available: https://octavosystems.com/octavo_products/osd32mp15x

[9] Arm, "Auterion's PX4," 2023. [Online]. Available: https://www.arm.com/company/success-library/made-possible/px4

[10] M. Mosdorf and J. Sosnowski, "Fault injection in embedded systems using GNU Debugger," *Measurement Automation Monitoring*, vol. 57, no. 8, pp. 825–827, January 2011.

[11] A. V. Fidalgo, M. G. Gericota, G. R. Alves, and J. M. Ferreira, "Real-time fault injection using enhanced on-chip debug infrastructures," *Microprocessors and Microsystems*, vol. 35, no. 4, pp. 441–452, June 2011.

[12] M. G. Trindade, R. P. Bastos, R. Garibotti, L. Ost, M. Letiche, and J. Beaucour, "Assessment of Machine Learning Algorithms for Near-Sensor Computing under Radiation Soft Errors," in *IEEE International Conference on Electronics, Circuits and Systems (ICECS)*, 2020, pp. 494–497.

[13] D. Oliveira, V. Frattin, P. Navaux, I. Koren, and P. Rech, "CAROL-FI: An Efficient Fault-Injection Tool for Vulnerability Evaluation of Modern HPC Parallel Accelerators," in *Computing Frontiers Conference (CF)*, 2017, pp. 295–298.

[14] P. Maistri and J. Po, "A Low-Cost Methodology for EM Fault Emulation on FPGA," in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2022, pp. 1185–1188.

[15] J. Gava, V. Bandeira, F. Rosa, R. Garibotti, R. Reis, and L. Ost, "SOFIA: An automated framework for early soft error assessment, identification, and mitigation," *Journal of Systems Architecture*, vol. 131, p. 102710, October 2022.

[16] G. Abich, J. Gava, R. Reis, and L. Ost, "Soft Error Reliability Assessment of Neural Networks on Resource-constrained IoT Devices," in *IEEE International Conference on Electronics, Circuits and Systems (ICECS)*, 2020, pp. 329–332.

[17] F. R. da Rosa, R. Garibotti, L. Ost, and R. Reis, "Using Machine Learning Techniques to Evaluate Multicore Soft Error Reliability," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 66, no. 6, pp. 2151–2164, June 2019.

[18] F. Libano, P. Rech, L. Tambara, J. Tonfat, and F. Kastensmidt, "On the Reliability of Linear Regression and Pattern Recognition Feedforward Artificial Neural Networks in FPGAs," *IEEE Transactions on Nuclear Science*, vol. 65, no. 1, pp. 288–295, January 2018.

[19] R. L. Rech Junior, S. Malde, C. Cazzaniga, M. Kastriotou, M. Letiche, C. Frost, and P. Rech, "High Energy and Thermal Neutron Sensitivity of Google Tensor Processing Units," *IEEE Transactions on Nuclear Science*, vol. 69, no. 3, pp. 567–575, March 2022.

[20] M. Peña-Fernández, A. Lindoso, L. Entrena, I. Lopes, and V. Pouget, "Microprocessor Error Diagnosis by Trace Monitoring under Laser Testing," *IEEE Transactions on Nuclear Science*, vol. 68, no. 8, pp. 1651–1659, August 2021.

[21] S. Blower, P. Rech, C. Cazzaniga, M. Kastriotou, and C. D. Frost, "Evaluating and Mitigating Neutrons Effects on COTS EdgeAI Accelerators," *IEEE Transactions on Nuclear Science*, vol. 68, no. 8, pp. 1719–1726, August 2021.

[22] L. Lai, N. Suda, and V. Chandra, "CMSIS-NN: Efficient Neural Network Kernels for Arm Cortex-M CPUs," 2018. [Online]. Available: http://arxiv.org/abs/1801.06601