Enhancing RISC-V Ecosystem for SEU-Resilient Inference on FPGA-Based Implementations

Giorgio Cora, Eleonora Vacca, Corrado De Sio, Sarah Azimi, Luca Sterpone
Politecnico di Torino, Italy
name.surname@polito.it

Abstract— Recently, platforms combining RISC-V processors with deep-learning accelerators are gaining traction even in safety-critical applications like avionics and space. However, ensuring both high performance and fault tolerance is mandatory. Through this paper, we propose an FPGA-based architecture that integrates a RISC-V processor with a systolic array accelerator, a fault detection, correction, and execution recovery mechanism. Faults in the accelerator's datapath are corrected using partial reconfiguration, allowing the processor to trigger reconfiguration when errors occur. This approach ensures execution continuity with minimal downtime by resuming inference from the last correct step, providing a reliable and high-performance solution.

I. INTRODUCTION

The increasing complexity of Deep Learning (DL) models has driven the need for high-performance platforms, with RISC-V gaining attention due to its open-source nature. Researchers enhance RISC-V-based solutions by extending the ISA [1] or integrating accelerators, particularly systolic arrays (SAs), for efficient matrix multiplication in neural networks [2]. In particular, SA-based architectures can optimize parallel computation, reduce memory access, and improve data reuse, making them ideal for deep neural networks (DNNs).

However, in safety-critical applications, computational efficiency alone is insufficient, and ensuring reliable inference execution is equally important. Existing fault detection methods for SA accelerators and DNN models do not fully address reconfigurable FPGA platforms, which are vulnerable to configuration memory (CRAM) errors, particularly in space and avionics systems.

Indeed, common fault scenarios in reconfigurable platforms often involve radiation-induced Single Event Upsets (SEUs) in CRAM, which lead to bit flips in memory cells. These bit flips alter the configuration status of programmable resources used to implement the target circuit, potentially changing the circuit's netlist. This alteration can result in system behavior anomalies, ultimately leading to system failure[3][4].

Traditional fault mitigation strategies typically involve introducing redundancy at the architectural level to ensure correct output in the presence of faults. However, this approach is not feasible for Deep Neural Network (DNN) accelerators, which already require significant computational resources. Moreover, such strategies do not correct errors, allowing them to accumulate over time. As a result, these methods are often combined with periodic memory scrubbing, which requires reloading the bitstream into the CRAM. This process can take several seconds, significantly reducing system availability. Other techniques, such as embedding the SEM-IP module [5] in the design, leverage logic resources to detect and correct SEUs. However, they fail to meet real-time constraints, as detection and correction can take seconds, and they cannot detect SEU patterns within the same CRAM word, which may mask the errors.

This work addresses these challenges by integrating partial reconfiguration with an efficient, low-resource fault detection mechanism to ensure the reliable execution of AI applications in space environments. This is achieved by coupling a RISC-V processor, which not only processes data for inference execution but also manages the reconfiguration routine, with an open-source AI accelerator, whose Instruction Set Architecture (ISA) is extended to support self-test capabilities. This combination enhances platform reliability without compromising performance.

II. STATE OF THE ART

The introduction of the RISC-V ISA has led to significant advancements in the development of efficient computing platforms for DNN applications. Various works have explored custom architectures and optimization strategies to enhance system performance. For instance, authors in [6] proposed a high-performance multithreaded convolutional library in terms of Floating-Point Operations Per Seconds (FLOPS), whose efficiency can be compared with other state-of-the-art processors, including ARM cores. Additionally, the open-source nature of RISC-V enables ISA extensions to efficiently execute custom instructions oriented to DNN inference[7]. Research includes coupling RISC-V processors with co-processors for DNN inference, improving DDR access, and accelerating GEMM operations via SIMD extensions. A detailed comparison is performed in [8], where authors evaluate the differences in terms of execution speedup and performances when co-processors are instantiated inside or outside the RISC-V pipeline. Results show a speedup of up to 1.3x for the latter method, while both approaches are still able to grant improved performances against the plain version of the soft processor.

However, existing RISC-V-based solutions focus mainly on performance, partially neglecting reliability when such design operates in safety-critical environments. To address this, we propose a computing platform that combines a RISC-V processor with a Tensor Processing Unit (TPU) to enhance both performance and dependability. The platform integrates three main features: (1) a TPU error detection technique based on checksums on current inference execution, (2) fault correction via FPGA dynamic partial reconfiguration handled

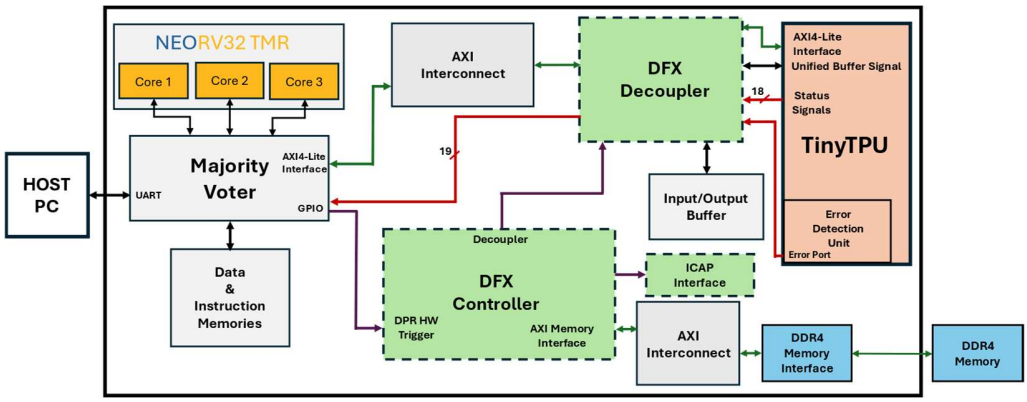

Fig. 1 The proposed platform.

by the RISC-V processor, and (3) a recovery routine to resume computation from the last correct operation. By extending the TPU's ISA with custom fault-detection instructions, our approach ensures real-time error handling while maintaining high performance with minimal overhead.

III. The Proposed Platform

The platform is based on two main computational units: the NEORV32 RISC-V Soft-Processor [9] and the TinyTPU accelerator [10]. The platform follows a processor-coprocessor model where the NEORV32 manages execution by supplying the tinyTPU with microcode via an instruction FIFO for layer-by-layer computation. After each layer, the NEORV32 retrieves and reformats data for the next stage. Fault detection can be enabled per layer, and if an error occurs in the tinyTPU datapath, the NEORV32 triggers dynamic partial reconfiguration (DPR) to restore functionality and resumes execution from the last correct state, minimizing downtime.

Fig. 1 illustrates the architecture, showing AXI-based data exchange between processor and co-processor, UART for host communication, and GPIOs for TPU status monitoring. Faults are detected via interrupts, causing the NEORV32 to initiate DPR through the DFX Controller [11], which loads configuration data (CDATA) from DDR into the configuration memory of the FPGA. A DFX decoupler stabilizes interface signals during reconfiguration. If DPR fails to resolve the issue, a full FPGA reconfiguration is executed as a last resort.

a. NEORV32

The NEORV32 is an open-source RISC-V processor with a 4-stage pipelined multi-cycle architecture, operating at up to 250 MHz and supporting the RV32I ISA, as well as many extensions. In the proposed platform, AXI-Lite communication is enabled for interfacing with the TPU, while the UART module supports debugging and host communication. To enhance reliability, the NEORV32 is hardened with Triple Modular Redundancy (TMR).

b. TinyTPU and Fault Detection Mechanism

The tinyTPU is an open-source, VHDL-based machine learning co-processor inspired by Google's TPU v1[12]. It features a configurable systolic array (SA) ranging from 6×6 to 14×14 Multiply and Accumulate (MAC) units and includes an external accumulator bank used to perform operation tiling on large matrix multiplications. Designed for DNN execution, it supports hardwired quantized ReLU and Sigmoid activation functions. The TPU has dedicated buffers for weights, inputs, and outputs. It operates with a custom 80-bit ISA following a CISC architecture. The instructions used to perform DNN inference are *read_weights*, *matrix_multiply*, and *activate*.

To implement fault detection while minimizing resource overhead, we leverage the existing computational resources within the accelerator's datapath, combining Algorithm-Based Fault Tolerance (ABFT) with scan chain techniques. This approach detects faults by computing comparative checksum values on inference data generated by two independent computational paths.

Specifically, for the scan chain methodology, we apply test patterns to the SA core to generate checksum values for the current DNN weights loaded into the PEs grid. By exploiting the natural top-to-bottom dataflow within the SA—where partial products accumulate—checksum values on the weight data are produced at each column of the systolic core. During the test routine, an additional datapath is activated, connecting the weight buffer to the Accumulator bank. This enables the Accumulators, typically used for summing results of different matrix multiplications, to sum the DNN weights instead, generating the comparative checksum values.

The checksum verification process exploits 2's complement representation of data to determine whether the fault affects the weight registers (i.e., simple bit-flip) or the datapath structure (i.e., change in the netlist). The proposed method is integrated into the TPU ISA through two new instructions: *t_read_weight* and *t_matrix_multiply*.

- *t_read_weight*: loads weights in the PEs grid, as in normal operation, while also computing the checksum values on the Accumulators side. Enabling the Single Instruction Multiple Data (SIMD) mode, the Accumulators can perform checksum computations in parallel with standard operations.
- *t_matrix_multiply:* executes matrix multiplications on application workload data while simultaneously generating checksum values by appending test vectors to the workload.

These checksum values are compared through a detection unit, which triggers an interrupt in the NEORV32 if an error is detected. The method introduces only a minor delay of three clock cycles (due to test pattern applications), ensuring efficient error detection with minimal performance impact.

c. Recovery Routine

Upon fault detection, the TPU pipeline is flushed, and NEORV32 gathers details about the faulty instruction to ensure that the recovery starts from the last correctly executed

step. If all computations within a layer run in testing mode, fault detection latency is limited to a single matrix multiplication operation. Otherwise, if only key operations (e.g., the first and last matrix multiplications of each layer) are done in testing mode, execution must restart from the beginning of the faulty layer to prevent error propagation.

The NEORV32 manages TPU reconfiguration via the DFX controller. It waits for reconfiguration completion by polling an *alive* signal from the TPU, and if the TPU remains faulty after two consecutive reconfiguration attempts, a full FPGA reprogramming is initiated, resulting in a full restart.

IV. Experimental Results

The platform has been implemented on an AMD KCU105 Evaluation Board. Table 5-1 reports the resource utilization of the entire system.

Table 5-1. Platform Resource Utilization

Platform Modules	LUTs	FFs	BRAMs	DSP
TinyTPU	4,294	7,211	181	210
TMR NEORV32	3,219	3,180	3	0
DPR Logic	1,185	989	0	0
Glue Logic Resources	13,874	17,670	95.5	3
Total [%]	9.31%	5.99%	46.58%	11.09%

The area overhead introduced by interfacing the two cores and the DPR logic, listed as glue logic, is quite limited. The high number of BRAMs has to be attributed to the memory requirements of both the TPU, which uses the BRAM to store the DNN parameters and the RISC-V Processor, which requires a large data memory to perform intra-layer computations used to arrange convolutional layers as General Matrix Multiplication (GEMM) operations.

The entire platform operates at 100 MHz, and to further enhance the reliability of the system, the data and instruction memories of the TMR RISC-V Core were enhanced through ECC.

A. Experimental Analysis

Two different kinds of analysis were conducted on the platform. Through a dedicated fault injection campaign, we evaluated the developed fault detection capabilities of the TPU core. During this process, 20,000 distinct faults were injected in the TPU datapath by corrupting the configuration bitstream of the design, simulating the SEU effect on CRAM. Since bitflips in the configuration memory of the device can lead to various types of faults, including stuck-at, bridge, conflict and open faults, we were able to validate the performances of the system under different scenarios.

As a benchmark, two different classification tasks were selected, the MNIST and the CIFAR-10. For each injected fault, inference was performed on a random sample of 10 images selected from the test dataset to analyze possible data-induced fault-masking effects during processing. The inferences have been executed in testing mode for all the computations operations. The experimental results of the fault injection campaign are detailed in Fig. 2. The aim was to validate the proposed error detection technique, rather than assess the efficiency of the proposed platform. Therefore, partial reconfiguration was not yet implemented.

It is evident that the proposed approach demonstrated extremely good performances, detecting up to 94% of the faults. The execution overhead depends on the number of instructions that are executed in testing mode. Given that each testing instruction introduces an overhead of 3 clock cycles, the total overhead can be computed as *3 clock cycles * number of layers * number of matmul per layer.* Fig. 3 reports the worst- and best-case scenarios, corresponding respectively to the condition in which all instructions of the layers are executed in testing mode or when the testing mode is activated only on the last *matmul* of each layer.

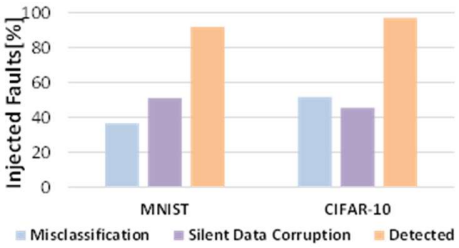

Fig. 2 Fault injection Results.

Despite executing a single testing instruction per layer reduces the overall overhead, it requires the full layer to be re-executed from the beginning when a fault is detected, increasing recovery overhead. On the other hand, executing every instruction in testing mode can provide many advantages especially when partial reconfiguration is implemented.

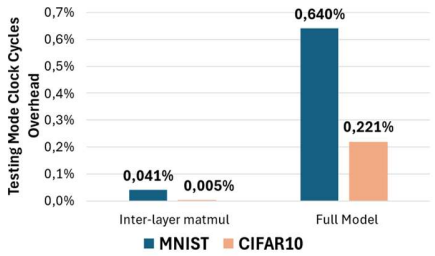

Fig. 3 Maximum and Minimum Clock Cycles Overhead.

Alongside the fault injection campaign, detailed analyses were conducted regarding the system performance, in terms of execution time, when the DPR is implemented. As highlighted in Fig. 4, the time required for DPR scale linearly with the systolic array size, reaching up to 14 ms for a 14x14 SA size, the maximum currently supported by design. On the other hand, full board reconfiguration, which scales with the device size, takes nearly 13 seconds on the KCU105, making the DPR around 870x times faster. This consideration remains constant as smaller FPGA devices are used since smaller devices would support smaller systolic arrays and reduce DPR times accordingly.

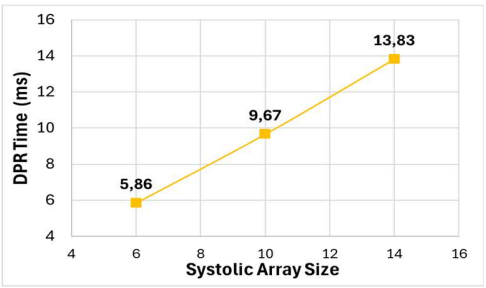

Fig. 4 TPU DPR time related to the SA size.

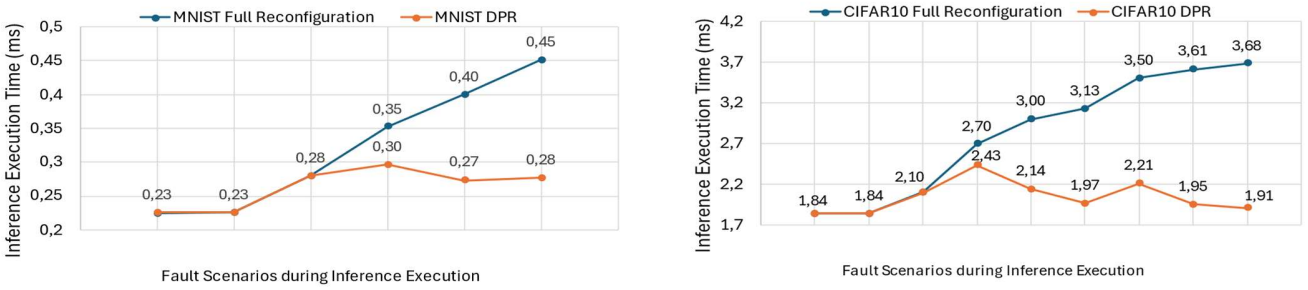

Fig. 5 Total inference execution time considering fault occurrences at different stages of the inference process with and without DPR for MNIST (a) and CIFAR10(b).

Additionally, considering the full board reconfiguration procedure, it requires restarting inference from the beginning, since no data are kept. In this case, the execution overhead strongly depends on fault timing. In the best case, a fault occurs at the first instruction of the first layer, minimizing re-execution. In the worst case, a fault at the end of the last layer, forcing a full re-run. Moreover, in the full board reconfiguration, the NEORV32 boot time introduces an additional overhead. On the other hand, when DPR is adopted, inference resumes from the last correct instruction or layer if all the instructions in all the layers are executed in testing mode, or at least the first and last instructions of all the layers are executed in testing mode, respectively. This is achieved by excluding the TPU input/output buffer from DPR to keep the processed data during the accelerator reconfiguration, while protection from SEU is achieved by enabling ECC in the BRAMs. The overhead of re-executing a faulty instruction is minimal, under 2 µs, especially when compared to the total inference time. However, if the layers are not fully executed in testing mode, for instance, only the first and last instructions are checked to ensure correct inter-layer processing, the total execution time with DPR depends on the layers' complexity and the CNN model itself. Fig. 5 details a comparison between the computational overhead for the DNN execution in the full reconfiguration or DPR cases. These values account only for the computational overhead caused by restarting the program, excluding reconfiguration time (which is in the order of seconds). If the fault occurs during the execution of the first layer, the computational overhead is the same for both DPR and full reconfiguration. In this case, the only differentiating factor between the two techniques is the total reconfiguration time. Starting from the second layer, the use of DPR becomes significantly more advantageous, allowing the computations completed before the fault to be preserved. In contrast, with full reconfiguration, the overhead increases as more layers are processed, as all computations must be performed again. Additionally, DPR offers significantly shorter reconfiguration times with respect to the full reconfiguration, further reducing the overall system downtime.

V. Conclusions

This research introduces a reliable and high-performance RISC-V-based system for DNN execution in safety-critical environments. The platform integrates a TMR-based NEORV32 RISC-V core with a systolic array accelerator, the tinyTPU, augmented with ISA extension to provide runtime fault detection mechanisms. In addition, an efficient error correction procedure based on partial reconfiguration is used to mitigate the impact of system faults. The implemented fault detection techniques achieved up to 94% accuracy with minimal hardware overhead. Furthermore, the use of dynamic partial reconfiguration significantly reduced system downtime, delivering up to a 900× improvement on medium-to-large FPGA devices.

References

1. “The RISC-V Instruction Set Manual, Volume I: User-Level ISA, Document Version 2.2”, Editors Andrew Waterman and Krste Asanovi´c, RISC-V Foundation, May 2017.
2. D. -Z. Li, H. -R. Gong and Y. -C. Chang, "Implementing RISCV System-on-Chip for Acceleration of Convolution Operation and Activation Function Based on FPGA," 2018 14th IEEE International Conference on Solid-State and Integrated Circuit Technology (ICSICT), Qingdao, China, 2018, pp. 1-3, doi: 10.1109/ICSICT.2018.8564810.
3. E. Vacca et al.,"On Assessing the Robustness of RISC-V Soft Cores for Space Systems by Mission-Tailored SEU Analysis," 2024 31st IEEE International Conference on Electronics, Circuits and Systems (ICECS), Nancy, France, 2024, pp. 1-4, doi: 10.1109/ICECS61496.2024.10848907.
4. E. Vacca et al. “Analyzing the SEU-induced Error Propagation in Systolic Array on SRAM-based FPGA," *2023 23rd European Conference on Radiation and Its Effects on Components and Systems (RADECS)*, Toulouse, France, 2023, pp. 1-4, doi: 10.1109/RADECS59069.2023.10767017.
5. AMD UltraScale Architecture Soft Error Mitigation Controller LogiCORE IP Product Guide (PG187).
6. Héctor Martínez et al, “Parallel GEMM-based convolutions for deep learning on multicore ARM and RISC-V architectures”, Journal of Systems Architecture, vol. 153, 2024,doi: j.sysarc.2024.103186.
7. Xingbo Wang et al, “RV-GEMM: Neural Network Inference Acceleration with Near-Memory GEMM Instructions on RISC-V”. in 21st ACM International Conference on Computing Frontiers (CF '24) pp. 302–305. doi: 10.1145/3649153.3649181
8. J. Wei, L. Zhang, Z. Yu and D. Liu, "Design Space Exploration for Heterogenous SoC Integrated with Matrix Accelerator," *2020 IEEE 2nd International Conference on Circuits and Systems (ICCS)*, Chengdu, China, 2020, pp. 40-43
9. S. Nolting and Contributors, “The NEORV32 RISC-V Processor.” Zenodo, Aug. 18, 2023. doi: 10.5281/zenodo.8260609.
10. Jonas Fuhrmann, “Implementierung einer Tensor Processing Unit mit dem Fokus auf Embedded Systems und das Internet of Things”,Germany,2018.,http://hdl.handle.net/20.500.12738/8527
11. AMD, Dynamic Function eXchange Controller v1.0 Product Guide (PG374).
12. N. P. Jouppi et al., “In-datacenter performance analysis of a tensor processing unit”, ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA), 2017, pp. 1-12