CNN-Oriented Placement Algorithm for High-Performance Accelerators on Rad-Hard FPGAs

Luca Sterpone, *Senior Member, IEEE*, Sarah Azimi, *Member, IEEE*, and Corrado De Sio, *Member, IEEE*

***Abstract*—Convolutional neural networks (CNNs) are quickly becoming one of the most common applications running on hardware accelerators. Considering field programmable gate arrays (FPGAs), due to their high flexibility and computational performance, they are suitable for fast classification tasks and therefore, pave the way for new machine learning inference approaches. In this work, we first designed a fully interconnected CNN architecture implementable on a single FPGA. Second, we developed a new neural node-oriented placement algorithm to enable resilient CNN accelerators on space-grade FPGAs. The proposed solution reduces the single event transient error sensitivity of CNN single neuron cores while achieving high performance and effective overall convolutional architecture fault tolerance. The developed approach has been applied and integrated into a state-of-the-art radiation tolerant FPGAs (RTG4) implementation flow. The experimental evaluation has been performed on a microchip test board through benchmark application performance evaluation and transient error analysis. Experimental results demonstrate an improvement of 27.2% of the maximal working frequency and a reduction of the transient error sensitivity of about three times with respect to the previous mitigation approaches.**

***Index Terms*—Convolutional neural network (CNN), fault tolerance, field programmable gate array (FPGA), place and route algorithm, radiation hardened technology.**

I. Introduction

THE PROGRESSIVE advent of vision-oriented elaboration algorithms adopting deep learning techniques increases the usage of hardware devices capable of supporting convolutional neural networks (CNNs) computation. On the one hand, graphic processing units (GPUs) are one of the most popular architectures to accelerate CNN computations thanks to their parallel arrays of streaming multiprocessors allowing a straightforward elaboration of high-level parallel software algorithms [1]. On the other hand, the high performance of recent field programmable gate arrays (FPGAs) as well as their capability to be reprogrammed easily lead them to be an appealing solution for high-performance demanding algorithms with limited power consumption and high efficiency [2]. Furthermore, by the advancement of high-level synthesis tools and therefore speeding up the designers' productivity [3], the implementation of FPGA-based CNN core accelerators and the implementation of GPU-like architectures on FPGA becomes feasible [4], [5].

In general, CNN operations require massive parallel computation. When the main computational core is implemented on hardware architecture, it is characterized by the convolutional product that requires several multiply and accumulate (MAC) cores as convolutional computation grows exponentially [6].

Hardware-implemented CNNs are adopted in many applications ranging from automotive to biomedical fields. Besides, nowadays an evident interest is manifested in aerospace applications which do not only require high-performance and low-power devices but also a high level of resiliency, especially with respect to radiation-induced effects that induce permanent and transient errors within integrated circuits [7], [8], [9], [10].

Today, various companies manufacture radiation-tolerant FPGAs, such as the rad-tolerant Kintex ultrascale FPGAs at 20 nm from Xilinx [10] and the rad-tolerant G4 FPGAs at 65 nm from microchip [11]. These devices guarantee high resiliency with respect to radiation effects, however, the circuits implemented on these devices may undergo design constraints related to modular redundancy and signal filtering that may limit the overall circuit performances [12]. Moreover, despite the reconfigurable capability as well as the radiation tolerant (RT) feature of rad-hard FPGAs, they have a drastic limit in the available hardware resources considering the acceleration goal. In this work, we propose a CNN architecture implementable on a rad-hard FPGA with limited resources. The proposed architecture not only improves the performance with respect to the original implementation but also reduces the single event transient (SET) error sensitivity thanks to the developed physical placement algorithm.

A. Main Contributions

The present work has two main contributions. The first one is the design of a CNN architecture fully implementable on a single space-oriented FPGA device with optimized external memory access dedicated to the NN weights able to reduce the data transfer overhead during the inference execution. This result has been achieved thanks to the application of selective quantization and pruning of the NN architecture to meet the

Manuscript received 14 July 2023; revised 19 October 2023; accepted 3 November 2023. Date of publication 10 November 2023; date of current version 21 March 2024. This work was supported in part by the European Space Agency under Grant 4000105142. This article was recommended by Associate Editor M. D. Santambrogio. *(Corresponding author: Luca Sterpone.)*

The authors are with the Dipartimento di Automatica e Informatica, Politecnico di Torino, 10129 Turin, Italy (e-mail: luca.sterpone@polito.it).

Digital Object Identifier 10.1109/TCAD.2023.3331976

rad-hard FPGA resource constraints. The second one is the development of a placement technique that can be integrated into commercially available FPGA development tools, capable of physically mapping CNN architecture on rad-hard devices while optimizing the computing performances and reducing the transient error sensitivity.

First, the placement algorithm extracts the structural information of the designed CNN architecture and identifies the hardware-oriented data quantization resource involved in the CNN architecture. Second, the algorithm generates FPGA resource constraints that aim to reduce the timing delay of each single neuron node. This is achieved considering the circuit's critical path and the memory efficiency of the convolutional layer while introducing mitigation constraints to improve the robustness of the CNN versus radiation-induced errors. The proposed approach is the first placement algorithm specifically targeting CNN design on Radiation Tolerant FPGAs. The design obtained using the developed placement algorithm shows higher performance and better resiliency compared with previous works.

This placement algorithm focuses on mapping the CNN circuit and improving the performances thanks to a selective placement of MAC and DSP resources of the CNN core. As a hardware benchmark, we selected the ZFNet CNN [13] adopting a 16-bit parallelism data size, five convolutional layers, and 3 fully connected layers.

We implemented the ZFNet CNN design on the rad-hard RTG4G150 device embedded in the RTG4 Development Kit manufactured by microchip. We evaluate the performance of the implemented CNN considering the maximal working frequency under different conditions and the resiliency versus transient error and total ionizing dose (TID) timing degradation. Please note that the maximal working frequency is measured during the execution of the implemented CNN on RTG4 devices while the transient error and TID analysis are performed using SET analysis and timing simulation with radiation dose accumulation.

Experimental results demonstrated that our approach is capable of achieving an improvement of the working frequency by 27.2% with respect to the original implementation of the ZFNet as well as improving the robustness against transient error more than three times compared to the traditional transient error electrical filtering approach. Besides, the obtained solution does not introduce any timing penalties when TID effects are considered.

This article is organized as follows. Section II describes the CNN architecture adopted in this work. Section III presents the developed implementation workflow and the integration with rad-hard mapping tools. Section IV describes the developed placement algorithm. Experimental results and analysis are presented in Section V. Section VI reviews previous works. Finally, Section VII drafts the conclusions and future works.

II. CNN Architectures on Rad-Hard FPGAs

The implementation of NN architectures on FPGA hardware is facing two challenges. The former is the fitting of the NN elements into a single rad-hard FPGA chip and the latter is the capability of the applied radiation tolerant technique to improve the resiliency while avoiding a drastic degradation of the system performance.

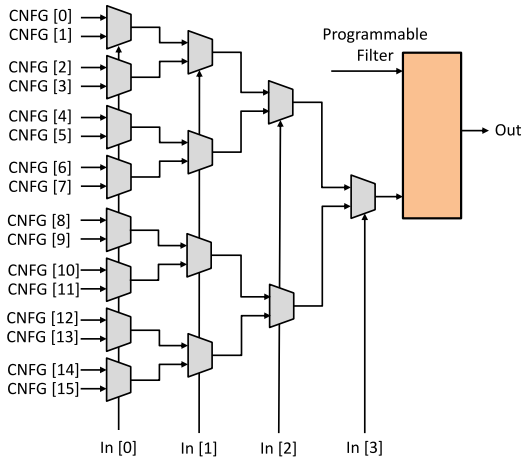

Fig. 1. LUT-4 architecture configuration of the RTG4 rad-hard FPGA. The programmable filter can be tuned with respect to the SET pulse width to be masked.

1) We developed a CNN design that is fittable in a rad-hard FPGA.
2) We developed a new placement technique that can be integrated into commercially available FPGA development tools with a focus on improving the resiliency of the design against transient errors while improving the design performance.

A. 65-nm Rad-Hard FPGA Technology Model

The flash-based FPGA device adopted for the proposed work consists of a logic element array of 4-inputs look-up tables (LUT-4) and a flip-flop (FF) that can be used independently from the LUT-4. The layout of the LUT-4 is based on four input pads driving four tri-state buffers and connected to a two-input multiplexers (MUX2) cascade architecture configured by 16 configuration memory cells. The output of the MUX2 cascade architecture drives a buffer that provides the output signal on the output pad through a programmable filtering module, as represented in Fig. 1. The FF could be configured as a D-type FF (DFF) or as a latch since it has a single data input and an optional enable and two load inputs: 1) synchronous and 2) asynchronous load with clear and preset configurations.

The LUT-4 and the DFF resources have the same supply voltage for the adopted 65nm technology is 1.14 V. The LUT-4 is configured using a flash-based configuration memory cell. The LUT delay has been measured from the input buffer to the output with a worst-case propagation of 119 ps. The propagation time will be considered during the execution of the timing-driven placement routine described in Section V.

The routing architecture consists of a channel-based routing structure organized in clusters. A cluster consists of a set of routing segments connecting the same type of resources. The device has three different types of routing clusters: 1) intralogic; 2) interface; and 3) I/O. The intralogic cluster

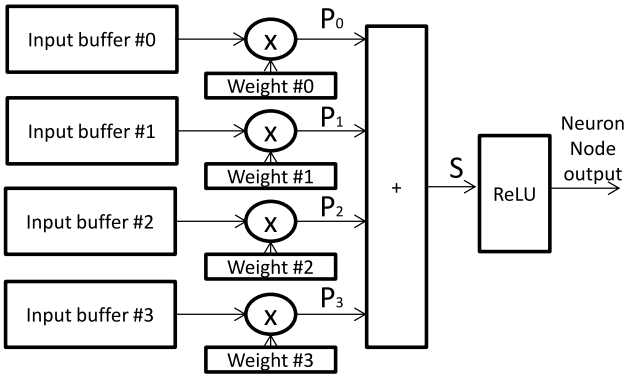

Fig. 2. Neuron structure: the basic element of the ZFNet CNN architecture.

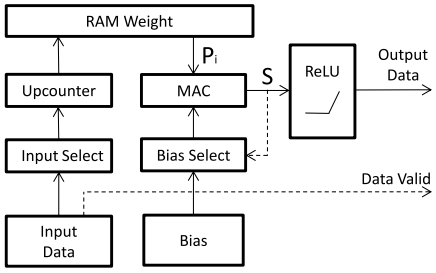

Fig. 3. Implemented structure of the hardware synthesizable neural neuron.

contains the routing MUXes of each FPGA logic element. The interface cluster is a combination of 12 interface logic segments. At the external part of the LUT-4 logic element, the fabric routing structure consists of two parts: 1) intercluster routing and 2) intracluster routing. The intracluster routing provides routing connections between a subset of LUT-4, while intercluster routing provides longer interconnections between clusters. The intercluster routing connects all the clusters and they never drive the input of the functional modules, such as a LUT-4 or a DFF. Furthermore, the routing architecture provides a further short routing segment between adjacent clusters on the same row in order to provide a fast interconnection, typically used for extra carry chains or specific propagation signals. These paths offer a better-propagation performance to the driven signals

B. CNN-Mapping on Rad-Hard FPGA

The mapping and implementation of a neuron on FPGAs require an accurate understanding of the clock timing and the parallel operation of the FPGA in order to achieve full optimization of the hardware. The important parameters that characterize the implementation of a neuron on FPGAs are related to the data movement, its implication on caching, memory requirements, and computational algorithms.

A typical neuron structure within a CNN architecture is composed of a set of data input buffers, also known as synapses, each one multiplied by a specific weight. The set of obtained products (P_i) is added and rectified by the rectified linear unit (ReLU) as illustrated in Fig. 2. The inputs, outputs, weights, and product outputs are floating-point data, however, for the sake of the developed implementation, we adopted fixed-point representation.

In order to implement a complete CNN, several parallel neurons must be instantiated. All the data flow traversing the structure from the synapse inputs up to the post-rectified linear output is represented by 16 bits. The product requires higher resolution for the multiplication and extra range for the accumulation to avoid overflow conditions of any arithmetic process. Therefore, the NN structure consisting of fully parallel neurons is not optimized for FPGA devices, since with a parallel structure the NN is limited in scale by the number of multiplier and accumulator modules available on the FPGA.

A feasible solution for an efficient and practical implementation of CNN on FPGA is based on customizing the MAC units depending on their architectural organization within the NN and tailoring their physical implementation depending on the FPGA hardwired resource availability and organization.

The scheme of the adopted implementation considered in this work is illustrated in Fig. 3. The structure of the hardware synthesizable neuron consists of an input stream of 256 16-bit data words simultaneously read by all the neurons in the same layer. The layer of parallel neurons reduces the limitations on the input bandwidth thanks to the essential data caching. The data inputs are multiplied by the weights; each weight value is associated with a neuron depending on the number of neuron instances.

A crucial implementation detail in producing a CNN mapped on a single FPGA device is the minimization of the movement of the data input to the MACs. The developed implementation exploits data reuse and caching, since all neurons in the same layer use the same input data, considering that a layer of parallel neurons is able to minimize the bandwidth for that computational part. In detail, at the end of the acquisition of the convolutional network, each weight value is used several times for every pixel position on the output. Therefore, for the considered layer, the weight buffer value is maintained synchronized to the multiplier. This structural aspect is dependent on the parametric usage ratio of the weight buffer values. In particular, in the FPGA device used for this purpose, a portion of the 18Kb memory blocks are used for each multiplier in synchronization with the MAC modules.

In the present work, we considered the ZFNet CNN [13], a convolutional network with reduced size, suitable to evaluate mapping and implementation tools. The architecture of ZFNet consists of five convolutional layers and three fully connected layers.

The input data stream consists of a 224 by 224 image crop with 3-color map convolved with 96 filters at the first layer, each one with a size of 7 by 7 and adopting a horizontal and vertical stride of 2. The feature map is then passed through a rectifier linear function, max pooled with a 3×3 matrix with stride 2, and finally normalized across feature maps generating a 55×55 elements feature map. The intermediate layers 2 to 5 repeat the same operation, while the final two layers are fully connected and elaborate the features from the top convolutional layer in a vector of 9 216 dimensions. Finally,

TABLE I
CHARACTERISTICS OF ZFNET RESOURCES REQUIREMENTS WITH 16-BIT DATA PARALLELISM

Layer	Neurons [#]	Routing Channels [#]	Weight SRAM [Kb]
#0	96	6	28
#1	256	192	1,228
#2	384	512	1,768
#3	384	768	2,654
#4	256	768	1,768
#5	4,096	18,432	75,496
#6	4,096	8,192	33,554
#7	1,000	8,192	8,192

the last layer is a soft-max function with i-way, with i being the number of classes. The characteristics of the ZFNet are shown in Table I; where, for each layer, we reported the number of neuron cores, the number of routing channels of the 16-bit data bus between the neurons, and the total amount of SRAM memory Kb demanded to store the CNN weights.

We performed the implementation of ZFNet on the RT4G150 radiation hardened flash-based FPGA embedded in the RTG4 Development Kit. The RTG4 is the first FPGA fabric for high-performance applications, resilient to radiation-induced effects in several space environments ranging from low-Earth orbit to deep space with high-TID robustness and SET filtering feature. However, we needed to face two design challenges before having a ZFNet working on our platform.

The former involves memory optimization. In fact, the total weight memory size for the convolutional network layers is 7.5 MB, while the total weight memory size for the fully connected layers is 116 MB. Storing all the weights on the FPGA fabric large SRAM is not feasible. Therefore, we used the DDR3 memory available on the FPGA Development Kit for storing all the weights, while the on-chip memory is used as a cache memory for the weights temporarily involved in the computation.

The second challenge has been the design of the convolutional layer with particular emphasis on the neural structure and its connection to the neuron within the same layer. Considering the architecture of ZFNet for the first five layers, the access to the data inputs is very homogeneous since the data for a given computed frame is stored in a contiguous buffer that can be read from the external DDR3 memory module. This configuration produces a data stream that can be buffered on the fabric's large SRAM memory with a size equivalent to the mask used for the convolutional product. This results in a memory page associated with each convolutional layer with a tunable size with respect to the computational mask of each convolutional filter. The memory is sufficiently deep to store the weights of the larger layer; however, this may result in a minimal overhead on extra memory resources when the computation is performed for the smaller layers.

We implemented on the Radiation Tolerant FPGA the first five convolutional layers of the ZFNet interfaced with a buffering controller which transfers the weights from the external DDR3 memory located on the bank0 to the internal fabric large SRAM memory and two DMA channels connected to the ZFNet input stream and to the output map generated by the last convolutional layer, linking the DDR3 bank 9 for transmitting and receiving the I/O data.

TABLE II
RESOURCE IMPLEMENTATION FOR THE ORIGINAL ZFNET INFRASTRUCTURE

Resource	Available	Utilization	
		[#]	[%]
4LUT	151,824	130,560	85.9
DFF	151,824	137,472	90.1
Math 18x18	462	192	41.6
LSRAM 18K	209	155	74.2

The implementation resource details of the original ZFNet infrastructure are reported in Table II. We provide the number of 4-input LUT, DFF, Math 18×18 DSP core, and block RAM of 18Kb. The results show that for the first five layers, we use 192 shared multiplication units, 384 accumulators, and 2.8 Mb of fabric large SRAM on 155 blocks for storing the internal ZFNet weights. The number of resources used for the buffering and DMA interfaces is around 4% for the 4-LUT and 6% for the DFFs. The architecture does not use any fabric micro-SRAM embedded in the RT4G device.

C. CNN Quantization and Pruning

The implementation requirements of ZFnet include a significant amount of storage, external memory bandwidth, and FPGA external computational resources. As demonstrated in the previous section, a large amount of memory storage is not supported by the RTG4 and hence the weights have to be stored on external memory and transferred to the FPGA during computation. This problem is exacerbated by the increasing number of layers since it is expected that future CNN models will get more complex and with a larger number of layers. In addition, different layers in CNNs have different characteristics resulting in different parallelism and memory access requirements.

In order to achieve a CNN implementation on a unique FPGA device, we applied a pruning method to the original ZFNet implementation. Pruning is an approach for removing nodes from the NN architecture without incurring drastic accuracy loss. It could be done either by removing weights, neurons, or even entire channels of the NN. We applied the following pruning approaches to the ZFNet original architecture.

1) *Unstructured Pruning of Random Weights:* Pruning of the largest memory fully connected layers and convolutional layers.
2) *Unstructured Pruning of the Smallest Weights:* Removal of neurons with weight values below a given threshold (K_{small}).
3) *Structured Pruning:* Specific removal of neurons belonging to the largest layers (#5, #6, and #7) with a maximal weight SRAM per layer of 5 Mb.

The architecture of ZFNet has been evaluated using 8,000 patches of 224×224 extracted from the European Space

TABLE III
ZFNET PRUNING TEST ACCURACY COMPARISON

ZFNet method	Pruned Parameters [%]	Test Accuracy[%]
Original	n.a.	99.26
Random Weights	39.19	88.87
Smallest Weights	38.26	74.85
Structured	76.17	90.24

TABLE IV
ZFNET LARGE LAYERS PRUNING CHARACTERISTICS

Layer	Neurons [#]	Routing Channels [#]	Weight SRAM [Kb]
#5	512	2,526	4,804
#6	512	2,048	4,096
#7	256	2,048	2,048

Agency (ESA) dataset of 26 unprocessed and raw images taken by the onboard camera of the OPS-SAT [20].

Table III reports the comparison of the different ZFNet architecture methods with the percentage of pruned parameters and the measured test accuracy. According to our analysis, both the unstructured pruning based on random and smallest weights have worse performance than structured pruning, while having a similar percentage in terms of pruned parameters. On the contrary, the structured ZFNet version removes a consistent percentage of weights exclusively on larger layers, while outperforming the test accuracy of the random and smallest weights methods. It is interesting to notice that, in case would be necessary a further reduction of area, it would be possible to adopt approximation techniques in order to minimize the storage and memory bandwidth.

We applied the pruning selection to the ZFNet hardware architectural model. Thanks to the structured pruning, we obtained a substantial reduction of the neuron nodes belonging to the large layers as illustrated in Table IV which accounts for 74.87% of the total neuron nodes and 76.06% of the total routing channels. Thanks to the structured pruning, we achieved two fundamental goals. The former consists of reducing the requested combinational and sequential resources for implementing simultaneously all the ZFNet layers on a single FPGA; the latter consists of reducing the maximal requested SRAM for the weight storage to less than 5.2 Mb (e.g., the maximal on-chip SRAM size for the RTG4 device considered), thus nullifying the data transfer overhead of the fully connected layer with an external memory bank. Thanks to this solution, we estimated a saving of around 9% of the overall computational load, this is due to the memory accesses performed by the fully connected layers during inference computation.

Finally, we implemented all the ZFNet layers on the radiation tolerant RTG4 FPGA while maintaining an external RAM memory for storing the I/O data stream and storing all the layer weights. The convolutional layers of the ZFNet are interfaced with a synchronized buffer controller which transfers the weights from the external DDR3 memory located

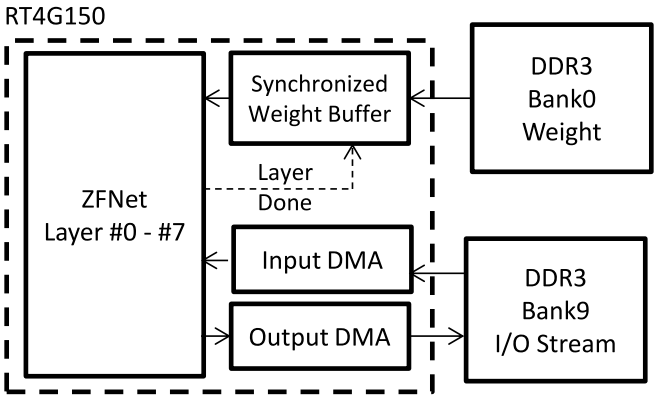

Fig. 4. On-chip implementation architecture of ZFNet on the RTG4 Development Kit

TABLE V
RESOURCE IMPLEMENTATION FOR THE ON-CHIP ZFNET

Resource	Available	Utilization	
		[#]	[%]
4LUT	151,824	120,904	85.9
DFF	151,824	123,084	81.1
Math 18x18	462	382	82.7
LSRAM 18K	209	202	96.7

on the Bank-0 to the FPGA internal fabric large SRAM memory. Two DMA channels are connected to the ZFNet input and output stream, linking the DDR3 Bank-9 for transmitting and receiving the I/O data.

The scheme is illustrated in Fig. 4, while the resource details are depicted in Table V, where we report the number of 4-input LUTs, DFF, Math 18×18 DSP core, and block RAM of 18Kb. As it is possible to notice, even if we are now mapping all the nine layers, the 4-LUT and DFF number is reduced by around 7%, on average, versus the original 5 layers ZFNet. This is due to the selective pruning that is also reducing the number of requested routing channels, therefore a reduction of the number of hardware neurons on the larger layers is impacting also the circuit resources of the first layer routing channels (essentially the connections between the layer fourth and the fifth). On the contrary, the number of Math DSP cores and Large Static RAM (LSRAM) drastically increased. A total of 382 DSPs are now used for shared multiplication units, while 428 accumulators are implemented with standard programmable logic. The fabric's large FPGA is used for 202 blocks for storing the maximal peak of around 4.8 Mb due to the fifth layer. Similarly to the original implementation, the number of resources used for the buffering and the DMA interfaces is around 5% of the total resources. The architecture does not use any fabric micro-SRAM embedded in the RT4G device.

The overall ZFNet implementation contains tensor types defined for the different fixed point precision data types used in the network layer. The input data, weights, bias, products, accumulators values, and rectified linear output registers are all represented by the same size of 16-bit type. This type of implementation may result in some performance degradation,

in the case of data input of a different data parallelism (e.g., 8-bit image data or unsigned values). In general, using a single arithmetic type may result in a loss of performance and higher-energy consumption.

In order to guarantee higher flexibility and further compatibility of the developed ZFNet with other FPGA platforms, we modified the ZFNet model with parametric and reconfigurable parallelism on the input image data size and weights. The former parameter affects the memory bandwidth and on-FPGA data memory cache, in particular, a lower-bit size will result in a deduction of the memory bandwidth requirement, while the latter affects the weight memory cache, a lower number allows more neuron computation to be mapped in parallel on the FPGA architecture.

Furthermore, we modified the ZFNet hardware description model adding the possibility to configure the convolutional structure, in particular, to guarantee the implementation of fully interconnected layers with a parametric number of neurons. In detail, we adopted a MAC-based unit shared between several neurons that allow us to implement the network even in the case of drastically limited resources. The number of neurons sharing the same MAC unit affects the real parallel computation since a higher number of parallel neurons increases the overall network throughput.

III. CNN-Oriented Implementation Flow

RT FPGAs have realized with radiation hardening by design (RHBD) rules; these devices are generally characterized by a very high resiliency against single event latch-up (SEL). Moreover, they are practically immune to Single Event Upsets (SEUs). In particular, the RT4G devices have logic elements with triple modular redundancy (TMR) FFs; therefore, the designer does not need to apply global or partial TMR techniques generally adopted for SRAM-based FPGAs.

The implementation workflow for RT FPGAs provided by the commercial tool is based on two main objectives: 1) limiting the timing degradation induced by the radiation dose accumulation or 2) TID and mitigating the impact of SET on the circuit behavior. Both TID and SET phenomena are mitigated by acting on EDA tool constraints. However, when these methods are applied to CNN, the performances are drastically limited. In this section, we describe a placement algorithm able to consider the characteristics of CNN circuitry and force logic element location on the FPGA architecture in order to optimize the circuit performance while satisfying the radiation-tolerant constraints regarding SET phenomena.

A. Physical Design Constraints Instrumentation

The CNN implementation on rad-hard FPGAs follows the traditional FPGA design steps, such as synthesis, mapping, and place and route. In order to apply the developed design flow, we act on the physical implementation of the place and route step controlling the executed algorithm by the physical design constraints (PDCs). We use the Verilog description of the CNN as input, and we merge the netlist information into a physical design description (PDD) file.

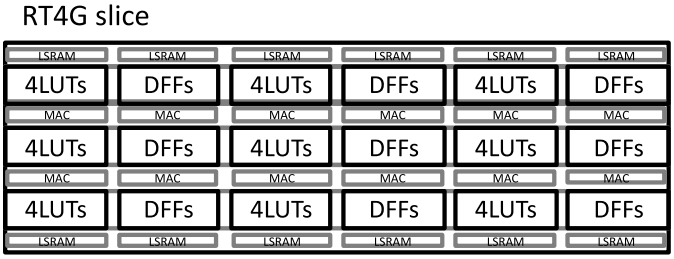

Fig. 5. Portion of the RT4G150 device: the RT4G slice.

The PDD contains the number of elements that belong to a specific function hardwired on the RT4G device that consists of the RT4G slice, as illustrated in Fig. 5; for each of the functions, it provides the identifier, name, coordinates, and the list of input and output elements connected. The PDD file is used as an input for the developed placement algorithm, as well as the interface with the commercial toolchain used to finally upload the CNN on the RTG4 device. The developed placement algorithm will require an area constraint associated with each CNN neuron before it is executed. In order to properly generate the area constraints, it is necessary to characterize the device topology.

The implementation flow is based on the computation of the centroid that allows defining any logic node position that minimizes the average distance with all the connected units considering the wire length and the delay. In order to identify the best location for the NN resources, we identify two strategies for each logic node of the NN.

1) *Centroid Strategy* where the centroid is connected to the NN resources that belong to a given layer. The computation of the centroid allows the definition of the LUTs and DFFs position that minimizes the average routing distance with the entire connected unit and therefore the wire length.
2) *Data-Path Strategy* where the centroid is computed again with respect to the NN resources belonging to a given layer, but then places the node within the area region related to the specific node minimizing the wire length related to the data path of the considered neuron.

Since it is not possible to elaborate the constraints independently from the layers and each specific neuron, an appropriate placement algorithm must be defined to take into account the two CNN implementation strategies and to implement the selective SET mitigation on the critical nodes.

B. Physical Placement

The placement workflow for CNN adopts the PDD file as an input description of the CNN netlist graph and it generates a physical constraint file (PDC), including the physical location of every single basic resource of the RT4G FPGA according to the CNN implementation flow. The flow is based on defining the CNN constraint regions illustrated in Fig. 6(a) and it is executed by the following three phases.

1) It defines the maximal area associated with each layer region. The constraints will limit the placement algorithm to allocate the resources belonging to a single layer (internal to each neuron or for connecting them)

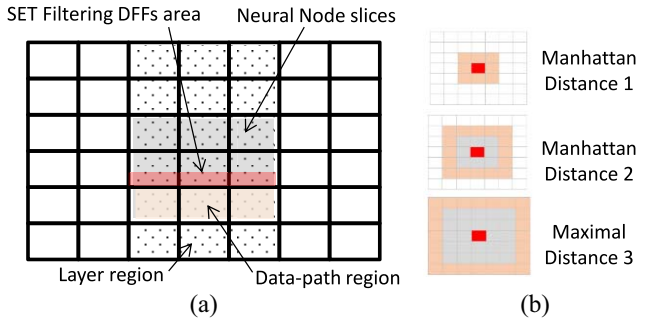

Fig. 6. Top-down overview of the CNN constraint regions (a): layer, neural node, data path, and SET filtering DFFs area. The three-searching path from Manhattan distance 1 to Manhattan distance 3 (b).

into a region that will be optimized for the layer performance.

2) The layer region is divided into neural node slices: groups of RT4G slices dedicated to implementing the single neuron computational capabilities. Each neural node slice may contain two LSRAM memory blocks and at least one MAC block.
3) The neural node slices are organized in data path regions and other resources. The data-path region is an exclusive region since only the four LUTs and DFF dedicated to implementing the data path of the neural neuron will be located there, while all the other resources are located outside of this region. This constraint will guarantee a better reduction of the timing delay due to the combinational logic resources associated with the multiplication, sum, and rectifier functions.

Once the coordinates of the layers, neural node slices, and data-path regions are fixed, the elements associated with a proper hierarchical region should be placed exclusively inside the region, while the remaining nodes must be placed only by connection with nodes of the same layer with the objective of not introducing any timing delay penalties.

To manage this kind of placement, a new algorithm for CNN core has been developed. A new set of fixed placement locations defines each specific neuron data path and when the placement algorithm is executed, it will set the position of the fixed logic node, and then, it will run a 2-D placement algorithm on the remaining nodes. In this way, the wire length will be optimized on the current logic dedicated to each single neuron data path, while the links to the memory block dedicated to the weight are managed without a timing priority. This process is repeated for all the neural neurons' logic and in each step, it manages the reduction of the wire length and the vertical connection to the memory blocks storing the weights. Finally, the DFF registers storing the input operands for the MAC unit will be placed within a specific SET filtering area. The filter area can be expanded up to the maximal Manhattan distance of three units, as illustrated in Fig. 6(b). At this stage, the filtering activation is not performed yet. Please note that this constraint is one of the most complex phases in relation to the mitigation capabilities of the obtained CNN circuit. In particular, the weight data will be transferred to the input of the MAC unit during the execution of the CNN, and the weight values will be stored in DFF registers and sampled during the synchronization of the MAC execution phase, during this phase and depending on the resistive and capacitive load of the routing segment a transient effect may be sampled by the CNN network and therefore should be filtered. Therefore, once the placement is performed, the last phase requires the selective activation of the SET mitigation filter.

IV. Placement Algorithm For CNN Nodes

Once the placement constraints are defined, the placement algorithm is executed. The solution may adopt a commercially available placement tool in order to define an intermediate solution. Otherwise, the placement algorithm should be instrumented to achieve better optimization for the convolutional nodes. In fact, placing the resources of each convolutional node of a specific layer, according to the routing delay minimization, led to fixing the 4-inputs LUTs position within the RT4G slice with respect to the distance with the MAC and the LSRAM hardwired component available. This means that each logic element should be optimized with respect to the coordinates of the neighborhood LUTs and DFFs and with respect to the hardwired component.

The developed placement algorithm for convolutional nodes has the objective of minimizing the inter-LUTs connections and the connections toward the hardwired components. The main optimization provided by the placement algorithm is the reduction of the wire length within each neural node slice without inserting a single point of failure while the LUTs are placed in the neighborhood of the MAC and LSRAMs. The placement is executed sequentially for each convolutional node resource, therefore the interconnections between resources not belonging to the same neural node are not considered.

Please consider that commercial placement algorithms do not achieve good results because the algorithm may locate the logic resources linked by a MAC or LSRAM connection in a distant position, thus increasing the wire length needed to connect, thus implicitly increasing the sensitivity of the routing segment to radiation-induced transient effects. For this reason, it is important to properly manage the logic element position aside from the hardwired component because they will heavily affect the total circuit wire length. Moreover, due to the topological location of the MAC and LSRAM within the RT4G slice, their interconnection channels, which are used toward the generic routing segments used for the programmable routing links with LUTs, are generally slow, and therefore they can easily create a critical path.

The execution of the placement algorithm is performed after the generation of the area constraints described in Fig. 6(a). During the generation of the area constraints, the hardwired components have been allocated to the respective layer and data-path regions according to the RT4G slice availability. Once all the resources are allocated to the different hierarchical regions, the placement algorithm is executed. It generates a detailed placement location for each combinational logic and sequential resource and the hardwired macro physical mapping.

The developed placement algorithm is based on five phases as described in the pseudocode reported in Algorithm 1: 1)

Algorithm 1 Pseudocode of the Placement Algorithm Oriented to the Convolutional Node for Radiation-Hardened FPGAs

```
CNN_placement (ports, nodes, layer_area)
nodes: list of logic/sequential elements
port: module pin ports
layer: area constraints (size and layer_edges)
nodes = sort (nodes)
for id in nodes do
    x,y = Centroid (id,nodes)
    adder = 1
    padding = 2
    n_Man = 1
    if y > layer_edge/2 then
        y = layer_edge + padding
    else
        y = –padding
    end if
    Closest_pos(id,nodes)
    while allocated != TRUE do
        if resource_free(x, y) in position_macro then
            (x, y)=locate(x, y)
            if (x, y) > 0 then //centroid-shifting then
                centroid_shifting (n_Man = n_Man + 1)
            else
                centroid_shifting (n_Man = n_Man - 1)
            end if
        else
            allocated = TRUE
        end if
    end while
    nodes[position]=(x, y)
    if nodes==combinational_side then
        add2layer (x, y, datapath) //re-allocation
    else
        add2layer (x, y, other)
    end if
end for
filtering_fill(x, y) //filtering area
```

centroid computation; 2) closest edge identification; 3) centroid shifting; 4) reallocation; and 5) filtering fill.

The algorithm starts by extracting the list of elements belonging to a given layer area and individuating the port pin interconnections and the area constraints in terms of size and edge locations. The first phase consists of computing the centroid for the considered set of combinational and sequential logic elements, including the LSRAM and the MAC ones. Once the centroid coordinates are computed, the algorithm identifies the closest edge of the area constraint and updates the padding, accordingly, as illustrated in Fig. 7. Please consider that the padding is used to separate the logic nodes between the hardwired element, and it may increase the initial size of the neuron area. Once the edge and the padding are calculated, the algorithm applies the centroid shift to the closest edge. If the new position is available, the resource is placed; otherwise, the algorithm iterates recursively to find a free location.

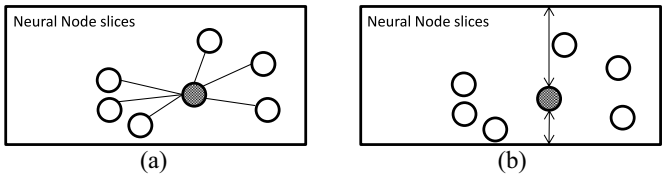

Fig. 7. View of the centroid computation (a) and the identification of the closest edge (b) phases of the developed placement algorithm.

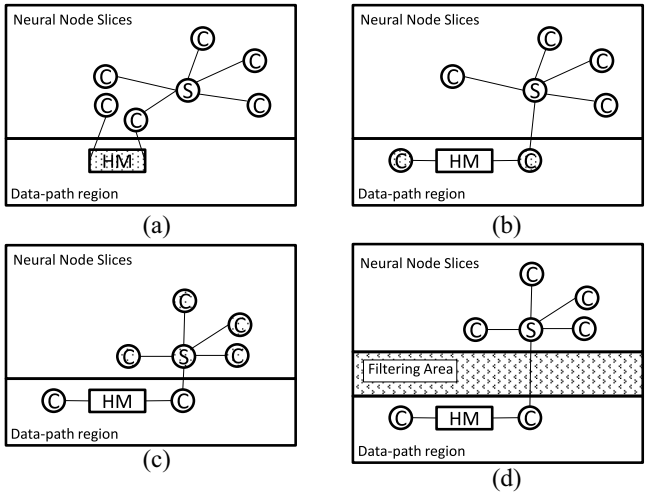

Fig. 8. View of the Placement execution performed for the hard macro MAC module (HM) surrounded by six combinational gates and one sequential element. The resources are placed within the data-path region (a), while the combinational resources directly linked to the Hard Macro are located within the data-path region (b), the sequential element is placed in the neural node slice area (c), the combinational logic filter is inserted in the final placement solution (d).

In detail, starting from the shifted centroid, if its position is not available, the *resource_free* function recursively checks, with a step initially set to n = 1, if the position on its maximal Manhattan distance n neighborhood is available. If one of these four locations is available, it becomes the final position of the resources. Otherwise, step n is incremented by one until a free location is found. The maximal Manhattan distance has been settled to 3. The $\text{position}_{\text{macro}}$ structure is used to keep track of all the placement spots already used by the resource. At the end of this phase, the position of the node is determined, and the position added to the layer detailed placement in the data-path region if they are combinational, and vice versa in the general neuron slice area. Finally, once all the layer resources have been placed, the *filtering_fill* function generates a dummy area that would be subsequently used to allocate the logic resources to implement the guard gate structure for transient effect mitigation. Since the requirements for filtering are not determined yet at this stage, its position is generally allocated in the bottom part of the neuron slice.

An example of the placement execution is illustrated in Fig. 8, where it is possible to notice how the hard macro (HM) resources are placed within the data-path region (a), the combinational resources aside from the HM are located within the data-path region (b), while the centroid sequential element surrounded by four combinational elements are placed within

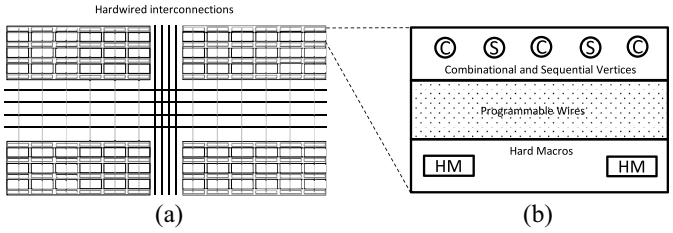

Fig. 9. Portion of the FPGA programmable logic array and the corresponding customizable placement setting parameters: hardwired interconnections, combinational and sequential vertex, programmable wires, and HMs (a) and the relative model, including parametric elements, mapping the realistic FPGA architecture.

the neural node slice area (c). Finally, the insertion of the filtering area is added to the obtained placement solution (d).

A. Placement Setting Customization

The placement algorithm is based on a graph representation that models the logic and routing resources of the FPGA physical layer. The graph representation includes the standard modeling of LUTs and FFs; the model of MAC modules, and the LSRAM modules. Besides, the FPGA architectural graph describes the routing segments used for the programmable interconnections within each routing box and to the hard-wired connection used for the long connectivity channel within the FPGA.

The architectural graph resources have been balanced according to the effective availability on the considered FPGA platform, in details each resource has been parametrized with a weight that can be adapted to the used FPGA architecture and that can be settled in order to manage routing congestions, clock-skew and logic cone delay balancing. In particular, the portion of the FPGA programmable logic consisting of at least four clusters of slices and connected with customized hardwired interconnections is illustrated in Fig. 9(a). The single slice, illustrated in Fig. 9(b), has a parametric definition of the number of combinational and sequential vertices, topology, and a number of programmable wires, and, finally, a distributed set of HM modules.

B. Integration With Radiation-Induced SET Filtering

The main consequence of radiation effects on radiation-tolerant FPGAs are SETs effect. The radiation-tolerant FPGA implements a SET filtering architectural approach. Each Data FF in the programmable logic array and at the input of the DSP multiplication blocks has a SET filter.

The filter can be selectively enabled on individual FF, functional blocks, or the entire FPGA. The innovation of the proposed method is to analytically evaluate the maximal duration of the transient pulse at the input of each sequential element once the placement is performed and to selectively activate the SET filtering capabilities provided by the RTG4 FPGA. The selective activation of the SET filtering will avoid drastic performance degradation due to the massive SET filtering application on each LUT.

In order to select the FFs candidate for the SET filtering, we analyzed the detailed placement, and we evaluated the propagation radiation-induced transient pulse toward the sequential

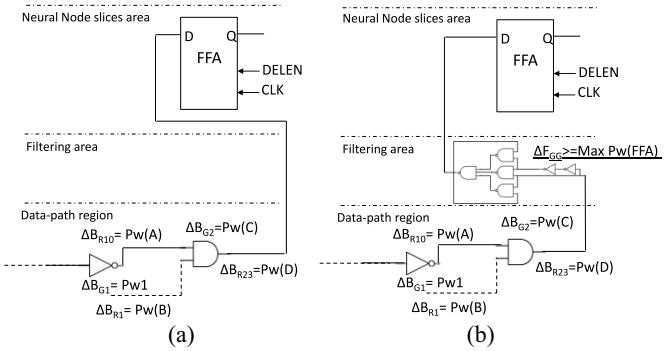

Fig. 10. Original detailed placement (a) and the selective insertion of the guard-gate structure within the filtering area (b).

elements located on the boundary between the neural node slice and the filtering area. We exploited the feature provided by the micro semi-implementation tool to apply the selective insertion of the SET filtering and we considered that for the RT4G device, the manufacturing filtering capability of transient pulses is settled at 0.6 ns.

We analyzed the transient pulse propagation using the analytical tool developed in [21] that provides the maximal SET pulse width observable at the input of a sequential resource, and we implemented two alternative SET filtering methods. In case the SET width is lower than 0.6 ns we enable the selective filtering on the considered FF by setting the mitigation option per instance, vice versa we insert a guard-gate logic structure before the input of the element located in the neuron slice area, as illustrated in Fig. 10. The guard-gate logic structure is composed of four combinational gates individually mapped on four LUTs and several inverter pairs able to insert a filtering capability. The number of inverter pairs is computed based on the maximal pulse width computed for the destination FF. As it is possible to notice in Fig. 9, the placement of the guard-gate logic resources and the inverter pairs requires a fixed placement location in order to guarantee the exact filtering capability.

V. Experimental Results

The developed design flow has been applied on the ZFNet implemented on the RT4G150 radiation tolerant flash-based FPGAs manufactured by microchip. The scheme of the implemented ZFNet is illustrated in Fig. 10. We physically implemented the eight layers of the network and we evaluated the performance and the reliability of a pretrained network using the OPS-SAT subset images.

A. ZFNet Architecture Implementation Details

The ZFNet architecture is a convolutional network model based on 8 layers, as described in Fig. 11. The input data is based on a 224 by 224 image crop with three color planes. This image is convolved with 96 different filters for the red component at the first layer. Each filter has a size of 7 by 7, using a stride of 2 in both x and y coordinates. The obtained feature map is then computed in three different phases.

1) It is passed through a rectifier linear function.
2) It is pooled with a kernel matrix of 3×3 regions, using a stride of 2 units.

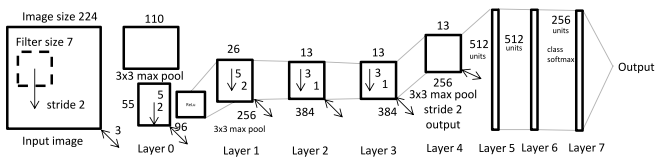

Fig. 11. Architecture of the pruned ZFNet implemented on the RTG4 radiation-hardened FPGA.

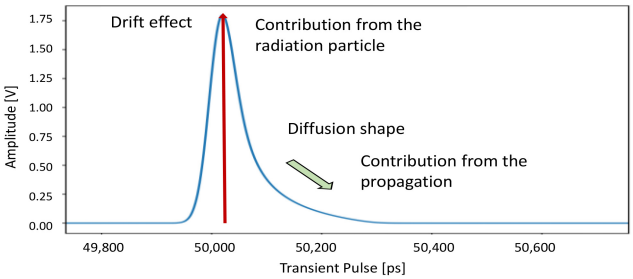

Fig. 12. SET pulse with an amplitude of 1.8 V and duration of 180 ps.

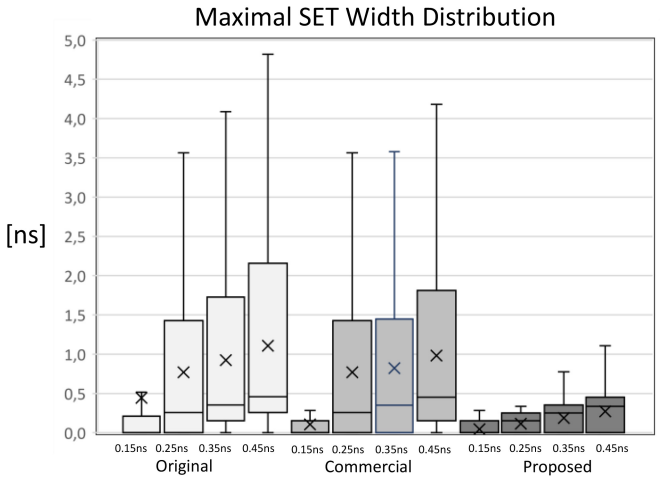

Fig. 13. Maximal SET pulse width distribution for the overall CCN sequential element (FFs and block RAMs).

3) It is contrast normalized across a feature map to generate 96 different 55 by 55 element feature maps.

The same operations are repeated in layers 1 to 4. Layers 5 and 6 are fully connected, elaborating features from the top convolutional layer as input in vector form of 512 dimensions. The final layer is a C-way softmax function with 256 functional units.

The pruned ZFNet architecture has been implemented on the RTG4 device using three approaches.

1) *Original:* Implemented with timing performance optimization and without any specific placement and mitigation constraints.
2) *Commercial:* Implemented with the SET filtering feature provided by commercial tools (i.e., up to 600 ps of filtering capabilities) for all the sequential resources on the device and without any specific placement constraint.
3) *Proposed:* Implemented with the developed placement constraints targeting DSP performance and LSRAM resources and adopting selective SET filtering.

B. Single Event Transient analysis

The SET analysis has been performed on the entire mapped circuitry on the radiation-hardened FPGA, therefore transient pulses have been injected and propagated on the resources belonging to the ZFNet architecture, on the synchronized weight buffer and on the input/output DMAs. For the purpose of our analysis, we used a drift-based transient pulse voltage glitch model introduced in [22] and developed an analyzer tool for FPGAs in [23] adapted to the radiation-hardened 65-nm flash-based FPGA technology.

The propagation of the SET pulse within the node has been done by modeling two pulse effects.

1) *Drift Pulse:* This function describes the physical generation of the pulses. The generation is performed according to the sensitive volume cross section. The amplitude of the generated pulse depends on the cumulative energy (E) on the crossed volumes and the relative distributed eV per node. The eV outcome values are used as the input point of the subset value for the SET generation. The results are the description of the drift glitch, as in Fig. 12, the left part of the glitch shape.
2) *Diffusion Pulse:* This function describes the physical behavior of the Voltage glitch diffusion effect. The shape of the diffusion is computed on the basis of the cell volumes involved in the propagation of the glitch, as illustrated in Fig. 12, the right part of the glitch shape.

We modeled four different SET pulses in order to mimic the overall scenario of radiation-induced SET pulses thus analyzing pulses with the following widths: 150, 250, 350, and 450 ps. We compared the original, commercial, and proposed ZFNet implementations with two analyses: 1) the former consists of evaluating the maximal pulse width at the input of each sequential element (e.g., DFF and input memory pin) with an exhaustive injection in all the circuit-sensitive nodes and 2) the latter consists of evaluating the pulse width distribution over a random injection of 10 000 transient pulses.

The exhaustive maximal pulse width distribution after the propagation is illustrated in Fig. 13. The propagation induced pulse broadening (PIPB) effect is strongly present in the original ZFNet implementation, especially for the longest source pulse that is broadened more than one order of magnitude, thus generating SET pulse glitches at the input of an FF with a duration of 4.8 ns. On the contrary, small source pulses have a reduced PIPB effect which is marginal for the three ZFNet implementations. The commercial ZFNet implementation adopting full filtering of 600 ps per each FF is only marginally reducing the impact of long transient pulses for 350 and 450 ps. The application of the proposed implementation flow efficiently filters the maximal transient pulse width duration, reducing the maximal pulse width duration by approximately 4 times with respect to the original version.

The random transient pulse width distribution has been computed using a Monte Carlo approach with a maximal injection of 10,000 transient pulses in the overall sensitive nodes of the ZFNet implementation, including FFs and block

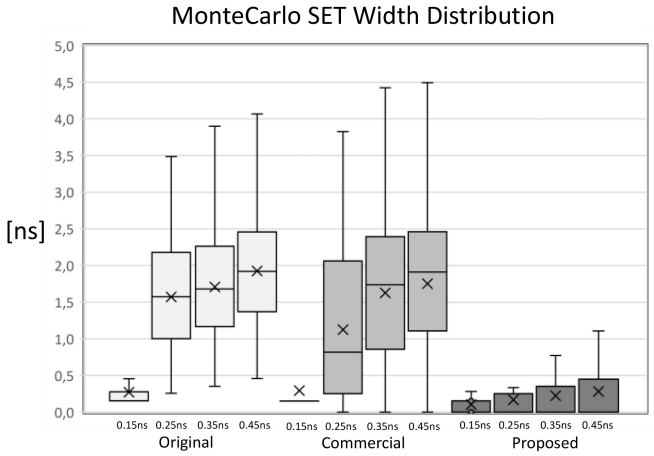

Fig. 14. Monte Carlo SET pulse width distribution obtained thanks to random fault injection on the CCN resources.

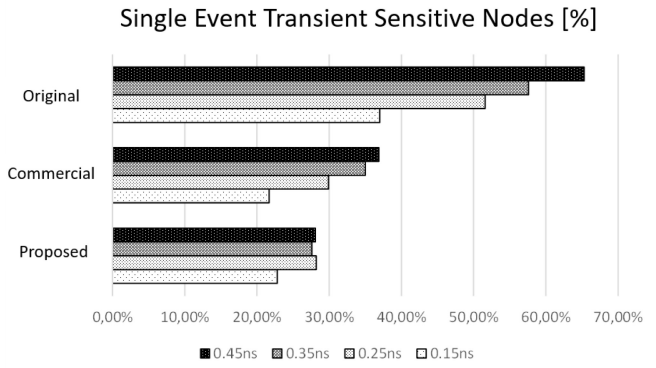

Fig. 15. CCN sensitive nodes comparison for the original, commercial, and proposed solution.

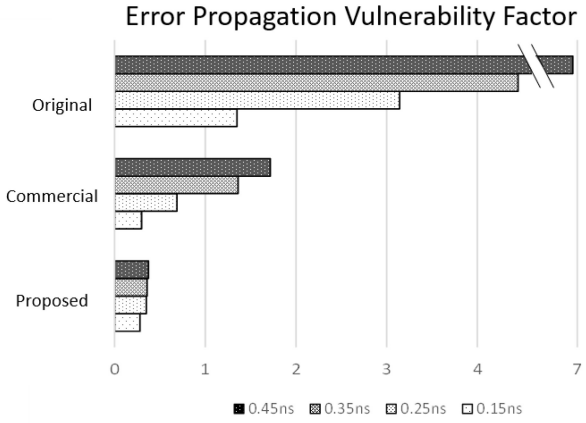

Fig. 16. CCN implementation error propagation vulnerability for the original, commercial, and proposed solutions.

RAM input resources. The results are illustrated in Fig. 14. The PIPB effect does not substantially change with respect to the exhaustive transient analysis, however, it is possible to notice that the median pulse width distribution is higher than the maximal pulse width ones. In particular, the original ZFNet implementation has a median pulse distribution higher than 0.5 ns on average for the 250, 350, and 450-ps original pulse. This means that more than 50% of the transient pulse will be broadened by a factor between 4 and 6 times the original radiation-induced pulse. The commercial implementation of the ZFNet does not improve effectively the original ones, since there is only a marginal reduction of the median pulse distribution of 0.12 ns while the overall median pulse distribution is broadened by a factor between 3 and 4 times with respect to the maximal width distribution. Finally, the proposed solution drastically reduces the median pulse distribution of a factor of more than 4 times, similar to the maximal pulse width distribution. This demonstrates that the proposed approach is not only able to effectively filter the critical sensitive nodes of the ZFNet impacting on the maximal pulse overshoot but it is also capable of effectively selecting the filtering nodes in order to achieve overall lower sensitivity to transient effects. This will result in a lower-error rate of the ZFNet computation.

C. Reliability Analysis

The evaluation of the maximal and random pulse width distribution is not sufficient to have an effective reliability analysis of the ZFNet behavior under radiation-induced transient errors. Therefore, we evaluated further metrics: the SET sensitive nodes and the error propagation vulnerability. The first metric aims to compare how many ZFNet circuit nodes are sensitive to a radiation-induced transient pulse, thus being able to trigger a transient pulse on their sensitive silicon regions. The second metric focuses on evaluating the propagation of the transient pulse after its generation on the sensitive node, thus evaluating how much a circuit is prone to propagate a transient pulse to a circuit sequential resource and then, transform the transient pulse into a computed behavioral error.

The result of the SET-sensitive nodes comparison is illustrated in Fig. 15. As it is possible to notice the ZFNet original implementation has from 37.0% to 65.3% of circuit nodes potentially sensitive to SET ranging from 0.15 ns up to 0.45 ns; the ZFNet commercial implementation is drastically reducing the sensitive node that ranges from 21.7% up to 36.9% of the overall ZFNet circuit nodes. The ZFNet proposed implementation is characterized by a further reduction of the percentage of sensitive nodes to a range from 22.8% to 28.1%.

The result of the error propagation vulnerability is illustrated in Fig. 16. The vulnerability factor is computed as the total number of circuit paths that undergo transient pulse propagation over the total number of injected transient pulses performed during the Monte Carlo random distribution. A factor lower than 1 means that the propagation is reduced, while a higher value means that the propagation is exacerbated by the circuit topology and that any eventual pulse filtering solution is not effective.

The analysis shows that the ZFNet original implementation propagation vulnerability is drastically high with a factor ranging from 1.35 up to 6.95. The ZFNet commercial implementation has effective propagation reduction only for small transient pulses while it is not efficient for longer pulses with a factor ranging from 1.36 up to 1.72. The proposed ZFNet implementation is acting efficiently in reducing the error propagation since for all the transient pulse scenarios,

TABLE VI
FAULT INJECTION ACCURACY RESULTS

Resource	Critical Errors [%]	Tolerable Errors [%]
ZFNet Original	23.5	23.4
ZFNet Commercial	12.9	12.1
ZFNet Proposed	5.2	5.3

TABLE VII
PERFORMANCE CHARACTERISTICS OF THE ZFNET

Resource	Max Frequency [MHz]	Max Clock-To-Out [ns]
ZFNet Original	228.833	12.394
ZFNet Commercial	182.183	12.688
ZFNet Proposed	201.884	12.456

the error propagation vulnerability factor is largely lower than 1 with a factor ranging from 0.28 up to 0.37.

The fault injection accuracy for the ZFNet implementations is reported in Table VI. We further investigate the SET propagation within a single CNN neuron node. In order to depict the critical locations, we extracted the locations of the routing segments and the LUT-4 and DFF. We use the SET model of the RTG4 to calculate the distribution of the transient pulse among the chains and we calculated the distribution per ion for every resource classifying the transient pulse with respect to the voltage amplitude: small if the pulse is between 0.1 and 0.5 V, middle in case of 0.5 and 1.0 V and high in case of 1.0 V up to 5.0 V. The model does not show any pulse provoking a transient glitch greater than 2.4 V. The transient effects affecting the sensitive nodes have widths ranging from 160 ps up to 550 ps. Finally, we observed that short and small SETs have a greater PIPB contribution that is progressively reduced with the increasing width of the pulses. Interestingly, the PIPB effect has a lower impact on high-voltage SET pulses.

D. Accuracy and Performance Analysis

In order to evaluate the overall impact of transient pulses on the ZFNet accuracy and performance, we used the framework proposed in [24] in order to electrically inject SET effects within the logic resources of the RTG4 device. The obtained results are presented in Table III, where we show the percentage of injections that erroneously affect the output of the ZFNet last layer.

The fault injection results are reported in terms of critical errors, for all the injections that provoke no masked effect, and tolerable errors, in the case that the injection does not affect the final network classification. The result shows that the proposed solution outperforms around three times the commercial mitigation approach. On the other hand, it is possible to notice a comparable trend between tolerant and critical errors, except for the ZFNet proposed solution where tolerable errors are more likely to happen. In addition, we analyzed the critical errors on ZFNet and we observed that DSP and block RAMs have similar protection for most of the FFs below the maximal pulse of 590 ps, while the proposed solution is the better alternative if the whole SET spectra are considered.

We compared the performance of the implemented ZFNet architectures by evaluating the maximal working frequency and the maximal clock-to-out delay. The result shows that the proposed ZFNet implementation adopting a DSP filtering priority combined with selective filtering on the NN logic resources outperforms the traditional mitigation solution by more than 27%. The performance comparison of the ZFNet implementations is described in Table VII.

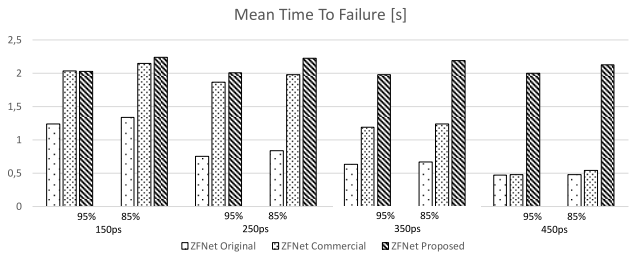

Fig. 17. Mean time to failure (MTTF) for the three ZFNet implementations considering two accuracy levels and with the injection of the SET pulse scenario.

We evaluated the performance of the implemented ZFNet architecture in terms of end-to-end performance characteristics. We chose a set of 40 images from the ImageNet collection on fauna has been selected for the evaluation set. The input has been preprocessed to be suitable for the 224 by 224 image crop with 3 colors, where we measured the average computing time for the given implementations. We observed that the ZFNet original was able to provide a complete classification within 19 ms, while the commercial solution took 24 ms. Vice versa, the proposed implementation was able to reach the classification in an average time of 21 ms.

E. Tradeoff Performance Analysis

In order to evaluate the overall characteristics of the ZFNet implementation, we performed a set of experimental measurements with the main objective of evaluating the performances under varying conditions of robustness and accuracy criteria. In particular, we analyzed the behavior of ZFNet in terms of mean-time-to-failure (MTTF), at different conditions of injected SET pulses and accuracy. The MTTF, which is the mean time between two faulty outcomes of the network, has been measured for all the evaluated SET pulses and in two different accuracy conditions at 95% and 85%. The obtained results are reported in Fig. 17.

The experimental analysis shows how the proposed ZFNet solution has an almost constant MTTF at different SET pulse conditions and with different accuracy criteria. It is interesting to notice that, the 10% accuracy difference slightly improves the final MTTF.

VI. RELATED WORK

Recently, several implementations of CNN architectures on FPGAs have been proposed. In particular, it has been demonstrated in [3] that is possible to achieve a complete design

flow for mapping CNN on FPGA. The FPGA reconfigurability provides the advantage of adapting their design to the CNN inference models without requiring significant modification of the hardware architecture [14]. Besides, FPGAs achieve extensive computational parallelism, enabling the usage of depthwise separable convolution instead of standard convolution CNN, reducing the number of used MAC modules [15]. Since convolutional computation requires many DSPs, a recent solution investigated the adoption of a CNN-optimized systolic array for improving the CNN working frequency on FPGAs [16]. Several solutions based on a sequential computation on a single layer per computation time have been provided; these methods require a sequential connection between layers at each computation time [17]. Even if these methods can exploit the reconfiguration capability of FPGAs, they have a drastic limit in the hardware resources considering the acceleration goal. Furthermore, these approaches cannot be applied in aerospace applications when the reliability requirements require the adoption of flash-based FPGA technologies that do not support dynamic reconfiguration. Considering the performance improvement, CNN architecture based on sequential computation suffers from scalability problems since the designed layers are generally not reconfigurable and each layer must be completely redesigned in case of network architectural modification.

Another issue is represented by the feature maps that typically contain tensors and neuron weights. Since they are stored within on-chip memory, sequential architecture is not applicable for deeper CNNs due to the limited availability of block RAMs on FPGAs [18]. A study proposing the evaluation of traditional selective redundancy techniques has been proposed in [19] showing the drastic degradation in terms of area and performance when mitigation approaches, such as TMR, are adopted. However, it is a known issue that the redundancy implementation of CNN on FPGAs would require a large amount of logic and memory resources. Therefore, the adoption of FPGA devices that are immune to radiation-induced changes in the configuration is a major need.

In the present work, we designed the CNN considering a nontimed model and adopting a bit-width parametric architecture. The CNN has been mapped for the first time on a rad-hard device that is immune to transient radiation effects within the configuration memory. Thanks to the robustness of the rad-hard FPGA, the circuit does not require scrubbing or reconfiguration of the FPGA in order to mitigate changes in the configuration due to radiation particles. Therefore, we focus the mitigation exclusively on avoiding transient pulse corrupting the CNN functionalities.

VII. Conclusion

We propose a physical implementation methodology of the ZFNet CNN on radiation tolerant flash-based FPGAs. The developed solution represents the first implementation of CNN on a radiation-hardened FPGA device. Furthermore, we evaluated the robustness capability and the performance characteristics of the network considering different types of mitigation solutions and configurations. The experimental result shows that a placement algorithm combining DSP and LSRAM routing optimization is the best tradeoff between accuracy, resiliency, and performance. In future works, we intend to evaluate the impact on SRAM-based radiation-tolerant FPGAs and to compare the implementation tool also considering the synthesis and the mapping phases.

References

[1] M. Imani, D. Peroni, Y. Kim, A. Rahimi, and T. Rosing, "Efficient neural network acceleration on GPGPU using content addressable memory," in *Proc. Des., Autom. Test Eur. Conf. Exhib. (DATE)*, 2017, pp. 1026–1031.

[2] O. Segal, N. Nasiri, M. Margala, and W. Vanderbauwhede, "High level programming of FPGAs for HPC and data centric applications," in *Proc. IEEE High Perform. Extreme Comput. Conf. (HPEC)*, Waltham, MA, USA, 2014, pp. 1–3.

[3] K. Guo et al., "Angel-Eye: A complete design flow for mapping CNN onto embedded FPGA," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 37, no. 1, pp. 35–47, Jan. 2018.

[4] R. Ma et al., "Specializing FGPU for persistent deep learning," in *Proc. 29th Int. Conf. Field Program. Logic Appl. (FPL)*, Barcelona, Spain, 2019, pp. 326–333.

[5] J. E. R. Condia, B. Du, M. Sonza Reorda, and L. Sterpone, "FlexGripPlus: An improved GPGPU model to support reliability analysis," *Microelectron. Rel.*, vol. 109, Jun. 2020, Art. no. 113660.

[6] D. Gschwend, "ZynqNet: An FPGA-accelerated embedded convolutional neural network," in *Proc. Comput. Vis. Pattern Recognit.*, 2020, p. 85.

[7] M. Wirthlin, "High-reliability FPGA-based systems: Space, high-energy physics, and beyond," *Proc. IEEE*, vol. 103, no. 3, pp. 379–389, Mar. 2015.

[8] C. De Sio, S. Azimi, and L. Sterpone, "FireNN: Neural networks reliability evaluation on hybrid platforms," *IEEE Trans. Emerg. Topics Comput.*, vol. 10, no. 2, pp. 549–563, Apr.–Jun. 2022, doi: 10.1109/TETC.2022.3152668.

[9] C. De Sio, S. Azimi, and L. Sterpone, "An emulation platform for evaluating the reliability of deep neural networks," in *Proc. IEEE Int. Symp. Defect Fault Toler. VLSI Nanotechnol. Syst. (DFT)*, 2020, pp. 1–4, doi: 10.1109/DFT50435.2020.9250872.

[10] N. Lusardi, F. Garzetti, N. Corna, R. D. Marco, and A. Geraci, "Very high-performance 24-channels time-to-digital converter in Xilinx 20-nm Kintex UltraScale FPGA," in *Proc. IEEE Nucl. Sci. Symp. Med. Imag. Conf.*, Manchester, U.K, 2019, pp. 1–4.

[11] N. Rezzak, J. Wang, D. Dsilva, and N. Jat, "TID and SEE characterization of Microsemi's 4th generation radiation tolerant RTG4 flash-based FPGA," in *Proc. IEEE Radiat. Eff. Data Workshop*, Boston, MA, USA, 2015, pp. 1–6.

[12] S. Azimi, B. Du, L. Sterpone, D. M. Codinachs, R. Grimoldi, and L. Cattaneo, "A new CAD tool for single event transient analysis and mitigation on flash-based FPGAs," *Integration*, vol. 67, pp. 73–81, Jul. 2019.

[13] M. D. Zeiler and R. Fergus, "Visualizing and understanding convolutional networks," in *Proc. Eur. Conf.*, vol. 8689, 2014, pp. 818–833.

[14] L. Chang, S. Zhang, H. Du, Y. Chen, and S. Wang, "A reconfigurable neural network processor with tile-grained multicore pipeline for object detection on FPGA," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 29, no. 11, pp. 1967–1980, Nov. 2021.

[15] L. Lu, Y. Liang, Q. Xiao, and S. Yan, "Evaluating fast algorithms for convolutional neural networks on FPGAs," in *Proc. 25th Annu. Int. Symp. Field-Program. Custom Comput. Mach. (FCCM)*, 2017, pp. 101–108.

[16] J. Zhang, W. Zhang, G. Luo, X. Wei, Y. Liang, and J. Cong, "Frequency improvement of systolic array-based CNNs on FPGAs," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, 2019, pp. 1–4.

[17] A. Boutros, B. Grady, M. Abbas, and P. Chow, "Build fast, trade fast: FPGA-based high-frequency trading using high-level synthesis," in *Proc. Int. Conf. ReConFigurable Comput. FPGAs*, Cancun, Mexico, 2017, pp. 1–6.

[18] M. Abusultan and S. P. Khatri, "Exploring static and dynamic flash-based FPGA design topologies," in *Proc. IEEE 34th Int. Conf. Comput. Design (ICCD)*, 2016, pp. 416–419.

[19] F. Libano et al., "Selective hardening for neural networks in FPGAs," *IEEE Trans. Nucl. Sci.*, vol. 66, no. 1, pp. 216–222, Jan. 2019.

[20] D. Derksen, G. Meoni, G. Lecuyer, A. Mergy, M. Martens, and D. Izzo, "Few-shot image classification challenge on-board OPS-SAT," in *Proc. 35th Conf. Neural Inf. Process. Syst. (NeurIPS 2021)*, 2021, pp. 1–3.
[21] L. Sterpone, N. Battezzati, F. L. Kastensmidt, and R. Chipana, "An analytical model of the propagation induced pulse broadening (PIPB) effects on single event transient in flash-based FPGAs," *IEEE Trans. Nucl. Sci.*, vol. 58, no. 5, pp. 2333–2340, Oct. 2011.
[22] R. Harada, Y. Mitsuyama, M. Hashimoto, and T. Onoye, "Impact of NBTI-induced pulse-width modulation on SET pulse-width measurement," *IEEE Trans. Nucl. Sci.*, vol. 60, no. 4, pp. 2630–2634, Aug. 2013.
[23] B. Du et al., "A novel propagation model for heavy-ions induced single event transients on 65nm flash-based FPGAs," in *Proc. 20th Eur. Conf. Radiat. Eff. Compon. Syst. (RADECS)*, 2020, pp. 1–4.
[24] L. Sterpone, N. Battezzati, and V. Ferlet-Cavrois, "Analysis of SET propagation in flash-based FPGAs by means of electrical pulse injection," *IEEE Trans. Nucl. Sci.*, vol. 57, no. 4, pp. 1820–1826, Aug. 2010.

Luca Sterpone (Senior Member, IEEE) received the M.S. and Ph.D. degrees in computer engineering from the Politecnico di Torino, Turin, Italy, in 2003 and 2007, respectively,

He is currently a Full Professor with the Department of Computer and Control Engineering, Politecnico di Torino. He has authored more than 220 papers. His current research interests include re-configurable computing, computer-aided design algorithms, fault tolerance architectures, and radiation effects on components and systems.

Prof. Sterpone received several awards for his research activities.

Sarah Azimi (Member, IEEE) received the Ph.D. degree from Politecnico di Torino, Turin, Italy, in 2019.

She is currently an Assistant Professor with the CAD and Reliability Group, Department of Computer and Control Engineering, Politecnico di Torino. Her research interests include fault-tolerant electronic design, intelligent informative systems, physical models, and validation platforms.

Corrado De Sio (Member, IEEE) received the B.S. and M.S. degrees in computer engineering from the University of Pisa, Pisa, Italy, in 2018. He is currently pursuing the Ph.D. degree with Politecnico di Torino, Turin, Italy.

He was a Research Assistant with the CAD and Reliability Group, Department of Computer and Control Engineering, Politecnico di Torino. His research interests include the reliability of reconfigurable devices, radiation effects, and soft errors.

Open Access funding provided by 'Politecnico di Torino' within the CRUI CARE Agreement