Design Techniques for Multi-Core Neural Network Accelerators on Radiation-Hardened FPGAs

# Design Techniques for Multi-Core Neural Network Accelerators on Radiation-Hardened FPGAs

Andrea Portaluri, Sarah Azimi and Luca Sterpone
*Dipartimento di Automatica e Informatica*
*Politecnico di Torino*
Turin, Italy

***Abstract*— Radiation-Hardened-By-Design (RHBD) FPGAs have gained a lot of attention thanks to their excellent compromise between costs and performance. Being of very limited use due to a lack of performance a few years ago, these devices are now capable of implementing a wide range of applications requiring high computational capabilities.**

**This work describes an implementation of a Very Long Instruction Word (VLIW) soft-core convolutional accelerator in the NanoXplore RHBD NG-Medium chip. Feasibility and timing performances have been analyzed in order to discover whether and how multi-core solutions can affect parallel acceleration. Placement also showed to heavily affect the delays, up to 70% more, based on the proximity to the output buffers.**

***Keywords—Radiation-hardened-by-design, SRAM-based FPGA, Convolution, Neural Network, Accelerator, Nanoxplore.***

## I. INTRODUCTION

In recent years, the computational power and data transfer capabilities required for modern deep space applications have critically grown. While rad-hard CPUs and controllers can no longer meet the requirements due to lack of performance, Radiation-Hardened-by-Design (RHBD) Application-Specific Integrated Circuits (ASICs) seem to achieve the best results, although being very expensive. On the other hand, SRAM-based Field-Programmable Gate Arrays (FPGAs) represent an attractive solution providing competitive performances, versatility, and low costs. These devices offer the possibility to reconfigure the hardware-implemented on them, opening a wide range of applications such as error correction or modification of parameters, while performing hardware-accelerated arithmetic, and data conversion, with low power consumption, compared to other devices. However, the major drawback of these devices is the susceptibility to radiation-induced soft errors. As a matter of fact, FPGAs are prone to corruption due to the interaction between high-energy particles such as protons, neutrons, and heavy ions and the Silicon bulk. Erroneous behaviors like bit-flips in configuration memory (i.e., Single-Event Upsets, SEUs) and transient pulses in the circuitry (i.e., Single-Event Transients, SETs) can arise and compromise the correctness of the application. Similarly to other architectures, FPGAs also have an RHBD counterpart that partially solves the issue making them immune to SEUs but still vulnerable to SETs. Therefore, RHBD FPGAs combine the re-programmability of SRAM-based devices with the robustness of space-grade layouts of the cells. Major vendors of FPGAs have their space-grade version, although this number is very limited, even smaller when looking for European chips. Currently, the main available options for high-performance space-grade FPGAs are Xilinx Virtex-5QV (SRAM, 65 nm) and RT Kintex UltraScale (SRAM, 20 nm), Microchip RTG4 (Flash, 65 nm) and RT PolarFire (SRAM, 28 nm). In this context, NanoXplore NG-Medium FPGA has been recently certified by the ECSS Standard as a space-grade component, reason that put this device in an excellent spot among the previous ones. This SRAM-based 65 nm RHBD chip includes space-specific features such as the SpaceWire interface and a scrubber module, together with the traditional programmable logic resources. Moreover, the recent rad-hard technology improvements, such as High-performances Block RAMs (BRAMs) and Digital Signal Processing (DSP) blocks, have unlocked a lot of potential applications, breaking through the performance limits of these devices. In fact, while CPUs and GPUs have fixed instruction set with rigid memory hierarchy, FPGAs grant flexible and fully customizable architecture. The latter can be adapted to the application, excelling at various types of parallelism such as bit manipulation, pipelined systems, and the necessity of wide datapaths. Among these, acceleration of Convolutional Neural Networks (CNNs) and soft processors might now be implemented in the programmable logic of most of these rad-hard FPGAs. These applications and mission-critical scenarios arise the necessity of performance assurance for RHBD devices now that it is no longer sufficient to rely on the robustness of the hardware alone.

In this paper, we discuss the feasibility of implementation and timing performance optimization of an accelerator for convolution running on the r-VEX Soft Processor. Assembly code for the VEX Instruction Set Architecture (ISA) has been developed in order to execute convolutional products, Rectified Linear Unit (ReLU) function activation, and Max Pooling. Multi-core solutions for parallel computations have been implemented alongside the single-core design in order to evaluate the trade-off between performances and the number of cores. The circuit has been implemented on the programmable logic of a NanoXplore NG-Medium chip.

The paper is structured as follows. Section II presents the related works on the Artificial Intelligent (AI) accelerator implementation on FPGAs and in particular RHBD FPGAs., while Section III gives a brief overview of the programmable logic of the NG-Medium device and the architecture of the r-VEX processor. Section IV describes the implementation of the accelerator in the soft processor and Section V gives details about the experimental analysis and results. Finally, Section VI draws conclusions and discusses future works.

## II. Related Works

Most of the studies available in the literature present data on cross-section and radiation test reports of soft error and Single Event Effects on RHBD FPGAs [1-4]. Other works describe the implementation of AI accelerators on these devices. The authors in [5] propose an AI architecture on RT Kintex UltraScale exploiting Xilinx AI processing units. In [6], the authors use the VectorBlox software development kit to deploy AI models on RT PolarFire. The work described in [7] compares different embedded devices running AI applications from the performance point of view where also a NanoXplore device can be found. Concerning the NanoXplore chips, in [8] the vendor claims a Single-Event Upset cross-section of about $1.00 \cdot 10^{-10}$ $cm^2$/bit with CMIC on. The Configuration Memory Integrity Check (CMIC) is the scrubber module of NanoXplore capable of reconfiguring frames of the configuration memory in case of corruption. The authors in [9] study the utilization of NG-Medium against high TID environments such as the CERN applications, while in [10] the authors evaluate the performances of the NG-Large chip heavily exploiting its DPS blocks.

To our knowledge, very few works have focused on the feasibility and implementation of Machine Learning accelerators for rad-hard FPGAs, especially targeting newer devices like NanoXplore's. This is most likely due to the unavailability of detailed data concerning placement and routing algorithms, architecture, and still poor customization through the CAD tool, compared to other major vendors.

## III. Background

### A. NanoXplore Programmable Logic Architecture

The NG-Medium chip by NanoXplore is a Radiation Hardened-by-Design (RHBD) SRAM-based FPGA manufactured with a 65nm technology process. This chip represents the first product of a fully fab-less European product chain with configuration memory cells and the circuit design based on STMicroelectronics technology, respectively STM C65 SPACE and RH65nm Skyrob Library, whose architecture is described in the following paragraphs [8]. The device's recent space-grade standards achievements made its applications and market exponentially increase and attention on this device have grown as well. The programmable logic architecture does not follow a traditional "matrix" layout, instead, a "cluster" solution is adopted. Three rows of arithmetic and combinatorial tile logic are interspersed with two rows of BRAMs and DSP blocks for two clock regions. Figure 1 presents the described arrangement.

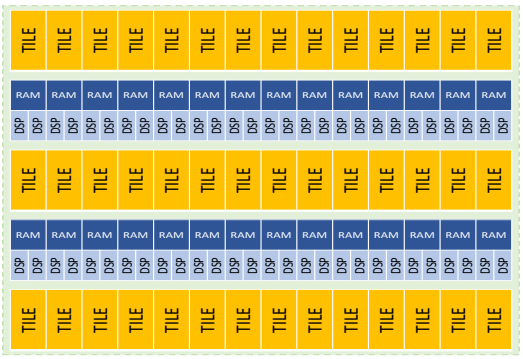


Fig. 1. Scheme of a Clock Region of the NX-Medium chip.

Within the single tile, Flip-Flops (FFs) and Look-Up Tables (LUTs) are coupled together in the so-called Functional Elements (FE). FEs are placed within Tiles (28 Tiles for each row, 14 per each Clock Region). Besides FEs, 2 rows of carry logic, a row of 24-input high-performances LUT (X-LUT), and a register file are also present. Currently, the placement of resources is quite constrained by the NanoXplore CAD tool *NXmap*. In fact, FEs locations cannot be arbitrarily assigned. The only degree of freedom is to choose the Tile, while the NXmap will place the resource in one of the 384 possible spots within it according to the embedded optimization algorithm. Each FE placement is then described by the tile and its 32-location sub-set, called Site (e.g., *lut_name TILE[2x6]:S5_149*). The distribution of the sites within the tile is shown in Figure 2. This constraint limits the possible mitigation approaches from the placement point of view. The routing details are also few at the moment but the vendor still identifies 4 different interconnections as follows:

- *Low skew network*: routing to provide a homogeneous distribution of global signals such as clocks and resets.
- *Direct interconnections*: routing between adjacent logic.
- *Logic internal routing*: routing into the same tile.
- *General routing resources*: general routing with longer propagation delay for inter-tile communication.


Fig. 2. Distribution of FEs and sites within a tile of the NG-Medium chip.

### B. r-VEX Soft Processor

The r-VEX processor is a Very Long Instruction Word (VLIW) soft core (i.e., implemented on programmable logic) designed by Delft University based on the VLIW EXample (VEX) ISA, introduced in [12] and loosely modeled upon the HP/STMicroelectronics LX/ST200 ISA of VLIW embedded cores. It offers a scalable technology, including instruction width, instruction set, and functional units written in VHDL. By default, a core has 4 Arithmetic and Logic Units (ALUs), 2 Multiply Unit Logics (MULs), 1 branch control unit, 1 memory access unit, 64 32-bit general purpose registers (GR), and 8 1-bit branch registers (BR). It also supports a 32 kB data and instruction memory cache. As a VLIW processor, each instruction includes (also parametrizable) 4 32-bit long syllables, fitting opcode bits, register addresses bits and meta-data bits in one syllable. These syllables can be seen as single RISC instructions. The VEX standard specifies the utilization

of three immediate operand types, namely short immediate operands, branch offset immediate operands, and long immediate operands. Each syllable in the VEX standard has a switch field for immediate that comprises 2 bits that identify the type of immediate operand within the syllable. Syllables in r-VEX also have additional meta-data bits, L and F, where the L bit indicates whether the syllable is the final syllable in an instruction, and the F bit signifies whether it is the first syllable in an instruction. The default configuration of the r-VEX and the syllable layouts are shown in Figure 3. The standard set of instructions includes 73 operations and the possibility to extend it through custom instructions. Data and instructions can be edited directly in the respective memories by modifying the hardware description files associated with them. After the traditional design flow (synthesis, implementation, and bitstream generation), the design can be implemented in the FPGA.

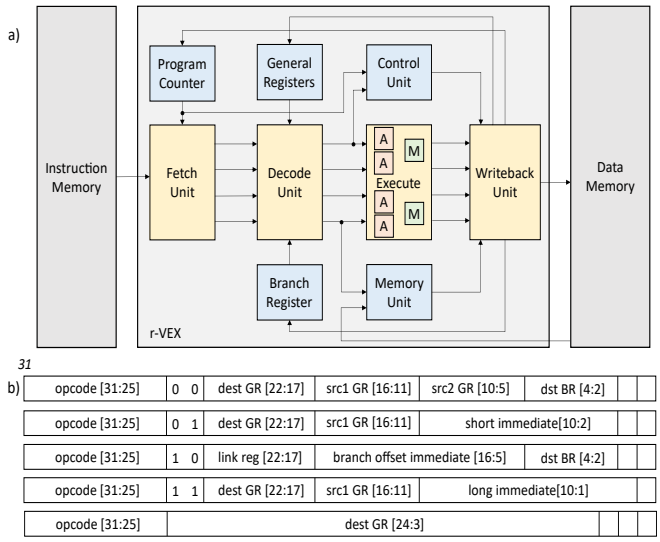


Fig. 3. a) Scheme of r-VEX architecture. b) Different instructions supported by the VEX ISA.

## IV. Convolution in VEX Assembly

In order to have a benchmark that could heavily exploit the computational capabilities of the board, the choice for the application to run in the r-VEX processor was to implement a convolutional accelerator of a reduced AlexNet architecture. AlexNet is a very efficient CNN for handwritten character recognition containing layers of 2D convolution, Max Pooling, and ReLU activation [13]. A squared input image is fed into a series of convolution products, Max Pooling, and activation in order to return a probability of that image being a digit between 0 and 9. Since the data memory of the r-VEX is limited, only a sub-section of the entire network has been implemented. However, we took care that this section could contain at least one of the different layers. As shown in Figure 4, the implemented computation includes 2 convolutional layers interposed with 2 Max Pool layers. In particular, the parameters of the slice of the network slice implemented are reported in Table I.

Since the r-VEX must be programmed in assembly language, code for the convolution in VEX instruction set has been developed. The assembly has been written exploiting the branch registers for comparing the values of rows and columns. In doing this, a parametrizable code has been developed. Therefore, stride and padding can be modified by changing the values in the code without modifying the body of the code itself.

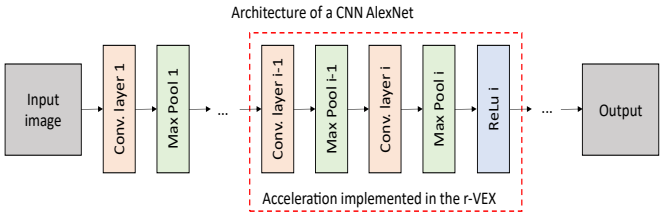


Fig. 4. Architecture of the AlexNet and its sub-section selected for the acceleration through the r-VEX.

TABLE I
LAYER PARAMETERS

| Layer | Input | Stride | Padding | Filter | Output |
|---|---|---|---|---|---|
| *Conv. Layer 1* | 12 x 12 | 1 | 1 | 3 x 3 | 12 x 12 |
| *Max Pool 1* | 12 x 12 | 0 | 1 | 2 x 2 | 11 x 11 |
| *Conv. Layer 2* | 11 x 11 | 0 | 1 | 3 x 3 | 9 x 9 |
| *Max Pool 2* | 9 x 9 | 0 | 1 | 2 x 2 | 8 x 8 |

**Algorithm 1** Algorithm for convolution, Max Pool and ReLu in VEX assembly.

**Input:** *image[N][N]*
*Load image from Data Memory*:
**while** *branch_col_register* **do**
  **while** *branch_row_register* **do**
    `mov` *image[i][j], gen_register j+N*i*
    `add` *1, row_register*
    `cmpeq` *1, branch_row_register*
  `add` *1, col_register*
  `cmpeq` *1, branch_col_register*
`goto` *Convolution*
*Convolution*:
**while** *branch_col_register* **do**
  **while** *branch_row_register* **do**
    execute convolution in *row_register, col_register*
    `add` *1, row_register*
    `cmpeq` *1, branch_row_register*
  `add` *1, col_register*
  `cmpeq` *1, branch_col_register*
`goto` *Max Pool*
*Max Pool:*
**while** *branch_col_register* **do**
  **while** *branch_row_register* **do**
    `max` *gen_register i, gen_register i+1*
    `add` *1, row_register*
    `cmpeq` *1, branch_row_register*
  `add` *1, col_register*
  `cmpeq` *1, branch_col_register*
`goto` *ReLU*
*ReLu:*
**while** *branch_col_register* **do**
  **while** *branch_row_register* **do**
    `max` *gen_register i, 0*
    `add` *1, row_register*
    `cmpeq` *1, branch_row_register*
  `add` *1, col_register*
  `cmpeq` *1, branch_col_register*

Algorithm 1. Assembly pseudo-code for the implemented convolutional steps.

The algorithm developed is presented in Algorithm 1. First, the input image has been uploaded from the data memory to the registers in order to minimize further accesses, then various functions for the different layers have been developed and pointed through the *goto* opcode in case of need. In the implemented configuration, the computation involved a 12-by-12 input image, producing an 8-by-8 output.

## V. Experimental Analysis and Results

In order to be able to discuss the performances of the implemented accelerator, analyses have been carried out on the design. Three different implementations of the benchmark varying the number of cores alongside the plain design (namely, *1-core*, *2-core* and *3-core*) have been analyzed with the goal of finding an optimal trade-off between performances and the number of computing cores. Firstly, a detailed description of the resource utilization and mapping directives has been presented by exploiting the NXmap design toolchain. Secondly, a timing evaluation through Static Timing Analysis has been carried out on the designs. The next subsections discuss the details and the obtained results.

### A. Hardware Benchmark

First, the reconfigurable hardware parameters of the r-VEX have been chosen. In particular, the number of registers, syllable issues, and computational units have been maximized according to the r-VEX version we used. The implementation details are shown in Table II. These values remain constant for all the cores in the designs.

TABLE II
IMPLEMENTED R-VEX CONFIGURABLE PARAMETERS

| | |
|---|---|
| **General Registers [#]** | 64 |
| **Branch Registers [#]** | 8 |
| **Arithmetic Logic Unit [#]** | 4 |
| **Multiplier [#]** | 2 |
| **Syllable-Issues [#]** | 4 |
| **Data Memory [kB]** | 32 |

For the implementation of the circuit, the NanoXplore toolchain has been used. First, the NXmap tool has synthesized, routed, and placed the design. In the Synthesis phase, the correct completion of the steps is required to add mapping directives for the design. These options allow the designer to map instances on a targeted resource rather than the default one e.g., a multi-R/W port RAM cannot be managed by the tool and has to be mapped as DFF. Moreover, the placement of resources has been entirely managed by the tool without user intervention or customized directives. From the tool, several reports on the timing and resource utilization of the device can be extracted, as well as a graphical representation of the implemented circuit in the programmable logic. This interface helps the designer through the use of the available placement options, highlights used instances and routing paths, and gives a summary visualization of the resource utilization, as shown in Figure 5.

The last step of the design flow generates the bitstream, an array of binary data that programs the resources of the FPGA according to the implemented design. Eventually, the bitstream is downloaded into the Brave NG-Medium NX1H35S CLGA625 DevKit using the NXbase software. Figure 6 summarizes the steps.

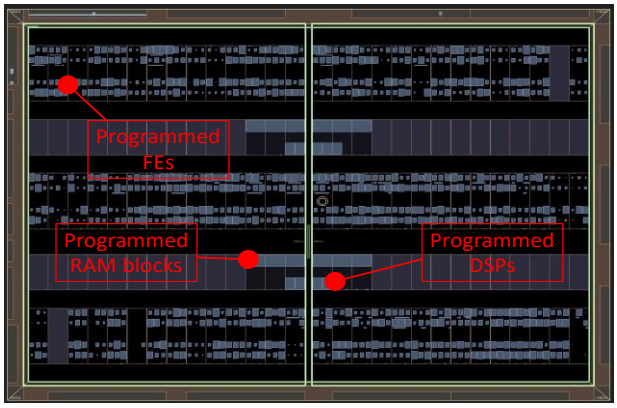


Fig. 5. Programmable logic view after place and routing in the NXmap tool. Lighter blue squares represent the programmed resources. In the example, the *3-core* circuit.

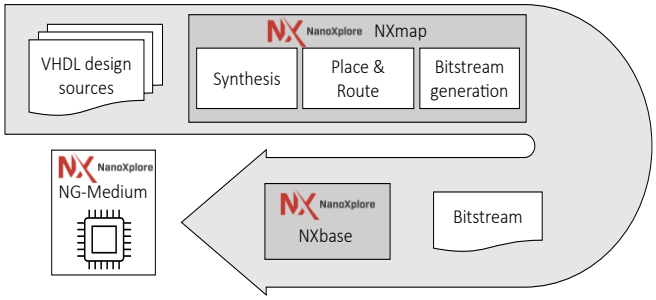


Fig. 6. NanoXplore design flow from the input of design sources to the bitstream downloading.

From the NXmap reports, the utilization of resources for the three designs has been extracted and presented in Table III with respect to the Brave NG-Medium NX1H35S device. It can be seen how the three configurations follow a resource utilization proportional to the number of cores implemented.

TABLE III
RESOURCES UTILIZATION OF EACH DESIGN

| Design | 4-LUT | DFF | Carry Logic | DSP | BRAM |
|---|---|---|---|---|---|
| *1-core* | 6,098 (19%) | 1,901 (6%) | 298 (4%) | 3 (3%) | 4 (8%) |
| *2-core* | 11,714 (37%) | 3,801 (12%) | 596 (8%) | 6 (6%) | 8 (15%) |
| *3-core* | 17,946 (56%) | 5,703 (18%) | 894 (12%) | 9 (9%) | 12 (22%) |

Additionally, the Brave NX-Medium NX1H35S chip does not have an embedded Universal Asynchronous Receiver-Trasmitter (UART) controller. In order to visualize the output data from the r-VEX, a configurable UART controller has been implemented alongside the circuit. The output pin has been mapped to a 3.3 V programmable pin of the board and an external USB-to-UART serial converter module has been used to transmit the data to the host PC with a 115,200 baud rate. Each benchmark uses the 25 MHz clock provided by the oscillator on the board as the maximum frequency. For the performance analysis, some reports have been extracted from the NXmap tool, in particular the Static Timing Analysis (STA) netlist and the routed VHDL netlist.

The following subsections give details about data and how they have been exploited.

### B. Transitions Delay

NXmap is capable of deploying the Standard Delay Format (SDF), which is an IEEE specification that represents circuit delays in ASCII format. In the most common SDF files, delays of $0 \rightarrow 1$ and $1 \rightarrow 0$ transitions are reported for the networks of the circuit although others can have up to 12 delay values (i.e., corresponding to all the possible transitions among 0, 1, Z, and X states). For the timing analysis, the average of the maximum and minimum values for the $0 \rightarrow 1$ and $1 \rightarrow 0$ are computed for each design in order to show the timing performances of the three. In particular, we divided timing results according to the type of interconnection described in Section III.B by cross-checking the SDF report with the routed netlist. Therefore, transition timing data about intra-tile resources (i.e., *internal routing resources*), inter-tile (i.e., *general routing resources*), and, global routing (i.e., clock and reset signals, *low-skew resources*) have been collected and shown in Table IV, V and, VI, respectively. As can be seen from the tables, transition timing seems to increase as the resource placement spreads in the programmable logic.

TABLE IV
AVERAGE TRANSITIONS TIMING OF INTERNAL ROUTING RESOURCES

| Design | $0 \rightarrow 1_{min}$ [ns] | $0 \rightarrow 1_{max}$ [ns] | $1 \rightarrow 0_{min}$ [ns] | $1 \rightarrow 0_{max}$ [ns] |
|---|---|---|---|---|
| *1-core* | 2.278 | 2.378 | 2.278 | 2.378 |
| *2-core* | 2.497 | 2.601 | 2.497 | 2.601 |
| *3-core* | 2.537 | 2.642 | 2.537 | 2.642 |

TABLE V
AVERAGE TRANSITIONS TIMING OF GENERAL ROUTING RESOURCES

| Design | $0 \rightarrow 1_{min}$ [ns] | $0 \rightarrow 1_{max}$ [ns] | $1 \rightarrow 0_{min}$ [ns] | $1 \rightarrow 0_{max}$ [ns] |
|---|---|---|---|---|
| *1-core* | 2.742 | 2.827 | 2.742 | 2.827 |
| *2-core* | 2.801 | 2.922 | 2.801 | 2.922 |
| *3-core* | 2.905 | 2.985 | 2.905 | 2.985 |

TABLE VI
AVERAGE TRANSITIONS TIMING OF LOW SKEW ROUTING RESOURCES

| Design | $0 \rightarrow 1_{min}$ [ns] | $0 \rightarrow 1_{max}$ [ns] | $1 \rightarrow 0_{min}$ [ns] | $1 \rightarrow 0_{max}$ [ns] |
|---|---|---|---|---|
| *1-core* | 6.605 | 6.667 | 6.605 | 6.667 |
| *2-core* | 6.587 | 6.646 | 6.587 | 6.646 |
| *3-core* | 6.683 | 6.743 | 6.683 | 6.743 |

As a matter of fact, the *1-core* design obtained the lowest delay values among the others. However, the highest difference among these has been registered in the internal routing being around $0.3 \cdot 10^{-9}$ s, which is the 0.75% of a clock period, therefore totally negligible delay. The *2*-core design is the only exception with a slightly lower delay value concerning the low skew network which is due to a more even distribution of resources around the clock and reset sources compared to the other designs.

### C. Data Delay

The STA report is a way of analyzing the timing performances of a circuit. STA breaks the design down into timing paths and computes the signal propagation delay along them, allowing to verify also timing violation. Each timing path consists of a start point, an endpoint, and the combinational logic between them. In this context, the main reason for this further analysis is the observation of the timing performance modification as the number of cores implemented increased. The STA has been performed in *typical* scenarios for the three designs and the data delay of the 10 longest paths of the *1-core* design has been compared with the same path of the other designs. The results are plotted in Figure 7.

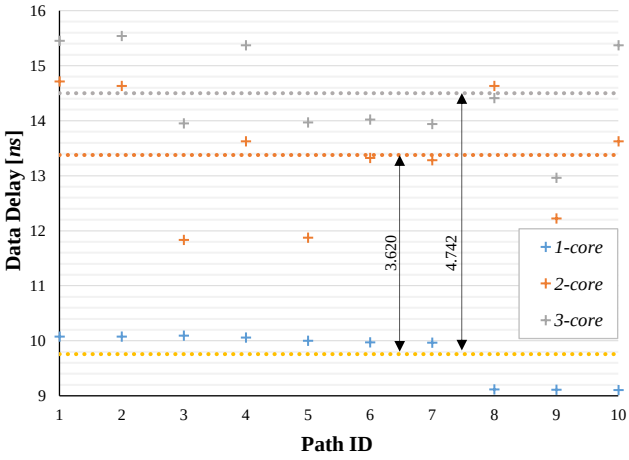


Fig. 7. Plot of the data delay of the 10 longest paths of the *1-core* design compared with the same paths of the other design. The dotted lines represent the average values.

As can be seen from the data above, data delays are clearly dependent on the resource utilization of the device. In particular, the displacement of the 2-core and 3-core average values from the 1-core one is 3.620 ns (+37.10%) and 4.742 (+48.59%), respectively. However, this shift appears to not be directly proportional to the increase in utilization. Max working frequency has been also affected by the placement congestion: data are presented in Table VII.

TABLE VII
MAXIMUM WORKING FREQUENCY OF EACH DESIGN

| Design | Frequency [MHz] |
|---|---|
| *1-core* | 32.753 |
| *2-core* | 27.586 |
| *3-core* | 23.384 |

The NXmap tool, like other CAD tools for FPGA design, offers the possibility to create and edit a timing constraint file. In this file, constraints on nets and timing paths can be written and, unless raising a timing violation flag, eventually applied to the design. For our designs, timing constraints have been exploited trying to increase the performances with respect to

the data shown in Figure 7. The constraints involved the same 10 paths analysed above and focused on the *2-core* and *3-core* designs. A constraint iteration of 0.05 ns on the maximum data delay has been applied until reaching the timing violation for each path. Figure 8 shows the obtained results.

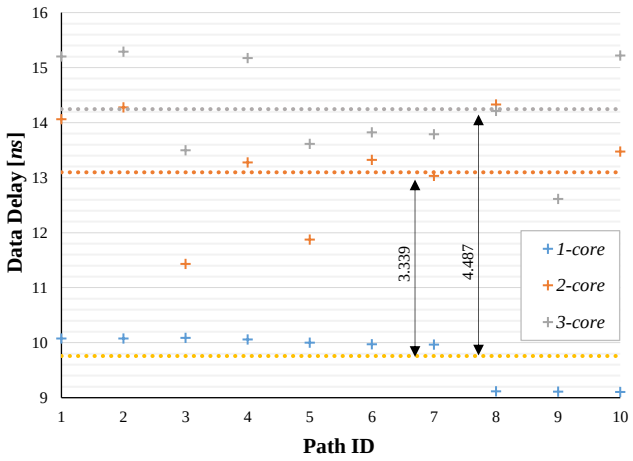


Fig. 8. Plot of the data delay of the 10 longest paths of the *1-core* design compared with the same paths of the other design with timing constraints. The dotted lines represent the average values.

Through the timing constraint file, all the 10 paths' data delays have decreased. Thus, a small improvement has been registered for both *2-core* and *3-core* designs whose average delay difference from *1-core*'s is now equal to 3.339 ns (+34.21%) and 4.487 ns (+45.98%), respectively. However, no improvement concerning the working frequencies has been registered.

### *D. Placement Constrained Data Delay*

In a common hardware-accelerated application, FPGA programmable logic is often densely populated with interconnections, memories, processors, and additional logic that manages the data flow to and from the accelerator. Therefore, the resources dedicated to accelerated computations are constrained in congested sub-sections of the configurable logic of the device. These placement constraints can also affect timing since proximity to clock and reset sources as well as output pin interfaces may vary. In our analysis, placement constraints on the cores have been applied in order to mimic the placement congestion of the logic. Through the NXmap tool, regions have been created and edited to which the cores have been assigned. After some trials, the minimum size of the region for containing a single core has been found to be equal to *11×6* (i.e., 11 rows and 6 columns) according to the NXmap programmable logic coordinate system. Two layouts (namely, L1 and L2) for each design have been proposed and analyzed, where the distance from the I/O buffers, located on the top-left side of the logic, have changed. L1 layouts are united by a higher proximity to the top-left corner, whereas L2 ones try to maximize this distance. Figure 9 shows the proposed layouts.

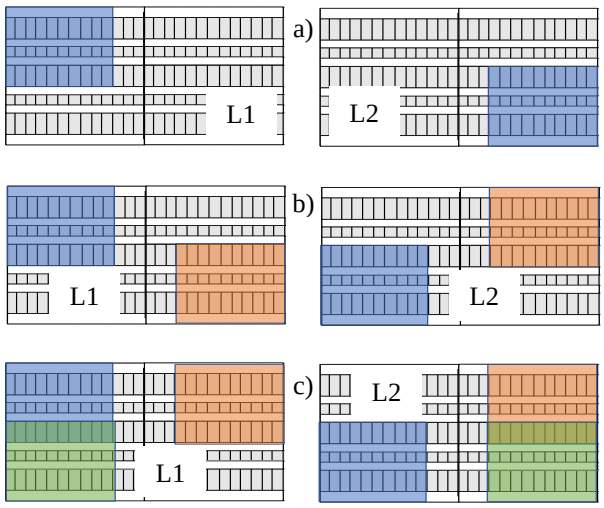


Fig. 9. Cores layouts analyzed for *a) 1-core b) 2-core c) 3-core*. Each colored block represents a constrained region for a single core.

Notice that, in the *3-core* layouts, two regions overlapped due to the unavailability of enough resources. STA analysis for each layout has been performed and data on timing delay collected and plotted in a box plot, shown in Figure 10. The data refer to the same 10 paths analyzed previously.

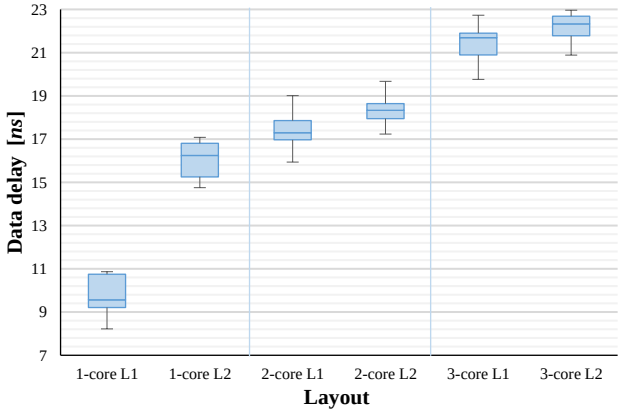


Fig. 10. Plot of the data delay of the 10 paths analyzed in Section V.C of the constrained layouts. L1 versions minimize the distance from the output buffer, while L2 maximizes it.

Overall, data show that an increasing distance from the top-left corner (i.e., where the output buffers are located) seems to deteriorate timing performances. In particular, the worst difference has been registered by the *1-core* layouts (around 6 ns), where the distance from the buffers is the highest. As a matter of fact, the degradation of performances in the other L2 layouts appears to be less since routing paths to the output are shorter. Therefore, timing-sensitive modules such as accelerators are widely placement dependent in terms of performance. Timing constraints may not be enough to fill the gap as shown previously, so careful analyses are necessary to fully exploit hardware acceleration in presence of a congested programmable logic. Finally, resource utilization shows no significant increment with the placement constraints active.

### E. VLIW Robustness Considerations

From the VLIW side, a few considerations on the performance vs. reliability trade-off must be taken into account. As a matter of fact, whenever a syllable is executed, all the instructions contained in it are run in parallel, virtually making the computation n-time faster, where *n* represents the number of instructions in a syllable [15]. On the other hand, a higher number of syllables (thus, a lower number of instructions per syllable) might help the filtering of transient faults since values in the registers are called at every computation like a sort of recovery checkpoint. In the proposed implementation, only 16% of the syllables are not full as we preferred to go with a performant computation. This value is due to unavoidable write conflicts since more instructions cannot write in the same register within the same syllable, thus needing to fill the issue with another no-conflict instruction or a *do nothing* opcode.

## VI. Conclusions and Future Works

As the deep space mission computational requirements become more challenging over time goes, RHBD FPGAs have witnessed an increase in attention due to the combination of versatility and reliability. In fact, the intrinsic reliability feature of these devices can guarantee the correct completion of mission-critical tasks when performing in harsh environments.

In this paper, we implemented several solutions, both single and multi-core, of a convolutional accelerator in the r-VEX soft processor on a NanoXplore NG-Medium device and evaluated timing performances. Collected data show that the placement location of the cores plays a key role in the timing performances, which can be seriously affected and timing constraints are often not enough to fill this gap.

As future works, we are investigating the placement optimization of NanoXplore devices and developing a tool to integrate with the NanoXplore toolchain to customize it according to reliability-driven placement algorithms.

## VII. Acknowledgement

This publication is part of the project NODES which has received funding from the MUR – M4C2 1.5 of PNRR with grant agreement no. ECS00000036.

## References

[1] Á. B. de Oliveira *et al*., "Analyzing the Influence of using Reconfiguration Memory Scrubber and Hardware Redundancy in a Radiation Hardened FPGA under Heavy Ions," *2018 18th European Conference on Radiation and Its Effects on Components and Systems (RADECS)*, Goteborg, Sweden, 2018, pp. 1-4, doi: 10.1109/RADECS45761.2018.9328683.

[2] G. Mantelet, M. Briet, G. Rouxel, S. Hachad, B. Bancelin and D. de saint Roman, "ATMEL ATF280E rad hard SRAM based reprogrammable FPGA SEE test results," *2009 European Conference on Radiation and Its Effects on Components and Systems*, Brugge, Belgium, 2009, pp. 606-608, doi: 10.1109/RADECS.2009.5994730.

[3] P. R. C. Villa *et al*., "Analysis of single-event upsets in a Microsemi ProAsic3E FPGA," *2017 18th IEEE Latin American Test Symposium (LATS)*, Bogota, Colombia, 2017, pp. 1-4, doi: 10.1109/LATW.2017.7906772.

[4] N. Battezzati, F. Decuzzi, M. Violante and M. Briet, "Application-oriented SEU sensitiveness analysis of Atmel rad-hard FPGAs," *2009 15th IEEE International On-Line Testing Symposium*, Lisbon, Portugal, 2009, pp. 89-94, doi: 10.1109/IOLTS.2009.5195988.

[5] J. Vidmar, P. Maillard, T. Jones, M. Sawant, G. Gambardella, and N. Fraser, "Space DPU: Constructing a radiation-tolerant, FPGA-based platform fordeep learning acceleration on space payloads," in *Proc. Eur. Workshop Board Data Process*. (OBDP), 2021, pp. 1–8.

[6] K. O'Neill, A. Severance, and D. Nandi, ''Using the VectorBlox software development kit to create programmable AI/ML applications in radiation-tolerant RT PolarFire FPGAs,'' in *Proc. Eur. Workshop Board Data Process*. (OBDP), 2021, pp. 1–8.

[7] V. Leon, G. Lentaris, D. Soudris, S. Vellas and M. Bernou, "Towards Employing FPGA and ASIP Acceleration to Enable Onboard AI/ML in Space Applications," *2022 IFIP/IEEE 30th International Conference on Very Large Scale Integration (VLSI-SoC)*, Patras, Greece, 2022, pp. 1-4, doi: 10.1109/VLSI-SoC54400.2022.9939566.

[8] NanoXplore SAS, "From Radiation Hardening to BRAVE FPGA devices," RADiation and reliability challenges for electronics used in Space, Avionics, on the Ground and at Accelerators (RADSAGA), Geneve, Switzerland, 2017.

[9] G. Tsiligiannis, C. Debarge, J. Le Mauff, A. Masi and S. Danzeca, "Reliability analysis of a 65nm Rad-Hard SRAM-Based FPGA for CERN applications," *2019 19th European Conference on Radiation and Its Effects on Components and Systems (RADECS)*, Montpellier, France, 2019, pp. 1-8, doi: 10.1109/RADECS47380.2019.9745713.

[10] V. Leon *et al*., "Development and Testing on the European Space-Grade BRAVE FPGAs: Evaluation of NG-Large Using High-Performance DSP Benchmarks," in *IEEE Access*, vol. 9, pp. 131877-131892, 2021, doi: 10.1109/ACCESS.2021.3114502.

[11] C. De Sio, S. Azimi, L. Bozzoli, B. Du and L.Sterpone, "Radiation-induced Single Event Transient effects during the reconfiguration process of SRAM-based FPGAs," in Microelectronics Reliability, vol. 100-101, 2019, doi: 10.1016/j.microrel.2019.06.034.

[12] J. A. Fisher, "Retrospective: very long instruction word archtectures and the ELI-512," in *IEEE Solid-State Circuits Magazine*, vol. 1, no. 2, pp. 34-36, Spring 2009, doi: 10.1109/MSSC.2009.932941.

[13] A. Krizhevsky, I. Sutskever and G. E. Hinton, "ImageNet classification with deep convolutional neural networks" in Commun. ACM, vol. 60, no. 2, pp. 84-90, doi: 10.1145/3065386.

[14] S. Azimi, B. Du and L. Sterpone, "Accurate analysis of SET effects on Flash-based FPGA System-on-a-Chip for satellite applications," *2016 16th European Conference on Radiation and Its Effects on Components and Systems (RADECS)*, Bremen, Germany, 2016, pp. 1-4, doi: 10.1109/RADECS.2016.8093203.

[15] S. Wong**,** T. van As**,** G. Brown**,** "ρ-VEX: A Reconfigurable and Extensible Softcore VLIW Processor**,"** in *International Conference on Field-Programmable Technology (ICFPT 2008)*, December 2008.