Impact of High-Level-Synthesis on Reliability of Neural Network Hardware Accelerators

Marcello Traiola, Fernando Fernandes Dos Santos, Olivier Sentieys, Angeliki Kritikakou

Impact of High-Level-Synthesis on Reliability of Neural Network Hardware Accelerators

Marcello Traiola, Fernando Fernandes dos Santos, Olivier Sentieys, and Angeliki Kritikakou

Abstract

We characterize the impact of High-Level Synthesis (HLS) on the reliability of Neural Networks on FPGAs exposed to neutron. Our results show that the larger the circuit generated by HLS, the larger the error rate.

Corresponding and Presenting Author: Marcello Traiola is with Institut national de recherche en informatique et en automatique (INRIA), France Campus de Beaulieu, 263 Av. Général Leclerc, 35042 Rennes, France, +33299847218.

Contributing Authors:

Fernando Fernandes dos Santos, Angeliki Kritikakou, and Olivier Sentieys are with Institut national de recherche en informatique et en automatique (INRIA), France.

Session Preference: SEE: Devices and ICs

Presentation Preference: Oral

I. Introduction

Neural Networks (NNs) are currently one of the most intensively and widely used predictive models in the field of Machine Learning (ML) [1]. NNs have proven to give very good results for many complex tasks and applications, such as object recognition in images/videos, natural language processing, satellite image recognition, robotics, aerospace, smart healthcare, and autonomous driving. Their incredibly high effectiveness comes at the price of an extremely large algorithmic complexity, thus requiring enormous computational power. To support energy-intensive data movement, speed of computation, and large memory resources required by AI to achieve its full potential, custom and embedded Artificial Intelligence HardWare architectures (AI-HW) are required [2].

However, the increased complexity of modern systems makes low level hardware design approaches less effective [3]. To mitigate the design complexity, high-level specification languages are used to design hardware circuits in higher abstraction. With the help of High-Level Synthesis (HLS) tools, such high-level descriptions can be interpreted to create digital hardware which implements the same functionality. By using high-level specification languages, such as C and C++, and HLs tools, the circuit design becomes less complex – compared to Hardware Design Languages (HDL) implementations [4] – as the circuit can easily be modified, expanded, and verified, speeding up significantly the hardware design process. As a result, custom and complex NN accelerators running on FPGAs can be designed faster with HLS.

When AI-HW are employed in safety-critical systems, reliability constraints become of paramount importance. Testing AI-HW under high-energy neutrons plays a key role in providing information about the hardware error model and the impact on safety-critical systems. Recent studies have evaluated the reliability of machine learning running on accelerators and FPGAs on high-energy neutron beams and have demonstrated that NN-based systems do not meet the reliability criteria for safety-critical systems [5]–[8]. However, to the best of our knowledge, no existing work has evaluated how the HLS tools can impact the reliability of obtained AI-HW on FPGAs.

To address this limitation, this work is the first to evaluate the reliability of NN accelerators generated by HLS tools running on FPGAs under high-energy neutrons, and explore the impact of the HLS tools on reliability. More precisely, we use HLS4ML [9], [10] to create different implementations of the same machine-learning algorithm using HLS, and then, test their resilience properties against high-energy neutrons. In our tests, while the devices are being irradiated, we ran NNs and checked the mismatches in the application output and the execution cycles. To study the effects and propagation of faults, we perform post-processing in order to identify the faults that are more likely to reduce a NN's reliability and make the system miss the critical constraints.

II. Design of NN hardware accelerators

The HLS design flow takes as input a high-level description of the functionality, an RTL component library, and a set of specific design constraints. As illustrated in Figure 1, HLS initially compiles the functional specification to a formal model, such as the Control and Data Flow (CDFG) graph that describes the data and control dependencies between the operations. Then, it allocates hardware resources, such as functional units, storage components, buses, etc., it schedules the operations to clock cycles, binds the operations to functional units, binds variables to storage elements, binds transfers to buses, and generates the RTL architecture [11]. Then, digital implementation takes place, including logic synthesis and place and route. The aforementioned steps are usually coupled with each other, e.g., when fewer hardware resources are allocated, area is saved, but this decision leads to a longer schedule. However, thanks to HLS, the hardware parallelism degree can be easily tuned, leading to implementations with different trade-offs between area footprint (e.g., occupied FPGA logic slices and LUTs) and latency.

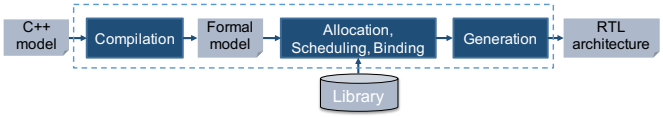

Fig. 1: HLS's design flow. The process starts with high-level language input (i.e., C++) with a standard compilation procedure. Then the tool will generate, based on the constraints and goals of the circuit, an architecture (RTL) ready to be synthesized into the platform.

As our case study, we use the HLS4ML framework to translate well-established open-source machine learning models into HLS-compatible high-level description models. HLS4ML is an open-source software-hardware codesign workflow to interpret and translate machine learning algorithms for FPGA implementation. Figure 2 sketches HLS4ML flow. Such models are configurable in terms of hardware parallelism and data width [9]. HLS4ML allow us to quickly deploy different configurations of NN accelerators on FPGA. More precisely, we design NN hardware accelerators for Multi-Layer Perceptrion (MLP) classifier for 28×28 pixel images, composed as follows: (i) 784 input neurons, (ii) 50 hidden neurons, and (iii) 10 output neurons. The NN is composed of two Fully Connected (FC) layers and two activation layers. The two FC

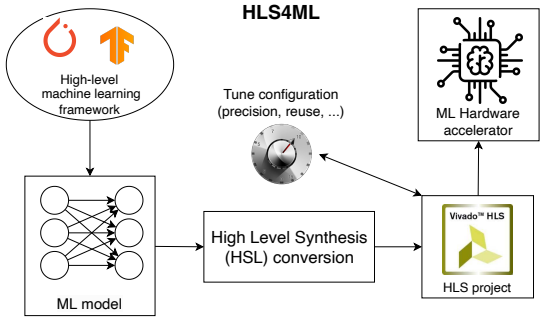

Fig. 2: HLS4ML flow [9], [10]. It allows translating conventional high-level machine learning algorithms to hardware implementation, after fine tuning of numeric precision and HW reuse parameters.

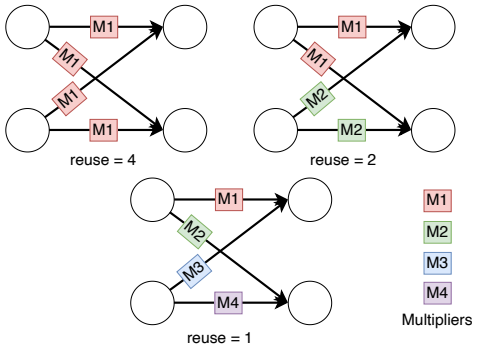

Fig. 3: The reuse parameter expresses the number of times that a given multiplier is reused when computing neuron values.

layers perform, for each output neuron O_j, the well-known weighted sum operation $O_j = \sum_{i=0}^{n} I_i * w_{ij} + b_j$, where I_i is the i-th input of the layer, w_{ij} the i-th weight for j-th output, b_j the j-th bias, and n the number of layer inputs. After the first FC layer, there is a relu activation layer, and after the second FC layer, there is a softmax activation layer. The data type used in the accelerator for NN weights and activations is 16-bit fixed-point, with 8 bits for the integer part and 8 bits for the fractional part. We trained the NN with the MNIST database of handwritten digits [12], reaching a top-1 accuracy of 95.7%.

Through HLS, we are able to explore different implementations where the designed accelerator can re-use multipliers to compute neuron values in a layer (see Figure 3) with different NN inference latency. Two re-use parameters are available for the NN under this study, which will be referred as R_1 and R_2. R_1 and R_2 values corresponds, respectively, to the number of times that a given multiplier is reused when computing neuron values in the first and in the second layer. Since the first layer has 784 inputs and 50 outputs, the total number of multiplication is $784 \times 50 = 39,200$; hence, possible R_1 values are, for example, 39,200 (1 multiplier), 19,600 (2 multipliers), 392 (100 multipliers), etc. Since the second layer has 50 inputs and 10 outputs, the total number of multiplication is $50 \times 10 = 500$, then possible values for R_2 are, for example, 500 (1 multiplier), 250 (2 multipliers), 10 (50 multipliers), etc. More information about the reuse factor parameter can be found in HLS4ML [9], [10].

TABLE I: HW attributes of tested accelerators. By only adjusting some constraints of the HLS tool, the final architecture can have huge differences (13.3× more DSP used comparing Fast vs. Slow configurations).

Config.		LUT	LUTRAM	FF	BRAM	DSP	Timing
Slow	Utilized	9850	235	12378	33	12	13.116ns
	Utilized (%)	18.51%	1.35%	11.63%	23.57%	5.45%	L 37.83% R 62.17%
Medium	Utilized	9469	239	10911	24.5	14	9.440ns
	Utilized (%)	17.79%	1.37%	10.25%	17.5%	6.36%	L 40.57% R 59.43%
Fast	Utilized	18824	1006	25619	101.5	160	8.282ns
	Utilized (%)	35.38%	5.78%	24.07%	72.5%	72.72%	L 5.06% R 94.94%
	Available	53200	17400	106400	140	220	

III. Experimental set-up

System under test: In this work, we consider a system composed of a software program – running on the host processor of the Zynq SoC – and the HLS-generated custom dataflow NN accelerator – implemented on the FPGA fabric of the Zynq SoC. Input images are initially stored in the external SD memory as a binary file; after the Linux-based Operating System (OS) boot, images are loaded into main memory by the host processor and fed to the accelerator on the FPGA. The accelerator then performs the inferences and sends the classification results back to the host. Finally, the host compares the results with the golden reference. Before comparing, the correctness of the golden reference is checked to improve the chance that a detected fault comes from the accelerator and not from a faulty golden reference.

HLS parameter configurations: We generated three versions of the proposed accelerator by combining R_1 and R_2 parameters. In particular, we refer to the configuration having $R_1 = 39,200$ and $R_2 = 500$ as *"Slow"*, to the configuration having $R_1 = 19,600$ and $R_2 = 250$ as *"Medium"*, and to the configuration having $R_1 = 392$ and $R_2 = 10$ as *"Fast"*. Table I reports the hardware attributes (absolute numbers and percentages) of the tested configurations in terms of total Lookup Tables (LUT), number of LUT used as the memory (LUTRAM), number of Flip Flops (FF), number of Block RAM Tile (BRAM), number of DSPs (DSP), and total timing needed to perform a NN inference – with timing percentage for logic (L) and routing (R). We can clearly find the well-known area-timing trade-off, where a configuration using a lot of hardware (35%) needs less time to complete an inference (8.2ns) w.r.t. a configuration using less hardware (18%) that needs longer time (13.1ns). As the error rate is directly related to resource usage, we expect the area differences will impact the system cross-section (see Section IV).

Platform: We experimented on two TUL PYNQ™-Z2 boards, based on Xilinx Zynq SoC. The SoC includes a 650MHz dual-core ARM Cortex-A9 processor, 512MB of DDR3 memory, and the FPGA fabric. The FPGA encompasses 13,300 logic slices, (each with four 6-input Look-Up Tables (LUTs) and 8 flip-flops), 630 KB of fast BRAM, and 220 DSP slices.

Beam Experiment: Our experiments were performed at the ChipIR facility of the Rutherford Appleton Laboratory, UK. Figure 4 shows the setup mounted in the ChipIR facility.

ChipIR facility delivers a beam of neutrons with a spectrum of energies similar to the atmospheric neutron one [13]. The available neutron flux is about $3.5 \times 10^6 n/(cm^2/s)$, ∼8 orders of magnitude higher than the terrestrial flux (13 $neutrons/(cm^2 \cdot h)$ at sea level [14]). The cross-section is calculated by dividing the number of observed errors by the received particle fluence ($neutrons/cm^2$).

Since the terrestrial neutron flux is low, it is improbable to see more than a single corruption during program execution in a realistic application. We have carefully designed the experiments to maintain this property (observed error rates were lower than 1 error per 2,000 executions). Each NN code was tested for at least 10 effective hours, not including the

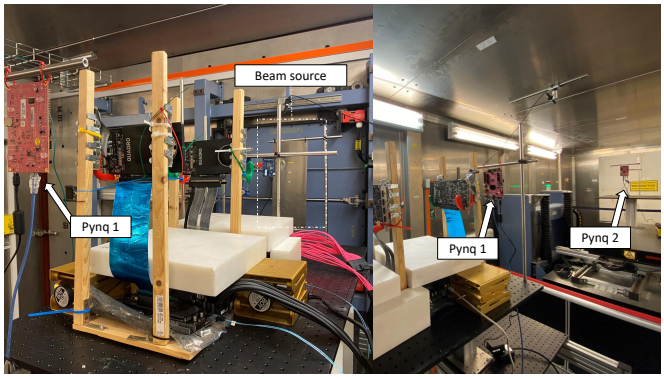

Fig. 4: Beam setup mounted at ChipIR beam line. The two boards are positioned in very different positions. Pynq 1 at 1.55m from the beam source and Pynq 2 at 4.68m from the beam source.

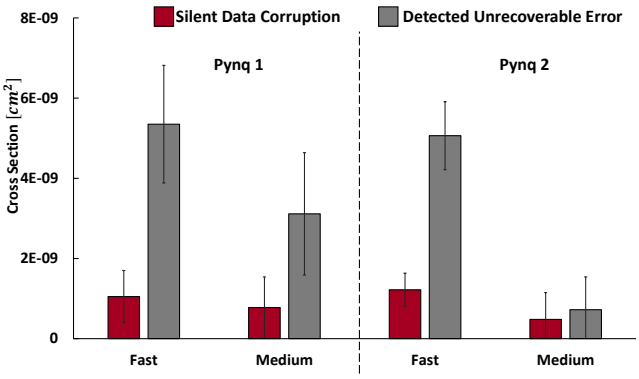

Fig. 5: Experimentaly measured cross sections for the two configurations for each board. Both Silent Data Corruption (SDC) and Detected unrecoverable Errors (DUEs) are shown.

setup, result check, initialization, and recovery from the DUE time.

We added setup software and hardware watchdogs to monitor the experiments. The software watchdog controls the application under test, and if it stops responding in a predefined time interval, the device is rebooted, and the experiment is relaunched. This watchdog detects kernel crashes or software hangs, i.e., application crashes or control flow errors that prevent the FPGA from completing assigned tasks (e.g., an infinite loop). The hardware watchdog is an Ethernet-controlled switch that performs a host computer's power cycle if the host computer itself does not acknowledge any ping requests in a predefined time interval. The hardware watchdog is necessary to detect when the operating system hangs.

Reliability metrics: A transient fault may lead to one of the following outcomes: (1) No effect on the program output (i.e., the fault is masked, or the corrupted data is not used). (2) Silent Data Corruption (SDC) (i.e., an incorrect program output). (3) Detected Unrecoverable Error (DUE) (i.e., a program crash or device reboot). Not all errors in the output of the NN will impact the classification. We classify the observed SDCs as Tolerable SDCs (i.e., that do not impact classification/detection) and Critical SDCs (i.e., that impact classification/detection). When radiation modifies the classification, we check if all images are correctly classified.

Important notes: During experiments, aligning the Zynq SoC of the two platforms directly on the beam has led to several continuous system Crashes, preventing the system from correctly booting. As a remedy, both platforms have been slightly misaligned w.r.t. the beam in order to reduce the number of observed Crashes and consequently enable the system to execute the NN. This is expected to reduce the error rate of both boards. Still, the proposed setup allows us to perform a meaningful analysis of the configurations.

IV. Neural Networks Cross Section

This section presents the results from radiation experiments for the different NN hardware configurations. Figure 5 shows the SDC and DUE cross sections (y-axis) of the different versions of the system under test (x-axis), running on the two Pynq boards (1 and 2). It is worth mentioning that the two Pynqs boards are positioned in two different places to measure the intrinsically added uncertainty from the setup. We show the results for Fast and Medium configurations. The executions of Slow configurations either ended up before the application even started (crashes at boot) or could not pass the application initialization phase. FIX THIS: Experimental data is presented with a 95% confidence interval.

DUE: DUEs are more probable than SDCs for all tested configurations. The DUE cross-section for Fast performance configuration for Pynq 1 is 5.35×10^{-9} and 5.06×10^{-9} for Pynq 2. For the Medium configuration configuration, the DUE is 3.11×10^{-9} for Pynq 1 and 7.22×10^{-10} for Pynq 2. In fact, in our experiments, the observed DUEs are mainly caused by the exceptions on the operating system that runs on the ARM processor, which performs background OS tasks not directly related to the NN. Additionally, prior works have demonstrated that the operating system can increase the DUE rate by one order of magnitude compared with bare-metal configurations on SoCs that have ARM and FPGAs devices [15].

SDC: SDCs, on the other hand, are directly related to FPGA resource usage. The larger the circuit, the larger the SDC cross-section. The SDC cross section of the Fast configurations is 1.04×10^{-9} for Pynq 1 and 1.21×10^{-9} for Pynq 2. For the Medium configurations, the SDC cross-section is 7.78×10^{-10} for Pynq 1 and 4.82×10^{-10} for Pynq 2. The SDCs are classified into *Critical* or *Tolerable*. The two boards generated a percentage of SDCs that are very similar (1.62% of average difference); thus, we report (in Table II) and discuss the results of Pynq 1. In general, over all configurations, we observe that Tolerable SDCs are more frequent (93.12%

TABLE II: Classification of the observed SDCs. The SDCs are divided into Tolerable and Critical. For each SDC type, we show the Average, Minimum, and Maximum percentages experimentally observed.

	Critical SDC			Tolerable SDC		
Config.	**Average**	**Min**	**Max**	**Average**	**Min**	**Max**
Medium	8.82%	0.00%	35.29%	91.18%	64.71%	100.00%
Fast	3.82%	0.00%	38.24%	96.18%	61.76%	100.00%
Total	**6.88%**	**0.00%**	**91.30%**	**93.12%**	**8.70%**	**100.00%**

on average) than Critical SDCs (6.88% on average). Then, we observed more Critical SDCs on average (8.82%) when running the Medium configuration than when running the Fast configuration (3.82%). Our insight is that this happens because, in Medium configurations, the same hardware is used for computing different neurons; therefore, a given fault may have an impact on multiple results.

V. Conclusions

This work, to the best of our knowledge, is the first to evaluate the impact of HLS tools on the reliability of generated NN accelerators running on FPGAs under high-energy neutrons. We report results for two different implementations of the same machine learning algorithm using HLS and measured their resilience on two Pynq boards. From the obtained results, the tolerable SDCs are more frequent than the critical ones, while more critical SDCs are observed for smaller circuits compared to larger ones. For the final paper, we will perform detailed fault injection campaigns at microarchitectural and HDL levels to demystify the source of faults and how they propagate through the system. With fault injection, we determine which resources are more prone to generate critical SDCs and how we can tune HLS tools to avoid critical SDCs.

Acknowledgment

This project has received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Sklodowska-Curie grant agreement No 899546 and 101008126 with the support of the Brittany Region and partially funded by ANR FASY (ANR-21-CE25-0008-01) and ANR RE-TRUSTING (ANR-21-CE24-0015-02). ChipIR and RADNEXT provided and supported neutron beam time experiments (DOI 10.5286/ISIS.E.RB2200303). We acknowledge the researchers that helped with neutron experiments, Dr. Christopher Frost and Dr. Carlo Cazzaniga.

References

[1] Y. LeCun *et al.*, "Deep learning," *Nature*, vol. 22, no. 3, pp. 436–44, 2015.

[2] B. Moons *et al.*, "14.5 envision: A 0.26-to-10tops/w subword-parallel dynamic-voltage-accuracy-frequency-scalable convolutional neural network processor in 28nm fdsoi," in *IEEE ISSCC*, 2017, pp. 246–247.

[3] Y.-H. Chen, J. Emer, and V. Sze, "Using dataflow to optimize energy efficiency of deep neural network accelerators," *IEEE Micro*, vol. 37, no. 3, pp. 12–21, 2017.

[4] S. Rokicki *et al.*, "What you simulate is what you synthesize: Designing a processor core from c++ specifications," in *IEEE/ACM ICCAD*, 2019, pp. 1–8.

[5] F. F. d. Santos *et al.*, "Analyzing and increasing the reliability of convolutional neural networks on GPUs," *IEEE Trans. Reliability*, vol. 68, no. 2, pp. 663–677, 2019.

[6] F. Libano, P. Rech, B. Neuman, J. Leavitt, M. Wirthlin, and J. Brunhaver, "How reduced data precision and degree of parallelism impact the reliability of convolutional neural networks on fpgas," *IEEE Transactions on Nuclear Science*, vol. 68, no. 5, pp. 865–872, 2021.

[7] D. Oliveira *et al.*, "High-energy versus thermal neutron contribution to processor and memory error rates," *IEEE Trans. Nuclear Science*, vol. 67, no. 6, pp. 1161–1168, 2020.

[8] A. Ruospo *et al.*, "Investigating data representation for efficient and reliable convolutional neural networks," *Microprocessors and Microsystems*, vol. 86, 08 2021.

[9] FastML Team, "fastmachinelearning/hls4ml," 2021. [Online]. Available: https://github.com/fastmachinelearning/hls4ml

[10] J. Duarte *et al.*, "Fast inference of deep neural networks in FPGAs for particle physics," *JINST*, vol. 13, no. 07, p. P07027, 2018.

[11] P. Coussy *et al.*, "An introduction to high-level synthesis," *IEEE Des. Test*, vol. 26, no. 4, p. 8–17, jul 2009.

[12] L. Deng, "The mnist database of handwritten digit images for machine learning research [best of the web]," *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 141–142, 2012.

[13] C. Cazzaniga *et al.*, "Progress of the scientific commissioning of a fast neutron beamline for chip irradiation," in *ICANS*, 2017, pp. 159–164.

[14] C. Slayman, *JEDEC Standards on Measurement and Reporting of Alpha Particle and Terrestrial Cosmic Ray Induced Soft Errors*, 2011, pp. 55–76.

[15] P. Bodmann, G. Papadimitriou, D. Gizopoulos, and P. Rech, "The impact of soc integration and os deployment on the reliability of arm processors," in *IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2021, pp. 223–225.