Analyzing and Improving the Reliability of Matrix Multiplication and Neural Networks on FPGAs

by

Fabiano Libano

A Dissertation Presented in Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy

Approved November 2021 by the
Graduate Supervisory Committee:

John Brunhaver, Chair
Lawrence Clark
Heather Quinn
Paolo Rech

ARIZONA STATE UNIVERSITY

December 2021

ABSTRACT

Neural networks are increasingly becoming attractive solutions for automated systems within automotive, aerospace, and military industries. Since many applications in such fields are both real-time and safety-critical, strict performance and reliability constraints must be considered.

To achieve high performance, specialized architectures are required. Given that over 90% of the workload in modern neural network topologies is dominated by matrix multiplication, accelerating said algorithm becomes of paramount importance. Modern neural network accelerators, such as Xilinx's Deep Processing Unit (DPU), adopt efficient systolic-like architectures.

Thanks to their high degree of parallelism and design flexibility, Field-Programmable Gate Arrays (FPGAs) are among the most promising devices for speeding up matrix multiplication and neural network computation. However, SRAM-based FPGAs are also known to suffer from radiation-induced upsets in their configuration memories.

To achieve high reliability, hardening strategies must be put in place. However, traditional modular redundancy of inherently expensive modules is not always feasible due to limited resource availability on target devices. Therefore, more efficient and cleverly designed hardening methods become a necessity. For instance, Algorithm-Based Fault-Tolerance (ABFT) exploits algorithm characteristics to deliver error detection/correction capabilities at significantly lower costs.

First, experimental results with Xilinx's DPU indicate that failure rates can be over twice as high as the limits specified for terrestrial applications. In other words, the undeniable need for hardening in the state-of-the-art neural network accelerator for FPGAs is demonstrated.

Later, an extensive multi-level fault propagation analysis is presented, and an ultra-low-cost algorithm-based error detection strategy for matrix multiplication is proposed. By considering the specifics of FPGAs' fault model, this novel hardening method decreases costs of implementation by over a polynomial degree, when compared to state-of-the-art solutions. A corresponding architectural implementation is suggested, incurring area and energy overheads lower than 1% for the vast majority of systolic arrays dimensions.

Finally, the impact of fundamental design decisions, such as data precision in processing elements, and overall degree of parallelism, on the reliability of hypothetical neural network accelerators is experimentally investigated. A novel way of predicting the compound failure rate of inherently inaccurate algorithms/applications in the presence of radiation is also provided.

DEDICATION

I would like to thank my family and friends, for their unconditional support throughout my life and my years as a doctoral student.

I would like to thank my advisor, John Brunhaver, for providing me with a myriad of opportunities and invaluable technical lessons.

I would like to thank my committee members, Lawrence Clark, Heather Quinn, and Paolo Rech, for their crucial feedback on the development of this dissertation.

Finally, I would like to thank my home country, Brazil, for backing my academic career from the start and being my safe harbor.

Obrigado.

TABLE OF CONTENTS

CHAPTER Page

LIST OF TABLES

LIST OF FIGURES

Figure Page

Chapter 1

INTRODUCTION

Self-driving vehicles will soon be pervasive. In fact, the addition of self-driving capabilities has been one of the main priorities within the trillion-dollar automotive industry (McKinsey 2021). At early stages, this movement was mainly led by Tesla (Tesla 2021): Since it is a comparatively small manufacturer (in terms on cars produced per year) (CNBC 2021b) (CNBC 2021a), it benefits from its efforts on being a highly tech-focused (almost boutique-like) company. However, more well-established premium brands, such as Mercedes-Benz (Mercedes-Benz 2021), BMW (BMW 2021), and Volvo (Volvo 2021), are quickly catching up to Tesla's advancements, as their clients are able to pay for the trickle-down cost of research, development, and deployment of autonomous systems. Other mainstream companies should eventually follow such tendency (in order to remain competitive), as soon as implementations costs start to diminish. Given this scenario, we expect to see a big percentage of road vehicles being autonomous, in a considerably short timeline.

Once widely deployed, self-driving vehicles will have a profoundly positive impact on society. According to (US Department of Transportation 2017), over 84 billion hours were driven in the United States in 2015. According to (US National Highway Traffic Safety Administration 2015), over 32 thousand fatal crashes happened in those hours. Considering each fatal crash as a failure, it follows that the Failure In Time (FIT) rate for US drivers in 2015 was approximatelly 380 failures per billion hours of operation. The ISO 26262 (ISO 2021) is a standard that establishes functional safety requirements for road vehicles. Within such requirements is the Automotive Safety

Integrity Level (ASIL) classification scheme. In order to reach the maximum ASIL certification (ASIL-D), FIT rate must be below 10. If, hypothetically, all cars had ASIL-D autonomous systems, only 84 fatal crashes would have happened in the same period. Therefore, it is of paramount importance to strive for appropriately reliable self-driving cars.

Similarly, the space exploration sector is also increasing its investments in autonomous vehicles for unmanned missions to destinations across the solar system and beyond. From the public standpoint, NASA has pioneered the movement, with a rover/helicopter combo for its latest arrival on Mars (NASA 2021a) (NASA 2021c). Evidently, acquiring information about our astronomical surroundings is of great importance for us as species, and doing so is often prohibitive to humans due to environmental characteristics. Therefore, having intelligent robots deployed in locations of interest is a key alternative. In this case, autonomy is especially relevant since the cosmological distances involved make remote control incredibly impractical. For instance, a radio signal can take between 5 and 20 minutes to travel between Earth and Mars, depending on planet positions (NASA 2021b). From the private sector's perspective, companies like SpaceX, Blue Origin, and Virgin Galactic, may choose to invest in autonomous vehicles to aid the exploration of precious metals and other valuable resources, which can be abundantly found in a number of asteroids not so far from Earth (NBC 2021), although such endeavor would involve a multitude of challenges other than having reliable automation.

From the aforementioned, it becomes clear that safety-critical applications require high levels of reliability. Since catastrophic failures can lead to human casualties (in the case of self-driving cars), or to significant financial losses (in the case of space missions), it is mandatory to probabilistically guarantee low failure rates. However,

reaching high levels of reliability is non-trivial, and typically incurs greater efforts in design, along with greater costs of implementation. As we know, electronic devices are prone to errors. Even more particularly so in harsh environments, with extreme variations and/or steady states of temperature, pressure, and radiation. In such scenarios, the Commercial-Off-The-Shelf (COTS) devices tend not to be sufficient, as they are not necessarily ready to handle errors in the middle of computation. In order to harden such components, modular redundancy is the most traditional solution. For instance, full triplication (followed by majority voting), in theory, guarantees immunity against single faults. However, its inherent cost increase is 200+% (Kastensmidt et al. 2005). Moreover, cost can be context-dependent (i.e. instruction triplication increases execution time, while circuit triplication increases area, and both increase energy consumption).

As a result, the general problem we address is that of **minimizing cost for a set level of resilience**. Evidently, other studies have had this same goal in different settings (Hari et al. 2021) (Draghetti et al. 2019) (Mitra and McCluskey 2000). Upon dissecting related works, it is generally understood that, by taking advantage of algorithm and/or application properties, it is possible to develop more efficient hardening strategies, delivering similar levels of protection with much lower overheads. However, extensive analytical and empirical efforts are necessary to achieve them. Luckily, some algorithms (such as matrix multiplication) are pervasive as the computational core across a myriad of applications. Therefore, a single study on a novel hardening technique is able to positively impact entire fields and industries that rely on the correct execution of said algorithm/application.

Recent developments within the Machine Learning (ML) field are leading to the increased adoption of Neural Networks (NNs) as key components in autonomous

systems. In particular, NNs have the ability to learn and subsequently perform tasks that are hard to explicitly program with traditional algorithms. As a result, NNs contribute to a much faster development cycle, and consequently reduce the time-to-market for consumer products, which ultimately contributes to maximizing profits. Moreover, we must highlight Convolutional Neural Networks (CNNs) as a special kind of NN, geared towards computer vision. Not very long ago, state-of-the-art CNNs have surpassed long-established image processing algorithms (Krizhevsky, Sutskever, and Hinton 2017). In addition, there has been a steady increase in accuracy over the years (Stanford Institute for Human-Centered Artificial Intelligence 2021) on the well-established ImageNet challenge (Deng et al. 2009), where classifiers are evaluated around their top-1 and top-5 scores (Fig 1). Consequently, the deployment of neural networks in autonomous scenarios has been expanded significantly, as state-of-the-art models are able to play critical roles in improving control and decision-making directives. However, in order to achieve high levels of accuracy, CNN topologies end up being deep and dense, which makes them computationally expensive. As an example, Google's InceptionV3 architecture executes over 5 billion arithmetic operations to process each input image (Bianco et al. 2018).

At the same time, safety-critical applications tend to have strict performance requirements, on top of reliability ones. For instance, a self-driving car is required to compute incoming frames and take proper action at least 30 times per second (i.e. 30fps). Moreover, minimum frame rate constraints only get harder to attain to as physical speed increases in different applications (i.e. drones, airplanes, spaceships) (Kagami 2010). Therefore, highly capable devices and specialized architectures are needed to ensure adequate performance. More specifically, we are interested in answering how parallel our accelerators should be, and how complex our processing

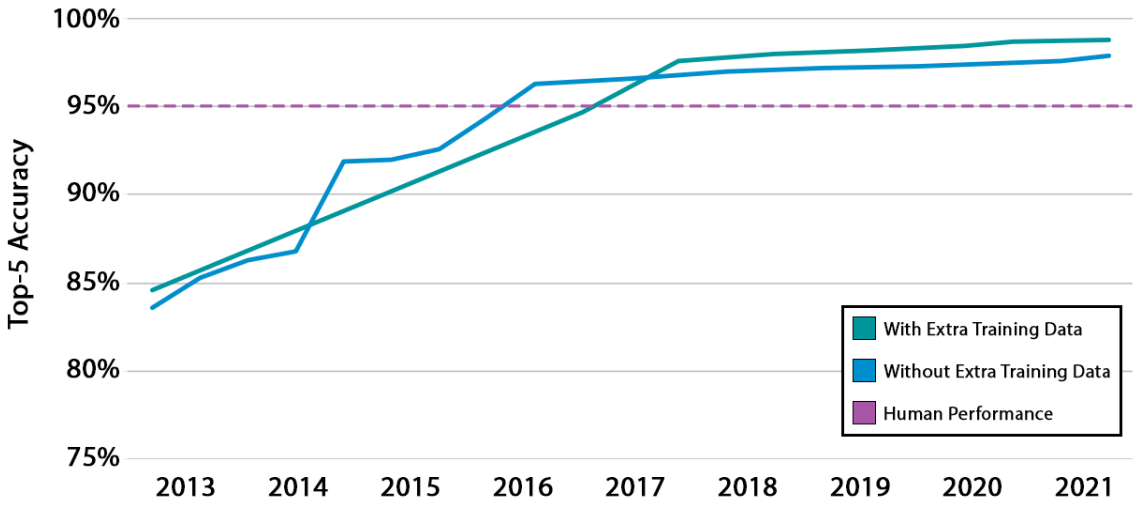

Figure 1: Top-5 accuracy on the ImageNet challenge over the years. (A measure of whether the correct label is within the classifier's top-5 predictions). *Source*: Stanford Institute for Human-Centered Artificial Intelligence (2021)

elements should get, since proper design decisions at early stages can deliver optimal performance and hardness trade-offs. Such customization of architectures thus requires appropriately fast and flexible devices.

Luckily, modern Field-Programmable Gate-Arrays (FPGAs) are very much suitable for our needs. Given the dedicated Digital Signal Processor (DSP) slices spread throughout the fabric, it becomes possible to implement high-speed arithmetic units, and easily interconnect them for data exchange. Furthermore, the reprogrammable nature of SRAM-based FPGAs allows for top-tier design flexibility, which is a key factor for a number of reasons. First off, compared to the development of an Application-Specific Integrated Circuit (ASIC), design effort is much lower and design iterations are much faster, since there is no need to fabricate the custom architecture. Second, having the ability to reprogram allows for easier in-field patching, fixing bugs, or even for deploying entirely new versions, which is a particularly relevant characteristic when it comes to long-life consumer products (such as automobiles). Therefore, it is not

a coincidence that FPGAs have been extensively deployed in datacenters (Microsoft 2021), and thoroughly studied for radiation-hardened space flight (Wirthlin 2015), in which the use of similarly capable Graphics Processing Units (GPUs) is often prohibited due to their high power consumption (Aldahlawi, Kim, and Kim 2019) (Piovesan et al. 2016). For instance, running Google's InceptionV3 on an Nvidia GTX980 GPU would consume 178mJ (Tschopp et al. 2016), while the same workload would only consume 44mJ on a comparable Xilinx Zynq-7000 FPGA (Posewsky and Ziener 2016).

As we know, radiation is a particular source of faults for electronic devices. Since transistors are made out of silicon, the interaction of both charged and uncharged particles can cause undesired logic switching, which can later lead to application failures in the macro scale (Baumann 2001). The space environment is especially harsh, due to radioactive emissions from stars (such as our Sun), and from other major celestial events. Fortunately for us, Earth's magnetic field serves as a shield, deviating the majority of particles away. However, sufficiently energetic cosmic rays collide with nuclei in our atmosphere and, as a result, produce a spectrum of secondary particles. Within those particles, neutrons tend to be the bigger threat, as they reach sea level, and thus are capable of upsetting terrestrial applications as well. In fact, the neutron flux has been experimentally measured to be of about $13n/(cm^2 \cdot h)$ in New York City (JEDEC 2006). Moreover, the atmospheric neutron flux is known to exponentially increase with altitude, up to about 400000ft (Fig. 2), which means that an airplane cruising at the typical 30000ft will experience a much higher flux than a self-driving car at sea level (Normand and Baker 1993). Hence, aerospace and military sectors were historically among the first to invest in reliability studies for satellites and aircraft.

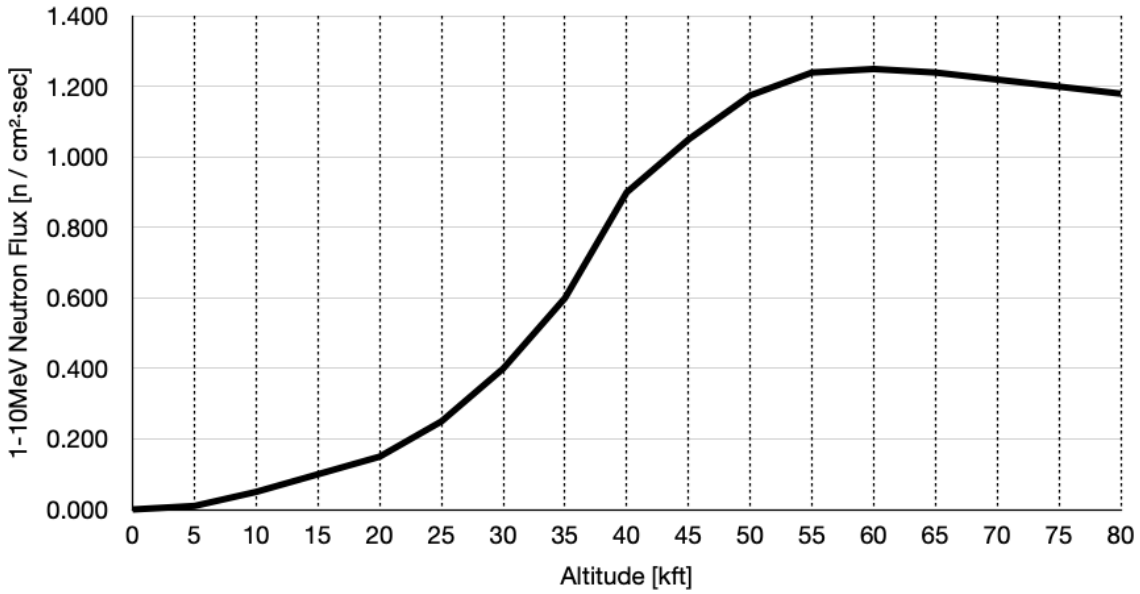

Figure 2: 1-10 MeV atmospheric neutron flux as a function of altitude.
Source: Normand and Baker (1993)

To complicate things further, technology shrink has resulted in a relative decrease in resilience. Over the years, the industry has advanced tremendously in its fabrication processes, which allowed for reduced transistor sizes, increased transistor density, and reduced operating voltages. While all of these achievements are great for performance, area, and energy, they also contribute to making silicon devices even more sensitive to radioactive particles (Baumann 2005). In fact, the aforementioned presence of neutrons at sea level was first detected on high-end computers around the 80s (Leray et al. 2004), but only really became a concern after technology nodes shrank past about 130nm. Luckily, new transistor geometries (i.e. FinFET) helped to significantly shift these trends: studies have shown that a FinFET device can be about 3 times more robust than a comparable CMOS in the same technology node (Oliveira, Lüdke, and Meinhardt 2019). Although promising, switch-device innovation is unlikely to

solve all radiation-related problems, which means that other countermeasures must be put in place to avoid catastrophic events.

Unfortunately, failures in safety-critical scenarios have already happened. For instance, in 2008 a flight operated by Qantas Airways had to perform an emergency landing, after a couple of uncommanded nose-down maneuvers by the auto-pilot system, which caused several passengers and crew members to accrue fractures, lacerations, and even spinal injuries (The Sidney Morning Herald 2017). Later investigation has found that multiple upsets to the Angle of Attack (AOA) data, in a short time interval, have caused the airplane's Flight Control Primary Computers (FCPCs) to command abrupt pitch-downs (Australian Travel Safety Bureau 2008). Furthermore, in 2007 a car crash involving a Toyota Camry has led to the death of the driver, after the electronic throttle control system has suddenly and inadvertently accelerated the vehicle (Live Science 2010). After hundreds of other similar reports, a full-scale investigation was launched, even involving NASA specialists, which concluded that the software contained a number of unprotected critical variables, and that as little as a single bitflip could cause severe malfunctions that would go undetected by any fail-safe mechanism (EE Times 2013).

As we can see, soft errors are not always trivial to spot. Radiation can cause a multitude of different effects on electronic devices, some of which are cumulative and happen over long periods of exposure, such as Total Ionizing Dose (TID) (Oldham and McLean 2003), and others that are so-called destructive (i.e. the device ceases to function entirely) (Bruguier and Palau 1996). However, events may also occur silently, when computation completes in a normal fashion, but its outputs present corruptions. These events are categorized as Silent Data Corruptions (SDCs) (Fiala et al. 2012),

and they pose a dangerous threat to systems' safety and reliability, as undetected errors can more easily turn into catastrophic failures.

In fact, we shall state that faults propagate across abstraction layers in the computing stack, as we illustrate in Fig 3. As previously mentioned, from a physical standpoint, we simply have radioactive particles interacting with silicon transistors, occasionally leading to unwanted switching activity. However, when *captured* by clock edges, transient pulses can turn into (micro) architectural faults in the form of bitflips on storage elements (such as memories and registers). Then, of course, the faulty architecture can cause (silent) data corruptions in the algorithm's execution. Finally, algorithm errors can become (critical) failures in an application context, depending on their level of severity. Therefore, we can clearly perceive that detection and mitigation techniques must be put in place at different levels of the stack, as a way of attaining highly reliable systems.

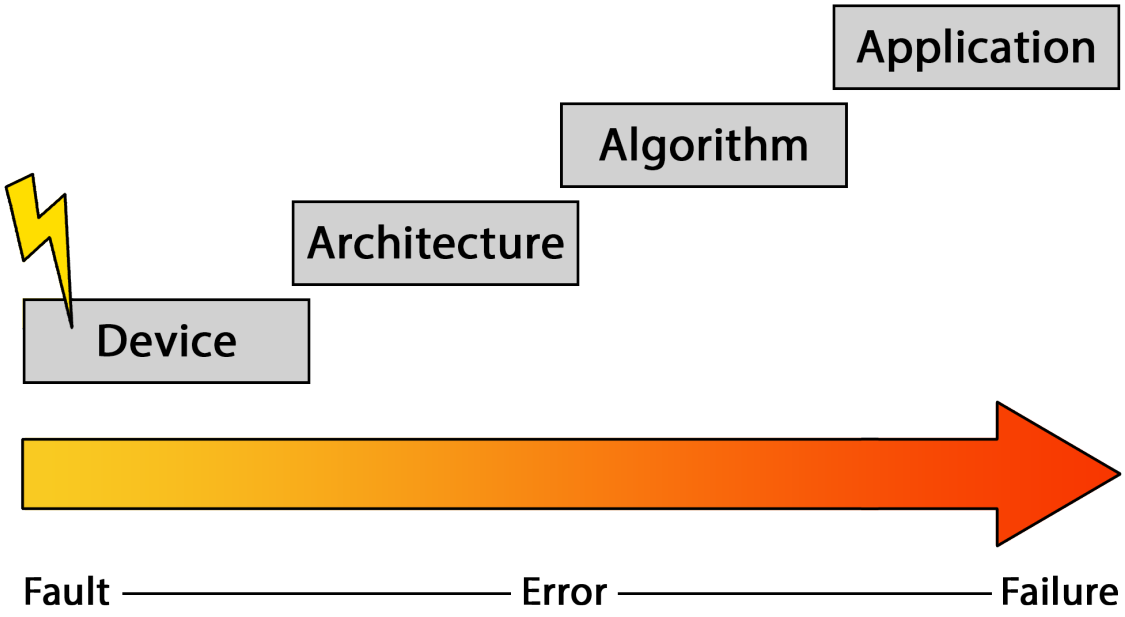

Figure 3: Depiction of fault propagation across abstraction layers.

Specifically, in the case of SRAM-based FPGAs, their very own virtue of reprogrammability is what establishes a particularly distinct fault model. As the logic functionality of the programmable fabric is determined according to the contents of the FPGA's configuration memory, upsets will exhibit a **persistent effect** until actively corrected (through periodic scrubbing or partial/full reconfiguration (Heiner et al. 2009) (R. Zhang et al. 2020)). As a direct consequence of having a corrupted configuration memory, functional correctness is lost, since the logic circuit's programmed behavior is compromised. Moreover, subsequent operations on said circuit are very likely to also present errors. Hence, when contemplating about appropriate hardening techniques for FPGAs, their peculiar fault model must be taken into account.

Similarly, the approximated nature of neural networks also establishes a unique fault model. Since NN/CNN models are typically not 100% accurate, the direct ramification is that they will provide erroneous answers some percentage of the time. Therefore, when put in harsh environments (i.e. radioactive), we have an approximated algorithm running in a fault-inducing surrounding, which increases the chances of application failures. As a result, we are interested in answering precisely how much should we worry about accuracy and radiation sensitivity. In other words, we pose that it is worth exploring the arbitrage between a model's complexity contribution to resilience, and hardening efforts' contribution to resilience.

Moreover, it is fundamental to observe that Matrix Multiplication (MxM) is at the core of CNN computation. This is due to the convolution operation being easily translated to an equivalent MxM, using the *im2col* method (Chellapilla, Puri, and Simard 2006), and the inner product operation between weights and inputs in fully connected layers being MxM by definition. In fact, over 90% of the arithmetic workload in modern CNN topologies is simply matrix multiplication (Hari et al. 2021).

Therefore, it follows that accelerating MxM accelerates CNN execution. In GPUs, highly optimized libraries implement the General Matrix Multiply (GEMM) algorithm, considering properties of the memory hierarchy in each target device (Volkov and Demmel 2008). In ASICs and FPGAs, specialized architectures are utilized with varying degrees of computing power, but it is generally accepted that systolic arrays (Kung and Charles 1978) deliver an excellent trade-off between performance and cost (Jouppi et al. 2017) (Xilinx 2019b). The simplicity of processing elements, along with the interconnection between neighbors, and the rhythmic style of data movement, add up to an accelerator architecture that provides low latency computation.

In particular, systolic arrays are optimal for high-performance 8-bit integer (INT8) matrix multiplication in modern (Xilinx) FPGAs. Given the novel quantization flows in state-of-the-art ML frameworks (i.e. TensorFlow (TensorFlow 2021)), it became possible to reduce the precision of trained CNN models from 32-bit floating-point (FP32) to INT8, with virtually no accuracy loss. Concomitantly, the DSP architecture of Xilinx FPGAs allows for packing two simultaneous 8-bit multiplications together (given one shared operator) (Véstias et al. 2017) (Xilinx 2017). On top of that, DSP slices can be clocked at higher frequencies than other logic resources on the programmable fabric, which means they can be easily time-multiplexed for even greater throughput (Xilinx 2019a). As the processing elements in MxM-focused systolic arrays are Multiply-Accumulate (MAC) units, using properly configured DSPs as arithmetic cores appears as a very sensible choice.

However, the superposition of persistent faults in FPGAs, and the systolic pattern of dataflow is problematic. Since each processing element within the array is used to calculate multiple output matrix values, a faulty PE is likely to affect many of those outputs. As a result, the majority of error patterns observed in this scenario tend to be

partially or fully corrupted lines in the result matrix. Additionally, the most traditional Algorithm-Based Fault Tolerance (ABFT) technique for matrix multiplication is capable of correcting single errors only (Kuang-Hua Huang and Abraham 1984), while other hardening methods pay very steep prices for their correction procedures (Rech et al. 2013), which would have to be triggered at every subsequent execution, until said persistent fault is removed. Moreover, implementing such correction procedures in hardware would incur extra area and energy costs. Therefore, for this particular context, we sustain that it is wiser to focus on cheap and efficient error detection mechanisms, followed by (partial) reconfiguration. In other words, we shall state that: *It is preferable to pay a slightly higher price every once in a while, instead of paying for insurance every time.*

1.1 Contributions

As a way of advancing the state-of-the-art towards reliable matrix multiplication, neural networks, and autonomous systems, this dissertation provides the following major contributions:

- An experimental evaluation of the state-of-the-art, based on neutron data with different configurations of Xilinx's Deep Processing Unit (DPU) (Xilinx 2019b), running Google's InceptionV3 neural network (Jouppi et al. 2017), trained on NASA's Mars HiRISE dataset (Wagstaff and Lu 2017).
- A multi-level fault propagation model that details how faults transition across abstraction layers, considering the following case-study stack: FPGA - Systolic Array - Matrix Multiplication - CNN.
- A novel, highly-efficient error detection strategy for matrix multiplication, specifically tailored for systolic arrays on FPGAs.
- An estimation of how simplification techniques (such as quantization), and how essential architectural aspects (such as level of data precision and degree of parallelism) impact the area, performance, and overall reliability of hypothetical neural network accelerators on FPGAs.
- A novel, generic equation that models the hybrid failure rate of inherently inaccurate algorithms in radioactive environments.

In addition, this dissertation also makes the following minor contributions, which can facilitate other related studies:

- A publicly available, high-performance systolic array implementation for integer matrix multiplication, optimized for Xilinx FPGAs, along with a parametric RTL code generator, and optional error detection capabilities.
- A publicly available, easily adaptable experimental setup for beam-testing FPGA/ASIC designs.

1.2 Manuscript Organization

The remainder of this dissertation is organized as follows: Chapter 2 provides a foundation on reliability concepts, safety-critical applications, artificial intelligence, machine learning, and neural networks. It also makes the case for hardware acceleration, briefly explains why radiation effects are a concern, and how traditional hardening techniques work. Finally, it goes over related work. Chapter 3 presents details regarding the case study devices that were utilized in our experimental phase. Chapter 4 describes the test methodology for both fault injection and radiation experiments, while explaining their complementary nature of information. Chapter 5 evaluates the radiation-induced upset susceptibility of a state-of-the-art neural network accelerator, through neutron beam experiments. Chapter 6 provides an experimentally-grounded multi-level fault propagation model, then proposes and validates an efficient hardening strategy for matrix multiplication. Chapter 7 elaborates on the impact of architectural aspects such as data precision and parallelism on the overall resilience of hypothetical neural network accelerators, while also discussing the relationships between accuracy, performance, and radiation sensitivity of inherently inaccurate algorithms. Chapter 8 sums up the conclusions and enumerates viable topics for future work.

Chapter 2

BACKGROUND

This chapter introduces fundamental notions for the understanding of our contributions. In particular, the computation (and corresponding acceleration) of modern neural network workloads, how radiation affects SRAM-based FPGAs, ways of measuring reliability, and the cost-benefit relationship surrounding traditional hardening methods. Finally, we go over topics of directly related work, which serve as points of comparison in later chapters.

2.1 Artificial Intelligence, Machine Learning & Neural Networks

Although extremely popular today, Artificial Intelligence (AI), as a field of computer science, arguably started a long time ago. In fact, the dawn of AI can be tied to the famous *"Can machines think?"* question posed by Alan Turing himself (Turing 1950). The author also proposed the equally renowned *Imitation Game* as a framework for answering said question, since it draws a very explicit line between the physical and intellectual capabilities of a human being, and thus, would be a great way to determine the imitation capability (intelligence) of the machine.

Evidently, the field has evolved tremendously since, with Machine Learning becoming the center of attention, allowing computers to learn from abstract data without having to be explicitly programmed. Therefore, ML allows for very straightforward solutions to a myriad of highly complex problems, where the construction of a deterministic algorithm may be very time-consuming and/or inefficient (Samuel 1959).

The aforementioned learning process generally involves several hours of supervised or unsupervised training (depending on whether the training data is labeled or not), where the ML model is continuously fed with a large enough dataset, and adjusts its own behavior after each iteration. Mathematically, the training procedure is nothing more than an optimization task, where the objective function can interchangeably be *minimum error* or *maximum accuracy* (Russell and Norvig 2009). Additionally, the *behavior* of the model is strictly determined by the parameters (weights) learned through training. Thus, the optimization converges to a set of weights that satisfies the objective function. Once a given ML model is properly trained, it becomes capable of processing never-before-seen inputs. Interestingly, the reasoning and decision-making style ingrained in such models tends to spot patterns that humans often cannot deterministically describe.

Although ML encompasses a number of different approaches, the focus of this dissertation is on Artificial Neural Networks (ANNs). To put it simply, ANNs use biologically inspired neuron abstractions to approximate mathematical functions. From another perspective, sets of interconnected neurons, organized in layers, propagate stimuli in a highly parallel fashion, aiming to accomplish a given task. Rather similar to the firing of neurons across synaptic pathways in human brains. One of the first successful types of ANNs was based around the *Perceptron* (Rosenblatt 1958), which is an algorithm for supervised learning. The simplest form of ANN is then the Single Layer Perceptron (SLP), capable of performing classification on linearly separable classes of problems (i.e. it is possible to draw a line in an euclidean plane, separating two sets of points). The SLP works by carrying out the *weighted sum* of its inputs, and comparing it to an *activation threshold*, ultimately outputting '0' (if under the threshold), or '1' (if over the threshold). Then, in order to expand the mathematical

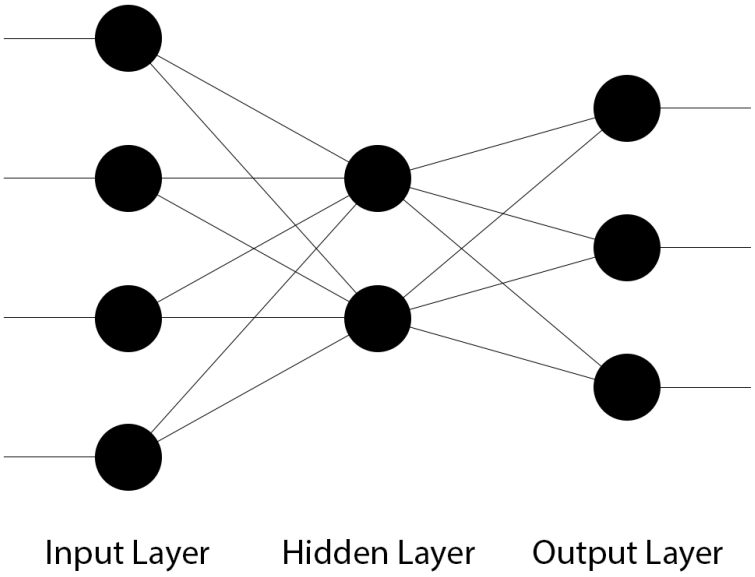

Figure 4: A simple ANN

modeling capability of the SLP, the Multi Layer Perceptron (MLP) was introduced by (Minsky and Papert 1988). It added the notion of *layers* of neurons (essentially, a pipeline of SLPs), as well as non-linear activation functions. Fig. 4 shows an example of a very simple ANN, constituted of only three layers. Every neuron in a given layer *N* is connected to every neuron from layers *N-1* and *N+1*. This property of MLP is called *full connection*. Moreover, each connection between two neurons has a weight associated with it. From a computer architecture perspective, every connection in Fig. 4 is a multiplier, while every neuron is an adder followed by an activation function (commonly of sigmoidal form).

However, when increasing task complexity (and the number of neurons in the topology), the fully-connected nature of MLPs leads to exponential increases in: (1) total arithmetic operations (meaning that more computing power is needed), and (2) number of learnable weights (meaning that training takes longer, and more memory is required for weight storage). In order to overcome these issues, researchers started

looking for ways to explore spatial and temporal invariance in recognition tasks. For instance, if we consider an image as the input for an MLP classifier, we would have to connect every pixel to every neuron in the following layer, performing a lot of unnecessary operations. Furthermore, our object of interest, (i.e. a *car* or a *pedestrian*), could be anywhere on that image, which means that the group of neurons receiving inputs from the lower-left corner of the image would have to learn to represent *pedestrian* independently from the group of neurons connected to the upper-right corner.

As an attempt to solve these problems and enable efficient/accurate image processing, Convolutional Neural Networks (CNNs) were introduced by (Lecun et al. 1998). The basic idea behind CNNs' success is the addition of a *feature extraction* phase before using an MLP, in which only the essential pieces of information about the input are propagated forwards. On images, for example, such information can include shapes, shadows, edges, colors, and others. In order to perform feature extraction, two new types of layers were introduced: the *convolutional layer* and the *pooling layer*. Convolutional layers have sliding *filters* (arrays of learnable weights) that move over the input image, executing element-wise multiplications, and accumulating the results for each position of the filter (as it moves from top left to bottom right, at a predetermined rate called *stride*). Pooling layers also adopt the idea of sliding windows over the input image, except for the fact that there are no learnable weights associated with them. Instead, pooling layers perform one of two very common operations: maximum and average. Given a subset of pixels, *max-pooling* propagates the element with the highest value, while *avg-pooling* propagates the average value. As examples, Fig. 5 and Fig. 6 illustrate how convolutional and pooling layers work, respectively.

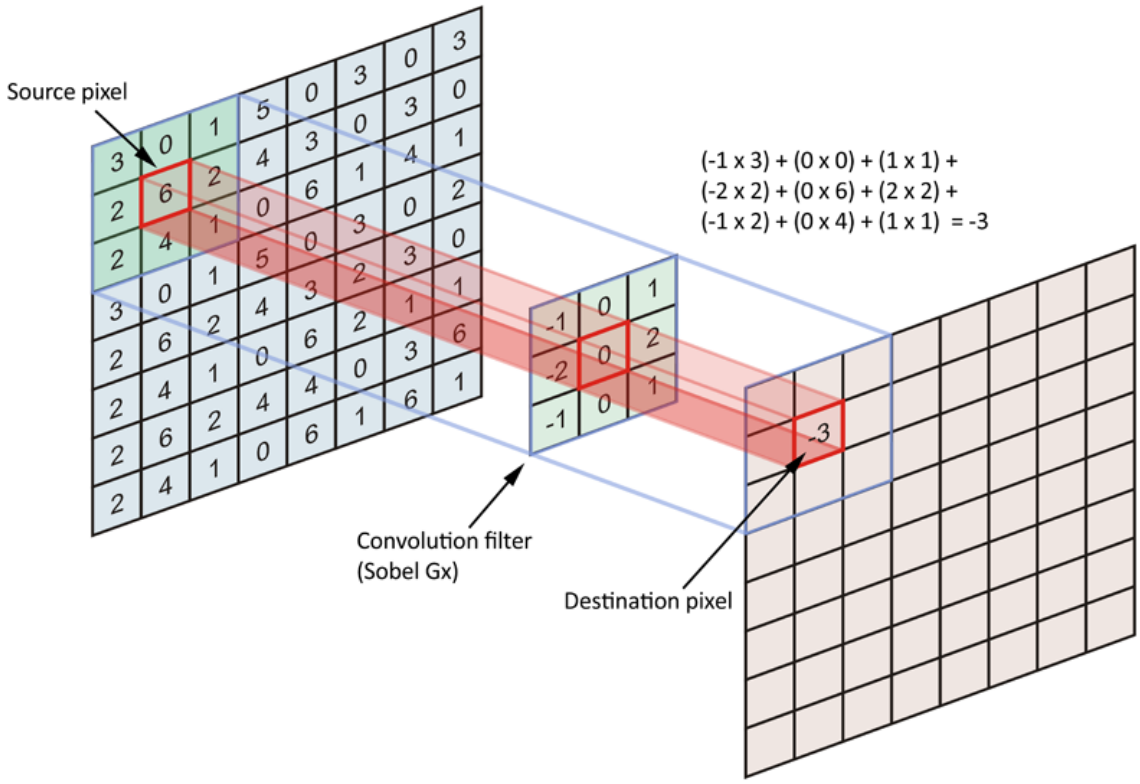

Figure 5: The operations in a convolutional layer

Source: Data Science Stack Exchange (2020)

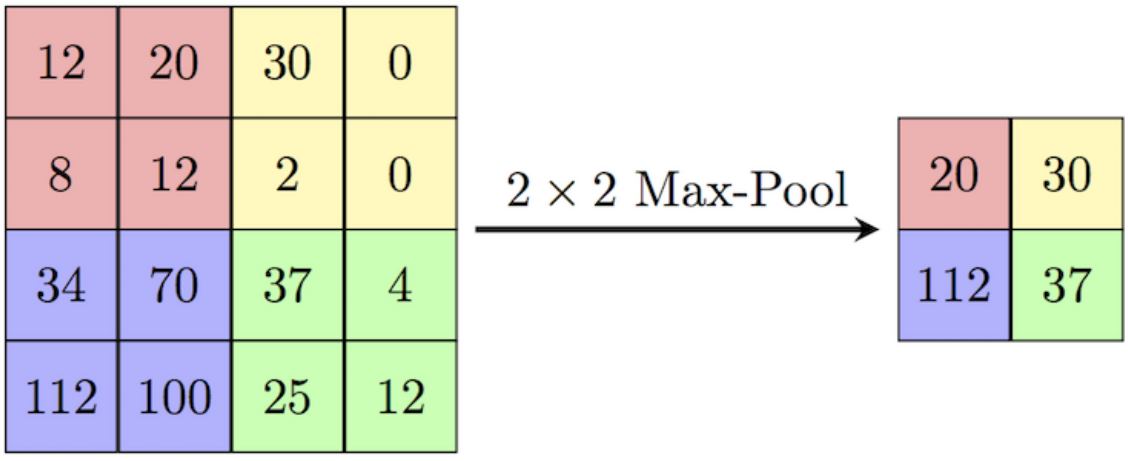

Figure 6: The operations in a pooling layer

Source: Cambridge Spark (2020)

State-of-the-art CNN topologies tend to be simultaneously wide and deep. To achieve high accuracy, some of the most successful models need to include tens of layers and hundreds of filters per layer. Therefore, the cost of executing such networks becomes inherently expensive. Luckily, the arithmetic involved in filter and neuron calculations can be performed with a high degree of concurrency. In other words, by adequately exploiting parallelism, significant speed-ups can be achieved.

As mentioned, matrix multiplication is at the core of CNN computation. By definition, the inner product operation between inputs and weights in fully connected layers is MxM. Additionally, the convolution operation can be easily translated to an equivalent MxM (Fig. 7), using the well-known *im2col* method (Chellapilla, Puri, and Simard 2006). The motivation behind such conversion originally came from the architectural characteristics of modern GPUs, and their Single Instruction Multiple Data (SIMD) style of computing. Nevertheless, ASIC/FPGA accelerators also tend to perform matrix multiplication as opposed to convolution, as we shall discuss next.

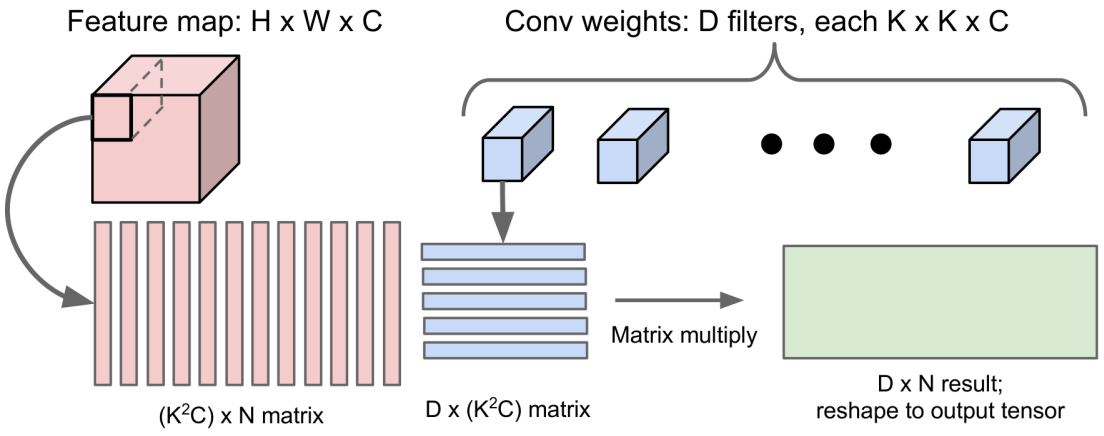

Figure 7: Convolution as an equivalent matrix multiplication

Source: dos Santos (2020)

2.2 Hardware Acceleration

As previously mentioned, real-time applications have very strict performance requirements (Shin and Ramanathan 1994). Given a deployment scenario, maximum latency and/or minimum throughput constraints tend to be well defined. For instance, in computer vision tasks, the performance rule of thumb to adhere to is, at least, 30 *frames per second (fps)* (Kagami 2010), which means that a real-time CNN must be able to fully process an input image in, at most, 1/30th of a second. Moreover, in a safety-critical context, not attaining to such constraints can also be considered an application-level failure in and of itself. Therefore, once a baseline workload is established, system designers must determine whether or not there is sufficient processing power available to compute it in time.

Traditional CPUs do not offer enough parallelism. CPUs are designed to have a low number of highly performant cores, meant to excel in serial tasks. As a consequence of having few compute units, accelerating inherently parallel algorithms becomes difficult. In order to address such limitation, appropriately parallel software must be written for GPUs, and specialized hardware must be thought of for ASICs/FPGAs.

However, GPU-based acceleration also has its drawbacks. Although optimized software libraries exist abundantly, a non-negligible programming effort is still required to integrate software components together. Particularly, data access patterns and memory allocation need to be carefully thought of, in order to match the parameters of the memory hierarchy in each target device (Volkov and Demmel 2008). In addition, most GPUs tend to be geared towards *throughput* (the number of items processed per unit time), rather than *latency* (the time between making a request and the completion of the response) (Andersch et al. 2015). Finally, despite being very parallel,

the architecture of modern GPUs is still general purpose. This means that some level of performance and energy efficiency is inherently sacrificed for genericism.

Alternatively, specialized architectures can deliver top-tier metrics, while justifying their associated increase in design effort. By giving up the general-purpose title, it is possible to explore application/algorithm properties as a way of optimizing compute units and dataflow patterns. For instance, modern quantized neural networks use 8-bit integers for representation, which means that executing their operations on 32-bit units would be wasteful. From the perspective of data, streaming-like interfaces are employed to ensure full pipeline utilization, while appropriate interconnect structures help to reduce global memory accesses by exploiting locality.

Systolic arrays are an example of specialized architectures, within the context of this dissertation. Through instantiating and connecting a set of processing elements, it becomes possible to efficiently solve a number of problems within the realm of linear algebra. In fact, depending on the geometry of the array, Fourier transform, LU decomposition, and (most importantly for us) matrix multiplication (Fig. 8), are all computable (Kung and Charles 1978). The term *systolic* implies that said computation is performed in a rhythmic/pipelined fashion, with data being pumped in and out of the structure at every clock cycle. Part of the improvements over traditional architectures comes from the drastic reduction in memory accesses, as PEs are connected directly to their neighbors and thus can move data around. Furthermore, the definition of a given systolic array is largely independent of most of its PE specifications. In other words, architects may choose to vary the complexity of their processing elements according to their algorithm/application needs. Systolic-based architectures have consistently outperformed CPUs and GPUs over the years (Borkar et al. 1990) (Pedram, Geijn, and Gerstlauer 2012). Recently, they have been extensively utilized to accelerate neural

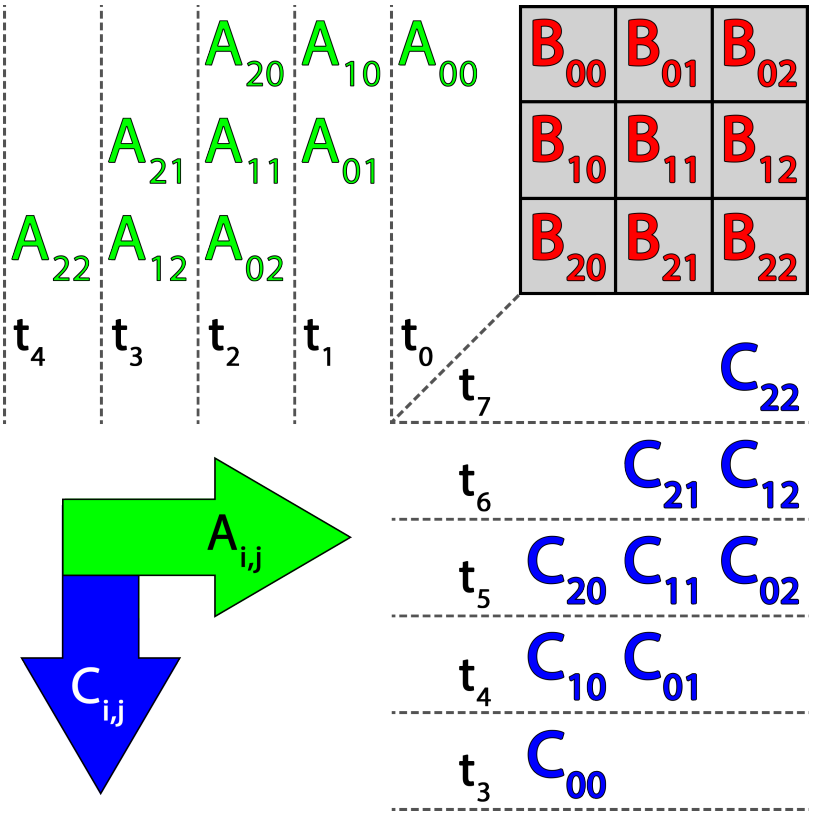

Figure 8: The functioning of a generic NxN weight-stationary systolic array for matrix multiplication.

The calculation in this example is $AxB=C$. The values of B are pre-loaded into the array. Then the values of A flow from left to right, while accumulations are propagated from top to bottom. The timing for inputs and outputs is specified as tx.

networks: Google's Tensor Processing Unit (TPU) (Jouppi et al. 2017) representing ASICs, and Xilinx's Deep Processing Unit (DPU) (Xilinx 2019b) representing FPGAs.

The characteristics of modern FPGAs make them excellent candidates for implementing specialized architectures and accelerating workloads. In particular, the high-speed DSP slices present in Xilinx devices can be used as the foundation for processing elements. These dedicated structures are able to run at much faster clock frequencies than the rest of the logic elements on the fabric, which enables them to be easily time-multiplexed for doubled/quadrupled throughputs (Xilinx 2019a). At the

same time, when dealing with 8-bit integer operators, bit slicing techniques can be used to pack two multiplications together and double the unit's throughput once more (Véstias et al. 2017) (Xilinx 2017). Furthermore, FPGAs have the aforementioned added benefit of design flexibility: If the application's needs happen to change over time, or if bugs need to be mitigated, a new configuration bitstream can easily be loaded into the device.

2.3 Radiation Effects

Radiation is a naturally occurring phenomenon that can be described as the transmission of energy through the interaction of atomic and subatomic particles. As previously mentioned, space is the main source of radiation, including a wide range of particles, which vary in mass and energy. Specifically, while stars are sources of visible light, they may also emit waves further along the electromagnetic spectrum, such as UV, X-ray, and gamma (Coblentz 1914). In addition, galactic cosmic rays originate from supernovas and other significant astronomical events, eventually reaching our solar system and forming a high-energy radioactive background (NASA 2021e).

Luckily, the majority of particles from outer space do not end up reaching Earth's surface, as its magnetic field acts as a natural protection mechanism. However, particles tend to remain trapped inside our planet's magnetosphere, forming the Van Allen belts of radiation (Gussenhoven, Mullen, and Brautigam 1996), which extend from about 640 to 58000km of altitude (Fig. 9). As a result, the development cycle of satellites and spacecraft must consider the specificities of deployment environments, as a way of preparing for inevitable radioactive interactions. Although extremely helpful, Earth's magnetic shield is not perfect. In fact, particle fluxes vary not only

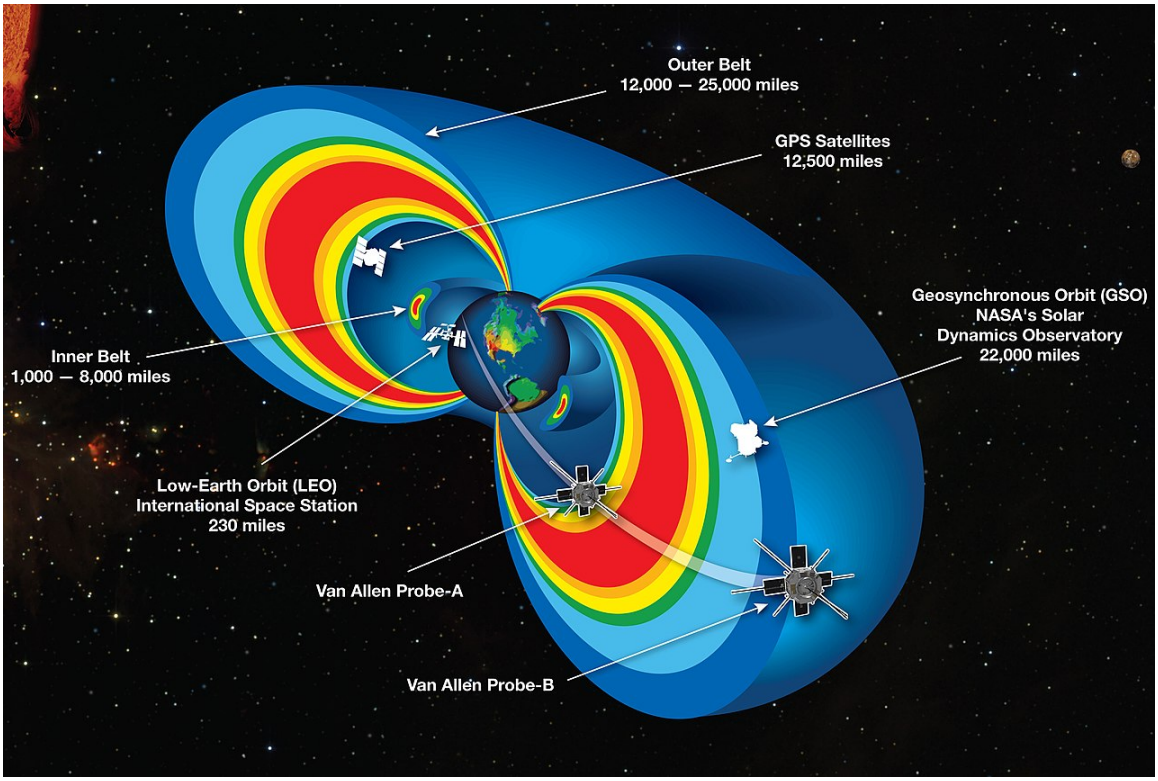

Figure 9: The Van Allen belts of radiation
Source: NASA (2021d)

with altitude, but also with latitude and longitude (Normand and Baker 1993), as the magnetic poles are not perfectly aligned, and the field is not normally distributed.

If sufficiently energetic, cosmic rays end up reaching our atmosphere, where their interaction with nuclei generates a shower of secondary particles, such as alpha, protons, and (mainly) neutrons. Even though neutron-induced upsets were not a major concern a few decades ago, the rapid advancements in silicon fabrication processes have led to extremely small transistor sizes, which in turn made modern electronic devices vulnerable to sea-level radiation (Baumann 2005) (Quinn and Graham 2005). Besides space, there are also a number of man-made radiation-rich environments, such as

particle accelerators and nuclear reactors. Evidently, critical control systems in those surroundings must also account for potentially harmful interactions.

Different particles interact in distinct ways with each type of electronic device. Ionizing radiation disrupts the functioning of silicon transistors by generating electron-hole pairs within the substrate. If enough charge generation occurs under the channel, temporary *OFF-ON* switches can take place. Non-ionizing radiation does not deposit any charge. Instead, neutrons leave a trail of ionization on the silicon, which only then creates charged particles capable of causing unintended events.

Radiation interactions from individual particles are referred to as Single Event Effects (SEEs). Even though SEEs can be classified as Destructive or Non-Destructive (Fig. 10), we only focus on the latter. Non-Destructive SEEs affect logic signals and storage elements. Induced pulses, such as Single Event Transients (SETs) can be *captured* by clock edges and written into registers and memory. If, for instance, a corrupted register happens to hold the relevant information within the context of a Finite State Machine (FSM), the circuit may cease to respond, which categorizes as a Single Event Functional Interrupt (SEFI). Finally, if a particle happens to change the content of a memory cell, it is named a Single Event Upset (SEU).

From the perspective of SRAM-based FPGAs, SEUs affecting their CRAMs are the main concern (Fig. 11). Since the logic behavior programmed on the fabric depends on the content of the configuration memory, upsets can alter circuit functionality. Moreover, configuration upsets will exhibit a persistent effect until corrected. This means that SEUs will likely affect multiple subsequent operations, instead of only a single one. Finally, the incidence of SETs on FPGAs is comparatively low, as their logic elements typically do not run at the high frequencies of ASICs/CPUs/GPUs (Keren et al. 2019).

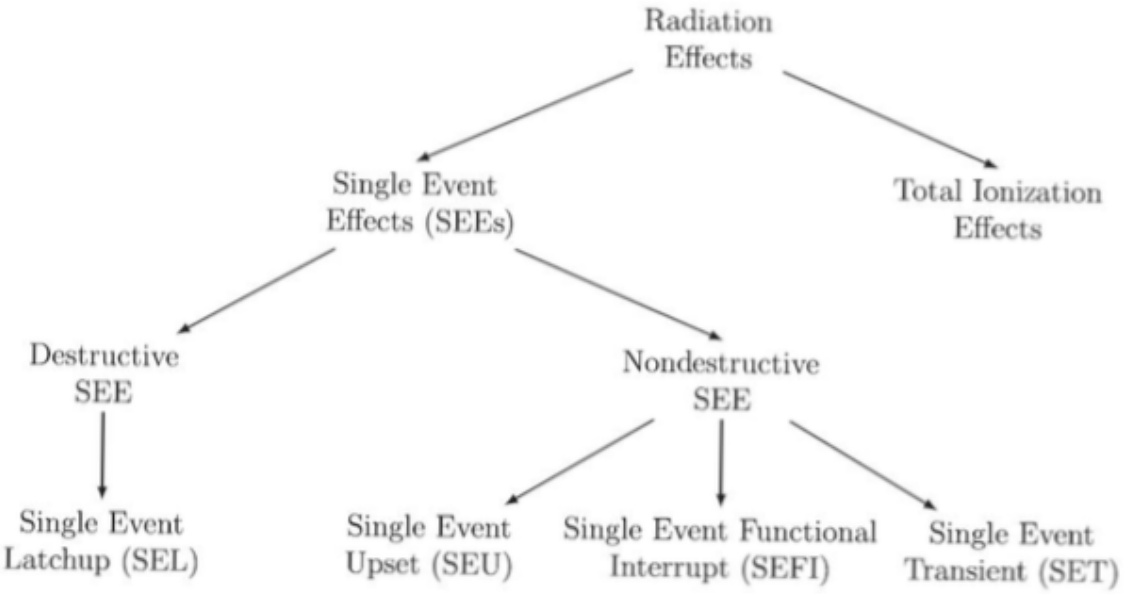

Figure 10: The radiation effects tree

Source: Siegle et al. (2015)

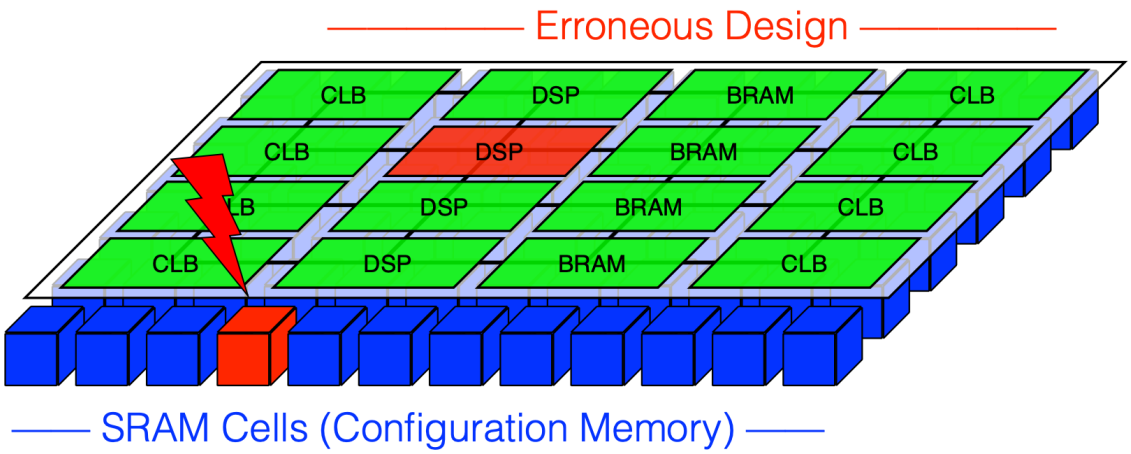

Figure 11: Naive representation of an SEU in an SRAM-based FPGA

2.4 Reliability & Safety-Critical Applications

Reliability has been a concern since the early days of computer history. In the pre-transistor era of the 40s and 50s, computers used to be made out of vacuum tubes, punched paper tape, and other sorts of rather untrustworthy components (Computer History Museum 2020). Thus, out of need, some of the cornerstone techniques in this field, such as duplication with comparison and triplication with voting, were first thought of and published by (von Neumann 1956). Then, in the 60s, the concept of fault-tolerance was first introduced (Pierce 1965) and expanded upon (Avizienis 1967), culminating with the formation of IEEE's Technical Committee on Fault-Tolerant Computing (1970), establishing a consistent set of concepts and terminology.

More recently, (Avizienis, Laprie, and Randell 2001) defined reliability and safety as *attributes* of dependability. In fact, the concept of dependable systems is broader, including other attributes such as availability, confidentiality, integrity, and maintainability. However, not all of them are within the scope of this dissertation. Furthermore, fault tolerance is defined as one of the *means* to attain dependability, as it is *"intended to preserve the delivery of correct service in the presence of active faults"*. Finally, the author enumerates *threats* to dependability. He states that a *fault* is active when it produces an *error*, which is a system state capable of generating a *failure*.

Specifically, safety-critical applications need to employ fault tolerance mechanisms as a way of avoiding catastrophic failures. Since reliability is a measure of continuous delivery of correct service, and safety is a measure of continuous safeness, the latter is often interpreted as an extension of the former. In most engineering contexts, reliability curves are plotted as exponential distributions of the form $R(t) = e^{-\lambda t}$, where λ is an estimated (or experimentally measured) failure rate. When it comes to radiation

environments with an average particle flux (i.e neutrons at sea level), it is possible to calculate the expected Failure-In-Time (FIT) rate simply multiplying such flux by the probability of a particle generating a failure. Said probability is experimentally measured as the Cross Section, as will be explained in Chapter 4. Naturally, by taking the inverse of FIT, we end up with a Mean Time To Failure (MTTF). Then, after dividing MTTF by the average execution time of an algorithm/application, we end up with the Mean Executions Between Failures (MEBF) metric, which, for us, is a more complete indicative of reliability, as it takes performance into account.

Since catastrophic failures can lead to human casualties, severe environmental damage, and significant financial losses, being able to probabilistically guarantee high levels of reliability/safety becomes mandatory. International standards, such as IEC61508 (International Electrotechnical Comission 2020) and ISO26262 (ISO 2021), exist for different classes of applications, regulating the set of solutions that can be explored in commercial products. However, the adoption of efficient fault tolerance techniques involves extensive research, development, and deployment costs.

Simultaneously, budgets vary widely across the spectrum of safety-critical applications. As mentioned, aerospace and military sectors were pioneers in terms of high-reliability requirements, given the harsh conditions of their environments of operation. For instance, a fighter jet worth hundreds of millions of dollars (Military Machine 2021) has the luxury of using expensive hardened-by-design parts, as they only represent a small fraction of the total cost. However, using a $100,000 FPGA in a $50,000 passenger vehicle is prohibitive. Therefore, as a way of maintaining reasonable prices for products aimed at common day-to-day users, fault tolerance needs to be added on top of cheap and widely available COTS devices.

2.5 Traditional Hardening Techniques

Hardening techniques involve some form of redundancy. Specifically, there are two types of redundancy: temporal and spatial. Temporal redundancy is usually applied at the software level, by executing a set of critical instructions multiple times. The word *set* here should be interpreted formally, since this set's cardinality can vary widely: We may choose to apply temporal redundancy to a function call, or to an entire program. Redundancy in time is ideal for mitigating transient faults, but, as we have mentioned, SETs are not very frequent in FPGAs. Moreover, if we were to perform a given operation multiple times in a (persistently) faulty circuit, we would get the same erroneous output over and over again. Therefore, spatial redundancy ends up being the most suitable alternative, in which computation is distributed across redundant hardware modules. However, redundancy always comes with a price, which is context-dependent, but quite self-explanatory: temporal redundancy costs time, spatial redundancy costs space, and both cost energy.

When error detection capabilities are sufficient, having Dual Modular Redundancy (DMR), or, equivalently, Duplication With Comparison (DWC) (Johnson et al. 2008), is one of the solutions. Given two redundant modules and a comparator, it is possible to detect that something went wrong. However, when something does go wrong, it is impossible to determine which module produced the error. As illustrated by Fig. 12, only one of the redundant outputs is passed along, which means that a DWC system is equally probable of producing *detections* and *false detections* (Aranda, Reviriego, and Maestro 2018) (González-Toral et al. 2018). Moreover, faults can also affect the comparator, in which case errors can go *undetected*. Table 1 provides all possible outcomes of a DWC scenario, assuming a simple 1-bit output.

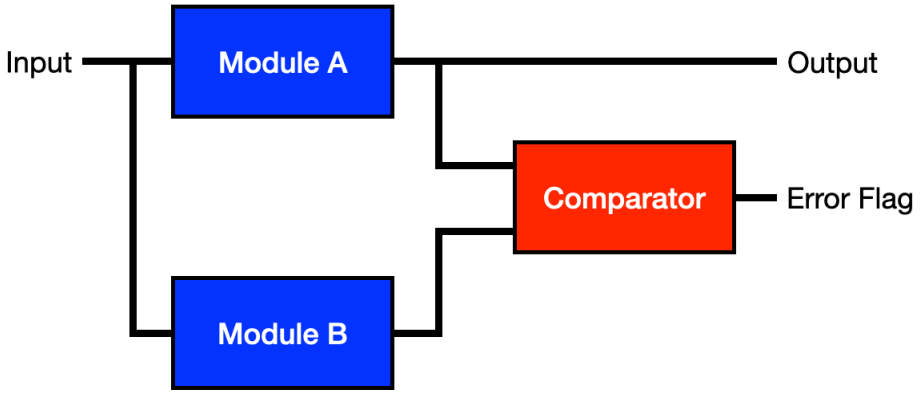

Figure 12: Duplication With Comparison (DWC)

Table 1: Possible outcomes with DWC.

Golden Output	A Output	B Output	DWC Status
0	0	0	Ok
0	0	1	Falsely Detected
0	1	0	Detected
0	1	1	Undetected
1	0	0	Undetected
1	0	1	Detected
1	1	0	Falsely Detected
1	1	1	Ok

Assuming that 'A' is the system's output.

To add correction capabilities to the system, Triple Modular Redundancy (TMR) (Kastensmidt et al. 2005) is required. The key difference of TMR, compared to DWC, is that, instead of simply comparing, a majority voting is performed. This allows for online fault-masking, eliminating errors altogether, as long as only one module is affected at the time. For even greater protection, voting must be performed bitwise. For instance, if each of the redundant modules outputs 32 bits, then TMR can correct multiple simultaneous faults, as long as they occur in different bit positions. However, the success of TMR is inherently dependent on the hardness of the voter. Some

systems opt for also triplicating the voters, or for using self-checking implementations, as a way of eliminating all single points of failure (Afzaal and Lee 2018).

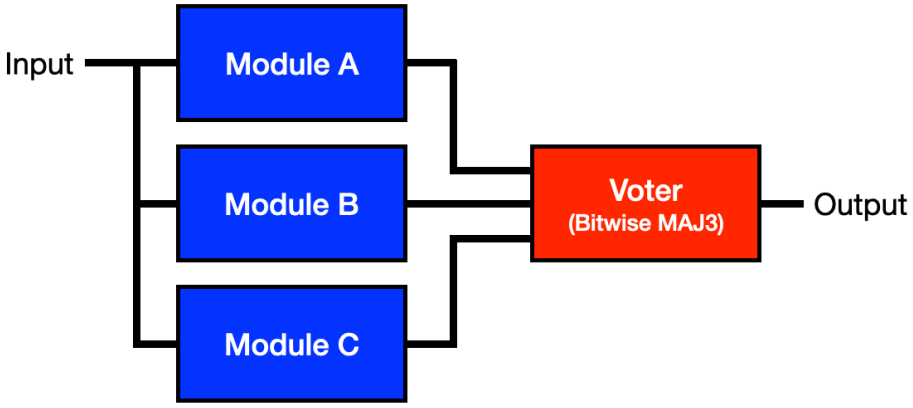

Figure 13: Triple Modular Redundancy (TMR)

In the context of FPGAs, redundancy-based hardening techniques must be paired with reconfiguration. For DWC systems, partial/full reconfiguration (R. Zhang et al. 2020) mechanisms can be targeted and eventual (i.e. only triggered on detections). For TMR systems, reconfiguration is typically cyclical and periodic (i.e. always running on the background (Heiner et al. 2009)). In fact, the gold standard for rad-hard FPGA systems is the combination of TMR'ed modules with a scrubbing mechanism. However, TMR'ing is not always feasible due to its inherent 200+% overhead. For instance, if a given module occupies 40% of the logic resources on a given FPGA, a TMR version of said module is impossible to place. Moreover, as resource utilization increases, routing efforts become more difficult, meaning that TMR is also known to negatively impact performance (Sterpone and Battezzati 2008) (Anwer, Platzner, and Meisner 2014).

2.6 Directly Related Work

Besides the foundational topics already discussed in this chapter, it is important to also provide a deeper overview of three key areas in which the main contributions of this dissertation fall into:

- **Fault Modeling:** A set of both analytical and empirical studies regarding fault propagation and error patterns across the computing stack, which allow for the development of more efficient hardening techniques (i.e. delivering high levels of hardness while costing much lower than traditional methods).
- **Algorithm-Based Fault Tolerance:** Existing strategies for error detection and error correction that explore algorithm (Matrix Multiplication) properties for improving reliability at lower-than-average costs.
- **Application-Specific Fault-Tolerance:** Hardening techniques for error detection and error correction that explore application (Neural Network) properties for improving reliability at lower-than-average costs.

2.6.1 Fault Modeling

A great number of works are predominantly concerned with fault modeling in a variety of contexts. For instance, (Wu et al. 2016) investigated the error patterns that emerged in matrix multiplication as a result of spatially and/or temporally separated faults in arithmetic and/or memory components, within the context of GPU-based clusters for High-Performance Computing (HPC). Some of the error patterns discussed by the authors are also observed in our experiments, despite occurring for different reasons, to be explained in Chapter 6.

From another standpoint, (M. Zhang et al. 2018) analyzed the potential impact of permanent fabrication defects in a TPU-like systolic array, through the injection of faults directly at the gate-level netlist of a synthesized design. The authors reported that, after introducing only 0.005% of *faultyness* to the array architecture, the test accuracy on their case-study neural network dropped steeply from a baseline of 74% all the way to 39%.

Moreover, (Ruospo et al. 2020) conducted software-based fault injection experiments corrupting weight values of floating-point and fixed-point variants of case-study CNNs. They reported weight corruptions in convolutional layers to be significantly more likely to generate classification errors than weight corruptions in inner product layers, due to inherent weight reutilization that exists in convolution.

Our fault model analysis is dissimilar to these works mainly because the persistent faults in FPGAs alter the *functioning* of the circuit, as opposed to simply disturbing single operations or data (inputs/outputs). Moreover, we analyze the impact of fault propagation across the computing stack, through the architecture and algorithm, all the way up to the application level.

2.6.2 Algorithm-Based Fault Tolerance

ABFT techniques go beyond naive redundancy as they tend to provide similar error detection and correction capabilities, but with considerably lower overheads. This is because good ABFT strategies are typically based on a deep knowledge of the algorithm's characteristics and fault model. In fact, the first-ever ABFT scheme was proposed specifically for error detection and correction on matrix operations, back in the 80s (Kuang-Hua Huang and Abraham 1984). We will revisit the authors'

contributions in the following sections, while highlighting two of its shortcomings (no consideration of persistent faults and underestimation of the algorithmic/architectural overheads).

More recently, an extended version of the MxM ABFT (Rech et al. 2013) addressed the original's limited correction capability (single errors only). By adding a number of extra steps (and computation) to the data reconstruction process, they reported the correction of a wider variety of error patterns. We, however, go in the opposite direction of the spectrum, eliminating the overhead necessary for error correction, and solely focusing on extremely low-cost detection. Since the majority of faults in SRAM-based FPGAs have a persistent effect in their configuration memories, without (partially) reconfiguring the device all subsequent computations would very likely trigger the algorithmic correction procedures. Furthermore, correction mechanisms described in an algorithmic context often turn into architectural implementations with significant area overheads, which further increase radiation sensitivity. Finally, for not-so-harsh deployment environments with a low expected error rate (i.e. terrestrial), the cost of recovery (recomputation) is diluted across all the error-free executions, which effectively means that paying for (rarely utilized) error correction capabilities ahead of time may be a sub-optimal design decision.

Beyond matrix multiplication, ABFT techniques are also well established for other pervasive algorithms, such as Fast Fourier Transform (FFT). In (Jou and Abraham 1988), the authors formally introduce an error detection scheme for single faults in complex adders or multipliers within butterfly stages. Using a matrix notation to define the FFT operation, the authors introduce redundancy of information through checksums, encoding, and decoding steps. Even though we are not focused on FFT, the

contributions in (Jou and Abraham 1988) indicate that modeling general algorithms with matrix operations often enables efficient ways of adding fault tolerance.

2.6.3 Application-Specific Fault Tolerance

The current prominence of neural networks (more specifically CNNs), and the desire to use them in real-time safety-critical applications, has significantly skewed research interest towards the proposal of CNN-specific hardening strategies. However, early work dates back to the 90s (Sequin and Clay 1990), where authors suggest a training procedure in which neurons are disabled at random for each epoch, leading to a more stable set of weights at the end. Nevertheless, the proposed technique was only validated for a 14-neuron topology, and becomes too expensive for state-of-the-art CNNs. Likewise, (Phatak and Koren 1995) recognizes that ANNs have an intrinsic baseline level of redundancy, and thus reliability. They point out the increasing cost (and vanishing benefits) of modular redundancy, while arguing the sufficiency of partial fault tolerance, depending on case-by-case hardness requirements.

Most notably though, (Hari et al. 2021) very recently presented a low-cost, checksum-based, error detection technique for convolution operations, and integrated their solution within standard ML flows to facilitate the deployment of hardened CNNs in state-of-the-art GPUs. The authors reported arithmetic overheads between 1% and 7%, while runtime overheads were between 6% and 23% in their case-study neural networks. An experimental demonstration of the technique's ability to detect all errors in convolutional layers was also provided. Such work builds on top of the algebraic observations made by (Marty, Yuki, and Derrien 2018), regarding convolution. The

authors of (Marty, Yuki, and Derrien 2018) were particularly interested in mitigating faults derived from overclocking, as opposed to radiation-induced.

Furthermore, (Santos et al. 2019) implemented the aforementioned MxM ABFT in convolutional and inner product layers of CNNs, while also proposing a novel ABFT technique for pooling operations, ultimately achieving over 90% correction of critical SDCs when executing state-of-the-art neural network topologies in GPUs.

For computer-vision, spatio-temporal correlations in CNNs' input image frames can be exploited, achieving over 80% error detection while adding less than 5% of runtime overhead in a CPU execution (Draghetti et al. 2019).

Specifically on FPGAs, our prior work has shown that the majority of errors in ANNs can be considered tolerable (Libano et al. 2018). Moreover, through fault injection campaigns, we evaluated the vulnerability factors of different layers in traditional MLPs and CNNs. A selective redundancy approach was experimentally validated, achieving over 40% of fault masking with a marginal 8% overhead in resource utilization, and 0% overhead in performance (Libano et al. 2019).

Chapter 3

CASE STUDY DEVICES

Although this dissertation is focused on FPGAs, our case study devices are heterogeneous System-on-Chips (SoCs), which means that they have both an FPGA and a CPU on the same die. Such devices were chosen because their heterogeneity allows for a simpler experimental setup (to be detailed in Chapter 4). Fig. 14 simplistically illustrates the chip architecture, which, by Xilinx's nomenclature, is divided into Processing System (PS) and Programmable Logic (PL), for CPU and FPGA, respectively. Data can be exchanged back and forth between PS and PL using a shared memory and its corresponding interfaces.

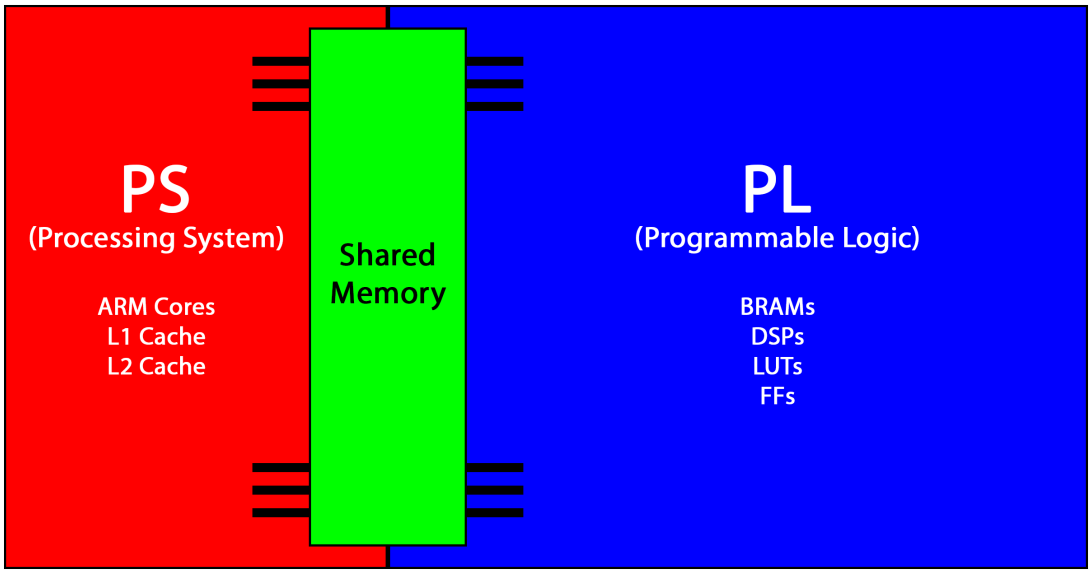

Figure 14: A simplistic architectural overview of Xilinx's Zynq devices.

3.0.1 Resource Availability

Resource availability varies significantly across Zynq devices. Depending on the demand for resources of a given DUT, a different target device must be chosen. For some of our experiments, we utilized the entry-level Zynq-7000 XC7Z020 part. For testing larger designs, we utilized the top-tier Zynq UltraScale+ ZU7EV and ZU9EG parts. Table 2 summarizes resource availability for each.

Table 2: Resource availability on the XC7Z020, ZU7EV, and ZU9EG.

	XC7Z020	ZU7EV	ZU9EG
BRAMs	140	312	912
DSPs	220	1,728	2,520
LUTs	53,200	230,400	274,080
FFs	106,400	460,800	548,160

Sources: Xilinx (2020g), Xilinx (2020f).

3.0.2 DSP Slices

DSP slices are fundamental components arithmetic units on FPGAs. Xilinx calls slices as DSP48E1 and DSP48E2 for Zynq-7000 and Zynq UltraScale+ families, respectively. The internal structure of DSP48E1 is shown in Fig. 15. In our context, DSP slices are always configured as MAC units, since multiply-accumulate is the main operation in matrix multiplication (and in neural networks). Furthermore, when using 8-bit integers as inputs, it is possible to use the pre-adder to pack two simultaneous multiplications together, given one shared operand (Véstias et al. 2017) (Xilinx 2017). Fig. 16 illustrates it with the specific bitwidths shown in Fig. 15. This scheme is used in our systolic array implementation (Chapter 6).

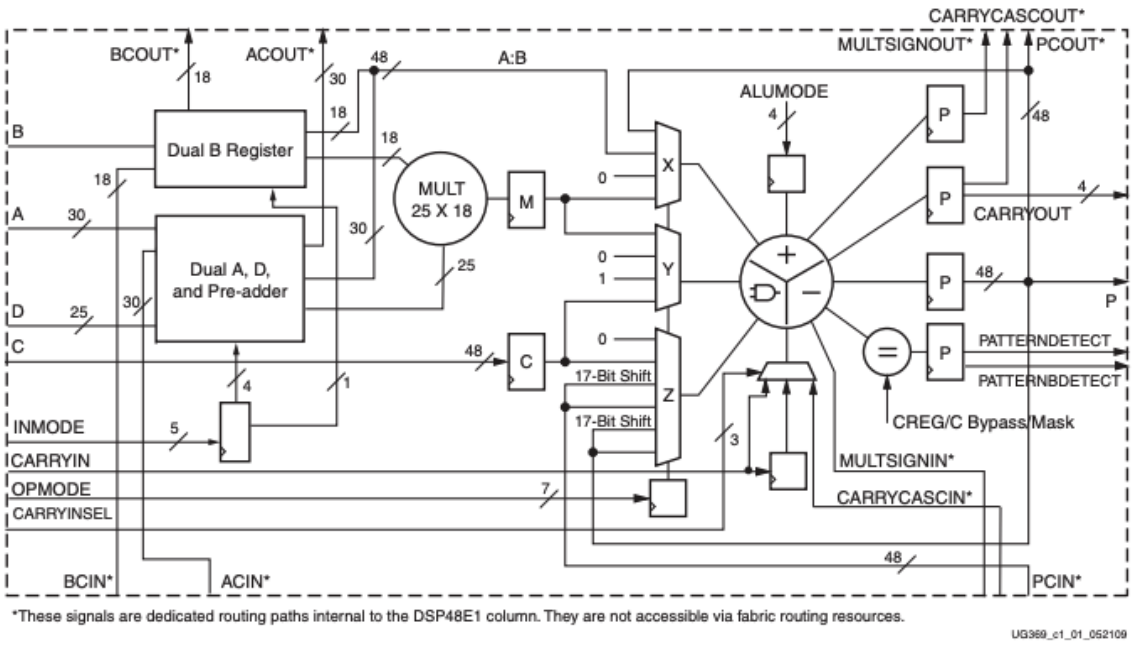

Figure 15: DSPs on the PL part of the Zynq-7000

Source: Xilinx (2020a)

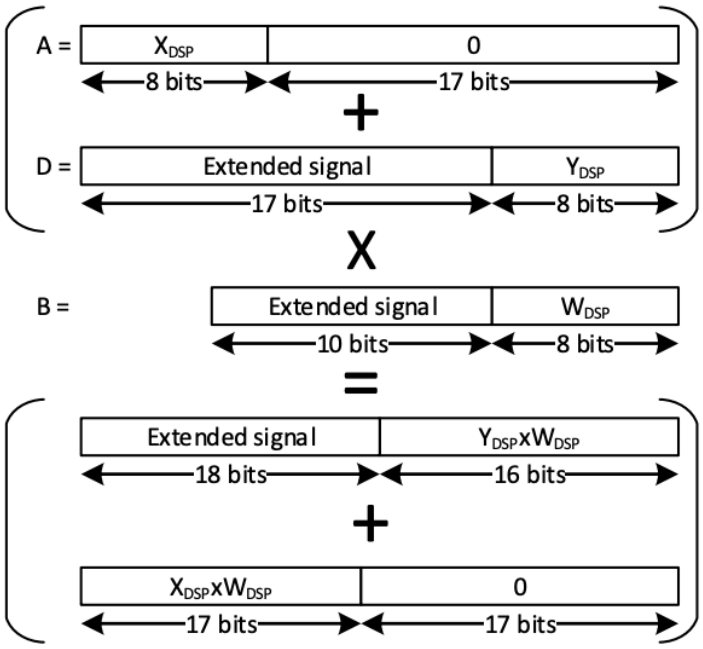

Figure 16: Packing two simultaneous 8-bit multiplications in a DSP48E1 slice.

Source: Véstias et al. (2017)

Chapter 4

EXPERIMENTAL METHODOLOGY

Among the methods to evaluate SEEs on electronic devices, the most truthful one is taking the device to its real application environment (satellites to space, for example). However, due to practical and/or financial constraints, this approach tends not to be a viable option. Moreover, it may take a long time (or a very high number of devices) to collect a statistically significant amount of data. As a result, accelerated radiation tests are the most common way to qualify the reliability of integrated circuits for SEEs (JEDEC 2006).

During beam experiments, devices are exposed to a radiation source with a much higher flux than the normal levels experienced in real application environments, which effectively helps to emulate years of operation in hours of experiment. Accelerator installations provide a variety of particles. Neutron facilities such as the Los Alamos National Science Center (LANSCE) (Los Alamos National Laboratory 2020) in the United States, and ChipIR the Rutherford Appleton Laboratory (RAL) (Science & Technology Facilities Council 2020) in the United Kingdom, are suitable for testing devices to be used in terrestrial and avionic applications. Proton sources such as the Paul Scherrer Institut (PSI) (Paul Scherrer Institut 2020) in Switzerland, are capable of generating protons with enough energy to simulate solar flares. Heavy ions beams like the 8UD Pelletron at Universidade de Sao Paulo (USP) (Department of Nuclear Physics - USP 2020) in Brazil, are appropriate for evaluating devices destined to deep space. Some facilities also provide particle cocktails, which is the case of CERN High Energy Accelerator Mixed Field (CHARM) (CERN 2020) in Switzerland.

A third way of qualification is fault injection by emulation or simulation. This method is, at the same time, the less costly and the most flexible, but it does not necessarily waive the execution of a radiation test. Usually, fault emulation is an attractive technique to predict the susceptibility of a system before submitting the device to an accelerated test. The approach consists in deliberately flipping bits of storage elements in the target device, through the assistance of an embedded circuit or a monitoring computer. Single Event Upsets (SEUs) can be emulated with random or sequential patterns, depending on desired coverage. Moreover, fault injection campaigns provide deeper information compared to radiation experiments, since the correlation between the injected fault and observed effect is known.

In this dissertation, both fault injection and radiation experiments are performed. The specifics of each experimental setup are described in the remainder of this chapter, along with their associated metrics. We also discuss the concept of error criticality, which is particularly relevant for classification tasks in neural networks.

4.1 Fault Injection

Xilinx provides two alternatives for fault injection in the FPGA's CRAM. In fact, the available resources are officially intended for partial reconfiguration of the device, but, they can be repurposed for SEU emulation (Xilinx 2020d). The most traditional method uses the Internal Configuration Access Port (ICAP), along with a specialized IP block called Soft Error Mitigation (SEM) core (Xilinx 2020e). However, since Zynq devices also have embedded ARM processors, they also allow for an additional reconfiguration method, through the use of the Processor Configuration Access Port (PCAP). Fig. 17 shows the configuration logic access on the Zynq-7000.

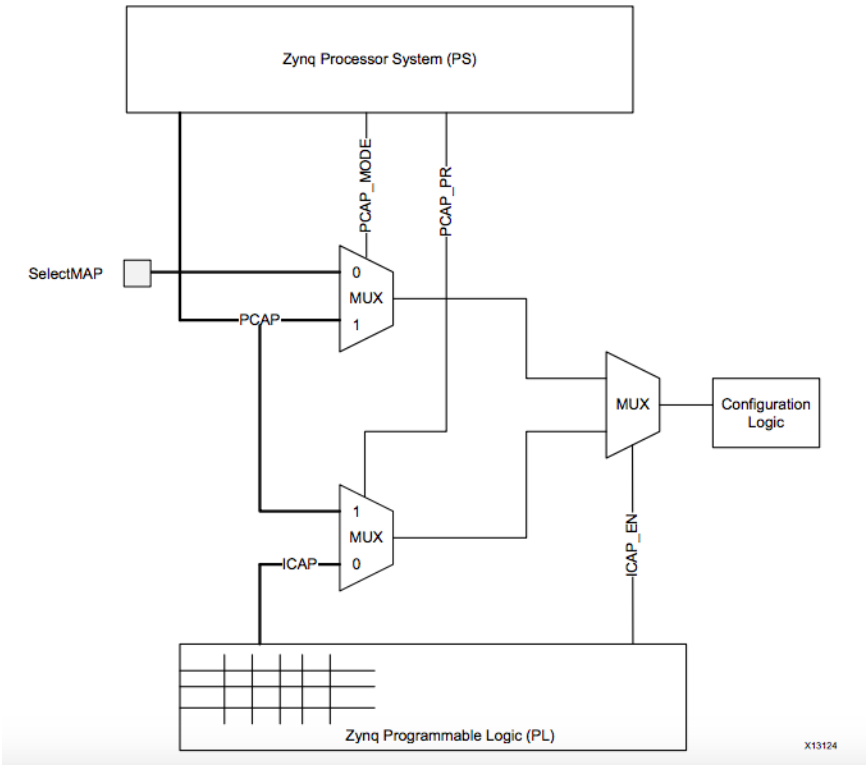

Figure 17: Configuration logic access on the Zynq-7000

Source: Xilinx (2020e)

Our fault injection procedures rely on both ICAP and PCAP, for the Zynq 7000 and Zynq UltraScale+ devices, respectively. Modules of interest within the architectural hierarchy are manually placed in separate *pblocks* on the programmable logic. Each pblock is a user-specified region of the fabric, containing a sub-set of resources. Given such placement information, it is possible to correlate a faulty component to its effect on the output. At this level we measure the *Architectural Vulnerability Factor (AVF)*, representing the probability for an injected fault to manifest at the output as a visible error/failure (Mukherjee et al. 2003). The AVF is simply calculated as:

$$AVF = \frac{(\#Errors)}{(\#Injections)}$$

4.2 Radiation Experiment

4.2.1 Laboratories & Facilities

Neutron beam experiments were conducted at the LANSCE facility of the Los Alamos National Laboratory (US), and at the ChipIR facility of the Rutheford Appleton Laboratory (UK). As shown in Fig. 18, both facilities provide a neutron spectrum that mimics the atmospheric one (particularly up to 10MeV, which encompasses the majority of particles at sea-level (Normand and Baker 1993)), and thus are suitable to emulate terrestrial neutron effects in electronic devices. Assembled experiments in each facility are shown in Fig. 19 and Fig. 20.

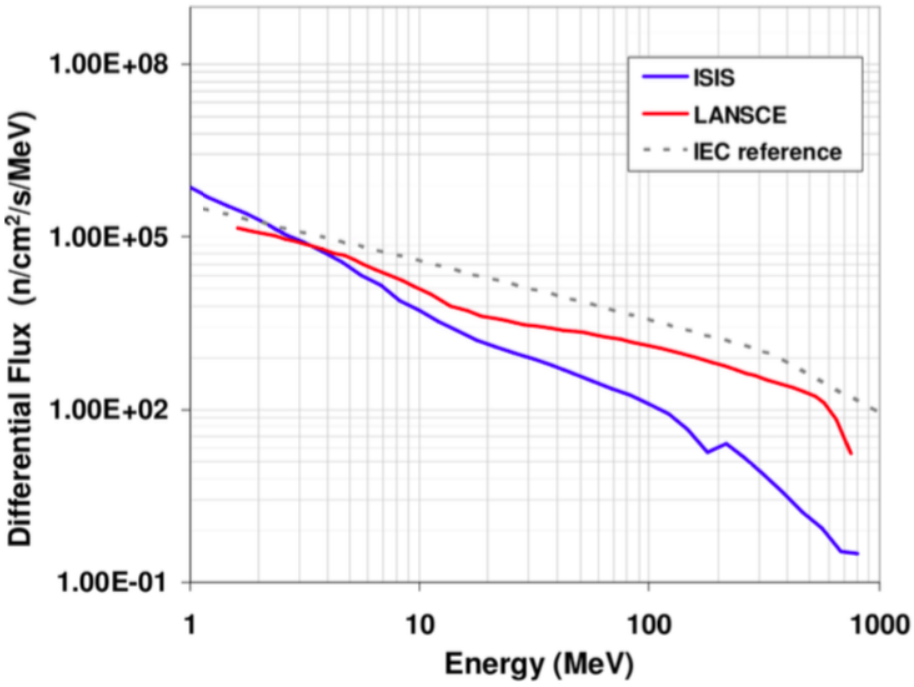

Figure 18: Neutrons spectra for LANSCE and ChipIR.
The neutrons spectrum at sea level is multiplied by a factor of 10^8 and plotted as a reference. *Source*: Violante et al. (2007)

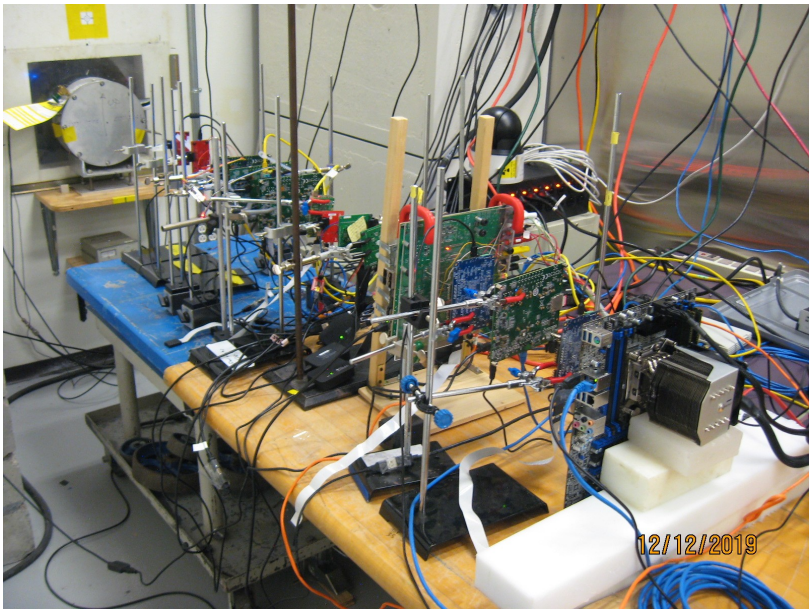

Figure 19: Radiation experiment at LANSCE.

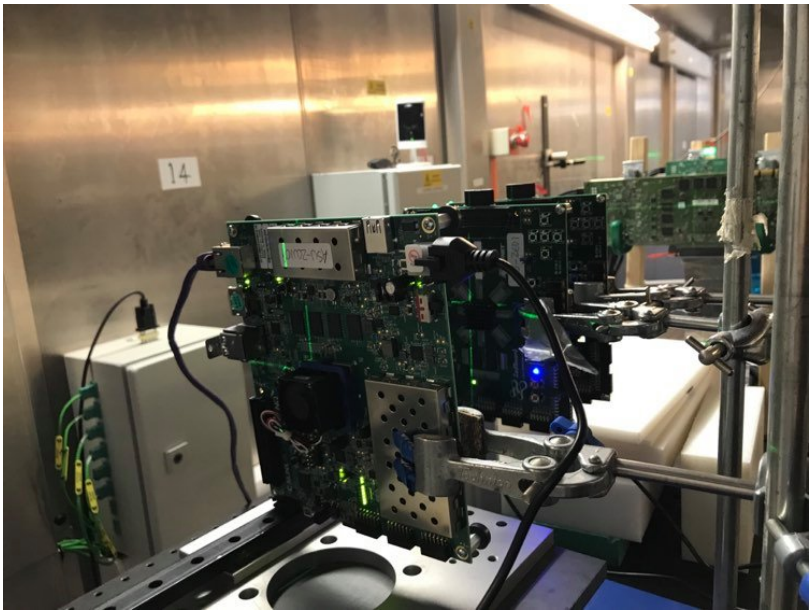

Figure 20: Radiation experiment at ChipIR.

4.2.2 Experimental Setup

This work's experimental setup for radiation tests takes advantage of the heterogeneity in Zynq devices. As the Design Under Test (DUT) is implemented on the PL part of the chip, the PS is responsible for feeding it with stimulus. At the same time, the PS periodically communicates with a computer that is responsible for monitoring the experiment from outside the beam room, informing whether the DUT is still *alive*, or if any output errors were observed (SDCs). The computer logs such events for posterior analysis, and has the ability to power cycle the DUT whenever necessary (when the system is unresponsive, or when a new bitstream needs to be loaded into the FPGA's configuration memory).

The communication between PL and PS is enabled through the use of Xilinx's AXI Direct Memory Access (DMA) IP, which acts as a bridge from the PL to the DDR controller, creating an illusion of direct data transfers. Furthermore, the communication between the PS and the outside world (i.e. the monitoring computer) is done through User Datagram Protocol (UDP) over Ethernet. The computer acts as a UDP server, and waits for packets to be sent by UDP clients (from inside the beam room). Note that this aspect of the setup makes it highly scalable, since multiple DUTs can be tested simultaneously, with no additional overhead other than the need for an ethernet switch (which is, by definition, transparent in a computer network). Conversely, the only apparent drawback of the setup is that the PS part of the chip is also exposed to radiation, and thus prone to errors. However, the area occupied by the ARM processors is much smaller than the area occupied by the FPGA (meaning that their failure rate is also much smaller), and the time necessary for resetting the device is, for the most part, negligible.

As a minor contribution to the field, the setup has been made available on GitHub (https://github.com/lllibano/RadiationSetup). The repository provides further technical details, and an usage example based on a trivial DUT.

In a radiation experiment, the most important metric is the *Cross Section*, which quantifies the system's sensitivity to radiation (JEDEC 2006). The cross section is simply measured dividing the number of observed errors (*# Errors*) by the total number of particles per unit area (typically cm^2) that hit the device (also known as *Fluence*).

$$\sigma = \frac{(\#Errors)}{Fluence}$$

4.3 Experiment Suitability

Each type of experiment is better suited for gathering a specific type of information. When performing beam tests, the entire device is irradiated and it is possible to obtain a more accurate prediction of the error rate in the deployment environment. However, errors can only be observed when they appear at the output, making it very difficult to deeply evaluate fault propagation through the DUT. When injecting faults, specific portions of the circuit are purposely corrupted, in order to correlate the fault source to the observed effect in the output. However, as faults are only being emulated, physical properties of the device/particles are not considered. Furthermore, the cross section is measured with a unit of area (cm^2), while the AVF, being a probability, is dimensionless. As such, within the context of this dissertation, we combine the information from both types of experiments to measure baseline radiation sensitivity, identify fault propagation patterns, and validate hardening strategies.

4.4 Error Criticality

In classification tasks, small output corruptions can be tolerated. This is because the final answer is obtained after comparing outputs for each class and determining the highest. To make this discussion more tangible, we may consider a hypothetical 3-class problem in Table 3. Notice that class 2 is the highest within the golden outputs. In the *tolerable* corruption example, class 0 is wrongly computed as 1.7, but class 2 remains as the highest value (therefore classification is correct). Differently, in the *critical* output corruption example, class 1 is wrongly computed as 9.1, which is higher than class 2 (therefore classification is wrong).

Table 3: Tolerable and critical corruptions in a hypothetical 3-class problem.

Class #	Golden Outputs	Tolerable Corruption	Critical Corruption
0	1.5	1.7	1.5
1	2.3	2.3	9.1
2	7.8	7.8	7.8

A more nuanced event can happen due to the inherent inaccuracy of classifiers. For instance, a trained model with a an accuracy of 98% is expected to wrongly classify 2% of the inputs. Thus, a significant corruption in the computation of a given input, supposed to be in said 2% group, can lead to a correct classification. We call such events as *benign*. Table 4 exemplifies with a 3-class problem.

Table 4: A benign corruption in a hypothetical 3-class problem.

Class #	Golden Outputs	Benign Corruption
0	1.5	1.5
1	2.3	9.1
2	7.8	7.8

The expected answer is *1*, but the model outputs *2*. The corruption is benign.

Chapter 5

THE RADIATION SENSITIVITY OF STATE-OF-THE-ART NEURAL NETWORK ACCELERATORS ON FPGAS

The contents of this chapter are based on a paper titled *"On the Reliability of Xilinx's Deep Processing Unit and Systolic Arrays for Matrix Multiplication"*, which was presented at the 2020 Radiation and its Effects on Components and Systems (RADECS). The majority of the text is reproduced verbatim, only with structural modifications.

5.1 Motivation

Before even considering hardening alternatives, one must measure the baseline susceptibility of an architecture/algorithm/application. Therefore, in this chapter we experimentally evaluate Xilinx's Deep Processing Unit (DPU) (Xilinx 2019b), running Google's InceptionV3 neural network (Jouppi et al. 2017), trained on NASA's Mars HiRISE dataset (Wagstaff and Lu 2017).

5.2 Background

5.2.1 Xilinx's DPU

As a way of offering users an easy and efficient way of running neural networks on its FPGAs, Xilinx has released a set of tools called *Vitis AI* along with their

Deep Processing Unit (DPU) IP (Xilinx 2019b). Given a ML model trained on an industry-standard framework such as TensorFlow (TensorFlow 2021), Xilinx's tools allow users to compile it, generating a set of instructions to be run on a DPU core. The DPU architecture is known to be systolic, similarly to that of Google's Tensor Processing Unit (TPU) (Jouppi et al. 2017).

5.2.2 Google's InceptionV3

As a case study for testing the DPU, we selected Google's InceptionV3 CNN architecture (Szegedy et al. 2016), which was the runner-up on the 2015 ImageNet Large Scale Visual Recognition Competition (ILSVRC). A high-level overview of the topology is shown in Fig. 21. The model is made up of 42 layers, organized in several blocks (at the end of which, filter concatenation is extensively used). In total, there are over 23 million learnable parameters. Over 5 billion arithmetic operations are necessary for processing each input image.

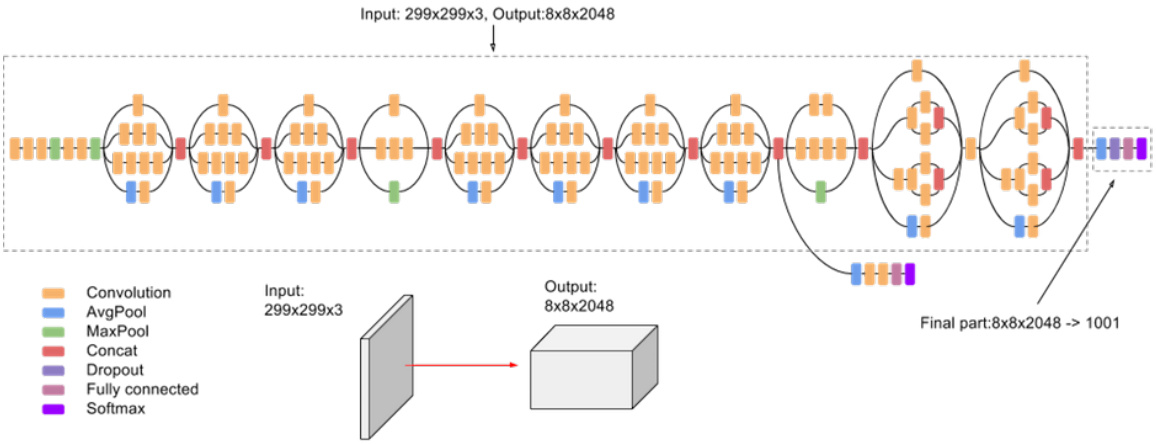

Figure 21: Topology of Google's InceptionV3
Source: Google (2020)

5.2.3 NASA's Mars HiRISE

We trained InceptionV3 on a real-world, expert-built dataset from NASA, achieving an accuracy of 93.54% (prior work only managed to achieve 90.6% (Wagstaff et al. 2018)). The dataset is made out of orbital images taken by NASA's *Mars Reconnaissance Orbiter*, whereas the acronym HiRISE stands for *High-Resolution Imaging Experiment* (Wagstaff and Lu 2017). The dataset is labeled as per Fig. 22.

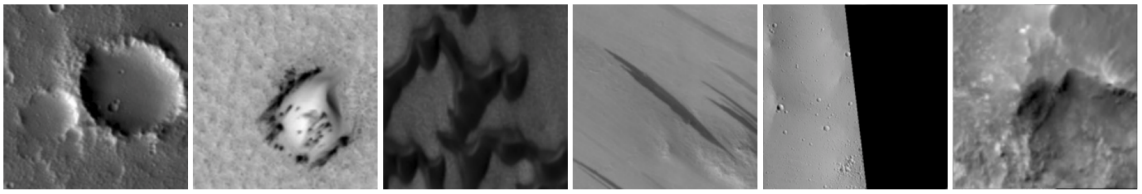

Figure 22: Image classes for NASA's Mars HiRISE dataset
Source: Wagstaff et al. (2018)

5.3 Experimental Methodology

We implemented three different configurations of Xilinx's DPU core on the 16nm FinFET Zynq Ultrascale+ MPSoC (Xilinx 2020f), embedded on the ZCU104 board. The details about resource utilization by the different DPU configurations can be seen in Table 5. The nomenclature of the designs is taken from Xilinx's own documentation (Xilinx 2019b), and represents the level of parallelism of the circuit, thus, directly correlates to how many operations it is capable of executing per clock cycle. The average execution times, for each DPU configuration to run our case-study neural network, were: 10.97ms, 7.85ms, and 6.14ms for B1024, B2304, and B4096, respectively.

Table 5: Resource utilization on the Zynq UltraScale+ (ZU7EV) to implement the DPU cores with increasing levels of parallelism.

DPU Configuration	DSPs	BRAMs	LUTs	FFs
B1024	154	46	34k	50k
B2304	326	63	43k	73k
B4096	562	85	56k	105k

Neutron beam experiments were conducted at the ChipIR facility of the Rutherford Appleton Laboratory, in the UK. We have collected a total fluence of about $4.3 \times 10^{11} n/cm^2$, which is equivalent to about 3.7 million years of natural exposure at sea level (JEDEC 2006).

5.4 Results

Fig. 23 shows the neutrons cross section of our DUTs. As the UltraScale+ is an heterogeneous chip (composed of PS and PL), the A53 ARM processor acts as a supervisor of the system, and is responsible for most of the control in the application, even though the DPU is implemented exclusively on the FPGA. Therefore, some instances where radiation-induced errors led to unresponsive/unrecoverable system state occurred and were classified as *Crashes*. As the PS part of the chip is always the same, it was expected to observe a similar crash rate across all three designs. However, the vast majority of events are Silent Data Corruptions (SDCs). As this is a classification task, we further divide the observed SDCs as *Tolerable* or *Critical* (according to our discussion in Section 4.4).

We can clearly see in Fig. 23 that, as we reduce the level of parallelism in our

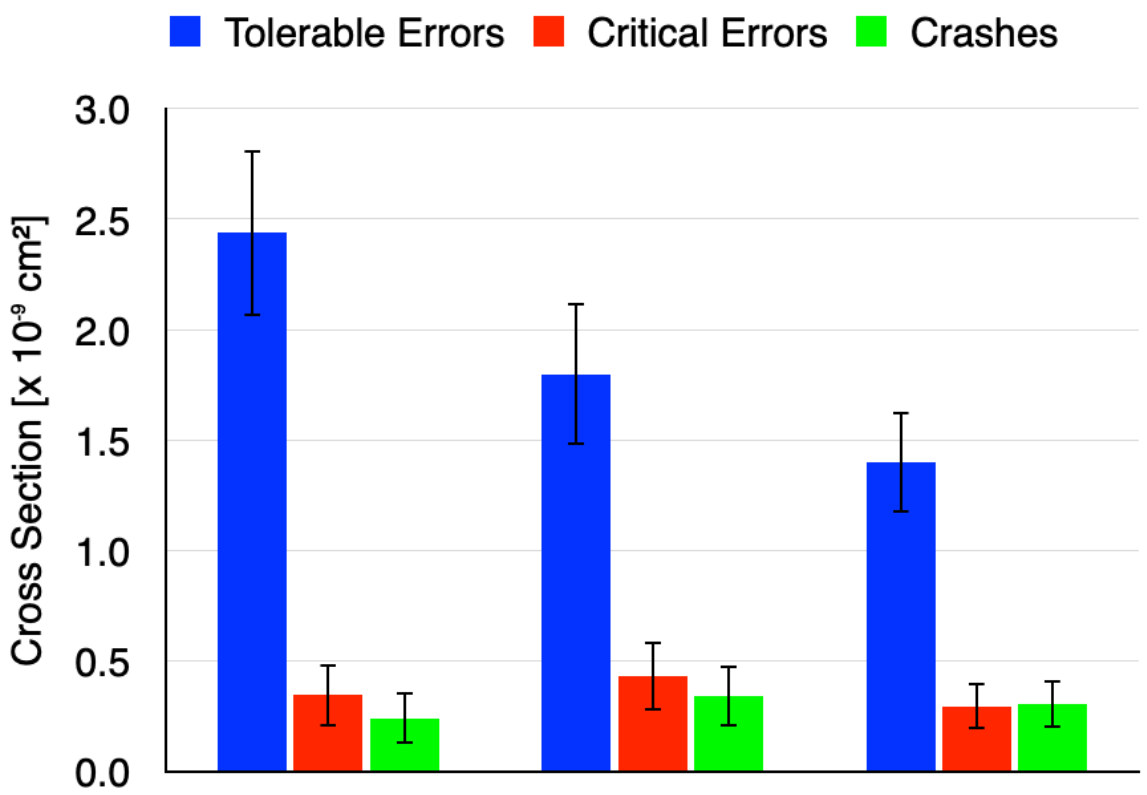

Figure 23: Neutrons cross section of the DPU core using three levels of parallelism (B4096, B2304, B1024).

DPU core, the cross section of the DPU is also reduced. Lower parallelism equates to lower resource utilization on the FPGA (as showcased in Table 5), which means that we also reduce the likelihood of an impinging particle hitting the circuit. The B2304 core was found to be 20% less sensitive to radiation than the top-tier B4096. Similarly, the B1024 core had a 14% lower SDC rate, compared to B2304.

When evaluating the error rate of a system, we must also account for performance. As previously mentioned, a less capable DPU core uses fewer resources, at the expense of increased execution time. Thus, in order to decide whether or not the cross section reduction actually pays off, we need to calculate the Mean Executions Between Failures (MEBF) metric (Rech et al. 2014). Multiplying the experimental cross section by the average neutron flux at a given environment (i.e airplane's cruising altitude (Taber

and Normand 1993)), gives us the Failure In Time (FIT) rate. The inverse of FIT gives us the Mean Time To Failure (MTTF). Finally, dividing MTTF by the execution time of our applications gives us MEBF. The values shown in Table 6 make it clear that, even though the B4096 core has the higher cross section, it also has the highest MEBF, which means that it will be able to, on average, complete a higher number of undisturbed executions (1×10^9 more than B2304, and 5×10^9 more than B1024). In other words, even though a smaller core uses less resources, it is subject to a higher number of particles during each computation. Therefore, a larger/faster DPU ends up being more reliable.

Table 6: Mean Executions Between Failures for the DPU core.

	B1024	B2304	B4096
MEBF [x 10^9]	59.59	63.32	64.76

Chapter 6

EFFICIENT ERROR DETECTION FOR MATRIX MULTIPLICATION WITH SYSTOLIC ARRAYS ON FPGAS

The contents of this chapter are, for the most part, based on a paper titled *"Efficient Error Detection for Matrix Multiplication with Systolic Arrays on FPGAs"*, currently under review at IEEE Transactions on Computers. The majority of the text is reproduced verbatim, only with structural modifications.

6.1 Motivation

As showcased in Chapter 5, state-of-the-art neural network accelerators are vulnerable to radiation-induced upsets. Although the DPU is Xilinx's IP (i.e. no architectural details are openly available), it is generally known that it is a systolic array, which effectively accelerates matrix multiplication operations. Therefore, the goal of reaching increased levels of reliability for neural networks on FPGAs can be achieved through the development of hardening techniques for matrix multiplication.

Moreover, matrix multiplication is commonly found at the core of a variety of compute-intensive applications, from high-performance computing, to low-latency radio, and image processing tasks. This means that the impact of enhancing the reliability of MxM extends much beyond just neural networks.

In particular, we aim to develop low-cost hardening techniques. As discussed in Chapter 2, traditional DWC and TMR strategies have inherent overheads of 100+% and 200+%, respectively. However, it is not always possible to duplicate/triplicate

a circuit that utilizes a high number of resources as-is. Furthermore, the adoption of modular redundancy is also known to hinder performance, since routing becomes more difficult as fabric congestion increases (Sterpone and Battezzati 2008) (Anwer, Platzner, and Meisner 2014), and, in the case of DWC, the added redundancy is also prone to producing *false detections* (i.e. when the output is correct, but a detection is wrongly signaled) (Aranda, Reviriego, and Maestro 2018) (González-Toral et al. 2018).

As an alternative, Algorithm-Based Fault Tolerance (ABFT) can be used to increase the reliability of specific applications with lower overhead than traditional modular redundancy, but, as we will discuss in later sections, existing ABFT methods for MxM (Kuang-Hua Huang and Abraham 1984) (Rech et al. 2013) are still sub-optimal for FPGAs. We show that, in order to maximize hardening efficiency (i.e. deliver maximum error detection rates with minimum added costs), one must first obtain an accurate fault model (which in turn heavily depends on target device, accelerator architecture, algorithm, and application).

6.2 Implementation Details

In this section, we present the implementation details of our high-performance systolic array. While this is not our main contribution, it is a necessary component of our work, as we use it, first, to study the fault model at the architectural level, and, later, to experimentally validate our novel error detection strategy. Nonetheless, we have decided to make it publicly available, along with a parametric RTL code generator (https://github.com/lllibano/LABFT) that allows anyone to deploy custom-sized arrays (with or without error detection capabilities) in a matter of minutes. Fig. 24 comprehensively illustrates our discussion.

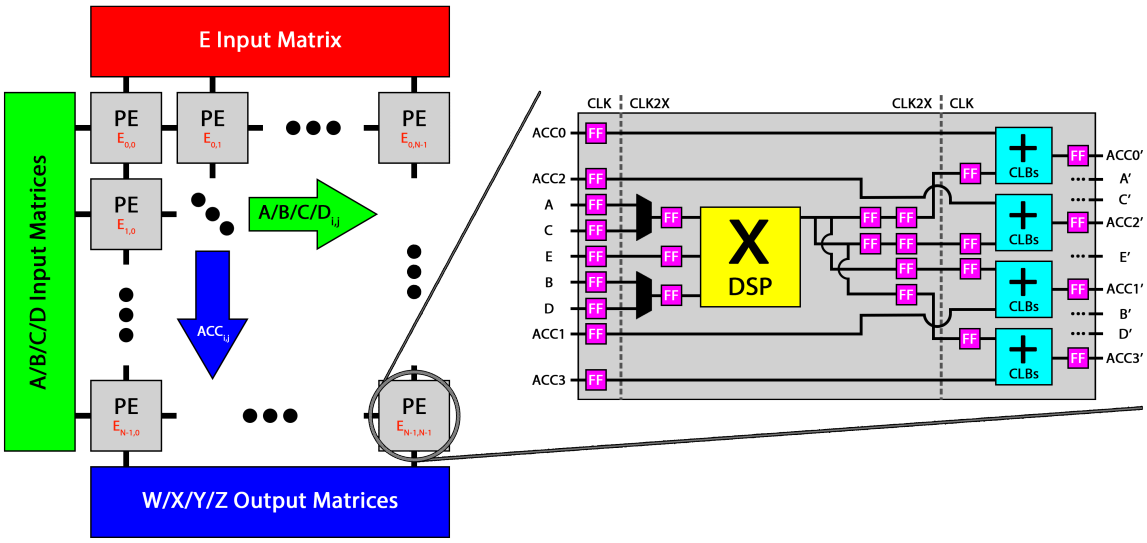

Figure 24: Our systolic array implementation, viewed from different granularity standpoints.

At the top level, input and output matrices get in and out of the unit through AXI-Stream interfaces (Xilinx 2020b). Inside the array, the values of E are pre-loaded into their respective PEs. Then, the values of $A/B/C/D$ flow from left to right, while the accumulation results are propagated from top to bottom. Inside each PE, we combine clever shifting and slicing of bits, with a standard time-multiplexing scheme, to achieve an effective throughput of 4 multiplications per cycle, using a single DSP. To match such degree of parallelism, we use CLBs to implement 4 adders.

Even though our MxM systolic array implementation is very simple and generic, there are two main particularities about it, both of which are closely related to current state-of-the-art neural network accelerators (such as Google's TPU (Jouppi et al. 2017) and Xilinx's DPU (Xilinx 2019b)). First, the inputs are *8-bit integers*. This is because industry-standard machine learning frameworks, such as TensorFlow (TensorFlow 2021), offer lossless quantization tools for reducing the precision of data representation in trained models. In other words, after the training process is complete (using 32-bit floating-point), the model goes through an additional step, in which it is essentially converted to an 8-bit integer equivalent, while maintaining the original accuracy level. Both the TPU and the DPU also use 8-bit integers as inputs. Second, our MxM array

is *weight-stationary*. This simply means that each element of the second input matrix is loaded into its respective $PE_{i,j}$, before the first input matrix starts flowing into the array (instead of both moving together). The reason for this design decision is twofold: having one stationary matrix mimics the behavior of convolutional filters in a CNN, and, it allows for optimal DSP utilization, as we shall explain next.

For matrix multiplication, the processing elements in the systolic array must be multiply-accumulate (MAC) units. In our PE, the center point is the DSP slice (as it should be, in any efficient FPGA arithmetic module). We simultaneously employ two techniques that allow us to maximize performance of each DSP. Such techniques are also known to be used in Xilinx's DPU (Xilinx 2019b). First, we take advantage of the adder that precedes the multiplier (conveniently named pre-adder) to execute $(A+B)\cdot E$. Given that the inputs are 8 bits wide, and that the pre-adder has a bitwidth of 25 and 27 (for DSP48E1 and DSP48E2, respectively (Xilinx 2019a)), it is possible to pack two 8-bit integer multiplications (AE, BE) into a single DSP, by shifting and slicing of bits. This DSP implementation is pretty much the same as (Véstias et al. 2017) (Xilinx 2017). Second, we adopt a simple and standard time-multiplexing scheme for the DSPs, as suggested by Xilinx (Xilinx 2019a). With proper muxing and instantiation of registers, we run the DSPs at twice the clock frequency used by the rest of the circuit. In short, the combined effect of these two techniques is a multiplier with a *parallelism factor of 4* (i.e. it *simultaneously* calculates AE, BE, CE, DE). In order to match this throughput, each PE also has four adders that are implemented with CLB elements (LUTs and CARRYs). At the top level, our implementation fully accommodates the degree of parallelism that can be accommodated by the PEs. In other words, there are four simultaneous matrix calculations being executed by the systolic array, at all times (W=AxE, X=BxE, Y=CxE, Z=DxE).

In terms of resource utilization, DSPs are the most important (and scarce) type of resource in modern Xilinx FPGAs. The DSPs allow designers to deploy fast arithmetic units, which means that an efficient FPGA accelerator must optimize their usage. Referring to Fig. 24, we can immediately see that our systolic arrays are *NxN*, and that each processing element uses a DSP. Therefore, DSP utilization is both predictable and parametric in our systolic architecture and code generator: an N-sized array will use N^2 *DSP48E{1,2}*.

In terms of performance, each of the N^2 PEs executes 4 multiplications and 4 additions at each clock cycle, totaling $8N^2$ *Operations Per Cycle (OPC)*. Then, if we multiply OPC by frequency, we get the Giga Operations Per Second (GOPS) metric. Since the maximum operating frequency depends on the target device, GOPS will vary. However, OPC exclusively depends on array size. In order to prove that our implementation is representative of the state-of-the-art, we compare two of Xilinx's DPU configurations (one *B1152* core, and three *B4096* cores) (Xilinx 2019b) to two of our design combinations (one *14x14* array, and two *32x32* arrays), constrained to the DSP budgets of two target devices (XC7Z020-3 and XCZU9EG-3). The numbers are shown in Table 7. The peak theoretical performance is considerably higher than that reported by Xilinx (36% and 19% for the 7 Series and the UltraScale, respectively), as is our DSP utilization (34% and 32%). Therefore, the numbers in Table 7 indicate that our MxM systolic array architecture represents the state-of-the-art for 8-bit integer computation on modern Xilinx FPGAs.

Table 7: Resource utilization and peak theoretical performance comparison between DPU configurations, and our design combinations

Device	**Design Combo**	**# DSPs**	**Frequency**	**GOPS**
XC7Z020-3	*B1152* x 1 (Xilinx 2019b)	146	200MHz	230.0
	14x14 x 1	196	200MHz	313.6
XCZU9EG-3	*B4096* x 3 (Xilinx 2019b)	1542	333MHz	4100.0
	32x32 x 2	2048	300MHz	4915.2

6.3 Experimental Methodology

In this section, we provide details pertaining to our fault injection methodologies. First, we run injection campaigns on the FPGA, to gain insights regarding the fault model of general MxM systolic arrays. Second, we run application-level campaigns, where we inject MxM error patterns, and quantify the criticality of persistent fault propagation in the context of the execution of CNN topologies with distinct complexities. Both of these steps are included in our multi-level fault propagation model (Section 6.4). Finally, we use our FPGA fault injection setup an additional time, to experimentally validate the effectiveness of our novel error detection strategy against persistent configuration upsets (Section 6.6).

In order to emulate SEUs in the FPGA's configuration memory, as well-established and well-validated in previous works (Di Carlo et al. 2014) (Tonfat et al. 2016) (Libano et al. 2018), we take advantage of the ICAP on an XC7Z020 device, as stated in Chapter 4. Generally, we use the PS to send/receive inputs/outputs to/from the DUT, as well as to perform a comparison between outputs and pre-computed golden values, as a way of evaluating the effect of every injected fault. A monitor PC logs all the relevant data for a later, more comprehensive analysis.

As a way of understanding and effectively measuring the consequences of persistent fault propagation to the application level, we have conducted fault injection experiments in a couple of case-study CNNs. More specifically, we have separately injected persistent random patterns within different MxM error categories (to be defined in Section 6.4) into the computation of convolutional and inner-product layers of each CNN, and evaluated its effects at the output. We have chosen to perform this set of experiments in software (as opposed to hardware) for (1) increased injection detail/control, and (2) speed of execution. Since we wanted to inject faults starting at multiple different steps (layers) of the CNN execution, being able to stop/continue computation partway through became mandatory. Each and every MxM operation is run in software as if it had to be broken down into blocks and executed in a parametric systolic array of size N. Thus, we are able to collect more data points (by running each CNN with multiple array sizes), while not compromising the preciseness of the fault model.

6.4 Multi-Level Fault Propagation Model

As previously stated, we are not the first to study the fault model of matrix multiplication (Rech et al. 2013) (Wu et al. 2016) (Gonçalves de Oliveira et al. 2016). However, prior work only describes faulty behavior as a result of transient events. As our focus is on FPGAs, our model revolves around the incidence of *persistent* faults. Furthermore, we aim to provide a broader context, by discussing the bottom-up propagation of faults/errors/failures across abstraction levels. In other words, we analyze a *device, architecture, algorithm, and application* case-study stack, respectively composed by: FPGA, Systolic Array, MxM, and CNN.

From the device (FPGA) perspective, permanent upsets in the CRAM are the main concern. Whenever configuration bits are affected, the circuit implemented on the fabric can start malfunctioning and providing erroneous outputs. This is because the FPGA's bitstream contains all of the relevant information regarding the routing between logic elements, the content of lookup tables, operating modes of DSPs, and more. Therefore, once the programmable logic's behavior is altered, faults can propagate to the architectural level. Technically, logic resources on the FPGA are prone to experiencing transient faults. However, as FPGAs do not run at frequencies nearly as fast as modern ASICs/CPUs/GPUs, the SET cross section pales in comparison to the SEU cross section of the configuration memory (the former typically being orders of magnitude higher) (Keren et al. 2019).

As, in this case, architecture (Systolic Array) and algorithm (MxM) are tightly coupled, we are going to discuss them together. Previous works (Rech et al. 2013) (Wu et al. 2016) (Gonçalves de Oliveira et al. 2016) have already outlined the following MxM error categories from both analytical and experimental standpoints:

- **Single:** only one element is corrupted.
- **Partial Line:** only one row/column is partially corrupted.
- **Full Line:** only one row/column is fully corrupted.
- **Full Matrix:** the entire matrix is corrupted.
- **Random:** none of the above.

Fig. 25 showcases pattern examples of each error category.

Using the methodology described in Section 6.3, we have injected over 618,000 faults in a small 4x4 version of our systolic array. For each injected fault, we ran ten different matrix multiplication operations, meaning that, effectively, we have performed more than 6 million tests. Our expected statistical error regarding the distribution of

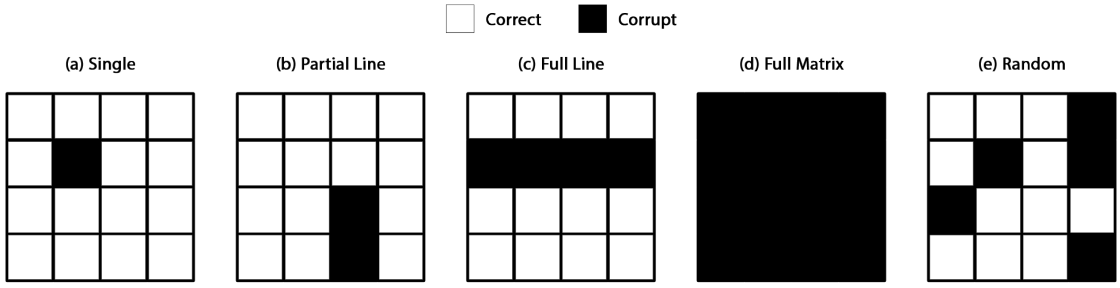

Figure 25: Pattern examples for output error categories in matrix multiplication.

Table 8: The distribution of MxM error categories, as a result of architectural faults injected in a systolic array.

Single	**Partial Line**	**Full Line**	**Full Matrix**	**Random**
27.07%	43.28%	26.68%	0.25%	2.72%

error categories is ±0.025% (95% confidence). Table 8 shows the distribution of errors within the aforementioned categories. Very clearly, we can see that *Partial Line* and *Full Line* errors account for the vast majority of cases (69.96% to be exact). As we previously discussed, this is due to the pattern of interconnection and data movement in a systolic array, where each processing element contributes to the calculation of multiple output elements. Therefore, whenever an upset alters the functioning of a PE, it has a high probability of affecting multiple outputs that share the same row/column. Namely, after the fault has been installed, every subsequent multiplication/addition in that given PE is likely to produce errors (except for possible input-dependent masking effects). In fact, for over 90% of injections that result in matrix errors, all ten output matrices present corruptions of the same error category. Since the patterns persist across the following operations in the pipeline, in an application context such as a CNN (executing MxM over and over again), stuck-at faults in a systolic array will exhibit a dangerous compounding effect.

Prior work has also observed a (slightly lower) predominance of line errors (49%), but in a GPU context (Rech et al. 2013). The authors argued that upsets in the memory hierarchy are the most probable reason why. Likewise, the authors of (Wu et al. 2016) and (Gonçalves de Oliveira et al. 2016) analytically noted that a single corruption in an input element will cause an entire line (row or column) corruption in the output matrix. As we will later explain in Section 6.5, the prevalence of line errors is one of the key motivations behind our novel hardening strategy. Finally, we shall state that, although our experiments were performed with a simple 4x4 array, the prevalence of line errors holds for any size N, since it is inherently linked to the superposition of persistent faults in PEs and systolic computation itself. In fact, we might intuitively argue that the frequency of line errors shall actually increase with N, as it only takes a single faulty PE within the array to induce them.

In order to contextualize the severity of propagating algorithm errors, we have chosen the simple and well-known LeNet CNN for handwritten digit classification (Lecun et al. 1998) as a foundational case study at the application level. We have separately injected random persistent patterns of *Single, Partial Line, Full Line, and Full Matrix* errors in the computation of MxM-based (convolutional and inner product) layers of the network. Our injections are persistent in the sense that, after selecting a first target layer, we proceed to inject the same pattern in all subsequent MxM computations. Moreover, as we mentioned in Section 6.3, we made sure to break each large matrix multiplication in appropriately-sized blocks, as a way to mimic the real execution on a (faulty) systolic array. For partial and full line patterns, line indexes were randomly generated for each injection, and, specifically in the case of partial line errors, the number of corrupted elements was also randomly chosen. We present the results in Fig. 26 measured as Program Vulnerability Factor (PVF) (Sridharan and

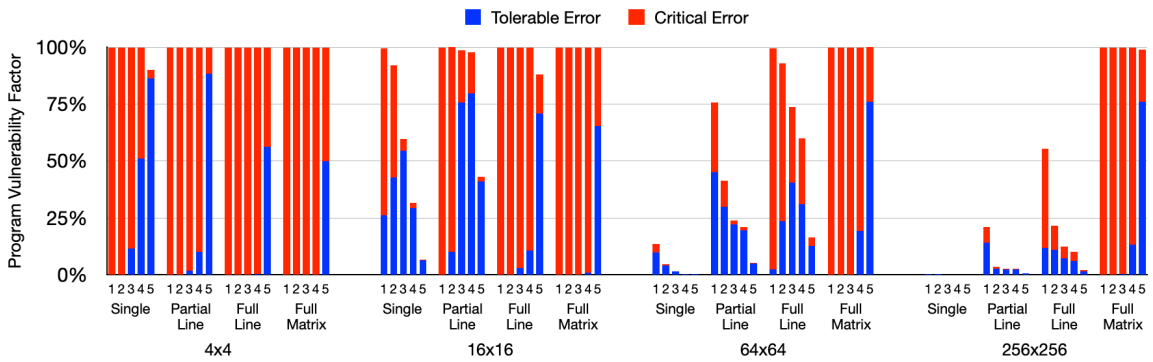

Figure 26: Program Vulnerability Factor (PVF) of the case-study LeNet CNN. Persistent matrix multiplication error patterns were injected at convolutional (1,2,3) and inner product (4,5) layers. Separate campaigns were performed for different systolic array sizes *N={4,16,64,256}*. Application errors are considered tolerable or critical according to whether or not the final output corruption was significant enough to compromise the image classification task.

Kaeli 2009) (i.e. the likelihood of an injected fault generating an observable error), subdivided in *Tolerable Errors* and *Critical Errors*, according to whether or not the final output corruption was significant enough to compromise the image classification task. For each bar plotted in Fig. 26, 1000 faults were injected, which establishes a statistical error of around ±3% in our PVF estimates.

As we can see, PVF is generally lower and less critical in larger arrays. This is because, given a large MxM operation, the number of sub-blocks required to complete the execution is inversely proportional to the available systolic array size. For instance, if we were to execute a 256x256 operation in a 4x4 array, then we would have to compute 2^{18} 4x4 blocks. Given that a persistent fault exists in the array, then most of the block computations are very likely to exhibit the same error pattern. In other words, a persistent fault in *KxK* array will cause many more corruptions in a particular matrix multiplication operation than the same persistent fault in a *LxL* array, given $K<L$. Moreover, we can clearly perceive that, independent of array size, PVF increases with the criticality of MxM errors, which is rather intuitive. For

example, a persistent fault generating partial line errors in a 64x64 array has a 75.6% probability of generating output errors, compared to only 13.7% with a stuck-at single error. This, of course, is also true in terms of criticality: a persistent fault generating full line errors in a 64x64 array has a 97% probability of generating critical errors in the CNN, compared to only 30.6% with partial line errors. Finally, we can also see from Fig. 26 that the earlier the persistent fault is injected, the higher the probability of compromising the CNN's correctness. For instance, if we inject a persistent full line error at the beginning of LeNet's computation (layer 1) using a 256x256 array, the likelihood of observing critical errors is 43.5%, whereas if we only start having faulty MxM operations from layer 2 onwards, that probability diminishes to 10.8%.

As a way of investigating how the criticality of persistent faults would change when increasing the complexity of the CNN, we have conducted additional fault injection experiments with the state-of-the-art 25-layer VGG16 topology (Simonyan and Zisserman 2015). Out of the 25 layers, 16 are MxM-based (13 of which are convolutional and 3 of which are inner product). Once again, we separately injected random persistent patterns of *Single, Partial Line, Full Line, and Full Matrix* errors in matrix multiplication operations from each of the target layers onwards. Furthermore, each operation is broken down into 256x256 blocks. The results (Fig. 27) measured as PVF, are subdivided as *Tolerable Errors* and *Critical Errors*. For each bar plotted in Fig. 27, 1000 faults were injected, establishing a statistical error of around ±3%.

Again, the criticality of CNN errors increases with the severity of persistent MxM error categories. In fact, for both *Full Line* and *Full Matrix* patterns, VGG16 always experienced misclassifications, regardless of which first target layer was chosen. In the case of *Single* and *Partial Line* patterns, it is clear that the earlier the fault is installed, the higher the probability of it manifesting as a CNN output error (and of such

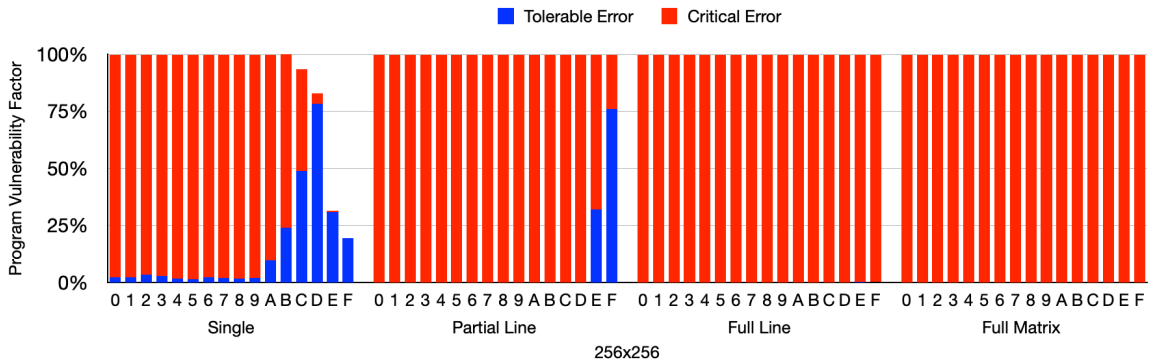

Figure 27: Program Vulnerability Factor (PVF) of the case-study VGG16 CNN. Persistent matrix multiplication error patterns were injected at convolutional (0-C) and inner product (D,E,F) layers. The campaign was performed considering a systolic array size *N=256*. Application errors are considered tolerable or critical according to whether or not the final output corruption was significant enough to compromise the image classification task.

error being considered critical), which was expected, and is now properly quantified. Moreover, we have also tested VGG16 considering an underlying 32x32 array. However, with the exception of single errors injected exclusively on the MxM computation of the last layer of the network, all other test cases led to 99+% probabilities of critical errors. As this graph would not be very insightful, we have opted to only report the 256x256 case. Conveniently, as 256 is the dimension of Google's TPU design (Jouppi et al. 2017), we believe it to be the single most important scenario to evaluate.

Finally, we must highlight the fact that, as real-time CNNs work at high frame rates, even if a persistent fault is first established at the last layer of frame f_i, it will very likely affect the computation of the first layer of frame f_{i+1}, and all the following frames. This scenario then motivates the design of a fast error detection and reconfiguration scheme.

6.5 Light ABFT

Motivated by the need for low-cost hardening strategies, and the aforementioned predominance of line errors, we present ***Light ABFT***: a lightweight error detection technique for matrix multiplication, specially tailored for high-performance systolic arrays on FPGAs. Next, we shall discuss the idea itself, and its costs of implementation, from algorithmic (arithmetic operations), and architectural (arithmetic units) perspectives. We will also compare our costs to the original and most traditional ABFT (Kuang-Hua Huang and Abraham 1984), and to the most recent work on convolution-specific error detection (Hari et al. 2021).

6.5.1 Overview & Formal Demonstration

The original ABFT (Kuang-Hua Huang and Abraham 1984) works because of redundancy of information. The calculation and attachment of summation vectors to the input matrices, prior to the execution of MxM, makes it possible to detect, and potentially correct, arithmetic errors at the end of computation. As it can be seen in Fig. 28, input matrices A and B grow by one row/column respectively.

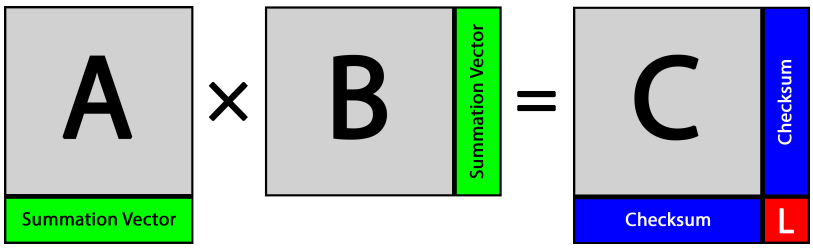

Figure 28: Visual comparison between ABFT and Light ABFT in terms of information redundancy and cost.

Light ABFT only calculates L, instead of the entire checksum row/column.

Given an input matrix A with i rows and j columns, then the j elements in A's summation vector (row *i+1*) are:

$$SV_{A_z} := A_{i+1,z} = \sum_{x=1}^{i} A_{x,z} \quad \text{for } 1 \leq z \leq j \tag{6.1}$$

Given an input matrix B with j rows and k columns, then the j elements in B's summation vector (column *k+1*) are:

$$SV_{B_z} := B_{z,k+1} = \sum_{y=1}^{k} B_{z,y} \quad \text{for } 1 \leq z \leq j \tag{6.2}$$

Carrying out the calculation with the original ABFT algorithm would then result in an output matrix C with *i+1* rows an and *k+1* columns. Effectively, this means that an extra *i+k+1* elements would have to be computed, on top of the useful workload. As proved by (Kuang-Hua Huang and Abraham 1984), the checksum information can be used to detect errors in C, and, in the case of single errors, to reconstruct corrupted data. However, as per Section 6.4, when considering systolic arrays in FPGAs, line errors are much more frequent, meaning that the benefit of having error correction capabilities would vanish, most of the time.

Therefore, the fundamental insight behind *Light ABFT* is that, by focusing solely on error detection, it is possible to eliminate a significant amount of unnecessary computation. More specifically, instead of calculating *i+k+1* extra output elements, we only calculate one, which is showcased in red with the letter L in Fig. 28. Directly from Fig. 28, we can see that ***L* is the dot product of summation vectors A and B**. At the same time, it is easy to prove that ***L* also equals to the sum of all elements in *C***.

Theorem 6.5.1.

$$L = SV_A \cdot SV_B = \sum_{x=1}^{i} \sum_{y=1}^{k} C_{x,y} \tag{6.3}$$

Proof.

The dot product between the summation vectors A and B can be rewritten as:

$$SV_A \cdot SV_B = \sum_{z=1}^{j} SV_{A_z} SV_{B_z} \tag{6.4}$$

Substituting Eq. 6.1 and Eq. 6.2 in Eq. 6.4 we get:

$$SV_A \cdot SV_B = \sum_{z=1}^{j} (\sum_{x=1}^{i} A_{x,z} \sum_{y=1}^{k} B_{z,y}) \tag{6.5}$$

Using distributive properties, we can rewrite Eq. 6.5 as:

$$SV_A \cdot SV_B = \sum_{x=1}^{i} \sum_{y=1}^{k} \sum_{z=1}^{j} A_{x,z} B_{z,y} \tag{6.6}$$

By definition, each element $C_{x,y}$ of output matrix C is calculated as the dot product between the x^{th} row of input matrix A (A_x) and the y^{th} column of input matrix B (B_y):

$$C_{x,y} = A_x \cdot B_y = \sum_{z=1}^{j} A_{x,z} B_{z,y} \tag{6.7}$$

Therefore, it is also true that:

$$\sum_{x=1}^{i} \sum_{y=1}^{k} C_{x,y} = \sum_{x=1}^{i} \sum_{y=1}^{k} \sum_{z=1}^{j} A_{x,z} B_{z,y} \tag{6.8}$$

Since the right-hand side of Eq. 6.6 is equal to the right-hand side of Eq. 6.8, by transitivity we have proved that Eq. 6.3 is true.

□

As a consequence of having two distinct ways for calculating L (via the inputs, and via the outputs), a simple comparison between the two is sufficient to verify the correctness of the matrix multiplication.

6.5.2 Cost Analysis

To facilitate the discussion about costs, we shall first enumerate the steps involved in the traditional ABFT (Kuang-Hua Huang and Abraham 1984):

1. Calculation of summation vectors
2. Matrix multiplication (useful work)
3. Calculation of checksums
4. Error detection
5. Error correction

For each step (with the exception of error correction), we will express cost in algorithmic (raw workload) and architectural (systolic array) terms. Once again, we will consider that input matrix A has i rows and j columns, and that input matrix B has j rows and k columns. As a direct consequence of matrix multiplication, output matrix C has i rows and k columns.

In order to calculate the summation vectors, we must perform j sums of i elements for matrix A and j sums of k elements for matrix B, totaling a cost of $j(i+k)$ sums. Architecturally, we need $2j$ extra adders to accumulate the summation vectors as data enters the systolic array, while maintaining a fully-streaming behavior.

Given the aforementioned matrix sizes, the intrinsic cost of the matrix multiplication is to perform j MACs for each of the ik output elements, totaling $j(ik)$ MACs. Since the original ABFT grows matrix C to $i+1$ rows and $k+1$ columns, the

cost becomes the calculation of j MACs for $(i+1)(k+1)$ output elements, totaling $j(ik+i+k+1)$. Therefore, ABFT's added cost is $j(i+k+1)$. Architecturally, the authors of (Kuang-Hua Huang and Abraham 1984) suggest growing the systolic array accordingly, adding a total of $i+k+1$ extra PEs. In the case of *Light ABFT*, the sizes of the matrices remain unaltered, and we only calculate L as the dot product between the summation vectors, incurring a cost of j MAC operations, which can be accommodated with a single extra PE. Interestingly, as (Hari et al. 2021) pointed out, the increase in size for the input matrices is actually one of the main reasons why traditional ABFT incurs such a high runtime overhead in GPUs: As a larger GEMM operation must be executed, a number of inefficiencies in cache arise due to additional online data management procedures, directly translating to considerably increased execution times.

In the error detection phase, the original ABFT algorithm performs i accumulations of k elements horizontally (row-wise), and k accumulations of i elements vertically (column-wise), along with $i+k$ comparisons against checksum values, totaling $2ik$ additions and $i+k$ comparisons. The authors do not explicitly suggest an architectural implementation cost, but $i+k$ adders and 2 comparators would be needed, at a minimum, to maintain a fully utilized systolic pipeline. In the case of *Light ABFT*, it is necessary to add all the ik output elements of C, as they come out of the array, and then compare it to L. This can be architecturally accomplished with $k+1$ adders and a single comparator. It must be mentioned, however, that the original ABFT is able to pin-point the x and y indexes of the error (which is needed for the following error correction step), while *Light ABFT* only signals that something went wrong (which can then trigger a system-level action, such as moving to a fail-safe state until corrective measures like partial reconfiguration can take place).

We summarize the algorithmic cost (number of extra arithmetic operations) and architectural cost (number of extra arithmetic units) of *Light ABFT* in Table 9, while comparing it to (Kuang-Hua Huang and Abraham 1984). As a way to simplify the data, we assume that a comparison/comparator costs the same as an addition/adder.

Table 9: Costs of ABFT compared to Light ABFT, in algorithmic (raw workload) and architectural (systolic array) terms.

Step	**Cost**	**Type**	**ABFT**	**Light ABFT**
1)	Algorithmic	ADD	$j(i+k)$	$j(i+k)$
	Architectural	ADD	$2j$	$2j$
3)	Algorithmic	MUL	$j(i+k+1)$	j
		ADD	$j(i+k+1)$	j
	Architectural	MUL	$i+k+1$	1
		ADD	$i+k+1$	1
4)	Algorithmic	ADD	$2ik+i+k$	$ik+1$
	Architectural	ADD	$i+k+2$	$k+2$
Total	Algorithmic	ADD	$2(ik+ji+jk)+i+j+k$	$ik+ji+jk+j+1$
		MUL	$ji+jk+j$	j
	Architectural	ADD	$2(i+j+k)+3$	$2j+k+3$
		MUL	$i+k+1$	1

For practical purposes, if we assume square matrices of size N (i.e. $i{=}j{=}k{=}N$), we are able to plot the total algorithmic and architectural expressions in Fig. 29 and Fig. 30, respectively. Note the staggering difference of cost in MUL.

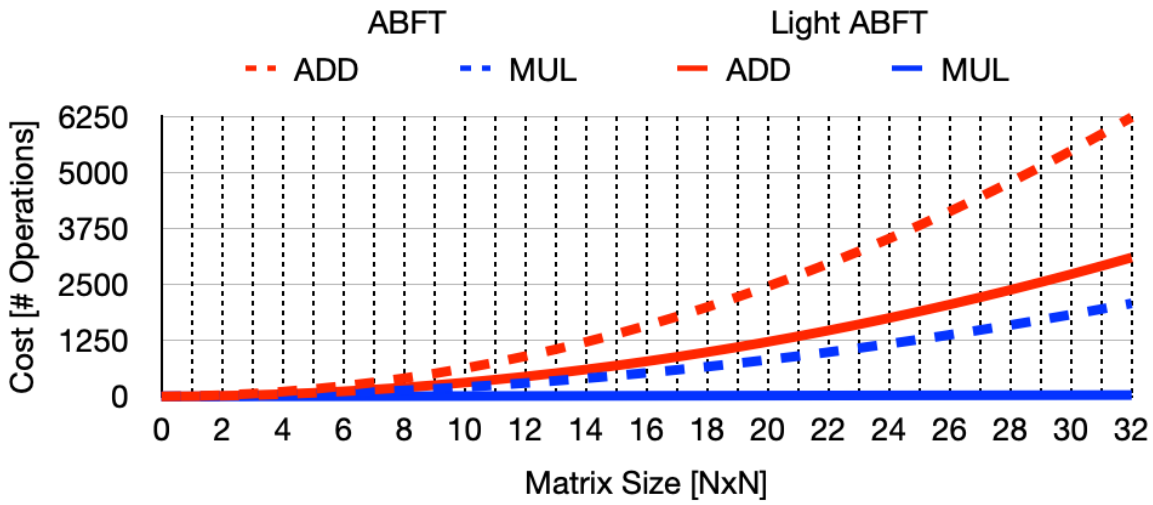

Figure 29: Algorithmic cost of ABFT compared to Light ABFT.

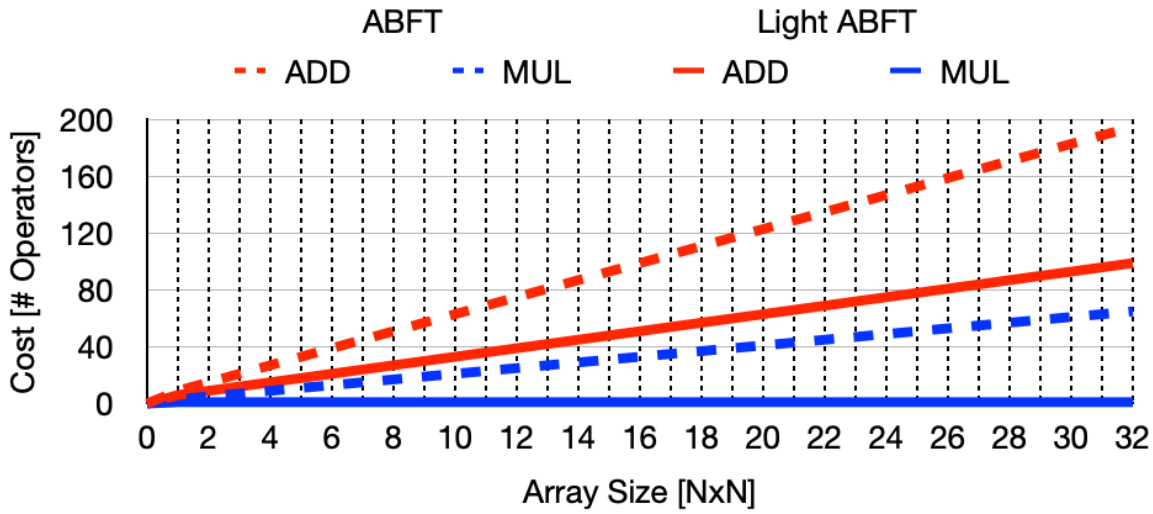

Figure 30: Architectural cost of ABFT compared to Light ABFT.

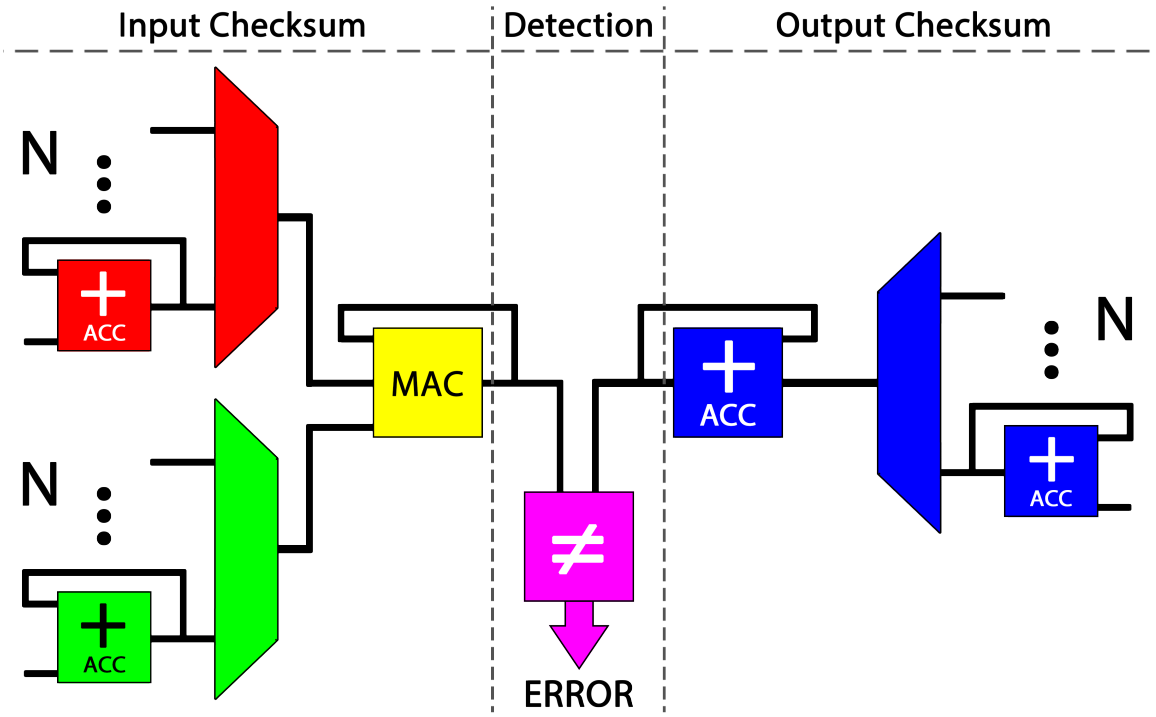

Figure 31: Our suggested Light ABFT architecture for a systolic array context. The array is assumed to be squared and of size N. Input matrices enter the unit from the left, while the output matrix enters the unit from the right. The unit then outputs a signal indicating if errors are present in the MxM computation.

Even though ASICs are not the main focus of our work, we believe that is also worthwhile to estimate area and energy costs for (Kuang-Hua Huang and Abraham 1984) and our suggested *Light ABFT* architecture (Fig. 31) in a given technology node, as a way of delivering a more accurate perspective on the cost difference between the two. More specifically, we have used a leading foundry 45nm CMOS Process Design Kit (PDK), which cannot be disclosed, to extract post-place-and-route area and energy numbers for 32-bit adders and multipliers. Then, we use those numbers to merge the *ADD* and *MUL* architectural cost expressions in Table 9. We plot normalized area and energy costs for systolic array sizes between 32 and 256 in Fig. 32. Very evidently, the cost gap between (Kuang-Hua Huang and Abraham 1984) and

Light ABFT grows larger and larger as N increases. This is mainly because, as Table 9 shows, *Light ABFT* only requires one multiplier for its implementation, whereas (Kuang-Hua Huang and Abraham 1984) requires *2N+1*. As a way of relating this result to the context of FPGAs, we shall also state that a low requirement for multipliers is paramount for any efficient hardening technique, in the sense that high-speed DSP units are the most scarce type of resource in the programmable fabric of modern devices.

Furthermore, we can also calculate and report the area and energy overheads of (Kuang-Hua Huang and Abraham 1984) and *Light ABFT* as the relationship between the costs curves in Fig. 32 and the intrinsic cost of the *NxN* systolic array itself. We plot such curves in Fig. 33. The overhead of both techniques diminishes with N. This is because all cost functions plotted in Fig. 32 have polynomial degrees equal to one. Meanwhile, the intrinsic architectural cost expression for an *NxN* systolic array has a polynomial degree equal to two. Therefore, the resulting overhead expressions have polynomial degrees equal to minus one. Nonetheless, even for a 256x256 systolic array, the area and energy overheads incurred by *Light ABFT* are still estimated to be 1/18th and 1/30th of those incurred by (Kuang-Hua Huang and Abraham 1984), respectively. In practice, such low overhead figures also mean that the likelihood of a real-world hardware accelerator implementation experiencing instances of false detections massively diminishes for larger systolic arrays.

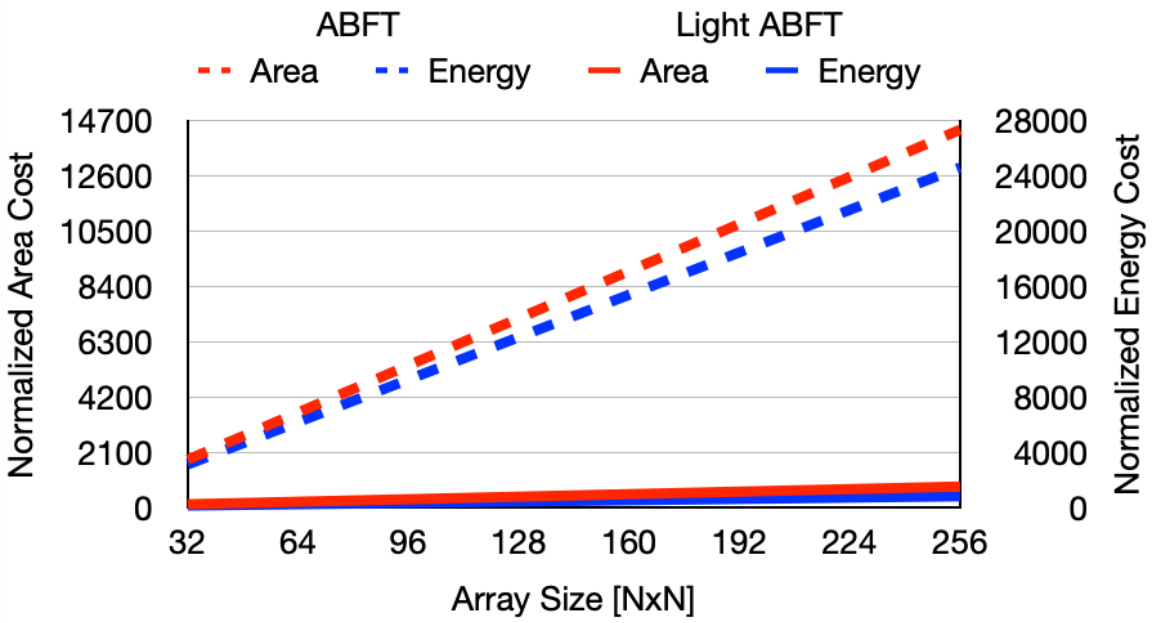

Figure 32: Normalized area and energy costs of ABFT compared to Light ABFT in a 45nm CMOS technology node.

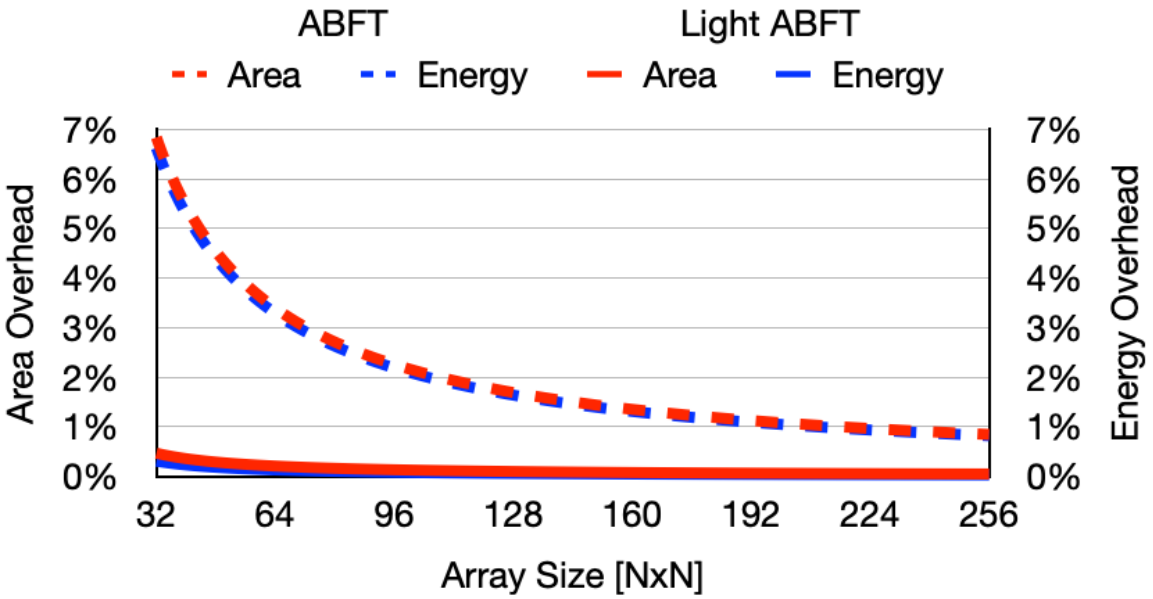

Figure 33: Normalized area and energy overheads of ABFT compared to Light ABFT in a 45nm CMOS technology node.

We conclude with a cost evaluation in a concrete real-world application scenario. Since we have previously mentioned CNNs as being one of the most prominent MxM-based applications, and (Hari et al. 2021) being the most recent and efficient piece of related work for CNN-specific ABFT-based error detection, an in-context comparison becomes very relevant. More specifically, the authors of (Hari et al. 2021) have chosen three well-known CNNs (VGG16 (Simonyan and Zisserman 2015), ResNet18, and ResNet50 (He et al. 2016)), and estimated the overheads incurred by two of their proposed convolution error detection mechanisms. They have included numbers for *average increase in number of operations*, as well as *runtime overhead in a GPU execution*. As the latter is not directly comparable to our work, we will stick to the former. Authors reported an average increase of arithmetic operations of $<$7% and $<$1% for their *FC* and *FIC* techniques, respectively.

In order to employ *Light ABFT* in a convolution operation, it suffices to translate it to an equivalent matrix multiplication, using the well-known *im2col* method (Chellapilla, Puri, and Simard 2006). Then we calculate the cost of adding *Light ABFT* to each of such MxM operations (using the expressions in Table 9), and the cost of the (unhardened) MxM operation itself. The ratio between the two aforementioned costs gives us the overhead of having *Light ABFT* as a percentage of useful workload. These calculated overheads are 0.35%, 0.81%, and 0.71% for VGG16, ResNet18 and ResNet50, respectively. Therefore, the average overhead of adding *Light ABFT* to the convolutional layers of the case-study CNNs comes out to 0.63%, which is at least as good as the state-of-the-art convolution-specific technique presented by (Hari et al. 2021). Moreover, *Light ABFT* can also be directly employed in the inner product (i.e. fully connected) layers of neural networks, as these are, by definition, matrix multiplication operations between inputs and weights.

6.6 Experimental Validation

Since we have already formally demonstrated that *Light ABFT* works for arbitrarily-sized matrix multiplication operations in Section 6.5, the goal of this section is simply to experimentally evaluate *Light ABFT* in a concrete FPGA scenario. We integrate the *Light ABFT* module (proposed in Section 6.5) into the hierarchy of our systolic array implementation (detailed in Section 6.2). As the unhardened DUT was already tested in Section 6.4, we simply repeat the experiments to measure the newly added error detection capabilities.

After injecting over 623,000 faults in our design, we have measured a detection rate of 97.4%, with a statistical error of ±0.15% (95% confidence). We believe that the remaining 2.6% of undetected faults are due to errors (1) in the interface of the module, (2) in control signals throughout the unit, or (3) in the *Light ABFT* hardware itself. Prior studies have investigated hardening alternatives for (1) and (2) (Kastensmidt et al. 2005) (Niranjan and Frenzel 1996), but they are out of scope for this work. Since the intention of *Light ABFT* is to protect MxM computation, our experimental results are very much indicative of success. Moreover, prior work has also reported experimental FPGA fault injection numbers that do not quite reach the theoretical 100% detection rate, even for coarse-grain DMR implementations. For instance, (Aranda, Reviriego, and Maestro 2018) achieves a maximum of 96.1% detection when duplicating an FIR filter design. Similarly, (González-Toral et al. 2018) proposes and experimentally evaluates a handful of ABFT-based error detection techniques for Fast Fourier Transform (FFT), reporting detection rates between 94.12% and 99.77%. This means that, in practice, there is an asymptotic limit for achievable error detection rates with redundant hardware.

Furthermore, the authors of (González-Toral et al. 2018) also report the occurrence of another known event in redundant circuits: *false detections*. As we mentioned in earlier chapters, the added redundancy, in virtually all error detection techniques, is also prone to experiencing upsets, in which case untrue detections may arise. In our experiments with a 4x4 array, we have measured a 2.39% probability for such events. For comparison purposes, (González-Toral et al. 2018) reports false detection rates as high as 45%. Luckily, in our case, since the percentual area overhead of *Light ABFT* diminishes with increasing array sizes (as per Section 6.5), so does the ratio between true and false detections in real radiation-rich environments. In other words, for sufficiently large arrays, the rate of false detections is vanishing. Nevertheless, for extremely strict scenarios, it would suffice to instantiate a *DMR Light ABFT* module, as a way of eliminating false detections, while still paying much less than with full DMR.

6.7 Discussion

As mentioned in Section 6.4, large matrix multiplication operations can be broken down into smaller *blocks*. In other words, large matrices can be partitioned, and execution can be perceived in steps. This characteristic becomes particularly useful in GPUs when it comes to fitting just the right amount of data into each cache level, as a way of minimizing latency in memory operations. Likewise, when thinking of systolic accelerators for ASICs/FPGAs, the size of the array is set at implementation time, which means that, in order to execute large MxM operations, computation must also be completed in steps. Precisely, if we intend to run an operation with square matrices of size N, and we choose to break such operation into square blocks of size

M, with $N \geq M$, and both N and M being positive powers of 2, then the number of M-sized matrix multiplications to be executed is $(N/M)^3$. This is somewhat intuitive if we recall that the total number of MAC operations involved in matrix multiplication must be maintained regardless of blocking:

$$N^3 = \left(\frac{N}{M}\right)^3 \cdot M^3 \tag{6.9}$$

However, when applying error detection techniques to blocks as opposed to the entire matrix, the algorithmic cost expressions derived in Section 6.5 no longer tell the whole story. In general, for both Kuang-Hua Huang and Abraham 1984 and *Light ABFT*, we have that the following relationship holds true for the hardening cost H:

$$H(N) \leq \left(\frac{N}{M}\right)^3 \cdot H(M) \tag{6.10}$$

At the same time though, when utilizing blocks we also increase the *granularity of checking*, which means that (1) errors can be detected in the middle of computation, as opposed to just at the end, and (2) the cost of recomputation diminishes, as we do not need to re-run the entire operation after an error detection.

To get a real sense of precisely when the benefits of blocking outweigh its costs, yet another cost expression F is in order. In addition to matrix size (N), and block size (M), our new function F also depends on the expected average number of executions between errors (R). The basic idea is that, if we expect to have, on average, an error after every R executions, then the cost of block recomputation is diluted throughout those R executions. Hence, F represents the *failure-free* execution cost. The first part of F (expressed as F_1) is the cost of R executions:

$$F_1(N, M, R) = R \cdot \left(\left(\frac{N}{M}\right)^3 \cdot \left(M^3 + H(M)\right)\right) \tag{6.11}$$

The second part of F (expressed as F_2) is the cost of recomputing a single block:

$$F_2(N, M, R) = 1 \cdot \left(M^3 + H(M)\right) \tag{6.12}$$

Therefore, it follows that the actual *failure-free* execution cost F is expressed as:

$$F(N, M, R) = \frac{F_1 + F_2}{R} \tag{6.13}$$

$$F(N, M, R) = \left(\left(\frac{N}{M}\right)^3 + \frac{1}{R}\right) \cdot \left(M^3 + H(M)\right) \tag{6.14}$$

Notice that, if R tends to infinity (i.e. no errors), then the term *1/R* vanishes, symbolizing a vanishing recomputation cost.

Fig. 34 plots the cost F divided by the inherent cost of matrix multiplication (N^3), minus 100%, with *N=256* and increasing values of R. Fig. 35 plots the same curves, for a very large matrix size (*N=4096*). As we can see on both graphs, for each R curve there is a different choice of M that delivers the minimum arithmetic overhead. Moreover, the higher the expected error rate, the sooner that inflection point appears. Given specific $N=N_0$ and $R=R_0$, it is possible to directly find the optimal M by solving:

$$\frac{\partial F(N_0, M, R_0)}{\partial M} = 0 \tag{6.15}$$

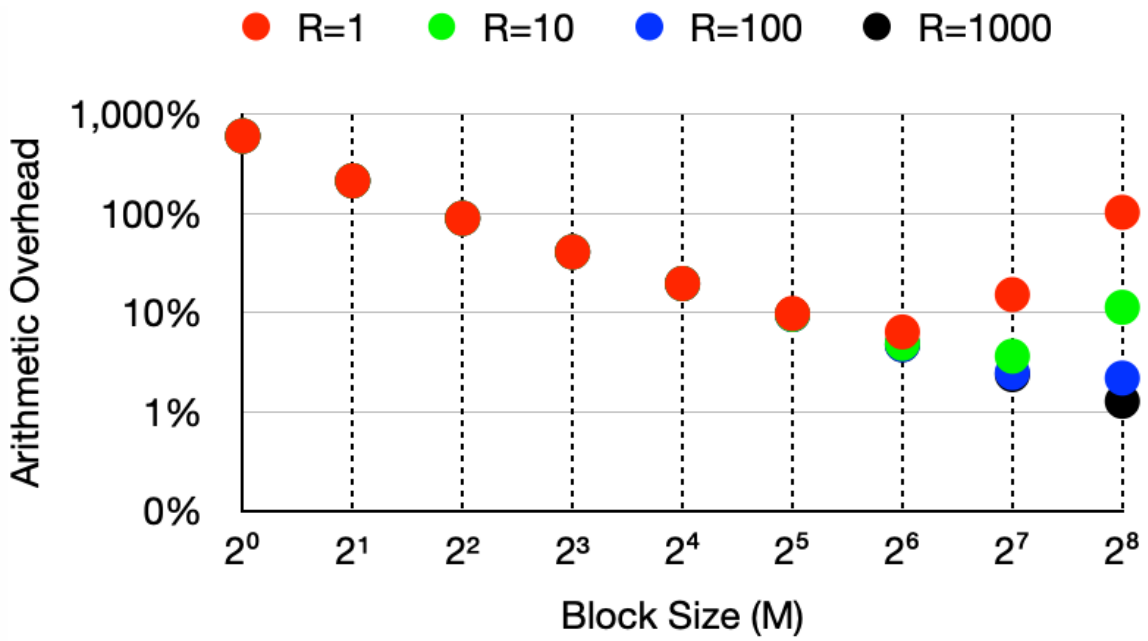

Figure 34: Arithmetic overhead of Light ABFT, for matrix size *N=256*, growing block size (M) and expected average executions between errors (R).

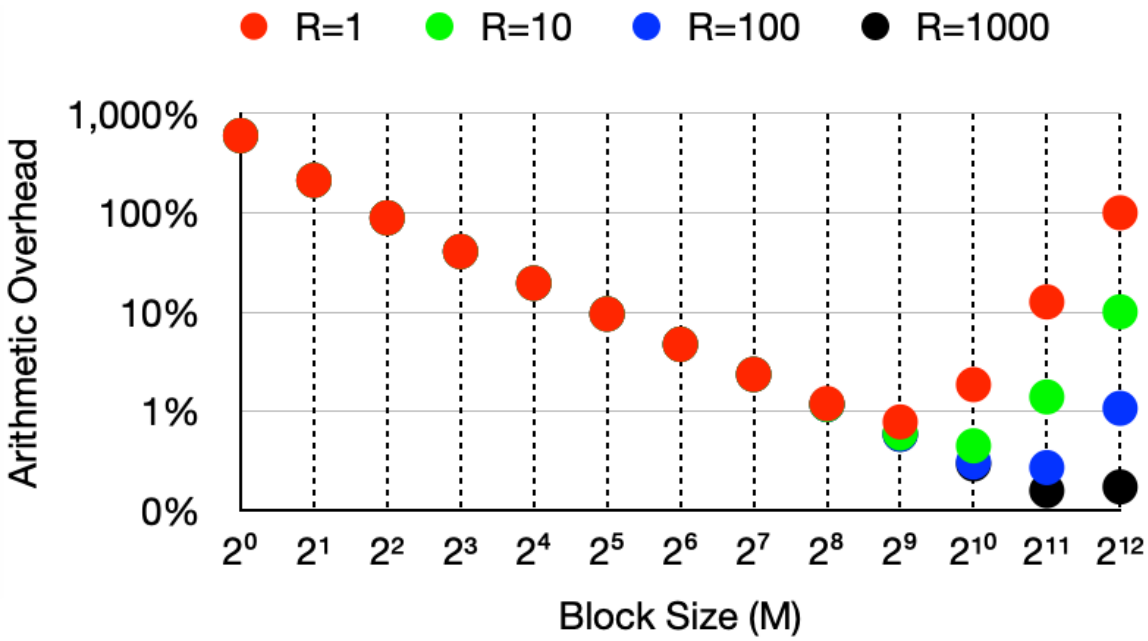

Figure 35: Arithmetic overhead of Light ABFT, for matrix size *N=4096*, growing block size (M) and expected average executions between errors (R).

Chapter 7

HOW ARCHITECTURAL DECISIONS AND ACCURACY LEVELS IMPACT THE RELIABILITY OF HYPOTHETICAL NEURAL NETWORK ACCELERATORS ON FPGAS

The contents of this chapter are, for the most part, based on a pair of papers titled *"How Reduced Data Precision and Degree of Parallelism Impact the Reliability of Convolutional Neural Networks on FPGAs"* and *"How Inherently Inaccurate Applications Behave In Radiation Environments"*, both published at IEEE Transactions on Nuclear Science (Libano et al. 2021) (Libano et al. 2020). The majority of the text is reproduced verbatim, only with structural modifications.

7.1 Motivation

Thus far, we have experimentally measured the neutron sensitivity of Xilinx's DPU (Chapter 5), and proposed a low-cost hardening technique for matrix multiplication within the context of systolic arrays (Chapter **??**). It is also worthwhile to evaluate the impacts of key architectural choices on the reliability of hypothetical neural network accelerators.

Particularly, we are interested in analyzing precision of data representation (i.e. complexity of processing elements) and degree of parallelism (i.e. quantity of processing elements), which are two of the most common design decisions architects make when constructing hardware accelerators.

Moreover, neural networks fall into a very specific class of applications. Since ML

models go through imperfect learning processes, their success is typically evaluated through the accuracy measured at the testing phase. Thus trained NNs are prone to fail even without being disturbed, at a rate proportional to their inherent inaccuracy. Once exposed to other sources of faults (i.e. radiation), global failure rates rise.

Particularly, we are interested in modeling the failure rate of approximated algorithms, considering both of the aforementioned sources of faults.

7.2 Background

7.2.1 Quantization of Neural Networks

In order to achieve high levels of accuracy, CNNs end up being computationally expensive. As a way of minimizing cost, a number of simplification techniques have been developed for neural networks, such as weight trimming and quantization (Hubara et al. 2017). Here, we focus on the latter. Quantization reduces the precision in which the weights of a given model are represented to speed up computation and reduce resource utilization. At training time, all industry-leading frameworks use 32-bit floating-point as a default. For the specific case of TensorFlow, they provide a subset of tools called TensorFlow Lite (TensorFlow 2020), which allows developers to quantize their models to lower precisions, such as 16-bit floating-point (IEEE's half-precision), and 8-bit integer. The clever thing about TensorFlow's tools is that their quantization process uses a small subset of the training data for calibration purposes. The end result is that the quantized models show little to no accuracy loss when compared to their 32-bit float counterparts.

More extreme precision reduction can go as far as utilizing a single bit to represent

the weights in a model. These are called *Binary Neural Networks (BNNs)*, in which both the filters in the convolutional layers, as well as the neurons in the inner product layers use weights constrained to *{-1, 1}*. The adoption of binary weights essentially makes multiplications trivial, which decreases resource utilization in hardware. However, differently than 8-bit integer quantization, binary quantization usually leads to some non-negligible decrease of accuracy.

7.2.2 MNIST Dataset & CNN Topology

We chose the well-known Modified National Institute of Standards and Technology (MNIST) as our case study dataset, since its simplicity translates to tractability for implementing and testing several design variations. We acknowledge that this aspect of our work limits the extent to which our experimental data can be fully generalized, as more complex CNNs could have different behaviors. The dataset itself is composed of 60,000 28x28 pixel images of handwritten decimal digits (from 0 to 9) (Lecun et al. 1998). As such, our neural network models receive 28x28 matrices as inputs, and produce 10 outputs (one for each decimal digit), where the index of the highest value corresponds to the classification.

7.3 Experimental Methodology

7.3.1 Designs Under Test

In order to evaluate the trade-offs associated with data precision reduction, we implemented three versions of the MNIST CNN (*FP32*, *FP16*, *INT8*) using the

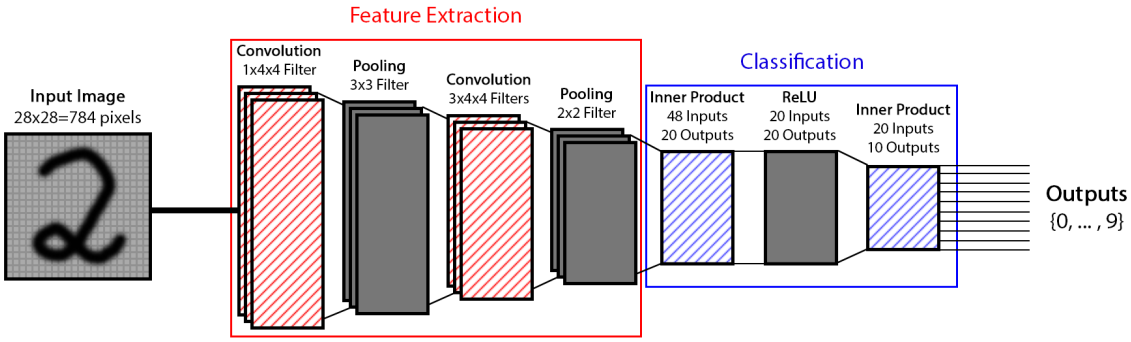

Figure 36: Topology of the MNIST CNN.
Striped layers have weights associated to them (which go through quantization).

28nm Zynq-7000 (XC7Z020) (Xilinx 2020g). For this comparison, we have utilized a very naive architecture (Fig. 37), with a single processing element per layer in the topology. Naturally, given a layer workload of N operations, N iterations (cycles) are necessary. Memory elements are interleaved between arithmetic units and used as scratchpads to hold and propagate intermediate results. Although not representative of state-of-the-art neural network accelerators, this style of architecture is similar to that of Xilinx's Fast Fourier Transform (FFT) core (Xilinx 2021). Table 10 and Fig. 38 detail resource utilization and execution times. The accuracy on all of the design variations has remained the same (95%), regardless of the quantization process performed with TensorFlow Lite. Although a higher accuracy level could have been achieved, we opted for a minimalist CNN topology, as detailed in Fig. 36.

Table 10: Zynq-7000 resource utilization to implement the MNIST CNN using 32-bit floating point, 16-bit floating point, and 8-bit integer.

MNIST Design	LUTs (Logic)	LUTs (Mem)	FFs	DSPs
32-bit Float (FP32)	6.5k	1016	336	8
16-bit Float (FP16)	3.4k	510	240	4
8-bit Integer (INT8)	1.5k	280	224	0

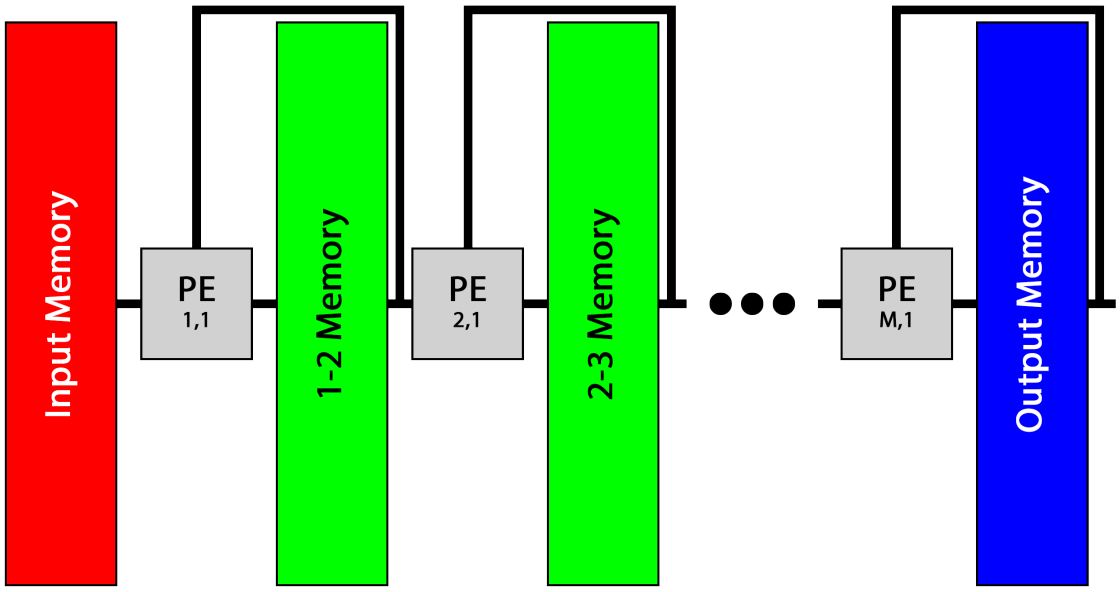

Figure 37: Iterative-based architecture for the MNIST CNN with one processing element per layer, and M layers.

For Convolution and Inner Product layers, PEs are MAC units. For Pooling and ReLU layers, PEs are comparators.

In order to evaluate the trade-offs associated with degree of parallelism, we implemented two versions of the MNIST CNN (*Min PEs*, *Max PEs*) using the 16nm Zynq UltraScale+ (XCZU9EG) (Xilinx 2020f). The first design (Min PEs) has *one* processing element per layer of the network, and represents the lower end of the parallelism spectrum. Its iterative architecture is the one shown in Fig 37. Likewise, the second design (Max PEs) has *all* of the necessary processing elements in each layer of the network, and represents the top end of the parallelism spectrum. Its streaming architecture is shown in Fig 39. In this case, FFs are utilized in between layers, as opposed to memories. Given a total of N operations involved in a layer's computation, the single PE version is going to take N iterations to finish, while the fully parallel implementation (in this case with N PEs), will take only one clock cycle. The aforementioned simplicity/tractability of the MNIST CNN, along with the use of

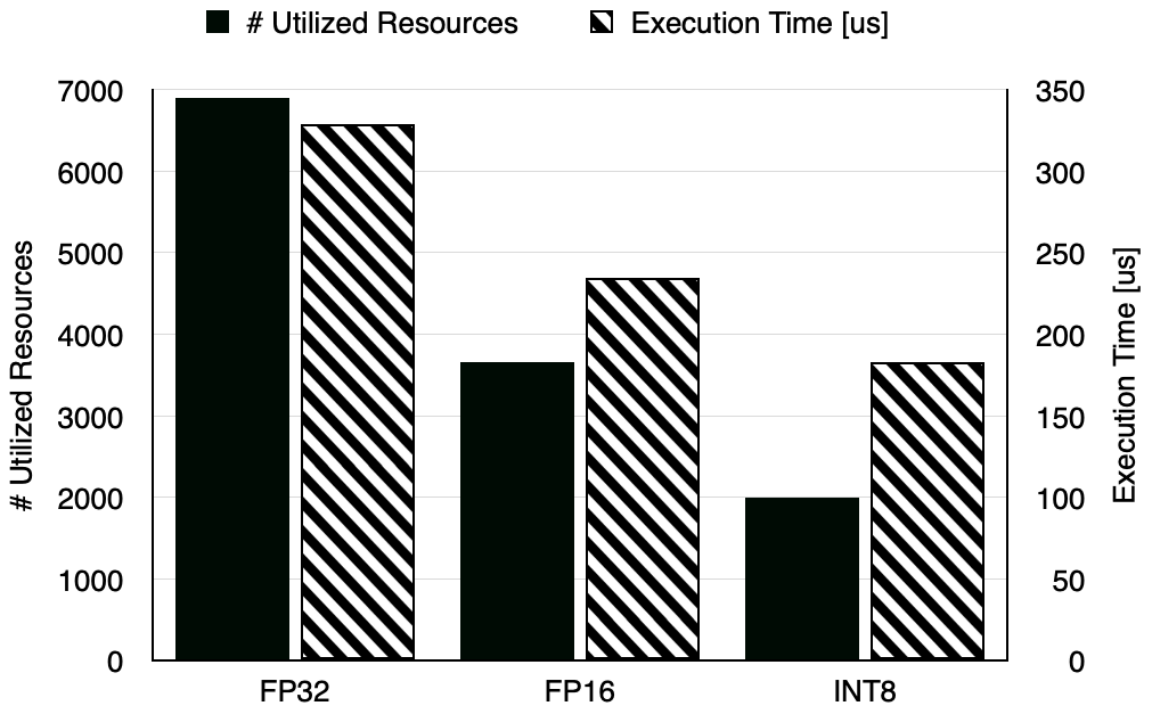

Figure 38: Total resource utilization and execution times for three levels of data precision on the MNIST CNN, implemented on the Zynq-7000.

a high-end FPGA (as the one found on the UltraScale+), allow it to be implemented in a fully parallel fashion. Table 11 and Fig. 40 provide details on resource utilization and execution times. Note that neither of the designs utilize DSPs. This is because both use 8-bit precision, which is too small for DSP inference at synthesis time. Also, note that the axes in Fig. 40 are in logarithmic scale.

Table 11: Zynq UltraScale+ resource utilization to implement the MNIST CNN with vastly different degrees of parallelism.

MNIST Design	LUTs (Logic)	LUTs (Mem)	FFs	DSPs
Min PEs	1.4k	280	240	0
Max PEs	212k	0	11.2k	0

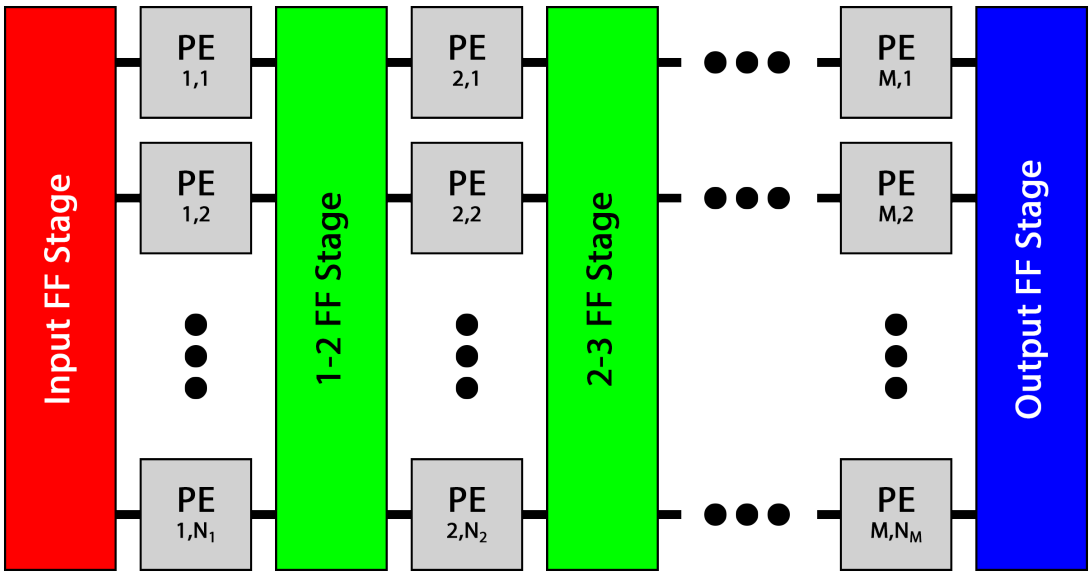

Figure 39: Streaming-based architecture for the MNIST CNN with Nx processing elements per layer, and M layers.

For Convolution and Inner Product layers, PEs are MAC units. For Pooling and ReLU layers, PEs are comparators.

In order to evaluate the impact of accuracy loss, we experimented with binary quantization. However, a fully binary version of our simplistic topology led to an unrealistic accuracy drop. Thus, instead, we only applied binary quantization to convolutional layers (responsible for the feature extraction process). The inner product layers (responsible for the classification process) were implemented using 8-bit integers. As this is only a partially binarized network, we called it a Hybrid Neural Network (HNN). For comparison purposes, we also implemented a baseline version using 8-bit integers across all layers. The architecture is the one shown in Fig. 39. Resource utilization details are show in Table 12. Accuracy levels are 93% and 88% for the Baseline CNN and the Quantized HNN, respectively.

Training phases and quantization processes for all of our DUTs were completed

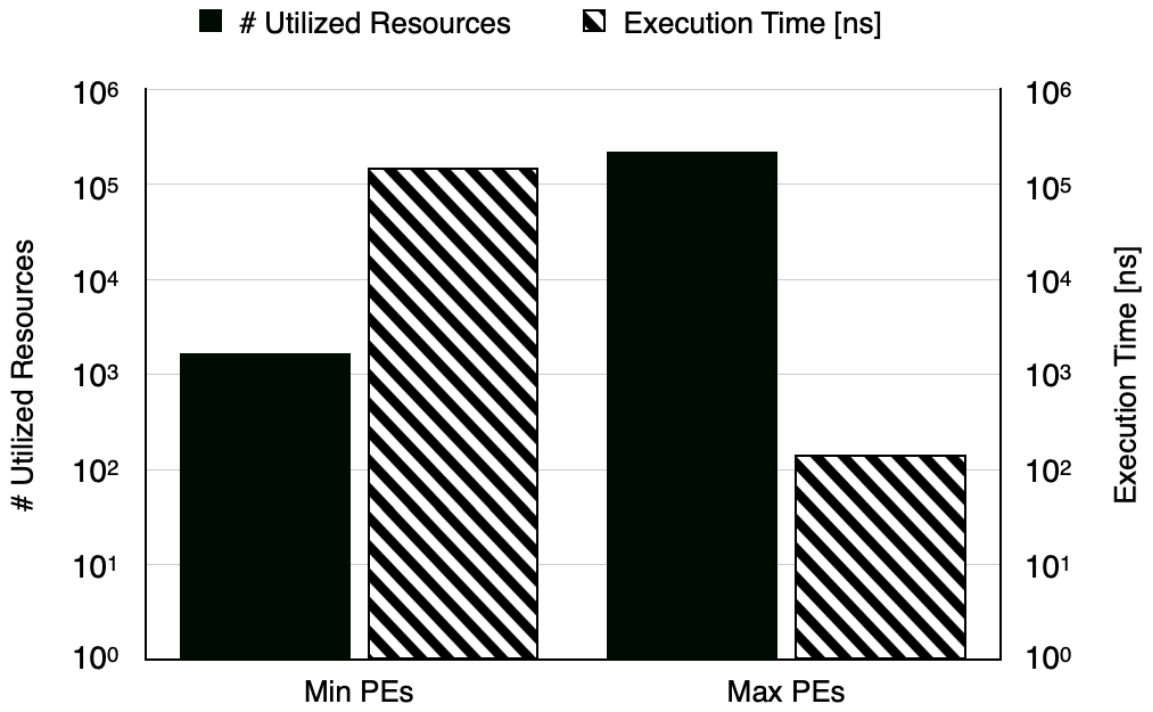

Figure 40: Total resource utilization and execution times for two degrees of parallelism on the MNIST CNN, implemented on the Zynq UltraScale+.

Table 12: Zynq Ultrascale+ resource utilization to implement the Baseline MNIST CNN and the Quantized MNIST HNN.

MNIST Design	CNN Portion	LUTs	FFs
Baseline CNN	Feature Extraction	165k	4.2k
	Classification	48k	0.4k
Quantized HNN	Feature Extraction	98k	4.2k
	Classification	48k	0.4k

ahead of the experiments (in a fault-free environment). In other words, the neural networks' set of weights was not affected by radiation-induced upsets in any capacity. Furthermore, our input stimuli is only a subset of 100 images from the MNIST dataset.

7.3.2 Radiation Experiment

Our radiation beam experiments were performed with the FP32, FP16, INT8, Min PEs, and Max PEs DUTs, at the Los Alamos Neutron Science Center (LANSCE) facility of the Los Alamos National Laboratory (LANL). While LANSCE's neutron spectrum mimics the atmospheric one, the particle flux is about 8 orders of magnitude higher than the average terrestrial flux ($(13n/(cm^2 \times h))$ at sea level (JEDEC 2006)). We ran our experiments with the respective Zynq devices for around 64 hours, accumulating a total fluence in our DUTs of $344 \times 10^9 n/cm^2$, roughly equivalent to 3 million years of natural exposure.

7.3.3 Fault Injection Experiment

Our fault injection campaigns were performed with the Baseline CNN and the Quantized HNN DUTs, using the PCAP on the Zynq Ultrascale+. We isolated the feature extraction and classification portions of the networks into separate regions of the FPGA fabric, using *Pblocks*. This way it is possible to correlate the effect of an injected fault to a specific portion of the network. Our campaigns randomly injected a total of about 11 million faults in our designs (out of around 39 million possible in an exhaustive method). Random fault-injection methods are well known and representative (Benevenuti and Kastensmidt 2019).

7.4 Results

7.4.1 Reduced Data Precision

Using the experimental methodology described in Section 7.3, we tested three versions of the MNIST CNN with varying levels of data precision (FP32, FP16, INT8). Fig. 41 plots the neutrons cross section of the three designs. We classify output errors as tolerable or critical, depending on whether or not the image classification was affected.

As we have shown in Table 10 and Fig 38, lower precision hardware means lower resource utilization, which consequently reduces the sensitive area of the FPGA. Specifically, the 16-bit floating-point implementation uses about 40% fewer resources than the 32-bit version, while the 8-bit integer design decreases resource utilization by 64%. Fig. 41 shows that, as we reduce precision (area), the probability of observing radiation-induced data corruptions diminishes. The 16-bit floating-point version of the network has a 22% lower chance of producing errors at the output, when compared to the original 32-bit design. Similarly, when using 8-bit integers, we see a massive 72% cross section reduction from FP32.

A more nuanced result that can be drawn from our experimental data is that the rate of critical errors is very different across the three designs. If we think about the way floating-point numbers are represented in computers (sign, exponent, and mantissa), it is easy to conclude that radiation-induced corruptions in the sign and in the exponent of a number can cause much greater discrepancy. As the IEEE 754 standard establishes, the 32-bit floating-point representation uses 1 bit for the sign, 8 bits for the exponent, and 23 for the mantissa, while the 16-bit floating-point

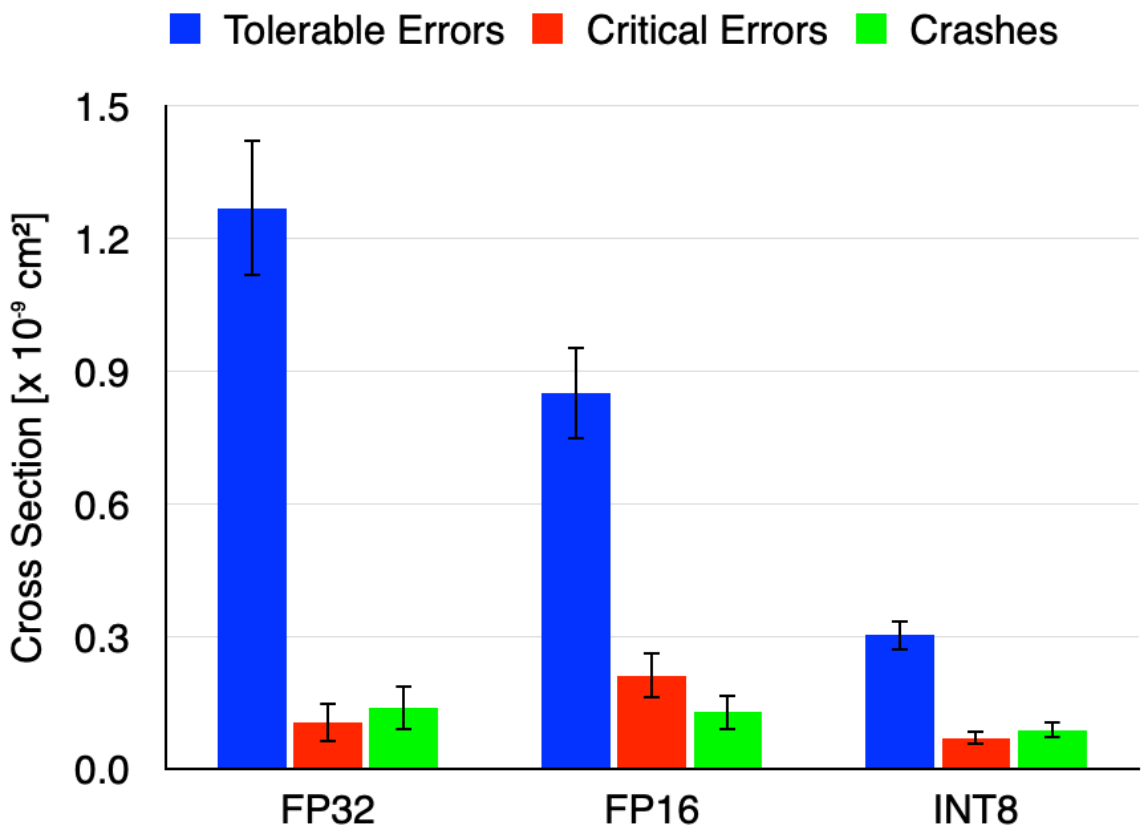

Figure 41: Neutrons cross section for the FP32, FP16 and INT8 versions of the MNIST CNN.

representation uses 1, 5, and 10 respectively. We can say that 28.13% (9/32) of the bits in an FP32 word have a high potential of causing significant discrepancies between expected and computed outputs, while 37.50% (6/16) of the bits in an FP16 word could lead to considerable differences. In our CNN, this ultimately means that, as we reduce precision from FP32 to FP16, the likelihood of an output error affecting the final classification of the input image increases. This result can even be intuitively generalized for a hypothetical FP8 representation, which would very likely have a critical error rate higher than FP16. However, this pattern does not continue when we further reduce precision to 8-bit integers. This is because there are no exponent bits in an integer representation, so when a given bit n gets corrupted, the difference between

expected and computed value can only be $\pm 2^n$, as opposed to a $x2^n$ discrepancy in an exponent corruption of a floating-point number. In our experiments, we have found that only 7% of the errors were critical in the FP32 design, while the error criticality rose to nearly 18% on the FP16 design and then fell back down to about 15% on the INT8 version of the MNIST CNN. Regardless, 8-bit integer representation had the lowest absolute critical error rate out of all three implementations.

Additionally, we have also observed instances where the CNN implemented on the FPGA did not provide any outputs after its expected execution time. This happens whenever the FSM responsible for the control logic of the hardware gets stuck, failing to reach its *done* state. Fig. 41 shows these as *Crashes*. As this type of event is much rarer than SDCs (only accounts for 10 to 20% of the total cross section), the error bars do not allow us to draw any conclusions from the experimental data. But, from an architectural perspective, we intuitively know that reducing the data precision only reduces the area occupied by the arithmetic pipeline, and not the area occupied by the FSM, which means that the likelihood of observing crashes should be roughly the same across all three versions of the MNIST CNN.

The impact of reducing data precision can further be explored as we make use of the notion of TRE (Tolerated Relative Error) (dos Santos et al. 2019). Fig. 42 shows how the neutron cross sections would be reduced if we were to allow a certain percentage of tolerance for discrepancies between expected and computed outputs. With a TRE of 0%, any bitflip in the output is considered an error, but as we increase the TRE, we start to establish intervals of tolerance, instead of boolean decisions. For example, with a TRE of 1%, any output corruption from 99% to 101% of the expected/golden value would still be considered correct. Interestingly, with a TRE level of only 1%, the cross section of tolerable errors on the 32-bit floating-point version of the CNN is

reduced by 43%. This is because, as we previously discussed, 23 out of 32 bits are reserved for the mantissa, which means that most of the radiation-induced corruptions will not have a very significant impact on an FP32 word. Extending the comparison, if we use the same 1% tolerance interval, the error rate on the FP16 implementation would only improve by 8%, while the 8-bit integer version would not change at all. It can then be said that, by treating neural networks as the inherently approximate computing units that they are, the perceived/effective radiation sensitivity can be significantly reduced. The caveat is that TRE only helps to reduce the tolerable portion of the cross section, as the image classification depends on the comparison between all ten outputs, as opposed to their relationship with gold.

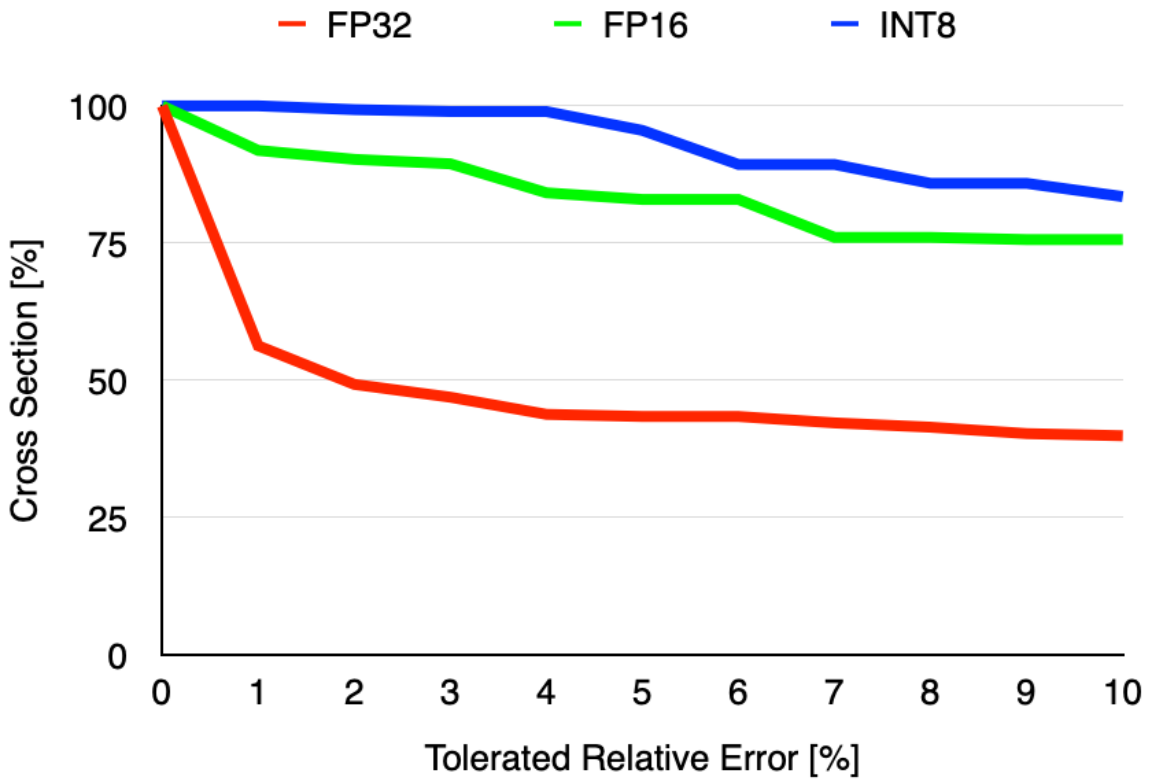

Figure 42: Reduction in the cross sections of the FP32, FP16 and INT8 versions of the MNIST CNN, when establishing incremental tolerance intervals.

In order to get a realistic estimate of the failure rate of an algorithm in a radiation environment, the MEBF metric is commonly utilized (Rech et al. 2014), given that it takes into account both radiation sensitivity as well as performance. However, as we will discuss in Section 7.4.3 neural networks fall into a special category of applications. Since they have an associated level of accuracy, they can experience application failures regardless of radiation. In this section though, we focus our analysis strictly on radiation effects, using the Mean Executions Between Radiation Failures (MEBRF) metric. We show the MEBRF of our designs in Table 13, considering the neutron flux at sea level (JEDEC 2006), and the entire SDC cross section measurements (tolerable+critical errors), as legislation typically does not distinguish between error categories (ISO 2021). The INT8 version of the MNIST CNN is able to complete a much higher number of failure-free executions than the floating-point implementations. FP32 experiences over 6 times the failure rate of INT8, while FP16 fails over 3 times as much as the integer-based design. This is because reduced precision hardware not only occupies less area, but is also faster. The end result is lower radiation sensitivity and higher reliability. A true win-win situation provided that accuracy remains stable.

Table 13: Mean Executions Between Radiation Failures (MEBRF) for the FP32, FP16 and INT8 versions of the MNIST CNN.

	FP32	FP16	INT8
MEBRF [x 10^{15}]	0.61	1.11	4.08

7.4.2 Degree of Parallelism

Using the experimental methodology described in Section 7.3, we tested two versions of the MNIST CNN with varying degrees of parallelism (Min PEs, Max PEs). As previously mentioned, the two designs in this analysis represent the two ends of the parallelism spectrum. On the one hand, we have a very small, iterative design, which uses few resources (therefore occupies less area), and takes longer to complete the processing of input images (i.e. has lower performance). On the other side, we have a very large, fully parallel implementation, which occupies almost 100% of a state-of-the-art FPGA, but delivers extremely low latency. Such scenario was made evident in Section 7.3.1 (Table 11 and Fig. 40), but it is worth reiterating.

Fig. 43 plots the neutrons cross section of the two designs. We classify output errors as tolerable or critical, depending on whether or not the image classification was affected. Noting that the y-axis is in a logarithmic scale, the difference in radiation sensitivity between the two implementations is striking: the fully parallel design is 133 times more likely to experience radiation-induced errors. Interestingly, it uses 130 times more resources than the version with far less processing elements, confirming the relationship between resource utilization and cross section on FPGAs.

Furthermore, the percentage of critical errors observed in our experiments was roughly the same in the two design variations (16% and 17%, for Min PEs and Max PEs, respectively). As there is no variation in data precision here, the impact of data corruption during computation is about the same in both cases, so this was an expected outcome. The more parallel version of the MNIST CNN did not experience any crashes (although one event is assumed). By being a fully streaming architecture, it does not need/have any sort of FSM, and therefore nothing to get stuck.

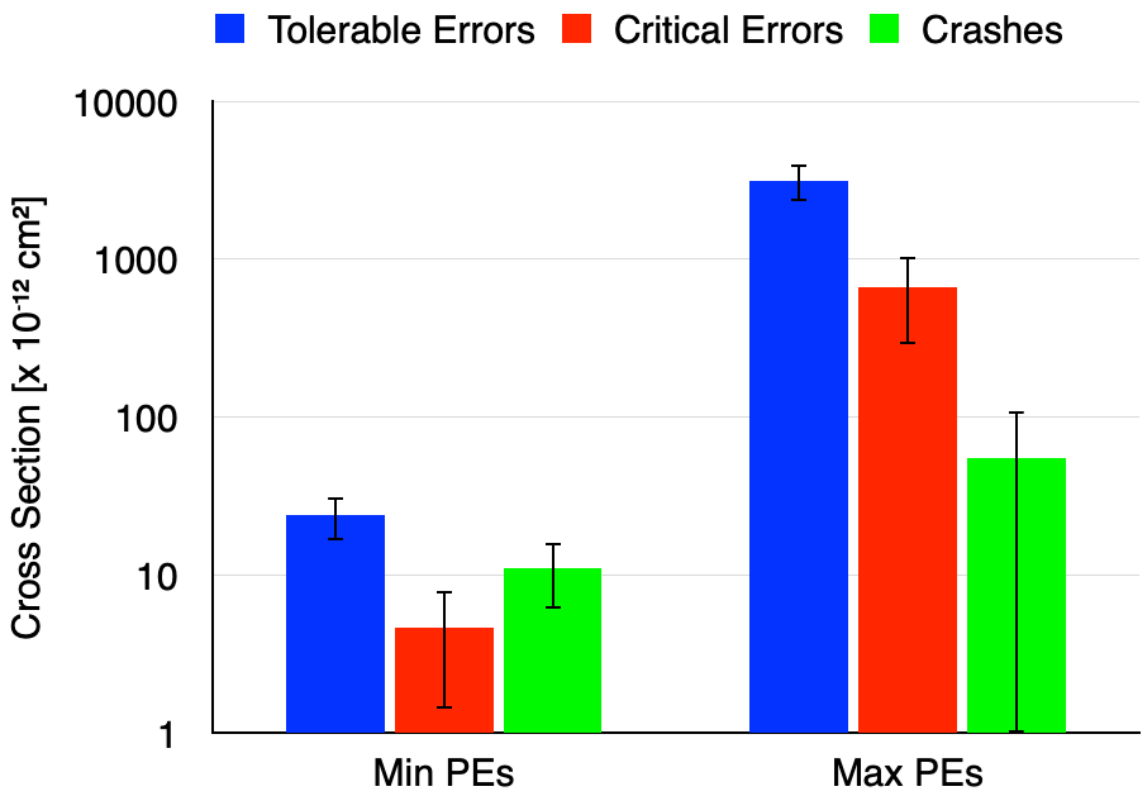

Figure 43: Neutrons cross section for the Min PEs and Max PEs versions of the MNIST CNN. Note that the y-axis is in logarithmic scale.

Again, we should emphasize that the cross section only measures the level of radiation sensitivity of an algorithm/circuit/device. In order to get an estimation of failure rate, one must also consider a performance metric (execution time) as a variable. Thus, we show the MEBRF of our DUTs in Table 14, considering the neutron flux at sea level (JEDEC 2006). Surprisingly, despite having a much higher cross section, the fully parallel implementation of the network is able to complete over 7 times more error-free executions than the smaller design. This is because, as reported in Section 7.3.1 (Fig. 40), the more parallel design is over 1000x faster.

Different than the analysis in Section 7.4.1, where the precision reduction led to smaller and faster hardware, we are seeing that the degree of parallelism in an

Table 14: Mean Executions Between Radiation Failures (MEBRF) for the Min PEs and Max PEs versions of the MNIST CNN.

	Min PEs	Max PEs
MEBRF [x 10^{17}]	0.65	5.00

architecture is always a trade-off between area and performance, which both directly impact the overall reliability of the system. Our experimental data indicates that a faster (more parallel) design will deliver the lower failure rate, since the performance gains outweigh the increased area and radiation sensitivity. However, as we said before, our case study CNN was specifically chosen to enable this analysis, and for most state-of-the-art neural network architectures, instantiating all of the necessary processing elements in the FPGA would be unfeasible. Therefore, it follows that the optimal neural network hardware accelerator architecture (from the reliability standpoint) should just be *as parallel as possible*.

7.4.3 Accuracy Level

Using the experimental methodology described in Section 7.3 we injected faults separately in the Feature Extraction and Classification parts of our Baseline MNIST CNN and our Quantized MNIST HNN. Fig. 44 shows our results expressed in terms of AVF. Furthermore, we divide the total AVF into the error categories discussed in Section 4.4. Although the impact of benign errors is almost negligible, the percentage of benign errors in the Quantized HNN is about 3 times higher than on the Baseline CNN (even though their accuracies are only 5% apart from each other). In addition, whenever faults occur in earlier stages of the neural networks (in the feature extraction part), benign errors are a bit more likely manifested at the output.

From Fig. 44, it is easily perceived that tolerable errors are the majority across all cases. In addition, a fault injected in the feature extraction portion of the Quantized HNN has a 24% higher chance of generating a critical error at the output, when compared to a fault injected on the feature extraction portion of the Baseline CNN. Likewise, a fault in the classification part of the Quantized HNN has a 49% higher chance of provoking an erroneous image identification, when compared to a fault in the classification part of the Baseline CNN. Thus, the binary quantization of convolutional layers led to lower architectural vulnerability, but higher error criticality.

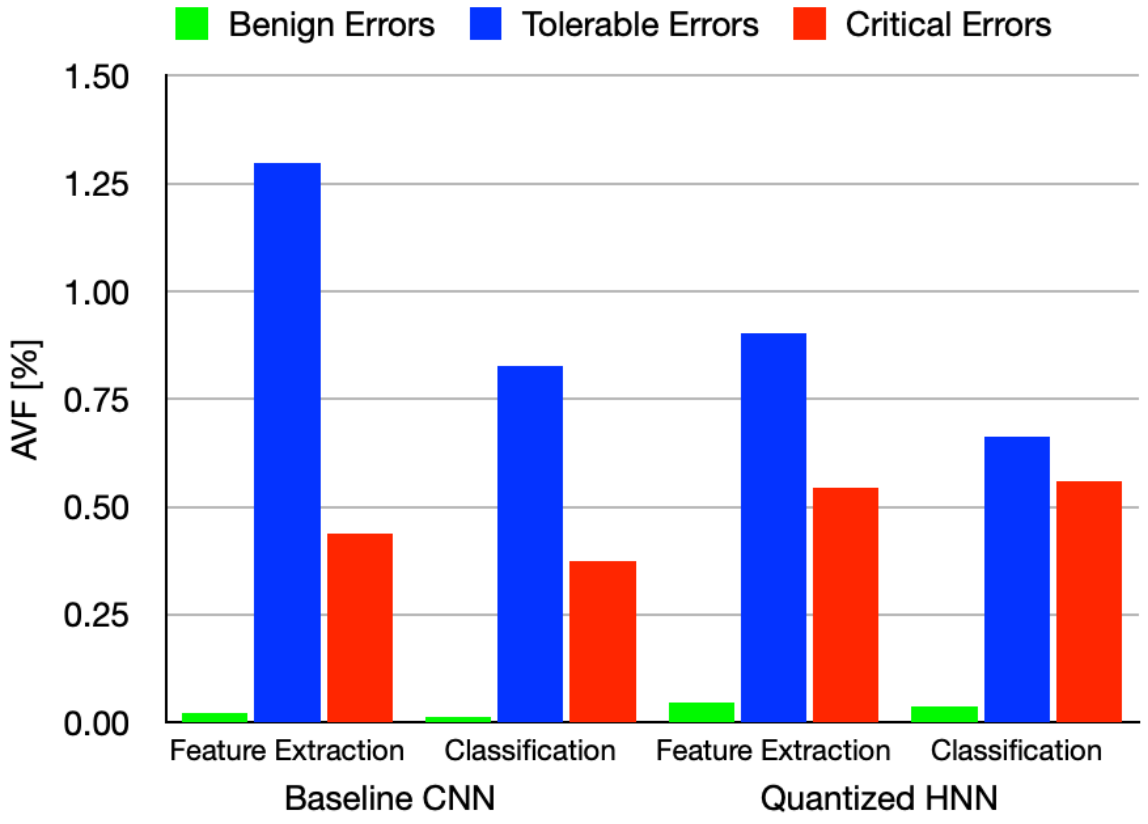

Figure 44: AVF calculated from fault-injection in the feature extraction and classification portions of the Baseline CNN and the Quantized HNN.

Furthermore, we have also used our fault-injection experimental data to calculate the impact of upsets on the accuracy of each neural network. In order to do so, we simply calculated a variation *(delta)*, from the starting fault-free accuracy to the final experimentally-calculated accuracy (accounting for the observed critical and benign errors). As the fault-free accuracy is obviously higher, the variations end up being negative (-0.34% for the Baseline CNN and -0.53% for the Quantized HNN). As the Quantized HNN has a higher level of error criticality, a larger percentage of upsets are going to disturb the final classification computed by the neural network, which ultimately leads to lower accuracy in a radiation environment.

Based on this scenario, we present a mathematical model that considers both sources of failures (critical errors): **inaccuracy** and **radiation**. From the inaccuracy standpoint, if a neural network is said to have an accuracy of a, it means that it will wrongly classify an input with a probability of *(1 - a)*. It follows that the MEBF (here named MEBIF, where 'I' stands for 'Inaccuracy') would be given by:

$$MEBIF(a) = \frac{a}{1-a} \tag{7.1}$$

It is worth noticing that Eq. 7.1 does not depend on the cross section, particle flux, or execution time of the circuit. Even though we are referring to *"circuit"* in the scope of this work, this analysis would also apply to devices other than FPGAs.

From the radiation point of view, all of these variables become relevant. If we multiply the cross section by a given particle flux (e.g. the neutron flux at sea level), we get the *Failure in Time (FIT)* metric, which, as the name suggests, indicates how many failures we expect to observe in a time interval. Thus, the inverse of the FIT becomes the *Mean Time To Failure (MTTF)* which is quite self-explanatory. Finally, dividing the MTTF by the execution time, we get the MEBF. Eq. 7.2 expresses the

MEBF (here named MEBRF, where 'R' stands for 'Radiation') as a function of *'c': cross section, 'f': particle flux, 'e': execution time*:

$$MEBRF(c, f, e) = \frac{1}{c \cdot f \cdot e} \tag{7.2}$$

From Eq. 7.2 we can see the following:

1) If the cross section is zero (meaning that our design is rad-hard), MEBRF goes to infinity (meaning that we will never see a radiation-induced failure).

2) If the particle flux is zero (meaning that we are in a radiation-free environment), MEBRF also goes to infinity.

3) If the execution time tends to zero, MEBRF goes to infinity again (meaning that there is only an infinitesimally small time interval in which an impinging particle could affect the circuit).

Lastly, we need to come up with an equation that combines Eq. 7.1 and Eq. 7.2. It is quite obvious that a critical error can only be induced by radiation when the expected (fault-free) execution originally led to correct classification. With that in mind, if we solve Eq. 7.3 for x, we will get a number in the interval [0,1] that symbolizes how the accuracy of the network is affected (attenuated) in the presence of radiation (noise).

$$MEBRF(c, f, e) = \frac{x}{1 - x} \tag{7.3}$$

Note that Eq. 7.3 looks very similar to Eq. 7.1. This is because, in a way of looking at it, we are calculating the *"accuracy of the accuracy"* of the neural network, or the *"radiation-induced attenuation factor"* on the original accuracy. Also note that

the higher the MEBRF, the closer x gets to 1 (meaning the less attenuation there is). Solving Eq. 7.3 for x gives us:

$$x(c, f, e) = \frac{MEBRF(c, f, e)}{1 + MEBRF(c, f, e)} \tag{7.4}$$

$$x(c, f, e) = \frac{1}{1 + (c \cdot f \cdot e)} \tag{7.5}$$

Our very last step is to calculate the overall MEBF as a function of *'a': accuracy, 'c': cross section, 'f': particle flux, 'e': execution time*:

$$MEBF(a, c, f, e) = \frac{a \cdot x(c, f, e)}{1 - a \cdot x(c, f, e)} \tag{7.6}$$

$$MEBF(a, c, f, e) = \frac{a}{1 + c \cdot f \cdot e - a} \tag{7.7}$$

Notice that for either c, f, or e equal to zero, Eq. 7.7 trivially becomes Eq. 7.1 (if there are no radiation effects, the only source of errors is inaccuracy).

Likewise, for a=1, Eq. 7.7 reduces to Eq. 7.2 (if the network is fully accurate, the only source of errors is radiation). These relationships between Eq. 7.1, Eq. 7.2, and Eq. 7.7 prove the consistency in our findings.

Finally, by looking at Eq. 7.7 we arrive to the following, very straight-forward, conclusions regarding how one would be able to reduce the failure rate of neural networks:

a) Increase the accuracy

c) Make it rad-hard

f) Get away from radiation

e) Reduce the execution time

Fig. 45 shows how the MEBF varies as a function of the *cfe* product for different values of accuracy. We can clearly see that the curves on Fig. 45 stay flat for the most part. But, precisely at the point when they start to drop, is where we should start worrying about radiation effects. The graph shows that, if we have a 99.99% accurate model, such tipping point is at $cfe=10^{-5}$. Likewise, a model with 99.9% accuracy should start facing significant radiation-induced problems at $cfe=10^{-4}$. As the red and the black curves follow a similar pattern, we can draw a generic conclusion: as we give up one *9* of accuracy, we postpone our radiation problem by 10x.

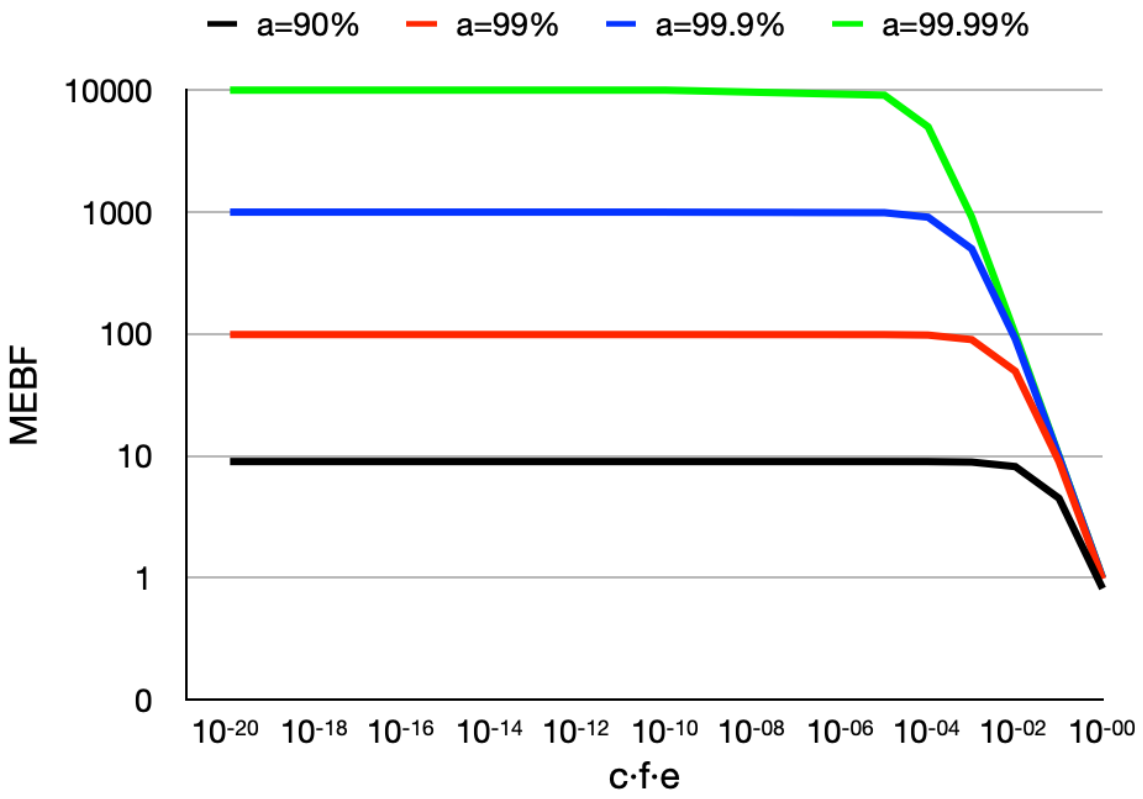

Figure 45: Mean Executions Between Failures (MEBF) as a function of inaccuracy and radiation.

7.4.4 Technology Node

Showcasing the difference between FPGAs in different technology nodes was not one of our goals, but the INT8 circuit that was tested on the Zynq-7000 happens to be the exact same as the Min PEs design that was tested on the Zynq UltraScale+. Therefore, we plot and report their dynamic neutron cross sections together (Fig. 46). The older 28nm CMOS is around 1 order of magnitude more sensitive to radiation than the newer 16nm FinFET, which, considering statistical errors, is in line with the static cross section numbers reported by Xilinx (Xilinx 2020c).

Previous works have explored the differences in radiation sensitivity across technology nodes, through a mixture of charge collection simulations, and real-world beam experiments with neutrons/heavy-ions (Fang and Oates 2011) (Nsengiyumva et al. 2016). As such, our experimental data is consistent.

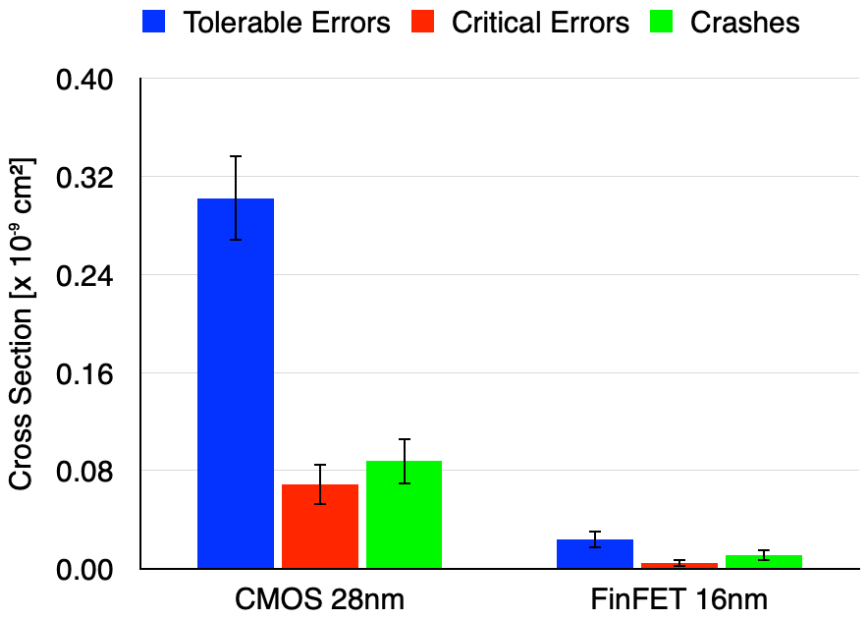

Figure 46: Neutrons cross section for the MNIST CNN on the 28nm Zynq-7000 and the 16nm Zynq UltraScale+.

Chapter 8

CONCLUSION

This dissertation intended to analyze and improve the reliability of matrix multiplication and neural networks on FPGAs. As mentioned before, matrix multiplication is a pervasive algorithm, dominating computational workloads across a multitude of socially significant applications, such as weather forecasting, financial market prediction, radio signal processing, and computer vision. Specifically, matrix multiplication is also at the core of neural network computation. By its turn, neural networks have been increasingly deployed in said areas, as a way of enabling autonomous systems. Since some of these applications have both real-time and safety-critical constraints, analyzing and improving the reliability of matrix multiplication and neural networks becomes mandatory. As made clear within the context of this dissertation, *analyzing* is a synonym for *experimenting*, and *improving* is a synonym for *hardening*. Moreover, our motto for hardening was centered on efficiency. In other words, we strove to achieve high levels of reliability with minimal added costs.

Although very cliché, when it comes to developing efficient hardening techniques, it typically holds true that *you cannot know where you are going without knowing where you are*. Less abstractly, before employing any efforts in improving reliability, it is fundamental to measure baseline radiation sensitivity. In our case, we have conducted neutron beam experiments with the current state-of-the-art neural network accelerator for FPGAs (Xilinx's DPU). As discussed in Chapter 5, the trade-off between area and performance for different DPU core configurations is slightly biased towards the latter. This is because, even though a more capable core occupies a larger area (thus

having a higher cross section), its performance increase contributes to a higher MEBF. However, our experiments have shown that the lowest measured FIT rate for the DPU is still twice as high as the limit specified by the ISO 26262 for self-driving vehicles. Therefore, we have established that improvements are indeed necessary.

Moreover, we have also highlighted that architectural choices have direct impacts on the reliability of neural network accelerators. Through the course of Chapter 7, we have experimented with several design variations that are particularly representative of the design space, which allowed us to reason about hypothetical architectures. We have seen that by reducing the data precision of arithmetic cores it is possible to greatly improve reliability, since area diminishes and performance increases. The only caveat, however, is that weight quantization procedures must be lossless (i.e. must maintain the neural network's original accuracy level). In addition, we have noted that architectures with higher degrees of parallelism tend to be more reliable. Even though larger groups or processing elements occupy larger areas, the corresponding performance gains overshadow such costs. Despite having a higher probability of getting hit by impinging particles, more parallel accelerators are able to complete a higher number of failure-free executions.

From another standpoint, we have noted that radiation-induced failures might be, in some scenarios, a secondary concern for the overall reliability of inherently inaccurate applications (such as neural networks). In other words, as another insight of Chapter 7, we acknowledge that neural networks can fail due to lack of accuracy or due to radiation. Therefore, we provide a novel expression for failure rate that properly considers the two aggregate failure probabilities. Most importantly, our equation allows for precisely identifying the onset point of radiation effects at any given level of accuracy.

Additionally, in Chapter 6 we have argued that hardening the matrix multiplication algorithm is an efficient way of hardening neural networks. Given that MxM represents a significant percentage (around 90%) of the workload in modern CNNs, improving MxM's reliability has a global impact at the application level. As thoroughly discussed, specific ABFT techniques exist for MxM, delivering error detection/correction capabilities at lower costs than traditional modular redundancy methods. Other layers in CNN topologies, such as MaxPooling and ReLU, tend to be less critical. Since they only propagate a few of their inputs forwards, these layers have an embedded masking ability. As we mentioned in Chapter 2, this characteristic was experimentally evaluated and thoroughly discussed in prior work (Santos et al. 2019). The authors also proposed a specific hardening method for pooling layers, achieving over 90% error coverage with low added cost. Therefore, we focus on matrix multiplication, and on the most compute-intensive layers in CNNs.

However, current MxM ABFT strategies are sub-optimal for FPGAs and their aforementioned peculiar fault model. Since faults affecting the FPGA's configuration memory exhibit persistent effects, once the PL's behavior becomes faulty, most subsequent arithmetic operations tend to present errors. As a result, the correction procedures established by said ABFT strategies would have to be triggered at every algorithm iteration, until the active scrubbing of the CRAM fault. Therefore, we believe that a wiser choice for such context is to focus on cheap detection methods. We have presented, formally proved, and experimentally validated *Light ABFT*, decreasing both algorithmic and architectural costs by over a polynomial degree, when compared to the state-of-the-art. Furthermore, we have provided an open-source architectural implementation of our novel technique that adds error detection capabilities to systolic arrays, with associated area and energy overheads of less than 1%.

Although the contributions in this dissertation have both breadth and depth, they too have some limitations:

- DPU experiments were performed with neutrons only, meaning that our evaluation accounts for terrestrial applications, but not for space applications (which would require heavy-ions).
- Architectural variations were only tested with a simple case study CNN topology, meaning that our results are indicative but not fully generic.
- The *MEBRF* equation might not be sufficient to model failure rates in complex systems, as with multiple devices/applications, multiple input variables would arise, requiring more intricate mathematical models.
- The architectural implementation of *Light ABFT* only considers 8-bit integer inputs, which means that further design effort would be required for its deployment in scenarios where floating-point representation is mandatory.

8.0.1 Future Work & Discussion

Naturally, as this dissertation does not completely solve the research field, there are plenty of future work opportunities available. Tackling the aforementioned limitations would be, of course, the most obvious path. Particularly though, we believe that adding a configurable option in our open-source module generator, for floating-point processing elements in our systolic array and *Light ABFT*, would significantly expand the spectrum in which our solutions can be deployed. We anticipate that some nuance would be required for an adequate comparison implementation within the *Light ABFT* module, as the floating-point calculations would likely have small differences in their mantissas.

Specifically, due to floating-point operations being non-associative, rounding errors could cause inexact (bitwise) comparisons. As in *Light ABFT* we compute the value of L via the inputs, and via the outputs, the difference between such values should be below a pre-established threshold. Previously, in Chapter 7, we used the notion of thresholds for determining intervals of tolerance in output errors (when compared on a percentage basis to undisturbed gold values). In this case, however, a numerical analysis would be necessary: running our algorithm for sufficiently large/random input patterns, and keeping track of the difference in each comparison, while calculating the mean and the standard deviation for a Gaussian distribution. Given such statistical data, it would be possible to appropriately choose a threshold value with an associated degree of both coverage and confidence. Of course, if the chosen threshold is too low, false error detections could be triggered. Likewise, if the chosen threshold is too high, actual errors could go undetected. Additionally, given fixed array sizes (and the number of sequential accumulations), it is possible to estimate the magnitude of the maximum rounding error. Using this information, an extra *sanity-check* could be implemented in the comparators: if the difference is higher than said maximum, the outputs most definitely contain an error.

Since our proof for Light ABFT is purely mathematical, it is also device-independent. Even though we have focused on FPGAs, our suggested architecture could easily be implemented on an ASIC. For instance, in the case of car manufacturers, fabricating an industry-standard component could be viable in the long term due to scale (i.e. the large number of vehicles that are produced and sold every year). On the other hand, FPGAs solutions could be better suited for spacecraft. As much fewer missions take place, it becomes harder to justify fabrications costs. Moreover, reprogrammability is even more important in space applications.

In an ASIC implementation though, some reprogrammability could be engrained as a part of the microarchitecture. For instance, given sufficiently capable processing elements, different levels of data precision could be used as needed (for different requirements of specific workloads and for energy-saving purposes). On a micro-scale (of a single systolic accelerator), *Light ABFT* would not have to change. On a macro-scale (with multiple arrays in parallel), more complex control mechanisms would be needed, in order to re-direct computation between units as errors occur. Luckily, since transient faults in computation would become the majority, the pipelined nature of the streaming architecture would automatically flush them away, and full device reconfiguration would likely not be necessary. In other words, by accounting for the more intricate fault model of FPGAs, simpler devices are also contemplated.

Finally, we could argue that *Light ABFT* could also be employed in adjacent, highly relevant arithmetic workloads, such as FFT. Given the famous Cooley-Tukey algorithm (Cooley and Tukey 1965), it is possible to factorize and interleave the original FFT operation using smaller blocks, which are commonly referred to as butterfly stages. The key aspect, in this case, is that FFT is a linear transformation. Thus, as the authors show, it can be mathematically modeled with a sequence of matrix operations. As a result, our approach to hardening matrix multiplication could be used agnostically regarding context. Moreover, Parseval's theorem (Hardy and Titchmarsh 1931) states that the Fourier transform is *unitary*: the integral (sum) of the square of the inputs is equal to the integral (sum) of the square of the outputs. In this sense, a simple checksum verification could be used to assert correctness in an FFT operation, which is quite similar to the principle behind *Light ABFT*. This error detection methodology could then be expanded for other mathematical operations and algorithms for which the unitary property holds true (e.g. large radius filtered image

convolution). It could also be adapted for *effectively unitary* cases, in which there is a predictable scalar difference between inputs and outputs. Generally though, the price of producing a highly efficient ABFT technique (and a corresponding architectural implementation) is only justified when the workload itself is inherently expensive. For simpler cases, with less strict area and energy budgets, traditional modular redundancy should suffice.

REFERENCES

Afzaal, Umar, and Jeong-A Lee. 2018. "A Self-Checking TMR Voter for Increased Reliability Consensus Voting in FPGAs." *IEEE Transactions on Nuclear Science* 65 (5): 1133–1139. doi:10.1109/TNS.2018.2824821.

Aldahlawi, A., Y. Kim, and K. K. Kim. 2019. "GPU Architecture Optimization For Mobile Computing." In *2019 International SoC Design Conference (ISOCC),* 247–248.

Andersch, M., J. Lucas, M. A. LvLvarez-Mesa, and B. Juurlink. 2015. "On latency in GPU throughput microarchitectures." In *2015 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS),* 169–170.

Anwer, J., M. Platzner, and S. Meisner. 2014. "FPGA Redundancy Configurations: An Automated Design Space Exploration." In *2014 IEEE International Parallel Distributed Processing Symposium Workshops,* 275–280. doi:10.1109/IPDPSW.2014.37.

Aranda, Luis Alberto, Pedro Reviriego, and Juan Antonio Maestro. 2018. "A Comparison of Dual Modular Redundancy and Concurrent Error Detection in Finite Impulse Response Filters Implemented in SRAM-Based FPGAs Through Fault Injection." *IEEE Transactions on Circuits and Systems II: Express Briefs* 65 (3): 376–380. doi:10.1109/TCSII.2017.2717490.

Australian Travel Safety Bureau. 2008. "In-flight upset - Airbus A330-303, VH-QPA, 154 km west of Learmonth, WA, 7 October 2008." Accessed July 1, 2021. https://www.atsb.gov.au/publications/investigation_reports/2008/aair/ao-2008-070.aspx.

Avizienis, Algirdas. 1967. "Design of Fault-Tolerant Computers." In *Proceedings of the November 14-16, 1967, Fall Joint Computer Conference,* 733–743. AFIPS '67 (Fall). Anaheim, California: Association for Computing Machinery. doi:10.1145/1465611.1465708.

Avizienis, Algirdas, Jean-Claude Laprie, and Brian Randell. 2001. "Fundamental Concepts of Dependability" (April).

Baumann, R. C. 2001. "Soft Errors in Advanced Semiconductor Devices-Part I: The Three Radiation Sources." *IEEE Transactions on Device and Materials Reliability* 1 (1): 17–22. doi:10.1109/7298.946456.

Baumann, R. C. 2005. "Radiation-induced soft errors in advanced semiconductor technologies." *IEEE Transactions on Device and Materials Reliability* 5 (3): 305–316.

Benevenuti, F., and F. L. Kastensmidt. 2019. "Comparing Exhaustive and Random Fault Injection Methods for Configuration Memory on SRAM-based FPGAs." In *2019 IEEE Latin American Test Symposium (LATS),* 111–116. March. doi:10.1109/LATW.2019.8704647.

Bianco, Simone, Rémi Cadène, Luigi Celona, and Paolo Napoletano. 2018. "Benchmark Analysis of Representative Deep Neural Network Architectures." *IEEE Access* 6 (October): 64270–64277. doi:10.1109/ACCESS.2018.2877890.

BMW. 2021. "The path to autonomous driving." Accessed July 1. https://www.bmw.com/en/automotive-life/autonomous-driving.html.

Borkar, Shekhar, Robert Cohn, George Cox, Thomas Gross, H. T. Kung, Monica Lam, Margie Levine, et al. 1990. "Supporting Systolic and Memory Communication in IWarp." *SIGARCH Comput. Archit. News* (New York, NY, USA) 18, no. 2SI (May): 70–81. doi:10.1145/325096.325116.

Bruguier, G., and J.-M. Palau. 1996. "Single particle-induced latchup." *IEEE Transactions on Nuclear Science* 43 (2): 522–532. doi:10.1109/23.490898.

Cambridge Spark. 2020. "Deep learning for complete beginners: convolutional neural networks with keras." Accessed March 30. https://cambridgespark.com/content/tutorials/convolutional-neural-networks-with-keras/index.html.

CERN. 2020. "CERN High Energy Accelerator Mixed Field." Accessed March 30. http://charm.web.cern.ch/.

Chellapilla, Kumar, Sidd Puri, and Patrice Simard. 2006. "High Performance Convolutional Neural Networks for Document Processing." In *10th International Workshop on Frontiers in Handwriting Recognition.* La Baule (France): Université de Rennes 1, Suvisoft, October. https://hal.inria.fr/inria-00112631.

CNBC. 2021a. "Tesla reports 499,550 vehicle deliveries for 2020, slightly missing target." Accessed July 1. https://www.cnbc.com/2021/01/02/tesla-tsla-q4-2020-vehicle-delivery-and-production-numbers.html.

———. 2021b. "Toyota beats Volkswagen to become world's No.1 car seller in 2020." Accessed July 1. https://www.cnbc.com/2021/01/28/toyota-beats-volkswagen-to-become-worlds-nopoint1-car-seller-in-2020.html.

Coblentz, W. W. 1914. "Note on the Radiation from Stars." *Publications of the Astronomical Society of the Pacific* 26 (October): 169. doi:10.1086/122330.

Computer History Museum. 2020. "First Colossus operational at Bletchley Park." Accessed March 30. https://www.computerhistory.org/timeline/1944/.

Cooley, James W., and John W. Tukey. 1965. "An algorithm for the machine calculation of complex Fourier series." *Mathematics of Computation* 19:297–301.

Data Science Stack Exchange. 2020. "Convolution." Accessed March 30. https://datascience.stackexchange.com/questions/23183/why-convolutions-always-use-odd-numbers-as-filter-size/23186.

Deng, Jia, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. 2009. "ImageNet: A large-scale hierarchical image database." In *2009 IEEE Conference on Computer Vision and Pattern Recognition,* 248–255. doi:10.1109/CVPR.2009.5206848.

Department of Nuclear Physics - USP. 2020. "Nuclear Physics Open Laboratory." Accessed March 30. http://portal.if.usp.br/fnc/en/nuclear-physics-open-laboratory.

Di Carlo, S., P. Prinetto, D. Rolfo, and P. Trotta. 2014. "A Fault Injection Methodology and Infrastructure for Fast Single Event Upsets Emulation on Xilinx SRAM-based FPGAs." In *2014 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT),* 159–164. doi:10.1109/DFT.2014.6962073.

dos Santos, F. F., P. Navaux, L. Carro, and P. Rech. 2019. "Impact of Reduced Precision in the Reliability of Deep Neural Networks for Object Detection." In *2019 IEEE European Test Symposium (ETS),* 127–132. May.

dos Santos, L. A. 2020. "Artificial Intelligence - Machine Learning - Convolution Layer - Making Faster." Accessed March 30. https://leonardoaraujosantos.gitbook.io/artificial-inteligence/.

Draghetti, L. K., F. F. d. Santos, L. Carro, and P. Rech. 2019. "Detecting Errors in Convolutional Neural Networks Using Inter Frame Spatio-Temporal Correlation." In *2019 IEEE 25th International Symposium on On-Line Testing and Robust System Design (IOLTS),* 310–315. doi:10.1109/IOLTS.2019.8854431.

EE Times. 2013. "Toyota Case: Single Bit Flip That Killed." Accessed July 1, 2021. https://www.eetimes.com/toyota-case-single-bit-flip-that-killed/.

Fang, Y., and A. S. Oates. 2011. "Neutron-Induced Charge Collection Simulation of Bulk FinFET SRAMs Compared With Conventional Planar SRAMs." *IEEE Transactions on Device and Materials Reliability* 11 (4): 551–554.

Fiala, David, Frank Mueller, Christian Engelmann, Rolf Riesen, Kurt Ferreira, and Ron Brightwell. 2012. "Detection and correction of silent data corruption for large-scale high-performance computing." In *SC '12: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis,* 1–12. doi:10.1109/SC.2012.49.

Gonçalves de Oliveira, D. A. G., L. L. Pilla, T. Santini, and P. Rech. 2016. "Evaluation and Mitigation of Radiation-Induced Soft Errors in Graphics Processing Units." *IEEE Transactions on Computers* 65 (3): 791–804. doi:10.1109/TC.2015.2444855.

González-Toral, Ricardo, Pedro Reviriego, Juan Antonio Maestro, and Zhen Gao. 2018. "A Scheme to Design Concurrent Error Detection Techniques for the Fast Fourier Transform Implemented in SRAM-Based FPGAs." *IEEE Transactions on Computers* 67 (7): 1039–1045. doi:10.1109/TC.2018.2792445.

Google. 2020. "Advanced Guide to Inception v3 on Cloud TPU." Accessed March 30. https://cloud.google.com/tpu/docs/inception-v3-advanced.

Gussenhoven, M.S., E.G. Mullen, and D.H. Brautigam. 1996. "Improved understanding of the Earth's radiation belts from the CRRES satellite." *IEEE Transactions on Nuclear Science* 43 (2): 353–368. doi:10.1109/23.490755.

Hardy, G. H., and E.C. Titchmarsh. 1931. "A Note on Parseval's Theorem for Fourier Transforms." *Journal of the London Mathematical Society* s1-6, no. 1 (January): 44–48. doi:10.1112/jlms/s1-6.1.44. eprint: https://academic.oup.com/jlms/article-pdf/s1-6/1/44/2436078/s1-6-1-44.pdf.

Hari, S. K. S., M. Sullivan, T. Tsai, and S. W. Keckler. 2021. "Making Convolutions Resilient via Algorithm-Based Error Detection Techniques." *IEEE Transactions on Dependable and Secure Computing:* 1–1. doi:10.1109/TDSC.2021.3063083.

He, Kaiming, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. "Deep Residual Learning for Image Recognition." In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR),* 770–778. doi:10.1109/CVPR.2016.90.

Heiner, J., B. Sellers, M. Wirthlin, and J. Kalb. 2009. "FPGA Partial Reconfiguration via Configuration Scrubbing." In *2009 International Conference on Field Programmable Logic and Applications,* 99–104. doi:10.1109/FPL.2009.5272543.

Hubara, Itay, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. 2017. "Quantized Neural Networks: Training Neural Networks with Low Precision Weights and Activations." *J. Mach. Learn. Res.* 18, no. 1 (January): 6869–6898.

International Electrotechnical Comission. 2020. "Functional Safety." Accessed March 30. https://www.iec.ch/functionalsafety/explained/.

ISO. 2021. "ISO 26262." Accessed July 1. https://www.iso.org/obp/ui/#iso:std:iso:26262:-2:ed-2:v1:en.

JEDEC. 2006. "Tech (2006). Rep. JESD89A, JEDEC Standard." Accessed October 10, 2018. https://www.jedec.org/sites/default/files/docs/jesd89a.pdf.

Johnson, J., W. Howes, M. Wirthlin, D. L. McMurtrey, M. Caffrey, P. Graham, and K. Morgan. 2008. "Using Duplication with Compare for On-line Error Detection in FPGA-based Designs." In *2008 IEEE Aerospace Conference,* 2322–2333.

Jou, J.-Y., and J.A. Abraham. 1988. "Fault-tolerant FFT networks." *IEEE Transactions on Computers* 37 (5): 548–561. doi:10.1109/12.4606.

Jouppi, Norman P., et al. 2017. "In-Datacenter Performance Analysis of a Tensor Processing Unit." In *2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA),* 1–12. New York, NY, USA: IEEE.

Kagami, Shingo. 2010. "High-speed vision systems and projectors for real-time perception of the world." In *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition - Workshops,* 100–107. doi:10.1109/CVPRW.2010.5543776.

Kastensmidt, F. L., L. Sterpone, L. Carro, and M. S. Reorda. 2005. "On the Optimal Design of Triple Modular Redundancy Logic for SRAM-based FPGAs." In *Design, Automation and Test in Europe,* 1290–1295 Vol. 2.

Keren, E., S. Greenberg, N. M. Yitzhak, D. David, N. Refaeli, and A. Haran. 2019. "Characterization and Mitigation of Single-Event Transients in Xilinx 45-nm SRAM-Based FPGA." *IEEE Transactions on Nuclear Science* 66 (6): 946–954. doi:10.1109/TNS.2019.2916151.

Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. 2017. "ImageNet Classification with Deep Convolutional Neural Networks." *Commununications of the ACM* 60, no. 6 (May): 84–90. doi:10.1145/3065386.

Kuang-Hua Huang and J. A. Abraham. 1984. "Algorithm-Based Fault Tolerance for Matrix Operations." *IEEE Transactions on Computers* C-33 (6): 518–528.

Kung, H. T., and E. L. Charles. 1978. "Systolic Arrays (for VLSI)." In *Sparse Matrix Proceedings 1978,* 1:1–29. Philadelphia, PA, USA: Society for Industrial / Applied Mathematics.

Lecun, Y., L. Bottou, Y. Bengio, and P. Haffner. 1998. "Gradient-based learning applied to document recognition." *Proceedings of the IEEE* 86, no. 11 (November): 2278–2324. doi:10.1109/5.726791.

Leray, J.-L., J. Baggio, V. Ferlet-Cavrois, and O. Flament. 2004. "Atmospheric neutron effects in advanced microelectronics, standards and applications." In *2004 International Conference on Integrated Circuit Design and Technology (IEEE Cat. No.04EX866),* 311–321. doi:10.1109/ICICDT.2004.1309974.

Libano, F., P. Rech, B. Neuman, J. Leavitt, M. Wirthlin, and J. Brunhaver. 2021. "How Reduced Data Precision and Degree of Parallelism Impact the Reliability of Convolutional Neural Networks on FPGAs." *IEEE Transactions on Nuclear Science* 68 (5): 865–872. doi:10.1109/TNS.2021.3050707.

Libano, F., P. Rech, L. Tambara, J. Tonfat, and F. Kastensmidt. 2018. "On the Reliability of Linear Regression and Pattern Recognition Feedforward Artificial Neural Networks in FPGAs." *IEEE Transactions on Nuclear Science* 65 (1): 288–295.

Libano, F., B. Wilson, J. Anderson, M. J. Wirthlin, C. Cazzaniga, C. Frost, and P. Rech. 2019. "Selective Hardening for Neural Networks in FPGAs." *IEEE Transactions on Nuclear Science* 66 (1): 216–222.

Libano, F., B. Wilson, M. Wirthlin, P. Rech, and J. Brunhaver. 2020. "Understanding the Impact of Quantization, Accuracy, and Radiation on the Reliability of Convolutional Neural Networks on FPGAs." *IEEE Transactions on Nuclear Science* 67 (7): 1478–1484. doi:10.1109/TNS.2020.2983662.

Live Science. 2010. "Toyota Recall Might Be Caused by Cosmic Rays." Accessed July 1, 2021. https://www.livescience.com/8170-toyota-recall-caused-cosmic-rays.html.

Los Alamos National Laboratory. 2020. "LANSCE." Accessed March 30. https://lansce.lanl.gov/.

Marty, T., T. Yuki, and S. Derrien. 2018. "Enabling Overclocking Through Algorithm-Level Error Detection." In *2018 International Conference on Field-Programmable Technology (FPT),* 174–181. doi:10.1109/FPT.2018.00034.

McKinsey. 2021. "The future of mobility is at our doorstep." Accessed July 1. https://www.mckinsey.com/industries/automotive-and-assembly/our-insights/the-future-of-mobility-is-at-our-doorstep.

Mercedes-Benz. 2021. "Mercedes-Benz Innovation: Autonomous." Accessed July 1. https://www.mercedes-benz.com/en/innovation/autonomous/.

Microsoft. 2021. "Project Catapult." Accessed July 1. https://www.microsoft.com/en-us/research/project/project-catapult/.

Military Machine. 2021. "How Much Does An F-22 Raptor Cost." Accessed July 1, 2021. https://militarymachine.com/f-22-cost/.

Minsky, Marvin, and Seymour Papert. 1988. *Perceptrons: An Introduction to Computational Geometry.* Expanded. Cambridge, MA: MIT Press.

Mitra, S., and E.J. McCluskey. 2000. "Which concurrent error detection scheme to choose?" In *Proceedings International Test Conference 2000 (IEEE Cat. No.00CH37159),* 985–994. doi:10.1109/TEST.2000.894311.

Mukherjee, Shubhendu S., Christopher Weaver, Joel Emer, Steven K. Reinhardt, and Todd Austin. 2003. "A Systematic Methodology to Compute the Architectural Vulnerability Factors for a High-Performance Microprocessor." In *Proceedings of the 36th Annual IEEE/ACM International Symposium on Microarchitecture,* 29. MICRO 36. USA: IEEE Computer Society.

NASA. 2021a. "Mars 2020 Mission Perseverance Rover." Accessed July 1. https://mars.nasa.gov/mars2020/.

———. 2021b. "Mars 2020 Mission Perseverance Rover - Communications." Accessed July 1. https://mars.nasa.gov/mars2020/spacecraft/rover/communications/.

———. 2021c. "Mars Helicopter." Accessed July 1. https://mars.nasa.gov/technology/helicopter/.

———. 2021d. "Radiation Belts with Satellites." Accessed July 1. https://www.nasa.gov/mission_pages/sunearth/news/gallery/20130228-radiationbelts.html.

———. 2021e. "Why Space Radiation Matters." Accessed July 1. https://www.nasa.gov/analogs/nsrl/why-space-radiation-matters.

NBC. 2021. “Mission to rare metal asteroid could spark space mining boom.” Accessed July 1. https://www.nbcnews.com/mach/science/mission-rare-metal-asteroid-could-spark-space-mining-boom-ncna1027971.

Niranjan, S., and J.F. Frenzel. 1996. “A Comparison of Fault-Tolerant State Machine Architectures for Space-Borne Electronics.” *IEEE Transactions on Reliability* 45 (1): 109–113. doi:10.1109/24.488925.

Normand, E., and T. J. Baker. 1993. “Altitude and latitude variations in avionics SEU and atmospheric neutron flux.” *IEEE Transactions on Nuclear Science* 40 (6): 1484–1490.

Nsengiyumva, P., D. R. Ball, J. S. Kauppila, N. Tam, M. McCurdy, W. T. Holman, M. L. Alles, B. L. Bhuva, and L. W. Massengill. 2016. “A Comparison of the SEU Response of Planar and FinFET D Flip-Flops at Advanced Technology Nodes.” *IEEE Transactions on Nuclear Science* 63 (1): 266–272.

Oldham, T.R., and F.B. McLean. 2003. “Total ionizing dose effects in MOS oxides and devices.” *IEEE Transactions on Nuclear Science* 50 (3): 483–499. doi:10.1109/TNS.2003.812927.

Oliveira, Rafael N. M., Alan D. Lüdke, and Cristina Meinhardt. 2019. “Radiation Effects in XOR Logic Gates at 16nm CMOS and FinFET Technology.” In *2019 26th IEEE International Conference on Electronics, Circuits and Systems (ICECS),* 590–593. doi:10.1109/ICECS46596.2019.8964801.

Paul Scherrer Institut. 2020. “Paul Scherrer Institut.” Accessed March 30. https://www.psi.ch/en.

Pedram, Ardavan, Robert A. van de Geijn, and Andreas Gerstlauer. 2012. “Codesign Tradeoffs for High-Performance, Low-Power Linear Algebra Architectures.” *IEEE Transactions on Computers* 61 (12): 1724–1736. doi:10.1109/TC.2012.132.

Phatak, D. S., and I. Koren. 1995. “Complete and partial fault tolerance of feedforward neural nets.” *IEEE Transactions on Neural Networks* 6 (2): 446–456.

Pierce, W. H. 1965. *Failure-tolerant computer design.* New York, NY: Academic Press.

Piovesan, T., H. C. Sartori, J. E. Baggio, and J. R. Pinheiro. 2016. “CubeSat Electrical Power Supplies Optimization — Comparison Between Conventional and Optimal Design Methodology.” In *2016 12th IEEE International Conference on Industry Applications (INDUSCON),* 72–78.

Posewsky, Thorbjörn, and Daniel Ziener. 2016. "Efficient deep neural network acceleration through FPGA-based batch processing." In *2016 International Conference on ReConFigurable Computing and FPGAs (ReConFig),* 1–8. doi:10.1109/ReConFig.2016.7857167.

Quinn, H., and P. Graham. 2005. "Terrestrial-based radiation upsets: a cautionary tale." In *13th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM'05),* 193–202. doi:10.1109/FCCM.2005.61.

Rech, P., C. Aguiar, C. Frost, and L. Carro. 2013. "An Efficient and Experimentally Tuned Software-Based Hardening Strategy for Matrix Multiplication on GPUs." *IEEE Transactions on Nuclear Science* 60 (4): 2797–2804. doi:10.1109/TNS.2013.2252625.

Rech, P., L. L. Pilla, P. O. A. Navaux, and L. Carro. 2014. "Impact of GPUs Parallelism Management on Safety-Critical and HPC Applications Reliability." In *2014 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks,* 455–466. June.

Rosenblatt, F. 1958. "The perceptron: A probabilistic model for information storage and organization in the brain." *Psychological Review* 65 (6): 386–408. doi:10.1037/h0042519.

Ruospo, Annachiara, Alberto Bosio, Alessandro Ianne, and Ernesto Sanchez. 2020. "Evaluating Convolutional Neural Networks Reliability depending on their Data Representation." In *2020 23rd Euromicro Conference on Digital System Design (DSD),* 672–679. doi:10.1109/DSD51259.2020.00109.

Russell, Stuart, and Peter Norvig. 2009. *Artificial Intelligence: A Modern Approach.* 3rd. USA: Prentice Hall Press.

Samuel, A. L. 1959. "Some Studies in Machine Learning Using the Game of Checkers." *IBM Journal of Research and Development* 3 (3): 210–229.

Santos, F. F. d., P. F. Pimenta, C. Lunardi, L. Draghetti, L. Carro, D. Kaeli, and P. Rech. 2019. "Analyzing and Increasing the Reliability of Convolutional Neural Networks on GPUs." *IEEE Transactions on Reliability* 68 (2): 663–677.

Science & Technology Facilities Council. 2020. "ISIS Neutron and Muon Source." Accessed March 30. https://www.isis.stfc.ac.uk/Pages/Chipir-publications.aspx.

Sequin, C. H., and R. D. Clay. 1990. "Fault tolerance in artificial neural networks." In *1990 IJCNN International Joint Conference on Neural Networks,* 703–708 vol.1.

Shin, K. G., and P. Ramanathan. 1994. "Real-time computing: a new discipline of computer science and engineering." *Proceedings of the IEEE* 82 (1): 6–24.

Siegle, Felix, Tanya Vladimirova, Jørgen Ilstad, and Omar Emam. 2015. "Mitigation of Radiation Effects in SRAM-Based FPGAs for Space Applications." *ACM Comput. Surv.* (New York, NY, USA) 47, no. 2 (January). doi:10.1145/2671181.

Simonyan, Karen, and Andrew Zisserman. 2015. "Very Deep Convolutional Networks for Large-Scale Image Recognition." In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings,* edited by Yoshua Bengio and Yann LeCun. http://arxiv.org/abs/1409.1556.

Sridharan, Vilas, and David R. Kaeli. 2009. "Eliminating Microarchitectural Dependency from Architectural Vulnerability." In *2009 IEEE 15th International Symposium on High Performance Computer Architecture,* 117–128. doi:10.1109/HPCA.2009.4798243.

Stanford Institute for Human-Centered Artificial Intelligence. 2021. "2021 AI Index Report." Accessed July 1. https://aiindex.stanford.edu/report/.

Sterpone, L., and N. Battezzati. 2008. "A Novel Design Flow for the Performance Optimization of Fault Tolerant Circuits on SRAM-based FPGA's." In *2008 NASA/ESA Conference on Adaptive Hardware and Systems,* 157–163. doi:10.1109/AHS.2008.59.

Szegedy, C., V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna. 2016. "Rethinking the Inception Architecture for Computer Vision." In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR),* 2818–2826.

Taber, A., and E. Normand. 1993. "Single event upset in avionics." *IEEE Transactions on Nuclear Science* 40 (2): 120–126.

TensorFlow. 2020. "TensorFlow Lite." Accessed January 25. https://www.tensorflow.org/lite.

———. 2021. "TensorFlow." Accessed July 1, 2021. https://www.tensorflow.org/.

Tesla. 2021. "Autopilot." Accessed July 1. https://www.tesla.com/autopilot.

The Sidney Morning Herald. 2017. "The untold story of QF72: What happens when 'psycho' automation leaves pilots powerless?" Accessed July 1, 2021. https://www.

smh.com.au/lifestyle/the-untold-story-of-qf72-what-happens-when-psycho-automation-leaves-pilots-powerless-20170511-gw26ae.html.

Tonfat, Jorge, Lucas Tambara, André Santos, and Fernanda Kastensmidt. 2016. "Method to Analyze the Susceptibility of HLS Designs in SRAM-Based FPGAs Under Soft Errors." In *Proceedings of the 12th International Symposium on Applied Reconfigurable Computing - Volume 9625,* 132–143. Berlin, Heidelberg: Springer-Verlag. doi:10.1007/978-3-319-30481-6_11.

Tschopp, Fabian, Julien N. P. Martel, Srinivas C. Turaga, Matthew Cook, and Jan Funke. 2016. "Efficient convolutional neural networks for pixelwise classification on heterogeneous hardware systems." In *2016 IEEE 13th International Symposium on Biomedical Imaging (ISBI),* 1225–1228. doi:10.1109/ISBI.2016.7493487.

Turing, A. M. 1950. "Computing Machinery and Intelligence." *Mind* LIX, no. 236 (October): 433–460. doi:10.1093/mind/LIX.236.433. eprint: https://academic.oup.com/mind/article-pdf/LIX/236/433/30123314/lix-236-433.pdf.

US Department of Transportation. 2017. "How Much Time Do Americans Spend Behind the Wheel." Accessed July 1. https://www.volpe.dot.gov/news/how-much-time-do-americans-spend-behind-wheel.

US National Highway Traffic Safety Administration. 2015. "Traffic Safety Facts." Accessed July 1. https://www.nhtsa.gov/sites/nhtsa.gov/files/documents/812409_tsf2015dataspeeding.pdf.

Véstias, M., R. P. Duarte, J. T. de Sousa, and H. Neto. 2017. "Parallel Dot-Products for Deep Learning on FPGA." In *2017 27th International Conference on Field Programmable Logic and Applications (FPL),* 248–252. New York, NY, USA: IEEE.

Violante, M., L. Sterpone, A. Manuzzato, S. Gerardin, P. Rech, M. Bagatin, A. Paccagnella, et al. 2007. "A New Hardware/Software Platform and a New 1/E Neutron Source for Soft Error Studies: Testing FPGAs at the ISIS Facility." *IEEE Transactions on Nuclear Science* 54 (4): 1184–1189.

Volkov, Vasily, and James W. Demmel. 2008. "Benchmarking GPUs to Tune Dense Linear Algebra." In *Proceedings of the 2008 ACM/IEEE Conference on Supercomputing,* 1–11. SC '08. Austin, Texas: IEEE Press.

Volvo. 2021. "Autonomous Drive: The next step in advancing safety." Accessed July 1. https://group.volvocars.com/company/innovation/autonomous-drive.

von Neumann, J. 1956. "Probabilistic Logics and the Synthesis of Reliable Organisms from Unreliable Components." In *Automata Studies,* 34:43–98.

Wagstaff, K. L., and Y. Lu. 2017. "Mars orbital image (HiRISE) labeled data set." Accessed March 30, 2020. https://doi.org/10.5281/zenodo.1048301.

Wagstaff, Kiri, You Lu, Alice Stanboli, Kevin Grimes, Thamme Gowda, and Jordan Padams. 2018. "Deep Mars: CNN Classification of Mars Imagery for the PDS Imaging Atlas." In *AAAI Conference on Artificial Intelligence.*

Wirthlin, M. 2015. "High-Reliability FPGA-Based Systems: Space, High-Energy Physics, and Beyond." *Proceedings of the IEEE* 103 (3): 379–389.

Wu, Panruo, Qiang Guan, Nathan DeBardeleben, Sean Blanchard, Dingwen Tao, Xin Liang, Jieyang Chen, and Zizhong Chen. 2016. "Towards Practical Algorithm Based Fault Tolerance in Dense Linear Algebra." In *Proceedings of the 25th ACM International Symposium on High-Performance Parallel and Distributed Computing,* 31–42. HPDC '16. Kyoto, Japan: Association for Computing Machinery. doi:10.1145/2907294.2907315.

Xilinx. 2017. "Deep Learning with INT8 Optimization on Xilinx Devices." Xilinx. Accessed July 1, 2021. https://www.xilinx.com/support/documentation/white_papers/wp486-deep-learning-int8.pdf.

———. 2019a. "UltraScale Architecture DSP Slice User Guide." Xilinx. Accessed July 1, 2021. https://www.xilinx.com/support/documentation/user_guides/ug579-ultrascale-dsp.pdf.

———. 2019b. "Zynq DPU 3.1 Product Guide." Accessed July 1, 2021. https://www.xilinx.com/support/documentation/ip_documentation/dpu/v3_1/pg338-dpu.pdf.

———. 2020a. "7 Series DSP48E1 Slice User Guide." Accessed March 30. https://www.xilinx.com/support/documentation/user_guides/ug479_7Series_DSP48E1.pdf.

———. 2020b. "AXI Reference Guide." Accessed March 25. https://www.xilinx.com/support/documentation/ip_documentation/ug761_axi_reference_guide.pdf.

———. 2020c. "Device Reliability Report." Accessed March 30. https://www.xilinx.com/support/documentation/user%5C_guides/ug116.pdf.

Xilinx. 2020d. “Partial Reconfiguration User Guide.” Accessed March 30. https://www.xilinx.com/support/documentation/sw_manuals/xilinx13_3/ug702.pdf.

———. 2020e. “Soft Error Mitigation Controller v4.1 LogiCORE IP Product Guide.” Accessed March 30. https://www.xilinx.com/support/documentation/ip_documentation/sem/v4_1/pg036_sem.pdf.

———. 2020f. “Zynq UltraScale+ MPSoC Data Sheet: Overview.” Accessed March 30. https://www.xilinx.com/support/documentation/data_sheets/ds891-zynq-ultrascale-plus-overview.pdf.

———. 2020g. “Zynq-7000 SoC Data Sheet: Overview.” Accessed March 30. https://www.xilinx.com/support/documentation/data_sheets/ds190-Zynq-7000-Overview.pdf.

———. 2021. “Fast Fourier Transform v9.1.” Accessed July 1, 2021. https://www.xilinx.com/support/documentation/ip_documentation/xfft/v9_1/pg109-xfft.pdf.

Zhang, M., H. Li, G. Xia, W. Zhao, S. Ren, and C. Wang. 2018. “Research on the Application of Deep Learning Target Detection of Engineering Vehicles in the Patrol and Inspection for Military Optical Cable Lines by UAV.” In *2018 11th International Symposium on Computational Intelligence and Design (ISCID),* 01:97–101. December.

Zhang, R., L. Xiao, J. Li, X. Cao, and L. Li. 2020. “An Adjustable and Fast Error Repair Scrubbing Method Based on Xilinx Essential Bits Technology for SRAM-Based FPGA.” *IEEE Transactions on Reliability* 69 (2): 430–439. doi:10.1109/TR.2019.2896897.