POLITECNICO DI TORINO

Master's Degree in Embedded Computing Systems

Master's Degree Thesis

Study and Development of Neural Network Architectures on Rad-Hard FPGAs

Supervisors

Prof. Luca STERPONE

Candidate

Mina Hauge NØSTVEDT

October 2023

Summary

This thesis investigates the use of radiation-hardened FPGAs for neural network implementation in radiation-prone environments. For this purpose, the European-based NG-Medium board is examined and compared to the non-radiation-hardened Pynq-Z2. A UART communication system and a neural network for color classification were designed to function on both boards without significant alterations to ensure equal compatibility for a fair comparison.

The difference between the utilization and performance of the NG-Medium and the Pynq-Z2 FPGA is interesting. The Pynq-Z2 demands more resources for code execution, whereas the NG-Medium struggles with proper code functioning, possibly contributing to its seemingly lower consumption. The Pynq-Z2 also performs better in speed due to the faster clock frequency. The classifier works well with Pynq-Z2 but does not function properly on the NG-Medium. The cause of the improper functionality is still unknown.

Integrating neural networks on radiation-hardened FPGAs is a complex task. While the NG-Medium's feasibility for neural networks remains unproven, this study can still be helpful due to its insights into the utilization and architecture between the two boards. The thesis helps bring forward the potential benefits of this combination despite being unable to validate neural networks on the NG-Medium.

While working on this project, new challenges and insights have come forward, which can help future researchers studying this topic. The use of the neural network on radiation-hardened FPGAs needs further investigation, exploring different neural network algorithms and training methods, addressing code issues on the NG-Medium, and performing radiation testing on the entire system. While the study could not determine if the NG-Medium is a good candidate for neural network implementation on board spacecraft, the study lays the groundwork for advancing intelligent systems in the future.

In conclusion, neural network implementation on radiation-hardened FPGAs requires more research. However, despite the challenges and lack of proper results in this thesis, the research sets the stage for further exploring this topic of deep learning for space application.

Acknowledgements

I would like to express gratitude to my supervisor, Professor Luca Sterpone, for trusting me with this thesis, even if my prior knowledge of the topic was limited. It has been a great learning experience for me. I am also grateful for the help and guidance that Andrea Portaluri has provided while working on this thesis. His insight and knowledge of the topic have been really valuable. Finally, I would like to thank my family, friends, and boyfriend for their support and encouragement throughout my whole academic journey.

Table of Contents

List of Tables

List of Figures

Acronyms

AI
Artificial Intelligence

BRAM
Block RAM

CLB
Configurable Logic Block

DFF
Delay Flip-Flop

DSP
Digital Signal Processing

FF
Flip-Flop

FPGA
Field Programmable Gate Array

HDL
Hardware Description Language

HPC
High-Performance Computing

IOB
Input/Output Block

LUT
Look-Up Table

MSE
Mean Square Error

NN
Neural Networks

Rad-hard
Radiation-hardened

RAM
Random Access Memory

RGB
Red, Green, Blue

RHBD
Radiation Hardened by Design

ROM
Read Only Memory

SEE
Single-Event Effects

SET
Single-Event Transient

SEU
Single Event Upset

SoC
System-On-Chip

TID
Total Ionizing Dose

UART
Universal Asynchronous Receiver/Transmitter

Chapter 1

Introduction

Any electrical system operating in space faces the constant threat of radiation-induced errors that can compromise the system's functionality and reliability. The choice of electrical components plays a crucial role, necessitating the exploration of radiation-hardened solutions that can withstand harsh conditions in space. This thesis delves into radiation-hardened Field-Programmable Gate Arrays (FPGAs) and Neural Networks (NN) and their role in paving the way for resilient and reliable artificial intelligence (AI) in radiation-intensive environments.

In recent years, rapid advancements in technology have led to increasing demand for sophisticated computational systems, artificial intelligence, and machine learning. Neural networks have revolutionized various fields, from image processing to computer vision, demonstrating incredible capabilities. However, neural network implementation on standard electrical components poses major challenges for radiation-prone environments like space. Neural networks often require high performance and processing power, making FPGAs a good hardware choice. FPGAs are widely used in digital circuit design due to their highly flexible and customizable hardware. Global companies have already deployed FPGAs for AI and machine learning to implement automatic spoken language recognition and autonomous driving. However, the use of neural networks and AI in space is a new thing. Unfortunately, standard FPGAs are not an exception from radiation-induced error, and the performance and fault tolerance may be compromised in radiation-prone environments. This challenge has motivated and driven the exploration of radiation-hardened FPGAs specially designed to mitigate the radiation effects. A limited number of FPGAs are currently qualified for space applications, even though they could prove very usable, especially for earth observation. Now, data is sent from satellites to Earth to be analyzed. However, it could be faster to have a neural network classifier on board for cloud detection and detection of oil spills, wildfires, tornadoes, and similar.

NanoXplore is a semiconductor company that offers a solution to this problem

by providing radiation-hardened FPGAs. Notably, the NG-Medium FPGA stands out as it has been radiation-hardened by design in configuration memories and registers. Moreover, it is also the first European FPGA to be qualified for European space missions, paving the future of European space missions.

As the use of neural networks on space-grade FPGAs has not been widely reported, the goal of this research is to gain an understanding of how rad-hard (radiation-hardened) FPGAs function with neural networks. The thesis will try to determine whether or not it is feasible to use a space-grade FPGA for a small neural network to detect colors. The performance, timing, and utilization are investigated and compared to a similar non-rad-hard FPGA. The question is: What are the performance trade-offs, such as power consumption, area utilization, and latency, between a radiation-hardened FPGA and a non-radiation-hardened FPGA when implementing neural networks in radiation-intensive environments?

This thesis will investigate how the NG-Medium compares to standard non-radiation hardened FPGAs regarding performance and utilization. Implementing the same neural network in both systems aims to understand the practical implications and potential advantages radiation-hardened solutions can pose in radiation-intensive environments.

The findings in this research could provide a deeper understanding of the field of neural networks for space exploration. By highlighting how the two designs differ in terms of performance, reliability, and fault tolerance, this thesis may provide valuable insight for engineers and researchers working in the field.

This thesis aims to contextualize the need for neural networks and radiation-hardened FPGAs in space activities. Firstly, a system is developed to communicate with the NG-Medium board and a simple neural network to classify colors from pixel input. The goal was to achieve a high-performance computing (HPC) design on hardware (not as a software model) to have the possibility of extending the research in the future by conducting radiation testing and seeing how the board reacts to communication and classification. The design was tested on two boards, the NG-Medium and a non-rad-hard FPGA, Pynq-Z2. The two boards have been tested individually to compare the results, and while they have similar architecture, the clock frequency differs, making some comparisons harder to conduct.

In the following chapters, you will find information regarding fundamental theory, empirical research, and the methodology used for the study. I will also present the design and the experimental setup, as well as the results and findings. The goal of this thesis is to shed light on the neural network implementation of radiation-hardened FPGAs, their advantages, and how they might be the gateway to resilient artificial intelligence systems in space.

Chapter 2

Related work

This chapter serves to establish a theoretical background of FPGAs and neural networks, along with their advantages and limitations in spacecraft operations.

A roadmap for the chapter is illustrated in Figure 2.1. Firstly, the fundamentals of neural networks are introduced, providing an overview of their structure, functionality, and components. Secondly, the architecture and design process of FPGAs are explained, to enable readers to understand the application of neural networks on FPGAs, and the existing literature in this field.

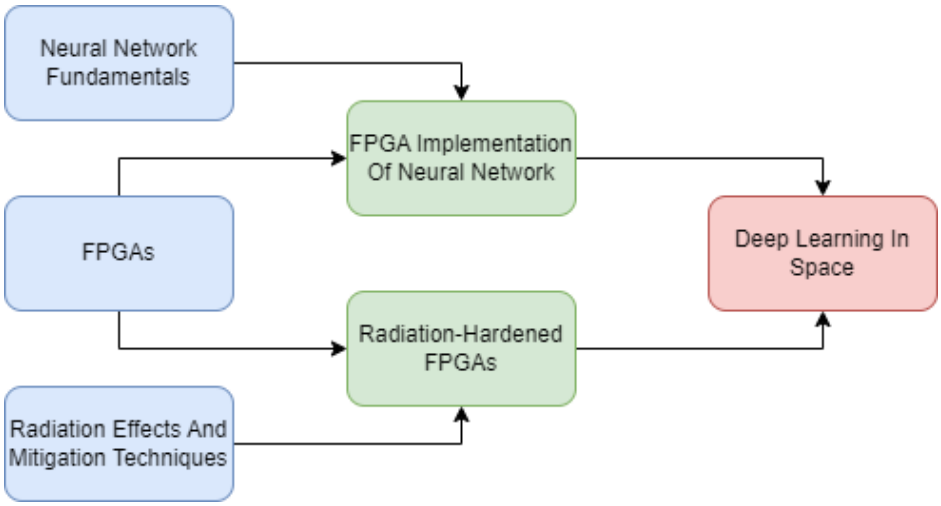

Figure 2.1: Theoretical Roadmap. This roadmap shows the flow of the fundamental topics covered in this section, and how the different sections build on each other.

A section on radiation effects and mitigation techniques is present to build the knowledge needed to dive into the existing research on radiation-hardened FPGAs.

Each section helps to build the fundamental knowledge required to understand the current research on deep learning in space and the utilization of neural networks in this field. By presenting the existing research and knowledge in this field, the intention of this chapter is to identify and highlight the areas where current research falls behind and where this thesis might contribute.

2.1 Neural Networks Fundamentals

The human brain contains billions of neurons [1], which communicate using electrical signals in the form of impulses or "spikes". The neurons are interconnected and are constantly receiving multiple input signals. All signals coming into the neuron are processed in some way and if the resulting signal exceeds a threshold, the neuron will fire or generate a voltage impulse, essentially sending information to another neuron. These connections are mediated by electrochemical junctions called synapses. This structure and functionality of an animal brain are the bases of artificial equivalents of biological neurons, or artificial neuron networks. They mimic the connections of neurons and synopsis inside the brain in order to create robust models to analyze and predict outcomes from large amounts of data.

Neural networks are helpful for intelligent systems in order to map, model, and recognize relationships in large amounts of data fast. These networks are used in various applications, like pattern and speech recognition, language processing, and image classification.

2.1.1 Structure

Each neuron cell can be regarded as a nonlinear transformation unit made up of weights multiplied with input data from the previous layer. Weights are defined during the training of the network. As shown in Figure 2.2, each weighted input signal is summed up together with a constant input, called a bias, and the sum is given to an activation function. Here, the sum is checked with a threshold, and if the activation exceeds the threshold, the neuron "fires" by producing a high output; otherwise, the output is low.

The entire network consists of interconnected neurons arranged in different layers, with one input layer, one or more hidden layers, and an output layer, as demonstrated in Figure 2.3. The input layer collects the data, while the output layer provides the classification results. The hidden layer captures the information from the input and transforms the information into a higher-dimensional representation. This way, the system is able to learn very complex relationships and detect patterns.

The network may vary in structure and size depending on the system's needs, and the number of neurons in each layer varies depending on the need. The input layer structure is typically determined by the dimensions of the input data, where

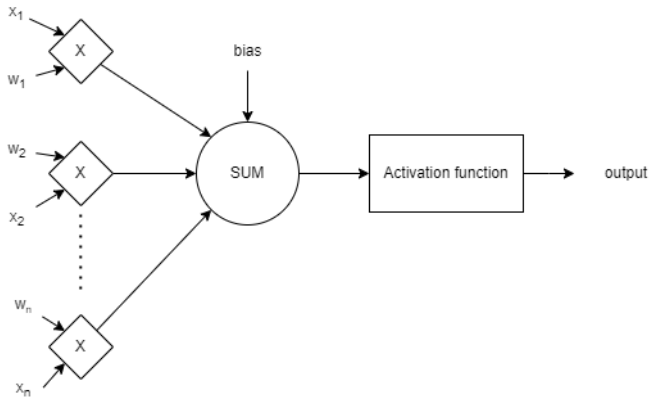

Figure 2.2: The Architecture of an Artificial Neuron. This diagram illustrates the building blocks of an artificial neuron and how inputs are combined, weighted, and added with a bias before it is passed through an activation function to produce an output.

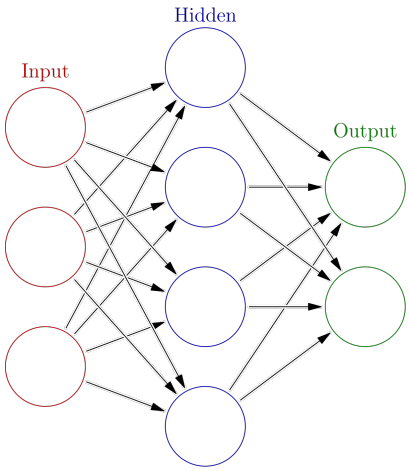

Figure 2.3: Neural Network Structure. Demonstrates the interconnected layers of a neural network, where input data flows through one hidden layer to the final output layer. In this example three input nodes, four hidden nodes, and two output nodes are present.

each neuron represents one attribute of the input data. The number of output nodes depends on the problem one hopes to solve. This could be the number of classes in a classification or only one node for binary solutions. Determining the number of hidden nodes is a bit more complex, and there is no correct answer. There are multiple rule-of-thumb methods for determining the correct number:

- The number of hidden nodes should be between the number of input nodes and output nodes.
- The number of hidden nodes should be 2/3 of the input nodes plus the number of output nodes.
- The number of hidden nodes should be twice the size of the input layer.

However, the number of hidden nodes is often determined by experimentation and tuning. Adding more hidden layers increases complicity, letting the network learn more complex patterns. However, it can also increase the risk of overfitting. It is ultimately the functionality, complexity, and accuracy of the network that determines how many nodes are needed.

Neurons can be arranged in many different structures, but there are two main categories: feed-forward structure, and recurrent structure [2]. The feed-forward structure is the most used one and is recognized by having no feedback from the output to the network's inputs. If, however, there are connections from the output of the network to the input, it is known as a recurrent neural network. In a feed-forward structure, each signal on the input will pass through the layers to the output. This structure is often used to detect patterns or for categorization. By allocating one output node per class, the node corresponding to a particular class will fire whenever the correct pattern is supplied on the input.

2.1.2 Training

The network itself is only able to process information through the weights. Training of the weights allows the network to detect patterns and relationships between data, to make accurate decisions.

Neural network training can be roughly divided into two categories; supervised and non-supervised learning. Supervised learning is a learning model where the network is trained using labeled data. This means that the input data is given to the network together with the correct label to let the network create a model and learn how to map the data correctly. Contrary to this, unsupervised learning is not provided with any labels, and the goal of this algorithm is that the network develops classification labels[3]. Unsupervised learning is valuable when searching for patterns or finding correlations in large data sets.

Under the umbrella of supervised learning, there are many different learning algorithms. One of the most used ones is the back-propagation algorithm, often used for feed-forward neural networks. In essence, it works by computing the gradient of the loss function, in this case, the mean square error (MSE), and gradually adjusting the weights and biases to lower the error or loss through the layers. Since the back-propagation algorithm requires labels, it is considered a supervised learning algorithm.

The step size is named the "learning rate", and it is a scalar that determines how much the weights and bias change in each iteration, and therefore how quickly the network learns. The back-propagation algorithm can run until it reaches a minimum error set by the user or for a certain amount of learning rounds, called epochs.

MSE-based training aims to train the total weights matrix W [4], and the MSE is calculated as shown in equation 2.1.

$$MSE = \frac{1}{2}\sum_{k=1}^{N}(g_k - t_k)^T(g_k - t_k) \tag{2.1}$$

where g_k is the predicted value, while t_k is the target value. For a linear classifier g is defined as the matrix multiplication between the input data x and the weights matrix W, and adding a bias. For non-linear classification, the result goes through the activation function f, as seen in equation 2.2.

$$g = f(Wx + bias) \tag{2.2}$$

In the next learning round, the gradient is subtracted from the weights matrix W by a step factor α, as shown in equation 2.3 in order to reduce the loss.

$$W(m) = W(m-1) - \alpha\Delta_W MSE \tag{2.3}$$

where m is the epochs and α is the learning rate that determines how fast the solution is obtained.

2.2 Field-Programmable Gate Arrays (FPGAs)

Field programmable gate arrays are semiconductor devices comprising integrated circuits that provide configurable and re-configurable hardware solutions for various digital logic functions. FPGAs have gained wide popularity for their numerous advantages, some of which include:

- Flexibility and adaptability
- High performance
- Parallelism
- Programmability
- Cost Efficiency

FPGAs are highly flexible and customizable, making them a good choice for various applications. The user can create custom hardware for specific tasks, allowing for optimized performance, low latency, and energy-efficient computations. Additionally, it is a strong competitor to CPUs and GPUs, due to the lack of a fixed instruction set and memory hierarchy, making them highly adaptable and offering a fully customizable architecture. This benefit helps the FPGA break through the performance limitations typically encountered in CPUs and GPUs.

Parallelism is a key feature in the FPGA architecture, enabling parallel computations in hardware, which has led to its popularity in various fields. These features make the FPGAs excel in implementing and optimizing algorithms, delivering high performance, and enabling real-time processing.

The programmability of FPGAs means that even after a circuit has been implemented, the FPGA can still be modified and updated. This feature means that designer can adapt their design by updating and improving it as they go, without requiring complete hardware replacement. Ultimately, this feature reduces both the effort and cost required for long-term use and maintenance of the chips and makes the FPGA extremely cost-efficient.

2.2.1 Architecture

The flexibility of the FPGA is due to the configurable logic blocks (CLBs) [5] it contains, which provide basic logic and storage capabilities.

CLBs are the fundamental building blocks of the FPGA. They consist of essential logic elements like look-up tables (LUTs), flip-flops (FF), or delay flip-flops (DFF), and other elements allowing logic implementation and data storage. The number and arrangement of the components inside a CLB varies by device. The interconnected structure enables signaling between the CLBs and other components, ultimately enabling interaction between the internal architecture of FPGA and the outside peripherals. While CLBs and interconnects care for the internal signals within the FPGA, Input/Output Blocks (IOBs) serve as an interface between the internal logic of the FPGA and external devices. FPGAs can be categorized into three categories based on the programming technology used.

- SRAM-based FPGA
- Antifuse-based FPGA
- Flash-based FPGA

SRAM-based FPGAs are the most common type, and it is the kind that is used when talking about re-programmability. SRAM-based FPGAs have a small static random access memory (RAM) bit for each programming element, making it re-programmable multiple times. The disadvantage is that they are also volatile,

meaning that power glitches can corrupt the device as they require continuous power to keep the code. This thesis will focus on SRAM-based FPGAs; therefore, antifuse-based and flash-based FPGAs will not be discussed or explained.

FPGAs typically contain multiple clock domains, and the design of the clock networks is important [6]. If the clock network cannot supply the required number of clock signals, some user circuits may be inefficient or impossible to implement. The FPGA needs to have a clocking network that distributes the clock signals to all components and manages the synchronous operations inside the chip. At the same time, it is essential to keep the balance between flexibility, speed, area, and power.

2.2.2 Programming and Design Flow

FPGAs contain no processor to run software and rely on user-designed circuits. The design is written using hardware description languages (HDL), like Verilog or VHDL, and is then synthesized into a bit file to configure the FPGA. In SRAM FPGAs, the configuration file is stored in the RAM, making it volatile.

Hardware description language is used to express the FPGA's structural, behavioral, and register-transfer-level architectures, meaning it can write executable specifications for the chip. HDL is also an excellent tool for creating test benches to simulate the circuit behavior. Designers can describe the logic and signal flow, specify the timing, and define the routing of signals to different components within the FPGA, letting them capture the desired functionality of the circuit and interconnects. The ability to simulate the progression of time allows the designer to verify the correctness and performance of the circuit before a physical implementation takes place.

There are multiple steps that one needs to go through before the design is physically implemented on the FPGA. The first step is synthesis. The synthesis converts the HDL into a netlist representation of the logic functions and interconnects in the circuit. During the synthesis, the design is also mapped onto the available FPGA resources, automatically or predefined by the designer. Placement is the next step of the implementation, where the goal is to determine the physical locations of the elements within the FPGA device. This involves mapping the synthesized elements into specific locations, minimizing critical paths, optimizing performance, and ensuring that resources are used efficiently. The last step before loading the design onto the FPGA is routing. This step determines the interconnections between the placed logic elements within the FPGA, in order to determine how signals should flow. After these steps, the complete bitstream can be loaded onto the FPGA.

FPGAs are powerful platforms for custom design, with HDL playing a vital role in expressing circuit behavior and structure. The FPGA's reconfigurable nature enables the implementation of various circuits, even though the implementation

process might be a bit tedious.

2.3 FPGA Implementations of Neural Networks

FPGAs are a good choice for implementing neural networks due to their high computing power, parallel architecture, low power consumption, and custom algorithm development. The parallel architecture in the FPGA allows for efficient parallel computation and is well suited by preserving the parallel architecture of the neurons in a layer. The high computing power of the FPGA can secure accurate results. Furthermore, the ability to implement the network directly on the hardware can also drastically increase the computations' speed, compared to implementation on software. With the flexibility of the FPGA, custom algorithms can be designed and optimized to fit the needs of the neural network.

One study focused on the flexibility of implementing neural networks on FPGA [7]. They found that even though the neural network classification reaches a slightly lower accuracy than an implementation in an ASIC circuit, the FPGA is still favorable. The reasoning for the conclusion is that the re-configurable FPGA allows for much greater flexibility compared to the ASIC circuit. This ultimately means that it allows for changing topology without the need for changing hardware, allowing for improved test results in the long run.

In the paper "Reconfigurable FPGA implementation of neural networks", Zbigniew Hajduk showed that the FPGA implementation of a neural network could be characterized as highly flexible[8]. It also has high calculation speed and reasonable accuracy compared to software realization of neural networks. Some tests revealed that even the most straightforward resource-saving implementations on FPGA carry out the calculations faster than the RAM professor clocked with much higher frequency.

A neural network designed directly on hardware, like on an FPGA, has multiple advantages compared to when designed in software. The first advantage is speed. For hardware devices that offer parallel computation, a significant increase in speed can be gained compared to software-based solutions. Hardware implementation can also reduce the total cost of the system[9], by lowering the total component count and decreasing power requirements.

One limitation of any uni processor is the vulnerability to stop functioning due to faults in the system, which is often due to a lack of proper redundancy in the processor architecture. One study [10] suggests that even with the advancement of multi-core PC processor architecture, efficient fault tolerance mechanisms are needed. On the other hand, parallel and distributed architectures allow the application to continue functioning, although a slight reduction in performance may be present. This performance loss, called graceful degradation, will let the system

continue functioning, even if faults are present in some components. For neural network applications, fault tolerance is extremely important as they often require complete availability and can be safety-critical. Parallel hardware applications can, therefore, offer a considerable advantage.

To summarize, neural network implementation on FPGAs can offer great flexibility when fine-tuning and testing the system due to the reconfigurability of FPGAs, while still offering high computing power and speed. The FPGA design also offers higher computing speed compared to design in software, due to the parallel nature of the FPGA. While some limitations are also present, like lower accuracy, it is still favorable due to the graceful degradation and high speed it offers.

2.4 Radiation Effects and Mitigation Techniques

Unlike most applications on Earth's surface, space applications are subject to high ionizing radiation. High levels of radiation might damage or upset the operation of electrical devices, which can lead to short and long-term errors and even fully dysfunctional devices. Luckily, several different radiation mitigating techniques can be used to lower the chances of unfortunate effects.

2.4.1 Effects on Electric Components

Radiation can have both long- and short-term effects on electronic components and can be categorized into single-event effects or cumulative effects [11]. This chapter will discuss the different types of radiation effects, and Figure 2.4 shows how they are connected.

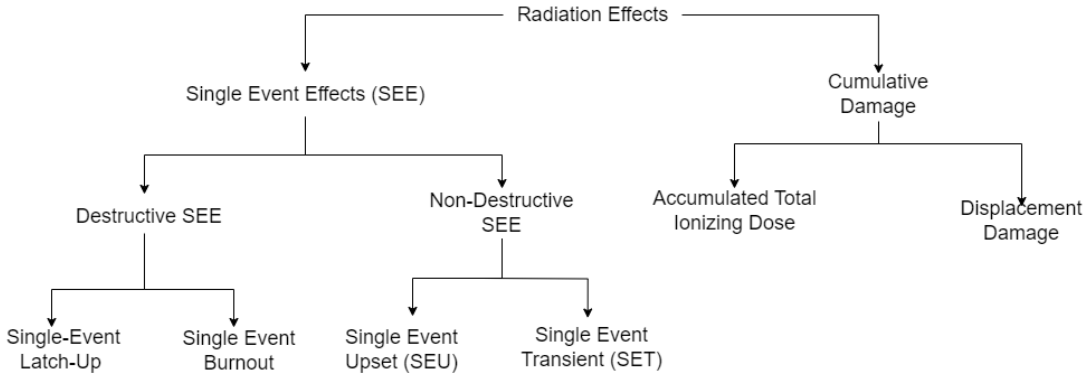

Figure 2.4: Radiation Effects Hierarchy. This schematic illustrates how the different types of radiation effects are integrated and to which subcategory they belong.

Single-event effects (SEE) include single-event latch-up and single-event burnout, as well as non-destructive single events that change the state of a digital memory element without causing permanent damage to component [12]. SEEs occur through the passage of a single charged particle through the device, and a charged particle will ionize the atoms it hits on its way and free up both electrons and holes as charged carriers. When this particle passes through the device, it can cause a spike of charge that impacts the system in multiple ways.

Single event latch-up occurs when a transistor gets set to a permanent high or low state [13]. What happens is that a high-energized particle can create electron-hole pairs in the semiconductor material, which can cause a low-impedance path between the supply rails. This can result in a drastic increase in current flow to the limit of the supply or the destruction of the device. Once this error has occurred, a power cycle might be able to recover normal operation, but there may still be latent damage.

Single-event burnout results from charge collection in a high electric field region. What happens is that the charged particle can initiate a second breakdown, which can create a chain event of charge multiplication. The result is typically complete failure of the device, caused by thermal runaway that causes local melding or ejection of molten material, ultimately leading to a small-scale explosion.

Single-event effects that do not result in device failure are called non-destructive single events. The most common are bit-flips or single event upset (SEU) in a memory cell. What essentially happens is that when a charged particle hits a cross-coupled transistor, the value of the memory cell will flip to the opposite value as before. This results in a bit-change in the stored memory, but otherwise, the circuit continues to function as expected.

The charge collected in a node can also appear as an analog signal, a voltage spike, known as a single event transient (SET). Spikes may damage devices with sensitive inputs, and if the signal passes through the combinational logic, it can create a digital upset if it latches into a register or clock edge.

Accumulated total ionizing dose (TID) is a radiation effect that affects metal-on-silicon transistors. When a positive charge is applied to the gate, the potential barrier is lowered, allowing electrons to tunnel from the source to the drain. If ionizing radiation passes through the device, electrons are liberated from the atom, creating holes of missing charge that act as positive carriers. These holes can eventually become trapped in the gate oxide, and the accumulation of trapped charge is known as the total ionizing dose. The effects of TID can lead to the transistor being stuck in a constant state.

Displacement damage effects are another form of cumulative damage that can arise from large doses of radiation. This happens when a particle, typically a proton or neuron, knocks an atom out of its position in a crystal, creating a void. This leads to a change in the electronic band structure of the material, hindering the

free flow of charge across the device. The result of this can be a decrease in the circuit's performance.

As seen, radiation effects on electronic components range from single-event effects to long-term cumulative damage. There is also a distinction between destructive and non-destructive damage, which indicates whether or not the effect causes permanent damage to the device. Understanding these effects is crucial when creating strategies to mitigate the radiation effects.

2.4.2 Radiation Mitigation Techniques

To make electronics more resilient to the various radiation effects that can take place in space, radiation mitigation techniques can be implemented. Techniques such as shielding, redundancy, error detection and correction, and radiation hardening by design can help to minimize or mitigate radiation effects in order to increase performance, reliability, and the lifespan of the devices. This makes them more suitable for applications in space missions and other radiation-prone environments.

Redundancy comes in different forms, but two main categories are logical redundancy and physical redundancy. Logical redundancy involves using redundant logic elements to provide alternative paths for performing computations or storing data. This involves error detection and correction codes and triple modular redundancy. Error detection and correction codes are logical techniques that involve implementing codes to detect and correct radiation-induced errors in data or instructions. Triple modular redundancy, on the other hand, involves employing triplicated logic structures and comparing their outputs to detect and correct errors caused by radiation [11].

Physical redundancy modifies the physical characteristics and structure of electronic components in order to enhance their resistance to radiation-induced effects [14]. One technique is to duplicate physical components, like transistors or memory cells, to have redundant resources. The duplicated resources can work in parallel or be used as a backup if an error is detected. Shielding is another physical radiation hardening technique where the device is shielded to reduce exposure to radiation. This is performed using materials with high atomic numbers to block radiation and prevent it from reaching sensitive components in the system.

Radiation-hardened by design (RHBD) is an approach in which electronic components are designed and optimized to withstand radiation-induced effects. This involves incorporating different design techniques, logical and/or physical, that enhance the device's resilience to radiation.

Radiation mitigation techniques play a crucial role in enhancing the resilience of electronic devices to radiation effects in space and mitigate the effects. These radiation mitigation techniques are examples of making more robust and reliable electronic systems for high-radiation environments.

2.5 Radiation-Hardened FPGAs

Increased interest in using FPGAs in space has led to a need for sustainable and commercially available FPGAs for radiation environments [15]. Radiation effects, together with satellite lifespans increasing far beyond the validity of telecom standards, programmability and the need for radiation-hardened devices in flight become a stringent requirement. Where software solutions are not possible, FPGAs may be the only solution, and a rapid increase in the use of FPGAs in space applications is expected.

FPGAs have been used in space for over a decade, with mixed results because of the challenging environment above Earth's atmosphere. Only a few devices have been used on European spacecraft due to the sensitivity to involuntary reconfiguration induced by radiation. The number of FPGAs in space is low; even less for European chips. NanoXplore is a European-based company that crafts state-of-the-art radiation-hardened components for FPGA devices to be used for aerospace missions. The NanoXplore NG-Medium FPGA is an SRAM-based 64nm RHBD chip using high-performance block RAMs (BRAMs) and DSP blocks. It is the first entirely European FPGA to be approved for space missions.

2.6 Deep Learning in Space

Machine learning and AI have been used to analyze space data for decades. However, mobile deep learning directly on space crafts is a new field. With the rapid development in both space devices and the access to configurable devices like FPGAs, deep learning in space could be the way to save energy, make quick decisions, and increase automation and control of spacecraft.

Satellite data has traditionally been downloaded to a ground station before machine learning or manual data processing occurs. Sending and receiving large volumes of data daily is power-consuming and requires large data storage solutions. [16] suggests that using deep learning directly on the spacecraft might save at least half the power by restricting which data is transferred to Earth.

Transmitting image data to ground stations can be very expensive, and by employing deep learning on board, a satellite can reduce the amount of data transmitted. Pre-processing on board the spacecraft can discard images of no interest, for example, images of poor quality or distorted view due to sun reflection or cloud convergence. [17] estimated that the global annual cloud coverage is about 66%, meaning that if cloud observation is not the mission large amounts of image data might be unwanted. By using different image processing techniques, images can be classified and discarded. By only transmitting the regions of interest, it is possible to save power and cost.

Deep learning and machine learning-based data-driven techniques have a lot of potential and can be used in various areas, from ice monitoring to discovering new objects in deep space. [18] suggests that using image classification in space, it is possible to categorize ice and ice development with over 80% accuracy. The article focuses on the monitoring of ice on the planet in terms of using image processing to categorize the ice and monitor the change of the ice. Old image data was used for training and testing the algorithms. While all classification was performed on non-rad-hard FPGAs, on-board classification could be a possibility for the future of ice monitoring. Similar techniques have also been tested for cloud detection [19], greenhouse gas concentration estimations [20], and monitoring oil spills. Oil spill monitoring can prove especially helpful because it could be possible to predict a spill's spread direction and flow rate. This could help the cleanup crew and be used for monitoring the after-effects of the accident by monitoring shoreline characteristics and severity.

Identifying objects in thousands of astronomical images is highly complex and time-consuming. Here, neural networks can be used to help astronomers to detect and classify phenomena of interest. The James Webb Space Telescope is one of the first to use deep learning in data post-processing to detect galaxy clusters [21]. Onboard classification and learning could mean continuous searches, but for now, it is only possible to use this technique post-data transfer.

Automation and control are other aspects of space flight that might benefit from different machine-learning techniques. Automation is currently used in satellites such as Rosetta and Venus [22], but with deep learning, the onboard control can be improved by event detection and subsequent planning. When autonomous control was first introduced, it showed that the autonomic control system was highly reliable, even during extreme geophysical events [23]. Introducing machine learning and artificial intelligence to spacecraft control would allow for great flexibility, reduce lead time, and reduce risk for mission operations, as the spacecraft would be able to automatically avoid crashes and incidents by giving it a better view of the surroundings. [24] suggests that rovers can benefit from onboard deep learning for control and path planning in uneven terrains. While this has not been implemented yet, multiple test systems and algorithms are showing promising results to let a neural network judge the navigability of the terrain to plan a safe path for the vehicle.

In summary, using deep learning and AI directly in space can save energy, offer fast decision-making, and increase automation and control in spacecraft operations. Continued research and implementation of these techniques can pave the way for advancement in space exploration, opening up new possibilities and improving our understanding of the universe.

2.7 Research Gap and Weaknesses

While neural network implementation and radiation-hardened electronics are widely researched, we have barely scratched the surface of what is possible with deep learning in space. Radiation-hardened FPGAs have demonstrated remarkable reliability and flexibility, offering the advantages of being configurable and reconfigurable for system updates and improvements. Neural networks and artificial intelligence in space also have many benefits, including power and storage savings and increased control system flexibility. However, there is a notable lack of research on using FPGAs and neural networks onboard space crafts. More research is needed to fully understand and address the challenges of using neural networks on radiation-hardened FPGAs in space environments and unlock their potential.

This chapter introduces the fundamentals of neural networks and FPGAs, explaining the architecture and functionality behind these. Additionally, radiation effects and mitigation techniques were presented briefly, as well as current research on all topics. The findings show that while research on neural networks and FPGAs is widely available, it is lacking in terms of FPGA implementation of neural networks and the use of radiation-hardened FPGAs. This highlights the need for further research on neural networks on FPGAs for space application and deep learning.

Chapter 3

Proposed methodology

In this chapter, we present the methodology adapted to design and implement the system, consisting of a universal asynchronous receiver/transmitter (UART) based communication system and a neural network for color classification. The chapter reviews the steps taken to design and collect data from the system, including the development and limitations of each. The designed UART and the overall communication system within and outside the board are presented. Additionally, the design and optimization of the neural network is reviewed. By detailing the methodology, the aim is to provide a clear understanding of the process and techniques used to create the design and collect data for this project.

To explore how a rad-hard FPGA responds to neural networks, a quantitative approach was used. There is not yet a lot of research available on neural networks for rad-hard FPGAs, as this is a new field of research and there are limited boards commercially available. To gain insight into how a rad-hard FPGA compares to a non-hard-hard FPGA, the design was tested on two boards; The rad-hard NG-Medium, and the non-rad-hard Pynq-Z2. By implementing the same neural network for color detection on both boards, one can evaluate the differences between them, in terms of performance and utilization, to determine if neural network implementation on the NG-Medium is suitable.

3.1 The Design

The overall design comprises three main hardware components, a personal computer, a UART to USB adapter, and an FPGA board containing the necessary inputs and outputs. Figure 3.1 shows the high-level architecture of the system design, as well as the communication between the different hardware. The figure also illustrates the internal processes of the FPGA, including the UART and the neural network. For each classification, the neural network required one set of pixel values, consisting of

red, green, and blue 8-bit values. When the FPGA receives the data, it is decoded and manipulated, before the values are put on the neural network input. The aim of the neural network is simple; get a pixel input, and try to determine if the pixel's color is yellow, blue, or neither. The output is then manipulated back into ASCII characters before transmitting them to the computer by the transmitter.

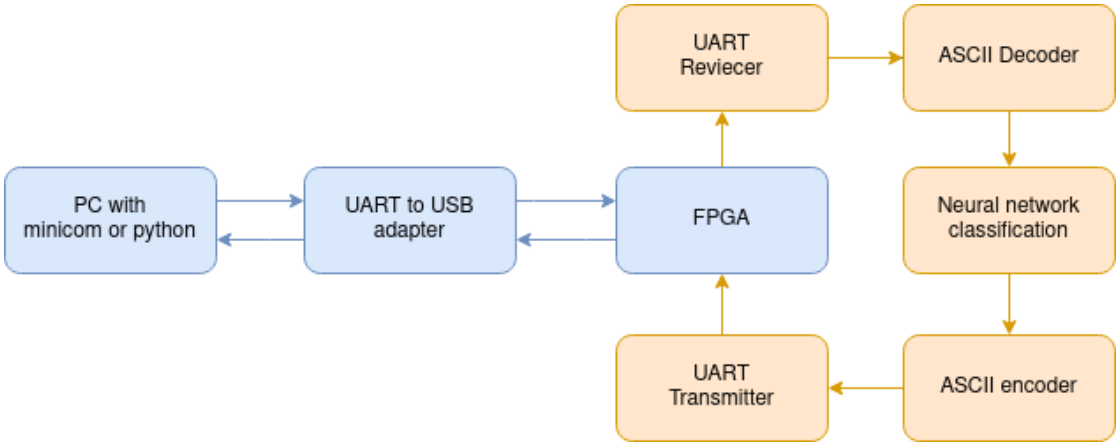

Figure 3.1: System Communication Cycle. Illustrating the internal processes inside the FPGA and the communication from the computer to the board. Internal processes inside the FPGA are marked with red.

UART

The UART module is implemented to serve as the communication interface between the FPGA and an external computer. The implementation enables the system to receive input data from the computer and send back the results efficiently. The UART is configured to operate at a baud rate of 115200 bits per second, and carries eight data bits, making it suitable to send and receive ASCII characters. Additionally, the UART uses one start bit and one stop bit during the communication process.

When an external computer intends to send data to the FPGA, it sends a start bit, followed by the 8-bit data packet, using a serial communication interface. The 8-bit packet has to be ASCII characters representing numbers between 0-9, or a space. On the FPGA side, the UART module continuously monitors the line for incoming data. The UART will then validate the received data, and wait for the stop bit before it extracts relevant information from the data pack for further processing within the FPGA. Once a stop bit is received, the data is echoed back to the computer to verify the correctness.

Figure 3.2 shows the connections between the personal computer and the board

when communicating with the FPGA. The computer is connected via USB to a USB to UART adapter. This adapter has the receiver and transmitter connected, as well as a common ground with the board. The device is also powered by 5V from the board.

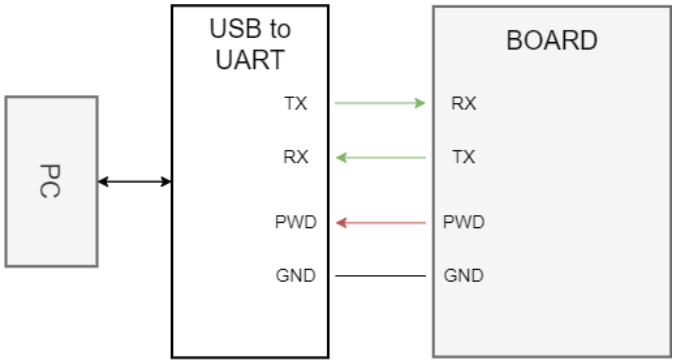

Figure 3.2: Hardware Connections. Illustrating the hardware connections between the PC, the UART-to-USB adapter, and the board.

Neural Network

The neural network is implemented to classify received pixels as blue, yellow, or other. It is based on a feed-forward structure that consists of three layers: one input layer, one hidden layer, and an output layer. Figure 3.3 shows the organization of the layers for the neural network. The NG-Medium has a relatively limited number of bit carriers, and a smaller network is preferred. The network has three input nodes, one for each color of the RGB (red, green, blue) pixel values, seven hidden nodes, and two output nodes for classifying yellow and blue.

3.1.1 UART Implementation

The UART controller consists of a top-level, a receive module, and a transmit module. The receiver takes in the raw serial data, reads it, and returns the data packet to the top level as well as a signal indicating that a byte was received successfully. The transmit module ensures the data is transmitted correctly depending on the baud rate and will send the start bit, data, and stop bit correctly. Additionally, it ensures the line is clear before sending data on the line. The top-level guarantees a correct data flow in the system, and Figure 3.4 shows how the data is manipulated going through the system. The ASCII data bytes from the receiver are first converted to binary values and saved in a temporary array.

Each pixel on the computer is deconstructed into its three separate components: red, green, and blue. When transmitting the data via the UART, the RGB values

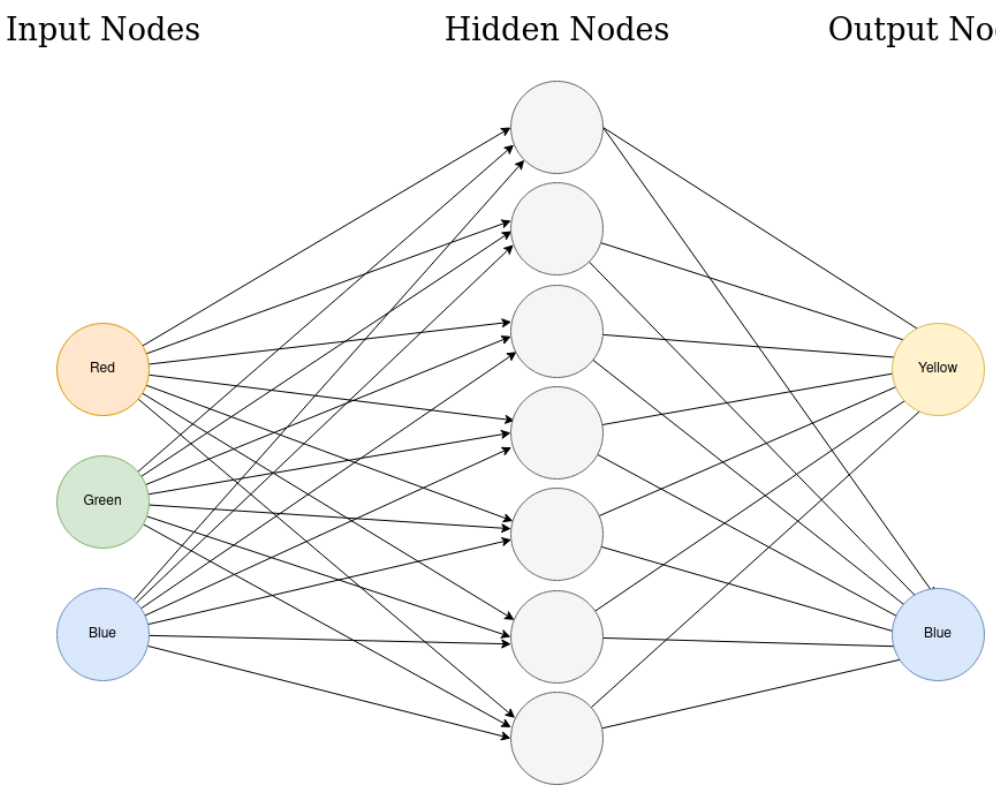

Figure 3.3: Neural Network Structure. The neural network structure includes three input nodes, seven hidden nodes, and two output nodes.

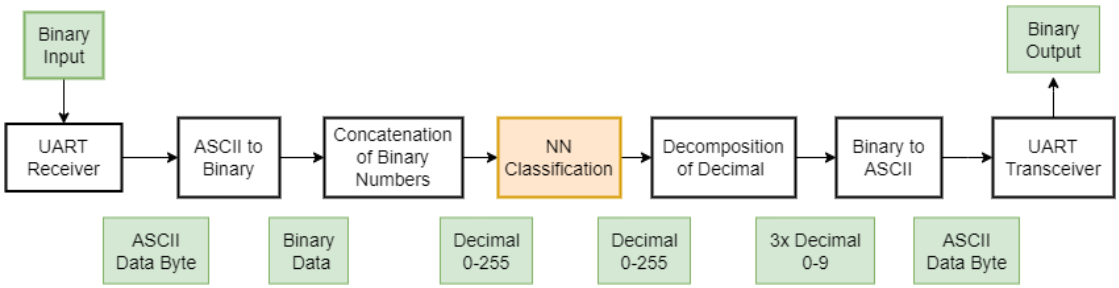

Figure 3.4: Data Decoding and Encoding. Illustrates how the input data is decoded and encoded to be properly used by the UART and neural network.

are sent one digit at a time, starting with the hundreds component and ending with the last digit. A space is sent between each element to indicate the number's end before the next element's hundreds component is transmitted. For example, if one wanted to transmit the color white (255, 255, 255), the order of transmission

would look like this:

$$2,5,5,\ space,2,5,5,\ space,2,5,5,\ space \tag{3.1}$$

The FPGA top-level module saves all digits in an array until a space is received, signifying the end of the number. Then, the digits will be concatenated to form a single number representing values from 0 to 255. When the total RGB pixel value is received, the values are passed through the neural network for classification. The classifier's output is then decomposed into decimal digits and converted into ASCII characters before each digit is transmitted individually. Similar to receiving, the end of each pixel element is indicated by a space.

The process ensures that the RGB pixel data can be accurately and efficiently transmitted and received by the computer and the FPGA. Additionally, it ensures that the values are correctly manipulated to be used by the neural network.

The two boards operate with different clock frequencies. The NG-Medium has a clock frequency of 25GHz, while the clock frequency of the Pynq-Z2 is five times as high at 125GHz. Because of this, a few code changes are required for the UART on the two boards to operate identically. Clockpulses per bit sent by the UART are decided by the clock frequency of the boards as well as the baud rate of the UART. This is calculated by the formula:

$$\frac{ClockFrequency}{BaudRate}$$

Table 3.1 shows how many clock pulses are required for the two devices.

Device	**Clock Frequency**	**Clocks per bit**
NG-Medium	25MHz	217
Pynq-Z2	125MHz	1085

Table 3.1: Number of clock pulses required to send one bit via UART with a baud rate of 115200

3.1.2 Neural Network Implementation

The neural network implementation consists of five parts: top-level, control system, multipliers, neurons, and sigmoid functions. The network is inspired by the work done by Marco Winzker [25], with alterations to account for timing and custom memory initialization. The top level wires the inputs and outputs from the neurons, monitors reset signals, and triggers classification by setting the "data enable" bit. This module also manipulates the output pixels depending on the classification outcome. The neural network control makes sure that the UART module is informed

when a classification is finished to make it start preparing the data for transmission. Since the network contains seven hidden nodes with three inputs each and two output nodes with seven inputs each, the system performs 35 multiplications and additions per classification and uses nine sigmoid functions, one for each node that is not an input.

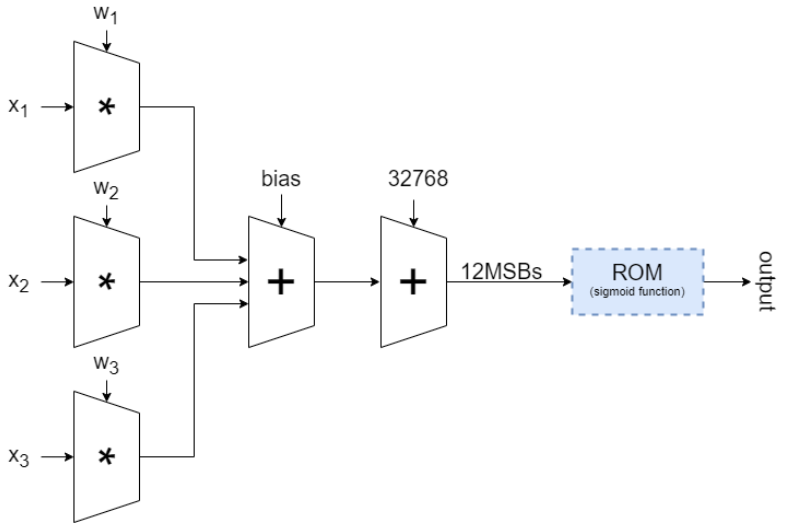

Figure 3.5: Neuron Structure. Showing necessary calculations that are performed inside of a hidden neuron. A 12-bit address is derived from the calculation's 12 most significant bits. The address is used to access a ROM, which returns the output. Output neurons follow the same principle but have seven inputs, and not three.

Each neuron in the neural network undergoes a series of computations, demonstrated in figure 3.5. Firstly, the inputs are multiplied with the corresponding weights before they are summed up with a bias. The result yields a 16-bit signed number converted to a 16-bit unsigned number by adding 32768. Using the output's 12 most significant bits, a 12-bit address can be derived. The address is then given to an activation function, which, in this case, is a sigmoid function.

$$h(x) = \frac{1}{1 + e^{-x}} \tag{3.2}$$

The nonlinearity of a sigmoid function is an advantage over linear functions as it allows the network to model a more complex relationship between the inputs and outputs. A generic sigmoid function is illustrated in figure 3.6; however, in our case, it will have a y-axis varying from 0 to 255, and an x-axis to satisfy a 12-bit unsigned memory address. It will therefore return a value between 0 and 255. The return value from the function is put on the output from the neural network, where

the output from the two nodes is comprehended and evaluated. If the output value is greater than 127, that label is chosen for the classification. If both are more significant than 127, the highest return value is chosen for the classification.

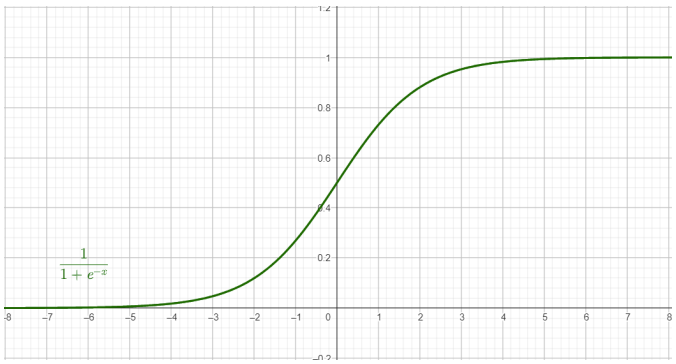

Figure 3.6: The Sigmoid Activation Function Graph. The sigmoid function's characteristic S-shaped curve with a steep middle portion and flattened top and bottom is ideal for neural networks. Its nonlinearity allows neural networks to model complex relationships between inputs and outputs, to see complicated patterns in large amounts of data.

Using two output nodes, the neural network is trained to categorize pixels into three categories: yellow, blue, and other. As the network classifies pixel by pixel, it is possible to train the network using only one large image seen in Figure 3.7. This image is 1280*720 pixels large and gives a training dataset of 921600 samples.

As a supervised learning algorithm is chosen, each training sample also contains a label indicating if the pixel is blue, yellow, or other. Labels are created by taking the training image and marking the yellow and blue areas with a definite RGB value; in this case, blue is indicated with white, and yellow is shown with gray, as seen in Figure 3.8.

The network is trained using a backpropagation algorithm with 400 epochs with a learning rate of 0.00001. A versatile memory initialization of a sigmoid function was used in order for the same design to function across both boards, ensuring independence from external packages. The read-only memory (ROM) is composed of 8 data bits, and 12 address bits, giving it a depth of 4096 words. The ROM stores the 8-bit sigmoid function lookup table.

Figure 3.7: Training Image. Used for training the neural network by providing different shades of yellow and blue.

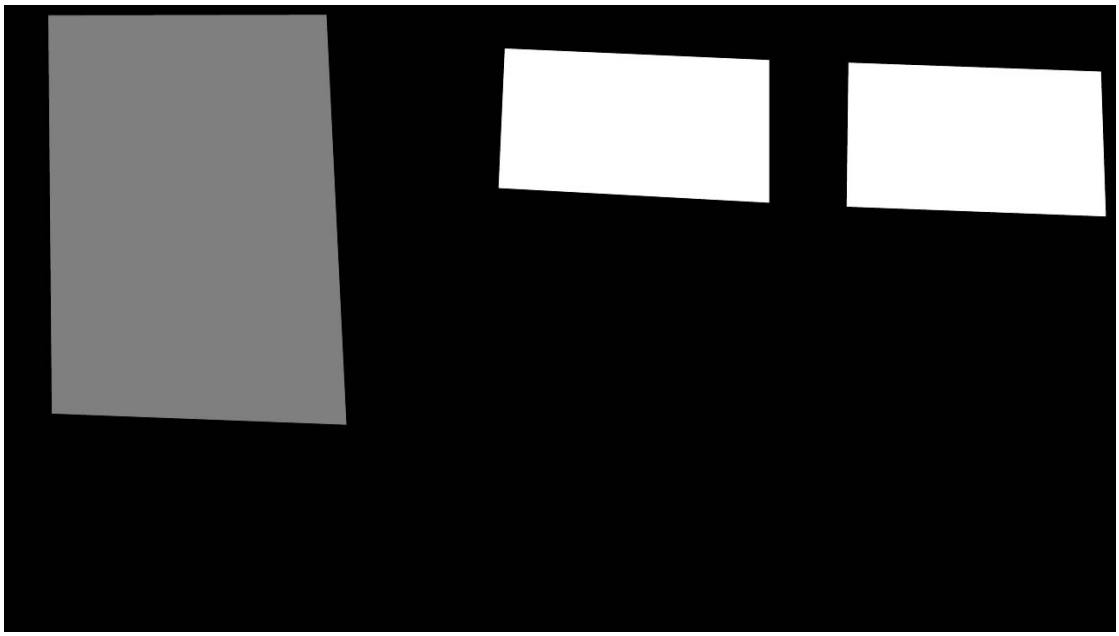

Figure 3.8: Training labels. Each individual pixel represents a label for the training image and is used when training the neural network. Gray represents the yellow areas, white represents the blue areas, and black represents everything else.

3.2 Data Collection

An experimental research strategy was chosen for this study. This is performed by testing the design on both a commercial FPGA and a rad-hard FPGA, to look at the physical differences between the boards and study the results. By doing this, the commercial FPGA will act as a control to determine if the design can be deemed efficient for rad-hard FPGAs.

Data were collected by analyzing log files and the design using Vivado and NXMap, for Pynq and NG-Medium, respectively. After the results and data are extracted, a comparison between the boards is performed in terms of performance and utilization. Data is extracted for each part of the design, where the UART, the ROM unit, and the neural network are all synthesized and analyzed separately to get a fuller picture of which parts of the system are more resource-heavy.

It is important to note that this data collection type has flaws. Considering that the data for each board are collected using different programs, the terminology is not the same across the platforms. Both data about utilization and timing are different. It's also essential to note that the internal clock on the boards is different by a factor of five, meaning that timing would not have an accurate comparison. A drawback to separating the different parts for data synthesis and collection is that it's not always easy to know how the program will react and how to allocate resources for the separate parts. This means that the estimation of resources used for the remaining part is not 100% accurate.

This chapter has provided an overview of the methodology adapted for the design and implementation of the system, consisting of a UART-based communication system and a neural network for color classification. The steps taken to create the system and how data collection was performed for the project have been reviewed. Furthermore, the design and optimization of the neural network have been briefly visited. The goal of providing the methodology and design is to offer more understanding of the project and ensure readers can reproduce results.

Chapter 4

Experimental Results

This chapter presents the experimental setup needed to gather the results, including a brief presentation of the hardware used. The experimental activity of my thesis consists of developing the emulation platform of a neural network on Pynq-Z2 and NG-medium. For this, the performance, utilization, and timing are analyzed and discussed to highlight the differences between the non-rad-hard FPGA, and the NG-Medium.

4.1 Hardware Characteristics

The system is tested using a computer, a USB-to-UART adapter, and two FPGAs; the commercial Xilinx Pynq-Z2, and the rad-hard NanoXplore NG-Medium.

4.1.1 Pynq-Z2

The Pynq-Z2 board, shown in Figure 4.1, is a commercialized Xilinx Zynq-based system-on-chip (SoC). It uses Python language and libraries to simplify the design process while still letting the designer exploit the benefits of programmable logic and microprocessors to build compatible electric systems. It consists of one 650MHz dual-core Cortex-A9 processor. The programmable logic comprises 13300 logic slices, each with 6-input LUTs and 8 flip-flops. It has fast block RAM, multiple clock management tiles, DSP slices, and an on-chip analog-to-digital converter.

4.1.2 NG-Medium

NanoXplore is the leading European company in designing and developing SoC FPGA technology. The NanoXplore Rad-hard BRAVE NG-Medium NX1H35S SRAM-based FPGA, shown in Figure 4.2, is explicitly designed to survive radiated

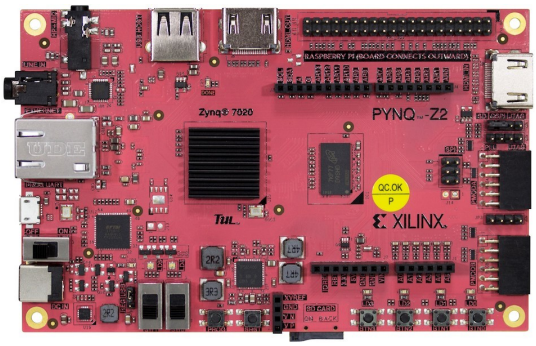

Figure 4.1: Pynq-Z2 board

environments like space. The FPGA is radiation hardened by design in configuration memories and registers, making it suitable for aerospace industry use. The NG-Medium is designed using 65nm STm C65-SPACE process technology, consists of 34272 4-input LUT tables, and has high-performance carry chains. It contains user memories with variable width and depth, allowing custom memory initialization.

Since the NG-Medium has a generally limited memory capacity and fewer LUTs than the Pynq-Z2. Deeper networks require more memory usage than smaller networks, and due to the limited memory of the NG-Medium, only smaller networks are suitable.

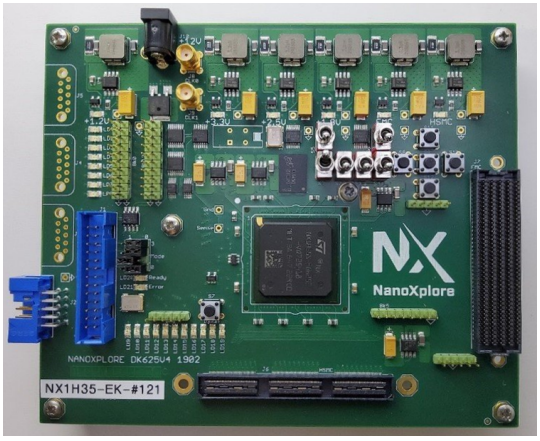

Figure 4.2: NG-Medium board

Using two different boards makes it possible to achieve valuable insight in comparison. The non-radiation hardened board, Pynq-Z2, has been used for neural networks in multiple studies ([26] [27] [28]) and has already established that it has a high-performance and reliability in non-radiation environments. On the other hand, the research on NG-Medium is lacking in terms of neural networks

and could provide valuable data on how radiation-hardened FPGAs respond to such applications. The comparative analysis can help researchers understand the advantages and challenges of using radiation-hardened FPGAs for neural network implementations for space applications.

4.1.3 WITMOTION USB-UART Converter

To be able to communicate between the computer and the boards using UART, the Witmotion USB-UART converter was utilized. This is a 3.3-5V Serial Adapter with a CH340 chip, making it a suitable solution for this project.

4.1.4 Personal Computer

A personal computer was used to transfer data and communicate with the boards. The computer needs to be able to send serial data via USB and have the appropriate synthesizing tools downloaded for both boards. NX-MAP was used for the NG-Medium, and Vivado was used for Pynq-Z2.

4.2 Experimental Setup

The experimental setup consists of testing and comparing the performance of the two boards.

The board is connected to a computer via the USB-to-UART adapter to send and receive data. The communication was tested in multiple steps to ensure everything was working correctly. Initially, a simple echo was used to ensure that UART communication and the board initialization were set up correctly. The next step was the test of a simple multiplier to ensure the boards could store inputs and process them before returning the correct answer and number of digits. The last step consisted of receiving a pixel value composed of red, green, and blue values ranging from 0 to 255 and returning those correctly.

Baudrate	115200bps
Start bit	1 (high)
Stop bit	1 (low)
Indication of number ending	space (0x20)

Table 4.1: UART settings and configurations

Table 4.1 shows the UART setting used for the setup. Single digits are transmitted using a baud rate of 115200bps, with one (high) start and one (low) stop bit. The end of the RGB value is indicated by sending a space after the last digit.

When communicating with the board, a serial communication application was used to send single-pixel values, and Python was used to send and receive larger amounts of data. Each step was tested on both boards to ensure the results were equal.

The design running on the boards comprises multiple parts, as described in Chapter 3.

1. **The main state machine**, is used to control the communication with UART and the neural network.
2. **Memory module**, memory initialization of a sigmoid function.
3. **Neural network**, implementation, and neural network control.

The memory was tested sequentially, reading from all memory locations and checking if the expected data was returned. The results were tested by altering the UART to return the direct reading from the memory locations instead of the result from the classification itself. Because of the relatively small neural network size, it was possible to test it by returning the outputs from each neuron. By creating a Python script that would do all the same calculations as the neural network and print the predicted output from all eleven neurons, the output from the network could be checked appropriately. Each neuron output was returned using the UART to ensure that the expected result matched the output from the neural network.

The whole system with all its modules was tested by sending known pixel values of different shades of yellow and blue, as well as other colors, and observing the outcome of the classification.

4.3 Results

In this section, the results will be presented and discussed. First, a rough introduction to the classification training results will be presented and the results achieved when running the classifier on the two boards. Next, the board's performance regarding utilization, efficiency, and area will be compared.

4.3.1 Neural Network Classifier

After training the neural network, the training image was tested to see what the minimum expected error could be. The output resulted in a loss of 4%, which is generally high, as one often aims for an error below 1 percent. By analyzing the training image, it was revealed that the image only consists of 10% blue and 16% yellow labels. This could mean the classifier might not have had enough data to work with. Looking at the output image, Figure 4.3, it is possible to see that the classifier is struggling with dark yellow colors.

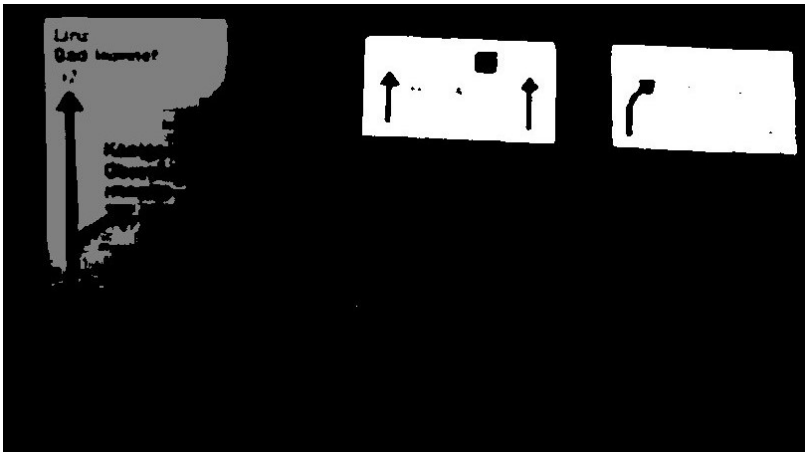

Figure 4.3: Classification Training Output. Output from testing the training image with the current weights shows that not all yellow areas are classified correctly.

Figure 4.4: Training Image Analysis. Illustration showing how some colors might be too similar for the system to categorize correctly, as colors are subjective to the viewer, and individual pixels might, therefore, not be categorized correctly.

This is most likely due to either not having an optimal training method or the colors in the images overlapping too much. Figure 4.4 show that the dark yellow spots might be too similar to several other points in the image and is therefore not classified as yellow, even if the human eye perceives it as yellow. From these results,

it could have been better to do area searches instead of single-pixel classification to teach the network that connected areas are often the same color. However, the classifier was not perfected as it still classified vibrant yellow and blue colors.

Figure 4.5 illustrates the image used to test the algorithm using a simulator, in this case, Vivado. The image is a screenshot from Google Maps, chosen because it has some different elements from the training images, while still containing clear yellow and blue areas.

Figure 4.5: Test image. Screenshot taken from Google maps [29] and converted to PPM format. Demonstrates blue sky and water, with a yellow stripe in the road.

Figure 4.6 shows the result when testing the algorithm using a simulator. It is possible to see that it is able to correctly classify the ocean and sky as blue, as well as the yellow stripe. It does however struggle with colors that are not bright. One can see that the light blue areas of the sky are not classified, and some green and gray areas are incorrectly classified as yellow or blue. The classification results show that the algorithm needs more training and possibly more input data for yellow and blue areas. However, minor classification mistakes are not a concern, as it is not the main component of the thesis and only serve to be a proof of concept.

Figure 4.6: Classification Result. Result from testing the neural network design using a simulation tool. Demonstrating that the system is able to correctly classify bright blue and yellow areas.

Pynq-Z2 Results

The result of the classifier on the boards was not calculated as the perception of what a user would classify as yellow and blue differs. The Pynq-Z2 did, however, manage to classify yellow and blue colors in a wide range correctly and returned shades of gray when it was exposed to other colors.

NG-Medium Results

Currently, the neural network is not working correctly on the NG-Medium, and the issue occurs on the neural network's output. All input and hidden nodes were tested individually and are working correctly; however, the output nodes are incorrect. The cause of the issue is still unknown, and it is unknown if the radiation-hardened capabilities of the NG-Medium cause the issue or if it is simply user error.

Besides the neural network, the board responds well to UART communication and can retrieve correct information from the implemented ROM.

4.3.2 Board Utilization and Performance Differences

The architecture of the two boards has been evaluated in terms of area, frequency, performance, and working capabilities. Additionally, each module of the design has been synthesized and evaluated separately. By doing this, it is possible to see which parts of the design require the most resources. The parts that are evaluated are the UART and the ROM module. Additionally, the resources needed for the neural network have been roughly estimated by subtracting the two others from the final design.

Table 4.2 shows the difference between the bit utilization between the two boards, and it is possible to see that the NG-Medium uses a lot fewer bits than the Pynq-Z2, not only in the total amount of bits but also in terms of the fraction total bits available.

	NG-Medium	PYNQ-Z2
Total bits	6 189 616	25 697 632
Critical bits	57 975	260 939
Percentage used	0.93%	1.02%

Table 4.2: Bit Utilization and percentage of total bits available

This is possibly due to the different hardware architecture between the two broads and them being synthesized and optimized using two different software applications.

There exist both advantages and limitations to utilizing fewer bits. A few advantages include the design consuming fewer FPGA resources, leaving more for other functionalities. This will also allow for a smaller FPGA, reducing cost. The design's power consumption will also generally be smaller, as the fewer bits consume less power, making the system more power efficient. Fewer bits might also achieve higher operating frequencies and faster performance, making timing closure easier to achieve.

Table 4.3 shows the shortest, average, and longest time taken from the 10 longest paths. NG-Medium operates with a clock frequency of 25 GHz, while Pynq-Z2 operates at 125 GHz. These frequencies define the maximum rate at which signals can be sampled or processed within their respective clock domains. It can be noted that even though the clock of NG-Medium is five times slower than the opponent, the path timing is only about three times longer. Since the difference between the critical paths is less severe than expected, this indicates that the critical path is not proportional to the clock frequency difference. And suggests that Pynq-Z2 is experiencing some routing delays that are not present in NG-Medium. Nonetheless, the Pynq-Z2 achieves a shorter timing delay than the counterpart, making it higher in performance.

	NG-Medium	PYNQ-Z2
Shortest	29.412ns	10.427ns
Average	29.426ns	10.611ns
Longest	29.444ns	11.057ns

Table 4.3: Ten longest paths timing

Table 4.4 shows some differences in the boards' utilization of resources. It is possible to see that the NG-Medium relies more heavily on 4-LUTs and DFFs, while the Pynq-Z2 relies more on DSP blocks. For the NG-Medium, some of the arrays, specifically the send and receive array used by the UART, were mapped from the RAM to DFF blocks, to make them readable and writable from multiple places. It is also possible to see that the Pynq-Z2 has more memory blocks available.

	NG-Medium		PYNQ-Z2	
4-LUT	709/32256	3%	238/13300	1.79%
Flip-flops (DFF)	1246/32256	4%	574/106400	0.54%
DSP	4/112	4%	33/220	15%
BRAM	9/56	17%	5/140	3.57%

Table 4.4: Memory utilization for the full design

When isolating only the Sigmoid ROM module, it was discovered that both boards utilized only one memory block.

NG-Medium Utilization of Isolated Models

	Full design	RAM	UART and Top-Level	NN (estimated)
4-LUT	709/32256		364/32256	354/32256
DFF	1246/32256		252/32256	994/32256
1-Bit carry	2923/8064		374/8064	2549/8064
Register file block	4/168		4/168	0
DSP	4/112		4/112	0
Memory block	9/56	1/56	9/56	

Table 4.5: Memory utilization for different parts of the system

Table 4.5 shows the utilization of the full design on the NG-Medium, compared to the isolated RAM, UART, and top-level, and the estimation of resources for the neural network. From this it is possible to see that the neural network is clearly the dominant user of resources, using magnitudes more DFFs and 1-bit carries

than the other parts of the system. The number of memory blocks utilized was not possible to estimate, as multiple models share parts of the memory.

Area Utilization

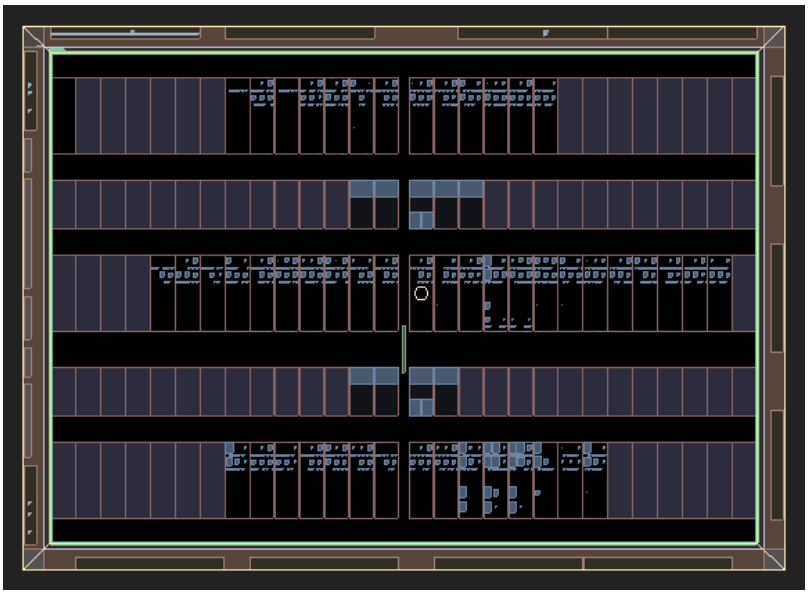

Figure 4.7: Placement on NG-Medium. Highlighted sections demonstrate the utilized areas of the NG-Medium.

Figure 4.7 shows the area utilization on the NG-Medium. The first, third, and fifth row consists only of clusters, which are made up of 408 LUTs and 384 DFFs each. The second and fourth line consists of RAM and DSP blocks. The lit-up places around the board represent used inputs and outputs, both simple and complex.

Figure 4.8 shows the area utilization of the PYNQ-Z2. The area on the board that is lighting up, shows the utilized resources.

In conclusion, the NG-Medium utilizes fewer resources than the Pynq-Z2, despite having the same design running on both boards. The Pynq-Z2 can classify bright yellow and blue colors well and works on a higher frequency with fewer delays than the NG-Medium. However, a fair comparison cannot be made due to the improper functioning of the classifier on the NG-Medium.

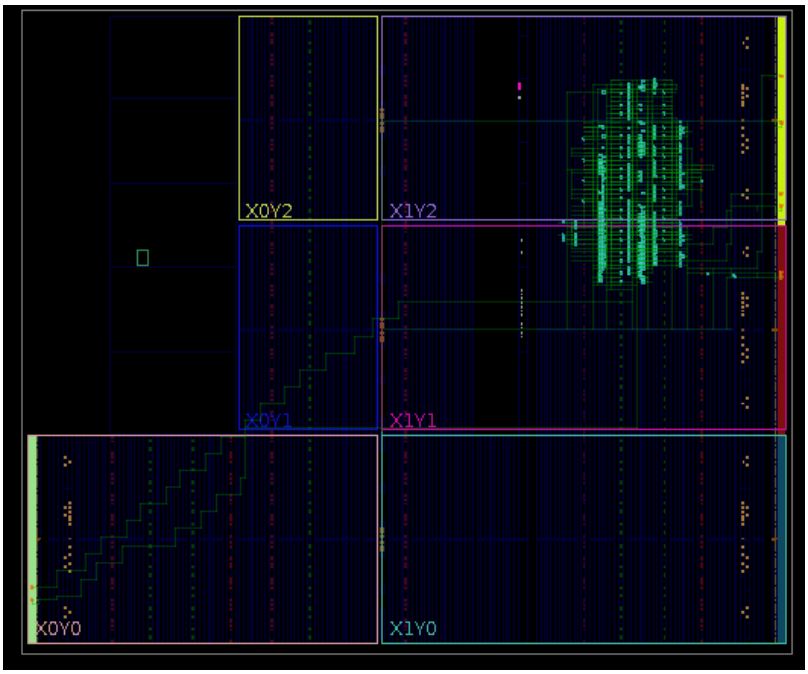

Figure 4.8: Placement on PYNQ-Z2. Highlighted sections demonstrate the utilized areas of the Pynq-Z2.

Chapter 5

Conclusion

This thesis investigated how the NG-Medium compares to standard non-radiation hardened FPGAs regarding performance, reliability, and fault tolerance. The goal was to understand the practical implications and potential advantages radiation-hardened solutions can pose in radiation-intensive environments.

For this purpose, a feed-forward neural network was trained using backpropagation and implemented on the non-radiation-hardened FPGA Pynq-Z2 and the radiation-hardened NG-Medium. A UART system was also developed to communicate with the boards and send and receive data digit by digit to and from the network. The results of the classification, utilization of the FPGAs, and timing analysis were compared, and potential advantages or implications were highlighted in the results section.

Concluding this thesis, it is clear that exploring neural networks on radiation-hardened FPGAs for space applications is complex but can prove very useful. While this research failed to prove the usefulness of neural networks on the NG-Medium, the study still sheds light on the potential advantages of combining neural networks with rad-hard FPGAs.

The finding presented in this study gives initial insight into the performance and utilization that can be expected when employing neural networks on a rad-hard FPGA like the NG-Medium. The unique architecture of the NG-Medium presented unforeseen challenges, where the system did not operate as expected. While the cause is unknown, the hope is that it can motivate others to research the topic further.

Furthermore, this developed system would need to undergo testing in simulated radiation environments before any final conclusions could be made. While this thesis aimed to lay out the groundwork for understanding the performance of neural networks on rad-hard FPGAs, it remains to see how these systems perform under real radiation conditions.

In conclusion, this thesis has opened the door for new possibilities and challenges

when combining neural networks and radiation-hardened FPGAs for space missions. Through research, experimentation, and collaboration, neural networks on rad-hard FPGAs might still be the future advancements needed in the space industry, providing onboard detection and classification for space crafts and rovers.

Bibliography

[1] Berndt Müller, Joachim Reinhardt, and Michael T Strickland. *Neural networks: an introduction*. Springer Science & Business Media, 1995 (cit. on p. 4).

[2] Murat H. Sazlı. «A brief review of feed-forward neural networks». In: *Communications Faculty of Sciences University of Ankara Series A2-A3 Physical Sciences and Engineering* (2006). DOI: 10.1501/commua1-2_0000000026 (cit. on p. 6).

[3] Vladimir Nasteski. «An overview of the supervised machine learning methods». In: *Horizons. b* 4 (2017), pp. 51–62 (cit. on p. 6).

[4] Tor A. Myrvoll, Stefan Werner, and Magne H. Johansen. *Estimation, detection and classification theory* (cit. on p. 7).

[5] Shubham Gandhare and B. Karthikeyan. «Survey on FPGA Architecture and Recent Applications». In: *2019 International Conference on Vision Towards Emerging Trends in Communication and Networking (ViTECoN)*. 2019, pp. 1–4. DOI: 10.1109/ViTECoN.2019.8899550 (cit. on p. 8).

[6] Julien Lamoureux and Steven J. E. Wilton. «FPGA Clock Network Architecture: Flexibility vs. Area and Power». In: *Proceedings of the 2006 ACM/SIGDA 14th International Symposium on Field Programmable Gate Arrays*. FPGA '06. Monterey, California, USA: Association for Computing Machinery, 2006, pp. 101–108. ISBN: 1595932925. DOI: 10.1145/1117201.1117216. URL: https://doi.org/10.1145/1117201.1117216 (cit. on p. 9).

[7] Leonard Rockett, Dinu Patel, Steven Danziger, Brian Cronquist, and J.J. Wang. «Radiation Hardened FPGA Technology for Space Applications». In: *2007 IEEE Aerospace Conference*. 2007, pp. 1–7. DOI: 10.1109/AERO.2007.353098 (cit. on p. 10).

[8] Zbigniew Hajduk. «Reconfigurable FPGA implementation of neural networks». In: *Neurocomputing* 308 (2018), pp. 227–234. ISSN: 0925-2312. DOI: https://doi.org/10.1016/j.neucom.2018.04.077. URL: https://www.sciencedirect.com/science/article/pii/S0925231218305393 (cit. on p. 10).

[9] Janardan Misra and Indranil Saha. «Artificial neural networks in hardware: A survey of two decades of progress». In: *Neurocomputing* 74.1 (2010). Artificial Brains, pp. 239–255. ISSN: 0925-2312. DOI: https://doi.org/10.1016/j.neucom.2010.03.021. URL: https://www.sciencedirect.com/science/article/pii/S092523121000216X (cit. on p. 10).

[10] Donald L. Hung and Jun Wang. «Digital hardware realization of a recurrent neural network for solving the assignment problem». In: *Neurocomputing* 51 (2003), pp. 447–461. ISSN: 0925-2312. DOI: https://doi.org/10.1016/S0925-2312(02)00627-6. URL: https://www.sciencedirect.com/science/article/pii/S0925231202006276 (cit. on p. 10).

[11] Felix Siegle, Tanya Vladimirova, Jørgen Ilstad, and Omar Emam. «Mitigation of Radiation Effects in SRAM-Based FPGAs for Space Applications». In: *ACM Comput. Surv.* 47.2 (Jan. 2015). ISSN: 0360-0300. DOI: 10.1145/2671181. URL: https://doi.org/10.1145/2671181 (cit. on pp. 11, 13).

[12] Michael J. Wirthlin. «FPGAs operating in a radiation environment: lessons learned from FPGAs in space». In: *Journal of Instrumentation* 8 (2013) (cit. on p. 12).

[13] Jeffrey S. George. «An overview of radiation effects in electronics». In: *AIP Conference Proceedings* 2160.1 (Oct. 2019), p. 060002. ISSN: 0094-243X. DOI: 10.1063/1.5127719. eprint: https://pubs.aip.org/aip/acp/article-pdf/doi/10.1063/1.5127719/14196223/060002_1_online.pdf. URL: https://doi.org/10.1063/1.5127719 (cit. on p. 12).

[14] Raoul Velazco, Pascal Fouillat, and Ricardo Reis. *Radiation effects on embedded systems*. Springer Science & Business Media, 2007 (cit. on p. 13).

[15] R. Katz, K. LaBel, J.J. Wang, B. Cronquist, R. Koga, S. Penzin, and G. Swift. «Radiation effects on current field programmable technologies». In: *IEEE Transactions on Nuclear Science* 44.6 (1997), pp. 1945–1956. DOI: 10.1109/23.658966 (cit. on p. 14).

[16] Miguel Ángel Vázquez, Pol Henarejos, Irene Pappalardo, Elena Grechi, Joan Fort, Juan Carlos Gil, and Rocco Michele Lancellotti. «Machine Learning for Satellite Communications Operations». In: *IEEE Communications Magazine* 59.2 (2021), pp. 22–27. DOI: 10.1109/MCOM.001.2000367 (cit. on p. 14).

[17] Jacob Høxbroe Jeppesen, Rune Hylsberg Jacobsen, Fadil Inceoglu, and Thomas Skjødeberg Toftegaard. «A cloud detection algorithm for satellite imagery based on deep learning». In: *Remote Sensing of Environment* 229 (2019), pp. 247–259. ISSN: 0034-4257. DOI: https://doi.org/10.1016/j.rse.2019.03.039. URL: https://www.sciencedirect.com/science/article/pii/S0034425719301294 (cit. on p. 14).

[18] C. O. Dumitru, V. Andrei, G. Schwarz, and M. Datcu. «MACHINE LEARNING FOR SEA ICE MONITORING FROM SATELLITES». In: *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences* XLII-2/W16 (2019), pp. 83–89. DOI: 10.5194/isprs-archives-XLII-2-W16-83-2019. URL: https://isprs-archives.copernicus.org/articles/XLII-2-W16/83/2019/ (cit. on p. 15).

[19] H. G. Lewis, S. Cote, and A. R. L. Tatnall. «Determination of spatial and temporal characteristics as an aid to neural network cloud classification». In: *International Journal of Remote Sensing* 18.4 (1997), pp. 899–915. DOI: 10.1080/014311697218827. eprint: https://doi.org/10.1080/014311697218827. URL: https://doi.org/10.1080/014311697218827 (cit. on p. 15).

[20] Ryoichi Imasu et al. «Greenhouse gases Observing SATellite 2 (GOSAT-2): mission overview». In: *Progress in Earth and Planetary Science* 10.1 (2023), p. 33 (cit. on p. 15).

[21] Matthew C Chan and John P Stott. «Deep-CEE I: fishing for galaxy clusters with deep neural nets». In: *Monthly Notices of the Royal Astronomical Society* 490.4 (Oct. 2019), pp. 5770–5787. ISSN: 0035-8711. DOI: 10.1093/mnras/stz2936. eprint: https://academic.oup.com/mnras/article-pdf/490/4/5770/30820714/stz2936.pdf. URL: https://doi.org/10.1093/mnras/stz2936 (cit. on p. 15).

[22] Massimo Ferraguto, Tim Wittrock, Mark Barrenscheen, Matti Paakko, and Ville Sipinen. «The On-Board Control Procedures Subsystem for the Herschel and Planck Satellites». In: *2008 32nd Annual IEEE International Computer Software and Applications Conference*. 2008, pp. 1366–1371. DOI: 10.1109/COMPSAC.2008.218 (cit. on p. 15).

[23] W Wimmer. «Remote control of satellites and applied automation». In: *IFAC Proceedings Volumes* 14.2 (1981), pp. 2323–2328 (cit. on p. 15).

[24] P. Blacker, C. P. Bridges, and S. Hadfield. «Rapid Prototyping of Deep Learning Models on Radiation Hardened CPUs». In: *2019 NASA/ESA Conference on Adaptive Hardware and Systems (AHS)*. 2019, pp. 25–32. DOI: 10.1109/AHS.2019.000-4 (cit. on p. 15).

[25] Marco Winzker. *NN RGB FPGA*. https://github.com/Marco-Winzker/NN_RGB_FPGA. 2021 (cit. on p. 21).

[26] Yahia Said. «Pynq-YOLO-Net: An embedded quantized convolutional neural network for face mask detection in COVID-19 pandemic era». In: *International Journal of Advanced Computer Science and Applications* 11.9 (2020) (cit. on p. 27).

[27] Thang Viet Huynh. «FPGA-based acceleration for convolutional neural networks on PYNQ-Z2». In: *International Journal Of Computing and Digital System* (2021) (cit. on p. 27).

[28] Xiaoying Hou, Meixia Fu, Xifang Wu, Zhongiie Huang, and Songlin Sun. «Vehicle license plate recognition system based on deep learning deployed to PYNQ». In: *2018 18th International Symposium on Communications and Information Technologies (ISCIT)*. IEEE. 2018, pp. 79–84 (cit. on p. 27).

[29] Google Maps. *Location on google maps, RV555, Sotra bridge*. Accessed July 6th, 2023. 2023. URL: https://www.google.com/maps/@60.3725788,5.1688432,3a,75y,92.9h,82.97t/data=!3m6!1e1!3m4!1srwiDZZBuuPrN51mf51t8mg!2e0!7i16384!8i8192?entry=ttu (cit. on p. 31).