The Impact of Single Event Effect Reliability of Convolution Neural Network Architectures and Hardening Approaches Implemented on SRAM FPGA

A Thesis Submitted to the
College of Graduate and Postdoctoral Studies
In Partial Fulfillment of the Requirements
For the Degree of Master of Science
In the Department of Electrical and Computer Engineering
University of Saskatchewan
Saskatoon

By

Yixiu Wang

PERMISSION TO USE

ABSTRACT

Convolution neural networks (CNNs) have powerful data processing and learning capabilities, which have been widely applied to image processing related applications, especially in autonomous driving, medical image classification, space exploration and military applications. Due to the low power consumption, high flexibility, and parallel characteristics of modern field-programmable gate arrays (FPGAs), they are frequently used in CNN implementation as a hardware acceleration platform. Two architectures are mainly used to implement CNNs on FPGAs: the streaming architecture and single computation engines (SCEs) architecture. In the streaming architecture of a CNN, each layer is implemented with one distinct hardware block and each block can be optimized separately. On the other hand, the single computation engine architecture uses a systolic array of processing elements or a matrix multiplication unit as a computation engine to execute the CNN layers sequentially. The control of the hardware and the scheduling of operations is performed by a control unit and associated software. The advantage of this design paradigm is that it consists of a fixed architectural template that can be scaled based on the input of CNNs and the available FPGA resources. Therefore, it is suitable to implement modern complex CNNs that may not fit into the streaming architecture.

SRAM-based FPGAs are sensitive to radiation effects, which can generate single event effects (SEEs) in the system. Designs are required to reduce the radiation effects in FPGA-based CNNs for many applications. Previous radiation effects studies mainly focused on streaming architecture and explored triple-modular redundancy (TMR) or selective hardening techniques. As far as the authors know, there are very few radiation effects studies on the CNNs implemented with SCEs architecture on FPGAs and no radiation effects evaluation between the two architectures with proton irradiation.

In this thesis, we implement a Modified National Institute of Standards and Technology (MNIST) CNN with two mainstream architectures, both streaming architecture and SCEs architecture, on a Xilinx Zynq UltraScale+ multiprocessor system on a chip (MPSoC) ZCU-102 evaluation kit. Then we evaluate their error, hang, and total failure rate with proton irradiation test at Tri-University Meson Facility (TRIUMF). The cross-section results for different architectures showed that the SCEs design has higher error cross-sections and total failure cross-sections than that of the streaming architecture, even though SCEs architecture uses much fewer hardware resources in FPGA. In addition, two resilience techniques for SCEs architecture named spatial TMR and temporal TMR are designed and adopted for the SCEs architecture with the same hardware structure and utilization by reusing process elements (PEs) or using multiple PEs to carry out each calculation. As a result, the cross-sections of the spatial TMR and temporal TMR SCEs architecture designs are reduced by 34.9% and 59.2%, with an execution time overhead of 14.2% and 21.4% compared with non-harden one, respectively. Thus, the study shows that SCEs architecture for FPGA acceleration has excellent potential for applications in a radiation environment with minimal overhead due to its scalability and flexibility, and spatial TMR and temporal TMR could effectively reduce the error rate and total failure rate with no extra hardware resources. This suggests that spatial TMR and temporal TMR propose in my project seems to be generic for SCEs architecture, and it could be a better redundancy choice for complex CNNs implement with not enough hardware resources.

ACKNOWLEDGEMENTS

This thesis can’t be finished without some other people’s help. I appreciate them and would like to say thank you so much!

Firstly, I would like to say thank you to my father and mother. Even if I am not by your side, once I encounter a problem that I can't solve, you will still try your best to help me. At the same time, give me the right to choose my own life, trust me and encourage me. Thanks for all those loves, I know I can afford it, but I will try my best to make a better life by myself. I will love you forever.

Secondly, I would like to thank you for Dr Chen, who is the most patient teacher I have ever seen. Dr. Chen not only teach my knowledge, but also teach me how to think and solve problems in my research. In addition, he also teaches me to find the inner motivation for my own life. Thank you, professor; without the help, I can publish the papers and finish my studies.

Thirdly, I do want to say thank you to Jin. He is a good and trustworthy person, as my best friend, he helps me a lot in daily life, as my teacher, he teaches me a lot of software knowledge and programming idea, as working partners, we have completed one after another project together, without his help, I can’t complete my research.

Also, I want to say thank you to Dr Teng and Dr Ko. I took their class during the graduate time. I learned a lot about verification and computer architecture knowledge, which let me understand my project.

Furthermore, I want to say thank you to Shuting and Zhiwu, and you guys carried on all those proton tests. Without that help, I could not get experimental so successfully.

Finally, I want to say thanks to all the people in Dr Chen’s group and my roommate; you guys give me a lot of help in lab and everyday life. Without that help, I could not devote all of myself to the project.

Thank you for all you help, patient, trust and encourage.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

ASIC	Application Specific Integrated Circuits
ARM	Advanced RISC Machine
AMBA	ARM Advanced Microcontroller Bus Architecture
AXI	The Advanced eXtensible Interface
APU	Application Processing Unit in ARM
BRAM	Block Memory
BNN	Binary Neural Network
BUFGMUX	Buffer Global Clock Mux
CNN	Convolutional Neural Network
CPU	Central Processing Unit
CLA	Carry-lookahead Adder
CLB	Configurable Logic Block
CONV	Convolutional Layer
CAN	Controller Area Network
DCM	Digital Clock Manager
DFF	D-flip-flop
FPGA	Field Programmable Gate Arrays
FC	Fully-connect Layer
FIT	Failure in Time
GPIO	General-purpose Input/Output
GPU	Graphics Processing Unit
IOB	Input/Output Block
LCU	Logic Control Unit
LSB	Least Significant Bit
MOU	Memory Optimization Unit
MCU	Memory Control Unit
MACs	Multiply Accumulate
MNIST	Modified National Institute of Standards and Technology
MPSoC	Multiprocessor System on a Chip
NSREC	Nuclear & Space Radiation Effects Conference

MSB	Most Significant Bit
PCIe	Peripheral Component Interconnect Express
PS	Processing System
PL	Programmable Logic
ReLU	Rectified Linear Unit
RPU	Real-time Processing Unit in ARM
SAs	Streaming Architectures
SRAM	Static Random-access Memory
SATA	Abbreviated from Serial AT Attachment
SCEs	Single Computer Engines
SEE	Single Event Effect
SET	Single Event Transient
SEU	Single Event Upset
TRIUMF	Tri-University Meson Facility
TMR	Triple Module Redundancy
UART	Universal Asynchronous Receiver/Transmitter

1. INTRODUCTION

1.1 Introduction:

As a new and disruptive science and technology, artificial intelligence is leading the transformation of science and technology and industrial applications [1]. With the development and application of artificial intelligence, human lifestyle and thinking structure are gradually being affected [2,3]. Artificial intelligence is different from the automation of conventional computers enables machines to learn [4], organize, adapt, and act. It can be said that artificial intelligence gives machines life and wisdom.

Since 1950s, Alan Turing, the father of artificial intelligence, who asked question about machine thinking, and published the thesis "Can Machines Think?" [6]. Then in 1970s, the first anthropomorphic robot born at Waseda University [5,7]. Later, Geoffrey Everest Hinton introduced backpropagation algorithm to multi-layers neural network training in 1980s. Until now in the 21st century, artificial intelligence has ushered in vigorous development. With the continuous improvement of artificial intelligence technology, this technology has also been put into more and more application fields, like the computer program 'AlphaGo' [8], speech recognition 'Siri' [9], facial and fingerprints recognition in mobile phone unlocking area [10-12], Tesla's autonomous driving [13], and automated production in machine manufacturing [14]. Even beyond earth, artificial intelligence in space, like exploration robot with build-in artificial intelligence [29]. Simultaneously, with the development of artificial intelligence, algorithm research, data training, artificial intelligence chips, and other scientific fields are also gradually advancing [15].

With the large-scale application of artificial intelligence in various human life scenarios showing in Figure 1.1, human life has brought many conveniences. At the same time, it has dramatically improved the quality of human life. While bringing these benefits to convenience, the relative has also produced Safety hazards [16,17], such as the burst of acceleration accident of Tesla's artificially driven car, even though the agency said it cause by pedal misapplication, but there are still doubt and worry about its reliability [27]. The aircraft accident for Ethiopia Airlines flight ET302, which may be due to the maneuvering characteristics augmentation system which is a flight law computer software algorithm [28]. These accidents caused by artificial intelligence have also led people to start research on the safety and reliability of the artificial intelligence field.

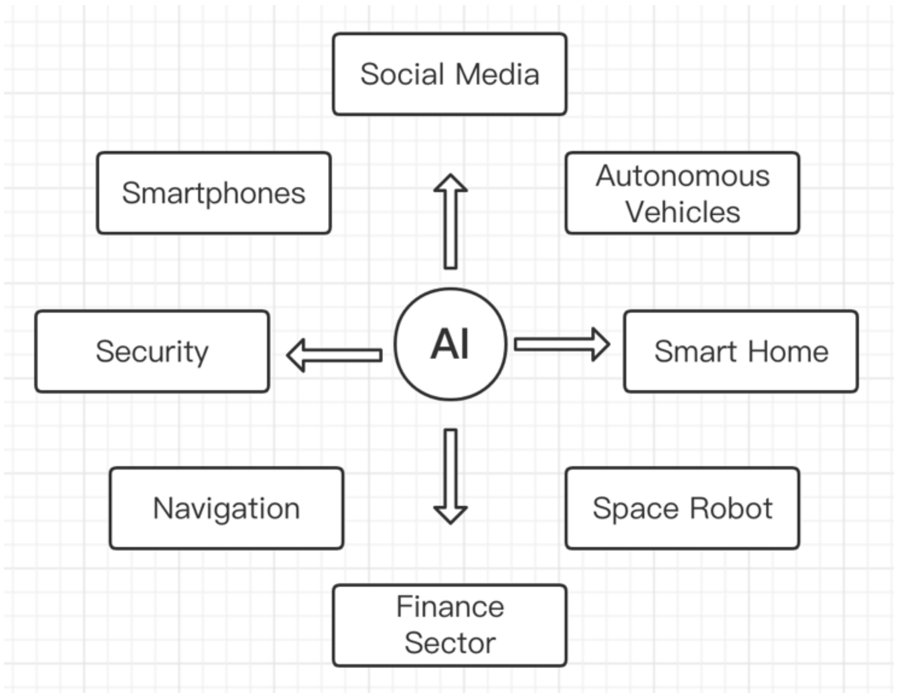


Figure 1.1 Example of AI in life

At the same time, as the development of artificial intelligence, the amount of computing and data processing has gradually increased. The types of equipment used have also increased. Simultaneously, as the amount of data processing has increased, the error rate will increase correspondingly. A significant application direction is the hardware field, such as Field Programmable Gate Arrays (FPGA), Application Specific Integrated Circuits (ASIC), Graphics Processing Unit (GPU), etc. This hardware all contains the part composed of Static Random-access Memory (SRAM) [26], which are sensitive to SEE. When a microelectronic device, such as a microprocessor, semiconductor memory, or power transistors [18-20] exposed to external radiation, such as (ions, electrons, photons...), the storage particles will be reversed or come with a current transient, it may cause configuration memory flip or output wave change. The CNN function will be destroyed, as is shown in Figure 1.2. Single event upset (SEU) happens on global memory, and single event transient (SET) occurs on PE unit. Therefore, research the reliability of SRAM-based hardware devices is beneficial for understanding and safe application of artificial intelligence.

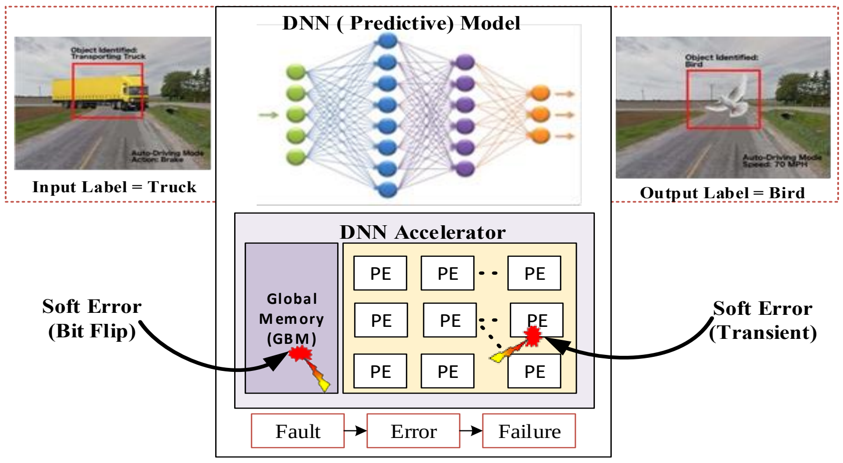


Figure 1.2 Single Event Upset and Single Event Transient

1.2 Motivation:

Since FPGA is called a field programmable logic Array, it has vital flexibility and wide reusability. Convolutional Neural Network (CNN) is widely used on SRAM-based FPGA. For example, in data image processing, autonomous driving, military industry and aerospace. Figure1.4 shows the FPGA-based image recognition system. These scenarios cover a large part of the application of FPGA scenes in our lives [21,22]. But due to the characteristics of SRAM storage particles, when it exposed to an irradiation environment, SEE may happen on SRAM-based FPGAs. Then, the output result will be influenced. Therefore, studying the reliability of CNN, which implement on SRAM-based FPGA, will help us to better understand and apply CNN on FPGA-based FPGA, also bring convenience to our lives.

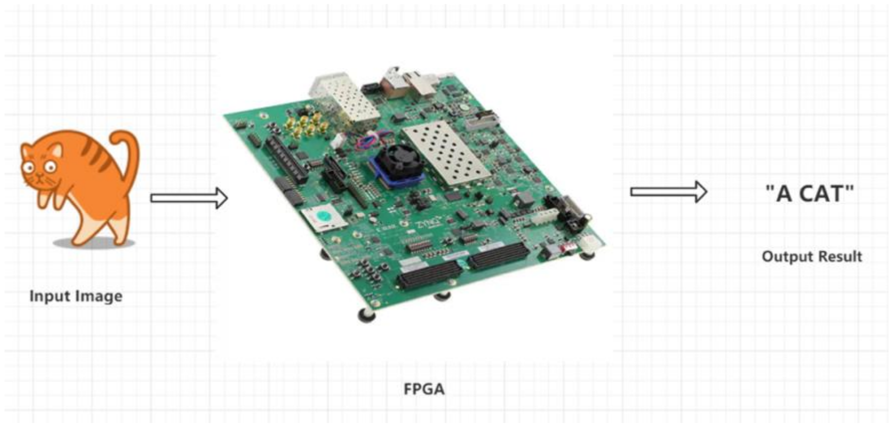


Figure 1.3 Image Identification based on FPGA

Nowadays, most CNNs used on FPGAs are composed of two architectures, Single compute engines (SCEs) architecture and streaming architectures [23]. Studying and comparing the reliability of the two architectures under irradiation will allow us to select the better architectures for application in different scenarios.

Recently, there have been many harden designs of CNNs on SRAM-based FPGA. Most of the design is based on the traditional Triple Module Redundancy (TMR). The reliability has been increased by adding additional redundant circuits for a voting handshake or reduce the calculation data, such as selective harden or Binary Neural Network based (BNN) methods. However, all these reinforcement methods mention before are all aimed at streaming architecture. There are relatively few reinforcement methods and research for the SECs architecture [24-25]. Therefore, studying SCEs and designing relative reinforcement methods can better understand and apply the SCEs architecture.

As artificial intelligence is widely used on SRAM-based FPGAs for data processing and accelerated calculations, at the same time, with the increase in data volume and the complexity of calculations, system reliability has gradually become a concern after processing frequency and power consumption.

Therefore, studying the reliability performance of different architectures of artificial intelligence on SRAM-based FPGAs and designing corresponding reinforcement methods will allow us to better understand how to select different architectures and corresponding reinforcement methods according to different situations, also make CNNs run and work more stable.

1.3 Objectives

According to the researching content in this work, the overall goal of the thesis is to study the impact of SEE reliability of CNN architectures and hardening approaches implemented on SRAM-based FPGA. The study will follow the steps.

1) Train the MNIST Neural Network.

   Select a basic and representative MNIST model, then use Keras to train a MNIST CNN that can recognize handwritten 0-9 numbers with an accuracy of 91%.

2) Implement MNIST by Single Computation Engines (SCEs) and Streaming architecture on FPGA.

   Design the trained MNIST CNN according to SCEs architecture and the Streaming architecture, then implement them on the FPGA, respectively.

3) Test and compare the reliability of the two architectures, analyze and draw a conclusion.

   Proton tests are performed on the MNIST network of two different architectures. Calculate the cross-section based on the test results. Analyze based on the combination of its architecture and utilization.

4) Design two harden methods for SCEs architecture.

   According to the characteristics of SCEs structure, design two harden methods called temporal and spatial TMR by multiplexing PEs. Without adding additional circuits.

5) Test and compare in general, analyze and draw conclusion.

   Test these two harden methods for SCEs architecture, compare with traditional SCEs, analyze and draw a conclusion.

1.4 Thesis Organization

The main content and structure of the thesis is organized as follows:

The first chapter focuses on the motivation and objectives of this study. It provides the risks of SEEs for SRAM-based FPGAs with the development and application of convolutional neural networks (CNNs). It then presents the objective of the research and reports the research status. Finally, it discusses the project's goals, specific plans, and corresponding steps.

The second chapter explains the prior knowledge and literature research involved in the thesis First, it explains the CNN and the primary CNN model based on the modified national institute of standards and technology (MNIST) database used in the project. Subsequently, the FPGA and its elemental composition are introduced, and the characteristics of the modern processing subsystem and programmable logic (PS+PL) heterogeneous FPGA are introduced. Then, the two primary structures, Streaming architecture and SCEs architecture, used in the project are described. Finally, the SEE and TMR are briefly explained.

The third chapter explains the design proposed in this thesis. Two basic CNN acceleration models were applied to FPGAs, namely to SCEs architecture and Streaming architecture. The related design and implementation were introduced according to the use of the MNIST network and the applied FPGA, followed by detailed design and data processing steps. The second part introduces the two hardened designs based on the SCEs architecture. The two hardening approaches are called temporal and spatial TMR. Its design features and how it is implemented on an FPGA are described. Then, its calculation steps as well as how redundancy is achieved are explained. Finally, the appendix related code and references is included.

The fourth chapter mainly describes the proton experiment. The details related to the experiment are first described. Then, the situations and classifications of the experiment and the experimental results are discussed.

In the fifth chapter, according to the relevant experimental data and the situation in the experiment, the different architectures are analyzed. The experimental phenomenon from the perspective of resource use and computing design is considered, and then the experimental results of different reinforcement methods are combined with their design. The calculation method (reuse rate and refresh rate) is analyzed, and the advantages and disadvantages of the reinforcement method are detailed.

The last chapter summarizes the implementation and progress of the project and the results obtained at this stage. First, it outlines the advantages and disadvantages of the two different architectures under irradiation environment and how to choose applications. Subsequently, two other reinforcement methods for the structure of SCEs are summarized, future work and related conjectures based on the current results are explained.

1.5 References

[1] H. Barrow, "Keynote Speaker-3: History of Computing and AI, a Personal Viewpoint," *2015 17th UKSim-AMSS International Conference on Modelling and Simulation (UKSim)*, 2015, pp. 5-5, doi: 10.1109/UKSim.2015.109.

[2] Tu Xuyan, "Life, Artificial Life and Generalized Artificial Life," *2005 International Conference on Neural Networks and Brain*, 2005, pp. 1425-1428, doi: 10.1109/ICNNB.2005.1614898.

[3] M. A. Titu and A. Stanciu, "Human and Aritificial Intelligence: the partnership that imparts hope," *2019 11th International Conference on Electronics, Computers and Artificial Intelligence (ECAI)*, 2019, pp. 1-6, doi: 10.1109/ECAI46879.2019.9041998.

[4] T. Shibata, K. Ohkawa and K. Tanie, "Spontaneous behavior of robots for cooperation. Emotionally intelligent robot system," *Proceedings of IEEE International Conference on Robotics and Automation*, 1996, pp. 2426-2431 vol.3, doi: 10.1109/ROBOT.1996.506527.

[5] Takanishi A. (2019) Historical Perspective of Humanoid Robot Research in Asia. In: Goswami A., Vadakkepat P. (eds) Humanoid Robotics: A Reference. Springer, Dordrecht.doi.org/10.1007/978-94-007-6046-2_145.

[6] P. T. Saunders, "Alan Turing and biology," in *IEEE Annals of the History of Computing*, vol. 15, no. 3, pp. 33-36, 1993, doi: 10.1109/85.222839.

[7] I. Kato et al., *Development of Bipedal Walking Robot (WABOT-1)*. Biomechanism, vol. 2 (edited and published by SOBIM printed by University of Tokyo Press, 1973), pp. 173—214

[8] Silver, D., Schrittwieser, J., Simonyan, K. *et al.* Mastering the game of Go without human knowledge. *Nature* **550,** 354–359 (2017).

[9] Siri Team, "Hey Siri: An On-device DNN-powered Voice Trigger for Apple's Personal Assistant" 2017.10. [Online]. Available: https://machinelearning.apple.com/research/hey-siri

[10] S. Subhash, P. N. Srivatsa, S. Siddesh, A. Ullas and B. Santhosh, "Artificial Intelligence-based Voice Assistant," *2020 Fourth World Conference on Smart Trends in Systems, Security and Sustainability (WorldS4)*, 2020, pp. 593-596, doi: 10.1109/WorldS450073.2020.9210344.

[11] E. Jiang, "A review of the comparative studies on traditional and intelligent face recognition methods," *2020 International Conference on Computer Vision, Image and Deep Learning (CVIDL)*, 2020, pp. 11-15, doi: 10.1109/CVIDL51233.2020.00010.

[12] Xiao Sun and Zhuming Ai, "Automatic feature extraction and recognition of fingerprint images," *Proceedings of Third International Conference on Signal Processing (ICSP'96)*, 1996, pp. 1086-1089 vol.2, doi: 10.1109/ICSIGP.1996.566282.

[13] M. A. A. Babiker, M. A. O. Elawad and A. H. M. Ahmed, "Convolutional Neural Network for a Self-Driving Car in a Virtual Environment," *2019 International Conference on Computer, Control, Electrical, and Electronics Engineering (ICCCEEE)*, 2019, pp. 1-6, doi: 10.1109/ICCCEEE46830.2019.9070826.

[14] J. -J. Aucouturier *et al.*, "Cheek to Chip: Dancing Robots and AI's Future," in *IEEE Intelligent Systems*, vol. 23, no. 2, pp. 74-84, March-April 2008, doi: 10.1109/MIS.2008.22.

[15] Y. Chen *et al.*, "DaDianNao: A Machine-Learning Supercomputer," *2014 47th Annual IEEE/ACM International Symposium on Microarchitecture*, 2014, pp. 609-622, doi: 10.1109/MICRO.2014.58

[16] Scherer, Matthew U. "Regulating artificial intelligence systems: Risks, challenges, competencies, and strategies." *Harv. JL & Tech.* 29 (2015): 353.

[17] OA Oso ba , and WI Welser. "The Risks of Artificial Intelligence to Security and the Future of Work." (2017). [Online]. Available: https://www.rand.org/pubs/perspectives/PE237.html

[18]E. Normand, "Single-event effects in avionics," in *IEEE Transactions on Nuclear Science*, vol. 43, no. 2, pp. 461-474, April 1996.

[19]S. Duzellier et al., "Low energy proton induced SEE in memories," in *IEEE Transactions on Nuclear Science*, vol. 44, no. 6, pp. 2306-2310, Dec. 1997.

[20]R. A. Reed et al., "Heavy ion and proton-induced single event multiple upset," in *IEEE Transactions on Nuclear Science*, vol. 44, no. 6, pp. 2224-2229, Dec. 1997.

[21] G. Wei, Y. Hou, Z. Zhao, Q. Cui, G. Deng and X. Tao, "Demo: FPGA-Cloud Architecture For CNN," *2018 24th Asia-Pacific Conference on Communications (APCC)*, 2018, pp. 7-8, doi: 10.1109/APCC.2018.8633447.

[22] T. Geng *et al.*, "FPDeep: Acceleration and Load Balancing of CNN Training on FPGA Clusters," *2018 IEEE 26th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2018, pp. 81-84, doi: 10.1109/FCCM.2018.00021.

[23] [2] Venieris, S. I., Kouris, A., &amp; Bouganis, C. (2018). Toolflows for Mapping Convolutional Neural Networks on FPGAs. ACM Computing Surveys, 51(3).

[24] F. Libano *et al.*, "Selective Hardening for Neural Networks in FPGAs," in *IEEE Transactions on Nuclear Science*, vol. 66, no. 1, pp. 216-222, Jan. 2019.

[25] F. Libano, *et al.*, "Understanding the Impact of Quantization, Accuracy, and Radiation on the Reliability of Convolutional Neural Networks on FPGAs," in *IEEE Transactions on Nuclear Science*, vol. 67, no. 7, pp. 1478-1484, July 2020.

[26] W. S. Yu, R. Huang, S. Q. Xu, S. Wang, E. Kan and G. E. Suh, "SRAM-DRAM hybrid memory with applications to efficient register files in fine-grained multi-threading," *2011 38th Annual International Symposium on Computer Architecture (ISCA), 2011, pp. 247-258.*

[27] Ian Duncan, "Bursts of acceleration in Tesla vehicles caused by drivers mistaking accelerators for breaks, feds conclude," [Online]. Available: https://www.washingtonpost.com/transportation/2021/01/08/tesla-brakes/

[28] Aircraft Accident Investigation Bureau Interim report. [Online]. Available: https://reports.aviation-safety.net/2019/20190310-0_B38M_ET-AVJ_Interim.pdf

[29] Artificial intelligence in space. [Online]. Available: https://www.esa.int/Enabling_Support/Preparing_for_the_Future/Discovery_and_Preparation/Artificial_intelligence_in_space

2. BACKGROUND

This chapter will explain the knowledge involved in this project, first, the basic knowledge about CNN and MNIST database. Then comes FPGA knowledge, including the architecture of the basic FPGA, the architecture of modern heterogeneous FPGAs, and the AXI protocol. After that, introducing two CNN hardware acceleration architectures based on FPGA named Streaming and SCEs architecture. Finally, the knowledge of SEU and Redundancy.

2.1 Convolution Neural Network

2.1.1 Introduction

CNNs are a deep learning model or multilayer perceptron similar to Artificial Neural Networks (ANNs), which is often used to analyze visual images [1]. The emergence of CNNs is inspired by biological processing [2] because the connection pattern between neurons is similar to the tissue of the visual cortex of animals [3]. Individual cortical neurons only respond to stimuli in a restricted field of vision, called the receptive field. The receptive fields of different neurons partially overlap so that they can cover the entire field of view.

CNN architecture, as shown in Figure 2.1, is very similar to the ANN architecture [4,5], especially the last layer of the network, which is fully-connected (FC) layer. An additional convolutional layer, activation layer, and pooling layer are used for feature extraction to simulate human brain processing.

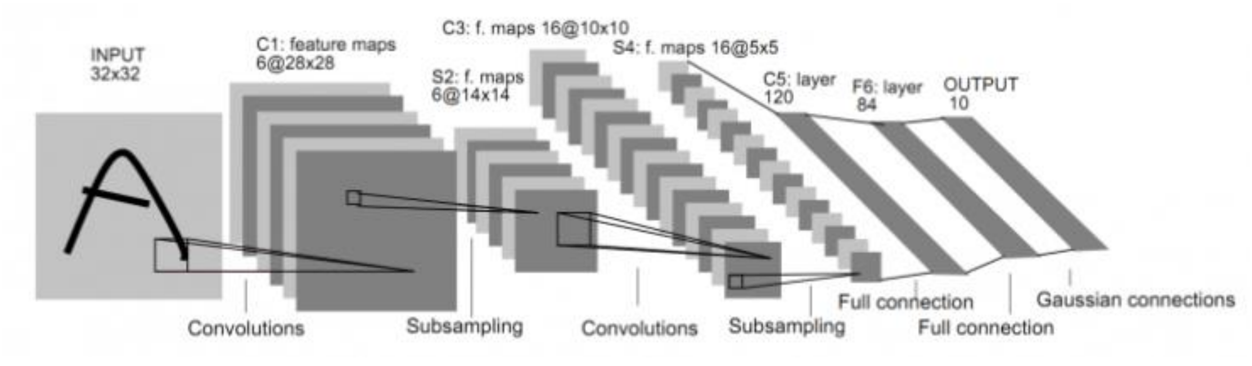


Figure 2.1 Convolution Neural Network Architecture Copyright

Figure 2.2 shows an example of the basic components of CNNs, which is a typical car plate recognition CNN [6]. It makes specific judgments based on the input image. It first extracts the features through the convolutional layer, and then performs linear conversion through the activation layer, after which the features are filtered through the pooling layer. Finally, the features extracted to obtain the final result by using a fully connected layer.

2.1.2 Construction

1) Convolutional Layer

In a CNN, each convolutional layer is composed of several convolution kernels, and the parameters of each convolution kernel are optimized using the backpropagation algorithm. The purpose of the convolution operation is to extract different input features [7]. The first layer of the convolutional layer may only extract some low-level features, such as edges, lines, and corners. More layers of the network can iteratively extract more complex features from the lower-level features.

2） Rectified Linear Units (ReLU) Layer

The activation function of this layer of nerves uses linear rectification [8]. The convolution layer performs multiple convolutions on the original image to generate a set of linear activation responses, whereas the nonlinear activation layer performs a nonlinear activation on the previous results from latest layers.

3） Pooling Layer

Typically, after the convolutional layer, a feature with a large dimension is obtained [9]. The feature is cut into several regions, and the maximum or average value is taken to obtain a new feature with a smaller dimension.

4） Fully-Connected Layer

The acquired local features obtained were fully connected and partially characterized by the weight matrix generated and were assembled into a complete picture [10].

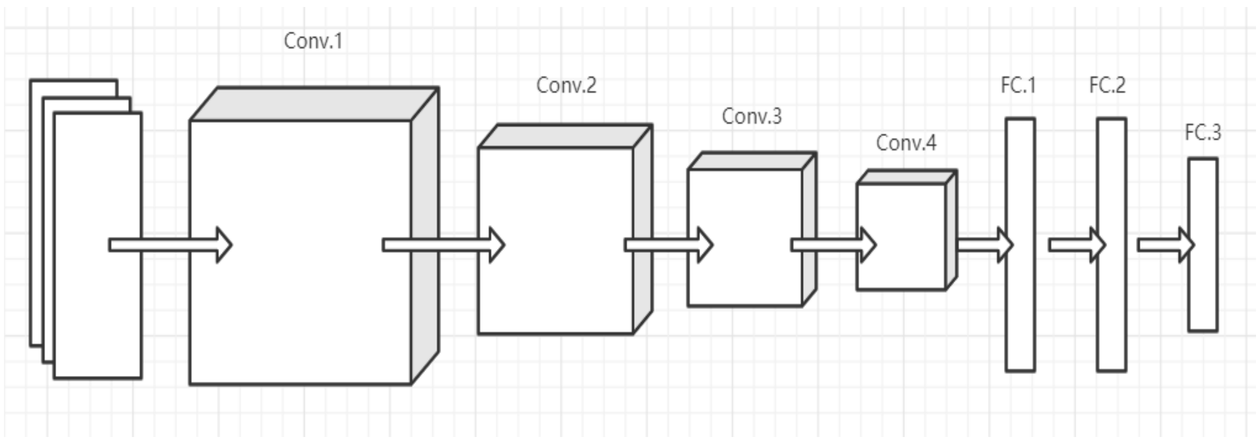


Figure 2.2 Convolution Neural Network construction example [6]

2.1.3 MNIST

The MNIST database is an extensive database of handwritten digits that is commonly used to train various image processing systems [11,12]. The MNIST database uses 28 × 28 pixels handwritten array as the dataset. The test and training sets are separated.

The main task of the MNIST handwritten digit recognition model is to input an image of handwritten digits and then recognize which digit is handwritten in the image [13], as shown in Figure 2.3. For this project, we trained and designed a MNIST-based network with an accuracy of 90% for subsequent hardware design.

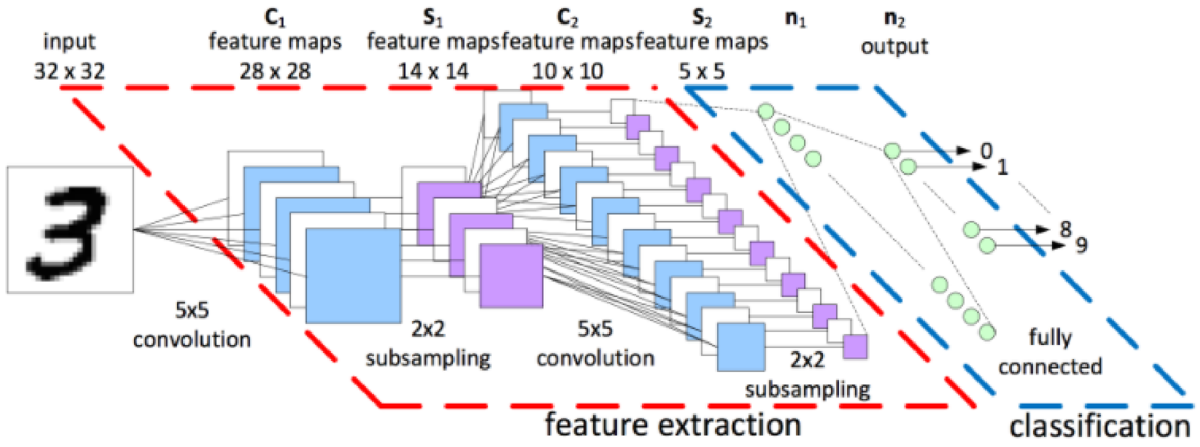


Figure 2.3 MNIST Construction Example Copyright

2.2 PFGA

FPGAs, also known as field programmable logic gate arrays, are an integrated circuit with programmable characteristics that are pre-designed and implemented on silicon [14,15]. It can be configured as a specified circuit structure according to the needs of designers, so that customers do not have to rely on chip manufacturers. It is widely used in prototype verification, communications, automotive electronics, industrial control, aerospace, data centers, and other fields.

2.2.1 Component

The internal structure of the FPGA (mainly for XILINX FPGAs) includes a programmable input/output unit (I/O bank), configuration logic block (CLB), digital clock management module (DCM), embedded block RAM (BRAM), global clock network (global clock mux) wiring resources and embedded underlying functional units [16]. The components of the standard FPGA are shown in Figure 2.4.

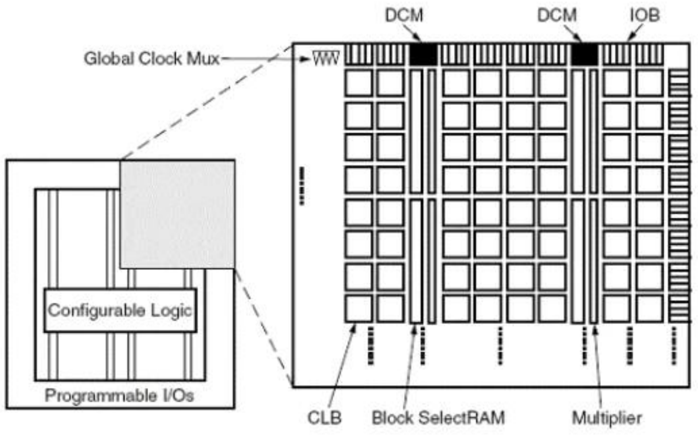


Figure 2.4 FPGA Constructions Copyright @ Xilinx

(1) Input/output Blocks

At present, most FPGA I/O units are designed as programmable modes, which can be flexibly configured through software and constraints to adapt to different electrical standards and corresponding I/O physical characteristics [17]. The user can adjust the supply voltage, adjust the input drive current, and match the corresponding impedance and resistance.

(2) Configurable Logic Blocks

The basic configuration logic blocks of the FPGA are composed of look-up tables (LUTs) and registers. The function of the LUT is to complete a pure combinational logic function [18]. In this way, synchronous or asynchronous reset and clock enable can be determined, and it can also be configured as a latch. The sequential design is completed through the register in the FPGA, and the relatively complex design is completed through the register and the LUT together. Classic configurable logic blocks are composed of LUTs and registers. The internal structure of the registers and LUTs of different manufacturers are different. The number and combination modes are also different. For example, in the Xilinx 7 Series FPGA, each CLB can be configured as either a 6-input LUT with one output or as two 5-input LUTs with two separate outputs.

(3) Global Clock Mux

Most of the device modules in the FPGA are synchronized. Therefore, it is necessary to use a standard clock to make synchronizers when developing an FPGA project. In FPGA timing analysis, due to the uncertainty of the system frequency and combinational logic delay, the time for the signal to reach each synchronous device inside the FPGA is different, which affects the reliability and functionality of the system operation.

The role of the global clock mux is to solve the clock synchronization problem [19]. The clock signal is connected to the root of the tree as the origin point. Different synchronization devices are connected to the leaves as the endpoints, and through the buffer global clock mux (BUFGMUX) the same clock signal reach different synchronization devices with the same delay.

(4) Digital Clock Manager

DCM is an important device for processing the clock inside an FPGA [20]. There are three main components: clock de-skew, frequency synthesis, and phase shifting.

2.2.2 Construction

(1) UltraScale+ Multiprocessor System on Chip (MPSoC)

The ZCU-102 FPGA used was the Zynq UltraScale+ MPSoC series, which is the second-generation Zynq platform of Xilinx. It contains a complete ARM processing subsystem (PS) in its FPGA, including a quad-core Cortex-A53 processor or a dual-core Cortex-A53 plus a dual-core Cortex-R5 processor [21]. The entire processor was built around the processor. Moreover, the processor subsystem integrates a memory controller and a large number of peripherals such that the processor core is completely independent of the programmable logic unit in Zynq. That is, if the programmable logic unit (PL) is not used temporarily, the subsystem of the processor can also work independently, which is essentially different from the previous FPGA. The UltraScale+ MPSoC is shown in Figure 2.5.

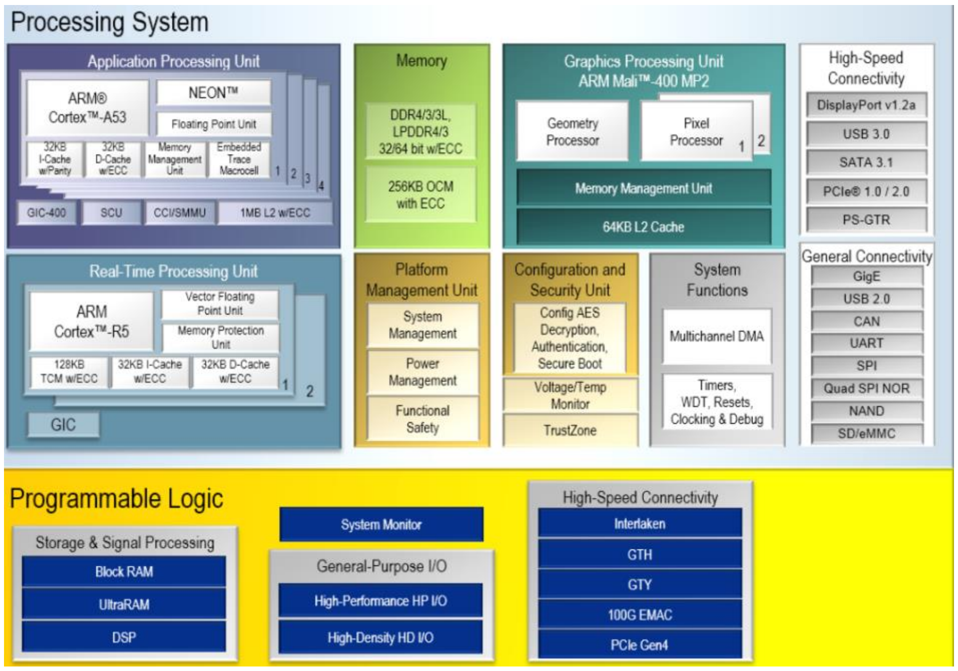


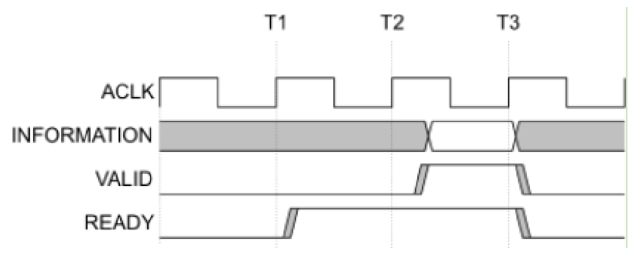
Figure 2.5 Zynq UltraScale+ MPSoC Architecture Copyright @ Xilinx

The Zynq platform has two major functional blocks: the PS and PL parts. More directly, they are the SoC part of the ARM and the FPGA part. Among them, the PS integrates the APU ARM Cortex™-A53 processor, RPU Cortex-R5 processor, advanced microcontroller bus architecture (AMBA)® interconnection, internal memory, external memory interface (DDR controller), and peripherals. These peripherals include a USB bus interface, Ethernet interface, SD/eMMC interface, I2C bus interface, CAN bus interface, UART interface, GPIO [22], and high-speed interfaces, such as PCIe, SATA, and Display Port.

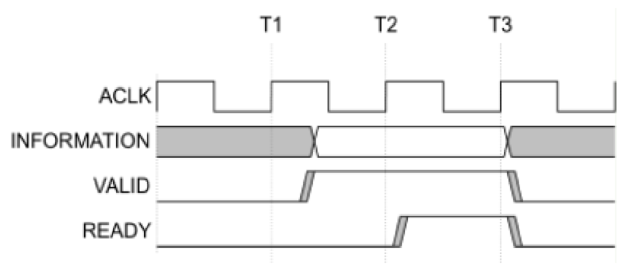
(2) Advanced eXtensible Interface (AXI)

AXI is an interface protocol introduced by Xilinx from the six series of FPGAs. It mainly describes the data transmission method between the master and slave devices. AXI continues to be used in Zynq, where the version is AXI4, so we often see AXI4.0 the Zynq internal equipment has an AXI interface. In fact, AXI is a part of the AMBA proposed by ARM [23]. It is a high-performance, high-bandwidth, and low-latency on-chip bus, which is also used to replace the previous AHB and APB buses. The first version of AXI (AXI3) was included in AMBA3.0, released in 2003, and the second version of AXI (AXI4) was included in AMBA 4.0, which was released in 2010.

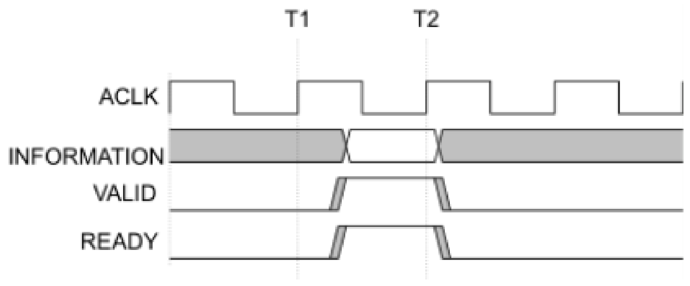
The AXI protocol mainly describes the data transmission mode between the master and slave devices. The master device and slave device establish a connection through a handshake signal. There are three types of AXI handshake methods. First, READY waits for VALID (as shown in Figure 2.6 (a)). The READY signal from the slave asserts after T1, which means that the slave is ready to receive data. Until the VALID signal becomes logic 1, the data in the information line will meet the handshake between the slave and master device. Then, the data are sent to the slave device.



(a)



(b)



(c)

Figure 2.6 AXI Hand-shake protocol Copyright @ Xilinx

The second type is the ready VALID first, as shown in Figure 2.6 (b). For the master device, the data plan to send is ready for transfer; therefore, in the information line, the data is waiting for the handshake between the slave and master to be done after T1. Then, until T2, the ready signal from the slave asserts and the data held in the information line are sent to the slave part. The third type is READY and VALID asserted simultaneously (as shown in Figure 2.6 (c)), where the VALID from the master and the READY single from the slave become logic 1 at the same time. Then, the information with the VALID signal will be sent to the slave part.

In any of these three handshake scenarios, the slave part sends out the READY signal when the slave device is ready to receive data. When the data of the master device is ready to be sent, it sends out and maintains the VALID signal with the data used in the transfer, indicating that the data is valid and stable. Data starts to transfer between the slave and master parts only when the VALID and READY signals are both valid at the same clock cycle, which is also a means of handshake. When these two signals continue to be valid, the master device continues to transmit the next data, which also means a burst in the AXI protocol. In each READY VALID period, all data in the information line can be sent from the master to the slave part. In this burst process, the master device can cancel the VALID signal at time, or the slave device can cancel the READY signal to terminate the transmission [23]. However, once the handshake agreement is not met, data transfer between the slave and master stops.

2.3 Neural Network Accelerator

2.3.1 Streaming Architectures

In the CNN network in the streaming architecture, various layers are placed in the FPGA, and each module corresponds to each CNN level [25]. The module was specially optimized and processed. Different hardware modules are arranged according to the CNN's structural order and placed in the FPGA species, as shown in Figure 2.7. All the CNN structural modules, the convolutional layer, the pooling layer, the activation layer, and the fully connected layers, are connected in turn. They are calculated and run in the FPGA in a pipelined manner [24].

Data will be passed from the PC to the FPGA and transmitted in each module component in sequence according to the streaming fluid structure through different parts of the neural network. Because this method uses the parallel characteristics between the various levels through the pipeline, the CNN running on FPGA has the characteristic of running scans in parallel, which greatly increases the speed of calculation. Simultaneously, due to the method of fluid mapping, the streaming architecture consumes a lot of logic resources, resulting in an inflexible design and demanding equipment requirements. At the same time, because of the fluid mapping method, the streaming architecture consumes a lot of logic resources. The design needs to change frequently and requires considerable time for compilation.

At this stage, the basic design of the fluid architecture network is to separate the CNN used and then to split each layer in order; each stage is mapped to the FPGA according to the module, and the pair can be parallelized. The pipeline design was carried out during the processing stage. Subsequently, the performance resources and space allocation are adjusted from each module to meet the needs of each layer and the overall design [25,26]. Different hardware systems in the streaming architecture are responsible for executing different CNN parts. Therefore, when there is a CNN architecture that needs to be implemented, the FPGA must be completely reconfigured. However, this design is less flexible and has low reusability. However, due to the pertinence and uniqueness of its design, the streaming architecture for different CNN models are very effective.

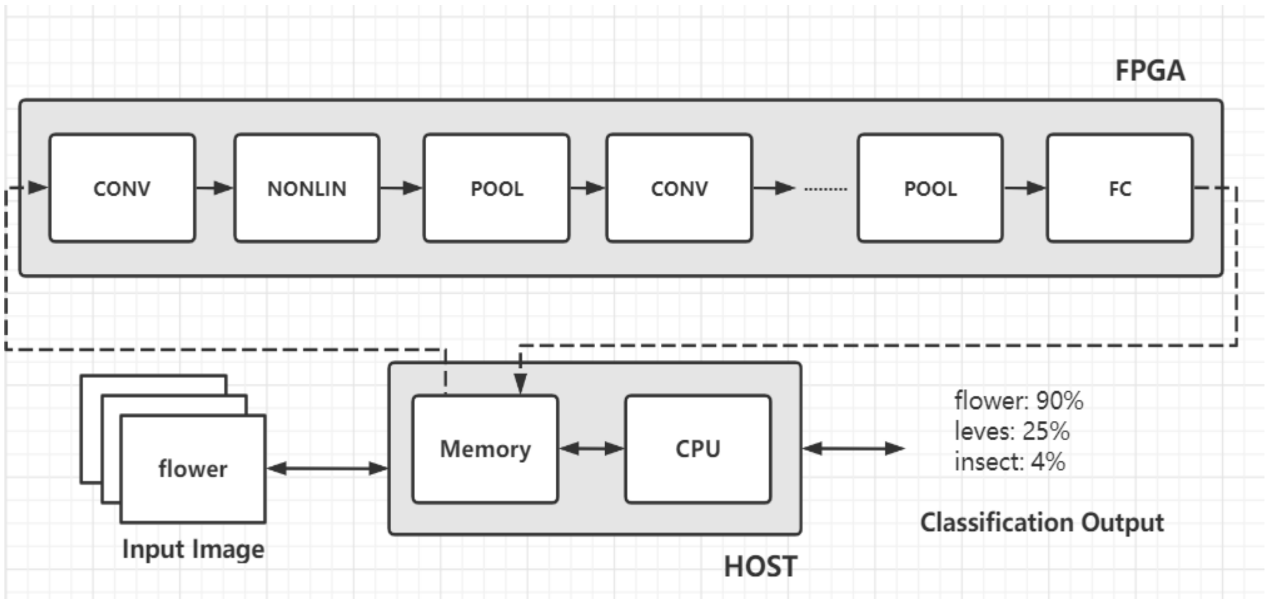


Figure 2.7 Example of a streaming accelerator architecture [24]

2.3.2 Single Computation Engines Architecture

This design method is more flexible than a partially customized streaming structure and saves resources. The SCEs architecture is composed of arithmetic units that become the processing engine, which usually executes the data received from the buffer to process the systolic array or matrix calculations [24]. The calculation process and calculation method of the hardware in the SCEs architecture is scheduled and determined by the software, as shown in Figure 2.8. This design is a fixed template. The hardware end is composed of many processing elements with a control unit [27]. To mobilize and allocate the computing unit, we read the corresponding computing data. In terms of software, the CPU will need to execute the corresponding CNN level and transmit the corresponding instructions to the control unit to read and manipulate the data. The use of a template can be scaled correspondingly, according to the number of resources the FPGA board uses and the CNN being applied.

Through this scheme, when different CNN networks are executed, different instruction sequences can be communicated through the CPU, so that the hardware side can allocate and call processing elements according to the new instructions to perform arithmetic processing. This method highlights the flexibility of the design, which does not need to target the CNN network and can configure and expand the modification system based only on the resources of the FPGA to be used. Therefore, after a compilation and design, the same hardware design bitstream can carry multiple CNN networks without additional time overhead due to redesign and configuration. Although the SCEs architecture has greatly improved the flexibility, the processing efficiency is reduced due to the processor-like control mechanism. A template was used at the same time. Adapting to multiple CNNs will lead to large performance differences between different CNNs and different workloads.

In the network using SCEs at this stage, different instruction sequences are first designed for the CPU and mapped to the FPGA hardware. Then, the hardware is designed according to the characteristics of the hardware and the amounts of resources. Then, it was passed according to different CNNs. The CPU sends different instruction sequences to the hardware to execute the CNN.

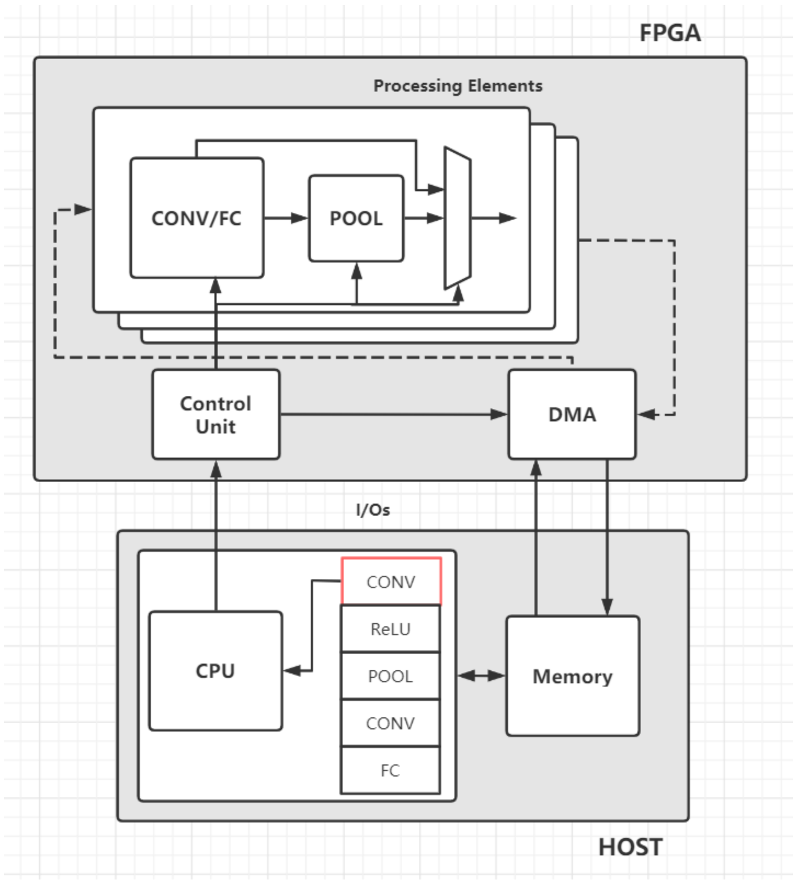


Figure 2.8 Example of a single computer engine accelerator architecture [24]

2.4 SEU and Redundancy

SEEs refer to a radiation effect that causes abnormal changes in the state of the device when a single high-energy particle passes through the sensitive area of a microelectronic device, including single-event flipping, single-event latch, and single-event upset [28]. SEEs cause these two main errors. The first type of error is a “soft error.” When a soft error happens, a temporary also non-destructive error will occur in the system. It will not lead a permanent failure and can recover by itself. The second type of error is a “hard error” which causes permanent failure for the entire system.

2.4.1 Single Event Upset

Sequential logic circuits and combinational logic circuits are the two main categories of digital circuits. Both the current input and current can determine the output for the sequential logic circuit. This means that a memory block or element should be set in a sequential logic circuit to store the previous data. In Figure 2.8, it is the basic transistor-level schematic of a D-flip-flop (DFF) and the most basic register element in integrated circuit (IC) design. There are two D-latches, each containing a latch and transmission gate. Two latch inverters between 0 and 1 to store the 1-bit data. The transmission in INV1 and INV4 is open when the clock is 0. The data are latched in the first latch; that is,, the data is stored in the first latch and cannot pass to the second latch because of the close for gate in INV2. In this period, the second latch holds the previous data until the clock signal asserts. As the clock rising edge arrives, gate1 and gate4 will close, and the gates in INV2 and INV3 will be switched on. At the same time, data will be passed to latch2 and stored in latch1, and stable data can be grabbed in this moment.

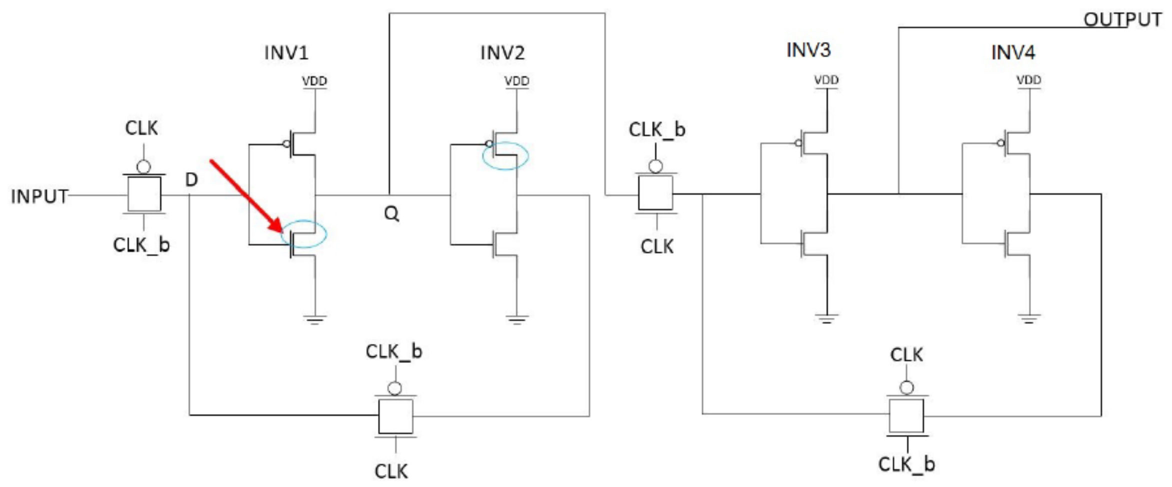


Figure 2.9 Transistor-level schematic of D-flip-flop

The first paragraph describes the basic DFF process. The DFF will work properly (store 0 or 1) when it matches the set-up and hold-up time. However, when a single high-energy particle passes through the sensitive area, such as the PMOS in DFF, it may cause the store bit in the latch to flip, and the output data from the DFF will be incorrect. In this bit flip situation, the minimum total charge to upset the data stored in the latch is called the critical charge (Q_C), which is the most important parameter to judge whether the data stored in latch will reverse. Equation 2.1 shows that the critical charge can be considered as the product of the total capacitance of the transistor node (C_n) and the voltage of the power supply (V_{dd}).

$$Q_C = C_n \times V_{dd} \tag{2.1}$$

$$\sigma = n/N \tag{2.2}$$

$$FIT = \sigma \times \phi \times 10^9 \tag{2.3}$$

When an SEU occurs in a microelectronic device, which means that in a radiation environment, the sensitive node has collected enough charge (greater than the critical charge), data in latch 1 is 0 (Q=1, D=0) when the clock signal asserts. The NMOS in INV2 and the PMOS in INV1 are closed in this situation. The PMOS in INV2 and NMOS in INV1 are turned on at the same time, which indicates that both PMOS in INV1 and NMOS in INV2 are drained. This is shown in Figure 2.8. If a single high-energy particle hits the NMOS drain area in INV1. The black arrow indicates this case. During the charge collection period, a current pulse occurs in the sensitive area. If the pulse is larger than the critical charger, the logic values of the Q output will reverse from 0 to 1. The data store in latch is flipped totally (Q=0, D=1), and the wrong data will be held until the next clocking rising edge. This phenomenon is called SEU, and the possibility that SEU may occur in storage memory is represented by the “cross-section,” as shown in Equation 2.2, where the cross-section (σ) is equal to the total errors (n) in one device divided by the particle total fluence (N).

A single event flip is a single high-energy particle acting on a semiconductor device, causing an abnormal change in the logic state of the device. A single event upset is the most common and typical of multiple SEEs caused by space radiation, and it mainly occurs in data storage or instruction-related devices. The device error caused by a single event flip is a "soft error,” that is, it can be restored to the normal state by system reset, re-power on, or re-write.

These soft errors can be divided into single-bit upsets and multiple-bit upsets As shown in Figure 2.10, in the 8-bit configuration register, the default value changes from 10110010 to 10111010, as the rad arrow pointer the third bit changes from 0 to 1. This causes the configuration data to be wrong; 10110010 implies a write operation in the device instruction, but it changes to 10111010, which means read operation in device instruction, which causes the device operation to be wrong, but it can be recovered by a system reset. The second part is a multiple-bit upset, and the configuration bit in Figure 2.10 changes from 10110010 to 00111011. The last significant bit (LSB) and the most significant bit (MSB) together with the third bit reverse from the original logic value. As in one 4-LUTs, it has 4-bit input and 1-bit output. The configuration generated in the bitstream is 10110010, but when this device received irradiation and when multiple-bit upset occurred, the configuration changed to 00111011, and the function this LUTs represent is wrong. Finally, this may cause the device to run in error.

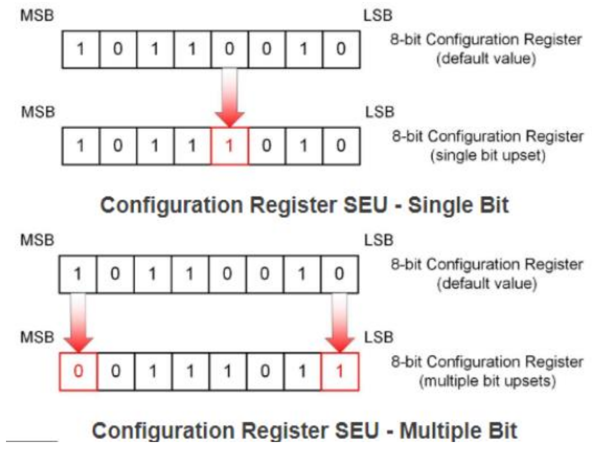

MSB
LSB
1 0 1 1 0 0 1 0
8-bit Configuration Register (default value)
MSB
LSB
1 0 1 1 1 0 1 0
8-bit Configuration Register (single bit upset)
Configuration Register SEU - Single Bit
MSB
LSB
1 0 1 1 0 0 1 0
8-bit Configuration Register (default value)
MSB
LSB
0 0 1 1 1 0 1 1
8-bit Configuration Register (multiple bit upsets)
Configuration Register SEU - Multiple Bit

Figure 2.10 Single Event Upset copyright Copyright @ NASA

Another common unit to measure the Soft Error Rate (SER) is Failure in Time (FIT), it also considered as the possibility for an SEU happened in electronic device. A one-time failure in one billion device hours is one FIT. Here is the example, the FIT value for a single chip electronic system is 10000, that means between per 114 years the whole system will get one failure. FIT can be considered as the cross-section (σ) multiply the fluence of a particle in the environment (ϕ)with 10^9. This equation is shown in equation 2.3.

2.4.2 TMR

TMR implies that the system deliberately configures duplicate parts and skills to improve its reliability. In computer science, TMR is sometimes referred to as triple-mode redundancy. As shown in Figure 2.7, there are three gates in the green rectangle; these three gates are same but constructed with different circuits. Each of these gates can perform the same function in this system. All the results of the gates will flow direction to the majority gate module, which works for voting to obtain the final results. In a normal situation, these three gates will generate the same output result and send it to the majority gate; then, the majority gate chooses any one to be the output, because all the three outputs from the three gates are correct. However, when this system is in an irradiation environment, and an SEE occurs in the system with only one gate, the wrong result will come from the device. However, if TMR is implemented here, such as the design in the green rectangle, if one of those three gates occurs in SEE, the other two will work normally with a high probability, because the probability that both sets of gates occur with SEE is very small, so the majority can vote to the right result by voting those sets of right results from those two normal gates.

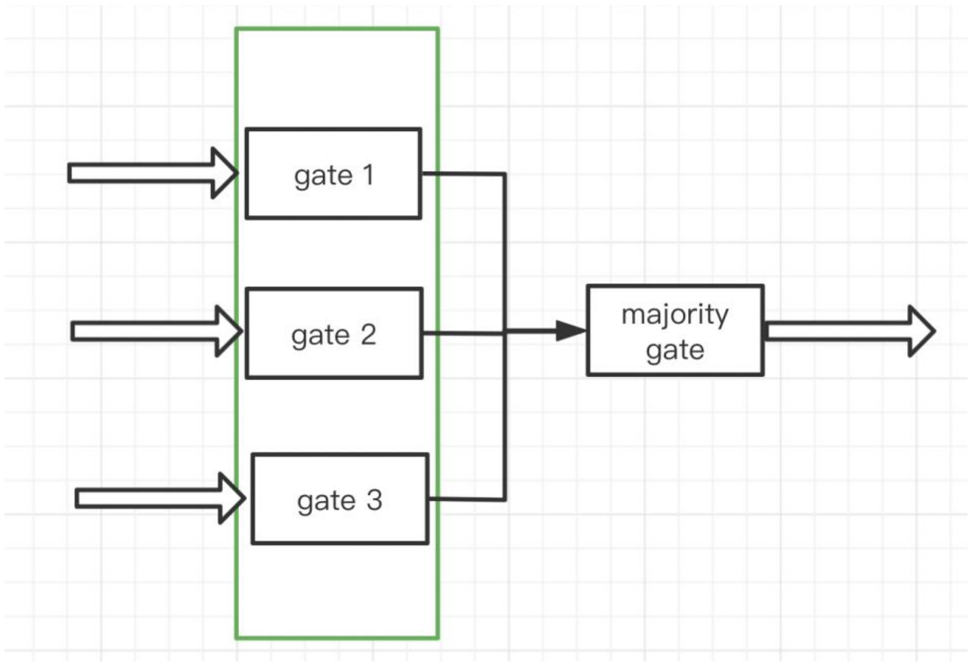


Figure 2.11 Triple modular redundancy

The TMR is a fault-tolerant form of N-modular redundancy. Three identical systems were used to perform the same function, and then through a majority-voting system, the output of the majority was taken as the final output. If only one of the three systems was damaged and the other two remained normally, the majority voting system used the two pairs of outputs and the result for those The TMR is a fault-tolerant form of N-modular redundancy. Three identical systems were used to perform the same function, and then through a majority-voting system, the output of the majority was taken as the final output. If only one of the three systems was damaged and the other two remained normally, the majority voting system used the two pairs of outputs and this vote result was considered to be the final output.

2.5 References

[1] S. U. Amin, K. Agarwal, and R. Beg, “Genetic neural network based data mining in prediction of heart disease using risk factors,” in Proc. IEEE Conf. Inf. Commun. Technol., Apr. 2013, pp. 1227–1231.

[2] Pedraza, A., Gallego, J., Lopez, S., Gonzalez, L., Laurinavicius, A., & Bueno, G. (2017, July). Glomerulus classification with convolutional neural networks. In *Annual conference on medical image understanding and analysis* (pp. 839-849). Springer, Cham.

[3] Bernal, J., Kushibar, K., Asfaw, D. S., Valverde, S., Oliver, A., Martí, R., & Lladó, X. (2019). Deep convolutional neural networks for brain image analysis on magnetic resonance imaging: a review. *Artificial intelligence in medicine, 95*, 64-81.

[4] Islam, Kh Tohidul, Ram Gopal Raj, and Abdullah Al-Murad. "Performance of SVM, CNN, and ANN with BoW, HOG, and image pixels in face recognition." *2017 2nd International Conference on Electrical & Electronic Engineering (ICEEE)*. IEEE, 2017.

[5] Petersson, Henrik, David Gustafsson, and David Bergstrom. "Hyperspectral image analysis using deep learning—A review." *2016 sixth international conference on image processing theory, tools and applications (IPTA)*. IEEE, 2016.

[6] S. Lee, K. Son, H. Kim and J. Park, "Car plate recognition based on CNN using embedded system with GPU," *2017 10th International Conference on Human System Interactions (HSI)*, 2017, pp. 239-241, doi: 10.1109/HSI.2017.8005037.

[7] Uchida K, Tanaka M, Okutomi M. Coupled convolution layer for convolutional neural network. Neural Netw. 2018 Sep; 105:197-205. doi: 10.1016/j.neunet.2018.05.002. Epub 2018 May 16. PMID: 29870927.

[8] Agarap A F. Deep learning using rectified linear units (relu)[J]. arXiv preprint arXiv:1803.08375, 2018.

[9] Yu, Dingjun, et al. "Mixed pooling for convolutional neural networks." *International conference on rough sets and knowledge technology*. Springer, Cham, 2014.

[10] Wohlhart P, Lepetit V. Learning descriptors for object recognition and 3d pose estimation[C] *IEEE conference on computer vision and pattern recognition*. 2015: 3109-3118.

[11] Deng L. The mnist database of handwritten digit images for machine learning research [best of the web][J]. *IEEE Signal Processing Magazine*, 2012, 29(6): 141-142.

[12] G. Cohen, S. Afshar, J. Tapson and A. van Schaik, "EMNIST: Extending MNIST to handwritten letters," *2017 International Joint Conference on Neural Networks (IJCNN)*, 2017, pp. 2921-2926, doi: 10.1109/IJCNN.2017.7966217

[13] Wu M, Zhang Z. Handwritten digit classification using the mnist data set[J]. Course project CSE802: Pattern Classification & Analysis, 2010.

[14] Robert Keim," What Is an FPGA? An Introduction to Programmable Logic," [Online]. Available: https://www.allaboutcircuits.com/technical-articles/what-is-an-fpga-introduction-to-programmable-logic-fpga-vs-microcontroller/

[15] Serrano, J "Introduction to FPGA design." (2008). [Online].
Available: http://www.jps-pcb.com/upfile/2017/02/20170204171513_891.pdf

[16] S. Trimberger, "FPGA Technology: Past, Present and Future," ESSCIRC '95: Twenty-first European Solid-State Circuits Conference, 1995, pp. 12-15.

[17] Linde, Arne & Nordström, Tomas & Taveniku, Mikael. (1992). Using FPLs to Implement a Reconfigurable Highly Parallel Computer. Selected Papers from: Second International Workshop on Field Programmable Logic and Applications. 705. 199-210. 10.1007/3-540-57091-8_45.

[18] I. P. Dugganapally, W. K. Al-Assadi, V. Pillai and S. Smith, "Design and Implementation of FPGA Configuration Logic Block Using Asynchronous Semi-Static NCL Circuits," *2008 IEEE Region 5 Conference*, 2008, pp. 1-6, doi: 10.1109/TPSD.2008.4562768.

[19] Christian, Schuck & Bastian, Haetzer & Becker, Juergen. (2009). An Interface for a Decentralized 2D Reconfiguration on Xilinx Virtex-FPGAs for Organic Computing. International Journal of Reconfigurable Computing. 2009. 10.1155/2009/273791.

[20] N. Fujieda, M. Takeda and S. Ichikawa, "An Analysis of DCM-Based True Random Number Generator," in *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 67, no. 6, pp. 1109-1113, June 2020, doi: 10.1109/TCSII.2019.2926555.

[21] ZCU102 Evaluation Board User Guide, UG1182(v1.6) June 12,2019. Xilinx. [Online]. Available: https://www.xilinx.com/support/documentation/boards_and_kits/zcu102/ug1182-zcu102-eval-bd.pdf

[22] Zynq UltraScale+ MPSoC Data Sheet: Overview, DS891(v1.8) October 2,2019. Xilinx. [Online]. Available: https://www.xilinx.com/support/documentation/data_sheets/ds891-zynq-ultrascale-plus-overview.pdf

[23] AXI Reference Guide, UG1037 (v4.0) July 15, 2017. [Online]. Available: https://www.xilinx.com/support/documentation/data_sheets/ds891-zynq-ultrascale-plus-overview.pdf

[24] Venieris, Stylianos I., Alexandros Kouris, and Christos-Savvas Bouganis. "Toolflows for mapping convolutional neural networks on FPGAs: A survey and future directions." arXiv preprint arXiv:1803.05900 (2018).

[25] Szegedy, Christian, et al. "Inception-v4, inception-resnet and the impact of residual connections on learning." *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 31. No. 1. 2017.

[26] Rehan Hameed, Wajahat Qadeer, Megan Wachs, Omid Azizi, Alex Solomatnikov, Benjamin C. Lee, Stephen Richardson, Christos Kozyrakis, and Mark Horowitz. 2010. Understanding Sources of Inefficiency in General-purpose Chips. In Proceedings of the 37th Annual International Symposium on Computer Architecture (ISCA '10). ACM, 37–47.

[27] Kenneth A. LaBel, Single Event Effects (SEEs) Specification Approach. [Online]. Available: https://radhome.gsfc.nasa.gov/radhome/papers/SEEspec.pdf

3. Architecture Design and Hardening Approach

This chapter introduces the MNIST architecture used in the project and how to implement it on the ZCU-102 board with SCEs and Streaming architectures in detail. Then, two harden TMR designs based on SCEs architecture are introduced with design ideas, detailed implementation steps, and related code.

3.1 Streaming and SCEs Architectures on FPGA

The MNIST CNNs are implemented on the 16-nm Fin-FET Zynq UltraScale+ MPSoC (Xilinx XCZU9EG-2FFVB1156 on the evaluation board ZCU102). It is composed of a processing system (PS) which uses a quad-core ARM A53 with programmable logic (PL). FPGA-based CNN accelerators can be classified into two types: streaming architecture and single computation engines (SCEs) architecture [1]. This study implemented both streaming and SCEs architectures and evaluated the architectural vulnerability of each by using a higher level of granularity which is accomplished by dividing the architecture into two parts, the PL and the PS in the FPGA instead of looking at each layer.

3.1.1 MNIST Architecture

The CNN in this project uses the standard MNIST data set. The standard MNIST is a data set of 28 × 28 pixels images of handwritten decimal digital from 0 to 9. For the beginning we write a python code to train a CNN by Keras with standard MNIST data set. The accuracy for this MNIST CNN is 90%, which means for each of the ten handwritten decimal numbers our network can correctly recognize nine decimal numbers from zero to nine.

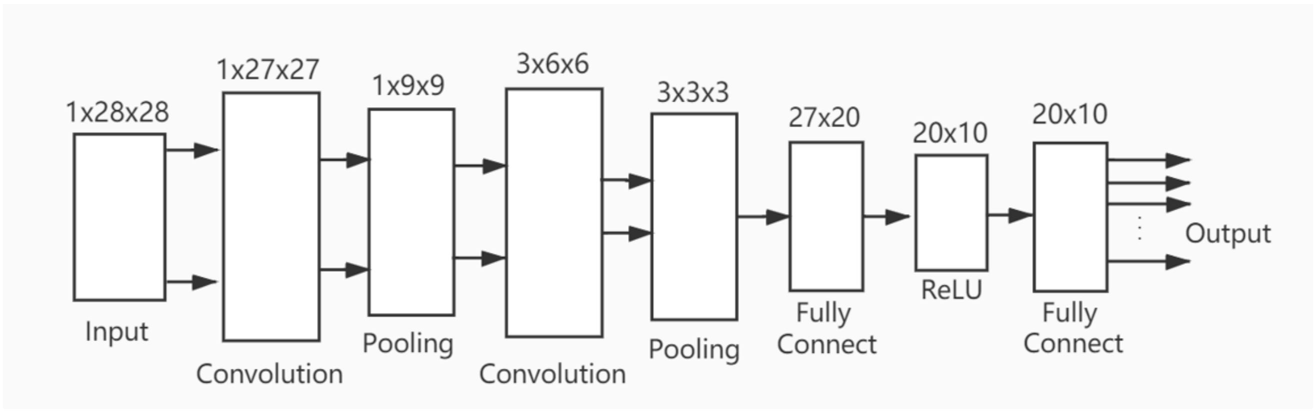


Figure 3.1 MNIST CNN topology

As shown in Figire 3.1, the MNIST CNN receives a data set of 28 × 28 pixels images of handwritten decimal numbers as a total of 784 pixels of input data. Then a round padding will add to the 28 × 28 matrix. In other words, a circle of 0’s will be added to the periphery of the matrix which makes the size of this input matrix 30 × 30. After this it will pass through the first convolution layer, CONV1, which is a 1 × 4 × 4 matrix. Then a 27 × 27 size new matrix will be generated as the new input matrix for pooling layers 1. The size of the first pooling is 3 × 3, and the function for those two layers is to find the max number value in each 3 × 3 matrix. After first max pooling layer the feature for the input is extracted to generate a 9 × 9 size matrix. Convolution layer2 CONV2 will scan this 9 × 9 matrix with 3 sets of 4 × 4 size matrices. The output for CONV2 are then 3 sets of 3 × 3 matrices. By using a flatten layer, those three matrixes will become 27 individual outputs, all of which will pass the 27 × 20 size fully connected layer multiply and adding bias. The result from fully connected layer1 will be the input for the ReLU activation layer, generating a set of linear activation responses. Those 20 individual responses will act as the last input for the fully connected layer2. At the end, 10 individual numbers will be the final output, with each one representing the possibility for the decimal number between 0 to 9. The highest value in those ten outputs will be used as the MNIST CNN identified result. Table 3.1 shows the MNIST CNN construction. Tt can be combined with the Figure 3.1, the topology for MNIST CNN, to better understand the data process.

Table 3.1 MNIST CNN construction

Input	784 pixels
Conv layer1	1 × 4 × 4 filter
Max pooling1	3 × 3 filter
Conv layer2	3 × 4 × 4 filter
Max pooling2	2 × 2 filter
Flatten layer	3 × 3 × 3 matrix inputs, 27 outputs
Fully connect1	27 inputs, 20 outputs
ReLU1	20 inputs, 20 outputs

Fully connect2	20 inputs, 10 outputs
Output	The image classification - the result can be any integer between 0 to 9

3.1.2 Streaming Architecture

The MNIST are implemented on a ZCU-102 FPGA with Streaming architecture, which consists of three parts. The first part is the PS which communicates with the host PC by UART. It can also send the final result to the host PC and the instructions to the controller in PL. The second part is the controller in the PL. It works as a decoder, for the instructions from PS. The last part is the CNN calculation part, which performs whole CNN calculation.

In the streaming architecture the entire CNN layers are placed into the PL part of the FPGA, including its convolutional layer, pooling layer, flatten layer, and fully connected layer. Each layer is implemented into different blocks of the FPGA and form the pipeline for the calculations. The block diagram of the streaming architecture CNN is shown in Fig 3.2. Please note that the PS is also included in the diagram. It reads the calculation results from the PL and sends it to terminal by a UART port.

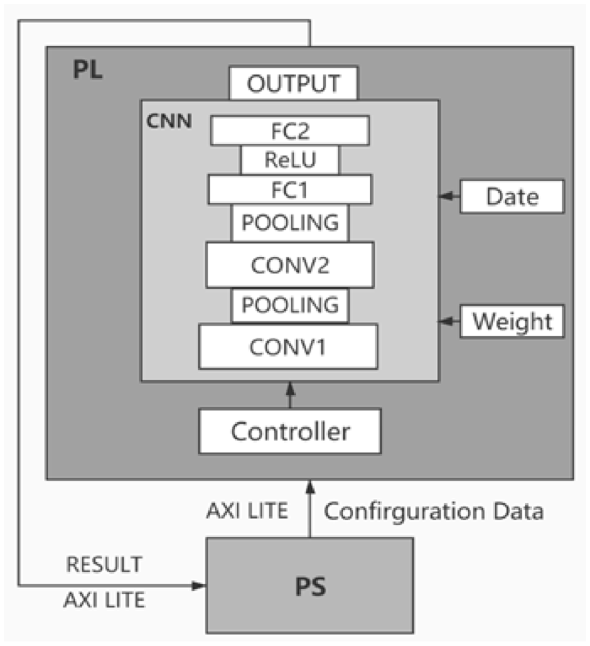


Figure 3.2 Streaming architecture design diagram

After receiving the instruction from the host PC, the PS side transmits it to the PL through AXI-lite, where the CNN and the controller are packaged together to form an IP package. This new AXI IP supports the AXI1-lite protocol (connected through AXI-interconnect and share the same clock). The PL part receives the 32 bits of control information, then the controller module unpacks and decoder it into the corresponding number of operations rounds (indicated by 8-1 bits). In the 32 bits operation code the operation interval (indicated by 10-9 bits) generates the corresponding start signal (0 bits).

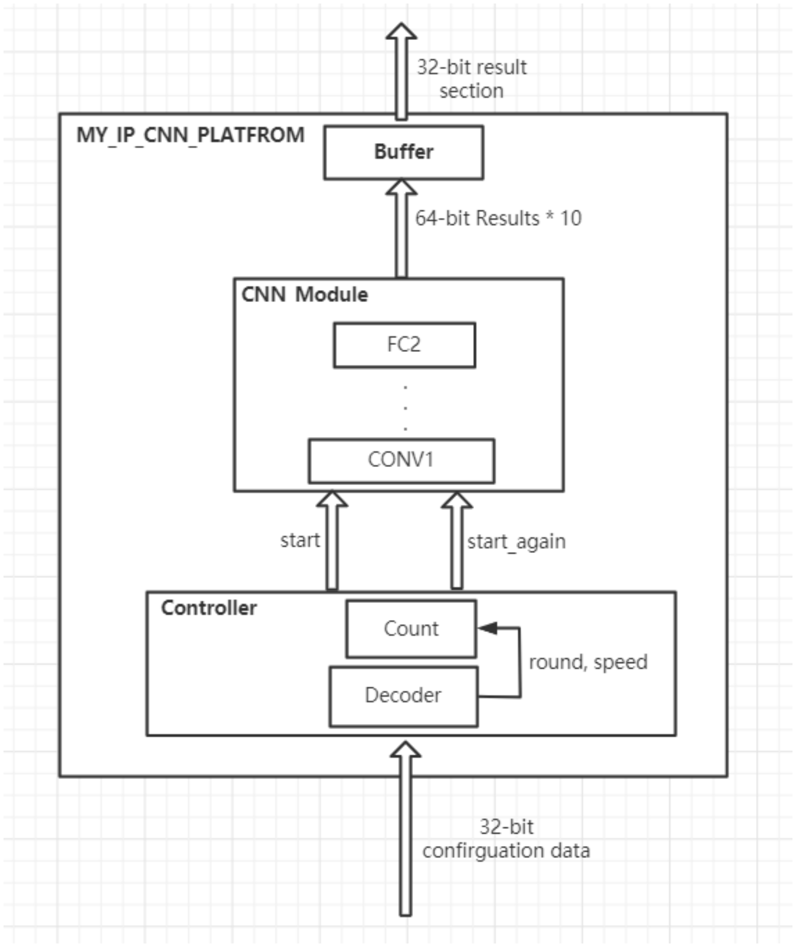


Figure 3.3 The structure of CNN_PLATFROM

Figure 3.3 shows the detailed structure for the CNN-PLATFROM module in the PL part. When the 32-bit configuration data is received and decoded by the controller module, as explained in the previous paragraph, the corresponding number of operations round and the operation interval will be controlled by the count block. The related code will be showed in chapter 3.3. The corresponding round and speed that was decoded will then be generated to homologous start and start_again signals, which are outputs from the controller module. Those two signals will then become two inputs for the CNN module. They will be used to start, refresh, and restart the process in calculation. The CONV1 module starts to extract the 28 × 28 data stored in the ROM (stored in 8 bits form) and the 4 × 4 weight (8 bits form) is also stored in ROM. These corresponding data and weights are required by the first convolutional layer. After obtaining the data it will first add a layer of padding according to the settings of the MNIST model. To elaborate, a circle of 0's will be added to the periphery of 28 × 28 matrix and the size will become 30 × 30. Then, the scanning will start according to the corresponding MNIST matrix parameters and it will generate the corresponding matrix and perform the convolution operation on the obtained matrix.

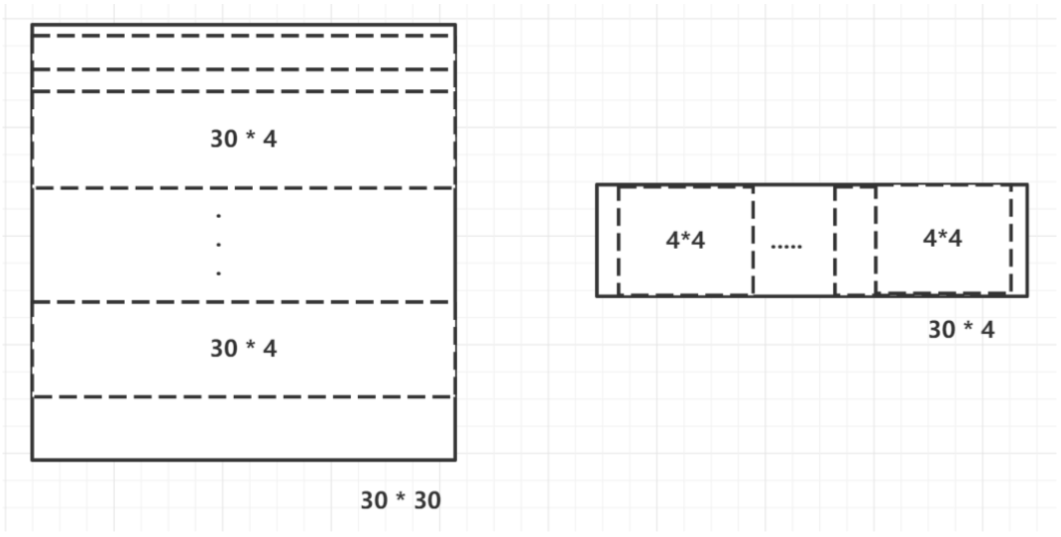


Figure 3.4 CONV1 Scanning process for Streaming architecture

In this design, the first round of data is divided into 27 groups. The Figure 3.4. shows the scanning process in CONV1. This 30 × 30 matrix will be scanned by a 30 × 4 matrix and it will be scanned 27 times in total - each time a 30 × 4 size of matrix is generated. The 27 groups of matrixes are operated in sequence, and each 30 × 4 matrix will be scanned by its weight matrix, the 4 × 4 matrix, as shown in Figure 3.4. In each 4 × 4 matrix all the data will first perform the multiplication operation. After, 16 individual products will be generated then those products will be added together. Each time the multiplication and adding is finished the CONV1 module will double the bit-width for each result from 32-bit to 64-bit in order to ensure the accuracy of the result. Each convolution result will then be scanned and stored in corresponding RAM.

The CONV1 module will wait for all the data scanning operations to be completed, then final results for CONV1 will be stored in temporal RAM. Then CONV1 results will be sent to a 3 × 3 max pooling, and the scan procedure is similar to the process in CONV1 - A 27 × 27 size matrix will be scanned by a 3 × 3 matrix. In each pooling matrix the biggest number in those 9 digital numbers will be chosen to be the result. The corresponding pooling matrix is generated in turn to get a new 9 × 9 matrix. Then, CONV2 is executed, and the process is the same as the process for CONV1. The related code will be showed in chapter 3.3.

Since the CONV2 layers have three sets of weight, the input data for CONV2 will be copied three times and will generate three sets of 9 × 9 matrices. Each 9 × 9 matrix will be scanned by the 4 × 9 scan matrix, then it will be divided into 6 parts. After that each 4 × 9 matrix will be scanned by the weight 4 × 4 array, and the same procedure in CONV1 will be processed in CONV2. Each output after this process will be re-arranged and stored in temporal RAMs. Three sets of 6 × 6 matrices will be generated as the output from the CONV2 layer.

Then, pooling layer2 of a 2 × 2 matrix size will follow. The process in pooling layer2 is same as pooling layer1. After pooling layer2 the system will generate three sets of 3 × 3 matrices as the input of the flatten layer. In the flatten layer, the matrix will be serial data, where 27 individual 64-bit data will produce a 1728-bit output. Fully connected layer1 (FC1) will use those 27 individual pieces of data to multiply with a 27 × 20 weight array. After that, 20 results will be generated, and all those results will add with corresponding bias. The results obtained from FC1 will enter the ReLU activation layer. The function for this layer is to find the result which is bigger than logic 0. If the result is bigger than logic 0, the output will be itself. If not, the result for the input logic number will be logic 0 (here, the ReLU activation function is simulated by hardware).

After the activation layer matrix, data will pass the final FC2 layer to get the result. It will then multiply with the FC2 array, which is size 20 × 10, the process in FC2 is same with FC1. In the end, the FC layers will get ten 64-bit binary number representing the probability of each decimal number from 0 to 9. The product will be divided into two 32bits and transmitted back to the PS through AXI lite, as shown in Figure 3.5. Each time the PS part will send the 32-bit address signal to the output buffer by AXI protocol, each address points to a corresponding RAM in output buffer. PS part will send the data in sequence to traverse each RAMs with count block in the output buffer, then all the results will be sent back to the ARM core in PS, each time with a 32-bit result section.

At the same time, the final data will be accompanied by a number of rounds of this operation. As shown in the output buffer structure, the count block will add logic 1 after each entire calculate done, and the cycle number will also be transmitted to the PS part. Then, the PS terminal will be transmitted through UART and displayed on the host PC. The host PC will also generate a file for recording, including the test result and the test information.

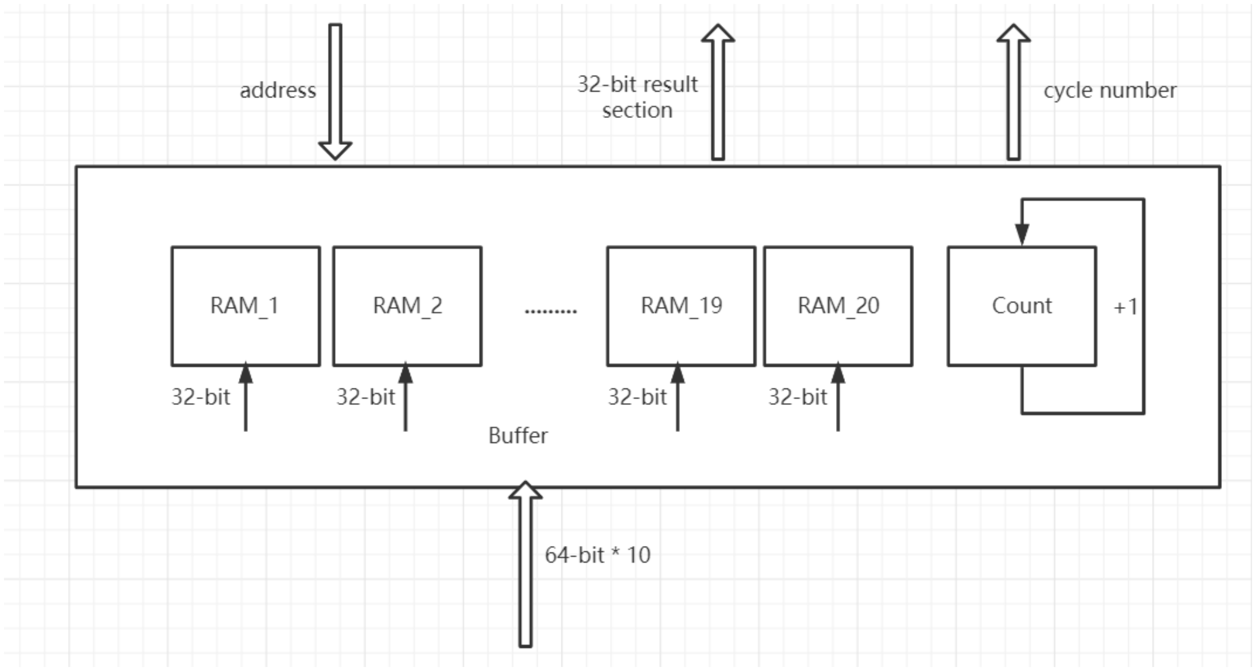


Figure 3.5 Output buffer structure

The PS side here does not participate in calculations, it only performs UART data transmission and send configuration data to PL part. According to the comparison with the MNIST on the PC side, the accuracy for this Streaming architecture version implemented on a FPGA is after 4 decimal places.

3.1.3 SCEs Architecture

SCEs architecture consists of four parts. The first is the logic control unit (LCU) which controls the whole CNN as it receives the results from the PL or sends data to PL. The second part is memory control unit (MCU). It accesses off chip memory or in chip cache to read input weight and data. The third part is the memory optimization unit (MOU) [2-4]. It works to make the AXI data_width adapt to the process engines (PE). In other words, the data_width PE used for the calculations is different than the AXI protocol. The purpose of the MOU is to help handle this difference. The last part is the PEs, which is one of the the most important parts in the PL section. It is consisted by some MACs (Multiply Accumulate), and the number of MACs and PEs is decided by design and FPGA utilization. Those MACs work to multiply and add, whereas the PE works for doing convolution calculations.

In the SCEs structure the CNN not only uses the PL to do the calculations, but also the PS in the FPGA will help perform the calculations, as shown in Fig 3.6. In this design, an array of PEs is implemented to carry out the calculation-extensive operations in convolution layer1 and convolution layer2. There are eight PEs implemented in the design. The calculations for the less calculation-intensive layers, such as the pooling layer, the flatten layer, and the fully connected layer are completed in the PS part.

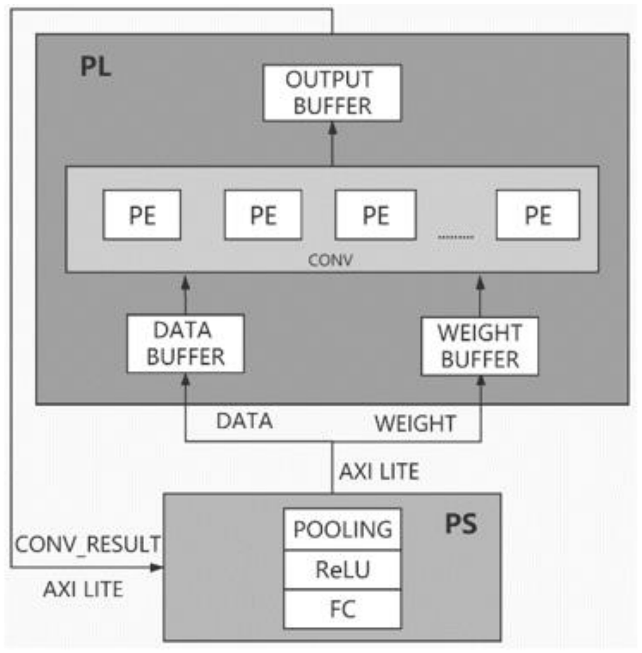


Figure 3.6 SCEs architecture design diagram

The data path for SCEs is as follows. First, the ARM core on the PS side will use the MCU to pre-set data that is size 28 × 28. Because the purpose of the project is to test the reliability of the SRAM-based FPGA CNN, the irradiation test is aimed at SoC chip in ZCU-102, so this system will not use Double Data Rate Synchronous Dynamic Random-Access Memory (DDR-SDRAM) in the FPGA. The ARM core will perform the padding process, a circle of logic 0 will be add on the periphery, and the size of new matrix will be 30 × 30. Then this matrix will be scanned by a 4 × 4 matrix, as shown in Figure 3.7. This 4 × 4 matrix will scan the entire 30 × 30 matrix horizontally and vertically. Each time it scans the LCU will send 8 × 4 × 4 matrix data to the PL side through the AXI protocol 32-bits at a time to the input buffer for a total of 16 × 8 times. Then the input data buffer will combine all the data together. This work is performed by the MOU in the SCEs architecture. Those 4 × 4 matrixes will be sent to each PE, and the MACs in each PE will do the calculations with each 4 × 4 data matrix and each 4 × 4 weight matrix. They will multiply each relative binary number and then add them, resulting in one 64-bit binary result.

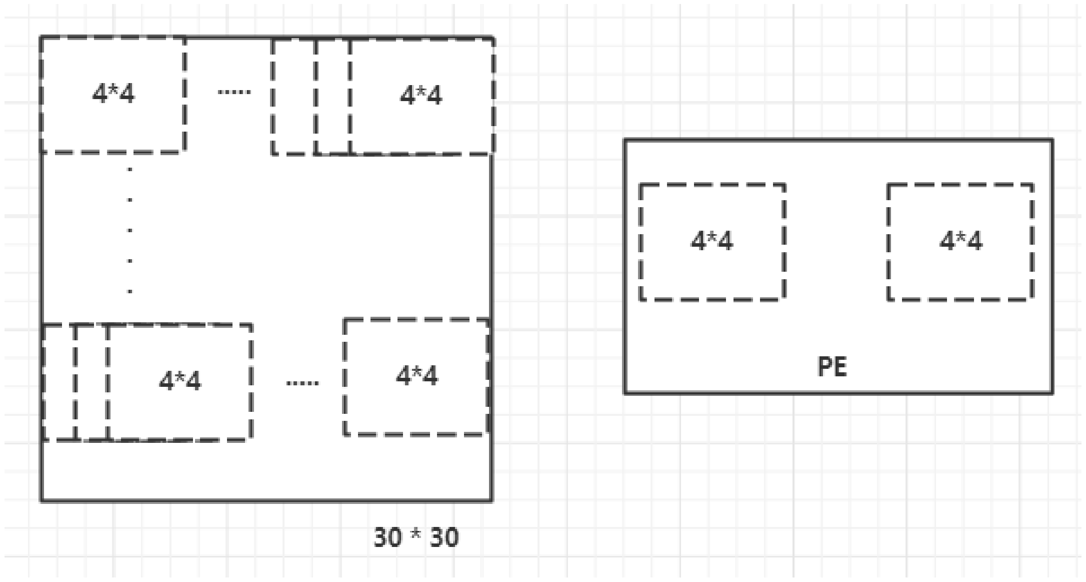


Figure 3.7 CONV1 scanning process in SCEs architecture

In the SCEs architecture, PE and MOU are also packaged together to generate an AXI-lite IP package. This uses an AXI-lite protocol to communicate between the PS and the PL. Since a total of 8 PEs are placed on the PL in the project design, each time the PS will transmit 8 sets of 32-bit binary number to the PL. The input buffer in the PL will obtain this data, and each 4 × 4 data matrix will be allocated to every PE. The weight buffer will also send the 4 × 4 weight array to the PE module. As shown in Figure 3.8, the weight matrix and the data matrix are the input for every PE. The PE module has two clocks, one is CLK_MAC which is the same clock with system clock, and the other is CLK_PE. This clock will assert every two clock cycles of CLK_MAC. The reason for CLK_PE in the design is because there are eight MACs in one PE, and each MAC works for 2 binary numbers, which are the data and weight values. Every CLK_MAC cycle it can calculate 16 sets of data and weight values, and there are a total of 32 weight and buffer. So it needs two rounds to calculate all the data and weight values. There are also 7 carry look adders (CLA) in each PE module, and each CLA works to add the value from every MAC, thus increasing the speed of calculation. The final CLA will also add the result of the previous round. The output bit width here will then be expanded to 64-bit to make the result more accurate.

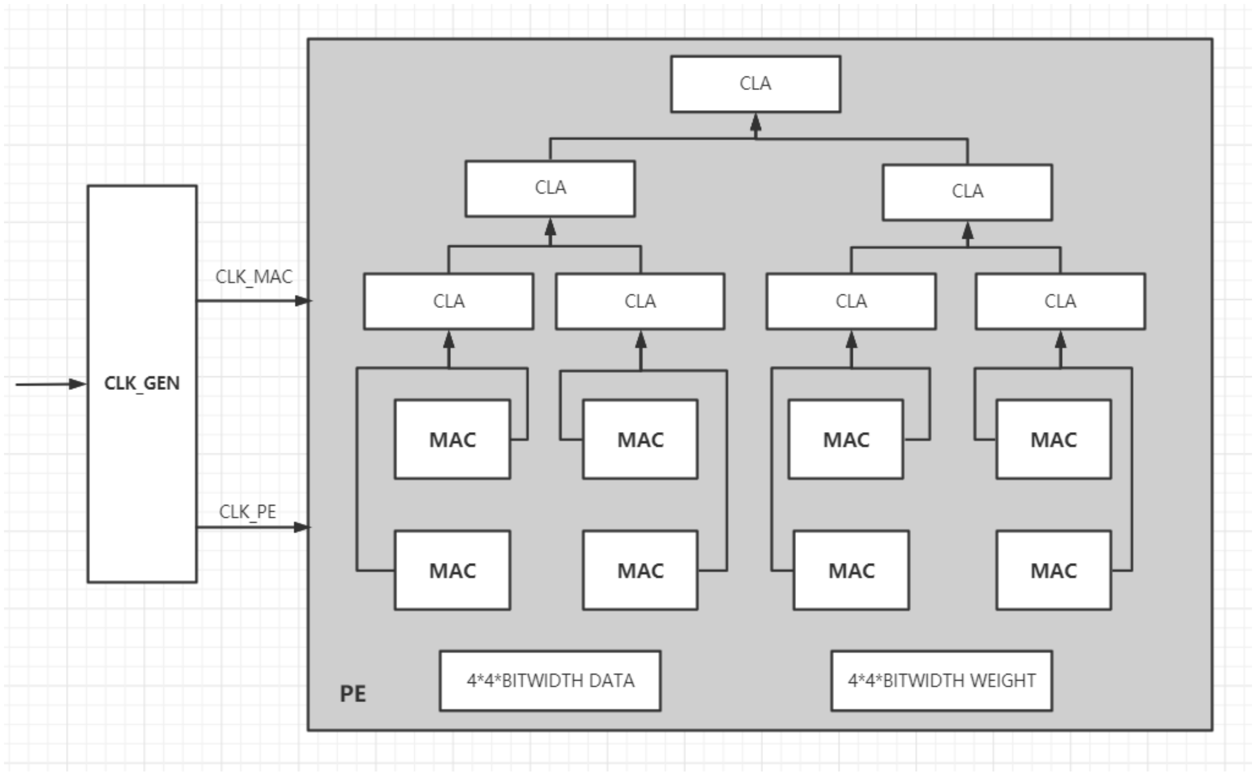


Figure 3.8 PE module structure

Each time the eight PEs complete an operation, eight results will be generated. These results will be passed back to the PS through the output buffer and then through AXI-lite, as shown in Figure 3.9. The output for each PE will then be sent to the output buffer module, totaling 8 sets of 64-bit binary numbers. Each 64-bit number will be stored via two 32-bit sections by two registers, then the PS will send the address signal to check and read the required data. In the output buffer each address refers to a RAM point that stores the output result, and each 64-bit result will be sent by two 32-bit sections. A done signal will then be sent back to the PS if the done signal is logic 1. The PS will also send the next group of data and weights to PL. With this process on the PS side, a new 27 × 27 matrix will be created sequentially and this matrix will be combined with each convolution round result. Then the new matrix will perform a 3 × 3 pooling operation in the ARM core to obtain a new 9 × 9 matrix and it will be used as the CONV2 data matrix. This pooling process is also the max pool operation, choosing the biggest binary number in every 9 numbers.

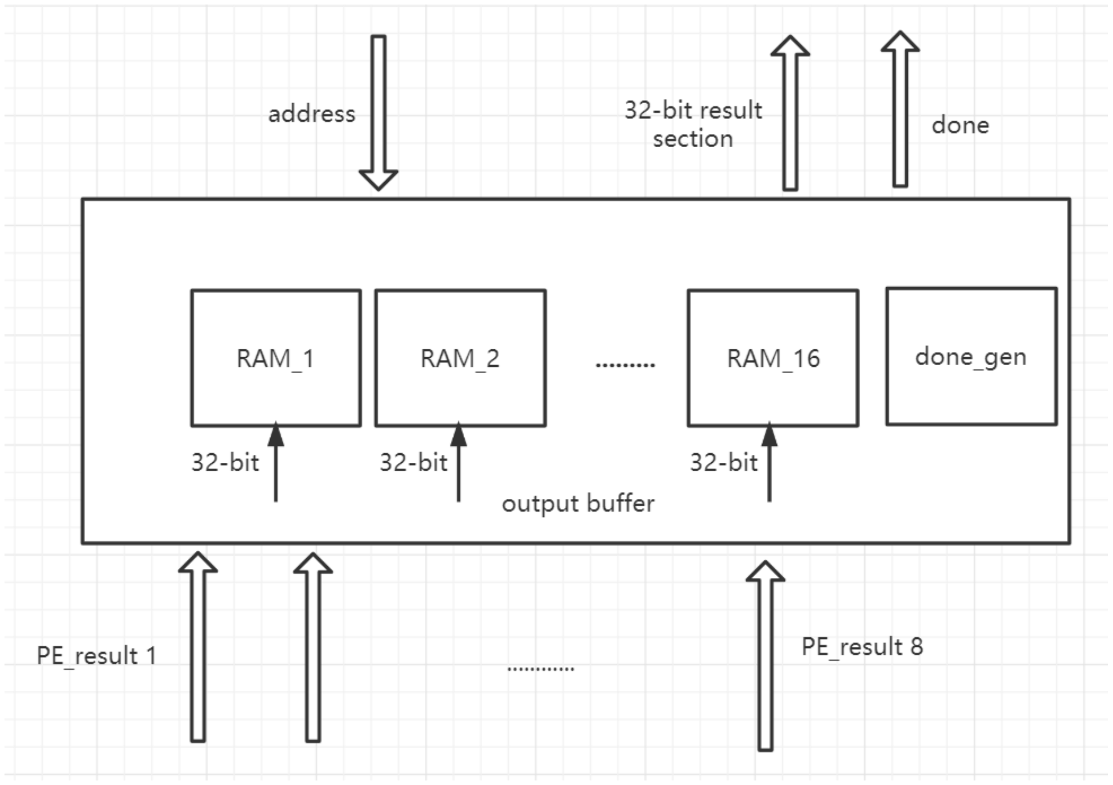


Figure 3.9 Output buffer module structure

After that, the PS side will scan and select the convolution layer2 matrix in turn through the LCU. This process is the same as the process in convolution layer1. The matrix will then be sent to the PL side for the second round of convolution operation. Once again, the calculation steps are similar to that of the operations in the convolution layer1. The required data is distributed to eight different PEs for independent operations. In each PE, every MAC will work as a multiplier and an adder. After the results are obtained, they are returned to the PS side through AXI protocol by output buffer, and the LCU on the PS side will reassemble a new matrix as the next layer input. When convolution layer2 completes the operation, there a 3 set of 6 × 6 matrix are generated in ARM core. The newly generated 3 × 6 × 6 matrix will be used as the input for pooling layer2, and it will use the 2 × 2 matrix to scan this input matrix, in order to find the biggest number in each 2 ×2 matrix. The outputs for pooling layer2 are three set of 3 × 3 matrices, totaling 27 individual binary numbers Those three matrixes will pass through the flatten layer and these three sets of matrices will be sequentially tiled into a total 27 numbers with 64-bit binary data.

The output from the flatten layer will enter the fully connected layer. In FC1 layers, the data will be multiplied with the 27 × 20 matrix and then added to the corresponding bias to obtain 20 individual 64-bit binary numbers. These 20 outputs, acting as the input of FC2, will be calculated with the 20 × 10 weight matrix and added with bias to get the final result. Results are composed of 10 numbers, which represent the probability of different decimal numbers from 0 to 9. All these results will be transmitted to the host PC via UART via the ARM core in the PS, and the PC will store and record every result in a plain text file.

In the SCEs architecture the PL side is responsible for the CONV1 and CONV2 layer operations, which require a large number of calculations. Because every module is processed in parallel in the PL, the overall speed is accelerated by the characteristics of PE parallel operations. While the ARM core on the PS side performs operations with a small amount of data, such as pooling layer, flatten layer and activation layer, those layers just require small amount of computing operations. So, it can be easily running on PS part. To compare the result between the PC software and SCEs architecture, the accuracy of SCEs architecture is 4 digital decimal places.

3.2 Hardening Approach for SCEs

In this thesis, two types of TMR algorithms were designed with the same hardware architecture. One is temporal TMR, and another is spatial TMR. TMR is used improve the radiation tolerance of the CNN designs. Those two hardened designs achieve the same effect with TMR redundancy, also different with traditional TMR or selective TMR [5,6]. Two TMR approaches only slightly increase the computation time without increasing the resources used in the FPGA, as shown in Table 3.1. For those four designs, the streaming architecture is using a 50MHz clock, and all SCEs designs are using a 100MHz clock, so that has made streaming architecture takes 3 seconds longer to run 100 rounds than all SCEs versions. The number of FFs and LUTs for the TMR design and unhardened design are same, this is because in those two TMR designs, only way to use the PEs for temporal TMR is to use the same PE to do the calculation three times with the same weight and data features. So, the running time for 100 rounds increased from 1.4s to 1.6s. For spatial TMR, same weight and data feature matrix will be calculated by three different PEs. Here, are only 8 PEs being in the SCEs architecture, and 3 different PEs will be used for one operation, so 8 PEs can do 2 sets of weight and data. This causes the timing consumption to be bigger than temporal TMR. The running time for 100 rounds will increase up to 1.7s due to this.

Table 3.1 UltraScale + MPSoC Resource utilization and processing time

Architecture	**hardening**	**FFs**	**LUTs**	**DSPs**	**Time (100 rounds)**
Streaming	Unhardened	45k (8.3%)	216k (78.8%)	0.72k (28.6%)	4.98s
SCEs	Unhardened	18k (3.36%)	21k (7.92%)	0	1.4s
SCEs	Temporal redundancy	18k (3.36%)	21k (7.92%)	0	1.6s
SCEs	Spatial redundancy	18k (3.36%)	21k (7.92%)	0	1.7s

3.2.1 Temporal TMR

Temporal TMR is realized by performing each calculation three times with the same PE unit. The temporal TMR design is shown in Figure 3.4. The embedded processor in the PS will control the PE units to carry out the calculations three times in sequence. The PS part will send the same data feature and weight matrix to each PE three times. This means, under normal circumstances, those three operations will generate same result. Those three results will then be sent back to the PS part in turn, and the voter in the PS will choose a result that is shown at least two out the three times. Ideally, these three results should be same, but under irradiation circumstances latches can be changed by the occurrence of SEU. This will result in error. So, if any result of those three are incorrect, the voter will work and choose the same two result as the final result to form the new matrix. This voter system will work for both CONV1 and CONV2.

For traditional TMR it will add the same circuit to the module that it is designed to reinforce. For example, in the SCEs architecture two extra PEs will be add to each PE, thus the total number of all PEs will become 24. This will effectively triple the utilization for logic resource usage. This temporal TMR design is similar to traditional TMR by adding additional redundant virtual circuits to perform the same operation three times. Whereas in temporal TMR one PE will process same calculation three times. This just increases the corresponding computing time, as shown in Table 3.1. The utilization for Temporal TMR does not change, and that means that the circuit area for traditional SCEs architecture and Temporal TMR is the same. For the timing consumption, it increases from 1.4s to 1.6s for each 100 cycle runs - a relatively small increase.

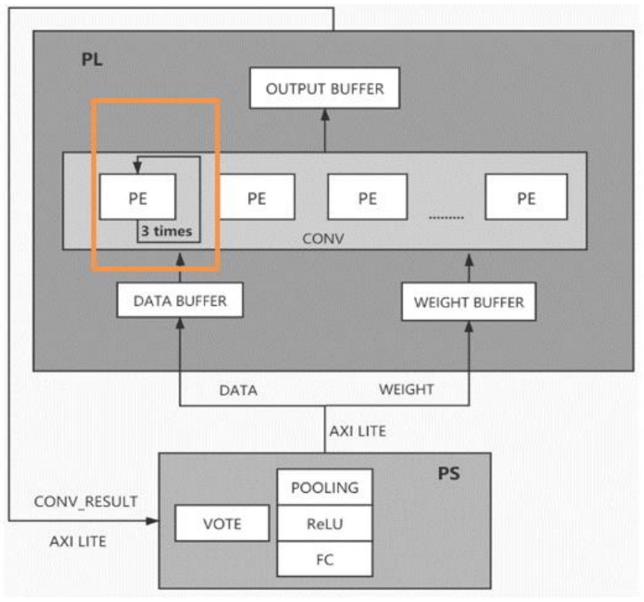


Figure 3.10 Temporal TMR design diagram

Compared with the original SCEs version, the PE in the PL will still perform the convolution operation of each set of data in turn. However, when the operation is over the input buffer will not read the next set of operation data. The PS side will repeatedly send the data of this operation to the buffer to refresh the buffer and to perform the same operation again to obtain the result. This behavior will be repeated three times for each PE.

Each PE will also perform three operations for the same data, and the results obtained each time will be sent back to the PS side. The voter on the PS side will compare these three sets of data and will select the result that occurs at least two times. This final result will then be put into the newly generated matrix. At the same time, the group of the PE that caused the error will be captured and recorded. This method of performing the same operation three times on the same PE is executed in CONV1 and CONV2 layer.   Through the reuse of single PE being used three times the same effect as traditional TMR has been achieved.

3.2.2 Spatial TMR

Spatial TMR is realized by using three PE units in parallel to carry out the same calculation. When a new input enters the CNN, three different PE units are assigned to carry out the same calculation. The diagram for Spatial TMR is shown in Figure 3.5. In the orange rectangle, three PEs will process the same calculation in parallel. A set of 4 × 4 data matrices and 4 ×4 weight matrices will be multiplied and added in those three PEs. Afterwards, each result for each PE will be sent back to PS part through the output buffer. Then the ARM core in the PS part will vote on the three copies of the calculation results and select the majority one to form the new matrix. This spatial TMR process is implemented in CONV1 and CONV2. Please note that the same number of PE units are used in this hardening approach. Therefore, the same hardware resources were used in this design, but with longer processing time up to 1.7s.

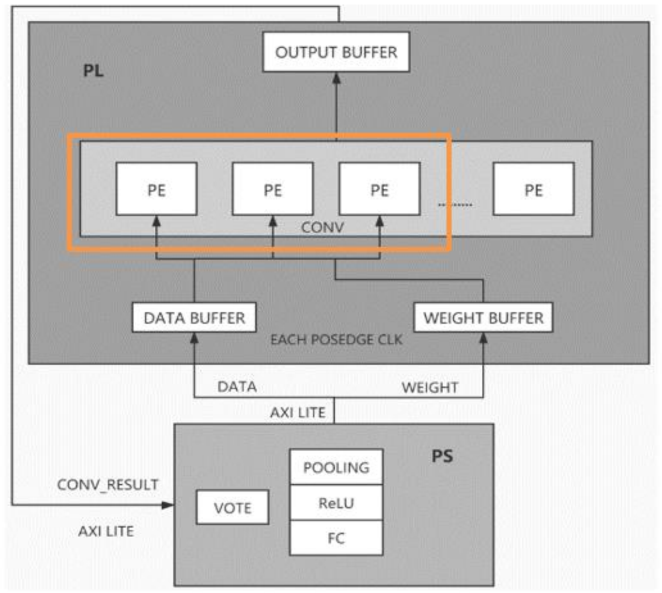


Figure 3.11 Spatial TMR design diagram

For spatial TMR the first PS part will send 2 sets of data matrices and weight matrices to PL part. The input buffer receives the input data, and the MOU will not distribute different data and weight matrices to each PE as in the original SCEs design. Instead, one set of the same data is divided into three different PEs for calculations. The first group of data will be sent to PE1 as well as PE2 and PE3. Then PE4 to PE6 will process the second set of data and weight. In spatial TMR every three PEs will get the same data, and every cycle will calculate six results that will be send back to PS part. The voter located on the PS side will compare these three data sets and select the result that was calculated identically at least two times. At the same time, if a SEU happened in the calculation process, it will also capture and record the PE that caused the error. After that, the PS will send the new data into the data buffer, and the three PEs will perform the same operation in turn. After completing the calculations of the entire convolutional layer, the PS side will vote to select the final result and combine it into a new matrix. CONV1 layer will generate a 27×27 size matrix as the final result, and CONV2 it will generate three set of 6×6 matrix results.

In spatial TMR, by using three different PEs to perform the same operation for the same data, the PE is multiplexed from the perspective of space and time, thus achieving an effect similar to TMR. This is same as traditional TMR, but it does not add extra PEs for redundant use. This design method turns a PE into a virtual PE, then uses those virtual PEs to process same calculation. Another design approach could be implemented with more PEs for this spatial TMR, but then more FPGA resources are needed, with higher processing speeds. In this thesis, both streaming and SCEs architectures implemented MNIST CNNs on the Xilinx UltraScale+ MPSoC FPGA. The resource utilization is shown in Table 3.1. It is important to note that SCEs designs only used one third the number of the FFs, and one tenth the number of the LUTs compared to the steaming architecture design. The temporal TMR design uses the same resources as the spatial TMR, since they are using the number of PEs. The calculations were done three times in sequence by the same PE for the temporal TMR, while the calculations were done by three PEs concurrently for the spatial TMR. Since spatial TMR uses more PEs for each step of calculation, the processing time is slightly longer than that of the temporal design.

3.3 Reference

[1] Venieris, Stylianos & Kouris, Alexandros & Bouganis, Christos. (2018). Toolflows for Mapping Convolutional Neural Networks on FPGAs: A Survey and Future Directions. ACM Computing Surveys. 51. 10.1145/3186332.

[2] W. Li *et al.*, "Soft Error Mitigation for Deep Convolution Neural Network on FPGA Accelerators," *2020 2nd IEEE International Conference on Artificial Intelligence Circuits and Systems (AICAS)*, Genova, Italy, 2020, pp. 1-5.

[3] K. He, X. Zhang, S. Ren, and J. Sun. 2016. Deep Residual Learning for Image Recognition. In 2016 *IEEE Conference on Computer Vision and Pattern Recognition* (CVPR). 770–778.

[4] Ma, Yufei, et al. "ALAMO: FPGA acceleration of deep learning algorithms with a modularized RTL compiler." *Integration* 62 (2018): 14-23.

[5] F. Libano *et al*., "Selective Hardening for Neural Networks in FPGAs," in *IEEE Transactions on Nuclear Science*, vol. 66, no. 1, pp. 216-222, Jan. 2019.

[6] F. Libano, *et al*., "Understanding the Impact of Quantization, Accuracy, and Radiation on the Reliability of Convolutional Neural Networks on FPGAs," in *IEEE Transactions on Nuclear Science*, vol. 67, no. 7, pp. 1478-1484, July 2020.

3.4 Appendix

(1) Part of Controller code:

```
// from_ps configuration data
   // 0x0000 0001
   // 0x0000 0002;
   // 0x0000 0006;  run 100 times
   // 0x0000 000a;  run 500 times
   // 0x0000 0012;  run 1000 times
   always@(posedge clk) begin
      if(!rst) begin
         rounds_number<=0;
         dont_stop<=0;
      end
      else
         case(from_ps)
            32'd6: rounds_number<=13'd100;
            32'd10: rounds_number<=13'd500;
            32'd18: rounds_number<=13'd1000;
            default: rounds_number<=13'd0;
         endcase
   end

   // each round 50clk
   initial begin
      count_50<=0;
      rounds<=0;
   end
```

```
// count_cycle 50 can be decided
always@(posedge clk) begin
    if(!rst)
        count_50<=0;
    else if(count_50==8'd60)
        count_50<=0;
    else if(start_reg_1)
        count_50<=count_50+8'd1;
end

// logic for start_again signal
always@(posedge clk) begin
    if(!rst)
        start_again<=0;
    else if(rounds==rounds_number)
        start_again<=0;
    else if(count_50==8'd60)
        start_again<=1;
    else
        start_again<=0;
end
// logi for round_number
always@(posedge clk) begin
    if(!rst)
        rounds<=0;
    else if(rounds==rounds_number)
        rounds<=rounds_number;
    else if(count_50==8'd60)
        rounds<=rounds+1;
end
```

(2) Part of CONV1 scanning process code:

```
//generate data_array_with padding
    genvar c,d;
    generate
        for(c = 0; c < DATA_HEIGHT+2; c = c + 1) begin
            for(d = 0; d < DATA_WIDTH+2; d = d + 1) begin
                if((c < 1) || (c > DATA_HEIGHT)) begin
                    assign dataArrayWithPadding[c][d] = 0;
                end
                else if(d < 1 || d > DATA_WIDTH) begin
                    assign dataArrayWithPadding[c][d] = 0;
                end
                else begin
                    assign dataArrayWithPadding[c][d] = data_fature_array[c - 1][d - 1];
                end
            end
        end
    endgenerate
    //generate data_ce1 array; R1_WIDTH = 27; F_HEIGHT = 4; DATA_WIDTH = 28
    genvar e,f,g;
    generate
        for(e=0; e<R1_HEIGHT; e=e+1) begin
            for(f=0+e; f<e+F_HEIGHT; f=f+1) begin
                for(g=0; g<DATA_WIDTH+2; g=g+1) begin
                    assign ce1_array[e][((f-e) * (DATA_WIDTH+2) + (g))* BITWIDTH + BITWIDTH-
1: ((f-e) * (DATA_WIDTH+2) + (g))* BITWIDTH] = dataArrayWithPadding[f][g];
                end
            end
        end
   endgenerate
```

(3) Part of Output buffer code in IP package

```
// Implement memory mapped register select and read logic generation
 // Slave register read enable is asserted when valid address is available
 // and the slave is ready to accept the read address.
 assign slv_reg_rden = axi_arready & S_AXI_ARVALID & ~axi_rvalid;
 always @(*)
 begin
       // Address decoding for reading registers
       case ( axi_araddr[ADDR_LSB+OPT_MEM_ADDR_BITS:ADDR_LSB] )
         5'h00   : reg_data_out <= cnn_result[639:608];
         5'h01   : reg_data_out <= cnn_result[607:576];
         5'h02   : reg_data_out <= cnn_result[575:544];
         5'h03   : reg_data_out <= cnn_result[543:512];
         5'h04   : reg_data_out <= cnn_result[511:480];
         5'h05   : reg_data_out <= cnn_result[479:448];
         5'h06   : reg_data_out <= cnn_result[447:416];
         5'h07   : reg_data_out <= cnn_result[415:384];
         5'h08   : reg_data_out <= cnn_result[383:352];
         5'h09   : reg_data_out <= cnn_result[351:320];
         5'h0A   : reg_data_out <= cnn_result[319:288];
         5'h0B   : reg_data_out <= cnn_result[287:256];
         5'h0C   : reg_data_out <= cnn_result[255:224];
         5'h0D   : reg_data_out <= cnn_result[223:192];
         5'h0E   : reg_data_out <= cnn_result[191:160];
         5'h0F   : reg_data_out <= cnn_result[159:128];
         5'h10   : reg_data_out <= cnn_result[127:96];
         5'h11   : reg_data_out <= cnn_result[95:64];
         5'h12   : reg_data_out <= cnn_result[63:32];
         5'h13   : reg_data_out <= cnn_result[31:0];
         5'h14   : reg_data_out <= count_number;
```

```
        5'h15   : reg_data_out <= slv_reg21;
        default : reg_data_out <= 0;
      endcase
end

// Output register or memory read data
always @( posedge S_AXI_ACLK )
begin
  if ( S_AXI_ARESETN == 1'b0 )
    begin
      axi_rdata  <= 0;
    end
  else
    begin
      // When there is a valid read address (S_AXI_ARVALID) with
      // acceptance of read address by the slave (axi_arready),
      // output the read dada
      if (slv_reg_rden)
        begin
          axi_rdata <= reg_data_out;     // register read data
        end
    end
end
```

4. EXPERIMENTAL AND RESULT

The radiation experiments were conducted at beamline 2C, TRI University Meson Facility (TRUMF). The proton energy was 105 MeV. The FPGA implemented with the streaming architecture design was irradiated with a total fluence of 8.7×10^{10} protons/cm^2, and the SCEs architecture design was irradiated with a total fluence of 3.6×10^{10} protons/cm^2. Subsequently, two TMR designs were also evaluated using the same ZCU-102 board with a fluence of 6.4×10^{10} and 5.1×10^{10} protons/cm^2, respectively. The total fluence was approximately 23.8×10^{10} protons/cm^2.

4.1 Experimental Setup

First, we designed and built an experimental system based on the needs and characteristics of the test. A schematic of the test system is shown in Figure 4.1. First, a host PC was placed in the control room. Herein, we used our personal laptop as the host PC, which could read and write operations on it. The laptop controlled the desktop in the irradiation room via TeamViewer. Then, the ZCU-102 FPGA and a desktop were put in the irradiation room for restarting and configuration of the FPGA (write or write bitstream). In the proton irradiation test, if a crash occurred in the FPGA, the FPGA was frozen, and it will be restarted for the next test, which also seems to the error situation. The desktop in the irradiation room writes a bitstream to the ZCU-102 board by the USB port. It was also responsible for receiving and storing experimental data in the specified folder, the MNIST CNN results in every test round, and the number of rounds recorded in the selected file. At the same time, a person in the control room recorded the experimental data manually every time the FPGA hung and showed an error.

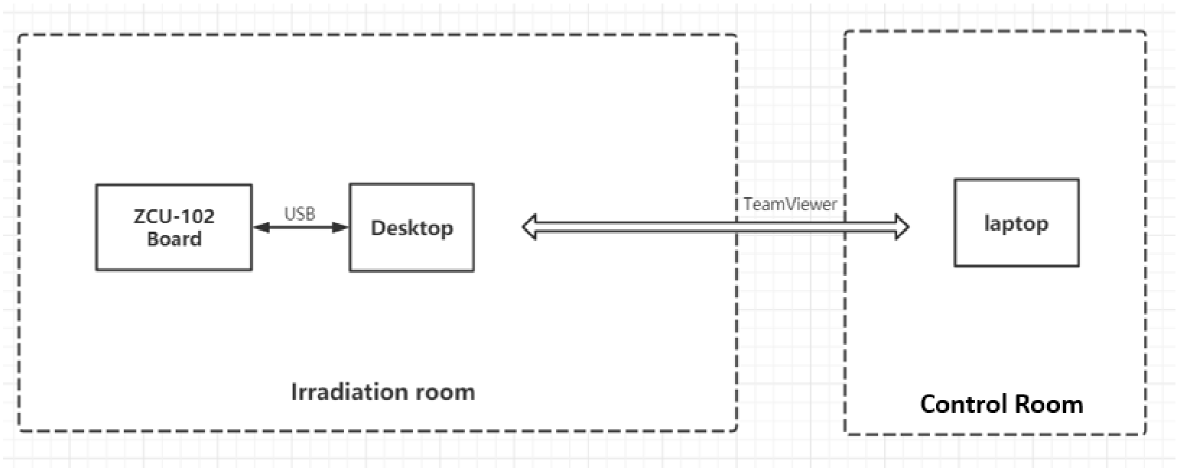


Figure 4.1 The schematic of the testing system for proton exposures test

In the experiment, we divided the encounters into two situations. The first is called a hang. When hang happens, the FPGA, which participates in the test, crashes, and cannot continue to perform CNN operations; under hang circumstances, the FPGA is frozen. The second is called an error. When error happens, the FPGA participating in the test continues to function, communicates with the desktop in the irradiation room, send configuration data and receive results. However, the result for this MNIST CNN is wrong, which means that the entire system cannot work adequately. For these two situations, we recorded the total fluence at the time of their occurrence, and the total fluence was recorded as the fluence for generating the hang and error. Subsequently, we restarted the FPGA by reconfiguring it. When the FPGA hung, we stopped the experiment to record the time and total fluence; the number of cycles the MNIST CNN ran was also recorded. Then, we powered off the FPGA and switch it on again, and a new bitstream was written to the board. For the error situation, when an error occurred, we recorded the time, total fluence, and the running cycle number. Then, we waited for 90 to 100 rounds, observed whether the result was corrected, or the system still sent the wrong result or the error changed. This type of error situation is recorded according to different phenomena. Then, we turned off the portable power, which powers the FPGA, and restarted it again. The FPGA was powered on, and the bitstream was rewritten.

We used the same set of test data, weight, and bias for loop testing regarding the experimental test data. First, bitstream was wrote to FPGA, and instructions were sent to perform image recognition operations through the PS. Then, the turn-on proton accelerated to irradiate the FPGA board. It then waited for the occurrence of hang or error to record total fluence. The irradiation was stopped, and the entire process was repeated.

In the experiment, we used the total fluence as the total dose of proton irradiation. To calculate the cross-section as the possibility that SEU may occur in storage memory, an SRAM-based FPGA was used. The calculation equation for the total fluence and error, hang, and total failure cross-section is expressed using Equations 4.1–4.3.

$$\sigma_E = \frac{n_{Error}}{N} \tag{4.1}$$

$$\sigma_H = \frac{n_{Hang}}{N} \tag{4.2}$$

$$\sigma_{Total} = \frac{n_{Total}}{N} \tag{4.3}$$

4.2 Experimental Result

4.2.1 Result Case

In the proton experiment, we considered four situations. These four situations were hang, error, error-correct, and TMR-correct. Hang and error occurred in both the Streaming architecture and SCEs architecture designs, as well as in the two TMR designs. However, error correction only occurred in the SCEs architecture design and the corresponding TMR design. In the TMR-correct situation, we checked the result; the voter in the PS part received the wrong result from the PL part and recorded the PE number that produced this wrong output. Because in normal time the result for each PE is the same, all the three results from PL should also be the same. This means that the TMR design corrects the result; this was observed only in the temporal TMR design and spatial TMR design. A detailed description of each situation is presented in Table 4.1.

Table 4.1 Description of each situation

Hang	FPGA stops communicating with desktop, data stop send to desktop, the whole system shut down, which means FPGA frozen.
Error	In this case, the FPGA continues to work, but due to the occurrence of SEU. FPGA cannot perform correct calculations, erroneous data is generated. Wrong data send back to the desktop, it cannot be corrected in the sequent calculation.
Error-correct	When an error occurs, the FPGA cannot perform the operation correctly due to the occurrence of SEU, but then it corrects itself and reproduces the correct output in the next calculation process.
TMR-correct	ERROR occurred during the calculation, but revised and corrected by TMR (In testing the TMR version, the voter was found to work in the file storing the experimental results).

The above four situations were combined into two situations. The reason for this final classification was that, as the irradiation test was carried out, the SEU was developed, which made the FPGA unable to perform correct operations, for error-correct, errors were generated and then self-corrected, but as the test goes on incorrect data was generated or the FPGA crashed. Similar results were obtained for TMR-correct. As the test progressed, the TMR design was unable to correct errors or vote for elections. Finally, it also produced errors or made the FPGA crash.

Table 4.2 Description of finally hang and error situation

Hang	means that the FPGA stopped the communication with the laptop and was frozen.
Error	means that the outputs of the CNN were not correct due to SEUs. Include Error-correct and TMR-correct.

4.2.2 Result for Different Architecture

The cross-section for hang and error has different meaning; for hang, it means the reliability threshold for the entire system operation under irradiation conditions: the larger the value of the hang cross-section, the more unstable the system, which means the system is more easily broken down. In contrast, the cross-section for error indicates the reliability of the system under irradiation conditions; if the value of the error cross-section is small, it means that it is more stable, and it can run suitably under irradiation. When hang and error cross-sections are considered together, it implies the maximum threshold for the entire system to work in an irradiated environment and indicates the reliability of its normal work under irradiation. Table 4.3 shows the cross-section results for the two different architectures, and it can be seen that the error cross-section for the Streaming architecture is smaller than the SCEs architecture, which is nearly three times smaller. However, in the hang cross-section results, the two architectures are very similar. The last cross-section, total cross-section, and Streaming architecture are still smaller than the SCEs architecture, but the gap between them is narrow.

Table 4.3 Test result for Streaming architecture and SCEs architecture

	Error cross-section	Hang cross-section	Total cross-section
Streaming architecture	2.78×10^{-11}	2.78×10^{-11}	5.66×10^{-11}
SCEs architecture	7.42×10^{-11}	2.12×10^{-11}	9.62×10^{-11}

4.2.3 Result for Different Harden Approach

The test results for the SCEs architecture and related hardening approach are listed in Table 4.4. For the error cross-section, the values for the two TMR designs are much smaller than those for the unhardened SCEs design. For temporal TMR and spatial TMR, the value of the error cross-section is very close, but for the hang cross-section, the values between temporal TMR and spatial TMR are nearly three times. Finally, for the total cross-section, the gap between the two TMR designs decreased.

Table 4.4 Test results for SCEs architecture and related harden approach

	Error cross-section	Hang cross-section	Total cross-section
SCEs architecture	7.42×10^{-11}	2.12×10^{-11}	9.62×10^{-11}
Temporal TMR	1.56×10^{-11}	4.69×10^{-11}	6.25×10^{-11}
Spatial TMR	1.96×10^{-11}	1.96×10^{-11}	3.92×10^{-11}

5. ANALYSIS

This chapter analyzes the two architectures of Streaming and SCEs based on the test results and combined with the design of the architecture and utilization, and then analyzes the two methods of TMR reinforcement for algorithm design.

5.1 Streaming and SCEs Architecture

Figure 5.1 shows that the cross-section of the Streaming and SCEs architectures is almost the same, which are 2.78×10^{-11} and 2.12×10^{-11}, respectively. Compared with the error cross-section, the SCEs architecture is three times that of the Streaming architecture. 7.42×10^{-11} and 2.12×10^{-11}, therefore leading to the overall cross-section, the value of SCEs is close to twice that of Streaming architecture, which are 9, 62×10^{-11} and 5.66×10^{-11}. The following will use the design architecture and corresponding resources to analyze the above data from the perspective of rate.

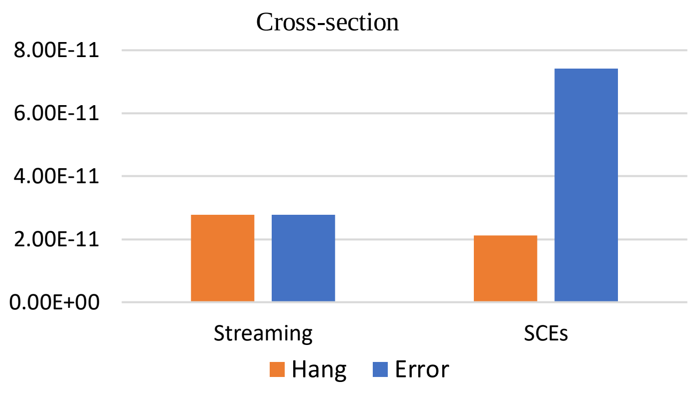


Figure 5.1 Cross-sections of hang and errors for each design

5.1.1 Architecture (PS and PL)

The cross-section of the hang indicates the threshold of reliability for the system, and hang occurs once the FPGA system fails to work. These two architectures can be divided into the PS and PL architectures. On the PL side, Streaming architecture perform all MNIST operations, while the PS side is only responsible for transmitting data. In the SCEs architecture, both the PS and PL sides were responsible for the MNIST operation. However, a large part of the operation it was performed on the PL side.

Therefore, from this perspective, the cross-section of the hang of the Streaming architecture and SCEs architectures should be similar, because the design is a combination of the PL side and PS side arm cortex A53. The cross-section and the occurrence of an error implies that the system has an SEU that causes the operation to yield an incorrect result. For the Streaming architecture, because all operations occur on the PL side, only the occurrence of SEU on the PL side cause an error in the operation.

For the SEC architecture, because both the PS and PL sides undertake the calculation process, the occurrence of SEU on the PS side or the PL side affect the generation of error results. Therefore, the error cross-section of the SCEs architecture can be much larger than the Streaming architecture data.

5.1.2 Utilization

From the perspective of resource usage, the number of LUTs used on the PL side of Streaming architecture is approximately 10 times that of the SCEs architecture, and the number of flip-flops used is approximately three times that of the SCEs architecture. Figure 5.2 shows the total error for the Streaming architecture and SCEs architectures.

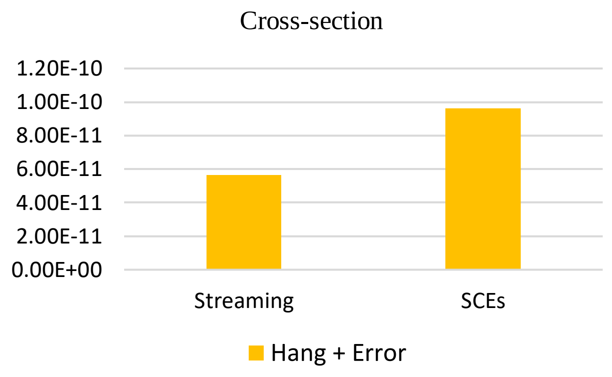


Figure 5.2 Cross-sections of total failures for each design

According to the resource utilization of the PL and combined with the previous SEU test data of the Ultra Scale+ series, we have estimated the cross-section of the two architectures. The estimated value of the cross-section of the Streaming architecture is 1.35×10^{-11} and the experimental result is 2.78×10^{-11}. The results of the two are relatively similar. For the SCEs architecture, the cross-section estimated by pervious test results is 5.4×10^{-12} compared to the actual test result 7.42×10^{-11}, which is much smaller than the actual result.

This is because the estimation result here only uses the resource occupancy of the PL side to estimate and the PS side was not considered for the estimated value (because there are no test data for the PS side SEU at this stage).

This result also verifies that the probability of SCEs generating errors is greater than that of the Streaming architecture. This is because after the PS side joins the calculation, SEU can occur on the PL side and on the PS side.

5.2 Harden Approach

Hang, error and total failures cross-section can be seen from Figures 5.3 and 5.4. The purpose of the two reinforcement methods is to maintain the correct result of the operation when the SEU is issued; therefore, for the error cross-section, both temporal TMR and spatial TMR are substantial improvement compare with the original SCEs architecture, the increase is 78.9% and 73.8%, respectively. And the increase for total failures cross-section is 34.9 and 59.2%. In the cross-section of the hang, temporal TMR increased nearly twice to reach 4.69×10^{-11}, while the spatial TMR just reached 1.96×10^{-11}, similar with the result of original SCEs architecture. Those cross-section results will be analyzed from the perspective of algorithm reuse PE and the perspective of the design refresh mechanism.

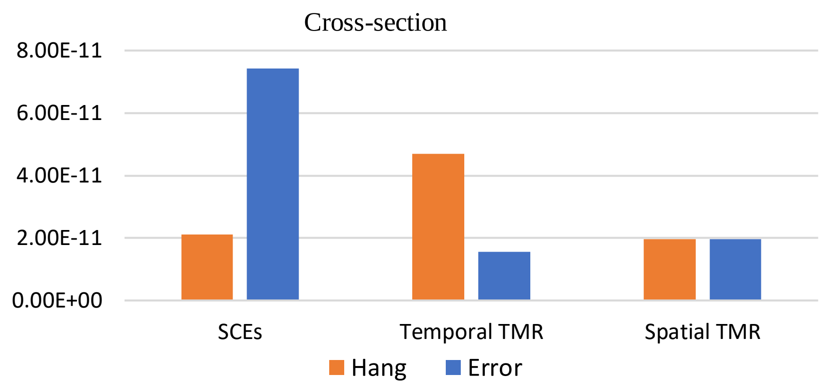


Figure 5.3 Cross-sections of hang and errors for each SECs design

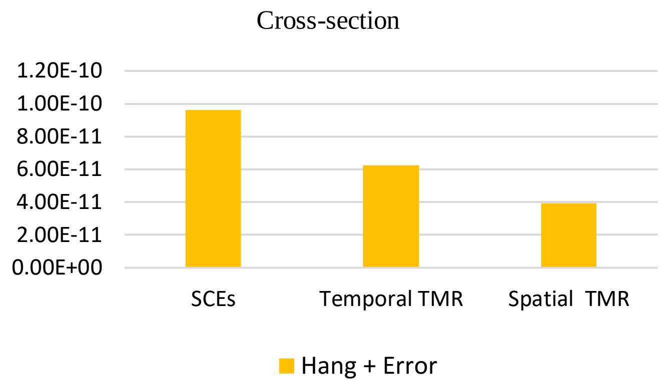


Figure 5.4 Cross-sections of total failures for each SCEs design

5.2.1 Reuse

Both temporal and spatial TMR designs achieve redundancy by reusing the PE operation unit to perform repeated calculations. Therefore, the error cross-section is significantly improved compared to the original system. For temporary TMR, each PE performs three operations on the same set of data. Once an error occurs in one of the operations, the other two operations is guaranteed to be correct, and therefore the probability of error decreases.

For spatial TMR, three different PEs were used to perform operations on the same set of data. Once one of the PEs has an error, the other two PEs can also guarantee the correct result to achieve the effect of redundancy. For the cross-section of the hang calculated from the existing data, the cross-section of the temporal TMR hang is almost twice that of the original. However, the spatial TMR was similar to the original one. It is speculated that because the voter mechanism is added to the PS side in the two TMR designs, the calculations on the PS side increase relatively.

Therefore, the probability of hanging during the test increases correspondingly. The experimental data in this study generally met the speculation results. However, testing and the corresponding error injection system for simulation and related experimental verification are required.

5.2.2 Refresh Rate

It is worth mentioning that the LUTs in the FPGA are refreshed only after reconfiguration, while the data and weight buffer shown in Fig. 3.4 are refreshed every time new data enters. We know that SEU may occur in the LUT or that it may occur in the buffer formed by the register. When it occurs in the LUT, it causes the final calculation data error and cannot be corrected. If this happens in the buffer, it will produce a new correct result because the data passed in each operation is refreshed. Due to this reason, a wrong output is generated, and then it is automatically corrected.

The refresh rate here can be considered as the refresh rate of the buffer, as well as the refresh rate of the calculation, which means the frequency of the calculation (will be tested for future work). For those two reinforcement methods, that is, temporal TMR and spatial TMR. Temporal TMR increases the refresh rate of the buffer, and the same data are sent to the buffer three times, so that the PE can be operated three times to achieve redundancy. In spatial TMR, three different PEs perform the same operation; the refresh for calculation and the refresh rate for the buffer are increased, so the redundancy and reliability can be increased.

6. Conclusions and Future Work

6.1 Summary

In this study, we implemented a CNN with both Streaming architecture and SCEs architectures on a Xilinx Zynq UltraScale+ MPSoC FPGA ZCU-102 board. Then, their error and hang rates were evaluated with proton irradiation. The results showed that the SCEs design has a higher error cross-section compared to that of the streaming architecture, but the SCEs architecture has the same hang cross-section as the Streaming architecture when implemented on a ZCU-102 FPGA. In addition, two resilience techniques were adopted for the SCEs architecture with the same hardware structure by using spatial and temporal TMR redundant techniques. By reusing the process engines in the PL part, the two TMR designs achieve the same effect as traditional TMR designs. The cross-sections of the two hardened CNN designs were reduced by 34.9% and 59.2% with execution time overheads of 14.2% and 21.4%, respectively. The study shows that the SCEs architecture for FPGA acceleration has great potential for applications in a radiation environment with minimal overhead owing to its scalability and flexibility.

6.2 Conclusions

An MNIST CNN was implement on a Xilinx UltraScale+ MPSoC FPGA with both streaming architecture and SCEs architecture. In addition, two hardening designs were developed based on the SCEs architecture by using temporal and spatial TMR approaches. The implemented CNNs were evaluated with accelerated protons at TRIMUF. The overall failure cross-section for streaming architecture is 41.2% lower than that of the SCEs architecture, even though streaming architecture CNN uses much less hardware resources in the FPGA, that's because crush for FPGA is based on the part in the design.

Compared to the unhardened SCEs structure design, the cross-section of the temporal TMR and spatial TMR design reduces 34.9% and 59.2%, respectively. These two implementations introduce no additional hardware resources and only 14.2% and 21.4% additional calculation time by reusing the same PE or different PEs. It shows that SCEs architecture has profound potential in designing complex CNNs for radiation-tolerant applications due to their flexibility and regularity. At the same time from a design perspective, SCEs architecture has higher flexibility. Temporal and spatial TMR, this kind of algorithm TMR for multiplexing PE resources on the PE units could effectively reduce the error rate in a radiation environment.

6.3 Contributions

The main contributions of this study can be divided into two parts. First, the two most frequently used CNN acceleration models are placed on the FPGA, and proton tests are performed to compare the cross-section data. It is concluded that the probability for SCEs architecture and Streaming architecture to crash in an irradiation environment is almost similar; at the same time, the Streaming architecture requires a larger amount of logic resources. The probability of error occurs, and the SCEs architecture is more likely to occur than the Streaming architecture. Therefore, the reliability requirements under irradiation conditions can be determined using the Streaming architecture as it can better ensure the reliability and correctness of the CNN.

Second, the two TMR designs, temporal TMR and spatial TMR, are different from the traditional TRM design; with no additional circuit, it can achieve the same effect as traditional TMR, and better than selective TMR in CNN design. This type of TMR design with no extra circuit will make the SCEs design more flexible, while also making the design of the SCEs architecture more stable than the Streaming architecture, which will expand the application of SCEs in irradiated environments.

6.4 Future Work

When analyzing the data, there is a small difference between the cross-section of the two TMR hangs and the analysis, therefore additional proton test needs to be performed to confirm this difference, and fault injection needs to be added to the corresponding test results.

The system clock in the refresh rate mentioned in the analysis section, it is speculated that when the FPGA uses a faster computing frequency, it increases the refresh rate of the buffer and calculation, which make the CNN more stable under irradiation conditions. This needs to be verified in future studies.

Finally, because the MNIST of the SCEs architecture designed and implemented in this study only used eight PEs, and the actual logical resource occupancy was only 10%, the number of PEs can be increased (or a more complex network can be designed) further, which can be used as a variable to detect the reliability of the CNN under irradiation.