

Design and Reliability Analysis of System-on-Chips for Artificial Intelligence Applications at the Edge

Submitted to
Graduate School of Informatics
Kyoto University

Quan Cheng

Publications

Journal Articles

- [J1] Q. Cheng, M. Huang, C. Man, A. Shen, L. Dai, H. Yu, and M. Hashimoto, "Reliability Exploration of System-on-Chip With Multi-Bit-Width Accelerator for Multi-Precision Deep Neural Networks," in *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 70, no. 10, pp. 3978-3991, Oct. 2023, doi: 10.1109/TCSI.2023.3300899.
- [J2] Q. Cheng, L. Dai, M. Huang, A. Shen, W. Mao, M. Hashimoto, and H. Yu, "A Low-Power Sparse Convolutional Neural Network Accelerator With Pre-Encoding Radix-4 Booth Multiplier," in *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 70, no. 6, pp. 2246-2250, June 2023, doi: 10.1109/TCSII.2022.3231361.

International Conference Papers

- [I1] Q. Cheng, L. Lin, M. Huang, Q. Li, Z. Yang, L. Dai, H. Yu, Y. Chen, Y. Shi, M. Hashimoto, "A 13-34 TOPS/W Edge-AI Processor Featuring Booth-Value-Confined Accelerator, Near-Memory Computing, and Contiguity-Aware Mapping," 2024 IEEE Asian Solid-State Circuits Conference (A-SSCC), Hiroshima, Japan, 2024.
- [I2] Q. Cheng, Q. Li, L. Lin, W. Liao, L. Dai, H. Yu, and M. Hashimoto, "How Accurately Can Soft Error Impact Be Estimated in Black-box/White-box Cases? – A Case Study with an Edge AI SoC –, " 2024 61st ACM/IEEE Design Automation Conference (DAC), San Francisco, CA, USA, 2024.

Summary

In recent years, the field of Artificial Intelligence (AI) has witnessed remarkable advancements, with a growing emphasis on large-volume computing. However, relying solely on cloud-based solutions for processing enormous data in Neural Networks (NNs) presents challenges such as high latency and the need for substantial bandwidth. These challenges underscore the importance of edge AI computing, where data processing occurs locally on the device. However, as NNs often involve tens of millions of weight and activation data, running AI applications on edge platforms often demands substantial computational resources, leading to significant hardware overhead and memory footprint. Besides computational resources, storing excessive weight and activation data often requires large-volume memories. In addition, on-chip memories, with their high data storage density and advanced manufacturing processes, are particularly more vulnerable to soft errors caused by particle radiation, such as neutrons and alpha particles. These soft errors in memories can drastically lead to data corruption (e.g., bit-flip), impacting the system's reliability, especially for those resource-constrained edge platforms without reliability-hardening mechanisms. Therefore, the challenges lie in how to realize energy-efficient edge platforms with high reliability for AI applications, especially for safety-critical applications. Ensuring that AI algorithms can efficiently process data while maintaining reliability is crucial for the effective deployment of safety-critical AI applications at the edge, such as those in autonomous vehicles, medical devices, and industrial control systems. Namely, integrating AI algorithms into safety-critical applications requires comprehensive exploration of the reliability and computational efficiency of AI-enabled systems at the edge. Therefore, the endeavors of this dissertation focus on addressing the challenges posed by safety-critical AI applications at the edge, providing valuable insights for developing robust, fault-tolerant, and efficient AI System-on-Chips (SoCs) in critical domains.

Focusing on computational efficiency in the realm of AI SoC at the edge, both the arithmetic unit and the data flow are key factors targeting performance optimization. At the arithmetic unit level, approximate Multiply-and-Accumulate (MAC) operations offer a valuable solution by leveraging the intrinsic approximations presented in NNs. These operations effectively optimize latency, power consumption, and area utilization without significantly affecting accuracy. Furthermore, adopting mixed-precision computing can also contribute to the deployment of NNs for edge devices. In many comput-

ing scenarios, high-precision data computations, not always necessary, also require high overhead in edge devices. Using low-precision computations while maintaining low accuracy loss yields great performance benefits. Besides, to leverage the approximate and quantization-compatible features of NNs, Neural Architecture Search (NAS) methods offer a promising approach by reducing data precision with limited accuracy loss, minimizing memory footprint, and mitigating hardware resource consumption. Thus, integrating NAS-based methods with Multi-precision (MP) MAC and/or approximate MAC units emerges as a viable solution for resource-constrained edge applications. At the data-flow level, a well-organized data flow plays a pivotal role in achieving computational efficiency. Systolic data flow, as exemplified by the Tensor Processing Unit (TPU), emerges as a powerful strategy, enhancing speed and power efficiency for matrix and convolutional operations while aligning seamlessly with the data-reusable feature of NNs. On the other hand, to address the limitations of traditional Von Neumann architectures, novel approaches like near-memory computing (NMC) technology have been introduced. NMC, featuring distributed memory blocks near Processing Elements (PEs), ensures larger bandwidth and balanced latency in the PE array. Therefore, our work embraces both approximate MAC operations, mixed-precision computing, and advanced data-flow designs, such as systolic data flow and near-memory computing, holistically enhance the computational efficiency of AI applications at the edge.

Ensuring the reliability of AI SoCs, particularly for safety-critical applications at the edge, is important as these applications become more prevalent. Striking a delicate balance between performance, accuracy, and reliability is essential. The approximate feature of NNs suggests that not all soft errors are catastrophic, and their impact on NNs' output may vary due to the differing fault tolerance capabilities of each weight and activation in the NNs. Consequently, some errors may be tolerant with minimal impact on the network's performance, while others could lead to significant deviations in the output. Simultaneously, highly precision-reduced NNs introduce complexities as each bit needs to carry more information, necessitating a thorough analysis of the reliability of these NNs. Besides, in edge devices, SoCs commonly consist of microcontrollers and peripherals, with hardware AI accelerators integrated as peripherals. Conducting reliability analysis, especially for AI applications, can be intricate due to potential black-box testing limitations. To address this, we designed various forms of AI SoCs and conducted fault injection experiments and radiation experiments from a white-box view, offering deeper insights into the reliability of our SoCs. Namely, comprehensive exploration is essential for identifying weak points and developing efficient error detection and mitigation techniques with minimal overhead. Thus, our work aims to explore the reliability and mitigation techniques of AI SoCs at the edge.

First, this dissertation aims to design energy-efficient AI accelerators catering to common operators such as convolution, pooling, and transformer operations. These operators are fundamental components in various NN architectures, playing crucial roles in tasks like feature extraction, down-sampling, and sequence modeling. By optimizing

for these widely used operators, the designed accelerators can effectively enhance the performance and efficiency of a broad range of AI applications. Thus, three distinct AI accelerators are introduced to evaluate performance across different aspects: multiplier design, data flow, and structural sparsity. Firstly, a multi-bit-width (MBW) Booth accelerator is proposed, supporting INT2/4/8 computation. Leveraging low-precision computation in NNs significantly reduces power consumption, and NAS effectively generates mixed-precision NNs with a high proportion of low-precision layers, enhancing power efficiency with minimal accuracy loss. The accelerator, implemented on Microchip MPF300T FPGA, achieves peak performances of 25.6 Giga operations per second (GOPS) for INT8, 51.2 GOPS for INT4, and 102.4 GOPS for INT2 in the convolutional layer, respectively. Secondly, recognizing that common NNs exhibit sparsity exceeding 60%, a low-power sparse accelerator is introduced to maximize MAC-unit utilization and power efficiency. This accelerator incorporates a radix-4 Booth multiplier for pre-encoding weights, reducing partial product count and encoder power consumption. Implemented on a 28nm process, the accelerator achieves 7.0325 Tera operations per second per Watt (TOPS/W) at 50% sparsity and scales up to 14.3720 TOPS/W at 87.5% sparsity. Finally, a Booth-value-confined MP accelerator is presented with a near-memory-computing-friendly data flow. The Booth-value-confined PE minimizes power and area by 82% and 70%, respectively, supporting MP computations. The NMC-friendly data flow, incorporating a partial sum scheduler, facilitates data scheduling in various NN models.

Next, this dissertation presents three SoCs incorporating the developed AI accelerators for AI applications. In flash-based FPGA design, a high-performance SoC with a MBW AI accelerator is proposed. A 2-stage open-source RISC-V core is selected as a centric controller, and a 16×8 PE array is realized for parallel computation. In ASIC design, an edge AI SoC is developed to accelerate MP NNs. Measured on a 22nm testchip with 16 PEs, the peak power efficiency is 25.76/28.61/29.11 TOPS/W when running DarkNet19/ResNet34/VGG16 networks. Furthermore, an optimized work about a multi-core multi-mode SoC is proposed for versatile applications. Accelerator mode is realized for the acceleration of convolutional operation and matrix multiplication with a peak power efficiency of 6.463 TOPS/W. The multi-core mode features a fully programmable structure supporting all general-purposed computations with a peak power efficiency of 5.44 TOPS/W. Furthermore, the dual-core-lock-step (DCLS) mode is adapted for safety-critical applications. Besides, the JTAG debugging port and scan chain are integrated into the chip to facilitate reliability analysis and debugging.

Finally, this dissertation presents the reliability assessment of our SoCs through fault injection (FI) experiments and irradiation experiments (neutron and alpha). The assessment aims to offer valuable insights and serve as an essential reference for future reliability-aware designs. First, the work evaluates the reliability of an FPGA-based AI SoC featuring the MBW accelerator. FI results highlight control and status registers (CSRs) as the most vulnerable component in the accelerator, with layers employing

INT2/4 data being more susceptible than INT8 data due to the increased information carried by each bit in low-precision compared to high-precision. Additionally, the study reveals that MBW networks are more fragile than single-precision networks, and high bits in low-precision are more prone to misclassification errors. Neutron radiation experiments pinpoint the RISC-V core, particularly the Instruction/Data Tightly Coupled Memory (I/DTCM), as the most sensitive component in the SoC. The bit-flips of CSRs in the accelerator contribute to critical Detected Unrecoverable Errors (DUEs), such as dead loops and unresponsiveness. Interestingly, weights and activations exhibit the lowest cross section to system failures in the entire SoC, implying their influence can be disregarded for MBW CNNs in safety-critical AI applications for edge devices. For the ASIC design of a single-core with Booth-value-confined MP accelerator, the dissertation offers a comprehensive evaluation of the reliability of a 22nm AI SoC from both pre-silicon and post-silicon perspectives, with post-silicon alpha experimental results serving as a golden reference. Notably, the I/DTCM emerges as the most vulnerable component, storing static program data, NN model information, and cache data. Flash execute-in-place (FlashXIP) execution mode leads to critical errors in the SoC due to the non-responsiveness of AI tasks caused by upsets in DTCM. The FI experiment on NNs indicates that the MP NN is considerably more reliable than other system components, suggesting minimal impact on the system from bit-flips in accelerator memories related to NN data. White-box and black-box simulation results align, emphasizing their applicability to Commercial off-the-shelf (COTS) platforms without error detection or correction mechanisms, as long as FI can operate on any memory at any given time.

Overall, this dissertation aims to achieve energy-efficient AI SoC designs with necessary and sufficient reliability consideration. The accelerator design leverages sparsity, MP computing, and approximate computing techniques. Additionally, an exploration of high-efficiency data flow enhances data reuse and facilitates efficient NN mapping. In SoC design, a proposed multi-core multi-mode is investigated to support efficient general-purpose computations and fault-tolerant AI applications. Furthermore, fault injection and radiation experiments are conducted to analyze the reliability of the proposed SoCs, providing valuable insights for designing reliable and energy-efficient hardware at the edge.

Acknowledgments

First of all, I would like to express my deepest gratitude to Professor Masanori Hashimoto in Kyoto University for providing me a precious opportunity and an excellent environment to study as a doctoral student in his laboratory. My research is greatly influenced and supported by his invaluable guidance. His advanced perspective and thoughtful advices led me to the achievements.

I would like to appreciate Professor Takashi Sato, Professor Kiichi Niitsu, Associate Professor Hiromitsu Awano, and Assistant Professor Ryo Shirai in Kyoto University for detailed reviews and insightful suggestions.

I would like to appreciate Assistant Professor Longyang Lin in Southern University of Science and Technology for his precious suggestions and enormous help throughout my doctoral research.

I would like to appreciate Associate Professor Yuan Liang in Guangzhou University for his precious suggestions in RF design.

My appreciation also goes to Mr. Mingqiang Huang in Chinese Academy of Sciences for technical discussions and suggestions on AI accelerator design and Dr. Yang Li in Xi'an Jiaotong University for the discussion on the irradiation experiments.

I would like to express my sincere appreciation to Professor Yiyu Shi in University of Notre Dame and Professor Jing-Jia Liou in National Tsing Hua University for precious discussions and support.

I would like to demonstrate my gratitude to my girlfriend Shiyang Yan for encouraging me when I was doing my doctoral research.

I would like to thank other colleagues, both current and former, from the Integrated Systems Engineering Laboratory in Kyoto University for their daily discussions and support: Dr. Ruilin Zhang, Mr. Zhenzhe Chen, Mr. Mingtao Zhang, Mr. Shaoxiong Zhang, Mr. Hiroya Murata, Mr. Kotaro Matsuoka, Mr. Yuibi Gomi and Mr. Jin Nakamura.

I express my heartfelt thanks to all members of Integrated Systems Engineering Laboratory in Kyoto University for having an interesting and comfortable time in the laboratory. I would like to thank Mrs. Shuko Nishiyama for her various support. I would also like to thank all of my friends for the good times we have shared.

I would like to express my heartfelt gratitude to the Kyoto University Graduate Division for awarding me the Kyoto University Graduate Division Fellowship. This presti-

gious scholarship has been instrumental in supporting me throughout my doctoral studies, enabling me to focus on my research and academic pursuits. Without this generous support, completing my Ph.D. would not have been possible. Thank you for your unwavering commitment to supporting me in my journey towards achieving my academic goals.

Finally, I give my thank to my family for supporting my livelihood.

Contents

1	Introduction	1
1.1	Research Status of Neural Networks for Image Processing	1
1.2	Research Status of Edge AI SoC Platforms	3
1.2.1	Multiplier-level Design	4
1.2.2	Accelerator-level Design	7
1.2.3	SoC-level Design	10
1.3	Research Background on Single Event Effects	14
1.3.1	Threat of Single Event Effect	14
1.3.2	Research Status of SEEs in SoCs	17
1.4	Research Objective and Dissertation Structure	19
2	Design of AI Accelerator for Edge Computing	23
2.1	Introduction	23
2.2	Related work	24
2.3	Design of Multi-bit-width Booth Accelerator	27
2.3.1	Implementation of MBW Booth Accelerator	28
2.3.2	Evaluation of MBW Booth Accelerator	33
2.4	Design of Low-power Sparse NN Accelerator	35
2.4.1	Implementation of Low-power Sparse NN Accelerator	35
2.4.2	Evaluation of Low-power Sparse NN Accelerator	42
2.5	Design of Booth-value-confined Multi-precision Accelerator	45
2.5.1	Implementation of BVC MP Accelerator	47
2.5.2	Evaluation of BVC MP Accelerator	54
2.6	Discussion	55
2.7	Conclusion	56
3	Design of System-on-Chip for Edge AI Applications	57
3.1	Introduction	57
3.2	Related Work	58
3.3	FPGA: Single-core SoC with MBW Accelerator	59
3.3.1	SoC Architecture	59
3.3.2	Performance Evaluation	61

3.4	ASIC: Single-core SoC with BVC MP Accelerator	62
3.4.1	SoC Architecture	62
3.4.2	Fabricated Edge AI SoC	63
3.4.3	Performance Evaluation	64
3.5	ASIC: Multi-core Multi-mode SoC	67
3.5.1	SoC Architecture	68
3.5.2	Operating Mode	69
3.5.3	Performance Evaluation	74
3.6	Discussion	75
3.7	Conclusion	78
4	Reliability Analysis of System-on-Chip in Edge AI	79
4.1	Introduction	79
4.2	Related Work	80
4.2.1	Optimization and Reliability of NN with Reduced Precision . .	81
4.2.2	Fault Injection Framework	82
4.2.3	Soft Error Mitigation Techniques	83
4.3	Fault Injection Framework	84
4.3.1	Pre-Silicon Fault Injection Framework	84
4.3.2	Post-Silicon Fault Injection Framework	85
4.4	Single-core SoC with MBW Accelerator Experiment	87
4.4.1	Fault Injection Experiment	87
4.4.2	Fault Tolerant Techniques	90
4.4.3	Neutron Beam Experiment	93
4.5	Single-core SoC with BVC MP Accelerator Experiment	97
4.5.1	Post-silicon Alpha Irradiation Experiment	98
4.5.2	Pre-silicon Fault Injection Simulation	100
4.5.3	Exploration Summary	103
4.6	Discussion	104
4.7	Conclusion	105
5	Conclusion and Future Prospects	107
5.1	Conclusion	107
5.2	Key Contributions	108
5.3	Future Work and Prospects	108

List of Figures

1.1	Multi-precision multiply-and-accumulate (a) 2D D&C ST. (b) SWP ST.	5
1.2	General system-on-chip diagram.	10
1.3	Matestro SoC architecture.	17
1.4	Overall organization of this dissertation.	20
2.1	Roofline model.	25
2.2	Architecture of multi-bit-width accelerator.	28
2.3	Multi-precision Booth multiplier.	29
2.4	Data flow in different systolic architectures (a) and (c) Classical SA. (b) and (d) Proposed vector SA.	30
2.5	NAS flow of multi-bit-width LeNet5.	32
2.6	The architecture of proposed sparse NN accelerator.	36
2.7	Partial products of 8×8 radix-4 Booth multiplier.	38
2.8	Gate level of <i>one/two/neg</i> radix-4 Booth code.	39
2.9	Weight offline encoding in the channel dimension.	39
2.10	Activation selector at different sparsity.	40
2.11	Improved Wallace tree based on customized standard cells.	41
2.12	Post-layout result of the proposed accelerator.	42
2.13	Power efficiency and MAC utilization at different channel-level sparsities.	43
2.14	Booth-value-confined multi-precision accelerator.	47
2.15	Independent and dependent bits in radix-4 and radix-8 Booth.	49
2.16	Booth-value-confined multiplier (N3R8B).	49
2.17	Booth-value-confined mapping from INT8 to BVC8.	50
2.18	Booth-value-confined multi-precision processing element.	51
2.19	Partial sum scheduling for near-memory engines.	52
2.20	Timing diagram in intra-/inter-NME paths for data-reuse.	52
2.21	Memory space contiguity-aware NN mapping with inter-/intra-NME-order (a) Convolutional operation. (b) Matrix multiplication.	53
3.1	Block diagram of SoC.	60
3.2	SoC implementation on MPF300T Eval Kit.	62
3.3	System architecture.	62

3.4	Die micrograph and metrics.	63
3.5	Test platform of the proposed chip.	64
3.6	Area/power breakdown and voltage scaling.	65
3.7	Layer-wise evaluation of different NN models.	65
3.8	Multi-mode system-on-chip architecture.	68
3.9	Accelerator mode architecture.	70
3.10	Multi-core mode architecture.	71
3.11	Vector-level computation using customized instruction.	71
3.12	Fault-tolerant mode architecture.	72
3.13	Error detection and correctness in Fault-tolerant mode.	72
3.14	Fully-access mode architecture.	73
3.15	Die micrograph and metrics.	74
3.16	Neural network acceleration hardware comparison.	77
4.1	A subset of NAS for training MBW NNs.	81
4.2	Simulation-based fault injection framework.	84
4.3	Fault injection environment for MBW accelerator.	85
4.4	Fault injection flow.	86
4.5	Hybrid triple modular redundancy (a) Encoder. (b) Decoder.	91
4.6	Architecture of multi-bit-width fault-tolerant accelerator.	92
4.7	Experimental setup for neutron irradiation at CYRIC, Tohoku University.	93
4.8	Neutron cross section for four reliability configurations.	96
4.9	Experimental setup for alpha irradiation.	98
4.10	Measured error cross-sections in SRAM and FF.	99
4.11	Proportions of error source components.	103
4.12	Cross-section comparison between measured, white-box, and black-box estimates.	103
4.13	Post-silicon cross-section vs. pre-silicon cross-section.	103

List of Tables

1.1	Weight amount and MAC operations for common NNs.	3
1.2	Primary types of single-event effects.	15
1.3	NASA SoC radiation experiment for SEEs.	16
2.1	Precision distribution on three different MBW models.	32
2.2	Resource utilization of MP-Booth MAC on FPGA.	33
2.3	Comparison of the delay in different systolic arrays.	34
2.4	Resource utilization of MBW accelerator on Microsemi MPF300T Eval Kit.	34
2.5	Radix-4 booth algorithm.	37
2.6	Truth table of FAN and HAN.	41
2.7	Performance at 50% random sparsity.	43
2.8	Performance analysis in this work.	43
2.9	Evaluated accelerator.	45
2.10	Non-zero Radix-4 Booth algorithm.	48
2.11	Non-three Radix-8 Booth algorithm.	48
2.12	Comparison of exact and approximate booth multipliers.	54
2.13	Comparison between MP-Booth PE and BVC MP PE.	55
2.14	Comparison of the proposed AI accelerators.	56
3.1	Key CSRs of MBW accelerator.	60
3.2	Resource utilization of our SoC on Microsemi MPF300T Eval Kit.	61
3.3	Evaluation of different NN models.	66
3.4	Performance comparison with state-of-the-art chips.	67
3.5	Performance comparison with state-of-the-art multi-core SoC design.	74
3.6	Comparison of our proposed AI SoCs.	76
4.1	Fault injection at different bit positions in weights.	88
4.2	Critical errors regarding CSRs in MBW accelerator.	90
4.3	Resource overhead of TMR_CSR and HTMR in accelerator.	91
4.4	Four reliability configurations for neutron irradiation.	93
4.5	Event statistics of three models in four reliability configurations.	93

4.6	Related works in soft error assessment of NN applications.	97
4.7	Event statistics in alpha irradiation experiment.	99
4.8	Accessible ranges in white- and black-box cases.	101
4.9	Event statistics in fault injection simulation.	102
4.10	Soft error assessment of our SoCs.	105

Chapter 1

Introduction

This chapter lays the foundation by presenting the background and objectives of the dissertation. The primary goal is to develop System-on-Chip (SoC) designs tailored for artificial intelligence (AI) applications at the edge and to delve into the reliability aspects of the envisioned SoC platforms. This chapter provides a brief overview of the current research landscape concerning neural networks (NNs) for image processing and then outlines the progress of hardware platforms in edge AI. Subsequently, this chapter delves into the research status of single-event effects (SEEs). In conclusion, the chapter outlines the objectives and provides an overview of the overall organization of the dissertation.

1.1 Research Status of Neural Networks for Image Processing

NN is the most widely used technique for image processing in machine learning. A basic NN model is composed of a few to hundreds of layers, each of which includes a number of neurons. The neurons are connected layer by layer through synaptic weights. The synaptic weights are optimized to provide sufficiently high accuracy through comprehensive training. NN Hardware system is mainly categorized into two types. The first one is the inference engine that processes a network with given pre-trained weights, and the latter is the training engine that has the additional capability of weight optimization in the training phase. Regardless of the inference or training engine, multiply-and-accumulate (MAC) arithmetic computation is the primary operation. The rapidly increasing trend of NN size to deal with more intricate and sophisticated problems explodes the amount of arithmetic computation, resulting in a strong demand for energy-efficient computation in edge AI devices.

In 2006, Hinton et al. [1] proposed a new deep learning network based on a multi-layer architecture, namely convolutional neural networks (CNNs). Using a layer-wise

training approach combined with backpropagation, CNNs utilize convolutional kernels to significantly reduce computational overhead, presenting an effective solution for optimizing the deep structure of networks. Shortly thereafter, Hinton et al. [2] introduced the Denoising Autoencoder network, widely applied today for data dimension reduction and reconstruction.

In 2009, Lai et al. [3] proposed a method using Sparse Autoencoders to represent images and study methods for image recognition and classification. By reducing the feature dimensions of images through Sparse Autoencoders, the main features of the images are extracted. When combined with multi-level deep learning networks, these systems exhibit outstanding recognition performance, thus finding widespread application in the field of machine learning.

In 2012, the Google Brain project, led by Dean and Andrew, constructed a massive deep clustering network with tens of thousands of CPUs, achieving ultra-high recognition resolution for animal images. In the same year, during the ImageNet competition, the AlexNet network demonstrated powerful performance in image recognition and feature extraction [4], surpassing the second place by a significant 10%. The AlexNet network primarily consists of five convolutional layers, three pooling layers, and three fully connected layers, enabling the recognition and classification of one thousand different types of images.

In 2013, the ZFnet network, with a structure similar to AlexNet, secured victory in the ILSVRC Image Classification competition and found applications in pathology image recognition [5]. In subsequent developments, the Visual Geometry Group at the University of Oxford introduced the VGGNet model with a deeper hierarchical structure, featuring 13 convolutional layers, 5 pooling layers, and 3 fully connected layers. This model has been widely applied in electroencephalography (EEG) signal recognition [6].

By this point, the development of multi-level deep learning models has been constrained by hardware platforms, making it challenging to achieve breakthroughs in depth. However, in 2014, Google made a groundbreaking advancement with the introduction of the GoogleNet deep learning model, which includes 59 convolutional layers, achieving a significant leap in network depth during the ImageNet competition that year. As network depth continues to increase, research on deep learning theory advances, and practical achievements from major companies rapidly emerge.

In 2016, Microsoft introduced the ResNet model, a massive CNN based on residual blocks [7]. It surpasses AlexNet in scale by 20 times and VGG16 by 8 times, with a hierarchical depth exceeding 150 layers, making it the deepest deep learning model at the time. Similarly, in 2017, DenseNet [8] was proposed with 117 convolutional layers. All layers are connected in a feed-forward way, which mitigates the problem of vanishing gradients and requires few parameters.

Furthermore, in 2020, the Google Brain Team proposed a Vision Transformer (ViT) network, attaining excellent results compared to state-of-the-art CNNs [9]. The pro-

Table 1.1: Weight amount and MAC operations for common NNs.

Performance Metrics	AlexNet	VGG16	GoogleNetV1	ResNet50	ViT-B/16
TOP-5 Error Rate [%]	16.4	7.4	6.7	5.3	2.7
# of Layers	8	16	58	54	12
# of Weight Parameters	61M	138M	7M	25.5M	86M
# of MAC Computations	724M	15.5G	1.43G	3.9G	23.44G

posed ViT-B/16 model achieves 79.9% accuracy on ImageNet, a significant improvement of 2% to training from scratch. These results show that this reliance on CNNs is not necessary and a pure transformer applied directly to sequences of image patches can perform very well on image classification tasks.

Therefore, while addressing problems that traditional algorithms find challenging, deep learning technology also challenges the computational capabilities of devices. Table 1.1 illustrates the weight data and the required number of MAC operations for five common NNs. Notably, the ViT-B/16 network requires a substantial 23.44 billion MAC operations for a single inference.

Even for mainstream desktop-level GPUs like the NVIDIA GeForce RTX 3090, the computational demands of ViT-B/16 are formidable, making it difficult to achieve power-efficient processing at even 1 Joule per image. This underscores the critical need for energy-efficient computing solutions in AI-related platforms. Particularly for mobile terminals, the challenge of local computational resources persists when running large NN models. Moreover, executing deep learning applications on mobile devices faces obstacles in achieving real-time performance for time-sensitive tasks. Historically, the demands for computational resources and computational efficiency have posed limitations on the advancement of AI at the edge.

As advancements in image processing continue to drive the capabilities of NNs, enabling more sophisticated analysis and recognition tasks, the demand for efficient and powerful computation grows exponentially. Focusing on high-efficiency hardware platforms for AI applications, the following section delves into the current research status of edge AI SoC platforms, exploring how these platforms reshape the landscape of AI deployment across various applications.

1.2 Research Status of Edge AI SoC Platforms

In recent years, AI has seen remarkable advancements with a growing emphasis on large-scale computing. However, relying solely on cloud-based solutions for processing vast amounts of data in NNs presents challenges such as high latency and substantial bandwidth requirements. These issues highlight the importance of edge computing, where data processing occurs locally on the device. Edge AI SoC platforms address these challenges by offering powerful, efficient, and real-time processing capabilities

directly at the edge. This approach not only reduces latency but also enhances the overall efficiency of AI applications in real-world scenarios. Besides, the edge AI SoC platforms are intricate architectures that integrate computation, communication, and control functions. This research endeavors to delve into the current status of edge AI SoC platforms, exploring key aspects at the multiplier, accelerator, and SoC levels.

1.2.1 Multiplier-level Design

In the realm of edge AI SoC platforms, the multiplier level represents a critical frontier, dictating the computational efficiency of these intricate architectures. This comprehensive exploration of the multiplier level in edge AI SoC platforms traverses the terrain of mixed-precision computing and approximate computing, unveiling the useful strategies employed to enhance energy efficiency while maintaining accuracy.

Mixed-precision Computing: Precision Tailoring with Quantization

To optimize AI computations for edge devices, a strategy known as precision tailoring has emerged [10]. Precision tailoring involves adjusting the numerical precision of computations to match the specific requirements of tasks, thereby maximizing efficiency without compromising accuracy. One aspect of precision tailoring involves quantization, which refers to reducing the number of bits used to represent numbers. Another technique within precision tailoring is mixed-precision computing [11], which strategically utilizes different levels of precision within NN computations. This approach can seek to achieve the best balance between model accuracy and computing efficiency while efficiently utilizing computational resources, helping to improve performance and power efficiency.

To balance computing efficiency and model performance, some opt for low-bit quantization (e.g., INT8/4/2) for edge computing devices to reduce power consumption while maintaining a low accuracy loss of NNs. Low-bit quantization techniques like binary NNs trade accuracy for power efficiency [12]. Furthermore, multi-precision (MP) NNs offer a compromise, providing an energy-efficient yet practical approach that can maintain model accuracy, suitable for edge devices [11]. This is achieved by further optimizing these networks through fine-tuning the bit-width of individual layers for specific tasks. Additionally, MP multipliers are precision-scalable hardware units that can switch between different levels of precision to accommodate the computational needs of MP NNs. For instance, Huang et al. [13] proposed a Booth multiplier capable of handling 2/4/8-bit precision. With the same hardware cost, the throughput of 2-bit computation is twice that of 4-bit, and the throughput of 4-bit is twice that of 8-bit. Utilizing 2/4-bit computation as much as possible can effectively enhance system computational efficiency while maintaining the accuracy of NNs. Therefore, supporting MP computation in edge AI devices is a promising strategy for high energy-efficiency computation and NN model deployment.

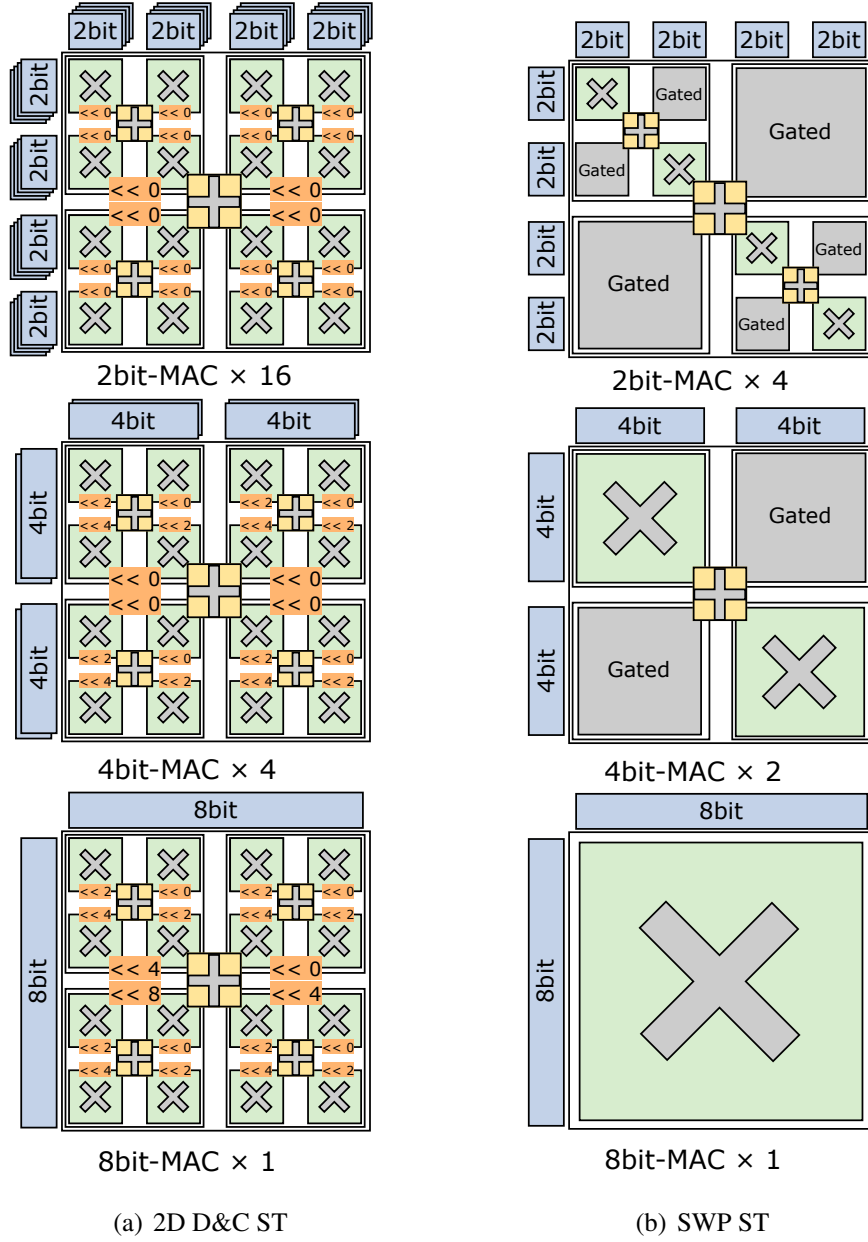


Figure 1.1: Multi-precision multiply-and-accumulate (a) 2D D&C ST. (b) SWP ST.

To support MP computations, a basic MP MAC is necessary. A review of MP MAC is presented in [10]. Basically, two typical multi-bit-width (MBW) MACs are 2D Divide-and-Conquer Sum Together (2D D&C ST) and Subword-Parallel Sum Together (SWP ST), as shown in Fig. 1.1. 2D D&C ST is based on bottom-up strategy configured for one 8b×8b, four 4b×4b, and sixteen 2b×2b. Conversely, SWP ST is based on a top-down strategy configured for one 8b×8b, two 4b×4b, and four 2b×2b.

Almost all MP MACs are implemented based on these two strategies.

2D D&C ST unit achieves MP MAC operations by adding low-precision multiplication units with configurable shifters. Fig. 1.1(a) shows 2D D&C ST design for 8-bit, 4-bit, and 2-bit modes, respectively. Each 2D D&C ST unit includes 16 basic 2-bit multiplications. According to the configuration of shifters in different precision modes (i.e., 2-bit, 4-bit, and 8-bit), these 16 shifted results are divided into 4 groups, and the final result is obtained by summing them up. SWP ST unit achieves MP MAC operations by splitting one high-precision multiplier into multiple lower-precision multipliers. Fig. 1.1(b) shows the SWP ST design. Compared with 2D D&C ST method, the throughput of low-precision cases is lower. SWP ST method sacrifices the utilization of adders in low-precision computation, resulting in the wastage of hardware resources. In general, in comparison to SWP ST MAC, the control logic, including shifters, selectors, and compensating logic, of 2D D&C ST MAC is more complex, resulting in higher resource consumption.

Besides, researchers are actively unraveling the impact of mixed-precision computing on multipliers within edge AI SoC platforms [13, 14, 15]. By judiciously allocating precision levels, these research works aim to elevate computational efficiency, especially in scenarios where the computational cost associated with higher precision can be mitigated without compromising the integrity of AI applications.

Approximate Computing: Speeding Up at the Edge

In the pursuit of computational acceleration, approximate computing emerges as a compelling paradigm at the multiplier level. This strategy consciously introduces controlled imprecision, prioritizing computational speed over absolute precision. Within edge AI SoC platforms supporting real-time processing, the trade-off between precision, NN accuracy, and latency becomes a strategic consideration.

Approximate computing focuses on reducing power consumption and design complexity in error-tolerant applications where computational accuracy is not critical. Most implementations of approximate computing are based on the simplification of partial product (PP) generation [16] and/or accumulation [17], or the dynamic adjustment of bit-width in multipliers [18]. AxBM multiplier [16] is proposed for the simplification of the PP generator. Two approximate radix-8 Booth (R8B) multipliers are implemented to reduce the consumption of LUT resources on an FPGA-based platform. The proposed PP generator can reduce half of the LUT6 consumption compared with conventional methods. Moreover, Sayadi et al. [17] introduced novel approximate multipliers and 4:2 compressors designed to decrease area and power consumption while maintaining accuracy, showing significant improvements in latency, area, and power metrics over exact multipliers, and demonstrating their effectiveness in image processing and NN hardware implementations. In addition, Hammad et al. [18] proposed a tiny two-class precision controller to utilize low-precision and high-precision approximate multipliers together. Using the static segment method (SSM) and the dynamic segment method

(DSM), only the significant segments are extracted from the magnitude bits and computed, while other segments are ignored. Nevertheless, the SSM mapped into the AI application can decrease the performance of the AI model, and the DSM model contains a complex logic to detect the leading one in a bit sequence.

However, since most edge devices solely perform model inference, the training of models is carried out on high-performance computing platforms such as GPUs. Additionally, the computations on GPU platforms are based on exact multipliers, and GPUs are unable to simulate the behavior of the aforementioned approximate multipliers. Therefore, using multipliers based on approximate computing in edge devices can lead to inconsistencies with the results trained on GPU platforms, thus affecting the model's accuracy. The research should scrutinize the integration of approximate computing techniques within multipliers, revealing the scenario where sacrificing precision is acceptable to targeting AI applications.

Challenges

The adoption of mixed-precision and approximate computing at the multiplier level for AI applications is not without challenges. Researchers are diligently addressing issues related to the robustness of AI models under reduced precision, potential error accumulations, and dynamic adaptation of precision levels in real-time applications. Current research works show that MP computation and approximate computation provide a promising avenue to improve the efficiency of resource-constrained edge devices. However, for MP computing, how to effectively transfer traditional single-precision NN models into mixed-precision models to achieve high energy efficiency while maintaining high model accuracy remains to be researched. At the same time, for approximate computing, since the training of large models is mostly conducted on platforms that support exact computing, running the trained models on edge devices based on approximate computing can cause significant precision loss. This challenge also urgently needs to be addressed. Therefore, it is crucial to implement integrated designs for both MP computing and approximate computing in hardware, and develop compatible training methods to achieve low accuracy loss in NNs.

1.2.2 Accelerator-level Design

In edge AI SoC platforms, accelerators serve as the key arithmetic components for efficient and high-performance computations. The in-depth exploration of the accelerator in edge AI SoC platforms traverses the realms of data flow processing and NN model mapping, unveiling the strategies employed to optimize computational efficiency and facilitate the seamless execution of complex AI workloads.

Data Flow Processing: Systolic Array

The core design of the accelerator is data flow processing. This is the basic strategy for scheduling data through the system. In edge AI SoC platforms, data processing with low latency is critical. The efficiency of data flow can directly determine the execution efficiency of AI tasks, so the design of data flow becomes crucial.

Researchers are actively delving into novel architectures and methodologies that streamline data flow, ensuring that data moves through the system with precision, aligning seamlessly with the computational demands of AI workloads. Notably, systolic-based data flow stands out as the most common structure in contemporary edge AI devices [19].

Systolic arrays have been proposed in matrix multiplication since 1976 [20]. Recently, the advantages of AI and the need for massive parallel matrix multiplication have motivated academia and industry to rethink the systolic arrays for accelerating NNs. Unfortunately, the NN function cannot be directly executed by the systolic array due to the unmatched matrix shape of activations and weights. As a result, the `Img2Col` method [21] is usually used to translate a convolutional operation into matrix multiplication that the systolic array can process, though it involves complex matrix reshape operations and incurs additional delay as well as energy consumption costs.

Besides, the systolic-based matrix multiplier can be categorized as non-stationary and stationary based on the way that the operands of the matrix multiplication. In the classical non-stationary systolic array [22, 23], none of the inputs stay in the Processing Elements (PEs) during the execution. Therefore, to guarantee the correctness of computations, the inputs must arrive at each PE at the right time, which requires a complex data flow control module. In the stationary systolic array [19, 24, 25], the input (weight matrix or feature matrix) can stay in the PE for several clocks. Thus, the order of execution loops in the matrix multiplication must be well defined. According to the above discussion, it should be aware that an applicable systolic CNN accelerator requires not only high input/output throughput in the systolic array but also an effective data flow mapping strategy for the NNs.

From pipeline architectures to systolic techniques, the landscape of data flow processing emerges as an important topic at the edge. This dissertation aims to construct effective data flow to enhance efficiency, reduce latency, and optimize the execution of edge-based AI applications.

Neural Network Model Mapping: Allocating Resources Strategically

The hardware utilization of accelerators depends on the NN model mapping strategy. NN mapping involves the allocation of hardware resources to different components in NNs. This strategic mapping ensures optimal efficiency in computations and resource utilization. Researchers are actively exploring mapping algorithms tailored to the unique characteristics of edge-based AI applications, aiming to strike a balance be-

tween computational efficiency and the balanced allocation of hardware resources.

M. Huang et al. proposed a vector accelerator that achieves seamless and efficient NN mapping by rearranging activations and weights based on the number of PEs and the size of on-chip memory [13]. This design allows for maximum data reuse same as the number of PEs, achieving a computational throughput of 2863.29 GOPS under INT2 on the ZCU102 FPGA development board. J Yue et al. presented an integrated compute-in-memory (CIM) accelerator design that achieves NN mapping in inter-macro and intra-macro directions [26], realizing the highest efficiency of 158.7 TOPS/W on TSMC 65nm process node. Nevertheless, the efficient memory bandwidth utilization due to unbalanced read/write patterns in this design leads to high memory access latency. Additionally, C. -Y. Du et al. introduced a hardware-utilization-aware NN accelerator that leverages MAC utilization, dynamically adjusting parallel levels across multiple dimensions to maximize MAC utilization [27]. Their design achieves a peak power efficiency of 11.2 TOPS/W on TSMC 28nm process node. However, this design mainly focuses on MAC-level utilization, and the inefficient NN mapping strategy can actually lead to substantial overhead of power consumption and latency. Besides, it is worth noting that F. Conti et al. proposed a multi-core design with an MP accelerator [28], where the basic structure of the accelerator supports only 3×3 and 1×1 convolutional operations. Therefore, running the actual NN model requires software co-compilation to facilitate the calculation of different models. On the other hand, its advantage lies in the programmable multi-core architecture, which can be used for general computing.

These studies explore the NN model mapping methods in AI accelerators. From array-based optimization to dynamic allocation strategies, researchers are working to develop mapping algorithms that adapt to different workloads at the edge. However, regarding the aforementioned problems, how to build an efficient NN mapping method based on the design of hardware that supports energy-efficient computing still needs to be overcome. Therefore, the content of this research also includes how to implement an accelerator architecture and its NN model mapping method that can seamlessly handle different NN architectures.

Challenges

While data flow processing and NN model mapping can realize several significant advancements and benefits (e.g., latency reduction, low bandwidth usage, power efficiency improvement) when implemented on edge hardware platforms, challenges persist. Addressing issues related to generality and dynamically efficient mapping, and ensuring scalability across different AI applications is still quite important. Dynamic mapping strategies and hardware utilization awareness are shaping the landscape of AI accelerators.

In conclusion, the accelerator for edge AI applications is a focal point of research, where the coordination of the data flow and the NN mapping strategy decides the computational efficiency at the edge. By exploring the intricacies of these approaches, this

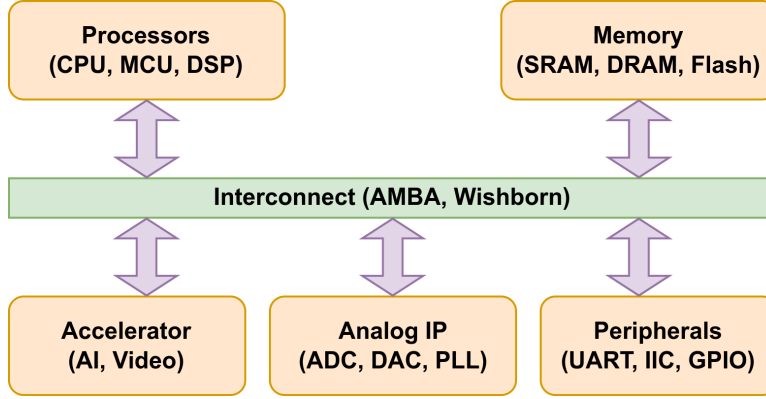


Figure 1.2: General system-on-chip diagram.

dissertation aims to optimize current architectures and propel the development of accelerators, enabling compatibility with diverse applications.

1.2.3 SoC-level Design

In the intricate ecosystem of edge AI SoC platforms, the SoC design features a high level of integration, including computational elements, memory subsystems, and communication interfaces. Currently, the research works of the SoC targeting edge AI platforms delve deeply into the strategic choices between single-core and multi-core architectures, unraveling their impact on efficiency, adaptability, and computational ability.

The International Technology Roadmap for Semiconductors (ITRS), released in 2009, definitively asserts that the characteristic dimensions of semiconductor devices have entered the nanoscale. SoC is identified as the primary trajectory for the forthcoming evolution of Very Large Scale Integration (VLSI). SoC plays a pivotal role in propelling the miniaturization of integrated circuits. Moreover, the integration of AI in edge devices further accentuates the significance of SoC in shaping the landscape of modern semiconductor technologies.

"SoC" refers to integrating computer or control systems on a single chip. Unlike traditional systems implemented on a single Printed Circuit Board (PCB) using discrete components, SoC employs embedded system design techniques. It integrates functionalities such as CPU, DSP, memory, bus, clock management, and peripheral circuits onto a single chip to realize a complete system, as shown in Fig. 1.2. Compared to PCB-based systems, SoC offers advantages such as miniaturization, low power consumption, high speed, high integration, and reliability. SoC utilizes Intellectual Property (IP) core and hardware-software co-design techniques, enhancing performance, portability, and reconfigurability, while reducing development cycles and costs. With the rapid development of AI, the integration of AI capabilities into SoC has become increasingly significant. SoC platform not only leverages the benefits of miniaturization but also

unleashes new potentials for intelligent processing at the edge. Numerous satellite systems, including satellite control, telemetry and control, numerical control, as well as communication, navigation, remote sensing, and data transmission, explicitly demand miniaturization. This trend aligns with the growing importance of AI applications in shaping the future of semiconductor technologies.

FPGA and ASIC Platforms for AI SoC Implementation

In the rapid evolution of AI, the need for efficient and powerful computation has driven the development of specialized hardware solutions. Field-Programmable Gate Arrays (FPGAs) and Application-Specific Integrated Circuits (ASICs) have emerged as key platforms for implementing AI SoC designs, each offering unique advantages tailored to specific application requirements.

FPGAs are highly adaptable hardware devices that can be reprogrammed to perform a variety of computational tasks. This reconfigurability makes FPGAs an ideal choice for AI SoC implementation, particularly during the development and prototyping phases. Designers can quickly iterate on design changes, optimizing NN architectures and data flow processing without the need for new hardware. This flexibility allows for the testing and fine-tuning of AI models in a hardware environment, significantly accelerating the development cycle. Moreover, FPGAs can efficiently handle parallel processing, making them suitable for the high-throughput demands of AI applications. They can be tailored to specific AI workloads, optimizing performance and energy efficiency. However, FPGAs typically consume more power and offer lower performance compared to ASICs when scaled to production. For example, an SoC design detailed in [13] implements a 64×32 PE array, consuming about 80% of the logic resources on the ZCU102 FPGA board, but operates at a low working frequency of 214MHz. This results in a low throughput of 784 GOPS and a high power consumption of 29.1 W. Additionally, due to the architecture of SRAM-based FPGAs, their configuration memory is inherently vulnerable to radiation, making them unsuitable for safety-critical applications. Fortunately, flash-based FPGAs provide a robust alternative, as flash devices are resilient to irradiation [29]. Therefore, despite the high resource consumption, further exploration of AI SoC designs using flash-based FPGAs is necessary from a reliability perspective.

In contrast, ASICs are custom-designed chips tailored to perform specific tasks with maximum efficiency and performance. However, once designed and fabricated, ASICs cannot be reprogrammed, but this specialization allows them to execute AI algorithms with high speed and power efficiency. For large-scale deployment of AI solutions, such as in data centers or edge devices, ASICs offer a significant advantage in terms of power efficiency and cost-effectiveness over time. From a reliability standpoint, ASICs are inherently more robust than FPGAs in several ways. Because ASICs are designed for specific tasks, their architecture can be optimized to minimize susceptibility to Single Event Upsets (SEUs) and other radiation-induced errors. This is particularly important

in safety-critical applications such as aerospace and autonomous driving. Additionally, ASICs can be fabricated using radiation-hardened designs [30] or processes [31], further enhancing their reliability in harsh environments. The high level of reliability, combined with their performance advantages, makes ASICs the preferred choice for mission-critical AI applications that require consistent, long-term operation.

Overall, while both FPGAs and ASICs have their distinct advantages, the choice between them depends on the specific requirements of the AI application, including flexibility, performance, power consumption, and reliability. By leveraging the strengths of each platform, this dissertation focuses on the design of optimized AI SoC solutions that meet the demands of modern AI workloads.

Single-Core and Multi-core Architecture

The adoption of single-core architectures within edge AI SoC platforms is driven by their simplicity and suitability for tasks with lower parallelism. Single-core designs, while lacking the parallel processing capabilities of multi-core systems, excel in scenarios where specific tasks dominate the computational landscape, often utilizing peripheral dedicated accelerators. This section discusses the strengths and limitations of single-core architectures, focusing on how their streamlined design impacts computational efficiency and adaptability in edge-based AI applications.

Typically, in AI SoC, efficient software and hardware collaboration is the key to deploying AI applications. Z. He et al. [32] designed an SoC with an open-source RISC-V core and an MP AI accelerator supporting INT2/4/8 computation. An agile hardware and software co-design for RISC-V-based MP deep learning microprocessor is developed to simplify and expedite the deployment of AI applications at the edge. In addition, it is crucial to efficiently obtain various data from the outside (such as sensors) and perform efficient calculations. P. D. Schiavone et al. [33] proposed a 22nm SoC featuring an autonomous I/O subsystem optimized to deal with the wide variety of sensors available in Internet of Thing (IoT) end-nodes. This SoC contains a 32-bit open-source RISC-V core. The system runs up to 2400 million equivalent RV32IMC instructions per second (MOPS) and achieves a peak power density of 6 uW/MHz, resulting in a power efficiency of 433 MOPS/mW. Besides, an efficient accelerator is the key to achieving efficient AI computing in a single-core system. Huang et al. [13] designed an AI accelerator with MP computation that achieves a high bandwidth usage and hardware utilization with a vector-systolic-based data flow and hardware-aware NN mapping method. Meanwhile, the low-complexity design of the single-core architecture makes reliability analysis easier.

Conversely, multi-core architectures emerge as powerhouses of parallel processing capabilities, crucial for efficiently handling the complex operations in AI workloads. The research landscape is characterized by an ongoing endeavor to optimize the configurations of multi-core SoCs to meet the diverse demands of AI applications. The advantages of parallelism offer prospects of accelerated processing, enhanced real-time

capabilities, and adaptive handling of intricate NN models.

Unlike single-core framework, the data acquisition from external components is often involved with complex scheduling. A. Pullini et al. [34] designed a 40nm SoC featuring a hierarchical architecture with a small (12K gates) RISC-V core. The proposed SoC features an efficient IO subsystem capable of transferring up to 1.6 Gb/s in less than 2.5 mW for efficient data exchange, and an 8-core floating-point computing cluster achieving a peak performance of 850 Mega MACs per second (MMACS) and 1 Giga floating-point operations per second (GFLOPS) and the power efficiency up to 15 MMACS/mW in fixed-point and 9 MMACS/mW in floating-point. Moreover, a hybrid architecture in which multi-cores and accelerators coexist can achieve better edge computing. Multi-cores are used to optimize complex calculations, and accelerators are used to accelerate specialized calculations (such as convolutional operations). E. Flamand et al. proposed a fully programmable RISC-V IoT computing engine [35], featuring an 8-core cluster with CNN accelerator, coupled with an ultra-low power MCU with 30 μ W state-retentive sleep power. This design delivers up to 10 Giga MACs per second (GMACS) for CNN inference at the power efficiency of 600 GMACS/W within a worst-case power envelope of 75 mW. Furthermore, F. Conti et al. presented Marsellus, an all-digital AI-IoT end-node heterogeneous SoC, including a 16-core RISC-V design and an MP AI accelerator, fabricated in GF 22nm FDX [28]. This design provides 12.4 TOPS/W peak power efficiency and 6.38 TOPS/W best power efficiency at ResNet20. However, the accelerators and multi-cores of these designs are separate components, so the inter-component latency when performing AI tasks will be large. Moreover, hardware resources cannot be utilized efficiently. Besides, due to the intricate architecture of multi-core systems, their reliability issues are likely to be more pronounced and deserve further in-depth analysis.

In summary, both single-core and multi-core architectures are crucial in edge AI SoC designs, each with its strengths tailored to edge-based AI applications. Single-core designs offer simplicity and straightforward reliability analysis, making them appealing for applications prioritizing fault tolerance and predictability. Conversely, multi-core architectures promise enhanced performance and scalability but present challenges in reliability analysis due to their complexity. Further exploration of both architectures is warranted to explore the robustness of edge AI SoC designs.

Challenges and Trade-Offs

Implementing AI SoCs on FPGA and ASIC platforms is pivotal in AI research. FPGAs offer adaptability and quick prototyping, while ASICs provide high performance and efficiency. Single-core architectures are simple and suited for tasks with fewer parallel computations, while multi-core architectures excel in handling complex workloads. However, these systems may face challenges in terms of reliability and efficiency trade-offs. Although multi-core systems may encounter more reliability challenges and higher overhead, their performance advantages make them essential in AI applications. Fur-

ther research is needed to refine computational efficiency and reliability analysis for both architectures and unlock their full potential in real-world applications. Therefore, by leveraging the strengths of FPGA and ASIC platforms, this dissertation aims to develop optimized edge AI SoCs to meet high-performance computing, high hardware utilization, and high-reliability requirements for modern AI applications.

1.3 Research Background on Single Event Effects

1.3.1 Threat of Single Event Effect

The challenges of radiation can induce Single Event Effects (SEEs) in SoCs, significantly affecting their reliability and lifespan. SEE occurs when a single particle traverses the sensitive region of a device, generating electron-hole pairs due to ionization. These pairs are collected by the device's reverse-biased PN junction, forming a pulse current and causing radiation damage to the device [36, 37]. The generation of SEE has various causes, including high-energy heavy ions produced by galactic cosmic rays, solar cosmic rays, Van Allen radiation belts, high-energy protons, alpha particles, and neutrons [38, 39]. Other sources include alpha particles generated by the decay of uranium and thorium impurities in packaging materials or the interaction of alpha particles generated by Boron-10 doping with semiconductor devices [40]. Near-Earth space neutrons or neutrons generated in a nuclear explosion environment can also induce SEE.

In the terrestrial environment, SEEs are mainly induced by alpha particles emitted from package materials and neutrons originating from cosmic rays. The mechanism of SEE varies depending on the type of incident particle. Generally, heavy ions directly cause ionization and induce SEE, while high-energy protons or neutrons require the indirect ionization produced by nuclear reactions to generate other heavy ions that cause SEE, hence the term indirect ionization. When the deposited charge of the incident particle exceeds the critical charge of the device, SEE occurs. The critical charge is related to the device's voltage and node capacitance. With the development of semiconductor devices, the working voltage and node capacitance of devices have decreased, making the critical charge for SEE smaller and smaller. Especially when the feature size of devices enters the nanometer range, the critical charge is less than 1 fC and continues to decrease. Numerous experiments have demonstrated that even low-energy protons can directly cause SEE in nano-scale SRAM [41, 42].

Since the first report of the J-K flip-flop (FF) in satellite systems generating this effect by Binder, Smith, and Holman in 1975, multiple incidents related to SEEs have been reported in spacecraft accidents from various countries [43]. According to NASA's statistical data [44], from 1974 to 1994, over 100 spacecraft experienced failures and anomalies. Through the analysis of 45% of these spacecraft issues, it is found that space radiation caused 86% of anomalies, with SEEs accounting for a significant portion. Several countries, including the United States, the European Space Agency, and

Table 1.2: Primary types of single-event effects.

Type	Description
Single Event Effect (SEE)	Changes in the logical state of storage devices
Single Bit Upset (SBU)	Logic state changes in a single bit of storage
Multiple Bit Upset (MBU)	Logic state changes in multiple bits of storage
Single Event Transient (SET)	Transient voltage pulses in analog or combinational circuits
Single Event Functional Interrupt (SEFI)	Abnormal functionality in control circuits
Single Event Latch-up (SEL)	Device locked in a high-current state
Single Event Snapback (SES)	Feedback current mode in NMOS devices
Single Event Burnout (SEB)	Device triggered high current resulting in burnout
Single Event Gate Rupture (SEGR)	Gate damage in power MOSFETs
Single Hard Error (SHE)	Irreversible errors in individual bits

Japan, have reported incidents of in-orbit satellite failures due to SEEs. For instance, the U.S. "TDRS-1" satellite experienced 4,468 SEEs between April 4, 1983, and March 7, 1993. The European Space Agency's "ERS-1" satellite suffered chip burnout in a precision rangefinder due to SEEs, and Japan's satellite "SUPERBIRD" was lost due to a SEE-induced misoperation. Therefore, SEEs have become the most serious radiation effect threatening the in-orbit safety of spacecraft, causing severe spacecraft failures and significant economic losses. Meanwhile, in terrestrial applications, SEEs also present significant challenges. In high-performance devices such as supercomputers and GPUs, high reliability is crucial due to the vast amount of data processed and the potential impact of data corruption. SEE-induced errors in these systems can lead to system crashes, data corruption, and significant downtime [45, 46]. Error Correction Code (ECC) and redundant systems are often employed to mitigate these risks, but they cannot eliminate the problem. In automotive systems, especially with the increasing adoption of advanced driver-assistance systems (ADAS) and autonomous driving technologies, the reliability of electronic components is critical [47]. SEE can cause transient faults in critical systems, potentially leading to malfunctions in safety systems such as braking, steering, and collision avoidance. This can pose significant safety risks, making it imperative to develop robust SEE mitigation techniques. Thus, automotive systems often use radiation-hardened components or implement system-level error detection and correction mechanisms to enhance reliability.

According to the different responses of devices or circuit units to incident particles, SEEs can be classified into various types [48], such as changes in the logic state of storage or registers, transient voltage pulses in combinational circuits, and damage to the power MOSFET gate, among others. The specific classification is shown in Table 1.2. Based on whether the incident particle causes permanent physical damage to the device or circuit unit, SEEs can be further categorized into soft errors and hard errors. Soft errors refer to types of errors that do not cause permanent damage to the device and are recoverable, such as Single Bit Upset (SBU), Multiple Bit Upset (MBU), Single Event Transient (SET), Single Event Functional Interrupt (SEFI), etc. Besides, SEU includes both SBU and MBU. Hard errors, on the other hand, result in permanent physical damage to the device, and if not addressed promptly, can lead to device burnout

Table 1.3: NASA SoC radiation experiment for SEEs.

SoC	Manufacturer	Processor	Experimental location
1st Generation RAD750	IBM	PowerPC750	TAMU
2nd Generation RAD750	IBM	PowerPC750	TAMU
UT699	Aeroflex Gaisler	LEON3FT	TAMU
P2020	Freescale	e500	TAMU&LBNL
P5020	Freescale	e500	TAMU&LBNL
P2020	Freescale	e500	TAMU
Maestro	Boeing	TILE64	TAMU&NSRL

or circuit failure. Examples of hard errors include Single Event Latch-up (SEL), Single Event Burnout (SEB), Single Event Snapback (SES), Single Event Gate Rupture (SEGR), Single Hard Error (SHE), etc.

Additionally, the rapid advancements in semiconductor devices, leveraging novel lithography techniques, semiconductor materials, innovative device structures, and enhanced process integration, are driving the continuous pursuit of integrated circuits (IC) featuring high speed, density, reliability, low power consumption, and cost-effectiveness. Amidst these developments, the evolution of nano-scale ICs introduces new mechanisms and variations in SEEs. The structural changes, reduced size, and altered materials contribute to a heightened phenomenon of multiple-bit flips in memories, particularly as integration density increases and operating voltages decrease. The elevated operating frequency amplifies SET, emerging as a significant factor impacting timing circuits and combinational circuits. The increasing complexity of functional modules in SoC designs poses challenges in mitigating SEFI, making radiation hardening more intricate. SoCs, encompassing CPUs, SRAM, peripheral circuits, etc., face a growing severity of nano-scale SEE occurrences, marked by diverse failure types.

In the context of AI applications at the edge, where SoC reliability is important, the integration of SoCs in various mission-critical applications, such as space missions, automotive systems, and supercomputers, gains heightened significance. The excellent characteristics of SoCs play a pivotal role in these electronic systems. However, the substantial threat posed by SEEs necessitates careful consideration and the implementation of radiation-hardening measures to enhance the reliability of SoCs in these environments. With commercial SoCs advancing to the 28nm and 16nm stages, urgently addressing the evaluation and analysis of nano-scale SoC SEEs has become a critical concern, garnering significant attention in safety-critical AI research. Recognizing the diverse sensitivities and failure mechanisms across modules, research on SoC SEE mechanisms, the establishment of reliable assessment methods, identification of weak points, and guiding radiation-hardening designs are imperative. This dissertation focuses on the assessment of AI scenarios with high-reliability requirements, particularly in the context of the growing complexity of nano-scale ICs in edge AI applications.

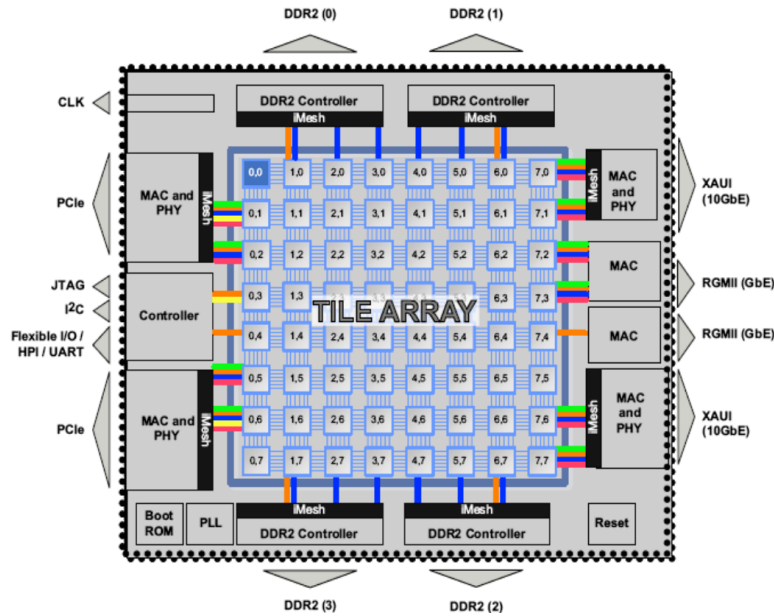


Figure 1.3: Matestro SoC architecture.

1.3.2 Research Status of SEEs in SoCs

Research on SEEs in large-scale ICs has long been underway. As early as the 1980s, experiments testing SEEs on SRAM and CPU were conducted. NASA began conducting experiments on microprocessor SEEs in 1986, testing Intel processors such as Pentium MMX, and Pentium III, as well as MOT processors like PC7455, PC7448, and AMD K7 processors, among others [49]. Moreover, SoC integration into aerospace applications has been a focus since the 1990s, with initiatives like NASA's X-2000 program, Boeing's OPERA project, and the U.S. Naval Postgraduate School's (NPS) CFTP program [50, 51, 52, 53, 54]. To ensure SoCs' longevity and reliability in space, NASA conducted ground-based accelerator experiments to study SEEs in SoCs [55, 56, 57, 58, 59]. These experiments provide valuable data on SoC radiation effects, as shown in Table 1.3, conducted at facilities such as Texas A&M University (TAMU) Cyclotron Accelerator, NASA Space Radiation Laboratory (NSRL) at Brookhaven, and Lawrence Berkeley National Laboratory (LBNL).

To enhance SoCs' reliability, Boeing's OPERA project developed the Maestro SoC as shown in Fig. 1.3, based on the TILE64 processor, integrating 49 radiation-hardened microprocessors. Each has an 8KB L1 data cache, 8KB L1 instruction cache, and 64KB L2 cache. SEU resistance is achieved through parity and ECC. The NPS's Configurable Fault Tolerant Processor (CFTP) program aims to improve radiation-hardening through reprogrammability and reconfigurability, integrating three PowerPC750 processors on a Xilinx radiation-hardened FPGA with Triple Modular Redundancy (TMR) and ECC. In 2011, NASA tested Freescale's P2020 and P5020 processors for SEEs. Results show

proton SEU cross-sections for the L1 cache with high susceptibility compared with the L2 cache and registers.

In Europe, the European Space Agency (ESA) has extensively researched radiation effects on microprocessors and radiation-hardened IP cores. In 2009, ESA began developing the multipurpose microprocessor SoC, NGMP, based on the SPARC V8 multi-core architecture, integrating four LEON4FT cores. It uses Bose-Chaudhuri-Hocquenghem (BCH) codes, parity codes, Reed-Solomon (RS) encoding, and TMR for fault tolerance [60, 61]. In 2010, Aeroflex Gaisler developed the UT699 SoC, based on the 32-bit LEON3FT processor, employing TMR and parity codes for radiation hardening [57], with SEEs tested at TAMU Cyclotron Accelerator [62]. Moreover, Gaisler introduced an SoC based on the LEON3FT-RTAX processor, implemented on an Actel FPGA, using fault-tolerant designs for cache and register files. Ground SEE experiments show a single-event FF Linear Energy Transfer (LET) threshold over 37 MeV/(cm²/mg) [63].

In addition, with the rise of small and cheap satellites, the use of commercial off-the-shelf (COTS) devices in space has become more common. These devices are favored for their higher performance, lower cost, and shorter development cycles [64] compared with the aforementioned reliability-hardening hardware platforms. However, their relatively poor radiation resistance necessitates lightweight hardening techniques to achieve necessary and sufficient reliability consideration.

In terrestrial environments, particularly concerning neutrons and alpha particles, SoCs encounter challenges in automotive driving and other high-performance systems. Neutrons and alpha particles can induce SEUs in these environments, necessitating testing and mitigation efforts for reliability beyond space applications. Furthermore, in the evolution of AI applications, hardware designs primarily aim to effectively meet the requirements of these complex systems using AI algorithms. Especially, regarding single-core areas in edge computing, a study using a single-core low-performance cortex-M4 chip assessed three machine learning models' resilience (ANN, support vector machine, and random forest) to neutron-induced soft errors on tiny machine learning computing systems [65]. Despite occasional interruptions, the random forest algorithm notably avoids misclassification during radiation testing. For multi-core edge design, evaluating error rates, particularly for safety-critical applications, is crucial due to the complex architecture. A study investigated the error rate of the commercial RISC-V ASIC platform, GAP8 [35], under neutron exposure, revealing a 3.2x increase in error rates for computing-intensive applications like CNN, with the majority of errors not leading to misclassifications [66]. Additionally, this work identifies application hangs as the primary cause of interruptions. Meanwhile, the mitigation techniques in these AI-related applications are also important. Traditional error mitigation approaches like replication may not always be the most efficient choice. Duplicating or triplicating NN operations can incur significant overhead. Fortunately, the probabilistic feature, redundancy, and computational flexibility of NNs enable the development of low-cost

mitigation strategies. For example, lightweight, design-specific solutions have been introduced to minimize overhead in GPUs [67] and FPGAs [68]. Similarly, lightweight solutions have been proposed for generic image processing applications as well [69].

However, these designs still introduce high power and area consumption. Additionally, they require intricate design and verification processes, leading to higher development complexity. The physical size and reliability of radiation-hardened components can also be a drawback. Furthermore, while commercial SoCs offer high performance and low cost, their poor radiation resistance limits their use in these applications, creating a trade-off between advanced technology and robustness. On the other hand, with the increasing use of COTS devices in these fields, a black-box/white-box discussion emerges [70]. Black-box methods treat the system as a whole without comprehensive analysis, while white-box methods involve detailed internal component analysis. COTS platforms often function as black boxes, making it challenging to conduct thorough fault injections and reliability analysis due to limited access to internal components. For example, while debuggers can access certain registers and memory segments, they typically cannot utilize scan chains to gain deeper insights into the chip. This raises the key question of accurately estimating soft error impacts on a black-box SoC.

In summary, ensuring high reliability in SoCs involves addressing various challenges, including susceptibility to SEUs, the implementation of lightweight hardening techniques, and the complexity of reliability analysis, particularly in black-box systems. SEUs, induced by radiation particles such as neutrons and alpha particles, pose significant reliability concerns, necessitating testing and mitigation efforts both in space and terrestrial environments. To mitigate SEUs, novel lightweight solutions have been developed, leveraging the probabilistic feature and redundancy of NNs. These solutions aim to minimize overhead in hardware platforms like GPUs and FPGAs. However, achieving high reliability in SoCs still comes with high costs, including increased overhead and complexity in design and verification processes. Additionally, the use of COTS devices, while offering high performance and low cost, presents challenges in reliability analysis due to limited access to internal components. The primary aim of this dissertation is to address reliability concerns through a combination of fault-tolerant mechanisms, lightweight hardening techniques, and thorough analysis, even in black-box systems, to ensure the robustness of SoCs in critical AI applications.

1.4 Research Objective and Dissertation Structure

The main objective and the contribution of this dissertation are described here. Targeting energy-efficient edge AI SoC designs with necessary and sufficient reliability consideration, this dissertation focuses on two aspects: (1) design of edge AI SoCs, and (2) reliability analysis of SoCs running AI applications.

The former aims at energy-efficient SoC designs, especially the parallel computation to improve power efficiency, while the latter intends to explore the reliability of our

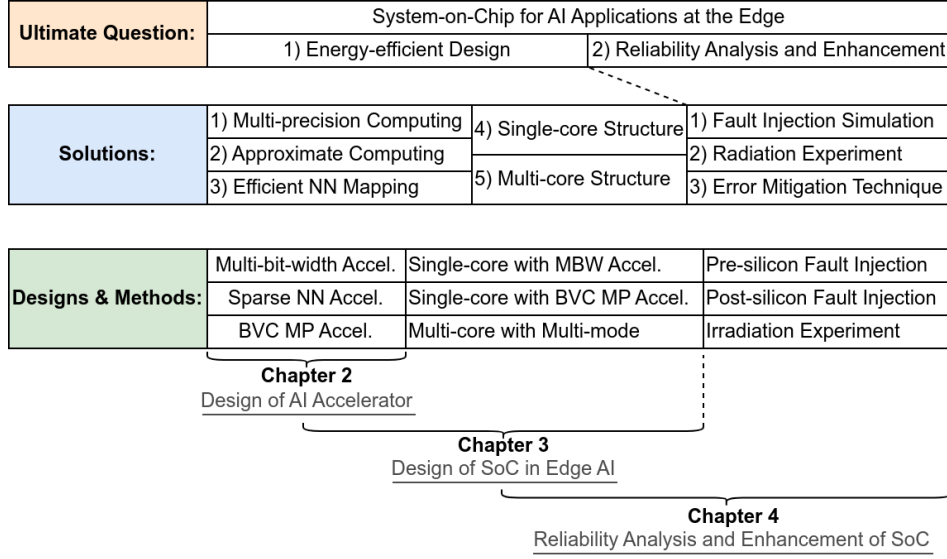


Figure 1.4: Overall organization of this dissertation.

proposed SoC platforms, providing valuable insights and serving as crucial references for future research. This dissertation addresses the unexplored components and the disadvantages of the existing works mentioned in Section 1.2 and Section 1.3 such as efficient NN mapping in AI accelerator, high power-efficiency MP multiplier in PE, and fault-tolerant framework in SoC platforms, and provide effective solutions to resolve or improve these issues. Besides, this dissertation designs three AI SoCs, implemented on flash-based FPGA or ASIC platforms.

Fig. 1.4 illustrates the organization of this dissertation. There are three research parts that are included, and each part represents different targets: (1) design and evaluation of AI accelerator, (2) design and evaluation of edge AI SoC, and (3) reliability analysis of the proposed SoCs. (1) can perform exceptional power efficiency in computing units and efficient data flow for data scheduling in NN mapping, (2) targets SoC designs for effective deployment of AI applications with hardware-software co-design, and (3) aims to explore the reliability of the proposed AI SoCs, enabling thorough analysis and optimization.

As for (1), Chapter 2 introduces three proposed AI accelerators. In the MBW Booth accelerator, an MP multiplier based on the Booth algorithm is proposed to realize INT2/4/8 multiplication for energy-efficient computation. Additionally, a vector-based systolic array is proposed for effectively accelerating the matrix multiplications. Regarding low-power sparse NN accelerator, a pre-encoding radix-4 Booth algorithm, highly compatible with data-reuse and weight offline pre-processing, is proposed to simplify multiplication for area and power consumption reduction. Furthermore, for low-power design, the non-zero weights above 50% sparsity are steered into arithmetic units, such that each computation logic supporting the computation of sparse data can

reduce the number of MACs by half and realize the same amount of computation as a regular accelerator without sparsity mechanism. Regarding the Booth-value-confined (BVC) MP accelerator, an R8B-based MP multiplier is proposed with lower overhead in comparison to traditional multipliers. Prohibiting the ' ± 3 ' cases in both the model training and inference stages and eliminating the encoder on the basis of the feature of offline weights, can simplify multiplication in NN calculations, substantially optimizing the latency, area, and power in the PE array. Moreover, we propose an accelerator with 16 near-memory engines (NMEs) enabling the data-reuse of weights and activations for NN inference from inter-/intra-NME-order views. The weight memory is based on the near-memory computing (NMC) framework and is directly connected to the PE array to reduce the FF consumption and the path for data movement.

As for (2), Chapter 3 aims for the design and construction of SoC platforms. Three SoCs are realized on FPGA and ASIC platforms. In regard to the flash-based FPGA platform, using an open-source lightweight RISC-V core as a foundation, we construct an SoC with an MBW accelerator to create a practical AI hardware platform. Pertaining to the ASIC platform, we design two SoCs at the edge. Leaning on a lightweight RISC-V core, we propose a single-core AI processor incorporating a BVC MP accelerator with NMC-friendly data flow. Measurement on a 22nm testchip with 16 PEs shows a peak power efficiency of 29.11 TOPS/W at VGG16. Furthermore, we propose a multi-core multi-mode SoC for versatile safety-critical AI applications. Targeting different application scenarios, the four supported modes are: 1) accelerator mode, 2) multi-core mode, 3) fault-tolerant mode supporting dual-core-lock-step (DCLS), and 4) fully-access mode.

To evaluate the reliability of the proposed SoCs in Chapter 3, Chapter 4 performs fault injection experiments and radiation experiments for the proposed SoCs running AI applications. Pre-silicon fault injection simulation framework is proposed to simulate the ASIC design from the netlist view. Post-silicon fault injection frameworks are constructed to inject the errors into hardware platforms from the physical view. Also, neutron radiation experiment and alpha particle radiation experiment are performed to thoroughly explore the reliability of the proposed SoCs.

The rest of this dissertation is organized as follows. Chapter 2 presents the proposed AI accelerators and evaluates the performance on dedicated hardware platforms. Chapter 3 illustrates the design of SoC platforms at the edge and their performance. Chapter 4 explores the reliability of the proposed SoCs through fault injection and radiation experiments. Finally, Chapter 5 gives the overall concluding remarks of this dissertation.

Chapter 2

Design of AI Accelerator for Edge Computing

This chapter comprehensively explores multi-precision computing, approximate computing, systolic data flow, structural sparsity, and near-memory computing techniques. Three distinct AI accelerators are designed. A multi-bit-width accelerator is proposed to achieve power efficiency and data throughput improvement under MP computation. Moreover, a sparse NN accelerator is constructed with a data selector to explore the advantage of spare NNs. Finally, a Booth-value-confined MP accelerator is designed to evaluate the benefit of both approximate computing and MP computing. Besides, an NMC-friendly data flow is built in the third design.

2.1 Introduction

Many commercial accelerators for the inference stage have not disclosed their architectures (e.g., Apple's Neural Engine). The open-source NVIDIA Deep Learning Accelerator (NVDLA) project [71] has put in insufficient development efforts, and many existing issues remain unaddressed. In the design of accelerators for the inference stage, several issues need to be accurately modeled, such as 1) the size and universal structure of local memories, 2) memory allocation strategies for weights and activations, 3) the scale and efficiency of the PE array, and 4) optimal partitioning schemes for different NN layers.

Currently, few systems can comprehensively address all these issues. Existing systems can only cover parts of them. This chapter aims to analyze and solve these problems, and design NN accelerators for edge devices, specifically tailored for the vision domain. Through collaborative efforts between software and hardware, this accelerator can speed up various networks, such as LeNet, VGG, AlexNet, DarkNet, ResNet, and more.

In this chapter, three distinct AI accelerators are developed, each with unique ad-

vantages and capabilities tailored for specific computational needs in edge AI. The first is an MBW accelerator designed to explore the benefits of MP computing, supporting a wide range of numerical precisions. It features a systolic array data flow, enabling efficient and flexible computations across various bit-widths. This design allows for optimized performance and power efficiency by dynamically adjusting precision based on the computational requirements. The second accelerator is focused on enhancing NNs through sparse computation. This sparse accelerator supports broadcast-based data flow and leverages the inherent sparsity in weights and activations, significantly reducing the computational workloads and memory bandwidth requirements. This approach not only accelerates the inference process but also contributes to power savings, making it ideal for edge applications where resources are limited. The third accelerator is aimed at combining MP computing with approximate computing. The proposed BVC MP accelerator utilizes Booth-value constraints to achieve efficient MP and approximate calculations. The design supports NMC-friendly data flow, minimizing data movement and enhancing computational efficiency. This accelerator is particularly suited for scenarios where a trade-off between accuracy and performance can lead to substantial speed and power consumption gains. These accelerators collectively represent significant advancements in edge AI computing, offering tailored solutions for diverse computational challenges at the edge.

The rest section is organized as follows. Section 2.2 reviews the related work for AI accelerators. Section 2.3 introduces the proposed MBW Booth accelerator and discusses its technical details. Experimental results of this design with flash-based MPF300T FPGA implementation are presented. Section 2.4 provides the sparse NN accelerator with SMIC 28nm ASIC implementation compatible with structural sparsity in NNs. Section 2.5 applies approximate computing, MP computing, and NMC-friendly data flow into our proposed BVC MP accelerator. Experimental results of this design with TSMC 22nm implementation are offered. Finally, Section 2.6 discusses the proposed AI accelerators and Section 2.7 concludes this chapter.

2.2 Related work

Regarding the design of accelerators for NNs, there is a history of research as follows:

In 2010, J. J. Martinez [72] implemented an accelerator for a specific convolutional layer using high-level synthesis, surpassing the performance of contemporary general-purpose computers (dual-core, 2.4GHz frequency, pure software computation). This demonstrates the feasibility of hardware acceleration in NNs. In 2014, Vinayak Gokhale et al. [73] implemented accelerators for the entire NN using FPGAs, achieving significantly higher energy efficiency than numerous CPUs and GPUs. This illustrates the advantage of customized AI accelerators in the mobile domain.

In 2014, Y. Chen et al. proposed the DaDianNao architecture [25]. The DaDianNao structure stores all weights in on-chip high-density eDRAM, which is distributed

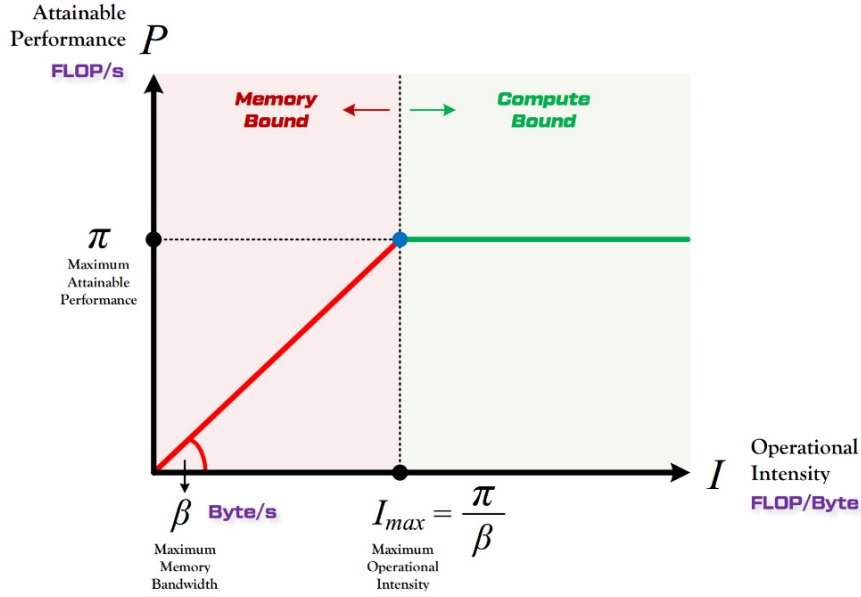


Figure 2.1: Roofline model.

throughout the chip, providing substantial internal bandwidth. Additionally, this storage approach results in significant storage overhead. Unlike many previous hardware designs with limited flexibility in pipeline structures, the authors adjusted the computation order of convolutional and fully connected layers, decomposing them into a series of fixed-size matrix multiplications. In terms of hardware, using a highly flexible DMA sends features and weight data to the PE array, and the output of the PE array is accumulated. In the pipeline structure design, convolutional and fully connected computations are typically implemented along the data flow. Generally, the flexibility of circuits based on pipeline design is relatively poor (such as restricting the size of convolutional kernels and not supporting variable convolution parameters, etc.). In contrast, the operation circuit based on the PE array addresses the problem of poor flexibility in pipeline structures through a flexible DMA. The fixed-size PE array can be used for computations in fully connected and convolutional layers, providing considerable computational capability [25].

In 2015, C. Zhang et al. [74] introduced the roofline model as shown in Fig. 2.1. Two key metrics of the computing platform are identified: Computing Power (π), also known as the performance limit of the computing platform, refers to the maximum number of floating-point operations per second (FLOPS) that a computing platform can achieve when fully utilized, and Bandwidth (β), also known as the bandwidth limit of the computing platform, refers to the maximum amount of memory exchange per second (in Bytes) that a computing platform can achieve when fully utilized. The computing platform's maximum operational intensity (I_{max}) is obtained by dividing these two metrics. It describes how many computations can be performed per unit memory exchange on

the computing platform. Its unit is FLOP/Byte. The formula for I_{max} is summarized as $I_{max} = \pi/\beta$. This figure highlights that, after a certain point, memory access becomes a bottleneck in the system's performance. Furthermore, the authors optimized memory access and tested their design on Xilinx VC707 development board. Their design achieves a performance of 61.62 GFLOPS on the VC707 board with a memory bandwidth of 1.55 GB/s and power consumption of 18.61 Watts. In 2015, Hiroki Nakahara et al. [75] proposed a pipeline structure based on shift register chains, which effectively cached input features for convolutional layers. However, this structure required the length of the shift register chain to be the same as the width of the input feature layer, limiting hardware flexibility.

In 2016, X. Han [76] expanded on the roofline model's concept and introduced various guiding methods to reduce memory requirements, such as on-chip caching of input data, convolution window reuse, and on-chip caching of temporary data. Additionally, they proposed using normalized memory access (MByte/GFLOP) to quantitatively describe the extent to which a hardware accelerator optimizes memory access. Also in 2016, R. Andri [77] proposed YodaNN. YodaNN has two main features: 1) It greatly reduced the generation of temporary data by adjusting the convolution computation order. Therefore, the circuit no longer required a large memory to store temporary data, and 2) It used binarized weights. Binarized weights significantly reduce the complexity of computation units and the external memory bandwidth required for weight shifting. This means that changing the data type can greatly reduce the circuit's complexity, thereby reducing power consumption and area. Meanwhile, considerable research indicate that using 8-bit or 16-bit data types does not have a significant impact on the accuracy of NN models. The utilization of low-precision computation has shown promise in significantly boosting power efficiency and warrants further exploration.

Google's Tensor Processing Unit (TPU) [19] has been a hot topic in recent years. TPU is a customized accelerator designed by Google for deep learning, specifically for TensorFlow. According to currently available information, TPU has a 256×256 size PE array, allowing for 65,536 operations of INT8 MAC per clock cycle. TPU also has a 24M-sized SRAM as an on-chip cache to store features, significantly reducing external memory access. Additionally, TPU includes circuits for pooling operations, local normalization operations, and nonlinear operations, enabling most computations required in a complete CNN and reducing the workload on the CPU.

At the end of 2017, NVIDIA introduced an open-source NN accelerator architecture called the NVDLA [71]. Similar to the DaDianNao and TPU architectures, NVDLA performs convolutional and fully connected layer computations through a Direct Memory Access (DMA) module and PE array. This indicates that the flexibility and generality of this structure have gained considerable recognition. Additionally, NVDLA includes a 512KB SRAM to cache activations/weights. As NVDLA is primarily targeted for the embedded domain, the on-chip SRAM size is much smaller than that of TPU. However, NVDLA is not yet mature at the time. The released versions has several

issues, such as low memory access efficiency, excessively large control circuit scale, timing discrepancies, and low efficiency of the MAC array. Moreover, some RTL codes in NVDLA are generated by scripts, resulting in poor readability, and it has not received much attention from the industry at present.

These hardware accelerator designs for NNs demonstrate advancements in performance and energy efficiency but face challenges such as limited flexibility in pipeline structures, low memory access efficiency, complexity in control circuits, and timing discrepancies. Achieving optimal performance and usability requires addressing these issues through improved architectural designs, including optimized arithmetic units for high-performance computing, data flow for seamless computation, and memory access strategies for high utilization of hardware. In light of these challenges, this dissertation will focus on proposing solutions to enhance hardware accelerator designs for NNs, aiming to pave the way for achieving optimal performance and usability in future hardware accelerators.

2.3 Design of Multi-bit-width Booth Accelerator

MP NN maintains the balance between network accuracy and hardware efficiency, thus enlightening a promising method for accurate yet energy-efficient edge computing. To satisfy the MP computation requirements, systematical study on MBW networks from high accuracy network generation to efficient hardware mapping is highly desired. Novel NN accelerators with both MBW data processing elements (PE) [14, 78, 79, 80] and high-efficiency systolic PE array [24, 19, 81] for the configurable MBW data flow are also required. In this section, we develop a state-of-the-art (SOTA) multi-bit-width accelerator for Neural Architecture Search (NAS) optimized deep learning NNs. The main contributions are summarized as follows:

- **Multi-bit-width Booth multiplier:** A hybrid multi-bit-width MAC unit is designed, enabling full utilization of both DSP and LUT resources during FPGA deployment.
- **Vector-style systolic data flow:** A novel vector-style systolic data flow is developed, integrating the strengths of the classical systolic array and parallel computing unit. This effective data flow facilitates the mapping of multi-bit-width CNNs onto hardware, achieving high performance.
- **NAS on MP NNs:** A systematic study on multi-bit-width networks is conducted, spanning from network generation (using NAS methods) to hardware mapping.

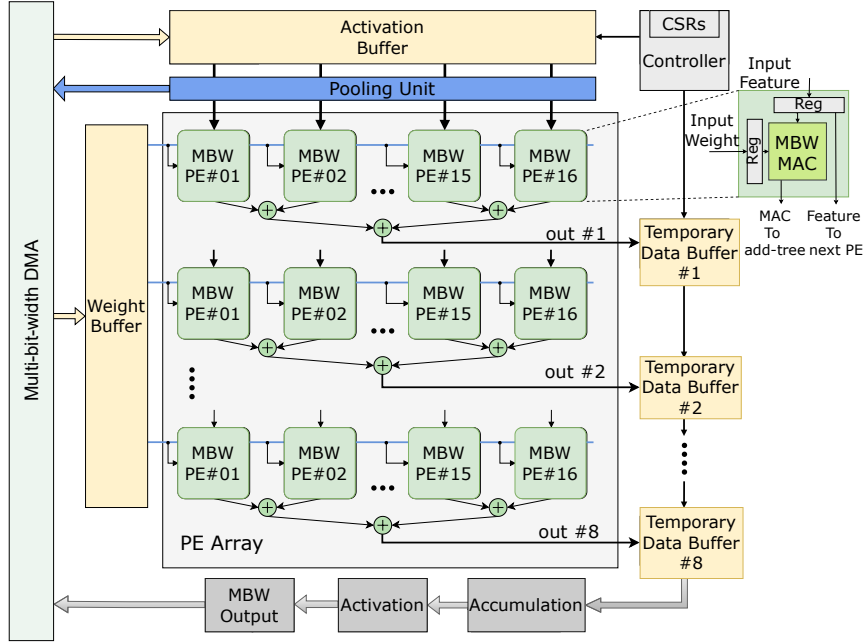


Figure 2.2: Architecture of multi-bit-width accelerator.

2.3.1 Implementation of MBW Booth Accelerator

To reduce power consumption and enhance energy efficiency, we propose a high-performance MBW vector systolic accelerator implemented on the Microsemi MPF300T Eval Kit FPGA platform. The choice of this platform is attributed to its flash-based FPGA architecture, which offers higher reliability compared to SRAM-based FPGAs as mentioned in Section 1.2.3.

The MAC unit in this accelerator is based on a Multi-precision Booth (MP-Booth) multiplier based on the Booth algorithm. Also, a vector systolic dataflow is proposed to improve the throughput of the whole system.

Fig. 2.2 shows the architecture of the MBW accelerator. The MBW DMA is used to interact with external memory for data exchange of weights and activations. The operating flow of our MBW accelerator is as follows: 1) Weights and activations are stored in DRAM, 2) Once the controller completes initialization, the MBW DMA reads data from DRAM and loads it into corresponding buffers sequentially, 3) The data is then sent to PE array of the convolutional unit or pooling unit, controlled by CSR values, 4) For pooling unit, the outputs are transmitted back to DRAM through the MBW DMA for further computations. For the convolutional unit, the outputs are first sent to temporary buffers until the accumulation operations are completed. Then, the final results will be transmitted back to DRAM.

Moreover, this design involves configuring various parameters of NNs, such as the weight/activation addresses, model size, and the number of slices. Once these param-

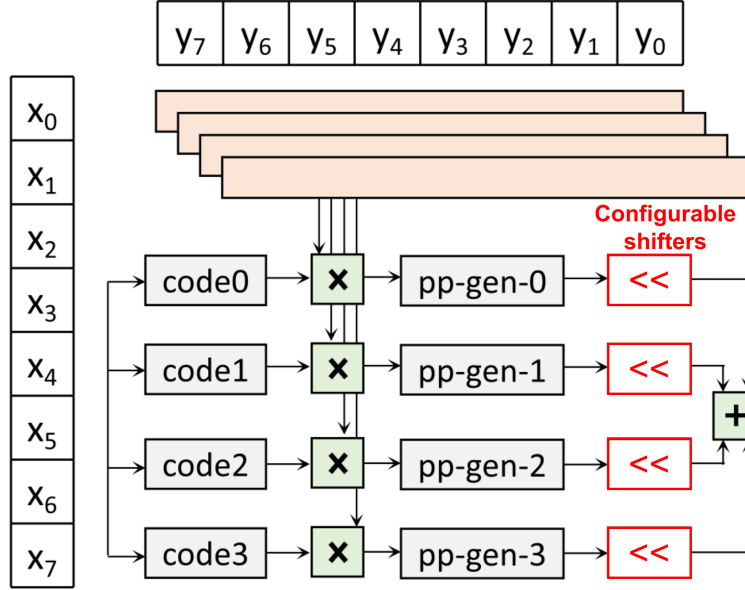


Figure 2.3: Multi-precision Booth multiplier.

ters in control and status registers (CSRs) have been set, one of the CSRs can be used to start the computation. The accelerator then conducts the computation according to these parameters, such as slicing the data into multiple rounds of calculation. Once the computation is completed, the results are sent back to the external memory. The accelerator then updates the CSR with a completion signal. This completion signal can be used for various purposes, such as triggering other processes or allowing the microcontroller to retrieve the outputs. Using CSRs to manage the computation process provides a convenient and efficient way to conduct AI model computations. Additionally, it facilitates seamless synchronization between the accelerator and microcontroller.

MP-Booth Multiplier

The Booth multiplier presents itself as a viable alternative for the multi-bit-width PE. Traditionally designed for fixed-width multiplication, the classical Booth multiplier can be adapted for MP computation with an intuitive approach. The key idea involves reusing the same hardware for various computation modes, eliminating the need for separate INT2, INT4, and other multipliers. Through efficient hardware reuse, this unit can perform 1 INT8 multiplication, 2 INT4 MACs, or 4 INT2 MACs. The Booth code generation for different computation modes (INT2/4/8) is based on the radix-4 Booth (R4B) algorithm, whose structure is illustrated in Fig. 2.3. The detailed mechanism unfolds as follows: Firstly, four Booth codes are generated irrespective of input bit-width. Then, these codes and the multiplicands undergo multiplication to yield four partial products. Regardless of bit-width modes, the corresponding partial products un-

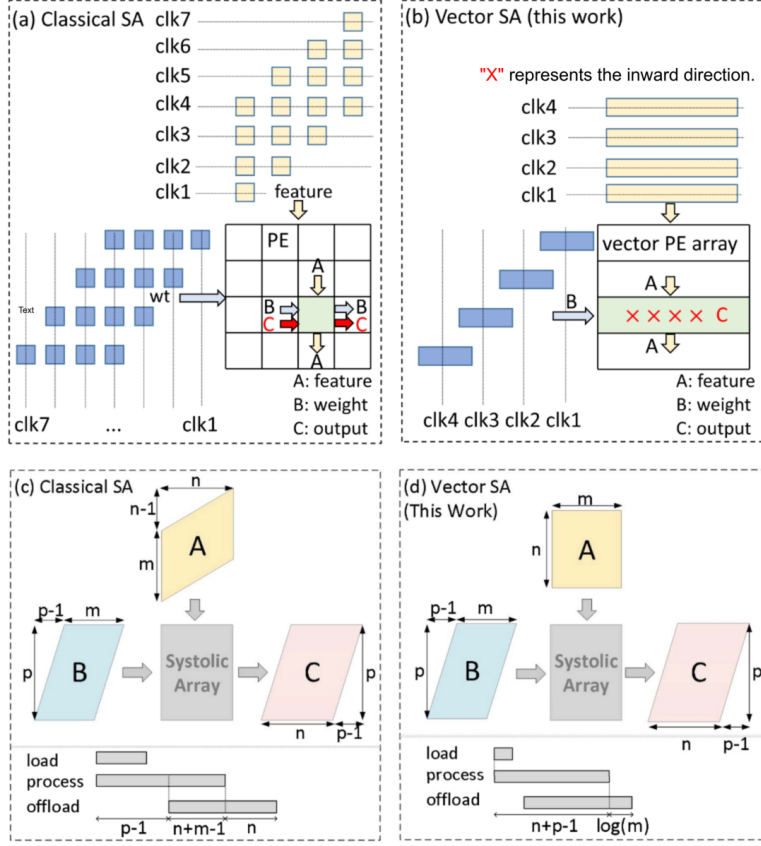


Figure 2.4: Data flow in different systolic architectures (a) and (c) Classical SA. (b) and (d) Proposed vector SA.

dergo configurable shift processing according to the calculation precision, culminating in their addition for the final results.

For INT8 computation, involving 16 input feature data and corresponding weight data, the "INT8×INT8" multiplication is controlled in one clock, with a subsequent 3-clock delay for the following adder tree (for summation of 16 inputs). In the case of INT4 computation, where there are 32 input feature data and corresponding weight data, the "INT4×INT4 + INT4×INT4" is controlled in one clock, reusing the adder tree for accumulation of the 16 temporary results. Similarly, for INT2 computation with 64 input feature data and corresponding weight data, the "INT2×INT2 + INT2×INT2 + INT2×INT2 + INT2×INT2" is controlled in one clock, with the adder tree being reused for accumulation of the 64 temporary results. Consequently, the total input-output delay for different data-width modes remains consistent at 4 clock cycles.

Vector Systolic Array

To clearly describe the working principle in systolic array architectures, we explore the following matrix multiplication as an example: $A[n][m] \times B[m][p] = C[n][p]$, where $A[n][m]$ is the input feature matrix, $B[m][p]$ is the weight matrix, and $C[n][p]$ is the output matrix. From the aspect of convolutional operation, p is the number of output channels (CH_{out}) which is independent of each other; m is the number of input channels (CH_{in}) that will be summed together in the matrix multiplication; n is the number of input feature pixels that share the same weight matrix in convolution. The operating process in systolic arrays contains three phases: feature/weight data loading, MAC processing, and output result offloading.

In classical systolic array (classical SA) (Fig. 2.4(a)) [22, 23]: The inputs (feature matrix and weight matrix) pass through the whole systolic array in two different directions step-by-step, and none of them will stay in the PEs during the execution. Upon the arrival of new inputs, each PE multiplies the two inputs coming from its left and up neighbors and adds them to the prior accumulated results. To guarantee the correctness of computations, the inputs must arrive at each PE at the absolutely right time, thus the timeline control module is much more complex than the row-by-row style. Due to the one-by-one loading strategy in classical SA, the total loading time is relatively longer, namely $T_{load} = n + m + p - 3$. Besides, the multiplication can start as soon as the first elements of the input arrive at the first PE. However, to finish the multiplication, all elements of the inputs feature and weight must pass through the PEs completely, thus the number of processing time steps is $T_{process} = n + m + p - 2$. The output latency equals to the time step moving from the top-left corner to the bottom-right corner in systolic PE arrays, namely $T_{offload} = n + p - 1$. In consideration of the time overlapping, the total latency is $T_{total} = 2n + m + p - 2$.

In the proposed vector systolic array (vector SA) (Fig. 2.4(b)), the weight data is horizontally loaded to the systolic array in a row-by-row style. At each cycle, only one new row of weight data will be uploaded to the PEs. Meanwhile, all of the others are stationary and will remain for several cycles (equal to the width of the feature matrix, namely n). The input feature data is non-stationary and will pass through the systolic array clock-by-clock. To guarantee the correctness of computations, the input features and weights must arrive at each row at the right time. In such a row-by-row systolic style, the computation process can start as soon as the first row of data is loaded, therefore the total loading time is $T_{load} = p$. The processing time equals the number of time steps that move feature data (with length n) over the systolic array (with length p), therefore the processing delay is $T_{process} = n + p - 1$. The output offload process will start as soon as the first result is generated from the adder tree. In consideration of the overlapping between process and output, the total time delay in our design is $T_{total} = n + p + \log(m) - 1$. Besides, the performance of two arrays will be compared in Section 2.3.2.

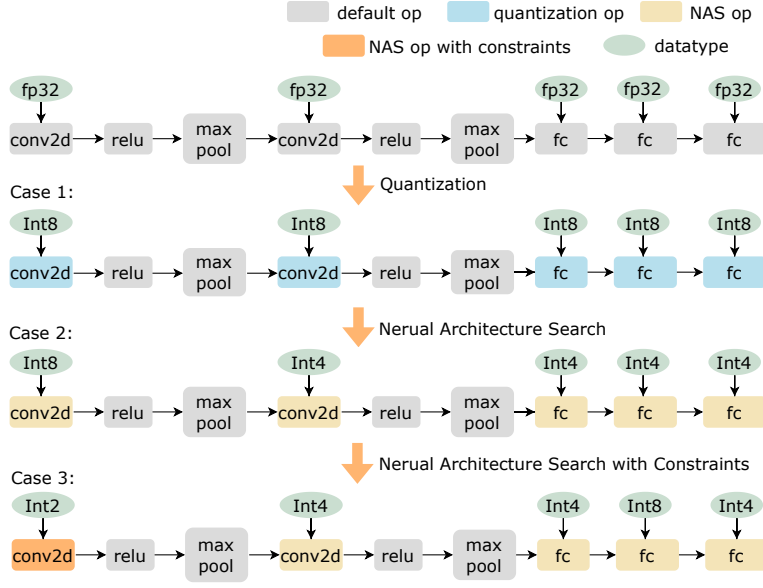


Figure 2.5: NAS flow of multi-bit-width LeNet5.

Table 2.1: Precision distribution on three different MBW models.

Layer	Lenet5 Backbone		INT8 model		INT4/8 model		INT2/4/8 model	
	Activation (NCHW)	Weight (OIHW)	Precision (Bit)	Weight Size (Byte)	Precision (Bit)	Weight size (Byte)	Precision (Bit)	Weight size (Byte)
Conv1	1×1×32×32	6×1×5×5	8	150	8	150	2	37.5
Pooling1	1×6×28×28	-	8	-	8	-	2	-
Conv2	1×6×14×14	16×6×5×5	8	2400	4	1200	4	1200
Pooling2	1×16×10×10	-	8	-	4	-	4	-
FC1	1×16×5×5	120×16×5×5	8	48000	4	24000	4	24000
FC2	1×120×1×1	84×120×1×1	8	10080	4	5040	8	10080
FC3	1×84×1×1	10×84×1×1	8	840	4	420	4	420
Total	-	-	-	61470	-	30810	-	35737.5
Accuracy	-	-	98.48%		95.42%		90.25%	

MP NN Training and Generation

To evaluate how MBW data impact the performance of CNNs, we prepare three cases with different precision distributions. The NAS flow based on differential NAS approach [13] for training MBW LeNet5 is shown in Fig. 2.5.

First, LeNet5 is trained on GPU and generate a full-precision model with FP32 weights. Second, to get a single-precision model of INT8 weights, all the FP32 weights are quantized into INT8 and re-trained to maintain accuracy as much as possible. In this case, FP32 is regarded as INT8 ranging from -128 to 127 with certain constraints (INT8 model). Third, the differential NAS strategy is applied to generate optimized MBW LeNet5 automatically. The obtained model consists of INT4 and INT8 (INT4/8 model). However, the precision distribution of the generated MBW LeNet5 does not

Table 2.2: Resource utilization of MP-Booth MAC on FPGA.

Utilization	2D D&C ST	SWP ST	Radix-4 Booth	MP-Booth
LUT	7230	4242	4641	2634
FF	5535	2060	550	278
DSP	0	0	0	48

contain INT2, indicating that INT2 data adversely affects the performance of LeNet5. Conversely, this work aims to analyze the reliability of comprehensive precisions, including INT2, INT4, and INT8. Therefore, special constraint items are introduced into the training method to generate a special case (INT2/4/8 model) for LeNet5.

These three models are shown in Table 2.1. INT4/8 model achieves about 50% weight size reduction from INT8 model with 3% accuracy loss. The differential NAS-optimized method has a balanced trade-off between accuracy and weight size. On the other hand, the top-1 accuracy of the INT2/4/8 model has a larger loss, and the weight size is even larger compared with the INT4/8 model. Although this model will not be adopted in terms of performance, we use it as an aggressively-quantized MBW model for reliability and performance analysis in Chapter 4.

2.3.2 Evaluation of MBW Booth Accelerator

Multiplier Evaluation

As mentioned in Section 1.2.1, the 2D D&C ST and SWP ST are two common strategies for MP computation. However, the limitations of the 2D D&C ST and SWP ST strategies stem from their reliance on complex bit split and combination schemes. These schemes involve a substantial number of temporary variables for bit-level data reconstitution, which can complicate the implementation process and reduce computational efficiency. Consequently, these strategies incur higher costs in logic and memory resources during FPGA implementation. Additionally, a fair comparison summarized in Table 2.2 has been conducted in the FPGA among different MP MAC units, confirming the advantage of the proposed Booth method in terms of efficiency. The comparison is performed under identical throughput and computational precision conditions for all units, ensuring that observed differences are solely due to design and implementation variations.

Data Flow Evaluation

Moreover, according to the discussion of SA in Section 2.3.1, the detailed analysis of systolic delay can be checked in Table 2.3. Compared to classical SA data flow, our proposed vector SA data flow has lower latency and high throughput. Specifically,

Table 2.3: Comparison of the delay in different systolic arrays.

Systolic Delay Analysis	Classical SA	Vector SA (this work)
PE type	multiplier-add	MBW MAC adder tree
T_{load}	$n + m + p - 3$	p
$T_{process}$	$n + m + p - 2$	$n + p - 1$
$T_{offload}$	$n + p - 1$	$n + \log(m) + p - 2$
Total Delay	$2n + m + p - 2$	$n + \log(m) + p - 1$

Table 2.4: Resource utilization of MBW accelerator on Microsemi MPF300T Eval Kit.

Component		Fabric 4LUT	Fabric DFF	Interface 4LUT	Interface DFF	Math (18×18)	uSRAM	LSRAM
Accelerator (100 MHz)	Conv Unit	31339	21839	13032	13032	136	54	208
	Pool Unit	33250	26064	408	408	4	22	0
	DMA	375	75	336	336	0	28	0
	Interface	71	55	0	0	0	0	0
	Total	65035	48033	13776	13776	140	104	208

assuming the size of both matrices being multiplied is 64×64 , and the size of the PE array is also 64×64 , then the delay for the classical SA would be 254 clocks, while for our proposed Vector SA, the delay would be 133 clocks with about 50% improvement.

Performance Evaluation

Considering the available resources in MPF300T Eval Kit, we build a 16×8 array, which means that the parallelism of inputs is 16, 32, and 64 for INT8, INT4, and INT2, respectively, and the number of output channels is 8. Besides, the resource utilization of our proposed MBW Booth accelerator is shown in Table 2.4. Note that a 16×8 PE array implemented on MPF300T FPGA achieves the peak performance of the convolutional layer can reach 25.6 Giga operations per second (GOPS) for INT8, 51.2 GOPS for INT4, and 102.4 GOPS for INT2, respectively. Thus, the adopted MBW accelerator is competitive at this moment.

In summary, this accelerator carries out a systematical study on MBW networks from network generation (NAS method) to hardware mapping. Firstly, the novel vector-style systolic accelerator is proposed, which combines the advantages of the classical systolic array and parallel computing unit. With the vector-style data flow, the processing time and logic resources can be reduced compared to the classical systolic array. Secondly, a hybrid MBW MAC unit is designed, thus both DSP and LUT resources can be fully utilized during FPGA deployment. Finally, the proposed method has been practically deployed on the MPF300T FPGA platform using a NAS-optimized network as a case study, indicating great potential for high performance and energy-efficient data processing at the edge.

2.4 Design of Low-power Sparse NN Accelerator

NN inference on edge devices is widely studied since NN models are becoming more complex with larger parameters. Currently, state-of-the-art NNs [82, 83] often contain up to hundreds of megabytes of weight parameters and over 600K operations per input pixel. Such large networks require huge computing power in hardware. Constrained by the limited hardware resources of edge devices, CNN inference accelerators need high energy efficiency and area efficiency in general matrix multiplication (GEMM) which accounts for more than 90% of CNN calculations. Namely, more than 90% NN computations are occupied by MAC operations [84].

Data sparsity has drawn great attention since it can be exploited in NN inference accelerators [85] as zero elements reduce the computation and storage cost significantly. Meanwhile, structural sparsity, which is based on the traditional sparse algorithm with modified penalty terms, aims to organize more regular matrices. Besides, to run these NNs, multipliers are necessary for hardware platforms to compute the model data. The radix-4 Booth algorithm can improve the performance of multiplication because it reduces the number of partial product (PP) rows by half [86]. Therefore, facing the challenges of high MAC computation costs in hardware and the large computational load of models, this accelerator will leverage sparse structure design to reduce the computational burden of models on hardware devices. Additionally, it will streamline MAC design to achieve lower computational overhead.

In addition to MP computation mentioned in Section 2.3, simplifying MAC units using the Booth algorithm and reducing the workload of NN models via sparse architecture are two other effective strategies to achieve energy-efficient computation. Thus, this section introduces a novel sparse CNN accelerator with a custom radix-4 Booth multiplier with high data-reuse performance. The main contributions are summarized as follows:

- **Pre-encoding radix-4 Booth multiplier:** A pre-encoding radix-4 Booth algorithm, highly compatible with data reuse and weight offline pre-processing, is proposed. Reducing the bit number of PPs and merging eight encoders into one pre-encoding module based on the features of our accelerator, could simplify multiplication in CNN calculations, which reduces area and power consumption.
- **Sparse CNN accelerator:** For low-power design, the weights above 50% sparsity are steered into arithmetic units, such that each block supporting the computation of sparse data can reduce the number of MACs by half and realize the same amount of computation as regular accelerator without sparsity mechanism.

2.4.1 Implementation of Low-power Sparse NN Accelerator

Structural sparsity refers to a form of sparsity in NNs where the non-zero elements are organized in a regular, structured pattern rather than being randomly distributed. This

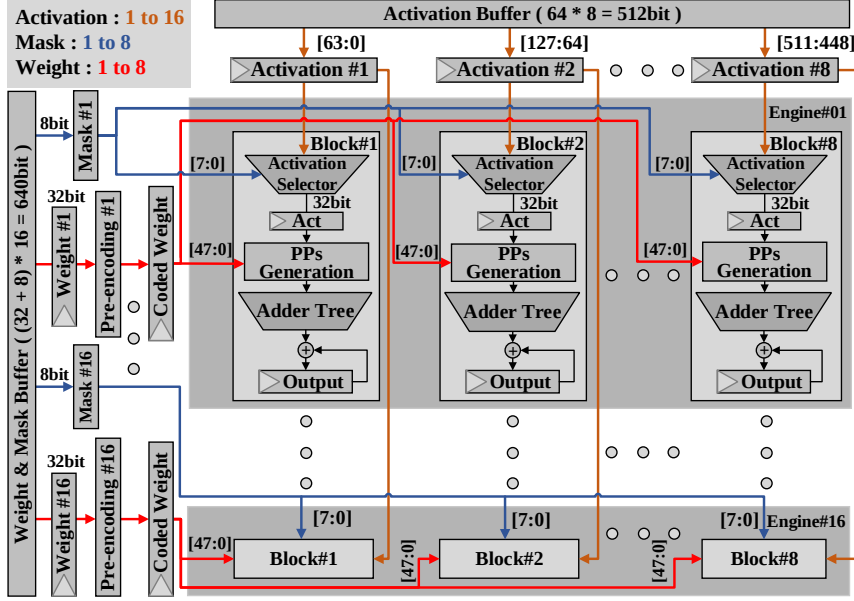


Figure 2.6: The architecture of proposed sparse NN accelerator.

regularity can simplify the process of leveraging sparsity for computational efficiency, making it easier to implement and optimize hardware accelerators.

Fig. 2.6 shows the architecture of the proposed sparse accelerator consisting of 16 engines. Each engine including 8 blocks can steer the same weights, corresponding control signals, and different activations into these 8 blocks. Each block including 4 MACs can select activations corresponding to non-zero weights and realize the computation operations in channel dimension. The 4 MACs integrated into the PP's generation and adder tree modules can realize the operation of the sum of 4 multiplications. Different blocks in the same engine can generate different pixels in the same output channel, and different engines can generate pixels in different output channels. As shown by the red and yellow lines in Fig. 2.6, the parallel processing is implemented such that each weight is shared to drive 8 blocks to generate 8 pixels in the same output channel and each activation is shared to drive 16 blocks having the same number in 16 different engines to generate pixels in different output channels simultaneously.

This design introduces two novel methods and refers to two existing strategies to achieve low-power computation. Firstly, we separate the encoders from the multipliers and combine them into a single pre-encoding module to drive 8 blocks, reducing $7\times$ area of the encoders. Secondly, we propose a method to reduce the bit number of resultant PPs by exploiting the feature of the radix-4 Booth algorithm and offline weights pre-processing. Then, we propose a spatially-unrolled sparse architecture similar to [87] and improve it to support a full range of sparsity from 0 to 100%. Finally, we apply a low-power adder tree design [88] to the PPs accumulation and design some special standard cells to further reduce the power consumption.

Table 2.5: Radix-4 booth algorithm.

y_{2i+1}	y_{2i}	y_{2i-1}	M_i	Operation	neg_i	two_i	one_i
0	0	0	0	+0X	0	0	0
0	0	1	+1	+1X	0	0	1
0	1	0	+1	+1X	0	0	1
0	1	1	+2	+2X	0	1	0
1	0	0	-2	-2X	1	1	0
1	0	1	-1	-1X	1	0	1
1	1	0	-1	-1X	1	0	1
1	1	1	-0	-0X	1	0	0

Pre-encoding Radix-4 Booth Multiplier

The radix-4 Booth algorithm can facilitate the multiplication and improve the timing of the circuit. Taking $X(x_{n-1}x_{n-2}\dots x_0) \times Y(y_{n-1}y_{n-2}\dots y_0)$ as an example, an n -bit multiplication of radix-4 Booth algorithm can be expressed as follows:

$$X \times Y = \sum_{i=0}^{(n/2)-1} M_i 2^{2i} X.$$

The coefficient M_i is shown in Table. 2.5. A pre-encoded mechanism of radix-4 Booth algorithm is proposed in [89] for low-power multiplier design via detecting the ‘ $\pm 0X$ ’ case and turning off encoders and decoders, which is a multiplier-level design.

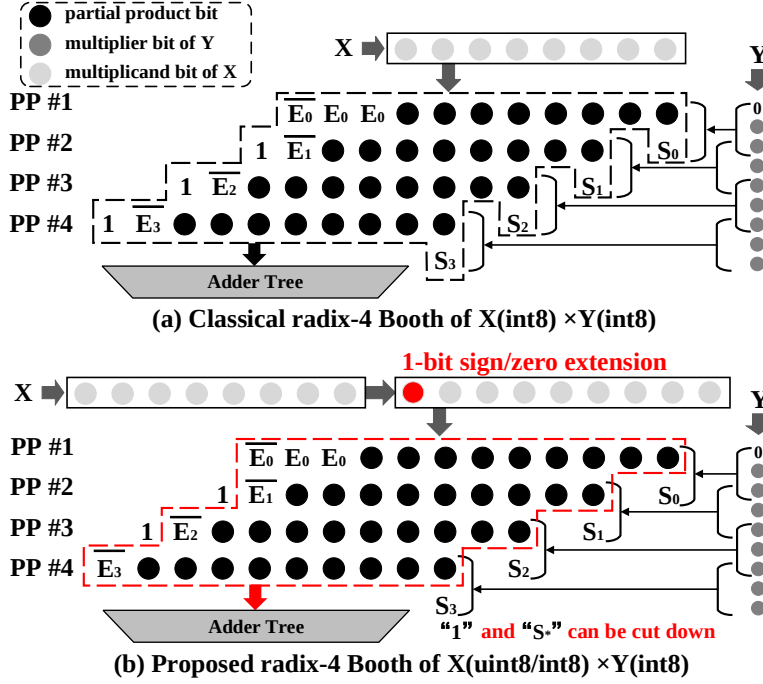
To pursue a system-level low-power design, we propose a new pre-encoding structure that takes advantage of the feature of the radix-4 Booth multiplier, the data-reuse characteristic of the accelerator, and the pre-processing of weights.

To support both *unsigned* $\times$ *signed* and *signed* $\times$ *signed* required in convolutional computation, we design a solution that extends 1-bit sign/zero in the highest bit of multiplicand. Taking *uint8* $\times$ *int8* and *int8* $\times$ *int8* as an example, the PPs generation of 8-bit multiplication with radix-4 Booth algorithm is given by:

$$\sum_{i=0}^{4-1} (-1)^{sign(M_i)} 2^{2i} |M_i| \text{concat}\{sign(X) \& is_sign, X\},$$

where *concat*{ } is bit-concat operation, *sign*() returns the sign bit of data, and *is_sign* is subjected to:

$$is_sign = \begin{cases} 1, & \text{if multiplicand is signed,} \\ 0, & \text{otherwise.} \end{cases}$$

Figure 2.7: Partial products of 8×8 radix-4 Booth multiplier.

Letting N_i be $|M_i| \text{ concat}\{\text{sign}(X) \& \text{is_sign}, X\}$, we can get:

$$\sum_{i=0}^3 2^{2i} ((\text{sign}(M_i) ? \bar{N}_i : N_i) + \text{sign}(M_i)).$$

Assuming $K_i = (\text{sign}(M_i) ? \bar{N}_i : N_i)$, and the bit width of K_i is 10, we can get:

$$\sum_{i=0}^3 2^{2i} (K_i + \text{sign}(M_i)).$$

Supposing $E_i = \text{sign}(K_i)$ and $S_i = \text{sign}(M_i)$, we could get the PPs on the basis of sign extension structure [90]:

$$\begin{aligned} PP_S = & [(\text{concat}\{\bar{E}_0, 2\{E_0\}, K_0[8:0]\} + S_0)] \rightarrow PP\#1 + \\ & [(\text{concat}\{1, \bar{E}_1, K_1[8:0]\} + S_1) < 2] \rightarrow PP\#2 + \\ & [(\text{concat}\{1, \bar{E}_2, K_2[8:0]\} + S_2) < 4] \rightarrow PP\#3 + \\ & [(\text{concat}\{\bar{E}_3, K_3[8:0]\} + S_3) < 6] \rightarrow PP\#4. \end{aligned}$$

Fig. 2.7 illustrates the PPs of $8\text{bit} \times 8\text{bit}$ discussed above. ' E_i ' is the sign bit of each PP, '1' is a normal constant, and ' S_i ' is the sign of M_i . Since the formula of the

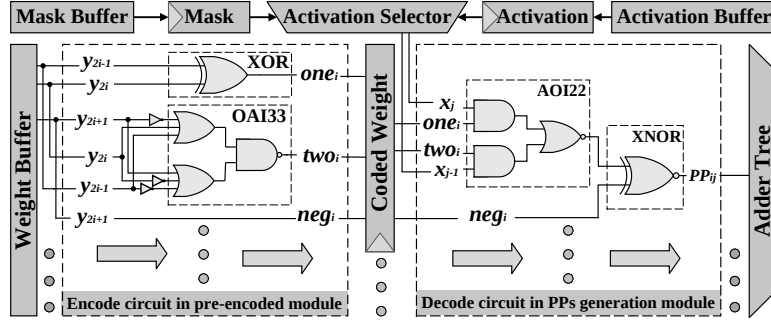
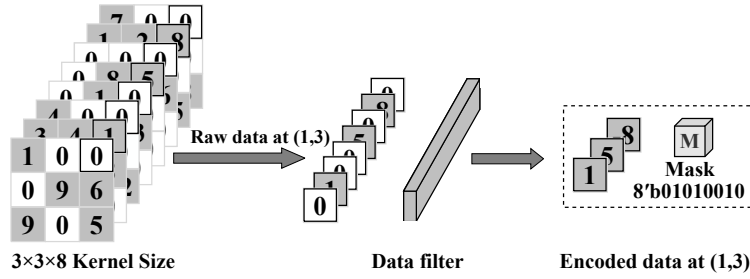
Figure 2.8: Gate level of *one/two/neg* radix-4 Booth code.

Figure 2.9: Weight offline encoding in the channel dimension.

convolutional layer is $w \times x + b$ and the weights can be pre-processed offline, we propose to add '1' and ' S_i ' to bias parameter b beforehand. Namely, an offset value added to bias b offline for adjustment per clock cycle can be expressed as follows:

$$offset = 5 * 2^{14} + \sum_{m=1}^4 \sum_{i=0}^3 2^{2i} S_i^m,$$

where m is the number of multipliers in each block. As a result, the parts surrounded by the dotted line in Fig. 2.7 are summed up in the accelerator, which reduces the bit number of PPs. The bit numbers of each PP (#1 to #4) given to the adder tree are 12, 10, 10, and 10, respectively. Namely, the bit number that this approach could reduce in each multiplier is 6.

To further reduce the power consumption, we propose to extract the encoder from the radix-4 Booth multiplier as a pre-encoding module. Fig. 2.8 shows its gate-level circuit. The outputs of the pre-encoding module can be shared by the 8 blocks to which the same weight is given as shown in Fig. 2.6 because the outputs of the pre-encoding module depend on the weight value, not on the activation value.

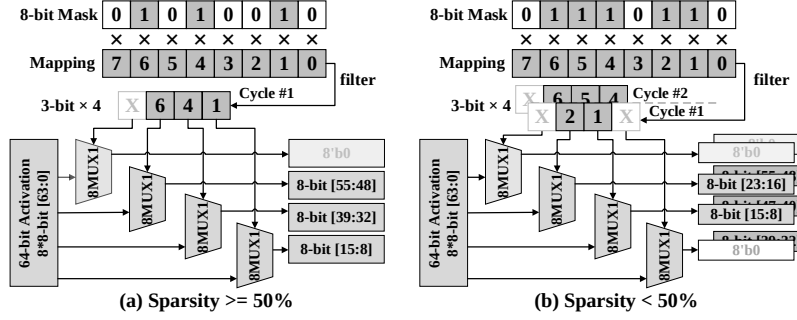


Figure 2.10: Activation selector at different sparsity.

Activation Selector

Basically, CNNs contain four kinds of structural sparsity from irregular level to regular level (i.e., fine-grained, vector-level, kernel-level, filter-level). Regular sparsity makes the design of the hardware accelerator easier. To exploit the structural sparsity at different levels, a flexible accelerator is necessary.

A sparse PE [91] accepts compressed sparse column (CSC) format to skip zero elements. However, the control logic of PE is too complex. A time-unrolled systolic structure [87] achieves 100% utilization of MACs. However, this structure increases not only register accesses for data accumulation but also the number of registers in the data flow. On the other hand, since the weights are known in advance, we could pre-process the raw weights. Based on [87], Fig. 2.9 gives an example of using a kernel size = $3 \times 3 \times 8$. The weights are coded by removing the zero elements and appending the bitmask M indicating the location of non-zero elements. M signal can be shared by the eight blocks as depicted in Fig. 2.6.

Besides, a spatially-unrolled structure [87] supporting variable sparsity density bound block (VDBB) is presented. However, it does not support the sparsity under 50%. We then propose an improved activation selector to support a full range of sparsity. As shown in Fig. 2.10, this selector can complete the filtering of activations using 8-bit mask data. We adopt a 1-bit control signal in each engine to set whether the data could be computed within one or two clocks. When the sparsity is greater than or equal to 50%, the selector can finish selecting eight data within one clock cycle (Fig. 2.10(a)), otherwise in two consecutive clock cycles (Fig. 2.10(b)). Firstly, we map 8-bit mask into 3-bit $\times$ 4 data as selection signals for 8:1 multiplexers (8MUX1s). Secondly, we integrate four 8MUX1s into our selector selecting at most four 8-bit data from eight activations using four 3-bit selection signals. Finally, the selected data is output. In addition, when some engines execute one-cycle computations with sparsity above 50% and others execute two-cycle computations with sparsity under 50%, one-cycle computation engines are gated for one extra cycle.

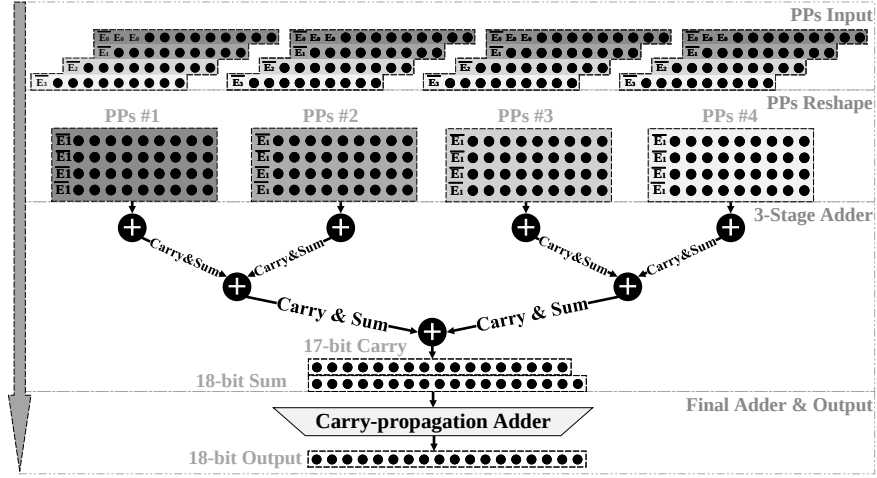


Figure 2.11: Improved Wallace tree based on customized standard cells.

Table 2.6: Truth table of FAN and HAN.

A	B	Cin	FAN_CoN	FAN_SN	HAN_CoN	HAN_SN
0	0	0	1	1	1	1
0	0	1	1	0	1	1
0	1	0	1	0	1	0
0	1	1	0	1	1	0
1	0	0	1	0	1	0
1	0	1	0	1	1	0
1	1	0	0	1	0	1
1	1	1	0	0	0	1

Adder Tree

An optimized Wallace tree [88] is proposed for the accumulation operation, utilizing carry save addition (CSA) to eliminate carry propagation delay. This strategy is incorporated into our design to construct the adder tree structure, as illustrated in Fig. 2.11. To reduce the bit extension during the accumulation phase, we first reshape the PPs from #1 to #4, then add the same level of PPs up at the first stage. Then, to optimize the timing of the adder tree, we only generate the carry and sum outputs at each stage using CSA, avoiding carry propagation. Finally, we use a carry-propagation adder (CPA) to calculate the final result once we get the final two numbers.

To further reduce the power and area consumption, we develop special standard cells to form an adder tree. Full adder (FA) and half adder (HA) are the most frequently used standard cells in our adder tree. Taking FA with CoN and SN (FAN) and HA with CoN

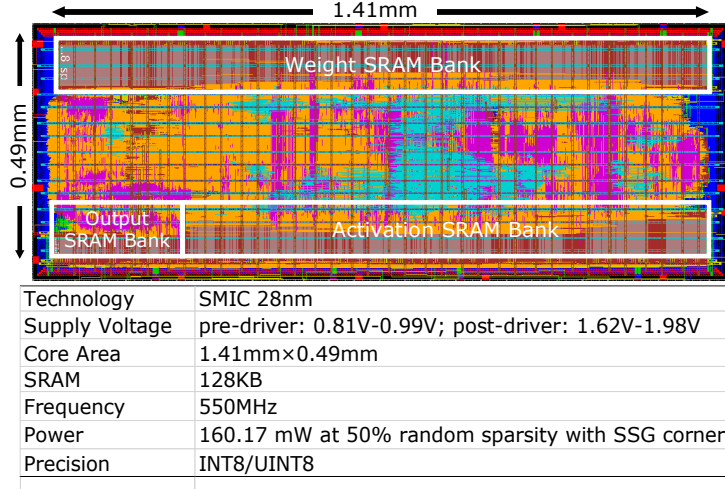


Figure 2.12: Post-layout result of the proposed accelerator.

and SN (HAN) as an example, the truth table is shown in Table 2.6, and the output of FAN and HAN is the invert of carry (CoN) and the invert of sum (SN), which could reduce the number of transistors in our circuit for invert operation. We adopt these cells (e.g., FA, FAN, HA, HAN) to construct the adder tree collaboratively.

2.4.2 Evaluation of Low-power Sparse NN Accelerator

Methodology

The proposed accelerator is designed by Verilog HDL language and synthesized for SMIC 28nm using Synopsys Design Compiler. The clock frequency is set to be 550MHz with 20% uncertainty, and the input/output latency is set to be 65% clock period. Besides, we develop the above-mentioned standard cells with specified functions and generate the relevant libraries. Furthermore, the layout design is implemented using Cadence Innovus with two corners: 1) 0.81V/125C SSG corner with RC-worst case, and 2) 0.99V/0C FF corner with RC-best case. The power, performance, and area (PPA) values are obtained using Design Compiler, PrimeTime, and Innovus.

The chip layout and its implementation results are shown in Fig. 2.12. The core area is 1.41mm×0.49mm with 128KB SRAM including 62.5KB weight SRAM, 50KB activation SRAM, and 15.5KB output SRAM. The typical pre-driver voltage is 0.81V, and the power consumption is 160.14mW with 50% sparsity at 550MHz based on the SSG corner and RC-worst case.

Table 2.7: Performance at 50% random sparsity.

Component	Power, mW	Area, mm ² (%)
Pre-encoding Module	^s 0.762/ ^p 1.112	^s 1.860e-4 (0.038%)
Activation Selector	^s 12.768/ ^p 25.796	^s 1.784e-2 (3.686%)
PPs Generation	^s 14.592/ ^p 24.265	^s 2.368e-2 (4.893%)
Adder Tree	^s 28.672/ ^p 56.100	^s 4.506e-2 (9.311%)
Reg/Buffer/Controller/Oscillator	^s 33.906/ ^p 41.715	^s 3.318e-2 (6.856%)
SRAM (128KB)	^s 10.010/ ^p 11.182	^s 0.3640 (75.215%)
Total	^s 100.71/ ^p 160.17	^s 0.4839 (100%)

^sSynthesis result; ^pPost-layout result

Table 2.8: Performance analysis in this work.

Case	Area (mm ²)	Power Efficiency (TOPS/W) at different sparsity		
		0%	50%	87.5%
Case 1	0.5975	7.1429	8.2090	12.7034
Case 2	0.5655	8.2090	9.8214	17.5826
Case 3	0.4864	7.0777	11.1354	24.0348
This work	0.4839	7.1088	11.1846	24.1410

*All the data is synthesis results in this table.

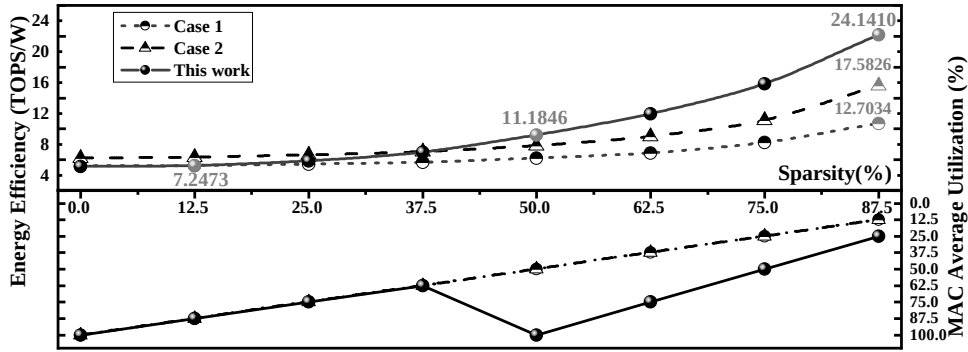


Figure 2.13: Power efficiency and MAC utilization at different channel-level sparsities.

Performance Analysis

With 50% random weight sparsity, the PPA metrics of each part are provided in Table 2.7. Considering the computing amounts and convolutional layer size of NNs (e.g., VGG16, AlexNet), setting the size of SRAM to 128KB is appropriate and could re-

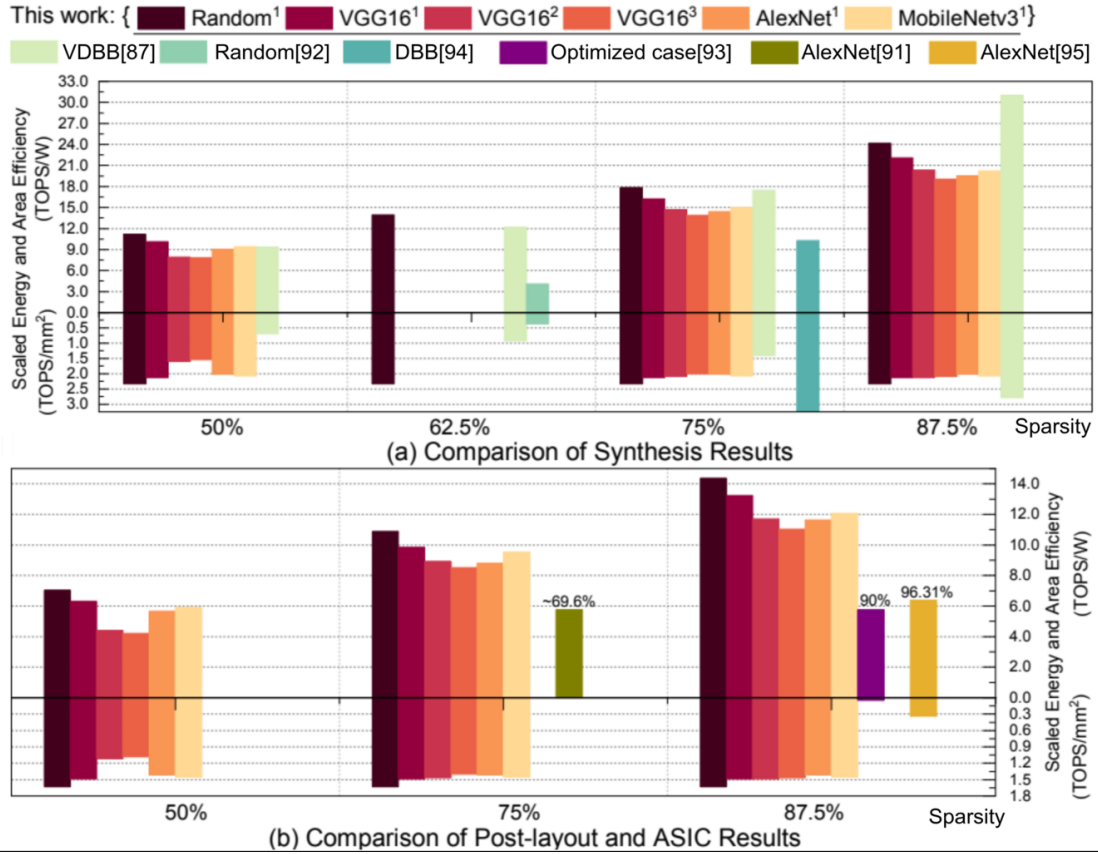
duce memory access. To evaluate the performance of our multiplier and accelerator, we construct three cases for comparison. Case 1, whose block realizes the sum of eight multiplications, adopts the traditional 8×8 multiplier generating eight PPs without a pre-encoding module and sparsity mechanism. Case 2, whose block realizes the sum of eight multiplications without a sparsity mechanism, adopts a pre-encoding radix-4 Booth multiplier. Case 3, whose block realizes the sum of four multiplications with a sparsity mechanism, adopts the radix-4 Booth multiplier without sharing the pre-encoding module. The area and power efficiency metrics are shown in Table 2.8. Specifically, the cases without sparse mechanism have a higher area consumption but similar power efficiency compared to the cases with sparse mechanism when the data sparsity is less than 50%. In contrast, the power efficiency has a significant improvement in the cases with sparse mechanisms when the data sparsity is greater than 50%. Also, the pre-encoding module has a slight optimization for area reduction. The synthesis results of power efficiency and MAC average utilization from 0% to 87.5% weight sparsity are shown in Fig. 2.13. These two cases without sparse mechanism have to compute more zero elements especially when the data sparsity is greater than 50%. Thus, this design has higher MAC utilization compared to these cases without a sparse mechanism. In summary, the proposed accelerator achieves a 90.03% improvement in energy efficiency compared with case 1 and the highest power efficiency (24.1410 Tera operations per second per Wate (TOPS/W)) at 87.5% sparsity.

Comparison with Prior Work

Table 2.9 lists the metrics of the state-of-the-art sparse CNN accelerators for evaluation. To compare with these INT8 accelerators, we scale the area and power efficiency of these works into our design as references following the approach of [96, 97]. From 50% to 75% weight sparsity, our design offers higher area and power efficiency than others. Moreover, by applying the quantization [98] and group sparsity [99] strategies to channel-level CNN training, we get some benchmarks (i.e., VGG16, AlexNet, MobileNetV3) with different sparsity. Also, we train these CNNs based on VDBB [87] and random format for comparison. Since this design focuses on channel-level sparse computation, it achieves higher power efficiency using channel-level structural sparsity methods compared with VDBB and Random sparsity methods. Besides, the bit width of quantized weights and activations is 8. Also, since the minimum computational unit for each block is 4 MACs, our accelerator cannot achieve high MAC utilization in small-size convolutional layers and fully connected layers (e.g., when the channel depth is less than 4) as some MACs are used to compute zero elements. Similarly, as shown in the figure in Table 2.9, at 87.5% sparsity, the power efficiency of this design is lower than that of VDBB [87]. This is because the minimum computational unit in our design is 4 MACs, leading to some hardware resources being wasted on computing zero elements at higher sparsity levels. Therefore, our design achieves significant improvements primarily when the sparsity is between 50% and 75%.

Table 2.9: Evaluated accelerator.

Accelerator	Technology (nm)	Freq.	Chip memory	Throughput [†] (TOPS)	Power Efficiency [†] (TOPS/W)	Area Efficiency [†] (TOPS/mm ²)	Weight Sparsity
This work	28	550 MHz	128KB	0.5632 0.5632 0.5632	^s 11.1846/ ^p 7.0325 ^s 17.8278/ ^p 10.8980 ^s 24.1410/ ^p 14.3720	^s 2.3278/ ^p 1.6304 ^s 2.3278/ ^p 1.6304 ^s 2.3278/ ^p 1.6304	50.0%Random ¹ 75.0%Random ¹ 87.5%Random ¹
Liu et al. [87]	16	1 GHz	2.5MB	4	^s 16.8/ ^e 9.3644	^s 2.13/ ^a 0.6955	50.0%VDBB
	16	1 GHz	2.5MB	4	^s 31.3/ ^e 17.4468	^s 4.29/ ^a 1.4008	75.0%VDBB
	16	1 GHz	2.5MB	4	^s 55.7/ ^e 31.0475	^s 8.52/ ^a 2.7820	87.5%VDBB
SMT-SA [92]	16	1 GHz	2.5MB	4	^s 7.4/ ^e 4.1248	^s 1.13/ ^a 0.3690	62.5%Random
Matsubara et al. [93]	12	800 MHz	~41.6MB	66	^c 13.8/ ^e 5.7692	^c 0.2621/ ^a 0.0481	Optimized case (90% sparsity)
EyerissV2 [91]	65	200 MHz	246KB	0.1536	^p 0.9629/ ^p 5.9813	^p 0.5053/2695K gates	sparse AlexNet (~69.6% sparsity)
Kang et al. [94]	65	1 GHz	58KB	0.5	^s 1.65/ ^e 10.2495	^s 1.01/ ^a 5.4429	75%DBB
Sticker [95]	65	200 MHz	170KB	0.1024	^c 2.82/ ^e 6.3857	^c 0.0630/ ^a 0.3395	Pruned AlexNet (96.31% sparsity)



^sSynthesis result; ^pPost-layout result; ^cASIC result; [†]Peak hardware throughput; [†]Calculated with sparsity and network structure
^ePower efficiency scaling = Power efficiency $\times \frac{\text{process}}{28\text{nm}} \times \left(\frac{\text{Voltage}}{0.81\text{V}}\right)^2$; ^aArea efficiency scaling = Area efficiency $\times \left(\frac{\text{process}}{28\text{nm}}\right)^2$

¹Channel-level sparsity; ²VDBB sparsity; ³Random sparsity

*Note that all synthesis and post-layout results of our design are obtained based on 0.81V/125C SSG corner and RC-worst case.

2.5 Design of Booth-value-confined Multi-precision Accelerator

AI applications can easily reach tens of millions of weight data, leading to a huge memory footprint for real-time applications [82]. Arguably, the amount of memory access

and the computing performance are two key factors limiting the performance of various devices, especially edging applications.

From the view of the computation unit, MAC, the basic arithmetic unit, is the most common computing component in a PE array. It puts a spanner in the edge application due to the huge power requirement for computations and high latency in the critical timing path. In this regard, approximate computing provides a solution to enhance performance in terms of latency, power, and area with low accuracy sacrifice on the basis of the approximate characteristics of NNs [16, 17, 18]. Additionally, MP computing can also be conducive to the deployment of NNs in edge devices. In many AI model computation scenarios, high-precision data computations are not always necessary. Using more low-precision computations while maintaining a relatively low loss in accuracy still yields great performance benefits [100, 101].

In addition, a well-organized data flow can effectively improve data reuse and reduce memory traffic, which in turn reduces the power consumption for data movement. Systolic data flow used in TPU aiming to improve the speed and power efficiency of matrix and convolutional operations fills an urgent need for reducing the memory footprint, and its data-reusable feature will be useful for NNs [19]. Although systolic data flow can leverage the memory access and the computing performance of the whole system, it cannot break the memory wall since the access efficiency is directly restricted by the memory framework. However, near-memory computing approach consisting of distributed memory blocks placed nearby PEs supports larger bandwidth compared with Von Neumann architecture.

From the NN training perspective, not much work has been devoted to model training based on approximate MAC, and major efforts have been directed at quantization and pruning for NNs. Since the pioneering work of relevant approximate MACs for NNs, the problem of model training has received a lot of attention. However, in most works, the models are trained based on the normal method with exact multipliers but just verified on the approximate multiplier-based platform, leading to an inconsistency between the training and inference process. Namely, the training method based on approximate MAC, suitable for the migration of specific hardware framework, still faces various flaws.

To address the aforementioned issues, we will further refine our design based on the two accelerators previously developed. The MBW Booth accelerator design in Section 2.3 includes MP computation and Vector systolic data flow. Our design will incorporate its MP computation and integrate the proposed Booth-value-confined approximate multipliers to construct MP approximate multipliers, thereby further enhancing computational efficiency. Additionally, our design will combine NMC with the vector systolic data flow to develop an NMC-friendly data flow, further improving power efficiency. Furthermore, based on this data flow, we will develop an efficient NN mapping algorithm. Finally, to further enhance the computational efficiency of the compute units, our design will leverage the pre-encoding multiplier design principles from Section 2.4

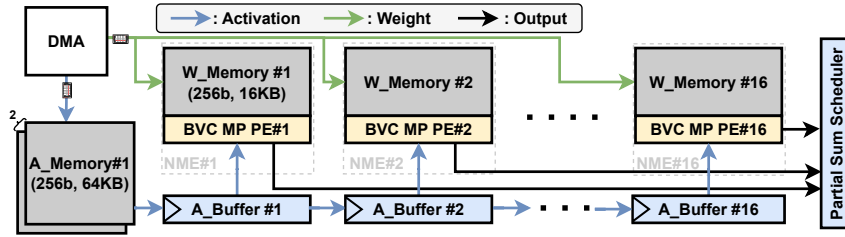


Figure 2.14: Booth-value-confined multi-precision accelerator.

by stripping away the encoder and pre-encoding weights offline, reducing area overhead in PEs and improving power efficiency. To do so, this section introduces a BVC MP accelerator. The main contributions are summarized as follows:

- **Booth-value-confined MP multiplier:** a radix-8 booth (R8B)-based MP multiplier is proposed with lower overhead in comparison to traditional multipliers. Prohibiting the ‘ ± 3 ’ cases in both the model training and inference stages and eliminating the encoder on the basis of the feature of offline weight, could simplify multiplication in NN calculations, substantially optimizing the latency, area, and power in the PE array.
- **Near-memory-computing-friendly data flow:** an accelerator is proposed to enable the data reuse of weights and activations for NN inference from inter-/intra-NME-order views. The weight memory is based on an NMC framework and directly connected to the PE array to reduce the FF consumption.

2.5.1 Implementation of BVC MP Accelerator

As shown in Fig. 2.14, the proposed BVC MP accelerator is employed for convolutional and matrix computations, supporting various layers and kernel sizes. It internally comprises a total of 16 near-memory engines. Within each NME, the computation units are directly connected to the weight memory. The data flow accommodates three types of data streams: 1) pipeline data flow for activations, inter-NME data reuse for activations, 2) near-memory data flow for weights, intra-NME data reuse for weights, and 3) stationary data flow for temporary output. Each NME can process 64 bytes per clock cycle. The basic arithmetic unit is the proposed BVC MP MAC, supporting three different precisions ($4b \times 3b$, $8b \times 6b$, and $8b \times 8b$). Also, weight memories sum up to 256KB while activation memories total 128KB. The memory size is determined based on the size of prevalent AI models and the number of on-chip PEs.

Table 2.10: Non-zero Radix-4 Booth algorithm.

Bit sequence			Booth Value	neg_i	$shift_i$
w_{2i+1}	w_{2i}	w_{2i-1}			
000			+0	0	0
001			+1	0	0
010			+1	0	0
011			+2	0	1
100			-2	1	1
101			-1	1	0
110			-1	1	0
111			-0	1	0

Table 2.11: Non-three Radix-8 Booth algorithm.

Bit sequence				Booth Value	neg_i	$multiple_i$
w_{3i+2}	w_{3i+1}	w_{3i}	w_{3i-1}			
0000				+0	0	0
0001				+1	0	1
0010				+1	0	1
0011				+2	0	2
0100				+2	0	2
0101				+3	0	4
0110				+3	0	2
0111				+4	0	4
1000				-4	1	4
1001				-3	1	2
1010				-3	1	4
1011				-2	1	2
1100				-2	1	2
1101				-1	1	1
1110				-1	1	1
1111				-0	1	0

Booth-value-confined Multiplier

Concerning computation units, radix-4 Booth (R4B) and radix-8 Booth (R8B) are two common algorithms used in exact multipliers. To reduce the complexity of the encoder and decoder in Booth, an intuitive way is to cut down the cases of booth value. In a traditional radix-4 Booth multiplier, the operation includes 5 cases (i.e., 0X, 1X, 2X, -1X, and -2X) as shown in Table 2.10, which means at least 3-bit signal can represent

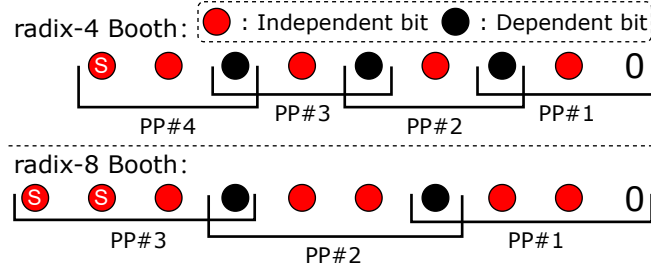


Figure 2.15: Independent and dependent bits in radix-4 and radix-8 Booth.

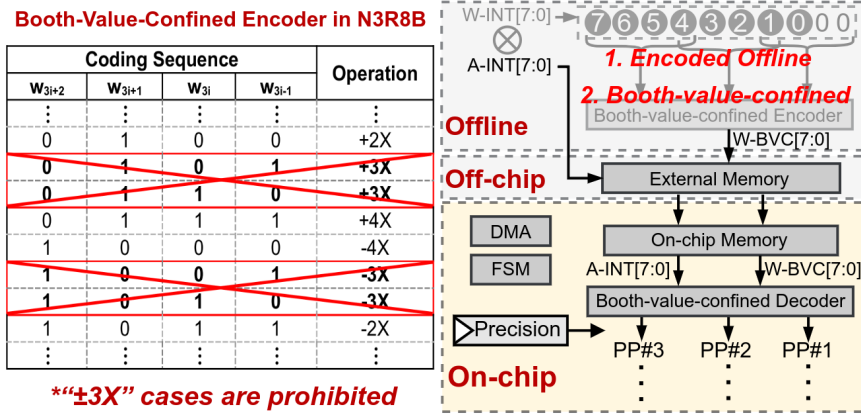


Figure 2.16: Booth-value-confined multiplier (N3R8B).

these 5 cases. To simplify these 5 cases, we propose a non-zero radix-4 Booth (NZR4B) algorithm by replacing the '0X' case with the ' $\pm 1X$ ' case (i.e., amending the '000' and '111' bit sequences into '010' and '101' as depicted in Table 2.10). After the elimination of the '0X' case, only a 2-bit signal is needed to represent the left 4 cases (i.e., $1X$, $2X$, $-1X$, $-2X$), which reduces the complexity of the circuits and improve the latency in MACs. Also, it has no impact on the bandwidth compared with the exact multiplier. In addition, $AxBM$ multiplier [16] can be regarded as a reference for an approximate radix-8 Booth design. Similarly, we propose a non-three radix-8 Booth (N3R8B) algorithm via approximating the ' $\pm 3X$ ' cases into ' $\pm 4X$ ' or ' $\pm 2X$ ' cases to simplify the calculation scenario as depicted in Table 2.11, since the ' $\pm 3X$ ' cases cannot be computed using simple shift operation. Thus, only a 3-bit signal can represent the remaining 7 cases.

In addition, in the 8-bit case, an exact multiplier can express 256 unique numbers. Even though most NN applications are error-tolerance applications, keeping as many unique numbers as possible while reducing the cases of Booth value is also necessary. Generally speaking, the less we amend the encoder cases, the more unique numbers we can get, which is beneficial for NN training. Since the numerical precision (i.e., unique numbers) that the multiplier can represent is higher, the performance of NNs after train-

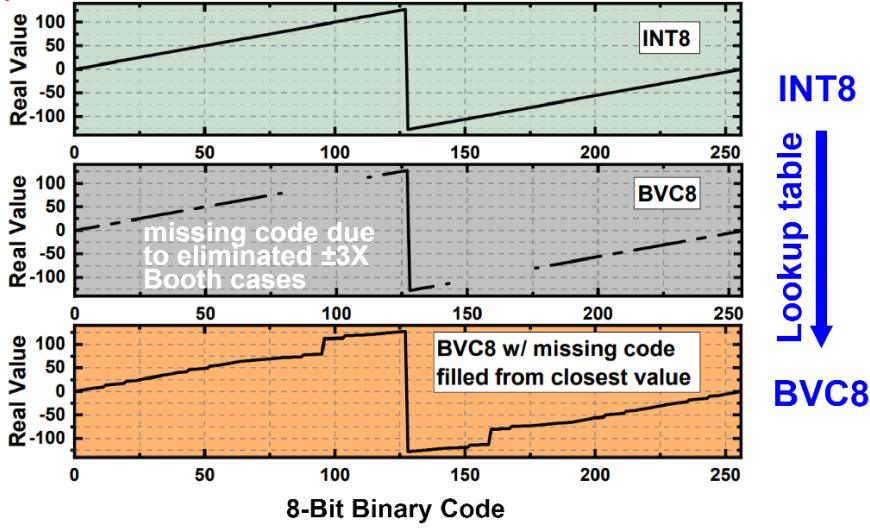


Figure 2.17: Booth-value-confined mapping from INT8 to BVC8.

ing is better. In our design, the eliminated cases are not selected randomly. In contrast, it is necessary to maintain the number of different Booth value cases as much as possible. Simultaneously, as the Booth algorithm splits the multiplicand into several bit sequences as shown in Fig. 2.15, the mutual influence between adjacent bit sequences needs to be avoided. Regarding the aforementioned proposed multipliers, we only modify as few cases in relevant multipliers as possible. Moreover, due to the encoding mechanism of the Booth algorithm, it is very suitable for the simplification of MAC operation via modifying the Booth value. However, for applying to AI training flexibly, we recommend only adjusting the independent bits in Fig. 2.15 since modifying the red dots does not affect the generation of other PPs. Taking 8-bit N3R8B multiplier as an example as shown in Fig. 2.16, compared to the traditional R8B algorithm, we directly prohibit the " $\pm 3X$ " encoded cases to simplify the computation units. Additionally, due to the " $\pm 3X$ " cases being disabled, the encoded data can maintain the same bit width as the original data, ensuring no impact on the bandwidth of the system's data flow. Also, as the weights are known before loading into the chip, we can apply the R8B encoding to the weights offline, further reducing hardware overhead. Furthermore, we can use the exact multiplier to simulate the behavioral function of the proposed approximate multiplier. As shown in Fig. 2.17 (i.e., INT8-to-BVC8 in N3R8B), due to the constraints on Booth values, BVC8 (grey plot) can only represent a subset of values within the 8-bit binary code. Therefore, to ensure all 256 values in 8-bit can be represented, the nearest values are filled in empty positions through a search process. Finally, a lookup table is used to map the INT8 values to the filled BVC8 values, allowing the exact multiplier to simulate the behavior of the proposed N3R8B multiplier. Namely, high-performance platforms such as GPUs can simulate the computation of the proposed multipliers using the integrated exact multipliers, which means consistency can be assured between

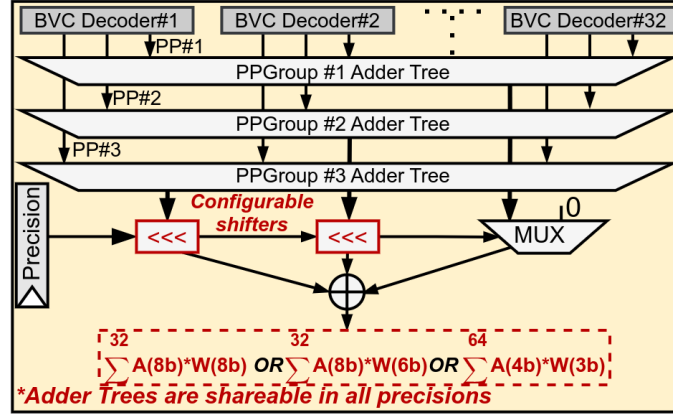


Figure 2.18: Booth-value-confined multi-precision processing element.

GPU platforms and this design. Also, combining the NAS training method mentioned in Section 2.3.1, the BVC-based MP NN can be trained on GPU. Otherwise, if we adopt the AxBM multiplier [16] to AI training, the exact multiplier cannot simulate its function since some cases are involved with the modification of dependent bits in Fig.2.15, which also affects the generation of near PPs. Therefore, the main modification should be on independent bits rather than dependent bits. This can maximize the number of unique numbers. As mentioned above, in the R4B and R8B algorithms, we only modify the corresponding bits in the bit sequence, which does not affect the results of other bit sequences.

Furthermore, since the weights are known in advance and the edging devices only run inference procedures, we could directly encode the weights offline so that the encoder circuits in our proposed multiplier can be cut down. Since only simplified signals (i.e., 1-bit *shift* and 1-bit *neg* in N3R4B, 2-bit *multiple* and 1-bit *neg* in N3R8B) need to be decoded, the decode circuit is relatively simpler in comparison with that of traditional Booth algorithm.

N3R8B-based BVC Multi-precision Processing Element

As the R8B algorithm generates three PPs in 8-bit multiplication, configurable shifters are built to achieve different combinations of partial product accumulations, enabling three data precisions (BVC3/BVC6/BVC8) as shown in Fig. 2.18. Lastly, to enhance computational parallelism, we construct 32 BVC MP MACs within each PE for parallel computation. Additionally, the adder tree can be shared among three different precisions. The overall reductions in power and area reach 82% and 70% compared with the proposed MP booth-based PE as mentioned in Section 2.3.1. Remarkably, $A(\text{INT4}) \times W(\text{BVC3})$ multiplication only generates one PP, resulting in a doubling of throughput compared to other cases. Besides, the unique number of three different precisions in the BVC multiplier (i.e. 8b, 6b, and 3b) is 144, 36, and 7, respectively.

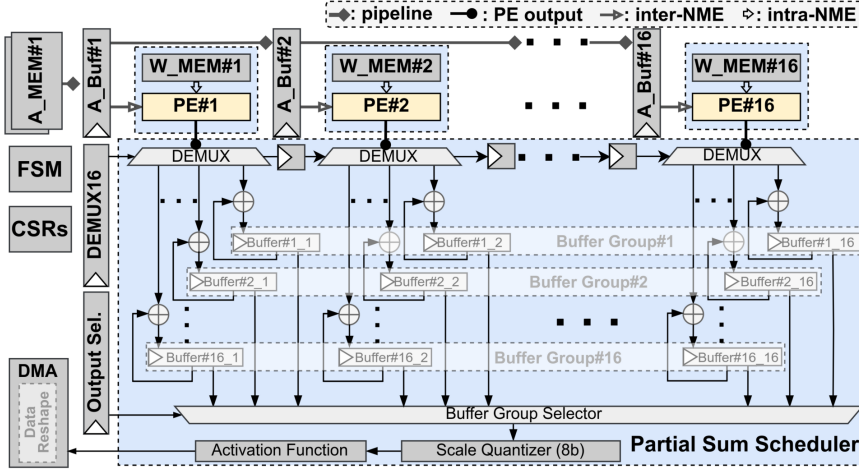


Figure 2.19: Partial sum scheduling for near-memory engines.

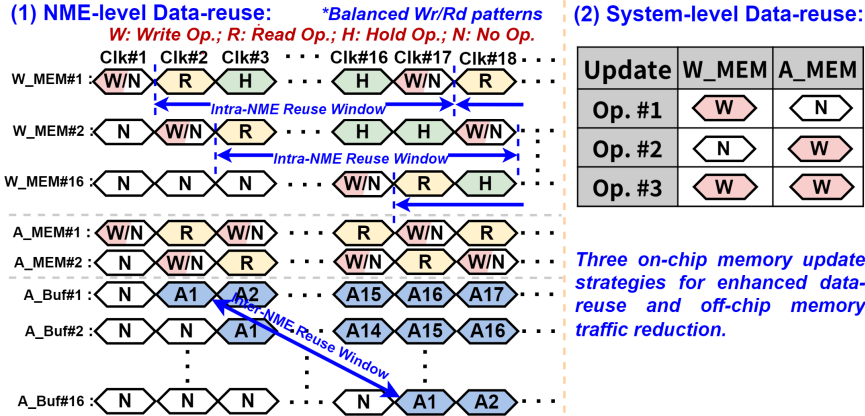


Figure 2.20: Timing diagram in intra-/inter-NME paths for data-reuse.

Data Flow for Near-memory Computing

As the outputs cannot be calculated at once in most large-size NNs, accumulating the temporary outputs to generate the final outputs cannot be avoided. Thus, the Partial Sum Scheduler (PSS) is utilized for the scheduling of the temporal NME outputs as shown in Fig. 2.19. Due to the reuse of each activation and weight up to 16 times, the output results of each NME need to be scheduled to 16 buffers through a demultiplexer (DEMUX). In this design, the 16 data points within each buffer group represent data from the same position in different output channels pertaining to inter-NME activation reuse, while data points from the same position in different buffer groups represent different positions in the same output channel pertaining to intra-NME weight reuse.

To explicitly explain the data scheduling of activations and weights, the PSS timing diagram is shown in Fig. 2.20. Each weight memory updates data every 1-to-16 cycles,

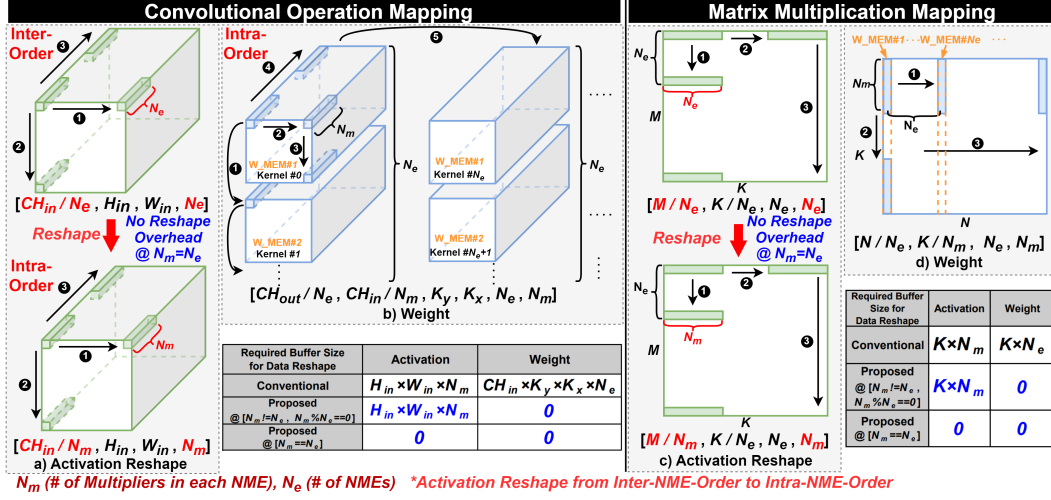


Figure 2.21: Memory space contiguity-aware NN mapping with inter-/intra-NME-order (a) Convolutional operation. (b) Matrix multiplication.

with all NMEs alternating data updates. This means that each weight can be reused 1-to-16 times depending on the configuration from the intra-NME view. The number of times that data is reused also determines the quantity of PEs utilized in the current computation round. Furthermore, since the pipeline stage is 16, activation data can be reused 1-to-16 times from the inter-NME view. Finally, as large-size NNs need to be split into multiple chunks for sequential computations, the weight or activation data used in the current computation round might be called upon again by the next round. Therefore, three options are offered for updating the data, including 1) updating weights, 2) updating activations, and 3) updating both weights and activations. Besides, to address memory access competition and enable efficient data reuse, all memories in the system employ round-robin queue-style data updates. Furthermore, since the activation memories operate in a pipeline-based manner, we exploit two activation memories that alternate in supplying data to the pipeline, eliminating read-write competition issues. The weight memories in the near-memory data flow inherently avoid read-write competition. Therefore, the read and write operations between on-chip and off-chip memories can reach a balanced bandwidth in the design.

Memory Space Contiguity-aware NN Mapping

Moreover, efficient on-chip NN mapping based on intra-/inter-NME-order architecture is crucial to realize a high data-reuse rate. The proposed memory space contiguity-aware (MSCA) NN mapping strategy is shown in Fig. 2.21. The primary idea is to ensure that the input model's data format matches the output model's data format as closely as possible, specifically N_e (the number of NMEs) equals N_m (the number of MACs in each NME). This eliminates the need for additional hardware overhead

Table 2.12: Comparison of exact and approximate booth multipliers.

Design (INT8)	Unique number	Area (μm^2)	Power (μW)	Delay (ns)
Exact radix-4	256	168.588	28.4	1.35
Exact radix-8	256	178.290	28.4	1.75
AxBM1	64	146.034	27.8	1.66
AxBM2	64	146.160	28.5	1.67
NZR4B	81	158.760	29.6	1.37
N3R8B	144	145.782	26.6	1.52
Exact radix-4*	256	169.596	30.7	1.39
Exact radix-8*	256	182.700	28.1	1.67
AxBM1*	NA	145.782	27.6	1.48
AxBM2*	NA	145.782	27.6	1.48
NZR4B*	81	154.098	28.9	1.30
N3R8B*	144	137.970	24.3	1.28
*: without encoder				

required for data alignment. However, due to limitations in area and power budgets, fulfilling the above condition might not always be feasible. In such cases, it is preferable to ensure that Nm is divisible by Ne , thereby eliminating the required buffer and logic consumption for weight data reshaping, as indicated by the blue text in the tables. Also, to align the activations between two adjacent layers for seamless computation, activation reshaping from Ne -based inter-NME-order to Nm -based intra-NME-order is still necessary as shown in Fig. 2.21. Furthermore, as illustrated in the tables, the proposed intra-/inter-NME-Order mapping method can reduce the on-chip buffer for data alignment. Especially, in Ne equal to Nm case, there is no extra buffer consumption for activation data alignment.

2.5.2 Evaluation of BVC MP Accelerator

Multiplier Evaluation

Four metrics as shown in Table 2.12 are provided for a comprehensive analysis of the proposed multipliers: 1) area consumption, 2) power consumption, 3) latency, and 4) unique number. Since the proposed works demonstrate their performance with different precision datatypes (e.g., INT8, INT16, INT32), this part aims to replicate these designs and normalize them into INT8 type for a comprehensive comparison. All the multipliers are implemented in Verilog, synthesized using Synopsys Design Compiler and the TSMC 22nm standard cell library, and simulated with VCS. The relevant metrics are reported using the Design Compiler. The proposed N3R8B multiplier demonstrates better

Table 2.13: Comparison between MP-Booth PE and BVC MP PE.

	MP-Booth PE	BVC MP PE
Area [μm^2]	20345	6129
Power [mW]	6.28	1.14

performance compared with other multipliers. Above all, the proposed N3R8B-based BVC multiplier without encoder, depicted in Fig. 2.16, is selected as a baseline for the PE design to achieve energy-efficient computation.

Processing Element Evaluation

To analyze the PE performance, a comparison between the proposed BVC MP PE mentioned in Section 2.5.1 and the proposed MP-Booth PE mentioned in Section 2.3.1 is conducted. Supposing each PE contains 32 8b MACs, the power and area consumption evaluated in 22nm CMOS is shown in Table 2.13. As the BVC MP PE supports the implementation of the accumulation operator using shareable adder trees, the accumulation logic is simpler compared with MP-Booth PE. Also, the simplified decoder logic of BVC MP is shareable. Thus, the BVC MP PE reduces power and area by 82% and 70% respectively in comparison to MP-Booth PE.

Data Flow Evaluation

The data flow latency and throughput are similar to the detailed analysis mentioned in Section 2.3.2. Furthermore, the NMC-friendly data-flow structure can be extended to the deployment of 2.5D/3D stack technology. Also, the NMC-friendly data flow for NME with inter-/intra-NME data reuse optimizes memory bandwidth utilization and minimizes off-chip memory traffic by 170X to 206X for various NN models as mentioned in Section 3.4.3.

2.6 Discussion

The three proposed accelerators adopt different multipliers, data flows and data-reuse strategies for energy-efficient computing at the edge. As shown in Table 2.14, the MBW Booth accelerator and BVC MP accelerator are both based on vector systolic data flow, belonging to fine-grained data flow. Meanwhile, the BVC MP accelerator incorporates an NMC-friendly data flow structure. In contrast, the sparse NN accelerator adopts a broadcast-based data flow, without a pipeline-like structure, belonging to coarse-grained data flow. Furthermore, the sparse NN accelerator also has support for structural sparsity, but structured sparse NNs have higher algorithmic requirements during training

Table 2.14: Comparison of the proposed AI accelerators.

	MBW Booth Accel. (Section 2.3)	Sparse NN Accel. (Section 2.4)	BVC MP Accel. (Section 2.5)
Platform	FPGA	28nm CMOS	22nm CMOS
Sparsity	No	Yes	No
Data flow	Vector Systolic	Broadcast	(1) NMC-friendly (2) Vector Systolic
Multiplier Feature	(1) Radix-4 Booth-based (2) Multi-precision	(1) Radix-4 Booth-based (2) Pre-encoding	(1) Radix-8 Booth-based (2) Approximate BVC (3) Multi-precision (4) Pre-encoded
Precision	(1) INT8 * INT8 (2) INT4 * INT4 (3) INT2 * INT2	(1) INT8 * INT8 (2) UINT8 * INT8	(1) INT8 * BVC8 (2) INT8 * BVC6 (3) INT4 * BVC3
Data-reuse Feature	Stationary-reuse for weight Pipeline-reuse for activation	Block-order reuse for weight Engine-order reuse for activation	Intra-NME-reuse for weight Inter-NME-reuse for activation

and are not suitable for deployment. Additionally, compared to the MBW Booth accelerator, the BVC MP accelerator also supports approximate computing under multiple precisions, further reducing computational overhead. Finally, the data flow of all three accelerators supports the reuse of weights and activations, and the efficiency of reuse is essentially the same. All three designs are suitable for edge AI applications. The MBW Booth accelerator is particularly well-suited for multitasking AI applications, optimizing power efficiency by matching different precision levels to varying task complexities. Similarly, the BVC MP accelerator is applicable, especially in scenarios with higher efficiency requirements, given its NMC and approximate computing capabilities. Lastly, for scenarios with bandwidth limitations, the sparse NN accelerator is preferable due to its architecture effectively reducing bandwidth overhead by leveraging the sparsity of NNs and its own computational units.

2.7 Conclusion

This chapter delves into the in-depth exploration of research on multi-precision computing, approximate computing, structural sparsity, and efficient NN mapping for edge AI devices. It introduces three distinct AI accelerator architectures: 1) a vector systolic array accelerator based on the radix-4 Booth algorithm for MP computing, 2) a sparse-supporting accelerator with the pre-encoding multiplier, and 3) an accelerator with NMC-friendly data flow under MP approximate computing based on radix-8 Booth algorithm. It is noteworthy that both the first and the last accelerators support hardware-aware NN mapping, maximizing the utilization of computing units. Moreover, due to the benefits derived from MP and approximate computing, these two accelerators achieve high power efficiency while maintaining a very low accuracy loss in NNs. Additionally, all three accelerators are capable of efficient on-chip data reuse within their respective data flows, thereby reducing external memory access.

Chapter 3

Design of System-on-Chip for Edge AI Applications

On the basis of the proposed accelerators discussed in Chapter 2, this chapter seeks SoC-level designs to achieve power efficiency for general-purpose platforms. First, a single-core SoC design is proposed, which integrates the MBW Booth accelerator as mentioned in Section 2.3. An open-source 2-stage RISC-V core is the main controller in this SoC to schedule the entire system. This SoC is implemented on a flash-based MPF300T FPGA platform. Then, a single-core SoC design is proposed, which incorporates the BVC MP accelerators as discussed in Section 2.5. This SoC, including a similar microcontroller, is realized on TSMC 22nm technology node. Finally, a multi-core multi-mode SoC is proposed for versatile safety-critical AI applications to explore high-efficiency computation for general-purpose computation. Besides, the performance evaluation of these three SoCs is provided.

3.1 Introduction

The evolution of AI SoC is intricately linked to the continuous advancements in computer architecture and embedded systems. The research begins with a focus on enhancing the performance of general-purpose computing units, notably CPUs and GPUs. However, the landscape undergoes a transformation with the rise of deep learning, necessitating dedicated NN acceleration like Google's TPU and NPU. Key technological pillars in AI SoCs encompass multi-core processors, dedicated NN accelerators, and the adoption of heterogeneous computing. Multi-core processors leverage parallel processing, while dedicated accelerators optimize hardware for efficient NN computation. Heterogeneous computing involves the integration of diverse processing units on a single chip, enhancing both performance and flexibility.

On the software front, deep learning frameworks such as TensorFlow and PyTorch play pivotal roles. These frameworks provide high-level Application Programming In-

interfaces (APIs) that simplify the deployment and optimization of complex NN models. Compiler technology tailored to specific hardware architectures further optimizes the translation of high-level models into low-level hardware instructions, contributing to enhanced execution efficiency.

AI SoCs have found applications across diverse domains. In edge computing, these chips empower devices like smart cameras for real-time, local decision-making, reducing reliance on cloud processing. Autonomous driving leverages AI SoCs for processing perception and decision-making modules, contributing to intelligent driving capabilities. In IoT end-nodes, embedded AI SoCs enable intelligent sensing and decision-making. Despite advancements, AI SoC development faces challenges. Power efficiency optimization is crucial, requiring a balance between computational performance and power consumption. Balancing generality and customization while maintaining flexibility is an ongoing research area.

In this chapter, three distinct AI SoCs are developed targeting edge AI applications. The first SoC realized on a flash-based FPGA platform, incorporates the MBW accelerator mentioned in Section 2.3 and an open-source 2-stage RISC-V core. Also, to improve the performance, the second SoC, realized on 22nm CMOS technology node, incorporates the BVC MP accelerator mentioned in Section 2.5 and a similar RISC-V core. Finally, targeting safety-critical complex AI applications, a fault-tolerant SoC integrating 16 RISC-V cores and one AI accelerator is proposed.

The rest of this chapter is organized as follows. Section 3.2 reviews the related work about SoC in edge AI. Section 3.3 introduces the first proposed SoC implemented in a flash-based FPGA platform. Section 3.4 describes the proposed SoC with a BVC MP accelerator, realized in 22nm technology node, and also presents performance. Section 3.5 illustrates the proposed 22nm multi-core multi-mode SoC. Finally, Section 3.6 discusses the proposed AI SoCs and Section 3.7 concludes this chapter.

3.2 Related Work

The landscape of AI SoC development has witnessed significant contributions and innovations from researchers worldwide. Notable studies have explored various aspects, starting with early endeavors to optimize computing units for neural network processing.

In 2023, Z. He et al. [32] designed an SoC with an open-source RISC-V core and an MP AI accelerator supporting INT2/4/8 computation. An agile hardware and software co-design is developed to ramp up the deployment of AI applications at the edge. However, this architecture can only accelerate the convolutional operation and matrix multiplication. Other operators need to be calculated in the RISC-V core, which leads to inefficient computation and high latency. P. D. Schiavone et al. [33] proposed a traditional 22nm SoC on the IoT side. This SoC contains a 32-bit open-source RISC-V core. The system achieves the best power density of 6 uW/MHz, resulting in an power

efficiency of 433 MOPS/mW. Nevertheless, this SoC cannot accelerate the computation of NNs at the edge since it does not integrate parallel computing units.

Furthermore, A. Pullini et al. [34] designed a 40nm SoC featuring a hierarchical architecture with a small (12K gates) RISC-V core. The proposed SoC features an 8-core compute cluster achieving a peak performance of 850 MMACS and 1 GFLOPS and a power efficiency up to 15 MMACS/mW in fixed-point and 9 MMACS/mW in floating-point. However, this GPU-like SoC design cannot effectively accelerate the convolutional operation and matrix multiplication due to the latency in multi-core communication. In contrast, E. Flamand et al. proposed a fully programmable RISC-V IoT-edge computing engine [35], featuring an 8-core cluster with CNN accelerator, coupled with an ultra-low power MCU with 30 μ W state-retentive sleep power. The incorporated CNN accelerator can be adopted for the acceleration of NNs. However, the end-to-end latency between multi-core and accelerator is essentially high and leads to low hardware utilization. Also, F. Conti et al. presented Marsellus, an all-digital AI-IoT end-node heterogeneous SoC fabricated in GF 22nm FDX [28]. This design provides 12.4 TOPS/W peak power efficiency and 6.38 TOPS/W best power efficiency at ResNet20. This design integrates adaptive body biasing to enable on-the-fly adaptation of transistor threshold voltages for safety-critical applications.

Despite these advancements, challenges persist in accurately modeling aspects of inference-stage accelerators, such as flexibility for different applications, memory models for weights and features, the scale and computation efficiency of PE arrays, and optimal NN mapping strategies for large NNs. Analysis of these aspects is essential for understanding and optimizing the design of AI platforms. Moving forward, the field of AI SoC continues to evolve, with ongoing research exploring efficient hardware architectures, algorithm optimizations, generalization, and software enhancements. This collaborative effort aims to address challenges and pave the way for more efficient and powerful AI SoCs tailored to a variety of applications.

3.3 FPGA: Single-core SoC with MBW Accelerator

To comprehensively evaluate the MBW Booth accelerator mentioned in Section 2.3 from an SoC level, we integrate a single RISC-V core as a controller and an MBW Booth accelerator into an SoC system. Also, a programmable microcontroller can schedule the peripherals and AI accelerator with high flexibility.

3.3.1 SoC Architecture

Fig. 3.1 shows the architecture of the proposed SoC. This SoC mainly consists of three components: 1) MBW accelerator, 2) HBirdv2 e203 RISC-V processor [102], and 3) DDR4 DRAM. HBirdv2 e203 is an open-source lightweight 32-bit RISC-V micropro-

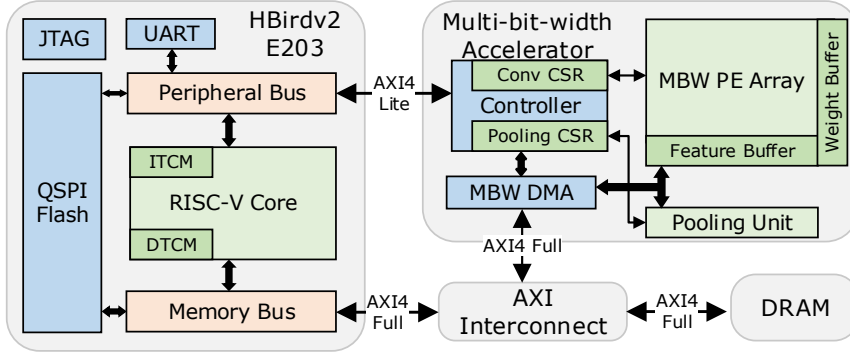


Figure 3.1: Block diagram of SoC.

Table 3.1: Key CSRs of MBW accelerator.

CSR	Offset Address	Description
STCONV	0x0000	Start conv operation
CBWIN	0x0008	Bit-width of input data
CFADDR	0x0050	Activation base address
CFSHP	0x0010	Shape of activation
CCONF	0x0024	Padding, Stride and Kernel
CWADDR	0x0060	Weight base address
COADDR	0x0088	Output data base address
CDONE	0x0094	If conv operation done
STPOOL	0x0200	Start pooling operation
PCONF	0x024C	Pad, Stride and Kernel
PIADDR	0x020C	Input data address
POADDR	0x0218	Output data address
PIOSHP	0x0228	Shape of activation
PBWIN	0x02AC	Bit-width of input data
PDONE	0x0204	If pool operation done

*Top half for convolution and bottom half for pooling.

cessor with 2-stage pipeline, supporting RV32IMAC or RV32EMAC instruction set architecture (ISA). E203 has a wealth of peripherals like UART, QSPI-flash, JTAG, etc.

To provide fast and efficient access to instructions and data, Instruction Tightly Coupled Memory (ITCM) and Data Tightly Coupled Memory (DTCM) are equipped in e203. Prior to the SoC design, prevalent open-source microcontrollers including Pulpino, HBirdv2 e203, Tiny RISC-V, rocket, BOOM, etc., are compared. To make the SoC applicable for resource-constrained platforms, HBirdv2 e203 is selected as a pertinent controller with low overhead and abundant peripherals.

Table 3.2: Resource utilization of our SoC on Microsemi MPF300T Eval Kit.

Component	Fabric 4LUT	Fabric DFF	Interface 4LUT	Interface DFF	Math (18x18)	uSRAM	LSRAM
Accelerator (100 MHz)	65035	48033	13776	13776	140	104	208
RISCV E203 (16 MHz)	18591	9414	2448	2448	0	12	64
PF_DDR4 (400 MHz)	18404	15219	1272	1272	0	43	21
AXI4 Interconnect	3458	3138	636	636	0	53	0
Others	61	36	0	0	0	0	0
Total	105549	75840	18132	18132	140	212	293

* ITCM Size: 64KB LSRAM; DTCM Size: 64KB LSRAM; Accelerator Buffer Size: 512KB LSRAM; CSR Size: 79*32Bits uSRAM

Two groups of AXI ports are extended on e203 to control the MBW accelerator and access the DRAM. Similarly, two groups of AXI ports are constructed in the accelerator. These two ports in the MBW accelerator are interconnected to the peripheral and memory bus in e203. In other words, to integrate this accelerator into our SoC, an AXI4-Lite bus is developed to control the accelerator, and an AXI4-Full bus is developed to exchange data with external memory (DRAM). E203 can directly configure the CSRs in the accelerator via the AXI-Lite bus. The MBW DMA can exchange data with DRAM automatically according to CSR configuration. In short, this accelerator connected to e203 is regarded as a peripheral with a DMA channel in the proposed SoC. The proposed accelerator and DRAM are extended from the unused space of e203 for user customization. In addition, 32-bit registers are allocated for both convolutional and pooling operations. The detail of key CSRs is shown in Table 3.1.

3.3.2 Performance Evaluation

In this SoC, the clock tree includes three domains. The accelerator, DDR4, and e203 run at the speed of 100MHz, 400MHz and 16MHz respectively. An AXI-interconnect IP is used to bridge these components for clock domain cross (CDC) communications. Table 3.2 shows the resource utilization. As e203 is a lightweight microprocessor, we only develop a bare-metal program on e203 to control the whole SoC. The accelerator with 16x8 PE array occupies 63.72% LUT, 65.77% FF consumption in the SoC. Besides, the accelerator takes over 70% larger SRAM (LSRAM) consumption. As the number of INT8/4/2 computations in one MP MAC is 1, 2, and 4, respectively, the peak throughput of INT8/4/2 computations in this SoC is 128 MACs/clock, 256 MACs/clock, and 1024 MACs/clock respectively. Notably, 2 operations are equal to 1 MAC.

So far, the SoC with the MBW accelerator has been constructed successfully. The SoC is simulated, synthesized, and implemented using Libero on Microsemi MPF300T Eval Kit board as shown in Fig. 3.2, which has 300K LUTs, 300K FFs, 924 Math blocks, 2772 uSRAMs and 952 LRAMs. Then, three same FPGA boards are adopted with the same Libero project but different weights and configuration parameters for the following experiments discussed in Section 4.4.

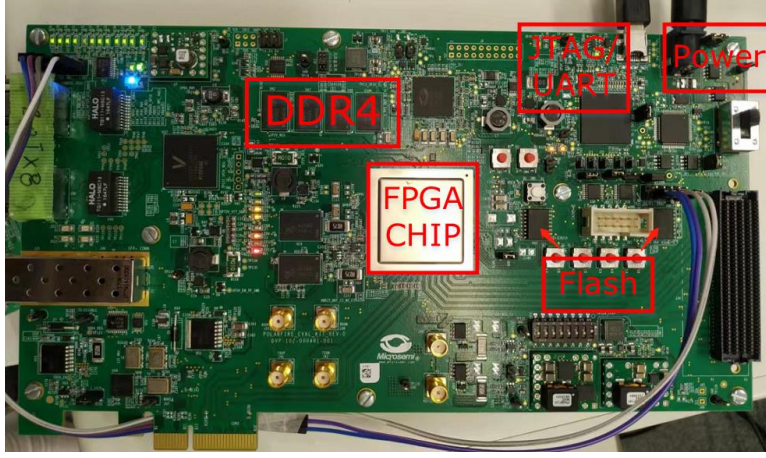


Figure 3.2: SoC implementation on MPF300T Eval Kit.

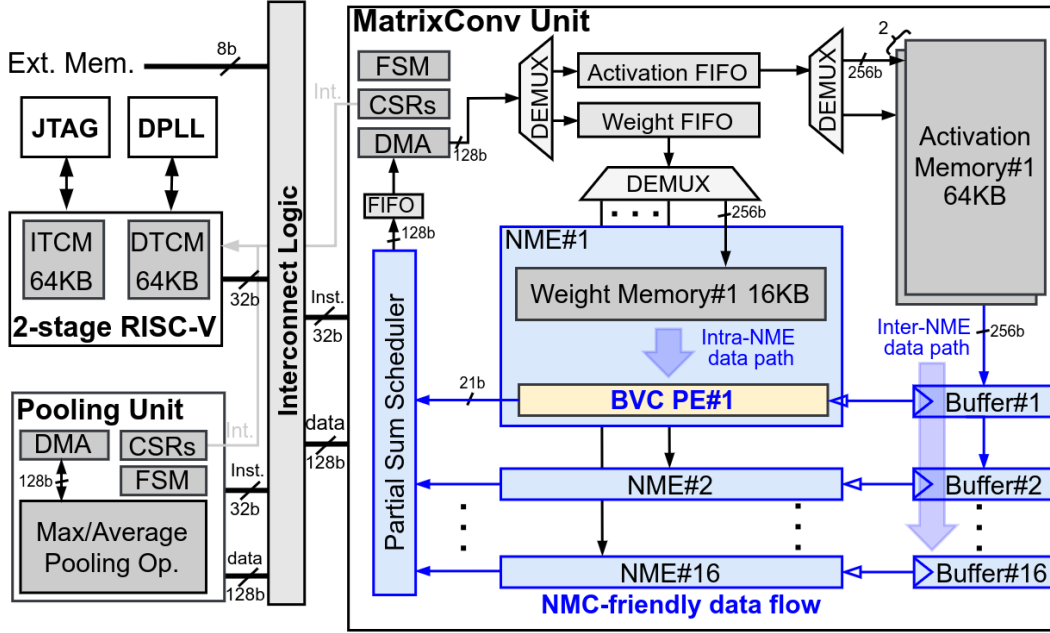


Figure 3.3: System architecture.

3.4 ASIC: Single-core SoC with BVC MP Accelerator

3.4.1 SoC Architecture

To optimize the power efficiency in edge devices and execute real edge AI applications, an edge AI SoC is proposed in this work, featuring: 1) power-/area-efficient PE with BVC multipliers and a shared adder tree, supporting BVC-based NN training with no accuracy loss, 2) NMC-friendly data flow supporting regular and balanced read/write

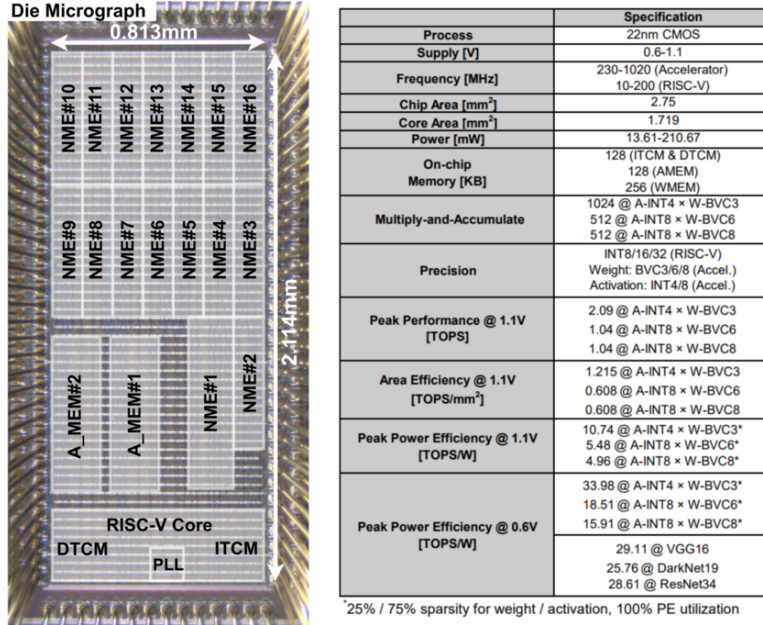


Figure 3.4: Die micrograph and metrics.

operation and optimized data-reuse, and 3) 2-stage RV32IMAC RISC-V as the main controller.

Fig. 3.3 presents the architecture of the proposed edge AI SoC, consisting of a 2-stage RV32IMAC RISC-V sub-system with rich peripherals as the main controller mentioned in Section 3.3.1, a digital PLL, a programmable pooling unit supporting both max and average pooling, and a programmable MatrixConv unit (i.e., BVC MP accelerator mentioned in Section 2.5) with 16 near-memory engines for accelerating the convolutional and matrix computations. Each NME contains a BVC PE with a 16KB weight memory directly connected for NMC. The BVC PE supports 3 customized precision, i.e. BVC8/6/3. The NMC-friendly data flow accommodates two key data streams: 1) inter-NME data flow for activation reuse, and 2) intra-NME data flow for weight reuse.

3.4.2 Fabricated Edge AI SoC

Fig. 3.4 presents the fabricated chip in TSMC 22nm process with 2.75 mm² area. This work exhaustively explores system-level edge AI chip implementation on approximate mixed-precision computing, NN mapping, data scheduling, and data reuse. It implements 128KB activation SRAM, 256KB weight SRAM, 64KB ITCM, and 64KB DTCM. The 16 NMEs, together with system-level mapping/scheduling, can execute convolutional computation and matrix multiplication to compose the on-chip execution flow for a complete NN model. Several exceptional special operators, such as batch normalization, will be executed on the RISC-V core.

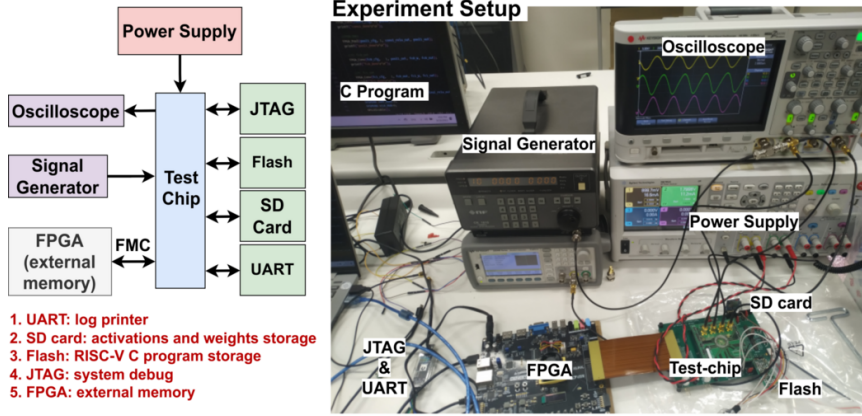


Figure 3.5: Test platform of the proposed chip.

3.4.3 Performance Evaluation

The test platform for this chip is depicted in Fig. 3.5. The UART module is used for printing log information to monitor the system operation in real-time. The SD card is used to store the model’s weight and activation data. The QSPI-Flash is used to store the C compiled program for the RISC core. The JTAG module is used for debugging the entire system and is controlled through the open-source On-Chip Debugger, OpenOCD [103]. The chip PCB is connected to the FPGA board via an FMC cable, with the FPGA serving as external memory to store model data. Additionally, the chip on the PCB is powered by a DC power meter, providing 1.8V standard IO power and 0.6-1.1V core power. The entire testing process operates as follows: firstly, the RISC-V core boots from flash, then the RISC-V core completes the decomposition of the model as well as the initialization and allocation of off-chip and on-chip memory. After that, the RISC-V imports model data from the SD card into the external memory simulated by the FPGA. Finally, the RISC-V configures the CSRs of the accelerator, and the accelerator completes the output of the final results after several rounds of computation and interaction with off-chip and on-chip memory data.

The power/area breakdown is shown in Fig. 3.6 (a) and (b). The matrixconv unit occupies 81.31% of the area cost and 91.15% of the power consumption cost. This is due to the 384KB of SRAM located within this unit, along with its computation PE array of 32×16 size. Therefore, this module essentially accounts for the majority of the area and power consumption costs of the entire chip. Additionally, the RISC-V contains 128KB of ITCM and DTCM, leading to the RISC-V core also occupying 12.59% of the area cost. Furthermore, the voltage scaling of this chip is shown in Fig. 3.6 (c) and (d). This chip can work at the supply core voltage of 0.6–1.1 V with the frequency of 230–1020 MHz of the accelerator and 10-200MHz of the RISC-V core. The peak power efficiency of BVC3, BVC6 and BVC8 are 33.98, 18.51 and 15.91 TOPS/W respectively at 75/25% activation/weight sparsity and 0.6V operating voltage.

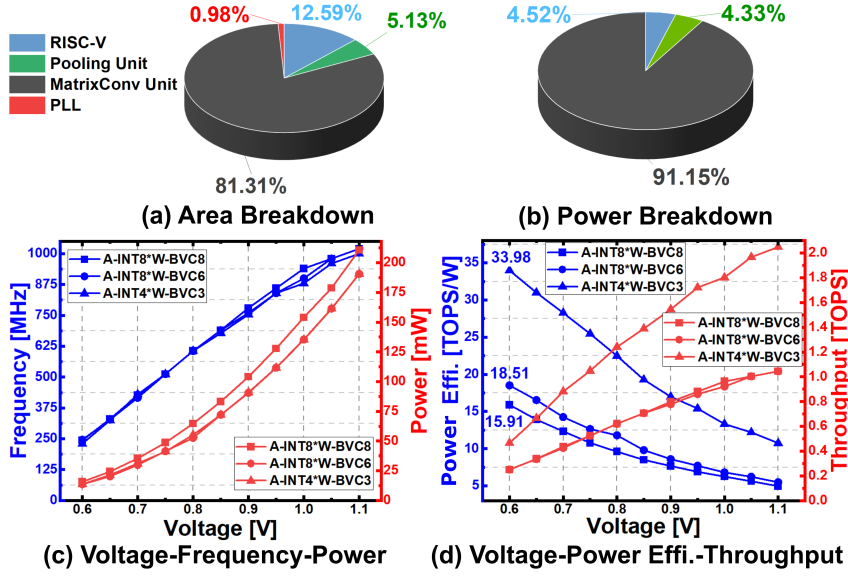


Figure 3.6: Area/power breakdown and voltage scaling.

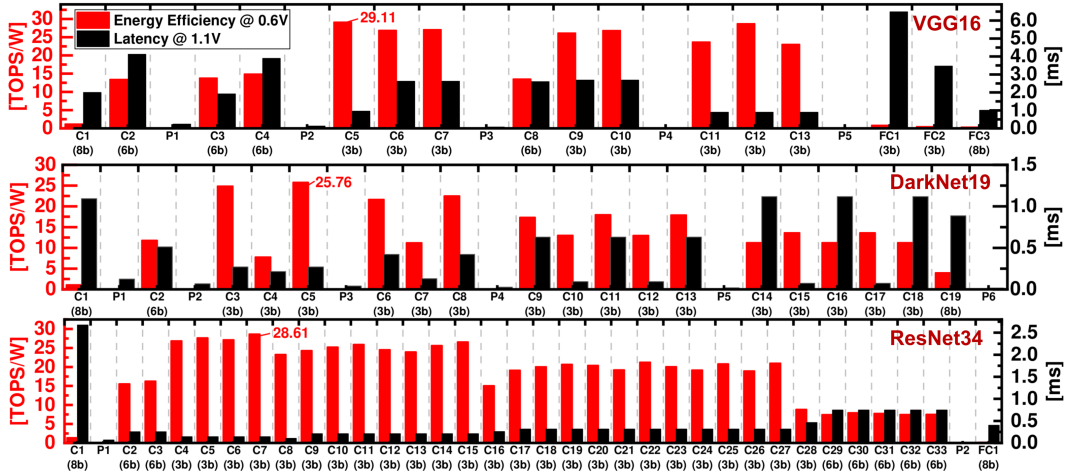


Figure 3.7: Layer-wise evaluation of different NN models.

Also, the peak throughput is 2.09, 1.04, and 1.04 TOPS at 1.1V 1020MHz, respectively.

The evaluation of three different NN modes from a layer-wise level (i.e., VGG16, DarkNet19, ResNet34) is shown in Fig. 3.7. The highest power efficiencies of the three CNN models are 29.11, 25.76, and 28.61 TOPS/W, respectively. Additionally, their computational precision is A-INT4×W-BVC3. For the vast majority of the time, hardware resources are fully utilized in both intra-NME-order and inter-NME-order, achieving maximum computational power efficiency. However, occasionally, as the lower number of input channels in the first layer leads to insufficient utilization of intra-NME-order, power efficiency is slightly lower. Additionally, the inability to reuse weights in

Table 3.3: Evaluation of different NN models.

Model	VGG16	DarkNet19	ResNet34
Dateset	ImageNet-1K	ImageNet-1K	ImageNet-1K
Accuracy [%]	71.17	69.9	70.58
Weight Sparsity	16.11%	20.95%	19.98%
Activation Sparsity	69.31%	59.89%	58.11%
Latency @1.1V [ms]	40.19	10.07	13.70
Average Power Eff. @0.6V [TOPS/W]	15.11	12.49	12.04
Peak Power Eff. @0.6V [TOPS/W]	29.11	25.76	28.61
Latency Reduction ¹	31.40	16.60	21.20
External Memory Access Reduction ²	206x	170x	188x

¹ Compared to an identical design w/o MSCA NN mapping strategy² Compared to an identical design w/o system-level data-reuse strategy

fully connected layers also increases memory footprint and prolongs latency.

The overall evaluation of the models is shown in Table 3.3. Due to the accuracy loss caused by MP computation, the accuracy of the three models is reduced by 1-2% compared to the golden reference. As discussed above, since the models cannot achieve efficient data reuse in every layer, the average power efficiency of the three models (12.04 to 15.11 TOPS/W) is 1.70x to 2.42x lower than their highest efficiency (25.76 to 29.11 TOPS/W). Additionally, since VGG16 has the most parameters and computational requirements among the three models, it has the highest latency. While DarkNet19 and ResNet34 have a comparable number of model parameters, ResNet34 has more layers, meaning that the chip scheduling requires more configuration rounds, thus making the latency of ResNet34 slightly higher than that of DarkNet19.

Table 3.4 summarizes the measurements of the proposed processor and compares it with prior arts [104, 105, 106, 15, 27, 28]. The measured power is 13.61-to-210.67mW with 230-to-1020MHz frequency under a voltage of 0.6V-to-1.1V. The peak power efficiency is 33.98TOPS/W with A-INT4×W-BVC3 at 0.6V and 230MHz. During NN evaluations for ResNet34, DarkNet19, and VGG16, our design achieves 12.04-15.11 TOPS/W with mixed precision computation for preserving accuracy. Compared with [15, 27, 28] in real CNN evaluations, our design improves the power efficiency by 1x to 3.5x and area efficiency by 4.6x to 35x at similar process nodes (except for [15] which is in 4nm process), thanks to the meticulously organized data flow that seamlessly aligned with our hardware architecture to facilitate high data reuse.

Table 3.4: Performance comparison with state-of-the-art chips.

Process	JSSC22[104] 28nm	ISSCC22[105] 28nm	JSSC23[106] 28nm	JSSC23[15] 4nm	ISSCC23[27] 28nm	ISSCC23[28] 22nm FDX	This work ¹ 22nm
Architecture	Digital Accel.	Digital CIM Accel.	Digital CIM Accel.	2xNPU core	Digital Accel.	1xRV32IMCFXpulp+ 16xRV32IMCFXpupnn+ RBE Accel.	1xRV32IMAC+ Digital Accel.
NN mapping	NA	NA	NA	NA	YES ⁵	YES ⁶	YES ⁷
Supply Voltage [V]	0.6-1.0	0.6-1.0	0.6-1.0	0.55-1.0	0.66-1.3	0.5-0.8	0.6-1.1
Frequency [MHz]	44-195	80-240	80-240	332-1196	100-500	420	230-1020@Accelerator 10-200@RISC-V core
Area [mm ²]	0.71	6.83	6.83	4.74	7.81	18.7	Core Area: 1.719 Chip Area: 2.750
Power [mW]	7.8-74	27.04-118.21	27.04-118.21	381-5133	17-174	12.8-123	13.61-210.67
On-chip Memory [KB]	144	192	192	2048	1120	1152	128 (ITCM & DTCM) 128 (AMEM) 256 (WMEM)
Precision	INT2/4/8	INT8/16	Approx. INT12	INT4/8/16 FP16	INT8	INT2/4/8/16/32(RISC-V) INT2-8(RBE)	INT8/16/32(RISC-V) Weight: BVC3/6/8(Accel.) Activation: INT4/8(Accel.)
Peak Throughput [TOPS]	1.42@INT2	0.37@INT16 1.48@INT8	0.52-4.07	39.3@INT4 19.7@INT8 9.8@FP16	1.024	0.637@RBE2x2	2.09 ² @INT4xBVC3 1.04 ² @INT8xBVC6 1.04 ² @INT8xBVC8
Area Efficiency [TOPS/mm ²]	3.30@INT2	0.054@INT16 0.217@INT8	0.597	6.90@INT4 3.45@INT8 1.72@FP16	0.131	0.034@RBE2x2	1.215 ² @INT4xBVC3 0.608 ² @INT8xBVC6 0.608 ² @INT8xBVC8
Power Efficiency [TOPS/W]	44.1@INT2	20.5@INT8 5.1@INT16	27.56 3.98@GPT-2	11.59@INT8 MobileNetTPU 4.26@FP16 DeepLabV3	11.2@InceptionV3 10.7@ResNet50 10.5@MobieNetV1	12.4@RBE2x2 6.38@Mixed ResNet20 5.83@RBE4x4 ResNet18	33.98 ³ @INT4xBVC3 18.51 ³ @INT8xBVC6 15.91 ³ @INT8xBVC8 15.11 ⁴ @Mixed VGG16 12.49 ⁴ @Mixed DarkNet19 12.04 ⁴ @Mixed ResNet34

¹ All on-chip components are included; ² Measured at the highest performance point @ 1.1V; ³ Measured at the highest efficiency point @ 0.6V;

⁴ Average power efficiency @ 0.6V; ⁵ Mainly focus on MAC utilization; ⁶ Supports only 3x3 and 1x1 convolutions; ⁷ Fully supported and programmable

3.5 ASIC: Multi-core Multi-mode SoC

As illustrated in Chapter 2, the proposed three AI accelerators mainly focus on the acceleration of convolutional and matrix multiplication. Thus, the aforementioned single-core SoC designs incorporating the proposed accelerators can only use the single RISC-V core to compute the other operators to support more complex AI workloads. However, due to the low performance of the single RISC-V core, these operators can lead to a significant latency for data computation. Meanwhile, the results computed by the core usually need to be transmitted to other terminals (e.g., accelerator), which can further increase the latency for data transmission. Also, more comprehensive hardware platforms mean a higher complexity of the architecture, underscoring the importance of estimating and improving the reliability of AI applications. In short, three prominent challenges that demand attention include high end-to-end latency, support for diverse computational operators, and the reliability of AI SoCs in harsh environments.

The first challenge revolves around addressing high end-to-end latency, a critical factor influencing the real-time responsiveness of AI SoCs. In many applications, especially those requiring instantaneous decision-making, latency is a critical concern. This challenge prompts the exploration of novel architectures and methodologies to minimize latency, ensuring that AI SoCs can meet the stringent requirements of time-sensitive tasks.

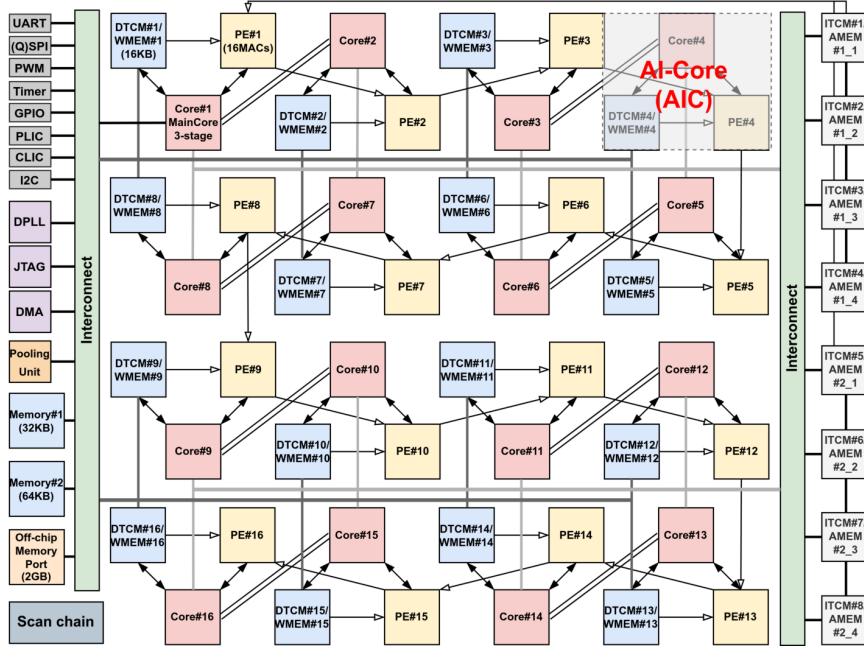


Figure 3.8: Multi-mode system-on-chip architecture.

The second challenge underscores the need for AI SoCs to seamlessly support a diverse array of computational operators. As the landscape of AI evolves, new operators and algorithms emerge, each with unique computational requirements. A robust AI SoC must be adaptable and versatile, accommodating various operators efficiently. This challenge inspires researchers and engineers to devise architectures capable of handling a wide spectrum of computational tasks, from traditional to cutting-edge algorithms.

The third challenge introduces the requirement for AI SoCs to operate reliably in radiation-prone environments. This consideration becomes particularly significant in applications such as space exploration, aerospace systems, and medical devices where radiation exposure is inevitable. Developing AI SoCs with inherent resilience to radiation challenges conventional design paradigms, prompting the exploration of fault-tolerant architectures.

In response to these challenges, a solution targeting edge AI applications is proposed: a multi-core multi-mode AI SoC. This SoC aims to tackle the aforementioned challenges, enhancing computation efficiency, adaptability, and reliability in AI SoC design.

3.5.1 SoC Architecture

As shown in Fig. 3.8, the proposed multi-mode SoC architecture comprises 16 3-stage vector RISC-V cores with 16 processing elements, featuring 16 16KB DTCM blocks, 8 32KB ITCM blocks, and 96KB shared memory blocks. To address the above 3 chal-

lenges, this SoC operates in various modes, including Acceleration Mode, Multi-core Mode, Fault-tolerant (FT) dual-core-lock-step (DCLS) Mode, and Fully-access Mode, it offers a versatile and resilient framework for diverse computing needs. Especially, the on-chip interconnect is based on crossbar connection [107], making it highly attractive for high-performance computing, and other applications requiring high-speed data transfer. The crossbar structure allows multiple inputs and outputs to communicate simultaneously, providing extremely high bandwidth as each input port can directly connect to each output port, reducing data transmission bottlenecks. Since each input and output in a crossbar structure has a direct connection, the data transmission path is very short, reducing data transmission latency. Additionally, crossbar connections support multiple data streams being processed in parallel, which is crucial for edge AI applications requiring high concurrency processing.

3.5.2 Operating Mode

The AI core operates in various modes to cater to different computing requirements. In the accelerator mode, it excels at convolutional operations and matrix multiplication acceleration, leveraging high-efficiency data flow and ensuring high data locality for efficient data-reuse strategies. Transitioning to the multi-core mode provides the flexibility of fully programmable general-purpose computing, accommodating diverse computational tasks. For safety-critical applications, the SoC offers a DCLS configuration, emphasizing reliability. Lastly, the fully-access mode facilitates fault injection and debugging, ensuring comprehensive testing and analysis capabilities.

Regarding end-to-end latency, the accelerator mode and the multi-core mode are integrated to eliminate the latency between core and accelerator. The memory blocks are shared in both these two modes, which means both the accelerator and these cores can access the data in these memories with low latency. In detail, after the completion of relevant computations done by these cores, the results can be directly stored in these memories. Then, after switching the chip into the accelerator mode, the accelerator can directly access these results with low latency and continue the following computations.

In terms of the adaptability for diverse AI workloads, as normal accelerators can only accelerate high-parallelism operations (e.g., convolution, and matrix multiplication), the multi-core mode is incorporated to accelerate other complex but less parallelizable operators (e.g., batch normalization). Thus, through the combination of the accelerator mode and the multi-core mode, the most prevalent NNs can be efficiently executed on one single chip with low latency.

Concerning reliability targeting safety-critical applications, an FT mode based on the multi-core mode is proposed. Through the configuration of DCLS, the adjacent two cores execute the same programs. Also, the intermediary data will be compared in real time to check the errors in the pipeline of these two cores. When the error is detected, the finite state machine (FSM) will flush the pipeline and roll back the pipeline state to

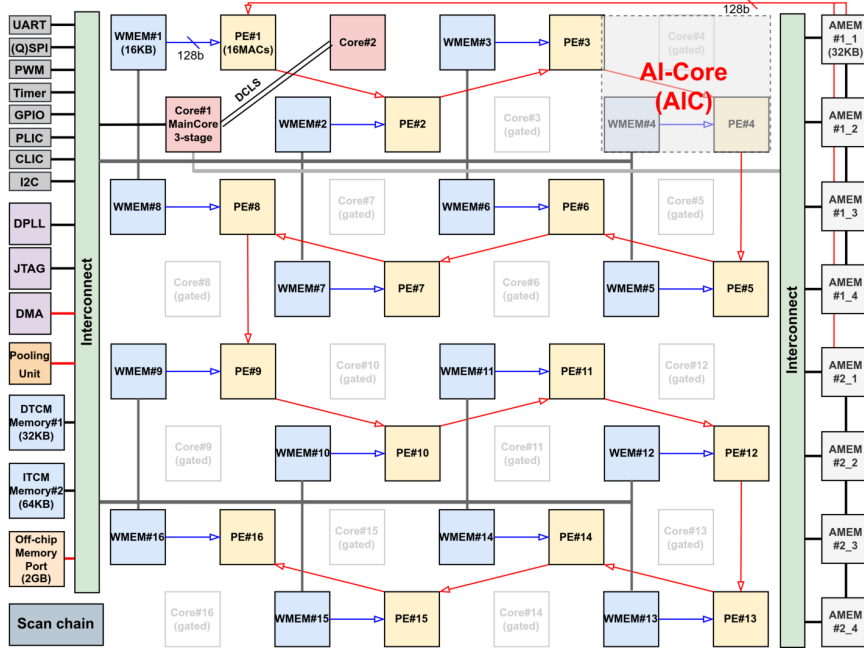


Figure 3.9: Accelerator mode architecture.

the previous stage to recover the error. Moreover, to evaluate the reliability of the SoC, the fully-access mode is offered to support effective fault injection via JTAG and scan chain.

In short, these four modes complement and assist each other in addressing the challenges of computational efficiency, latency, adaptability, and reliability in edge AI applications.

Integer-only Accelerator Mode

As shown in Fig. 3.9, an AI core (AIC) consists of one PE, one weight memory macro, and one RISC-V core. In the accelerator Mode, RISC-V core #1 and/or #2 are active, while the remaining cores are gated off. Each PE internally houses 16 8-bit MACs. Additionally, each PE is directly connected to a weight memory. The blue path, intra-AIC-order, represents the data flow for weights. The red path, inter-AIC-order, signifies the data flow for activations. In the accelerator mode, data in the memory interacts with external memory through DMA, and RISC-V core #1 is only required to configure the parameters of the accelerator's CSRs.

Multi-core Mode

In the accelerator mode, the SoC is limited to the acceleration of convolution and matrix calculations. To achieve compatibility with other general-purpose computing op-

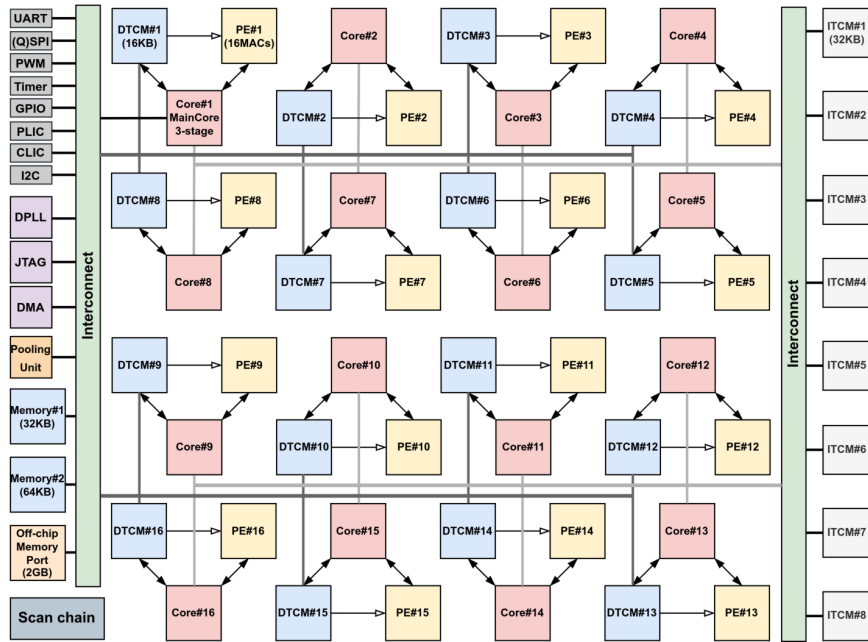


Figure 3.10: Multi-core mode architecture.

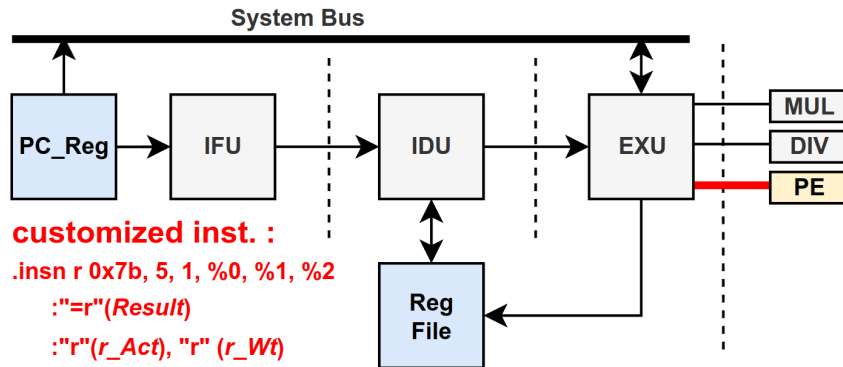


Figure 3.11: Vector-level computation using customized instruction.

erators, the multi-core mode is devised, as illustrated in Fig. 3.10. In this mode, 16 programmable cores are activated, with each core being associated with a PE. This indicates the reuse of PEs from the accelerator mode in the multi-core mode. Additionally, the weight memory from the accelerator mode is repurposed as the DTCM in the multi-core mode, while the activation memory is repurposed as the ITCM. Given that, except for the 3-stage multi-core components, the rest of the system's design can leverage the components from the accelerator, achieving a higher level of logic reuse. It is noteworthy that, owing to the need for the RISC-V cores to schedule PEs and perform vector-level computations targeting different operators, the instruction set is extended to enable RISC-V cores to control PEs. As depicted in Fig 3.11, traditional RISC-V

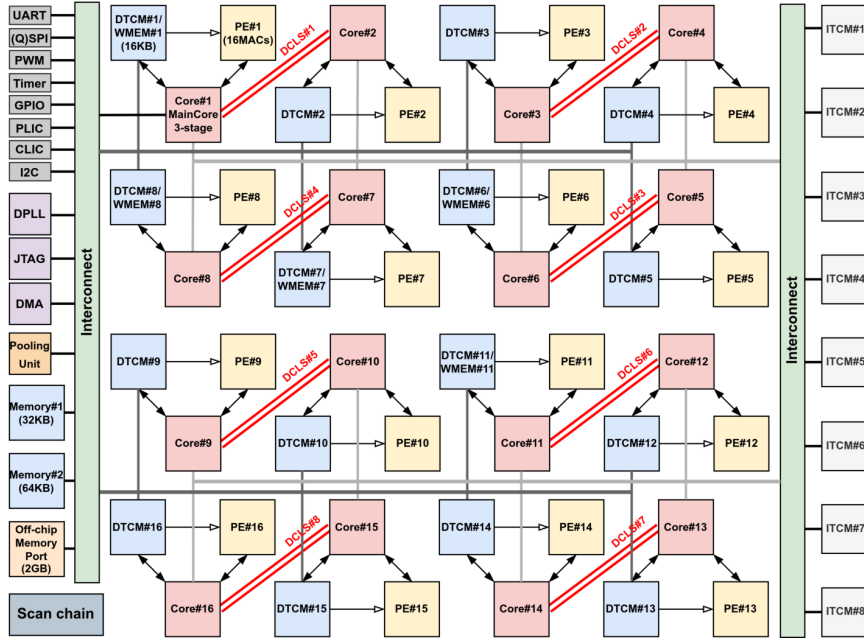


Figure 3.12: Fault-tolerant mode architecture.

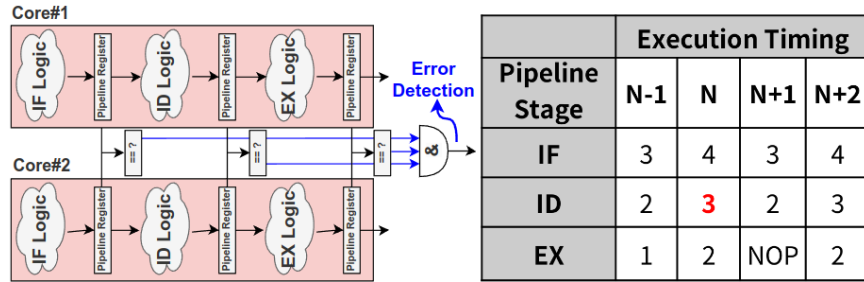


Figure 3.13: Error detection and correctness in Fault-tolerant mode.

cores only support single multiplication/division operations. The PE unit is physically connected to the execution unit and the programmable scheduling of PE through custom R-type instruction (i.e., r, 0x7b, 5, 1) is implemented. In vector-level computation, the immediate value in the register denotes the addresses of the two vector data to be computed. After certain clock cycles of computation, the result is written back to the register.

Fault-tolerant Mode

Through our fault injection and irradiation experiments in flash-based FPGA demonstrated in Section 4.4, the observation is that RISC-V cores are more susceptible to SEUs and exhibit greater vulnerability compared to the accelerator. As a countermea-

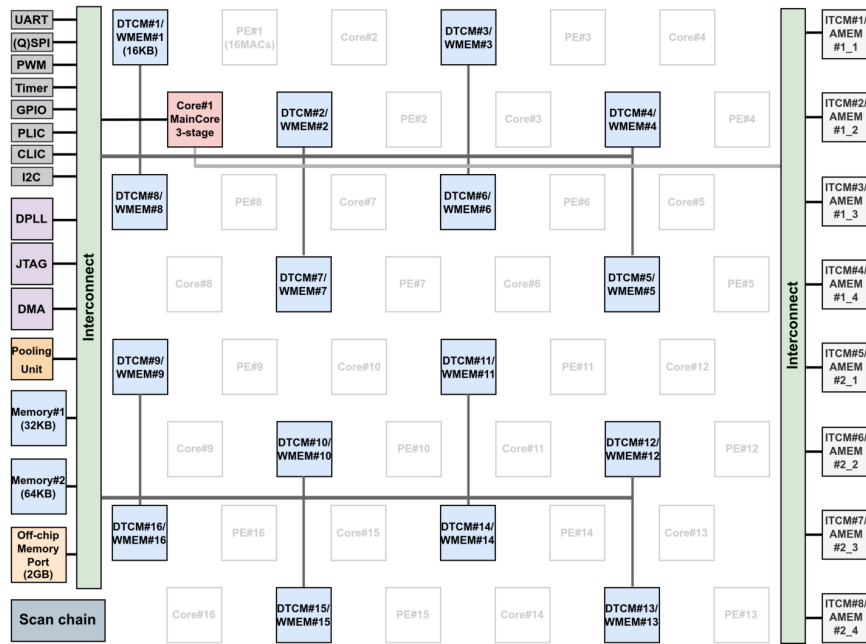


Figure 3.14: Fully-access mode architecture.

sure, an additional implementation of resilience is introduced based on the multi-core architecture, as illustrated in Fig. 3.12. This involves incorporating the functionality of DCLS. In this configuration, every two adjacent cores execute the same program concurrently. As shown in Fig. 3.13, an automatic flush mechanism is implemented by comparing the results at different stages of the pipeline, and in the event of a mismatch, the pipeline state is rolled back to rectify the error. For example, assuming the error is detected in the ID stage at timing N , the FSM will automatically roll back to the state of timing $N - 1$ at timing $N + 1$. At timing $N + 2$, the system can recover the error generated at timing $N - 1$. Simultaneously, the mismatch signal is accessible to users in real time through JTAG for monitoring and further analysis.

Fully-access Mode

To facilitate monitoring of the system's status, the fully-access mode has been introduced, as depicted in Fig.3.14. In this mode, both Core#1 and JTAG have full access permissions to the memory components. core#1 utilizes the C language to read and write to these memories, while JTAG can directly access them using an open-source debugger, OpenOCD. Furthermore, to access all on-chip registers, a built-in scan chain has been integrated into the design, accessible through the IO port, allowing access to all registers.

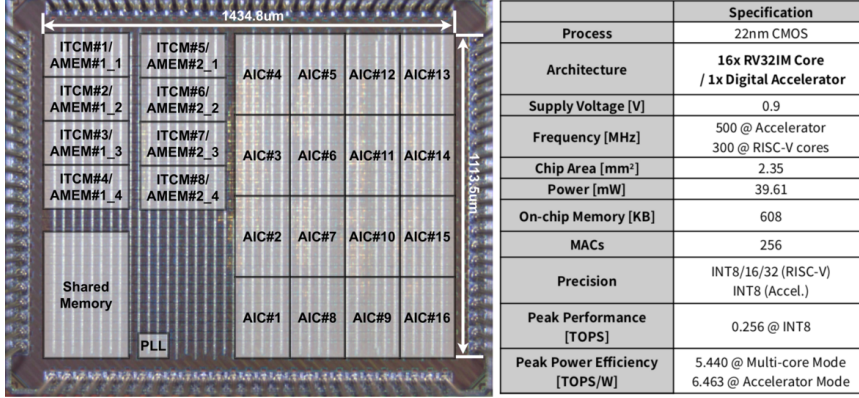


Figure 3.15: Die micrograph and metrics.

Table 3.5: Performance comparison with state-of-the-art multi-core SoC design.

	ISSCC23 [28]	This Work
Process	22nm	22nm
Architecture	1x RV32IMCFXpulp+ 16x RV32IMCFXpupnn+ RBE Accel.	16x RV32IM+ Digital Accel.
Voltage[V]	0.5-0.8	0.9
Frequency [MHz]	420	500@Accel. 300@RISC-V
Area [mm2]	18.7	2.35
Power [mW]	12.8-123	39.61
Memory [KB]	1152	608
Precision	INT2/4/8/16/32(RISC-V) INT2-8(RBE)	INT8/16/32(RISC-V) INT8(Accel.)
Throughput [TOPS]	0.637@RBE2x2	0.256
Power Efficiency [TOPS/W]	5.83@RBE4x4 ResNet20 6.38@RBE Mixed ResNet18	6.463@INT8, Multi-core 5.44@INT8, Accel.
Vector Computation in Core	No	YES
PE and Memory Reuse	No	YES
Fault Tolerance	No	YES
Mode Switch Overhead	NA	4 system cycles

3.5.3 Performance Evaluation

Fig. 3.15 presents the fabricated chip in TSMC 22nm process with 2.35 mm² area. This work exhaustively explores system-level edge AI chip implementation on the accelerator design and the multi-core design. It contains 608KB memory space. The accelerator, together with system-level mapping/scheduling, can execute convolutional operation, and matrix multiplication to compose the on-chip execution flow for a complete NN model. The several exceptional special operators and other general-purpose computations, such as batch normalization, will be executed on RISC-V cores in the multi-core mode.

The multi-core aspect of the proposed AI SoC capitalizes on the power of parallel processing. By integrating multiple processing cores, the computational workloads are

intended to be distributed efficiently, reducing end-to-end latency. This parallel architecture enhances the overall throughput of the AI SoC, enabling it to handle complex tasks with agility and speed.

Moreover, the multi-mode functionality introduces a dynamic dimension to the AI SoC's capabilities. In this context, multi-mode refers to the ability of the AI SoC to switch between different operational modes with a low latency (4 system cycles), each optimized for specific computational requirements. This adaptability ensures that the AI SoC can efficiently support a diverse range of computational operators, addressing the second challenge of computational diversity.

To address the unique challenge of radiation resilience, the proposed AI SoC incorporates a fault-tolerant mode (i.e., DCLS). By leveraging radiation-hardened components and fault-tolerant architectures, an AI SoC is intended to be created that can withstand and operate reliably in radiation-prone environments. This approach broadens the applicability of AI in domains where radiation resistance is important.

Furthermore, as shown in Table 3.5, it highlights the key advantages of this design over the SOTA design described in [28]. Notably, this design achieves a substantial power efficiency of 6.463 TOPS/W in the accelerator mode, which is comparable to the leading design's power efficiency. Additionally, the proposed vector RISC-V core supports vector-level computation, allowing for further improvements in computational efficiency, achieving 5.44 TOPS/W in the multi-core mode. This design also emphasizes logical reuse, enabling the PEs and memory to be shared across different modes, significantly reducing area overhead. Significantly, as almost all memories and PEs are reused in different modes, the additional overhead compared to the pure AI accelerator design only includes logic part of the RISC-V cores, which accounts for 12.2% of the entire SoC area consumption. Furthermore, the mode switch is highly efficient, requiring only four system cycles. Moreover, for reliability-critical applications, a fault-tolerant mode is introduced based on DCLS architecture, ensuring robust operation under various conditions.

In conclusion, the vision for a multi-core multi-mode AI SoC represents a holistic response to the challenges facing traditional AI SoCs. Through parallel processing, dynamic operational modes, and a focus on radiation resilience, this section aims to redefine the capabilities and scope of AI SoCs, paving the way for advancements in real-time processing, computational versatility, and reliability in challenging environments. This proposed paradigm shift holds the potential to unlock new possibilities for AI applications across diverse industries and sectors.

3.6 Discussion

In this chapter, the three proposed AI SoCs are involved with different technical contents as shown in Table 3.6. The single-core SoC with MBW Booth accelerator is a baseline in the design. This SoC lays the foundation of MP computation and data-reuse

Table 3.6: Comparison of our proposed AI SoCs.

SoC	Single-core w/ MBW Accel. (Section 3.3)	Single-core w/ BVC MP Accel. (Section 3.4)	Multi-core Multi-mode (Section 3.5)
Platform	Flash-based FPGA	22nm CMOS	22nm CMOS
Architecture	1x RVIMAC core + 1x MBW Accel.	1x RVIMAC core + 1x BVC MP Accel.	16x RVIM core or 1x INT8 Accel.
Supply [V]	1.8/3.3	0.6-1.1	0.9
Frequency [MHz]	16@RISC-V 100@Accel. 400@DDR4	230-1020@Accel. 10-200@RISC-V	500@Accel. 100@RISC-V
Area	123681 LUT4s 93972 FFs 140 DSPs 212 uSRAMs 293 LSRAMs	Core: 1.719mm ² Die: 2.750mm ²	Core: 1.598mm ² Die: 2.346mm ²
Power [mW]	NA	13.61-210.67	39.61
Memory [KB]	640	512	608
MACs	128 @ INT8 256 @ INT4 512 @ INT2	512 @ BVC8 512 @ BVC6 1024 @ BVC3	256 @ INT8
Multiplier Architecture	Multi-precision	Approximate Multi-precision	Conventional
General-purpose Acceleration	NA	NA	Vector Computation in Multi-core Mode
Fault Tolerance	NA	NA	Dual-core-lock-step
Fault Injection	UART	UART	OpenOCD + ScanChain
Peak Performance [TOPS]	0.0256@INT8 0.0512@INT4 0.1024@INT2	1.04@BVC8 1.04@BVC6 2.09@BVC3	0.256@INT8
Area Efficiency [TOPS/mm ²]	NA	1.215@INT4×BVC3 0.608@INT8×BVC6 0.608@INT8×BVC8	0.544@INT8
Accel. Power Efficiency [TOPS/W]	NA	33.98@INT4×BVC3 18.51@INT8×BVC6 15.91@INT8×BVC8 15.11@Mixed VGG16 12.49@Mixed DarkNet19 12.04@Mixed ResNet34	6.463@INT8
RISC-V Power Efficiency [TOPS/W]	NA	1.123	5.44

strategy. To further improve the computational efficiency and optimize the data flow, an approximate MP PE (i.e., BVC MP PE) and NMC-friendly data flow are constructed in the BVC MP accelerator. Therefore, a single-core SoC with a BVC MP accelerator is proposed targeting higher energy-efficiency computing. However, as the previous two SoCs can only accelerate the convolutional operation and matrix multiplication, the other operators need to be deployed and calculated on the RISC-V core. The computational efficiency is quite low for other operators. Therefore, a multi-core multi-mode SoC is proposed including 16 RISC-V cores. This SoC incorporates a logic reuse feature and can switch the operating mode between the multi-core mode and the accelerator mode with a 4-clock latency for mode configuration. In the multi-core mode, all 16 cores can be programmable to support the acceleration of different operators. Besides,

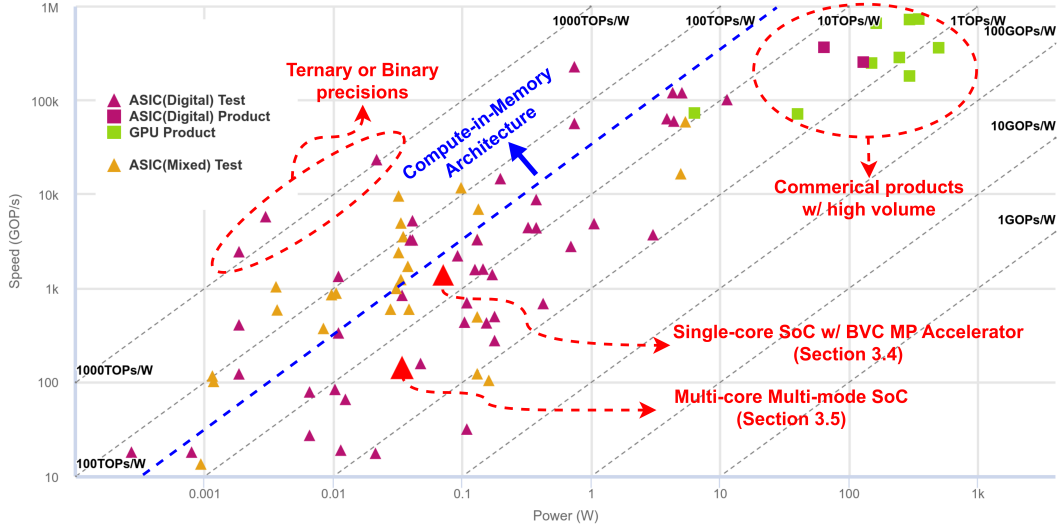


Figure 3.16: Neural network acceleration hardware comparison.

a fault-tolerant mode and a fully-access mode are built for safety-critical applications, fault injection, and debugging.

In addition, a comprehensive comparison is conducted for NN acceleration hardware platforms released in the past four years (2021-2024), as shown in Fig. 3.16, where the data was taken from [108]. Due to area limitations, most ASICs cannot match the throughput of commercial GPU chips. Furthermore, in academic designs, the highest power efficiency chips are based on ternary or binary precision calculations, achieving power efficiency exceeding 1000 TOPS/W. Moreover, the majority of chips with peak performance greater than about 50 TOPS/W are implemented using compute-in-memory architectures. However, these designs are primarily based on exploring peak power efficiency. Analysis of the average power efficiency when running actual NN models remains to be researched. Regarding the proposed designs, the single-core SoC with the BVC MP accelerator can achieve moderate power efficiency and throughput, with a minimum precision of BVC3. Additionally, since the multi-core multi-mode SoC is based on INT8 calculation precision, it does not excel in throughput and power efficiency. However, the advantage of this design lies in its high compatibility with various models and fault tolerance capabilities. Additionally, both the proposed designs support efficient data scheduling in their data flow. When data reuse is near full capacity, the average power consumption does not differ significantly from the peak power consumption.

3.7 Conclusion

To achieve high-performance AI SoC in edge computing, three different SoCs have been sequentially designed. Initially, a single-core RISC-V SoC system is implemented with a multi-precision accelerator based on the Booth algorithm on a flash-based FPGA, followed by a comprehensive evaluation and testing of the overall system. Subsequently, to further enhance performance, we optimized the data flow and PE units of the former and designed a single-core SoC system based on near-memory computing in 22nm CMOS technology. Finally, aiming for high versatility and reliability, a 22nm multi-core multi-mode SoC is constructed to achieve compatibility with specific operators and general-purpose computation. This design efficiently utilizes shared logic for reuse and incurs minimal overhead, facilitating seamless mode switching. Additionally, in the multi-core mode, it supports safety-critical AI applications. Besides, this design currently has only basic data available, with application-specific performance metrics to be determined in future testing. Also, the implementation of these NN accelerator hardware designs targets different hardware platforms, chip fabrication processes, operating voltages, operating frequencies, NN models, and other factors. Therefore, it is currently impossible to achieve an absolutely fair comparison of their performance parameters. Future work includes researching methodologies for reasonably comparing the performance of various NN accelerator hardware.

Chapter 4

Reliability Analysis of System-on-Chip in Edge AI

To evaluate the reliability of SoC in edge AI effectively, this chapter proposes a pre-silicon fault injection framework from a netlist view and a post-silicon fault injection framework from a physical view. Also, to comprehensively analyze the reliability of the proposed SoCs, we conduct neutron and alpha irradiation experiments to assess the vulnerable components in the SoCs. These simulations and experiments are mainly conducted on a flash-based FPGA platform and a 22nm ASIC platform. To enable thorough evaluation of our designs, we design all components in our SoC, which means these SoCs are white boxes for us. After a series of comprehensive simulations and experiments, we find that the RISC-V core, especially the memory in the core, is the most vulnerable part of the entire system. Besides, a hybrid-TMR method is proposed to enhance the reliability of the data flow of the accelerator.

4.1 Introduction

In pursuit of providing meaningful value to others and future endeavors, we have undertaken a comprehensive analysis of the reliability of our proposed AI SoC. This analysis encompasses both pre-silicon and post-silicon fault injection methodologies, each contributing unique insights into the robustness and performance of the system.

The pre-silicon fault injection approach is rooted in netlist-level simulation, offering a meticulous examination of potential faults before the physical implementation. This method provides a detailed understanding of how the AI SoC behaves under various fault scenarios, guiding preemptive measures for fault tolerance and resilience. Leveraging advanced simulation techniques at the netlist level allows for a thorough exploration of the system's response to faults, enabling proactive refinement of the design.

On the post-silicon front, a hardware-centric approach to fault injection has been employed. By implementing specific fault injection protocols in the actual hardware

environment, real-world scenarios are simulated to assess the system's fault-tolerance capabilities. This approach provides invaluable insights into the AI SoC's performance under actual operating conditions, bridging the gap between simulation and real-world behavior.

To further enhance the post-silicon analysis, fault injection experiments are conducted on an SoC realized on a flash-based FPGA mentioned in Section 3.3. These experiments involve both post-silicon fault injection and neutron radiation exposure, utilizing an SoC platform featuring a single RISC-V core and a multi-precision (INT2/4/8) Booth accelerator mentioned in Section 2.3. This multifaceted approach allows for a holistic evaluation of the AI SoC's reliability, encompassing both injected faults and the impact of radiation in a practical hardware setting.

In terms of pre-silicon fault injection and radiation experiments, the SoC design mentioned in Section 3.4 is implemented using a 22nm CMOS process as a prototype for reliability analysis. The pre-silicon fault injection methodology provides insights into the system's behavior when subjected to predefined faults, aiding in the identification and mitigation of vulnerabilities. Concurrently, alpha radiation experiments are conducted on the same SoC platform, featuring a single RISC-V core and an accelerator employing BVC MP architectures mentioned in Section 2.5. This dual-pronged approach offers a comprehensive understanding of the AI SoC's robustness at both the architectural and hardware levels.

In summary, the fusion of pre-silicon and post-silicon fault injection methodologies, coupled with radiation experiments, forms the cornerstone of the reliability analysis for the designed AI SoC. This meticulous examination not only sheds light on potential weaknesses in the system but also provides a foundation for refining the design, ensuring its resilience in real-world applications. The deployment of these fault injection strategies serves as a forward-looking approach to preemptively address challenges and deliver a robust and reliable AI SoC for current and future applications.

The rest of the paper is organized as follows. Section 4.2 reviews related works that inspire the importance of reliability analysis of AI SoC at the edge. Section 4.2.2 describes the proposed pre-silicon and post-silicon fault injection frameworks. Section 4.4 and Section 4.5 provide experimental results based on fault injection simulations and irradiation experiments evaluation through several case studies. Finally, Section 4.6 discusses the experimental results and Section 4.7 concludes this work.

4.2 Related Work

For safety-critical applications, soft errors need to be detected and alleviated in both hardware and software, and it is essential to reduce the probability of critical failures. To make a comprehensive exploration of the reliability of MBW applications in edging SoC devices, we adopt two proposed SoCs with an accelerator for relevant experiments. For this purpose, we survey the literature related to the reliability of MP NNs, fault

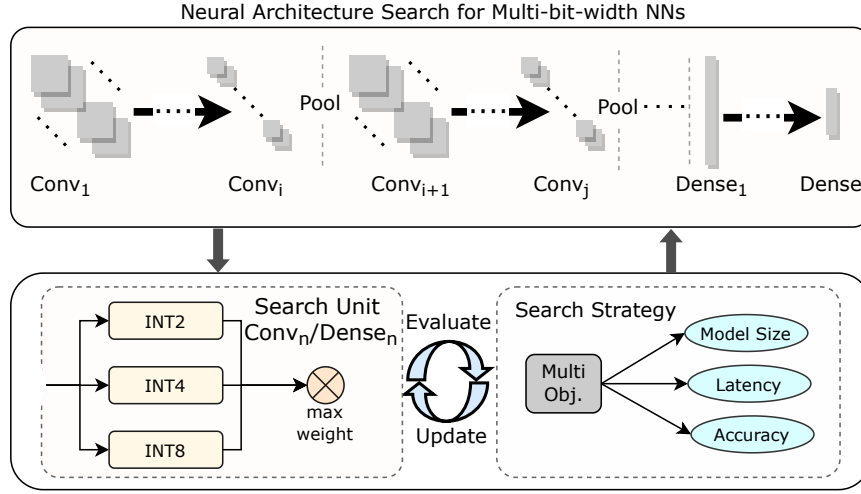


Figure 4.1: A subset of NAS for training MBW NNs.

injection framework, and soft error mitigation techniques in this section.

4.2.1 Optimization and Reliability of NN with Reduced Precision

As discussed in Section 1.1, NNs are highly effective but often consume significant computational power when running on high-performance devices. To address the challenges of low latency, memory overhead, and power consumption, researchers have proposed quantizing weights and activations into low-precision data, such as INT8/4/2, for edge devices [109, 110].

However, extremely low-bit quantization NNs, such as binary NNs [12], often sacrifice accuracy to improve power efficiency [78]. In contrast, MBW NNs strike a balance between model accuracy and power efficiency by assigning appropriate bit widths to individual layers [11], making them feasible for deployment on edge devices. To achieve high power efficiency in resource-limited edge devices, B. Lu et al. proposed performance predictors that consider power consumption, inference latency, and accuracy [111]. Nevertheless, determining the layer-wise bit-width of different layers in deeper NNs remains challenging to automatically obtain satisfactory NNs based on these predictors. Fortunately, NAS can address this challenge by automatically searching for efficient and low-latency MBW NNs with precision reduction, as shown in Fig. 4.1. NAS has also demonstrated better performance than uniformly aggressively quantized networks [112]. Moreover, NAS has shown unprecedented performance on large-size NNs, including accuracy-oriented and size-oriented tasks [79], making it applicable for energy-efficient data processing in edge devices [113].

Furthermore, to study the impact of data precision, Basso et al. compared the reliability of FP16, FP32, and FP64 in Matrix Multiplication [45]. They observed that

reducing precision improves not only the GPU performance and efficiency but also its reliability. In [114], Libano et al. deployed MNIST CNN on FPGAs using FP32, FP16, and INT8. The experiment showed that reducing the data precision in CNNs can significantly improve overall reliability. The reliability improvement comes from small memory usage. Libano et al. reported that binary quantization of weights in convolutional layers leads to lower vulnerability factors, but increases error criticality of misclassification [115]. The MBW NNs optimized by NAS also encounter an increase in error criticality in our FI experiment.

4.2.2 Fault Injection Framework

Fault injection is a standard technique for evaluating the reliability of systems[116]. Any fault, whether transient or permanent, can lead to application failures, potentially causing disasters in our daily lives. A high-efficiency FI environment is critical to evaluate the system's reliability. FI techniques are categorized into two types: 1) hardware-based (injecting faults into hardware), and 2) software-based (simulating hardware faults at abstract levels). No matter what kind of FI environment is deployed, the essence is to compare the differences between the golden version and the dirty version to analyze the system's integrity. In addition, the experimental results of hardware-based FI platforms [117] are closer to real situations, while the deployment of software-based FI platforms [118] is cheaper and more convenient for reliability analysis. It is important to note that hardware-based FI platforms typically offer higher throughput, enabling faster fault injection and simulation rates due to their direct control over hardware at lower levels. On the other hand, the throughput of software-based FI platforms depends on the performance of the software itself and the system's workload. Generally, hardware-based FI platforms have higher throughput compared to software-based ones.

While both methodologies apply to a wide range of applications, FI for accelerators can be directly realized on FPGAs or simulated by software on computers. As the AI accelerator is peripheral in the SoC, the output of CNNs in the accelerator will not crash the entire SoC but produce some mismatched outputs. It is important to note that mismatched outputs can potentially lead to significant failures at the application level, while the SoC itself continues to operate. On the other hand, unacceptable CSR values can cause dead loops or unresponsiveness in the SoC since CSR registers are used for the control of the AI accelerator. As a result, in the experiment, we focus on analyzing (1) how the impact of faults leads to silent data corruption (SDC) and (2) how misconfigured CSRs cause detectable unrecoverable errors across our SoC. Moreover, given the large size of NNs, it is crucial to develop a highly efficient FI platform for conducting reliability analysis.

To study the reliability of CNNs, Xu et al. injected faults into a processor and effectively stimulated the probability of NN failure caused by soft errors while analyzing node data of visual CNNs using the Simics simulation platform [119]. Similarly, Bar-

birotta et al. performed FI and analyzed the reliability of RISC-V cores on the UVM platform, relying on the co-design of software and hardware platforms [120]. Paiva et al. proposed a comprehensive FI platform for CubeSats software based on JTAG and UART interface [121], providing a good example of FI on the hardware side. Additionally, Mamone et al. presented a Python-based host computer software to replicate the effects of single-event upset (SEU) [122].

Overall, when analyzing the reliability of an AI accelerator, a dedicated FI platform that co-designs host software and hardware platforms is essential to improve test efficiency and flexibility.

4.2.3 Soft Error Mitigation Techniques

Mitigating soft errors is accompanied by the reliability analysis of SoC. Related works in literature are implemented based on redundant code and replicated modules. Triple Modular Redundancy is a very common and useful technique in soft error mitigation, and it is adopted in many studies. Wilson et al. [123] applied TMR techniques to a VexRISC-V processor via triplicating all FFs, LUTs, BRAMs, and DSPs on FPGAs, which achieved a 10× improvement in SEU-induced mean fluence to failure but with a cost of 4× resource consumption. Meanwhile, a popular mitigation technique at the instruction level, introduced by [124], involves implementing TMR to recover from soft errors occurring in the register file, and triplicating the verification points.

Although these works can be regarded as good references, the high hardware consumption is often unacceptable if the design aims for industrial products. Additionally, the register allocation technique (RAT), which is a compiler-based software technique that restricts the number of available CSRs when executing specific functions [125], is also used in many safety-critical designs. RAT reduces the effective bits exposed to radiation, contributing to reliability improvement. RAT does not involve code redundancy and is an architecture-independent approach. Abich et al. [126] attempted to improve soft error reliability via the adoption of two software-based mitigation techniques: Partial-TMR (P-TMR) and RAT.

Another related work includes a fault-tolerant RISC-V processor that incorporates TMR and Hamming code to protect vulnerable parts [127]. Similarly, Cho et al. [128] analyzed the effects of soft errors on two RISC-V cores: Rocket and Berkeley Out-of-Order Machine (BOOM). Dos Santos et al. conducted an irradiation experiment to assess the soft error susceptibility of a multi-core RISC-V ASIC platform (GAP8) by executing four simple algorithms and MNIST CNN [66].

Currently, although a number of soft error mitigation works are proposed aiming for processor systems, there is no related research that explicitly considers peripheral circuits, such as AI accelerators and their interfaces. Especially, a high-efficiency low-power SoC running AI applications is needed in reliability-demanding applications. Therefore, to analyze and enhance the reliability of SoC, constructing a 100% white-

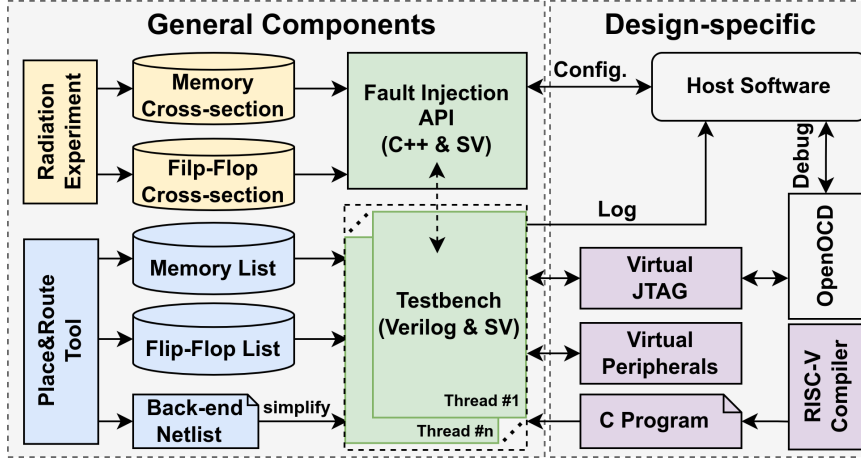


Figure 4.2: Simulation-based fault injection framework.

box SoC with a SOTA accelerator is crucial, allowing for an exhaustive exploration of all the details.

4.3 Fault Injection Framework

4.3.1 Pre-Silicon Fault Injection Framework

The FI process is designed to evaluate the soft error rate of our design in both white-box and black-box scenarios. To assess the overall reliability of our SoC, we develop a fault injection framework that operates on Synopsys VCS, as illustrated in Fig. 4.2. It is important to note that while the black-box approach typically involves injecting faults into COTS platforms via debuggers, our simulation-based framework also simulates this aspect of black-box FI. This is achieved by limiting the FI targets, which will be discussed in the next subsection.

In the general parts on the left, the cross-section data for memory and FFs plays a crucial role. This data, representing the likelihood of these units being affected by a radiation particle strike, serves as a benchmark for FI simulations. In the radiation community, ‘cross-section’ refers to the effective area where a striking particle can cause an upset, a key parameter in evaluating semiconductor devices’ reliability. This data, linked to the chip fabrication process, remains constant in specific radiation environments. We use this cross-section data to set the behavior for data bit-flips. The lists of memory and FFs include all available units for FI and are referenced during bit-flip executions. The back-end netlist is utilized to construct the basic testbench.

Furthermore, to simulate bit-flip operations effectively, the Direct Programming Interface (DPI) allows integration with foreign languages, like C and C++, with the simulator. We develop our FI API for backdoor access to memories/FFs, using a hybrid pro-

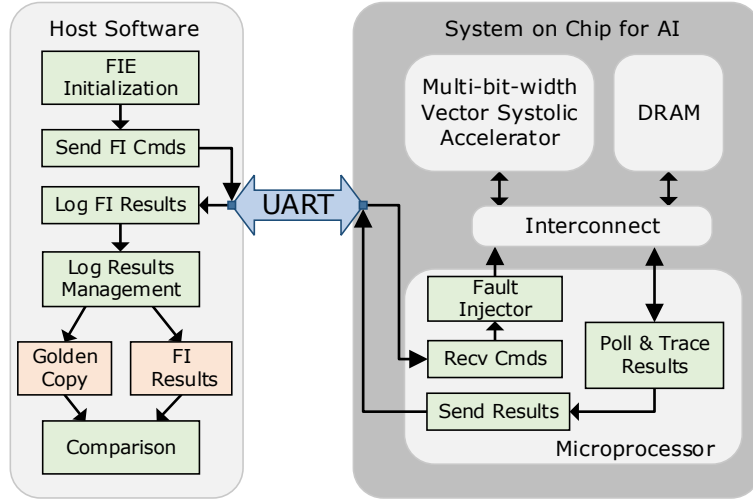


Figure 4.3: Fault injection environment for MBW accelerator.

gramming approach with C++ and System Verilog. Additionally, in randomly injecting errors, the timing is determined by a random function, ensuring consistent cross-section values for memories/FFs.

In terms of design-specific parts on the right, since our system includes a processor, we also require a C-language compiler to build AI applications. Additionally, to facilitate the monitoring of the entire system, we developed a host software application based on C++. This application can directly configure and retrieve runtime data from the testbench and can monitor and debug the entire system. For example, after the SoC crashes, we can interrupt the SoC and read register and memory values using an open-source On-Chip Debugger, OpenOCD [103]. By and large, this simulation environment not only allows for injecting errors randomly throughout the entire system to assess its reliability but also enables pinpoint FI to analyze which components within the system are more vulnerable. Besides, as our simulation only focuses on function, all components are simplified with fast functions.

4.3.2 Post-Silicon Fault Injection Framework

For the reliability analysis of the MBW accelerator, it is vital to develop a FI environment to reproduce the effects of SEU, in particular single bit upset (SBU), focusing on the weights, activations, states of the controller, and configuration parameters of CNNs. In addition, logging the results and saving them via host software for analysis is also necessary.

Our FIE is a hardware-software co-design. The SoC side of FIE, which is developed by C language, works on the RISC-V microprocessor e203, and the host side, developed by C++ language, works on the Qt platform as host software. The SoC for AI is

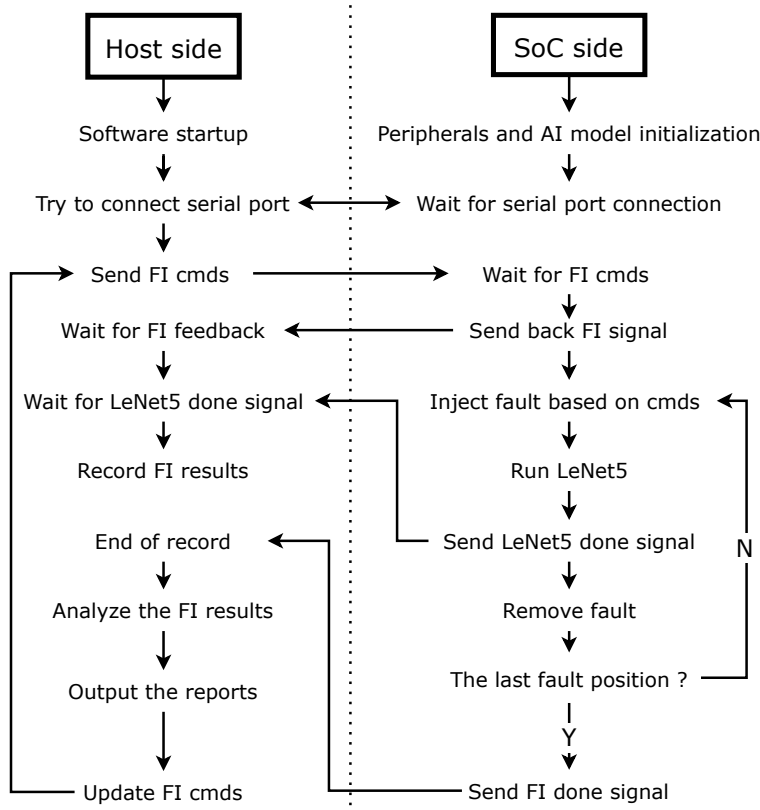


Figure 4.4: Fault injection flow.

controlled by the Qt software that can configure FI campaigns and record the experimental results. Also, our FIE can inject SBUs in different targets (e.g., the location of weights, activations, and CSRs), and count errors in NNs automatically. As shown in Fig. 4.3, the FIE is based on the UART protocol between MPF300T and the computer. Firstly, we define relevant data frames as FI commands and send them to the SoC side via UART after initialization. Secondly, the SoC side receives and decodes the commands. Thirdly, the SoC side injects the fault into the target object via the system bus based on the decoded FI commands. Fourthly, the poll and trace module will get the injection results and send them to the host side via UART. Finally, the host side retrieves all the results and compares them with a golden copy. Besides, the FIE can perform FI campaigns automatically, and all the faults can be injected into desired bits of the data in designated memory locations at given times. Meanwhile, the user has the option to enable or disable this feature, allowing host software to automatically restart the SoC in case of unresponsiveness or when it enters a dead loop without sending completion signals within a specified timeframe.

Fig. 4.4 depicts the FI flow of the whole system. Retrieving the instructions from the non-volatile memory and loading them into ITCM in RISC-V to initialize the MBW

accelerator and relevant peripherals (e.g., UART, GPIO) is the first setup for the SoC. Simultaneously, the host side initiates the software and builds a connection with our SoC. After building the connection, the host side can send the FI commands to the SoC side and wait for the feedback. Then, the SoC side will execute the FI commands (bit-wise operations) and run the AI benchmark after the completion of FI.

During the running period, the host side will poll one CSR to check whether the testing of the benchmark is finished. Once the benchmark is finished, the host side will record all the experiment results and the SoC will remove the faults. Furthermore, our FIE supports long-term FI experiments. So, if the FI commands contain multiple error injections, the SoC will loop the process of FI and run the benchmark periodically until all FIs are completed.

4.4 Single-core SoC with MBW Accelerator Experiment

4.4.1 Fault Injection Experiment

Fault Injection Experimental Methodology

The fault injection experiment aims to assess the sensitive parts in MBW LeNet5. To achieve this, we conduct an exhaustive 3-month experiment for data acquisition. First, we train the MBW LeNet5 using PyTorch with three different precision distributions, as explained in Section 2.3.1. This allows us to create three models with varying weights. Specifically, we have 61470, 30810, and 35737.5 bytes of weights in these models as shown in Table 2.1. Second, all the weights, images, and configuration parameters for the MBW accelerator and C program of RISC-V e203 are stored in external flash memory on FPGA boards before the interaction phase. Finally, we adopt our self-developed FIE to coordinate with the MPF300T hardware platform for SBU simulation.

At the beginning of the SBU simulation, all the weights and images will be transferred from external flash to DDR memory under the control of the RISC-V processor. Then, as described in Section 4.3.2, the processor will receive FI commands from the host via UART and decode them. It will then perform FIs on the corresponding weight data in DDR memory based on decoded commands. Configuring CSRs will activate the accelerator to perform model computations. Once the computation is completed, it will output the results to the host software and rectify the polluted weight data. The above process will be repeated until all weight errors have been injected. Namely, each bit in the weights is flipped exhaustively one by one. Also, we randomly select 100 data from the test dataset in the experiment, which means we can get 49.176, 24.648, and 28.59 million dirty samples for these cases, respectively. Besides, in this experiment, after linking the FPGA board to our FI host software and initializing the FI mode in FIE, the host software can coordinate with the MPF300T board and collect and analyze all the

Table 4.1: Fault injection at different bit positions in weights.

Layer	Bit Position	INT8 Model			INT4/8 model			INT2/4/8 model		
		Samples	Errors	Error Ratio	Samples	Errors	Error Ratio	Samples	Errors	Error Ratio
Conv1 (1)	0	15K	0	0%	15K	0	0%	15K	120	0.8000%
	1	15K	0	0%	15K	7	0.0467%	15K	217	1.4467%
	2	15K	1	0.0067%	15K	13	0.0867%	-	-	-
	3	15K	0	0%	15K	57	0.3800%	-	-	-
	4	15K	1	0.0067%	15K	94	0.6267%	-	-	-
	5	15K	0	0%	15K	127	0.8467%	-	-	-
	6	15K	0	0%	15K	150	1.0000%	-	-	-
	7	15K	5	0.0333%	15K	143	0.9533%	-	-	-
LAVS(esb) / Dur ^l (ns) / RT(ns)		8.2220e-5 / 1174570 / 299562.5			6.9417e-3 / 1174570 / 299562.5			3.9583e-3 / 1174570 / 301937.5		
Conv2 (2)	0	240K	0	0%	240K	408	0.1700%	240K	90	0.0375%
	1	240K	0	0%	240K	610	0.2542%	240K	206	0.0858%
	2	240K	1	0.0004%	240K	893	0.3721%	240K	302	0.1258%
	3	240K	11	0.0046%	240K	1415	0.5896%	240K	486	0.2025%
	4	240K	35	0.0146%	-	-	-	-	-	-
	5	240K	75	0.0313%	-	-	-	-	-	-
	6	240K	95	0.0396%	-	-	-	-	-	-
	7	240K	78	0.0325%	-	-	-	-	-	-
LAVS(esb) / Dur ^l (ns) / RT(ns)		5.8437e-4 / 198090 / 155250.0			6.5885e-3 / 198090 / 155375.0			2.1473e-3 / 198090 / 156562.5		
FC1 (3)	0	4800K	0	0%	4800K	347	0.0072%	4800K	1394	0.0290%
	1	4800K	0	0%	4800K	766	0.0160%	4800K	2320	0.0483%
	2	4800K	0	0%	4800K	1778	0.0370%	4800K	4784	0.0997%
	3	4800K	0	0%	4800K	2688	0.0560%	4800K	9349	0.1948%
	4	4800K	0	0%	-	-	-	-	-	-
	5	4800K	0	0%	-	-	-	-	-	-
	6	4800K	1	0.0000%	-	-	-	-	-	-
	7	4800K	46	0.0010%	-	-	-	-	-	-
LAVS(esb) / Dur ^l (ns) / RT(ns)		4.2300e-8 / 90 / 163937.5			5.0211e-6 / 90 / 164562.5			1.6062e-5 / 90 / 165625.0		
FC2 (4)	0	1008K	0	0%	1008K	64	0.0063%	1008K	7	0.0007%
	1	1008K	0	0%	1008K	200	0.0198%	1008K	38	0.0038%
	2	1008K	0	0%	1008K	436	0.0433%	1008K	67	0.0066%
	3	1008K	0	0%	1008K	699	0.0693%	1008K	81	0.0080%
	4	1008K	0	0%	-	-	-	1008K	93	0.0092%
	5	1008K	0	0%	-	-	-	1008K	102	0.0101%
	6	1008K	0	0%	-	-	-	1008K	178	0.0177%
	7	1008K	0	0%	-	-	-	1008K	378	0.0375%
LAVS(esb) / Dur ^l (ns) / RT(ns)		0 / 90 / 114812.5			1.2082e-6 / 90 / 107062.5			8.1527e-7 / 90 / 115437.5		
FC3 (5)	0	84K	0	0%	84K	22	0.0262%	84K	28	0.0333%
	1	84K	0	0%	84K	40	0.0476%	84K	42	0.0500%
	2	84K	0	0%	84K	74	0.0881%	84K	74	0.0881%
	3	84K	0	0%	84K	140	0.1667%	84K	161	0.1917%
	4	84K	0	0%	-	-	-	-	-	-
	5	84K	0	0%	-	-	-	-	-	-
	6	84K	2	0.0024%	-	-	-	-	-	-
	7	84K	56	0.0667%	-	-	-	-	-	-
LAVS(esb) / Dur ^l (ns) / RT(ns)		3.4800e-8 / 90 / 101500.0			1.656e-7 / 90 / 100062.5			1.8300e-7 / 90 / 101437.5		
ONAVS(esb) / Total RT(ns)		6.6667e-4 / 835062.5			1.3537e-2 / 826625.0			6.1227e-3 / 841000.0		

* "Dur^l" refers to the effective duration in the longest case, not the overall duration, of the weight of each layer staying in the buffer.
 * Supposing the operating period of the accelerator is P , the effective duration of each weight in LeNet5 can be expressed as:

$$Dur = P \times (1 + T_{out_current}) + P \times [(H_a - H_w + 1) \times (W_a - W_w + 1) - 1] \times ceil(l_w \div T_{in}) \times H_w \times W_w \times T_{out_current}$$
 Where $T_{in} = 16$ at INT8, 32 at INT4, and 64 at INT2; $T_{out} = 8$; $T_{out_current} = (O_{remaining} \geq T_{out}) ? T_{out} : O_{remaining}$
 * "RT", directly collected from the FPGA in practice, refers to the execution time of each layer per inference.
 * Error represents misclassification of NNs. Only SDCs, i.e., No DUEs are observed.

prediction outputs automatically.

The aforementioned experiment only focuses on FI to the data in NNs. Meanwhile, we need to estimate the effect of CSRs since CSR values can also be affected by radiation-induced upsets. Therefore, we inject faults to some key CSRs, as mentioned in Section 3.3. We execute the FIs similar to the aforementioned experiment. Namely, we flip every bit in correlated CSRs before starting the computation in the accelerator and test them using the same 100 image data.

Fault Injection Experimental Results

Table 4.1 lists FI experimental results to NN data. “Errors” represents the misclassification of NNs due to SDCs, where no DUEs are observed. We define Layer Architectural Vulnerability Scoring (LAVS) and Overall Network AVS (ONAVS) to analyze the reliability of weights throughout the entire model. LAVS and ONAVS are the metrics that consider the duration of each weight in the buffer, the averages of AVF across the layer, and the number of network parameters. LAVS and ONAVS are expressed as:

$$\text{LAVS}(l) = [\text{Error}(l)/\text{Sample}(l)] * \text{Dur}(l) * \text{Weight}(l),$$

$$\text{ONAVS} = \sum_{l=1}^L \text{LAVS}(l),$$

where l is the layer number, $\text{Error}(l)$ is the total error number in l -th layer, $\text{Sample}(l)$ is the number of FI samples of l -th layer, and $\text{Weight}(l)$ is the bit number of weights in l -th layer. Then, $\text{Error}(l)/\text{Sample}(l)$ represents the average AVF of l -th layer. One special treatment in LAVS and ONAVS is that we multiply the averages of AVF by the weight duration, Dur , of corresponding layers, respectively, for fair comparison since a longer duration time means a higher probability of having an upset in the accelerator. Here, as our accelerator’s data flow involves data reuse, we provide the duration equation in Table 4.1 for reference. Also, LAVS and ONAVS include $\text{Weight}(l)$ since the larger number of weight bits experience more bit flips. With LAVS and ONAVS, the network error rate per inference is expressed as $\text{ONAVS} \times (\text{SEU rate } [/\text{bit/s}])$. For a particular design, the SEU rate is identical, and hence LAVS and ONAVS represent the NN vulnerability.

The main conclusions are: 1) MBW networks have a higher susceptibility to critical errors than single-precision networks. The ONAVS parameters, error-to-second-to-bit (esb), of single INT8 LeNet5, INT4/8 MBW LeNet5 and INT2/4/8 MBW LeNet5 are $6.6667\text{e-}4$ esb, $1.3537\text{e-}2$ esb and $6.1227\text{e-}3$ esb, respectively. 2) The high bits, especially the sign bit, are more sensitive to faults. In this experiment, around 77.1% SDCs come from the high two bits. These conclusions have not been explicitly addressed in one paper, but have been mentioned in different contexts across multiple papers. In the literature, Libano et al. prove that low-precision layers heighten the error criticality of misclassification [115], aligning with our experiment. Additionally, some studies demonstrate that reducing precision can enhance reliability [45, 114]. However, merely reducing data precision does not guarantee an improvement in reliability. In our experiment, we observe that the LeNet5 model, being relatively small and having limited data reuse, exhibits similar execution times with different precisions. Meanwhile, the experimental results show that MBW LeNet5 has a higher misclassification rate, several hundred times greater than INT8 LeNet5. For larger networks, low-precision NNs can achieve significantly lower latency, several times faster than high-precision models (e.g., INT8 and FP32 VGG16). However, if the ONAVS of low-precision NNs is much

Table 4.2: Critical errors regarding CSRs in MBW accelerator.

CSR	Error Ratio / Acceptable (%)	Unacceptable (%)
STCONV	0 / 0	100.00
CBWIN	91.7557 / 100.000	0
CFADDR	75.5439 / 100.000	0
CFSHP	76.8271 / 83.0952	16.9048
CCONF	28.6260 / 35.0000	65.0000
CWADDR	69.8139 / 100.000	0
COADDR	77.5191 / 100.000	0
CDONE	0 / 0	100.00
STPOOL	0 / 0	100.00
PCONF	0 / 0	100.00
PIADDR	66.0663 / 100.000	0
POADDR	79.0672 / 100.000	0
PIOSHP	59.3511 / 100.000	0
PBWIN	88.5496 / 100.000	0
PDONE	0 / 0	100.00

higher than high-precision NNs, the low-precision models may be less reliable. However, if the ONAVS of low-precision NN is similar to high-precision NNs, low-precision NNs are more reliable due to shorter data duration in memory. In general, the reliability of low-precision NNs is determined by multiple factors, such as hardware architecture, dataset complexity, training strategy, and so on, rather than simply by data precision reduction. Meanwhile, those factors result in the difference of Dur in LAVS and ONAVS. Thus, ONAVS can guide the network reliability exploration.

The FI experimental results of CSRs in the accelerator are shown in Table 4.2. “Acceptable range” of CSRs indicates that the SoC can still perform the model calculation and output the results, while “unacceptable range” indicates that the SoC cannot complete the model calculation, namely DUE errors. Also, the parameter “Error Ratio” means the misclassification ratio in overall errors. Unlike FI of weights, about 50% of CSR tests can inflict misclassification errors. Meanwhile, about 40% of the tests can lead to unacceptable errors. The main conclusions are: 1) the bit-flip of CSRs is far more sensitive than that of weights in NNs, 2) data errors in the acceptable range can lead to a high probability of misclassification, 3) error values in the unacceptable range cause the accelerator to enter into a deadlock or hang the AXI bus.

4.4.2 Fault Tolerant Techniques

The above results of our FI experiment indicate that improving the reliability of the accelerator in safety-critical scenarios is necessary. As mentioned in section 4.4.1, the

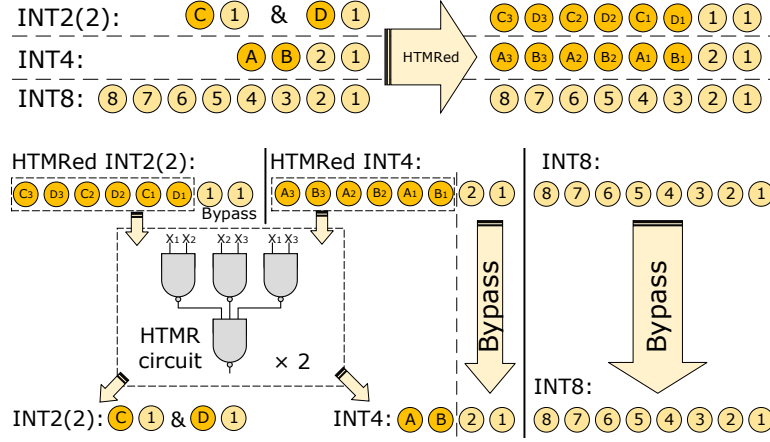


Figure 4.5: Hybrid triple modular redundancy (a) Encoder. (b) Decoder.

Table 4.3: Resource overhead of TMR_CSR and HTMR in accelerator.

	Conv Unit		Pool Unit	
	TMR_CSR	HTMR	TMR_CSR	HTMR
4LUT	1668	6406	658	757
DFF	1398	4311	724	785

accelerator is highly sensitive to CSR values. Therefore, we directly introduce the TMR method to CSRs to remedy this situation. Moreover, the FI experiment indicates that higher bits of data are more sensitive than lower bits. To improve the reliability in low precision, we can introduce suitable mitigation techniques based on the data format stored in the buffer. In the original accelerator, each INT8 data, each two-packed INT4 data, and each four-packed INT2 data occupy a 1-byte space. Exploiting this packing feature, we propose a hardware-based hybrid TMR (HTMR) method compatible with INT2 and INT4 computations to improve the reliability of the accelerator. The basic idea of HTMR is bit-selective TMR for INT2 and INT4 as shown in Fig. 4.5. Bits of C and D in two INT2s and bits of A and B in one INT4 are triplicated. Meanwhile, the HTMR mechanism sacrifices the throughput of INT4 and INT2 data. Namely, for each INT4 data or every two INT2 data, there are 4 redundant bits per byte. Besides, INT8 data is directly bypassed without any mitigation mechanisms due to its higher reliability in comparison to INT4/2 data as shown in Table 4.1. In this modified fault-tolerant design, each INT8 data, each INT4 data with HTMR, or every two packed-INT2 data with HTMR occupy 1-byte space.

Fig. 4.6 shows the architecture of our MBW fault-tolerant accelerator. For CSRs, we simply adopt the TMR method to triplicate every CSR and vote on three redundant inputs to determine final CSR values. As for data flow, we integrate HTMR encoder/de-

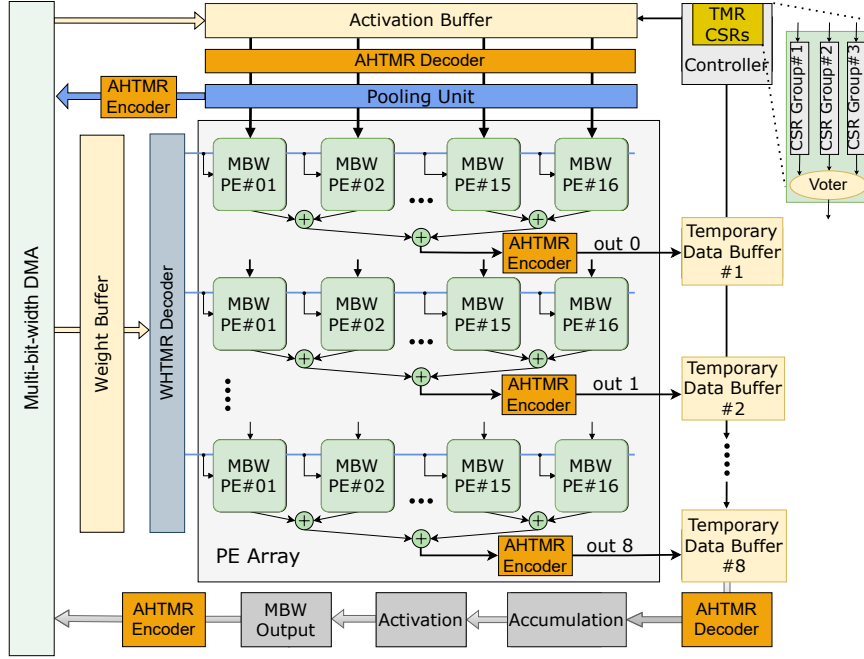


Figure 4.6: Architecture of multi-bit-width fault-tolerant accelerator.

coder modules into the accelerator without significantly changing the data flow in our accelerator. HTMR can effectively fix most data errors in three data paths: 1) from weight/activation buffers to PE units, 2) from PE units to temporary buffers, and 3) from output buffers to DRAM. After integrating the HTMR units, the SDC events caused by the SEU of weights in high 1 bit in INT2 and the SEU of weights in high 2 bits in INT4 are diminished.

We pre-encode weights and activations into HTMRRed format as shown in Fig. 4.5 and store these data in DRAM. All encoded data will be sent to the HTMR decoder and distributed to the PE array after being decoded. Regarding the pooling unit, the outputs will be directly transmitted into the Activation HTMR (AHTMR) encoder. As for the convolutional unit, since the final outputs cannot be calculated immediately, we incorporate another group of AHTMR decoder/encoder adjacent to temporary buffers for accumulation operations to enhance the reliability of INT2 and INT4 cases. The mitigation technique can effectively reduce 72.72% weight-induced errors in the INT4/8 model, and 74.93% in the INT2/4/8 model. The resource overhead required for applying TMR CSR and HTMR techniques is shown in Table 4.3, where the resource overhead is less than 9% of the entire SoC. Also, the throughput of INT2 and INT4 will decrease by half due to the transmission of TMR bits.



Figure 4.7: Experimental setup for neutron irradiation at CYRIC, Tohoku University.

Table 4.4: Four reliability configurations for neutron irradiation.

	Reset	TMR_I/DTCM	TMR_CSR
Config. 1	No	No	No
Config. 2	Yes	No	No
Config. 3	Yes	Yes	No
Config. 4	Yes	Yes	Yes

Table 4.5: Event statistics of three models in four reliability configurations.

Model	Board	Configuration 1					Configuration 2					Configuration 3					Configuration 4				
		Fluence (n/cm ²)	TCE	TAE	CCE	CAE	Fluence (n/cm ²)	TCE	TAE	CCE	CAE	Fluence (n/cm ²)	TCE	TAE	CCE	CAE	Fluence (n/cm ²)	TCE	TAE	CCE	CAE
INT8	MPF300T	1.563e+7	6	13	19	6	9.098e+6	4	1	2	1	1.407e+7	0	2	0	0	9.682e+6	1	0	0	0
INT4/8	MPF300T	1.511e+7	6	7	16	4	8.793e+6	2	0	2	4	1.360e+7	0	2	0	1	9.357e+6	0	0	0	0
INT2/4/8	MPF300T	1.461e+7	3	10	21	5	8.503e+6	5	4	2	1	1.315e+7	0	4	0	1	9.049e+6	0	2	0	1
INT4/8	Genesys2	1.414e+7	4	15	25	8	8.227e+6	10	11	3	16	1.272e+7	1	1	1	8	8.755e+6	1	2	0	4
Total		-	19	45	81	23	-	21	16	9	22	-	1	9	1	10	-	2	4	0	5

4.4.3 Neutron Beam Experiment

Neutron Beam Experimental Methodology

As shown in the previous section, we analyze the reliability of our accelerator using FI experiments. We have found that the AVF of CSRs is significantly higher than that of weights, and the FI of CSRs has a high probability of causing DUE in the SoC. In addition, we need to evaluate the robustness of the entire SOC in the radiation environment to make a comprehensive comparison. For this purpose, this work adopts a neutron irradiation experiment to evaluate the reliability of our SoC including microprocessors,

accelerator CSR, and weights. The neutron irradiation experiment is an effective way to evaluate the reliability of this SoC without any exceptions and overlooking and can help identify potential vulnerabilities. Thus, we can analyze the reliability of these components in our SoC through this experiment.

We perform a 4-day quasi-monoenergetic neutron irradiation experiment at Cyclotron and Radioisotope Center (CYRIC) at Tohoku University [129]. A 70-MeV proton source produces the neutron beam, and the neutron beam has a flux peak at the energy near 70 MeV. To cover INT8, INT4/8, and INT2/4/8 models mentioned in Section 2.3.1, we prepare three MPF300T boards that mimic ASIC SoCs thanks to radiation-tolerant CRAM, namely FFs and BRAMs are sensitive to radiation similar to ASIC SoCs. Moreover, to compare ASIC SoC and SRAM FPGA-based SoC, we prepare one extra Xilinx Genesys2 FPGA board running the INT4/8 model. Meanwhile, due to the fact that SRAM-based devices are sensitive to radiation, they may accidentally change the configuration information in CRAM. To mitigate this problem, we adopt SEM (Soft Error Mitigation) to scan CRAM in real time and recover configuration errors caused by radiation. Fig. 4.7 shows the board setup for the irradiation experiment.

To identify and analyze the vulnerable components of our SoC and rank them according to severity, we build four different reliability configurations using the control variable method for comparison as shown in Table 4.4. Config. 1 does not integrate any mitigation techniques and accumulates the errors without resetting operations until the system crashes. In Config. 1, we suspect the ITCM and DTCM SRAMs are the most sensitive parts of the SoC. Since we do not refresh the contents of these two SRAMs frequently, especially the ITCM SRAM, they will accumulate errors continuously and crash the SoC frequently, which increases the difficulty of data collection. To remedy this situation, we build Config. 2 which could reset the SoC (i.e., RISC-V, Accelerator, DDR and Bus interconnect) after each round of data collection. On the basis of the experimental data in Config. 2, we discover that ITCM and DTCM SRAMs still cause some critical errors and errors in the CSRs in the accelerator can lead to dead-loop or classification errors of NNs. To better estimate the vulnerability of SRAM in the e203 and the CSRs in the accelerator, Config. 3 replaces the normal SRAM with TMR SRAM to improve the robustness of the RISC-V core, where TMR is selected instead of ECC for just implementation simplicity. In comparison to Config. 2, we could identify the reliability of RISC-V. In addition, Config. 4 replaces the normal CSRs with TMR CSRs in our accelerator so that we can evaluate the reliability of the entire system and analyze the error contribution of CSRs when compared with Config. 3. Besides, in the experiment, we only found 2 samples of misclassification caused by weight/activation bit-flip in LeNet5. Then, we do not build an AI accelerator with the HTMR mechanism in this radiation experiment since enough events are difficult to obtain for comparison.

When running the CNNs, not all errors can be regarded as critical errors. Even though some outputs in the middle layers are not expected, the classification result is

still correct. Here, we identify four error classes: 1) Tolerable Core Event (TCE): RISC-V core experiences some misbehavior, but it does not affect the function of the NN application; 2) Tolerable Accelerator Event (TAE): classification result of a given image is correct, but the middle outputs are not as expected; 3) Critical Core Event (CCE): the RISC-V core runs away or crashes; and 4) Critical Accelerator Event (CAE): the accelerator has no correct response or correct classification result. Furthermore, we use the serial port to monitor all errors in the accelerator (i.e., TAE and CAE), as well as tolerable errors in the RISC-V core (i.e., TCE). When the RISC-V core encounters intolerable errors and is not able to respond to the commands from host-side software, we use the debug function in FPGA software to monitor the RISC-V privilege level registers to detect the error. This approach allows us to accurately identify and analyze errors during the experiment and comprehensively understand the SoC's fault tolerance and reliability.

Neutron Beam Experimental Results

Table 4.5 shows the event statistical results. The fluence data of four boards, ranging from $8.227\text{e}+6 \text{ n/cm}^2$ to $1.563\text{e}+7 \text{ n/cm}^2$, are also listed. Furthermore, the cross-section (cm^2), which is the number of observed errors divided by the entire irradiated neutron influence, is shown in Fig. 4.8. In the experiment, we select a set of data from the MNIST dataset that contains 100 image data as those mentioned in section 4.4.1. We regard the testing of 100 images as one event to make event statistics intuitive. In these four configurations, Config. 1 corresponds to a conventional design and operation without any countermeasures to soft errors, and the results reflect the reliability of the original SoC. In Config. 2, we reset our application after each sampling so that the RISC-V core itself would not accumulate errors continuously. Moreover, in Genesys2, even when we reset the programmable logic circuit, the CRAM may accumulate errors since we do not reload the bitstream to reconstruct the circuit. On the other hand, even though we have adapted SEM interface IP into Genesys2, the SEM IP itself has limited efficiency in scanning and rectifying the error CRAMs of SRAM-based boards. In fact, in Config. 2/3/4, even if we reset the system, the error with the highest proportion in SRAM-based FPGAs is CAE, whereas this phenomenon does not occur in flash-based FPGAs.

The results of MFP300T in Configs. 1 and 2 indicate that the RISC-V core is more sensitive to the accumulated errors than the accelerator, which comes from the observed large cross-section reduction of 81.59% in CCE from Config. 1 to Config 2 in MFP300T. In the RISC-V core, the data in I/DTCM determine the running of the application. When the data in I/DTCM experiences SEUs and the errors are accumulated, the SoC could crash easily. Meanwhile, after deploying the TMR I/DTCM in the RISC-V core (Config. 3), we find that the CCEs almost decrease to zero and the other events also have a significant decrease. Namely, the errors in the accelerator become dominant. Furthermore, after deploying TMR CSRs into our design (Config. 4), the

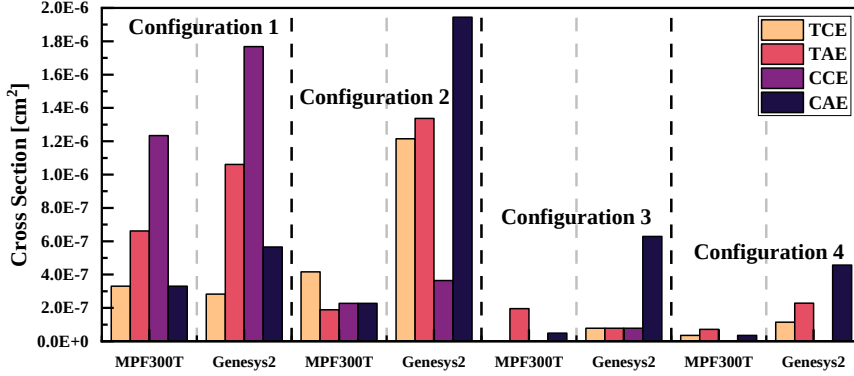


Figure 4.8: Neutron cross section for four reliability configurations.

errors generated at the accelerator decreased. The overall cross-section is reduced by 78.05% compared with Config. 1.

The observations indicate that the RISC-V core is more vulnerable than the accelerator, and implementing mitigation techniques (e.g., Error Correcting Codes (ECC), TMR) is necessary for instruction and data memory to strengthen the SoC. In addition, applying mitigation techniques to these memories could reveal the vulnerability of the interface between the processor and accelerator. Besides, the impact of precision differences in NNs is limited. Meanwhile, after deploying the above-mentioned countermeasures, the accelerator errors are dominant. In this case, implementing HTMR can effectively improve the reliability of weights. Thus, it is up to the specific application requirements to decide whether to deploy the mitigation techniques listed above. Depending on the criticality of the system and the level of reliability required, a combination of these techniques may be necessary to ensure the SoC's overall reliability to potential faults.

Comparison with Related Work

Table 4.6 lists the related works in soft error assessment of NN applications. Even though these designs are realized on different hardware platforms and have different architectures, there are two main ways to improve the reliability of the system: 1) using redundant mechanisms to repeat those crucial bits, susceptible to errors or vulnerabilities caused by FIs or radiation in AI applications, in vulnerable components and 2) reducing the usage of crucial bits in vulnerable components exposed to radiation. When analyzing our SoC architecture from a white-box perspective, it is important to select or design appropriate methods that improve system reliability while minimizing overhead. Therefore, as the data flow in our accelerator is shareable at different precisions, we propose the HTMR method to improve both the INT2 and INT4 cases. Moreover, as I/DTCM in the microcontroller and CSRs in the accelerator are the most vulnerable parts verified by the irradiation experiment, we can adopt TMR or ECC methods to these components

Table 4.6: Related works in soft error assessment of NN applications.

Work	Model / Dataset	Datatype	Approach	Platform	Mitigation
[114]	CNN / MNIST	FP32, FP16, INT8	Neutron irradiation	Zynq-7000 & UltraScale+	-
[126]	MobileNet / ImageNet	Mixed INT2/4/8	Simulation	Cortex-M7	P-TMR & RAT
[130]	2-L NN / Iris Flower & 7-L CNN / MNIST	-	Emulation & Neutron irradiation	Zynq-7000	P-TMR & F-TMR
[131]	ANN / MNIST	-	Neutron irradiation	NeuroShield	P-TMR
[132]	7-L CNN / CIFAR-10	Fixed point	Neutron irradiation	Cortex-M7 & Cortex-M4	RAT
Ours	LeNet5 / MNIST	Mixed INT2/4/8	Emulation & Neutron irradiation	MPF300T	HTMR & F-TMR
P-TMR: Partial TMR; F-TMR: Full TMR; L: layer					

to further improve the reliability of our SoC. In summary, our efforts contribute to the development of SoC design with accelerator architectures from a white-box perspective, prioritizing reliability in safety-critical applications. Our research on the reliability analysis of SoC with MBW accelerator provides valuable insights for designing reliable hardware for AI applications.

4.5 Single-core SoC with BVC MP Accelerator Experiment

This section focuses on a 22nm AI SoC prototype highly optimized for energy-efficient computation mentioned in Section 3.4, combining pre-silicon and post-silicon measurements to evaluate AI SoC reliability and radiation effects comprehensively. The discussion is secured by solid silicon results of the latest AI SoC.

In our study, the primary aim is to ascertain the reliability and accuracy of the estimated soft error impact for AI SoC in black-box/white-box cases. For this, we employ the SoC running an MP VGG16 network with 71.17% TOP-1 accuracy under the ImageNet dataset for both alpha irradiation experiment and fault injection simulation.

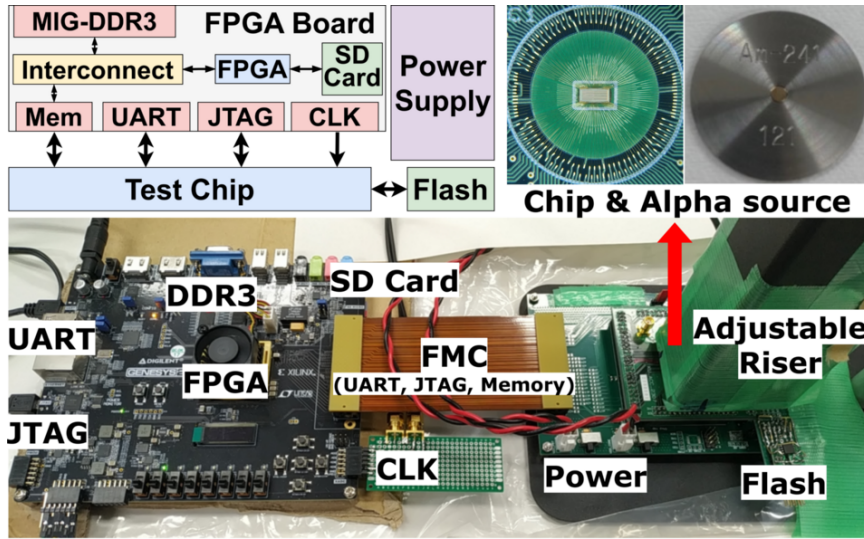


Figure 4.9: Experimental setup for alpha irradiation.

Alpha experimental results are regarded as a golden reference, and FI simulation results are analyzed on the basis of the definitions of white- and black-box cases mentioned in Section 4.5.2.

During the execution of NNs, not all errors lead to critical consequences. We categorize errors into four classes, based on the severity of their impact: 1) minor Silent Data Corruption (mSDC), where the classification outcome for an image remains correct despite unexpected intermediary outputs; 2) critical SDC (cSDC), where there is an incorrect classification result; 3) minor Detectable Unrecoverable Error (mDUE), the accelerator fails to respond or malfunctions but it can be restarted without power cycling; and 4) critical DUE (cDUE), which involves the RISC-V core either running out of control or crashing and requires the system rebooting. Notably, we do not observe any cSDCs in the experiment due to the high inherent reliability of MP NNs and the limited amount of irradiated particles.

4.5.1 Post-silicon Alpha Irradiation Experiment

Setup

To assess the robustness of our AI SoC and build a golden reference by measuring actual silicon operations, we adopt a 22nm AI SoC chip mentioned in Section 3.4 and conduct alpha irradiation experiments. This approach aims for a comprehensive analysis from a post-silicon perspective without omitting radiation physics and circuit operations, thereby forming a golden reference for comparing the white- and black-box cases.

Fig. 4.9 depicts the experimental setup employed for the irradiation experiment. We

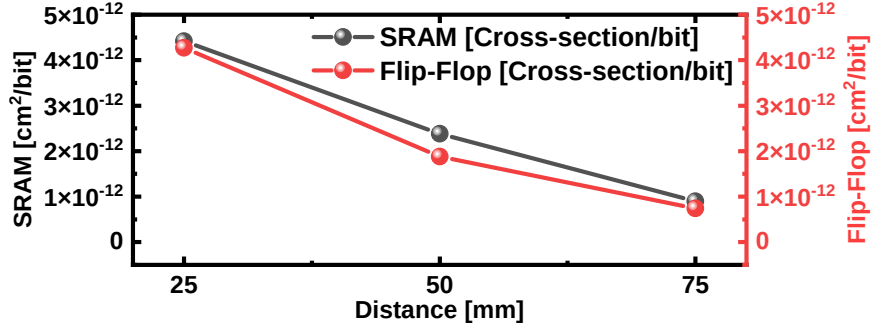


Figure 4.10: Measured error cross-sections in SRAM and FF.

Table 4.7: Event statistics in alpha irradiation experiment.

Distance (mm)	Execution Mode	Fluence (n/cm²)	mSDC	mDUE	cDUE
25	ITCM	7.8516e+09	168	387	244
50	ITCM	7.5936e+09	84	166	106
75	ITCM	6.6845e+09	55	116	69
25	FlashXIP	6.3114e+09	142	208	2
50	FlashXIP	7.7358e+09	91	134	1
75	FlashXIP	7.9577e+09	43	49	2

perform an irradiation experiment using an 8kBq Am-241 alpha source with a 2.4mm diameter, where the distance between the alpha source and the chip is varied to control the number and energy of alpha particles reaching transistors in the chip. We prepare a single board featuring our self-developed AI SoC. To begin with, we perform a simple test to estimate the error cross-sections of each type of memory and FFs. Fig. 4.10 shows the measured data, which is used as reference data in Section 4.5.2. Note that such cross-section data of basic storage elements could be provided from a foundry to industry.

Furthermore, we select 100 images from the dataset for verifying chip operation and classification accuracy. The testing of 100 images is regarded as one event to make event statistics intuitive. We employ the serial port and JTAG port to track all errors. The serial port is used for monitoring mSDC, cSDC, and mDUE. JTAG is employed to monitor cDUE. When the system crashes, JTAG dumps all core register data through the OpenOCD interface. All logs are recorded in the software backend.

Results

During the initial round of the experiment, we observed that the system has a relatively high likelihood of system crash, with the majority of errors originating from the ITCM. Consequently, we introduce an additional operating mode called Flash execute-in-place

(FlashXIP), where code can be executed directly on the external flash memory. This mode contrasts with the ITCM mode, where the program is first copied to the on-chip ITCM SRAM, and then the code is executed on the ITCM SRAM. Note that FlashXIP operation can be regarded as the chip implementation in which ECC is applied to ITCM, and correct instructions are always given¹.

Table 4.7 presents the statistical breakdown of events, accompanied by fluence data for all 6 cases, which span a range from $6.3114\text{e}+9$ n/cm² to $7.9577\text{e}+9$ n/cm². Fluence is defined as the number of irradiated particles per cm².

The results for our chip under ITCM mode and FlashXIP mode suggest that the ITCM is markedly more susceptible to error accumulation than the accelerator. This is evidenced by a substantial 98.8% reduction in cDUEs from ITCM mode to FlashXIP mode. In the RISC-V core, the integrity of instruction data within the ITCM is pivotal for application stability. Accumulation of errors within these regions could precipitate a system crash. This result suggests ECC adoption for reduced system crash rate.

Our alpha experiment highlights that the memory in the RISC-V core is more vulnerable compared to the AI accelerator. Moreover, the majority of mDUEs in the accelerator are due to bit flips in the I/DTCM detected by JTAG. In contrast, bit-flips in the memories of the AI accelerator have a minimal impact, as they seldom result in misclassifications. Ultimately, the implementation of error mitigation strategies will depend on the specific requirements of the application, the criticality of the system, and the desired level of reliability needed to safeguard against potential faults.

4.5.2 Pre-silicon Fault Injection Simulation

Soft Error Estimation from Black- and White-box Views

In recent years, black-box and white-box soft error analyses have been applied in the field of hardware design, particularly in the design of AI SoCs. Lotfi et al. evaluated the resiliency of NN-based automotive object detection running on a GPU through neutron beam experiments [46]. However, their fault injection experiment is limited to NN weights and input images, meaning that only black-box analysis is performed. On the other hand, Cheng et al. conducted an in-depth analysis of their custom AI SoC implemented on a flash-FPGA with neutron irradiation and fault injection [133]. However, their fault injection setup does not directly correspond to white-box or black-box analyses.

Furthermore, in assessing the reliability of these approaches, the study by Balasch et al. on COTS platform components through black-box testing highlights the increased complexity and challenge of fault injection in low-cost devices [70]. This underscores the difficulty of the black-box method in real-world applications. Conversely, Fiji-FIN

¹Rigidly speaking, even ECC and bit interleaving are applied to SRAMs, the output can be wrong at a very low probability due to multiple bit upsets.

Table 4.8: Accessible ranges in white- and black-box cases.

	White-Box		Black-Box	
	FF	Memory	FF	Memory
RISC-V	17176b	128KB	2620b	128KB
MatrixConv	45644b	384KB	793b	384KB
Pooling	26225b	0	607b	0
Total	89045b	512KB	4020b	512KB

[134], a comprehensive fault injection framework, is proposed for testing AI models on IoT devices, particularly effective in pinpointing vulnerable spots in QNNs. While it assesses the vulnerability of QNN accelerators to various errors, its focus remains limited to the component level without considering overall SoC reliability.

Therefore, to address the aforementioned issues, we need to conduct a comprehensive analysis comparing white-box and black-box estimations targeting edge AI SoCs. The white-box approach allows for more precise fault injection and monitoring due to its complete access to the system's internal components. By confirming the accuracy of white-box estimations, we can develop benchmarks and models to improve the reliability of black-box estimations. This comparison is crucial to ensure that despite black-box testing limitations, we can still accurately estimate the impact of soft errors on edge AI SoCs.

White- and Black-box Case Definition

Typically, COTS platforms do not grant full access to all on-chip components. Namely, only part of the components can be accessed. Therefore, it increases the difficulty of estimating the soft error impact, specifically from the system level. To better understand the difference from different standpoints, we define the following two cases for FI: 1) white-box case, all components can be accessed; 2) black-box case, only I/DTCM, CSRs, and General Purpose Registers (GPRs) in RISC-V core, and memories and CSRs in the accelerator can be accessed. In black-box cases, the internal logic cannot be accessed (e.g., FSM). Besides, all memory components and FFs do not have any error detection and correction mechanisms. The amount of accessible components of white-box case and black-box case is detailed in Table 4.8.

Results

Table 4.9 lists FI experimental results of our AI SoC. Even while operating in FlashXIP mode, the system still exhibits a high probability of generating mDUEs. Basically, our observations show that almost all mDUEs originate from bit-flips in DTCM. This is because the configuration information it contains is repeatedly invoked, making it insensitive to the timing of FI. This behavior is explained by the running RISC-V program.

Table 4.9: Event statistics in fault injection simulation.

Simulated Distance (mm)	Execution Mode	Fluence (n/cm ²)	mSDC	mDUE	cDUE
25	ITCM	8.8137e+08	19/20	45/47	33/33
50	ITCM	1.7636e+09	17/20	43/46	30/30
75	ITCM	2.8392e+09	22/25	51/52	27/27
25	FlashXIP	2.5352e+09	46/51	71/72	2/2
50	FlashXIP	2.9648e+09	28/32	44/46	2/2
75	FlashXIP	4.2891e+09	18/20	21/22	1/1

/ is black-box/white-box

The execution code for the NN model, including the functions to divide the model into multiple chunks, and any code used for data processing, NN forward, is stored in ITCM. Meanwhile, the foundational information of the NN model, including network weight size, activation function parameters, intermediate data, etc., is stored in DTCM.

Fig. 4.11 shows the origins of error components in ITCM execution mode according to the recorded log files. cDUEs are mainly caused by single-bit upsets (SBUs) in ITCM, mDUEs are mainly caused by ITCM and DTCM, and mSDCs are mainly caused by memory components in the AI accelerator. Also, the bit-flips of FFs only account for a tiny part of these error events.

Bit-flips in FFs cause only mDUEs in our fault injection experiment while, in theory, they can cause cDUEs as well. We observe that bit-flips result in mDUEs only when they occur at specific spatial and temporal points during operation. FFs typically refresh their values at each clock edge, often nullifying the impact of bit-flips. This tendency is especially notable in the FFs within data paths. Furthermore, since FFs constitute a relatively small proportion of the SoC, and a bit-flip in an FF must occur at a precise moment to trigger a cDUE, the probability of this happening is quite low. Therefore, we do not observe any cDUEs caused by bit-flips in FFs in our experiment.

Additionally, our statistical analysis reveals that most mSDCs are caused by bit-flips within the memories of the accelerator. This is partly because the accelerator memory volume is 384KB, making up 75% of the total system memory capacity and significantly exceeding the FFs in volume. Furthermore, another FI result obtained using GPUs shows that the cSDC probability for individual weight memory upsets is only 0.00492%; therefore, bit-flips within the accelerator memory predominantly lead to mSDCs. This result is consistent with the observation in [133] that the error propagation probability to misclassification is very low due to the inherent redundancy of NN computation. After a rough estimation, the cross-section of cSDCs should be less than $6.8\text{e-}10\text{cm}^2$ at 2.5mm distance, two orders of magnitude less than other error types.

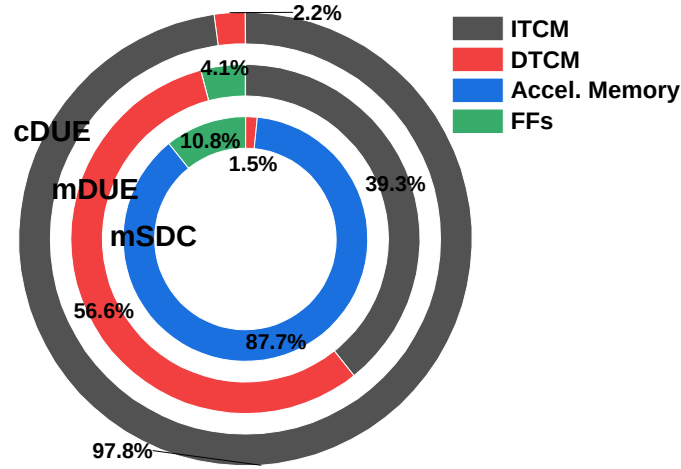


Figure 4.11: Proportions of error source components.

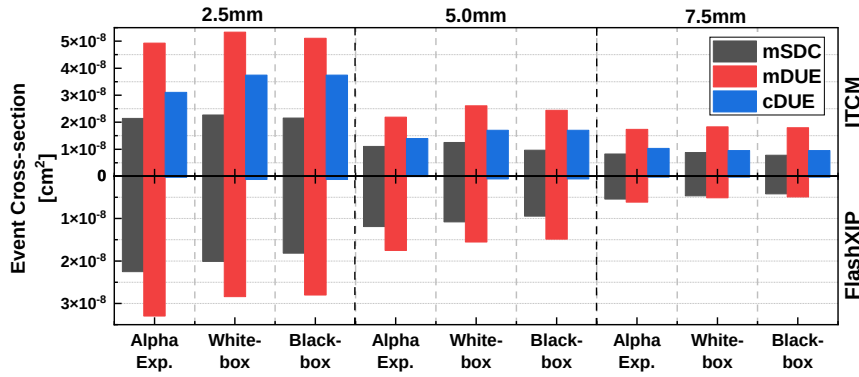


Figure 4.12: Cross-section comparison between measured, white-box, and black-box estimates.

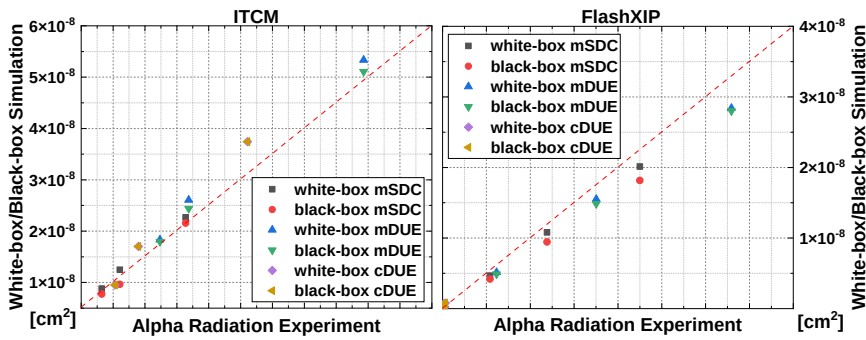


Figure 4.13: Post-silicon cross-section vs. pre-silicon cross-section.

4.5.3 Exploration Summary

Fig. 4.12 shows the event cross-section, which is the number of observed errors divided by the total alpha fluence. Both white- and black-box fault injection simulations exhibit

a strong correlation with those from the post-silicon irradiation experiments as depicted in Fig. 4.13. It can be observed that the results of fault injection in the white- and black-box cases are relatively reliable.

Both simulation and experimental findings indicate that the I/DTCM are the weakest components in the entire SoC. Furthermore, due to the fact that FFs account for only about 2% of the entire SoC, and a significant part of FFs is used for caching AI model data, coupled with the inherent high reliability of AI models, there is not much difference in the results of experiments in both white-box and black-box cases. Thus, for COTS platforms targeting AI applications, if FI into memory at any given time point is supported, the assessment of soft error impact remains quite reliable even in black-box cases. Besides, assuming the FI of memory components in the accelerator cannot be accessed, the mSDCs reproduced in the black-box FI could decrease significantly. Also, if the FI of ITCM/DTCM cannot be realized, the cDUEs cannot be observed. Meanwhile, if the memory in COTS platforms contains error detection or correction mechanisms (e.g., ECC, Parity), then the aforementioned analysis no longer applies. Additionally, we construct a special case where only errors are injected into FFs with a simulation distance of 2.5mm, and cDUEs are not observed in our limited simulation time. The rough estimate of the cDUE cross-section caused by FFs should be less than $2.17\text{e-}11\text{ cm}^2$.

This section provides a comprehensive evaluation of the reliability of a 22nm AI SoC from both pre-silicon and post-silicon perspectives. The post-silicon alpha experimental results are built as a golden reference. To analyze the difference between pre-silicon white-box and black-box views, we define two different scenarios. Additionally, all these results reveal that the ITCM/DTCM, which store the static program data, NN model information, and cache data, are the most vulnerable component in the AI SoC. Also, we discover that when using FlashXIP as the processor execution mode, the critical errors in the system are due to the non-responsiveness of AI tasks, mainly caused by the upsets in DTCM. Besides, the FI experiment on the NNs shows that the MP NN is considerably more reliable than other system components, and hence, bit-flips in accelerator memories concerning the NN data have almost no impact on the system. On the other hand, the white-box and black-box simulation results are both in line with each other. The comparison indicates that this conclusion applies only to COTS platforms without error detection or correction mechanisms, as it shows that the soft error impact on black-box AI COTS platforms can be well estimated as long as the fault injection can operate on any memory at any given time.

4.6 Discussion

In this chapter, the reliability of the proposed SoCs is comprehensively explored. In single-core SoC with MBW Booth accelerator, post-silicon FI based on UART port is used to verify that low-precision computation can increase fault criticality. Also,

Table 4.10: Soft error assessment of our SoCs.

SoC Design	Single-core w/ MBW Accel. (Section 3.3)	Single-core w/ BVC MP Accel. (Section 3.4)
Model	LeNet5	VGG16
Dataset	MNIST	ImageNet
Bit-precision	Mixed INT2/4/8	Mixed BVC3/6/8
Approach	(1) Post-silicon fault injection (2) Neutron irradiation	(1) Pre-silicon fault injection (2) Alpha irradiation
Platform	Flash-based FPGA MPF300T Eval Kit	22nm CMOS
Mitigation	HTMR & F-TMR	NA
Main Findings	(1) Low-precision increase fault criticality (2) I/DTCM in RISC-V is the most vulnerable	(1) Most soft errors in accel. memories are tolerable (2) White-/black-box cases is essentially indistinguishable w/o memory error correction

neutron irradiation is adopted to identify the vulnerable components in the SoC. Furthermore, to evaluate the reliability of SoC in the pre-silicon stage, a pre-silicon FI framework based on VCS is proposed to verify the reliability of single-core SoC with BVC MP accelerator. In the pre-silicon stage, two cases (black-box and white-box) are defined to confirm the applicability of FI. Alpha irradiation experimental results are constructed as a golden copy for reference. The experimental results in these two SoCs both showed that the reliability of memories used for activation and weight storage is quite higher than other components due to the inherent robustness of the NNs. Also, the results depicted that ITCM and DTCM are the most vulnerable components in the SoC, and have higher susceptibility to critical errors. Therefore, targeting reliability-demanding designs, the robustness of ITCM/DTCM should be prioritized in the first place. In contrast, the cost-performance of implementing error correction mechanisms into activation and weight memories is relatively low due to the robustness of NNs. In all, flexibly adopting suitable error detection or correction mechanisms onto different on-chip components on the basis of application scenarios is necessary.

4.7 Conclusion

This chapter introduced two fault injection techniques, one for pre-silicon and the other for post-silicon conditions, to assess the reliability of SoC systems in AI applications. Furthermore, through fault injection experiments and radiation tests on SoC platforms

in both FPGA and ASIC environments, the observations are as follows: 1) Soft errors in computations with lower precision tend to result in more severe errors. 2) The ITCM and DTCM in the RISC-V core are identified as the most unstable components in the entire system. 3) Due to the significantly higher proportion of memories compared to registers in AI SoC, the majority of critical errors stem from bit-flips in memories. Consequently, radiation mitigation techniques for memories become crucial. 4) the impact of soft errors in black-box edge AI SoC without internal error detection or correction mechanisms can be accurately estimated as long as the fault injection methods are capable of targeting any memory components at any given time. Besides, this chapter does not discuss the reliability analysis of multi-core SoC designs. Future work will focus on the estimation of the proposed multi-core designs, especially under the working scenario of DCLS mode.

Chapter 5

Conclusion and Future Prospects

This dissertation aims to design a high-performance and highly reliable SoC tailored for edge AI applications. It seeks to confirm the reliability of the SoC system during the execution of AI applications through reliability analysis.

5.1 Conclusion

To accelerate convolution and matrix calculations, this dissertation proposes three accelerator architectures. From the perspective of multipliers, the exploration is mainly conducted based on multi-precision computing and approximate computing. The proposed MP-Booth MAC, supporting INT2/4/8 computations, realized 43% LUT reduction compared with traditional radix-4 Booth MAC in FPGA Platforms. Moreover, the proposed radix-8 Booth-based error-free BVC MP PE, supporting BVC3/6/8 computations, achieved 82% power and 70% area reduction compared with MP-Booth PE. Besides, the proposed BVC MP MAC can be simulated on GPU platforms that only support exact computing, which means the NAS-based training results of GPU platforms can be mapped into our BVC-based platforms without any inconsistency.

From the data flow perspective, the construction of data flow is mainly approached through traditional on-chip cache structures, vector-style systolic structures, structural sparse structures, and near-memory-computing structures. Compared with classical systolic data flow, the proposed vector-style systolic data flow can realize about 50% latency optimization. Also, due to the limitation of bandwidth, a structural sparsity strategy is proposed to mitigate the transmission burden. Specifically, when the data sparsity is greater than 50%, half data can be filtered and pre-processed offline. Thus, the transmission bandwidth can be almost optimized by half. Furthermore, to achieve higher energy efficiency, the NMC-friendly data flow is proposed with the combination of vector-style systolic data flow. The PE is directly connected to the weight memory, realizing a peak energy efficiency of 33.98 TOPS/W in BVC3 precision.

Additionally, based on the three accelerator architectures mentioned above, this dis-

sertation introduces three SoC designs that can be mainly categorized into two architectures: single-core and multi-core. Since the accelerator itself cannot implement general computing operators, some computing operators need to be performed in a single core. Specifically, a 22nm edge AI SoC with BVC MP accelerator is designed with 4.96-to-33.98 TOPS/W peak energy efficiency. However, the RISC-V core can only realize 1.123TOP/W peak energy efficiency. Therefore, in complex AI workloads, many computing operators are in the core leading to inefficiency. To address this problem, a multi-core multi-mode programmable parallel architecture can effectively address this issue, featuring 1) dual-core-lock-step configuration for safety-critical applications, 2) accelerator mode for convolutional and matrix multiplication acceleration, 3) multi-core mode for the computation of versatile operators, and 4) fully-access mode for debugging and fault injection. Also, this design can realize the peak energy efficiency of 5.44 TOPS/W and 6.463 TOPS/W at INT8 computation in the multi-core mode and the accelerator mode, respectively.

Furthermore, to explore the reliability of the SoC system, this dissertation proposes two fault injection frameworks, namely pre-silicon and post-silicon, for reliability analysis. Additionally, neutron radiation experiments and alpha particle radiation experiments are conducted on physical flash-based FPGA and 22nm ASIC platforms to explore the reliability of the SoC system. The experimental result shows that ITCM and DTCM are the most vulnerable components. Besides, in pre-silicon simulation and post-silicon experiment, the results show most soft errors in the accelerator are negligible. The reliability improvement of SoC should focus on the enhancement of the RISC-V core, especially the cache. Also, the estimation of soft errors in black-box COTS platforms is reliable when these platforms do not incorporate error mitigation or correction techniques.

5.2 Key Contributions

The key contributions of this dissertation are as follows: 1) Introduced AI accelerators supporting multi-precision computing, approximate computing, structural sparsity, vector-style data flow, near-memory computing, and efficient NN mapping. 2) Proposed a multi-core multi-mode SoC system with high logic reuse targeting versatile edge AI applications. 3) Presented an efficient pre-silicon and post-silicon fault injection universal platform. 4) Conducted a comprehensive analysis of the reliability of the proposed SoCs in edge AI applications via simulation and radiation experiments.

5.3 Future Work and Prospects

The current study has successfully completed the design of both single-core and multi-core SoCs, achieving high-performance computing. Additionally, a comprehensive re-

liability analysis of the single-core SoC in bare-metal AI programs has been conducted. However, exploration of the multi-core SoC and reliability analysis of AI programs require further investigation. Furthermore, considering the prevalent use of multi-core configurations in commercial designs often running operating systems, short-term future work will predominantly focus on researching the reliability of multi-core SoCs running AI applications under bare-metal programs or operating systems at the edge.

In addition, the SoC designs in this dissertation have not addressed the integration of communication and sensing modules. The reliability analyses conducted so far assume the correctness of off-chip data. However, in common edge AI SoC systems, the communication module is crucial for interacting with off-chip AI model data, while the sensing module is essential for collecting source data for processing. Therefore, the reliability of these modules directly impacts the overall reliability of AI applications. Consequently, long-term future work will involve designing efficient AI SoCs that integrate edge sensing, communication, and computing, and analyzing their reliability in radiation environments.

Bibliography

- [1] G. E. Hinton, S. Osindero, and Y.-W. Teh, “A fast learning algorithm for deep belief nets,” *Neural computation*, vol. 18, no. 7, pp. 1527–1554, 2006.
- [2] G. E. Hinton and R. R. Salakhutdinov, “Reducing the dimensionality of data with neural networks,” *science*, vol. 313, no. 5786, pp. 504–507, 2006.
- [3] Z. Lai, Z. Jin, and J. Yang, “Global sparse representation projections for feature extraction and classification,” in *2009 Chinese Conference on Pattern Recognition*. IEEE, 2009, pp. 1–5.
- [4] Z.-W. Yuan and J. Zhang, “Feature extraction and image retrieval based on alexnet,” in *Eighth International Conference on Digital Image Processing (ICDIP 2016)*, vol. 10033. SPIE, 2016, pp. 65–69.
- [5] S. Akbar, M. Peikari, S. Salama, S. Nofech-Mozes, and A. Martel, “Transitioning between convolutional and fully connected layers in neural networks,” in *Deep Learning in Medical Image Analysis and Multimodal Learning for Clinical Decision Support: Third International Workshop, DLMIA 2017, and 7th International Workshop, ML-CDS 2017, Held in Conjunction with MICCAI 2017, Québec City, QC, Canada, September 14, Proceedings 3*. Springer, 2017, pp. 143–150.
- [6] H. Ke, D. Chen, X. Li, Y. Tang, T. Shah, and R. Ranjan, “Towards brain big data classification: Epileptic eeg identification with a lightweight vggnet on global mic,” *IEEE Access*, vol. 6, pp. 14 722–14 733, 2018.
- [7] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. Alemi, “Inception-v4, inception-resnet and the impact of residual connections on learning,” in *Proceedings of the AAAI conference on artificial intelligence*, vol. 31, no. 1, 2017.
- [8] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, “Densely connected convolutional networks,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 4700–4708.
- [9] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly *et al.*, “An image is

- worth 16x16 words: Transformers for image recognition at scale,” *arXiv preprint arXiv:2010.11929*, 2020.
- [10] V. Camus, L. Mei, C. Enz, and M. Verhelst, “Review and benchmarking of precision-scalable multiply-accumulate unit architectures for embedded neural-network processing,” *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 9, no. 4, pp. 697–711, 2019.
- [11] R. Gong, X. Liu, S. Jiang, T. Li, P. Hu, J. Lin, F. Yu, and J. Yan, “Differentiable soft quantization: Bridging full-precision and low-bit neural networks,” in *Proceedings of the IEEE/CVF international conference on computer vision*, 2019, pp. 4852–4861.
- [12] R. Zhao, W. Song, W. Zhang, T. Xing, J.-H. Lin, M. Srivastava, R. Gupta, and Z. Zhang, “Accelerating binarized convolutional neural networks with software-programmable fpgas,” in *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, 2017, pp. 15–24.
- [13] M. Huang, Y. Liu, C. Man, K. Li, Q. Cheng, W. Mao, and H. Yu, “A high performance multi-bit-width booth vector systolic accelerator for nas optimized deep learning neural networks,” *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 69, no. 9, pp. 3619–3631, 2022.
- [14] Y. Zhao, X. Gao, X. Guo, J. Liu, E. Wang, R. Mullins, P. Y. Cheung, G. Constantinides, and C.-Z. Xu, “Automatic generation of multi-precision multi-arithmetic cnn accelerators for fpgas,” in *2019 International Conference on Field-Programmable Technology (ICFPT)*. IEEE, 2019, pp. 45–53.
- [15] J.-S. Park, C. Park, S. Kwon, T. Jeon, Y. Kang, H. Lee, D. Lee, J. Kim, H.-S. Kim, Y. Lee *et al.*, “A multi-mode 8k-mac hw-utilization-aware neural processing unit with a unified multi-precision datapath in 4-nm flagship mobile soc,” *IEEE Journal of Solid-State Circuits*, vol. 58, no. 1, pp. 189–202, 2022.
- [16] H. Waris, C. Wang, W. Liu, and F. Lombardi, “Axbms: Approximate radix-8 booth multipliers for high-performance fpga-based accelerators,” *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 68, no. 5, pp. 1566–1570, 2021.
- [17] L. Sayadi, S. Timarchi, and A. Sheikh-Akbari, “Two efficient approximate unsigned multipliers by developing new configuration for approximate 4: 2 compressors,” *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 70, no. 4, pp. 1649–1659, 2023.

- [18] I. Hammad, L. Li, K. El-Sankary, and W. M. Snelgrove, "Cnn inference using a preprocessing precision controller and approximate multipliers with various precisions," *IEEE Access*, vol. 9, pp. 7220–7232, 2021.
- [19] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers *et al.*, "In-datacenter performance analysis of a tensor processing unit," in *Proceedings of the 44th annual international symposium on computer architecture*, 2017, pp. 1–12.
- [20] H. T. Kung and C. E. Leiserson, "Systolic arrays (for vlsi)," in *Sparse Matrix Proceedings 1978*, vol. 1. Society for industrial and applied mathematics Philadelphia, PA, USA, 1979, pp. 256–282.
- [21] Y. Jia, *Learning semantic image representations at a large scale*. University of California, Berkeley, 2014.
- [22] H.-T. Kung, "Why systolic architectures?" *Computer*, vol. 15, no. 1, pp. 37–46, 1982.
- [23] H. Lim, V. Piuri, and E. E. Swartzlander, "A serial-parallel architecture for two-dimensional discrete cosine and inverse discrete cosine transforms," *IEEE Transactions on Computers*, vol. 49, no. 12, pp. 1297–1309, 2000.
- [24] H. Kung, B. McDanel, and S. Q. Zhang, "Packing sparse convolutional neural networks for efficient systolic array implementations: Column combining under joint optimization," in *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, 2019, pp. 821–834.
- [25] Y. Chen, T. Luo, S. Liu, S. Zhang, L. He, J. Wang, L. Li, T. Chen, Z. Xu, N. Sun *et al.*, "Dadiannao: A machine-learning supercomputer," in *2014 47th Annual IEEE/ACM International Symposium on Microarchitecture*. IEEE, 2014, pp. 609–622.
- [26] J. Yue, Y. Liu, Z. Yuan, X. Feng, Y. He, W. Sun, Z. Zhang, X. Si, R. Liu, Z. Wang *et al.*, "Sticker-im: A 65 nm computing-in-memory nn processor using block-wise sparsity optimization and inter/intra-macro data reuse," *IEEE Journal of Solid-State Circuits*, vol. 57, no. 8, pp. 2560–2573, 2022.
- [27] C.-Y. Du, C.-F. Tsai, W.-C. Chen, L.-Y. Lin, N.-S. Chang, C.-P. Lin, C.-S. Chen, and C.-H. Yang, "A 28nm 11.2 tops/w hardware-utilization-aware neural-network accelerator with dynamic dataflow," in *2023 IEEE International Solid-State Circuits Conference (ISSCC)*. IEEE, 2023, pp. 1–3.

- [28] F. Conti, D. Rossi, G. Paulin, A. Garofalo, A. Di Mauro, G. Rutishauer, G. marco Ottavi, M. Eggimann, H. Okuhara, V. Huard *et al.*, “22.1 a 12.4 tops/w@ 136gops ai-iot system-on-chip with 16 risc-v, 2-to-8b precision-scalable dnn acceleration and 30%-boost adaptive body biasing,” in *2023 IEEE International Solid-State Circuits Conference (ISSCC)*. IEEE, 2023, pp. 21–23.
- [29] M. Wirthlin, “High-reliability fpga-based systems: Space, high-energy physics, and beyond,” *Proceedings of the IEEE*, vol. 103, no. 3, pp. 379–389, 2015.
- [30] L. R. Rockett and D. J. Kouba, “Radiation hardened 150nm standard cell asic design library for space applications,” in *2008 IEEE Aerospace Conference*. IEEE, 2008, pp. 1–8.
- [31] W. Mo, J. Ye, H. Liu, X. Xiao, W. Fan, Y. Song, W. Huang, T. Wang, Q. Zhang, D. Zhang *et al.*, “A novel sgt mosfet based on radiation-hardening technology and its total-ionizing-dose radiation effects,” *IEEE Electron Device Letters*, 2024.
- [32] Z. He, A. Shen, Q. Li, Q. Cheng, and H. Yu, “Agile hardware and software co-design for risc-v-based multi-precision deep learning microprocessor,” in *Proceedings of the 28th Asia and South Pacific Design Automation Conference*, 2023, pp. 490–495.
- [33] P. D. Schiavone, D. Rossi, A. Pullini, A. Di Mauro, F. Conti, and L. Benini, “Quentin: an ultra-low-power pulpissimo soc in 22nm fdx,” in *2018 IEEE SOI-3D-Subthreshold Microelectronics Technology Unified Conference (S3S)*. IEEE, 2018, pp. 1–3.
- [34] A. Pullini, D. Rossi, I. Loi, G. Tagliavini, and L. Benini, “Mr. wolf: An energy-precision scalable parallel ultra low power soc for iot edge processing,” *IEEE Journal of Solid-State Circuits*, vol. 54, no. 7, pp. 1970–1981, 2019.
- [35] E. Flamand, D. Rossi, F. Conti, I. Loi, A. Pullini, F. Rotenberg, and L. Benini, “Gap-8: A risc-v soc for ai at the edge of the iot,” in *2018 IEEE 29th International Conference on Application-specific Systems, Architectures and Processors (ASAP)*. IEEE, 2018, pp. 1–4.
- [36] P. E. Dodd and L. W. Massengill, “Basic mechanisms and modeling of single-event upset in digital microelectronics,” *IEEE Transactions on nuclear Science*, vol. 50, no. 3, pp. 583–602, 2003.
- [37] D. Munteanu and J.-L. Autran, “Modeling and simulation of single-event effects in digital devices and ics,” *IEEE Transactions on Nuclear science*, vol. 55, no. 4, pp. 1854–1878, 2008.

- [38] J. R. Schwank, M. R. Shaneyfelt, and P. E. Dodd, "Radiation hardness assurance testing of microelectronic devices and integrated circuits: Radiation environments, physical mechanisms, and foundations for hardness assurance," *IEEE Transactions on Nuclear Science*, vol. 60, no. 3, pp. 2074–2100, 2013.
- [39] J. L. Barth, C. Dyer, and E. Stassinopoulos, "Space, atmospheric, and terrestrial radiation environments," *IEEE Transactions on nuclear science*, vol. 50, no. 3, pp. 466–482, 2003.
- [40] R. C. Baumann, "Radiation-induced soft errors in advanced semiconductor technologies," *IEEE Transactions on Device and materials reliability*, vol. 5, no. 3, pp. 305–316, 2005.
- [41] D. F. Heidel, P. W. Marshall, K. A. LaBel, J. R. Schwank, K. P. Rodbell, M. C. Hakey, M. D. Berg, P. E. Dodd, M. R. Friendlich, A. D. Phan *et al.*, "Low energy proton single-event-upset test results on 65 nm soi sram," *IEEE Transactions on nuclear Science*, vol. 55, no. 6, pp. 3394–3400, 2008.
- [42] E. Cannon, M. Cabanas-Holmen, J. Wert, T. Amort, R. Brees, J. Koehn, B. Meaker, and E. Normand, "Heavy ion, high-energy, and low-energy proton see sensitivity of 90-nm rhbd srms," *IEEE Transactions on Nuclear Science*, vol. 57, no. 6, pp. 3493–3499, 2010.
- [43] D. Binder, E. C. Smith, and A. Holman, "Satellite anomalies from galactic cosmic rays," *IEEE Transactions on Nuclear Science*, vol. 22, no. 6, pp. 2675–2680, 1975.
- [44] R. Velazco, R. Ecoffet, and F. Faure, "How to characterize the problem of seu in processors & representative errors observed on flight," in *11th IEEE International On-Line Testing Symposium*. IEEE, 2005, pp. 303–308.
- [45] P. M. Basso, F. F. dos Santos, and P. Rech, "Impact of tensor cores and mixed precision on the reliability of matrix multiplication in gpus," *IEEE Transactions on Nuclear Science*, vol. 67, no. 7, pp. 1560–1565, 2020.
- [46] A. Lotfi, S. Hukerikar, K. Balasubramanian, P. Racunas, N. Saxena, R. Bramley, and Y. Huang, "Resiliency of automotive object detection networks on gpu architectures," in *2019 IEEE International Test Conference (ITC)*. IEEE, 2019, pp. 1–9.
- [47] T. Tanaka, W. Liao, M. Hashimoto, and Y. Mitsuyama, "Impact of neutron-induced seu in fpga cram on image-based lane tracking for autonomous driving: From bit upset to sefi and erroneous behavior," *IEEE Transactions on Nuclear Science*, vol. 69, no. 1, pp. 35–42, 2021.

- [48] F. W. Sexton, "Destructive single-event effects in semiconductor devices and ics," *IEEE Transactions on Nuclear Science*, vol. 50, no. 3, pp. 603–621, 2003.
- [49] F. Irom, "Guideline for ground radiation testing of microprocessors in the space radiation environment," Citeseer, Tech. Rep., 2008.
- [50] J. R. Marshall and J. Robertson, "An embedded microcontroller for spacecraft applications," in *2006 IEEE Aerospace Conference*. IEEE, 2006, pp. 9–pp.
- [51] R. Berger, L. Burcin, D. Hutcheson, J. Koehler, M. Lassa, M. Milliser, D. Moser, D. Stanley, R. Zeger, B. Blalock *et al.*, "The rad6000mc system-on-chip microcontroller for spacecraft avionics and instrument control," in *2008 IEEE Aerospace Conference*. IEEE, 2008, pp. 1–14.
- [52] T. Gallagher, S. H. Weiss, and J. Hahn, "Natural feature tracking on the opera maestro platform," in *2011 Aerospace Conference*. IEEE, 2011, pp. 1–7.
- [53] G. W. Caldwell, "Implementation of configurable fault tolerant processor (cftp) experiments," Ph.D. dissertation, Monterey California. Naval Postgraduate School, 2006.
- [54] D. A. Ebert, "Design and development of a configurable fault-tolerant processor (cftp) for space applications," Ph.D. dissertation, Monterey, California. Naval Postgraduate School, 2003.
- [55] R. W. Berger, D. Bayles, R. Brown, S. Doyle, A. Kazemzadeh, K. Knowles, D. Moser, J. Rodgers, B. Saari, D. Stanley *et al.*, "The rad750/sup tm/-a radiation hardened powerpc/sup tm/processor for high performance spaceborne applications," in *2001 IEEE Aerospace Conference Proceedings (Cat. No. 01TH8542)*, vol. 5. IEEE, 2001, pp. 2263–2272.
- [56] N. F. Haddad, R. D. Brown, R. Ferguson, A. T. Kelly, R. K. Lawrence, D. M. Pirkel, and J. C. Rodgers, "Second generation (200mhz) rad750 microprocessor radiation evaluation," in *2011 12th European Conference on Radiation and Its Effects on Components and Systems*. IEEE, 2011, pp. 877–880.
- [57] S. M. Guertin, C. Hafer, and S. Griffith, "Investigation of low cross section events in the rhbd/ft ut699 leon 3ft," in *2011 IEEE Radiation Effects Data Workshop*. IEEE, 2011, pp. 1–8.
- [58] S. M. Guertin and M. Amrbar, "See test results for p2020 and p5020 freescale processors," in *2014 IEEE Radiation Effects Data Workshop (REDW)*. IEEE, 2014, pp. 1–7.

- [59] B. Wie, M. Plante, A. Berkley, S. Guertin, and R. Nejad, "Static, dynamic, and application-level see results for a 49-core rhbd processor," in *2013 IEEE Radiation Effects Data Workshop (REDW)*. IEEE, 2013, pp. 1–9.
- [60] J. Andersson, J. Gaisler, and R. Weigand, "Next generation multipurpose micro-processor," in *Int. Conf. on Data Systems in Aerospace (DASIA)*, Hungary, 2010.
- [61] M. Hijorth, M. Aberg, N.-J. Wessman, J. Andersson, R. Chevallier, R. Forsyth, R. Weigand, and L. Fossati, "Gr740: Rad-hard quad-core leon4ft system-on-chip," *DASIA 2015-DATA Systems in Aerospace*, vol. 732, p. 7, 2015.
- [62] S. M. Guertin, "System on a chip devices—fy10," Pasadena, CA: Jet Propulsion Laboratory, National Aeronautics and Space . . . , Tech. Rep., 2010.
- [63] J. Gaisler, "Leon3-ft-rtax seu test results," *Gaisler Research—2005–8p*, 2005.
- [64] R. P. Perumal, H. Voos, F. D. Vedova, and H. Moser, "Small satellite reliability: A decade in review," 2021.
- [65] R. P. Bastos, M. G. Trindade, R. Garibotti, J. Gava, R. Reis, and L. Ost, "Assessment of tiny machine-learning computing systems under neutron-induced radiation effects," *IEEE Transactions on Nuclear Science*, vol. 69, no. 7, pp. 1683–1690, 2022.
- [66] F. F. Dos Santos, A. Kritikakou, and O. Sentieys, "Experimental evaluation of neutron-induced errors on a multicore risc-v platform," in *2022 IEEE 28th International Symposium on On-Line Testing and Robust System Design (IOLTS)*. IEEE, 2022, pp. 1–7.
- [67] S. K. S. Hari, M. B. Sullivan, T. Tsai, and S. W. Keckler, "Making convolutions resilient via algorithm-based error detection techniques," *IEEE Transactions on Dependable and Secure Computing*, vol. 19, no. 4, pp. 2546–2558, 2021.
- [68] F. Libano, P. Rech, and J. Brunhaver, "Efficient error detection for matrix multiplication with systolic arrays on fpgas," *IEEE Transactions on Computers*, 2023.
- [69] C. Bolchini, L. Cassano, A. Miele, and M. Biasielli, "Lightweight fault detection and management for image restoration," in *2020 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)*. IEEE, 2020, pp. 1–6.
- [70] J. Balasch, B. Gierlichs, and I. Verbauwhede, "An in-depth and black-box characterization of the effects of clock glitches on 8-bit mcus," in *2011 Workshop on Fault Diagnosis and Tolerance in Cryptography*. IEEE, 2011, pp. 105–114.

- [71] F. Farshchi, Q. Huang, and H. Yun, "Integrating nvidia deep learning accelerator (nvdla) with risc-v soc on firesim," in *2019 2nd Workshop on Energy Efficient Machine Learning and Cognitive Computing for Embedded Applications (EMC2)*. IEEE, 2019, pp. 21–25.
- [72] J. J. Martinez, J. Garrigos, I. Villó, J. Toledo, and J. M. Ferrandez, "Hardware acceleration on hprc of a cnn-based algorithm for astronomical images reduction," in *2010 12th International Workshop on Cellular Nanoscale Networks and their Applications (CNNA 2010)*. IEEE, 2010, pp. 1–5.
- [73] V. Gokhale, J. Jin, A. Dundar, B. Martini, and E. Culurciello, "A 240 g-ops/s mobile coprocessor for deep neural networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, 2014, pp. 682–687.
- [74] C. Zhang, P. Li, G. Sun, Y. Guan, B. Xiao, and J. Cong, "Optimizing fpga-based accelerator design for deep convolutional neural networks," in *Proceedings of the 2015 ACM/SIGDA international symposium on field-programmable gate arrays*, 2015, pp. 161–170.
- [75] H. Nakahara and T. Sasao, "A deep convolutional neural network based on nested residue number system," in *2015 25th International Conference on Field Programmable Logic and Applications (FPL)*. IEEE, 2015, pp. 1–6.
- [76] X. Han, D. Zhou, S. Wang, and S. Kimura, "Cnn-merp: An fpga-based memory-efficient reconfigurable processor for forward and backward propagation of convolutional neural networks," in *2016 IEEE 34th International Conference on Computer Design (ICCD)*. IEEE, 2016, pp. 320–327.
- [77] R. Andri, L. Cavigelli, D. Rossi, and L. Benini, "Yodann: An ultra-low power convolutional neural network accelerator based on binary weights," in *2016 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*. IEEE, 2016, pp. 236–241.
- [78] I. Chakraborty, D. Roy, I. Garg, A. Ankit, and K. Roy, "Constructing energy-efficient mixed-precision neural networks through principal component analysis for edge intelligence," *Nature Machine Intelligence*, vol. 2, no. 1, pp. 43–55, 2020.
- [79] C. Gong, Z. Jiang, D. Wang, Y. Lin, Q. Liu, and D. Z. Pan, "Mixed precision neural architecture search for energy efficient deep learning," in *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 2019, pp. 1–7.

- [80] C. Wu, M. Wang, X. Chu, K. Wang, and L. He, "Low-precision floating-point arithmetic for high-performance fpga-based cnn acceleration," *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, vol. 15, no. 1, pp. 1–21, 2021.
- [81] B. Asgari, R. Hadidi, and H. Kim, "Meissa: Multiplying matrices efficiently in a scalable systolic architecture," in *2020 IEEE 38th International Conference on Computer Design (ICCD)*. IEEE, 2020, pp. 130–137.
- [82] P. Gysel, J. Pimentel, M. Motamedi, and S. Ghiasi, "Ristretto: A framework for empirical study of resource-efficient inference in convolutional neural networks," *IEEE transactions on neural networks and learning systems*, vol. 29, no. 11, pp. 5784–5789, 2018.
- [83] B. Wu, X. Dai, P. Zhang, Y. Wang, F. Sun, Y. Wu, Y. Tian, P. Vajda, Y. Jia, and K. Keutzer, "Fbnet: Hardware-aware efficient convnet design via differentiable neural architecture search," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2019, pp. 10 734–10 742.
- [84] B. Moons and M. Verhelst, "An energy-efficient precision-scalable convnet processor in 40-nm cmos," *IEEE Journal of solid-state Circuits*, vol. 52, no. 4, pp. 903–914, 2016.
- [85] S. Han, X. Liu, H. Mao, J. Pu, A. Pedram, M. A. Horowitz, and W. J. Dally, "Eie: Efficient inference engine on compressed deep neural network," *ACM SIGARCH Computer Architecture News*, vol. 44, no. 3, pp. 243–254, 2016.
- [86] X. Cui, W. Liu, X. Chen, E. E. Swartzlander, and F. Lombardi, "A modified partial product generator for redundant binary multipliers," *IEEE transactions on computers*, vol. 65, no. 4, pp. 1165–1171, 2015.
- [87] Z.-G. Liu, P. N. Whatmough, and M. Mattina, "Sparse systolic tensor array for efficient cnn hardware acceleration," *arXiv preprint arXiv:2009.02381*, 2020.
- [88] F. U. D. Farrukh, C. Zhang, Y. Jiang, Z. Zhang, Z. Wang, Z. Wang, and H. Jiang, "Power efficient tiny yolo cnn using reduced hardware resources based on booth multiplier and wallace tree adders," *IEEE Open Journal of Circuits and Systems*, vol. 1, pp. 76–87, 2020.
- [89] Y.-J. Chang, Y.-C. Cheng, S.-C. Liao, and C.-H. Hsiao, "A low power radix-4 booth multiplier with pre-encoded mechanism," *IEEE access*, vol. 8, pp. 114 842–114 853, 2020.
- [90] M. D. Ercegovac and T. Lang, *Digital arithmetic*. Elsevier, 2004.

- [91] Y.-H. Chen, T.-J. Yang, J. Emer, and V. Sze, "Eyeriss v2: A flexible accelerator for emerging deep neural networks on mobile devices," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 9, no. 2, pp. 292–308, 2019.
- [92] G. Shomron, T. Horowitz, and U. Weiser, "Smt-sa: Simultaneous multithreading in systolic arrays," *IEEE Computer Architecture Letters*, vol. 18, no. 2, pp. 99–102, 2019.
- [93] K. Matsubara, L. Hanno, M. Kimura, A. Nakamura, M. Koike, K. Terashima, S. Morikawa, Y. Hotta, T. Irita, S. Mochizuki *et al.*, "A 12nm autonomous driving processor running 60.4 tops and 13.8 tops/w cnns with task-separated asil d control," *IEICE Technical Report; IEICE Tech. Rep.*, vol. 121, no. 139, pp. 48–53, 2021.
- [94] H.-J. Kang, "Accelerator-aware pruning for convolutional neural networks," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 30, no. 7, pp. 2093–2103, 2019.
- [95] Z. Yuan, Y. Liu, J. Yue, Y. Yang, J. Wang, X. Feng, J. Zhao, X. Li, and H. Yang, "Sticker: An energy-efficient multi-sparsity compatible accelerator for convolutional neural networks in 65-nm cmos," *IEEE Journal of Solid-State Circuits*, vol. 55, no. 2, pp. 465–477, 2019.
- [96] N. H. Weste and D. Harris, *CMOS VLSI design: a circuits and systems perspective*. Pearson Education India, 2015.
- [97] W. Liu, J. Lin, and Z. Wang, "A precision-scalable energy-efficient convolutional neural network accelerator," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 67, no. 10, pp. 3484–3497, 2020.
- [98] B. Wu, Y. Wang, P. Zhang, Y. Tian, P. Vajda, and K. Keutzer, "Mixed precision quantization of convnets via differentiable neural architecture search," *arXiv preprint arXiv:1812.00090*, 2018.
- [99] Y. Li, S. Gu, C. Mayer, L. V. Gool, and R. Timofte, "Group sparsity: The hinge between filter pruning and decomposition for network compression," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2020, pp. 8018–8027.
- [100] Z. Guo, X. Zhang, H. Mu, W. Heng, Z. Liu, Y. Wei, and J. Sun, "Single path one-shot neural architecture search with uniform sampling," in *Computer Vision—ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XVI 16*. Springer, 2020, pp. 544–560.

- [101] T. Veniat and L. Denoyer, "Learning time/memory-efficient deep architectures with budgeted super networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 3492–3500.
- [102] J. Yuan, Q. Zhao, W. Wang, X. Meng, J. Li, and Q. Li, "Audio denoising co-processor based on risc-v custom instruction set extension," in *Acoustics*, vol. 4, no. 3. MDPI, 2022, pp. 538–553.
- [103] D. Rath, "Open on-chip debugger," URL: <http://openocd.org/>. [Online], 2008.
- [104] S. Ryu, H. Kim, W. Yi, E. Kim, Y. Kim, T. Kim, and J.-J. Kim, "Bitblade: Energy-efficient variable bit-precision hardware accelerator for quantized neural networks," *IEEE Journal of Solid-State Circuits*, vol. 57, no. 6, pp. 1924–1935, 2022.
- [105] F. Tu, Z. Wu, Y. Wang, L. Liang, L. Liu, Y. Ding, L. Liu, S. Wei, Y. Xie, and S. Yin, "A 28nm 15.59 μ j/token full-digital bitline-transpose cim-based sparse transformer accelerator with pipeline/parallel reconfigurable modes," in *2022 IEEE International Solid-State Circuits Conference (ISSCC)*, vol. 65. IEEE, 2022, pp. 466–468.
- [106] Y. Wang, Y. Qin, D. Deng, J. Wei, Y. Zhou, Y. Fan, T. Chen, H. Sun, L. Liu, S. Wei *et al.*, "An energy-efficient transformer processor exploiting dynamic weak relevances in global attention," *IEEE Journal of Solid-State Circuits*, vol. 58, no. 1, pp. 227–242, 2022.
- [107] Y. Tamir and H.-C. Chi, "Symmetric crossbar arbiters for vlsi communication switches," *IEEE Transactions on Parallel and Distributed Systems*, vol. 4, no. 1, pp. 13–27, 1993.
- [108] K. Z. Z. Z. S. Z. S. H. Y. X. P. D. M. V. K. Guo, W. Li and Y. Wang, "Neural network accelerator comparison," URL: <https://nicsefc.ee.tsinghua.edu.cn/projects/neural-network-accelerator/>. [Online].
- [109] T. Yu, B. Wu, K. Chen, C. Yan, and W. Liu, "Energy-efficient oriented approximate quantization scheme for fine-grained sparse neural network acceleration," in *2022 IEEE 40th International Conference on Computer Design (ICCD)*. IEEE, 2022, pp. 762–769.
- [110] W. Cheng, C. Lin, and Y.-Y. Shih, "An efficient implementation of convolutional neural network with clip-q quantization on fpga," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 69, no. 10, pp. 4093–4102, 2022.

- [111] B. Lu, J. Yang, and S. Ren, "Poster: Scaling up deep neural network optimization for edge inference," in *2020 IEEE/ACM Symposium on Edge Computing (SEC)*. IEEE, 2020, pp. 170–172.
- [112] H. Alemdar, V. Leroy, A. Prost-Boucle, and F. Pétrot, "Ternary neural networks for resource-efficient ai applications," in *2017 international joint conference on neural networks (IJCNN)*. IEEE, 2017, pp. 2547–2554.
- [113] H. Cai, L. Zhu, and S. Han, "Proxylesnas: Direct neural architecture search on target task and hardware," *arXiv preprint arXiv:1812.00332*, 2018.
- [114] F. Libano, P. Rech, B. Neuman, J. Leavitt, M. Wirthlin, and J. Brunhaver, "How reduced data precision and degree of parallelism impact the reliability of convolutional neural networks on fpgas," *IEEE Transactions on Nuclear Science*, vol. 68, no. 5, pp. 865–872, 2021.
- [115] F. Libano, B. Wilson, M. Wirthlin, P. Rech, and J. Brunhaver, "Understanding the impact of quantization, accuracy, and radiation on the reliability of convolutional neural networks on fpgas," *IEEE Transactions on Nuclear Science*, vol. 67, no. 7, pp. 1478–1484, 2020.
- [116] M. Kooli and G. Di Natale, "A survey on simulation-based fault injection tools for complex systems," in *2014 9th IEEE International Conference on Design & Technology of Integrated Systems in Nanoscale Era (DTIS)*. IEEE, 2014, pp. 1–6.
- [117] M. Ebrahimi, A. Mohammadi, A. Ejlali, and S. G. Miremadi, "A fast, flexible, and easy-to-develop fpga-based fault injection technique," *Microelectronics Reliability*, vol. 54, no. 5, pp. 1000–1008, 2014.
- [118] J. Carreira, H. Madeira, and J. G. Silva, "Xception: A technique for the experimental evaluation of dependability in modern computers," *IEEE Transactions on Software Engineering*, vol. 24, no. 2, pp. 125–136, 1998.
- [119] X. Xu, Y. Fu, T. Liu, and H. You, "Fault tolerance research of visual convolutional neural networks based on soft errors," in *2020 IEEE 4th Information Technology, Networking, Electronic and Automation Control Conference (ITNEC)*, vol. 1. IEEE, 2020, pp. 566–570.
- [120] A. Aponte-Moreno, F. Restrepo-Calle, and C. Pedraza, "Reliability evaluation of risc-v and arm microprocessors through a new fault injection tool," in *2021 IEEE 22nd Latin American Test Symposium (LATS)*. IEEE, 2021, pp. 1–6.
- [121] D. Paiva, J. M. Duarte, R. Lima, M. Carvalho, F. Mattiello-Francisco, and H. Madeira, "Fault injection platform for affordable verification and validation

- of cubesats software,” in *2021 10th Latin-American Symposium on Dependable Computing (LADC)*. IEEE, 2021, pp. 1–11.
- [122] D. Mamone, A. Bosio, A. Savino, S. Hamdioui, and M. Rebaudengo, “On the analysis of real-time operating system reliability in embedded systems,” in *2020 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)*. IEEE, 2020, pp. 1–6.
- [123] A. E. Wilson, S. Larsen, C. Wilson, C. Thurlow, and M. Wirthlin, “Neutron radiation testing of a tmr vexriscv soft processor on sram-based fpgas,” *IEEE Transactions on Nuclear Science*, vol. 68, no. 5, pp. 1054–1060, 2021.
- [124] G. A. Reis, J. Chang, and D. I. August, “Automatic instruction-level software-only recovery,” *IEEE micro*, vol. 27, no. 1, pp. 36–47, 2007.
- [125] J. Gava, R. Reis, and L. Ost, “Rat: A lightweight system-level soft error mitigation technique,” in *2020 IFIP/IEEE 28th International Conference on Very Large Scale Integration (VLSI-SOC)*. IEEE, 2020, pp. 165–170.
- [126] G. Abich, J. Gava, R. Garibotti, R. Reis, and L. Ost, “Applying lightweight soft error mitigation techniques to embedded mixed precision deep neural networks,” *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 68, no. 11, pp. 4772–4782, 2021.
- [127] D. A. Santos, L. M. Luza, C. A. Zeferino, L. Dilillo, and D. R. Melo, “A low-cost fault-tolerant risc-v processor for space systems,” in *2020 15th Design & Technology of Integrated Systems in Nanoscale Era (DTIS)*. IEEE, 2020, pp. 1–5.
- [128] H. Cho, “Impact of microarchitectural differences of risc-v processor cores on soft error effects,” *IEEE Access*, vol. 6, pp. 41 302–41 313, 2018.
- [129] Y. Sakemi, M. Itoh, T. Wakui *et al.*, “High intensity fast neutron beam facility at cyric,” 2014.
- [130] F. Libano, B. Wilson, J. Anderson, M. J. Wirthlin, C. Cazzaniga, C. Frost, and P. Rech, “Selective hardening for neural networks in fpgas,” *IEEE Transactions on Nuclear Science*, vol. 66, no. 1, pp. 216–222, 2018.
- [131] S. Blower, P. Rech, C. Cazzaniga, M. Kastriotou, and C. D. Frost, “Evaluating and mitigating neutrons effects on cots edgeai accelerators,” *IEEE Transactions on Nuclear Science*, vol. 68, no. 8, pp. 1719–1726, 2021.
- [132] J. Gava, A. Hanneman, G. Abich, R. Garibotti, S. Cuenca-Asensi, R. P. Bastos, R. Reis, and L. Ost, “A lightweight mitigation technique for resource-constrained

- devices executing dnn inference models under neutron radiation,” *IEEE Transactions on Nuclear Science*, 2023.
- [133] Q. Cheng, M. Huang, C. Man, A. Shen, L. Dai, H. Yu, and M. Hashimoto, “Reliability exploration of system-on-chip with multi-bit-width accelerator for multi-precision deep neural networks,” *IEEE Transactions on Circuits and Systems I: Regular Papers*, 2023.
- [134] N. Khoshavi, C. Broyles, Y. Bi, and A. Roohi, “Fiji-fin: A fault injection framework on quantized neural network inference accelerator,” in *2020 19th IEEE International Conference on Machine Learning and Applications (ICMLA)*. IEEE, 2020, pp. 1139–1144.