EPFL

Thèse n° 10 393

EdgeAI-Aware Design of In-Memory Computing Architectures

Présentée le 22 janvier 2024

Faculté des sciences et techniques de l'ingénieur
Laboratoire des systèmes embarqués
Programme doctoral en microsystèmes et microélectronique

pour l'obtention du grade de Docteur ès Sciences

par

Marco Antonio RIOS

Acceptée sur proposition du jury

Prof. E. Charbon, président du jury
Prof. D. Atienza Alonso, directeur de thèse
Prof. F. Catthoor , rapporteur
Prof. I. O'Connor, rapporteur
Prof. A. P. Burg, rapporteur

École polytechnique fédérale de Lausanne

2024

Inspiration exists,
but it has to find you working.
— Pablo Picasso

To my loving wife;
To my supportive parents;
To my amazing friends.

Acknowledgements

First and foremost, I am deeply grateful to Prof. David Atienza. Over the course of the past four years during my Ph.D. at the Embedded Systems Laboratory (ESL), his guidance and mentorship have been instrumental in my personal and professional growth. As a remarkable individual and an exceptional professional, he consistently inspires everyone around him to strive for excellence. I admire his commitment towards creating a favorable atmosphere for students, bringing everyone in the laboratory together through social events, and enabling productive collaborations. The supportive environment he has created has significantly improved my research experience.

I want to take this opportunity to express my gratitude to the distinguished jury members—Prof. Edoardo Charbon, Prof. Andreas Burg, Prof. Ian O'Connor, and Prof. Francky Catthoor—for their valuable time and commitment in evaluating this thesis. I sincerely appreciate their willingness to dedicate their expertise and insights to the evaluation process.

I am thankful for the post-doctoral researchers who supported and guided me throughout my PhD. Dr. Alexandre Levisse and Dr. Giovanni Ansaloni greatly impacted my research. Alexandre provided invaluable insights and a strong commitment that helped shape my work. He generously shared his time and expertise to explain complex concepts. His passion for technology and circuits was contagious and inspiring. Giovanni's contributions helped broaden my perspective and explore new areas of inquiry. He had a meticulous approach to explaining intricate concepts and was incredibly patient. I admired his attention to detail in both his explanations and scientific writing. The qualities of Alexandre and Giovanni inspired me to strive for greater precision and clarity in my work.

I am immensely grateful for the incredible memories I shared with my colleagues from the ESL. Their presence and support have made my journey truly remarkable. In particular, I would like to express my heartfelt thanks to Ignacio Penas, whose companionship has been integral to my life during these years. I would also like to thank Renato Zanetti, Halima Najibi, and Dionisije Sopic, who have been instrumental in creating countless unforgettable moments together. Additionally, I am grateful to Andrew Simon and Flavio Ponzina, whose diligent work and dedication have been a constant source of inspiration for me during our collaborations. Thanks to Grégoire

Acknowledgements

Eggermann and Clément Choné for their invaluable assistance during the critical moments of the HEEPocrates tape-out. Their support came at a crucial moment and played a significant role in the successful completion of the project. I would also like to acknowledge Homeira Salimi for her tireless efforts in organizing social events and ensuring the smooth functioning of the lab. Her dedication and commitment have created a vibrant and harmonious environment that has greatly enhanced our experience in the ESL.
Furthermore, I am grateful to Christoph Muller for his assistance with the server and for sharing his knowledge, which proved invaluable in the tape-out process. His expertise and guidance were indispensable in overcoming challenges and achieving our goals. A special mention goes to John Maxwell, who skillfully managed our social media platforms, helping to showcase the valuable work being done in the ESL to a wider audience. His efforts in spreading awareness and engagement were greatly appreciated.

Lausanne, August 25, 2023 Marco Antonio Rios

Abstract

Driven by the demand for real-time processing and the need to minimize latency in AI algorithms, edge computing has experienced remarkable progress. Decision-making AI applications stand out for their heavy reliance on data-centric operations, predominantly characterized by matrix and vector manipulations. Consequently, due to these computational patterns, conventional computer architectures that separate CPU and memory units (Von-Neumann model) face limitations in meeting AI's performance and power requirements.

In contrast, IMC proposes to execute the workloads directly from memory access and has emerged as a potential solution to overcome these limitations. Particularly, SRAM-based architectures have demonstrated the ability to perform digital bit-wise operations by simultaneously accessing two words in memory (i.e., bit-line computing). However, despite their promising capabilities, these architectures also pose challenges in the context of AI computations. This thesis effectively bridges the gap between AI requirements and edge hardware constraints. It exposes the proposed architectures to these workloads and addresses their limitations, enabling them to be an efficient platform for executing edgeAI applications. The research focuses on circuit innovations that enhance in-memory linear algebra operations' speed, efficiency, and reliability while introducing innovative optimization methods for AI applications fully leveraged by the architectures.

Firstly, since bit-line computing architectures extract bit-wise operations in each memory access, they require several cycles to perform multiplications using the shift-add algorithm. In contrast, this thesis proposed accelerated signed two's complement multiplications with minimal area overhead, supporting overflow on multiplications and accumulation to enhance computational efficiency and accuracy.

Energy consumption is a critical concern in SRAM-based architectures, prompting the proposal of a hybrid bit-line computing architecture that combines SRAM and RRAM. This architecture leverages specific memory features to achieve higher energy efficiency. Moreover, voltage scaling, a well-established technique for energy

optimization, introduces challenges to performing operations within memory due to increased vulnerability to errors at lower voltage levels. To address this, the thesis proposes strategies to increase read margins and introduces an error correction and mitigation strategy compatible with bit-line computing.

Notably, enabling edgeAI necessitates a comprehensive co-design approach encompassing hardware and software considerations. The solutions presented in this thesis at the system-architecture level enable efficient data management on the proposed architectures, allowing for fine-grained optimizations of convolutional neural networks. These optimizations directly contribute to increasing the efficiency of the overall system. An end-to-end framework is also proposed to support decision-making AI algorithms, integrating advanced data compression and real-time decompression techniques.

Keywords: edge computing, artificial intelligence, edgeAI, in-memory computing, bit-line computing, SRAM memory, neural networks, convolutional neural networks, overflow management, emerging non-volatile memories, data compression, heterogenous quantization.

Résumé

Poussé par la demande de traitement de donnee en temps réel et la nécessité de minimiser la latence des algorithmes d'Intelligence Artificielle (IA), l'informatique périphérique a connu des progrès remarquables. Les applications décisionnelles de l'IA se distinguent par leur forte dépendance aux opérations centrées sur les données, principalement caractérisées par des manipulations de matrices et de vecteurs. Par conséquent, en raison de ces schémas de calcul, les architectures informatiques conventionnelles qui séparent le CPU et mémoire (modèle de Von Neumann) sont limitées pour répondre aux exigences de l'IA en matière de performance et de consommation d'énergie.

En revanche, le calcul en memoire propose d'exécuter les operations directement à partir de l'accès à la mémoire et est consideré comme une solution potentielle pour surmonter ces limitations. En particulier, les architectures basées sur la SRAM ont démontré la capacité d'effectuer des opérations numériques en accédant simultanément à deux mots dans la mémoire (c'est-à-dire le calcul sur la ligne de bits). Cependant, malgré leurs capacités prometteuses, ces architectures posent également des défis dans le contexte des calculs d'IA. Cette thèse permet de rapprocher les exigences de l'intelligence artificielle et les contraintes de l'informatique périphérique. Elle expose les architectures proposées à ces charges de travail et aborde leurs limites, leur permettant d'être une plateforme efficace pour l'exécution d'applications AI. La recherche se concentre sur des innovations de circuits qui améliorent la vitesse, l'efficacité et la fiabilité des opérations d'algèbre linéaire en mémoire, tout en introduisant des méthodes d'optimisation novatrices pour les applications d'IA pleinement exploitées par les architectures.

Les architectures de calcul sur la ligne de bits extraient des opérations bit à bit à chaque accès mémoire, ce qui nécessite plusieurs cycles pour effectuer des multiplications à l'aide de l'algorithme de décalage-addition. En revanche, cette thèse propose des multiplications accélérées à complément à deux signé avec une surcharge de surface minimale, prenant en charge les débordements de données lors des multiplications et des accumulations pour améliorer l'efficacité et la précision des calculs.

Résumé

La consommation d'énergie est un problème critique dans les architectures à base de SRAM, ce qui a conduit à la proposition d'une architecture informatique hybride à ligne de bits qui combine SRAM et RRAM. Cette architecture exploite les caractéristiques spécifiques de la mémoire pour obtenir une meilleure efficacité énergétique. En outre, la mise à l'échelle de la tension, une technique bien établie pour l'optimisation de l'énergie, introduit des défis pour effectuer des opérations dans la mémoire en raison de la vulnérabilité croissante aux erreurs dans des niveaux de tension plus faibles. Pour y remédier, la thèse propose des stratégies visant à augmenter les marges de lecture et introduit une stratégie de correction et d'atténuation des erreurs compatible avec le calcul sur la ligne de bits.

Notamment, la mise en œuvre de l'edgeAI nécessite une approche de co-conception complète englobant des considérations matérielles et logicielles. Les solutions présentées dans cette thèse au niveau de l'architecture du système permettent une gestion efficace des données sur les architectures proposées, ce qui permet des optimisations fines des réseaux de neurones convolutifs. Ces optimisations contribuent directement à augmenter l'efficacité du système global. Un modèle complet est également proposé pour soutenir les algorithmes d'IA décisionnels, intégrant des techniques avancées de compression de données et de décompression en temps réel.

Mots-clés : informatique de bord, intelligence artificielle, edgeAI, informatique en mémoire, informatique de ligne de bits, mémoire SRAM, réseau de neurones artificiels, Réseau de neurones artificiels convolutifs, gestion du débordement, mémoires non volatiles émergentes, compression de données, quantification hétérogène.

Contents

Contents

List of Figures

List of Tables

1 Introduction

In the realm of edgeAI, numerous solutions have been developed to alleviate the computational demands imposed by artificial intelligence (AI) algorithms [1]. Many in-memory computing (IMC) architectures and specialized hardware accelerators have also been proposed [2]. However, optimal solutions are only achieved through a thorough co-optimization of application and hardware. Therefore, this thesis advocates for the efficient execution of state-of-the-art CNNs within SRAM memories with minimal area overhead. The proposed IMC architectures are closely optimized to leverage AI's spatial and temporal data locality, robustness, and parallelization capability.

Cloud computing has been crucial in advancing AI. Still, depending on it to deploy decision-making and classification algorithms may not be optimal in many scenarios. In applications where AI is needed, the devices collecting data are often on the edge. Since these devices have a limited power supply, relying on wireless connections to send the data can drain the power quickly. Additionally, applications like bio-signals monitoring in implanted medical devices [3] or car driver assistance [4] require real-time responses. Indeed, the data transmission to and from the cloud significantly increases the latency. Additionally, cloud computing is highly dependent on a strong and stable network connection, which means that in areas with unreliable connections or devices that do not have connectivity, cloud-based AI applications may not be practical or feasible.

EdgeAI evades energy-hungry wireless data transfers by computing the data at the edge. Switching to edge computing offers many advantages, such as low latency and real-time AI applications. Also, sensitive data are processed locally, improving privacy and security by minimizing data exposure to external networks [5]. EdgeAI enables applications to be deployed in areas with limited or unavailable internet access, leading to faster and more efficient applications and paving the way for greater

security and seamless integration into our daily lives. Despite the challenges that come with this evolution, the advantages are undeniable, and we can look forward to a future where AI is more democratized than ever before.

However, edge computing comes with its own set of challenges. Devices at the edge often have limited processing capabilities and power supply, completely opposing AI's ever-growing demand. Thus, not only do AI algorithms and hardware accelerators require optimization, they must be thoroughly co-designed. Indeed, the best solutions consider co-optimization a two-way road: AI's required operations shape the accelerator. In contrast, the availability of hardware resources acts as a driving force to optimize AI models. In this context, this thesis proposes and develops a series of AI accelerators based on in-memory computing that has been closely co-optimized with AI applications.

1.1 Deep learning algorithms

Deep learning [6] algorithms are pivotal in AI, enabling systems to learn from data and make intelligent predictions or decisions. Among the diverse range of algorithms, Convolutional Neural Networks (CNNs) [7], Recurrent Neural Networks (RNNs) [8], and Graph Neural Networks (GNNs) [9] represent fundamental pillars in this field. CNNs have revolutionized computer vision tasks, enabling machines to perceive and interpret visual information. RNNs have significantly advanced natural language processing and sequential modeling tasks, enabling machines to understand and generate human-like text. Finally, GNNs have propelled graph analysis and representation learning, facilitating intelligent decision-making in complex relational domains.

Usually, these algorithms are first trained with labeled data to learn patterns and correlations. Since the training phase usually requires massive amounts of data and involves complex operations, such as differential equations, very powerful Graphics Processing Units (GPU) [10] are employed. Henceforth, once the models are trained, they can be uploaded into edge devices, as I consider in this thesis. Therefore, the inference phase refers to applying the trained model to unseen data to make predictions or generate outputs. Indeed, inferences in the edge remain challenging for their required amount of data and Multiply-ACumulate (MAC) operations that may overwhelm edge devices' computing and energy budgets. However, unlike training, the inference is entirely based on linear algebra operations.

1.2 Linear algebra

Linear algebra is a branch of mathematics that mainly deals with vector spaces, linear transformations, and matrices. Its primary focus is on these objects' algebraic properties and operations, including scalar multiplication, addition, and matrix multiplication. It provides various tools for manipulating and representing data, analyzing geometric and algebraic structures, and solving systems of linear equations. Hence, since deep learning algorithms use matrices and vectors to represent large data sets, linear algebra efficiently organizes and manipulates them.

CNNs are a type of neural network intended for processing data organized in a grid-like format, such as time series or images. They leverage convolutional layers to extract local spatial patterns and fully connected layers for classification or regression tasks. Hence, core operations of CNNs involve element-wise multiplication in matrices (Hadamard product) or vector-matrix multiplication. Likewise, RNNs and GNNs heavily rely on matrix dot products and matrix-vector multiplications. Ultimately, all these operations can be decomposed into a series of MAC operations.

Deploying these algorithms on general-purpose computer architectures (Von-Neumann model) becomes inefficient when dealing with large data sets. This is because the model isolates the memory units from the Central Processing Unit (CPU), and the memory-bounded linear algebra operations executed by neural networks bring to light the limitations of this separation [11]. Von-Neumann architectures are primarily designed for the sequential execution of instructions, restraining parallelism capabilities. Consequently, these architectures' efficiency is muted since matrix and vector operations heavily profit from high parallelism. Finally, the frequent data movement from and to the memory of many parameters, input data, and intermediate feature maps becomes the bottleneck of the algorithm execution. This is defined as the memory wall [12]. In contrast to the Von-Neumann model, In-Memory Computing (IMC) addresses these limitations. As the name suggests, IMC tumbles the memory wall by merging memories and computing units [13].

1.3 In-memory computing

In-memory computing (IMC) has numerous advantages over general-purpose computer architectures. Firstly, it processes data directly within the memory, resulting in high-throughput data processing and lessening data transfers between the processor and memory. This reduces latency and increases energy system efficiency. IMC is intrinsically a parallel computing paradigm since it allows the architectures to distribute

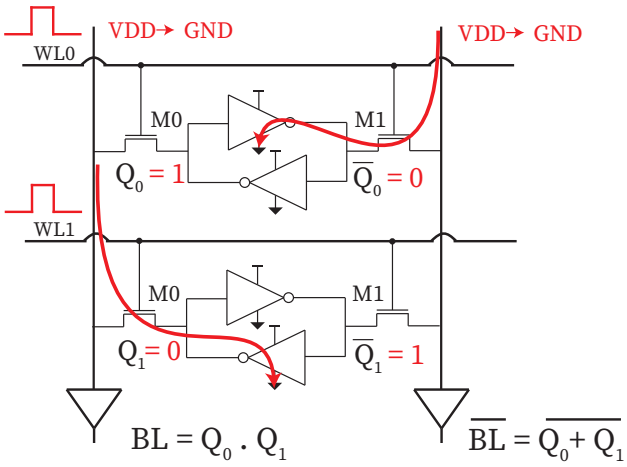
computation across multiple memory modules [14].

Clearly, IMC also offers more efficient memory utilization compared to traditional architectures, enhancing data spatial and temporal locality needed for linear algebra operations. In-memory processing ensures that the data required for computation is already present in the corresponding memory location. This efficient utilization of memory resources improves overall system performance and enables handling larger datasets. Overall, these advantages make IMC an attractive computing paradigm for a wide range of applications that require high throughput, energy efficiency, scalability, and optimized memory utilization.

Different methods of IMC were proposed in previous works, using either resistance-based or charge-based devices [2]. The former, also known as Emerging Non-Volatile Memory (eNVM) [15], offers several advantages, including high density and non-volatility. There are different types of eNVM, such as Resistive Random Access Memory (ReRAM) [16], Phase Change Memory (PCM) [17], and Magnetic Random Access Memory (MRAM) [18], among others. These devices use various physical properties to enable programmable resistance, allowing for multilevel resistance/conductance programming and enabling IMC operations in the analog domain. Thus, by leveraging Ohm's law and Kirchhoff's current summation laws, Resistive Crossbar Arrays (RCA) can perform matrix-vector multiplications, intrinsically benefitting from the memory structure.

However, RCAs face several limitations regarding the lack of maturity of resistive devices, such as write accuracy, resistance drift, and low endurance [19, 20, 21]. To minimize these drawbacks, solutions such as MAGIC [22] use resistive devices as logic units by assigning values to High Resistance State (HRS) and Low Resistance State (LRS), enabling bit-wise digital computations. Still, the resistive devices are constantly written during the computation, which represents a major drawback since eNVMs show limited endurance and high currents to write operations.

In contrast with those limitations, IMC architectures that rely on charge-based devices such as Static and Dynamic Random-access Memories (SRAM and DRAM) do not present problems of write endurance. They are also closer to commercial availability, thanks to their maturity. However, while SRAM arrays are CMOS-based, enabling seamless integration of digital logic, DRAM requires a specialized fabrication process, limiting the scope of operations [23]. Thus, in the following, SRAM-based solutions are highlighted.



Q_0	Q_1	BL	$\overline{BL}$
0	0	0	1
0	1	0	0
1	0	0	0
1	1	1	0

Figure 1.1. Bit-line computing between two SRAM bit-cell. BLs behave as the logic gates AND and NOR when concurrently accessing two memory cells on the same BLs.

1.3.1 SRAM-based architectures

Analog computing: In the search for strategies similar to RCAs, several solutions have emerged that use SRAM instead of resistive memory. Notable among these solutions are conv-RAM [24], IMAC [25], and Vesti [26], all of which leverage the analog domain for computation. By capitalizing on the intrinsic analog characteristics of SRAM cells, such as resistance and voltage levels, and employing pulse-width modulation for bulk matrix-vector computations, these solutions have achieved remarkable energy efficiency without the downsides of resistive memories. Conv-SRAM boasts an impressive 28 TOPS/W, exceeding the capabilities of any digital implementation.

However, the high energy efficiency these solutions offer is offset by substantial area penalties. For instance, IMAC which adopts high-density 6T bit-cells, shows a 150% increase in required architecture area for peripherals, in a 1KB array compared to the SRAM bit-cells array. Furthermore, the deep modifications these solutions impose upon the memory array limits the architecture to perform a single task and cannot be abstracted as a conventional memory, such ability that most of digital implementation have. Take, for instance, the case of conv-RAM, where the subarray operates at 6.7MHz while its Analog-to-Digital Converter (ADC) runs at 377MHz, thus it requires several extra frequency domains on top of the one from the system.

Beyond architectural intricacies, these approaches necessitate substantial modifications to CNNs models themselves. This requirement includes enforcing fixed quantization levels for weights, a constraint that may potentially hinder CNN accuracy and optimization efforts, considering the distinct impact that the CNN layers have on

accuracy. Consequently, these architectures are inflexible in their proposition, offering limited avenues for accommodating other neural network models and workloads. Thus, while solutions like conv-RAM, IMAC, and Vesti exhibit remarkable energy efficiency by harnessing the analog domain of SRAM, they suffer with trade-offs such as increased area overheads, design complexity and highly constrained models. Instead, digital computing with SRAM can still yield high efficiencies with minimal modification in conventional SRAM arrays by performing bit-line computing.

Digital computing: Bit-line computing was initially demonstrated in [27]. The authors introduced a technique that relies on simultaneous access to multiple words within an SRAM memory array, as depicted in Figure 1.1. Consequently, the bit-wise AND and NOR logic operations are carried out directly on the bit-lines. Notably, this technique enables Single Instruction, Multiple Data (SIMD) operations on the memory hierarchy while maintaining the original layout and structure of cache memory or other SRAM arrays. However, it must be noted that bit-line computing can lead to data corruption. When multiple word-lines are activated, the bit-cells on the bit-lines may short-circuit if they store different data, causing a current flow between them and resulting in a flip in their logic state.

To overcome this limitation, several works, such as DRC2[28] and Neural caches[29], propose adding read ports to the bit-cells to protect the data. However, this descreases area efficiency. Otherwise, in [30], safe operations are attained by underdriving the word-line, increasing the read stability at the cost of up to 50% drop on the operating frequency. Instead, in [31], the authors proposed BLADE (Bit-Line Accelerator for Devices on the Edge). It performs high-frequency bit-line computing using high-density 6T SRAM bit-cell thanks to its simple Local Group (LG) organization. LGs allow BLADE to separate the bit-lines that perform the memory accesses from the bit-lines in which the computing is carried out. Thus, data corruption risk is completely addressed by ensuring that the accessed words are placed on distinct LGs.

BLADE performs a series of bit-wise operations on its logic unit. With the integration of an adder, BLADE [32] also carries out array-level multiplications by performing repeated shifts and additions. This architecture was compared to the ARM's NEON architecture [33], a SIMD accelerator for edge devices. BLADE showed 4× performance gain and 6× energy reduction when executing cryptography algorithms dominated by bit-wise operations, such as XOR, AND, and NOT. However, when the workload is dominated by multiplications, such as a convolutional layer of CNNs, the efficiency gains are reduced to 3× higher performance and 1.5× energy reduction. The reason is that unlike analog computing, which intrinsically perform matrix-vector multiplication, bit-line computing relies on bit-wise operations, requiring several cycles to

Table 1.1. In-SRAM architectures comparison.

	Domain	Bit-cell	Tecnology	Frequency	Efficiency	area overhead
Conv-RAM [24]	Analog	10T	65nm	6.7MHz	28 TOPS/W	-
IMAC [25]	Analog	6T	65nm	-	-	150%
Vesti [26]	Analog	6T	65nm	0.55GHz	-	-
Neural Cache [29]	Digital	8T	28nm	4GHz	-	7.50%
BLADE [31]	Digital	6T	28nm	2GHz	35 GOPS/W	8%
AA-BC - Thesis contrib. [34]	Digital	6T	28nm	2GHz	1.78 TOPS/W	12%

finish a multiplication.

When compared to digital counterparts, BLADE stands out for its efficiency, reaching an impressive 35 GOPS/W. However, it's important to note that this efficiency is considerably lower when contrasted with the exceptional efficiency of conv-RAM, which surpasses BLADE by a significant margin of 800x[1].

Taking into account the contributions made by this thesis, the architecture AA-BC introduced in Section 2, along with the execution strategy detailed in Section 4, achieves an efficiency of 1.78 TOPS/W. While this figure may fall short by a factor of 15 compared to analogous implementations, it's crucial to recognize that this digital architecture retains all the fundamental advantages of digital design, including adaptability and ease of design. This balance between efficiency and the inherent benefits of digital design underscores the promise of the proposed approach. Table 1.1 compares the discussed architectures.

1.4 Optimization of CNNs

Indeed, executing large CNNs on SRAM arrays through bit-line computing is challenging. CNNs involve repeated access to filter weights and input data with specific spatial patterns. Even if bit-line computing architectures are intrinsically bounded with SRAM banks and improve spatial data locality, they may require additional data copies to accomplish convolutions or matrix-vector operations, hindering the potential efficiency gain of reducing data transfers. Moreover, CNNs often require high precision in their computations to maintain accuracy, usually, the baseline of these models is represented in floating-point. In [35], the authors propose bit-line computing architectures to perform floating-point operations. However, a single 32-bit multiplication may take up to a thousand cycles.

[1] It's worth considering that BLADE was designed in 28nm technology, while conv-RAM was designed in 65nm technology; therefore, this ratio might change if both architectures were produced using the same technology

CNNs are highly robust algorithms, and several software optimizations are possible without major accuracy degradations. E2CNN [36] aims to increase the models' robustness for memory upsets. The authors use pruning [37], which eliminates useless computing paths, to reduce the model size by a factor of N. By replicating the model N times, E2CNN achieves higher model robustness.

Focusing on leveraging the CNN's robustness, in [38], the authors propose a framework to quantize the models from floating-point to 16-bit and 8-bit words fixed-point notation, greatly impacting the memory requirement and simplifying arithmetic logic units. Additionally, In [39], the authors propose a three-stage pipeline to compress CNNs based on pruning, quantization, and losslessly compression of quantized weights, reaching compression rates over 50×. Moreover, speed-ups of up to 4× are shown when benchmarking with CPUs and GPUs. However, since these units do not support the non-conventional quantization levels proposed by the authors, such as 4- or 5-bit words, this compression phase does not translate into higher computational efficiency.

1.5 Contributions

Chapter 2 addresses the challenges in accelerated fixed-point arithmetic operations, addressing data overflow and enhancing performance. Chapter 3 further explores the versatility of bit-line computing by combining SRAM and RRAM, leveraging their unique characteristics for improved performance. Moreover, the proposed architectures also mitigate accuracy loss resulting from voltage scaling. Additionally, in Chapter 4, the optimization methods for mapping and compressing CNNs on IMC architectures are presented. In summary, the contributions are as follows:

- Chapter 2 presents two innovative bit-line computing architectures. These architectures were designed to enable and accelerate edge AI workloads. Unlike BLADE, which showed promising performance improvements compared to NEON but suffered from slow multiplications and data overflow, the presented architectures improve multiplication efficiency and fully tackle overflow issues. Thanks to the proposed methods, a bit-line computing architecture is proposed with 4× increase energy efficiency. Additionally, this chapter explores various strategies for enabling parallelism at the subarray and array levels.

- Then, Chapter 3 focuses on exploring the pursuit of energy-efficient bit-line computing architectures. The hybrid one integrates SRAM and RRAM technologies, while the resilient architecture is designed to support aggressive supply

voltage scaling. These architectures are tailored to the specific characteristics of deep neural networks. By efficiently managing data accesses at run-time, the hybrid architecture reduces the need for extensive data transfers during CNN inference. Voltage scaling is a crucial technique to achieve energy budgets. Therefore, the chapter presents methods to allow near-threshold voltage reading access and reduce energy consumption during programming operations in RRAM memory architectures. Additionally, the resilient architecture, fully designed with SRAM, includes an error mitigation technique to enhance robustness against memory errors. This architecture leverages the E2CNN strategy on the proposed error mitigation method and shows that significant energy savings can be achieved without compromising the initial accuracy of CNN models.

- Next, Chapter 4 presents a framework that optimizes, compresses, and executes CNNs within bit-line computing or other equivalent architectures. The framework begins with the compression of CNN models through a combination of heterogeneous quantization and lossless encoding of weights or activations (depending on the type of layer). This compression strategy significantly reduces memory requirements without compromising accuracy, offering a viable solution for resource-constrained edge devices. Furthermore, the section shows how to decompress the CNN parameters in real-time and cast to it instructions for highly parallelized execution. In general, this chapter provides valuable insight and solutions for improving the performance of edge AI applications.

- In conclusion, Chapter 5 of the thesis emphasizes how the proposed findings lead to more efficient and scalable AI deployments on resource-constrained edge devices. It discuss potential future directions, such as silicon verification to validate the concept, architectural exploration to facilitate data scaling and distributed learning at the edge, and system support for the proposed architectures.

2 Designing and optimizing in-SRAM arithmetic for edgeAI

As discussed in Chapter 1, Bit-line Computing (BC) architectures effectively improve the energy efficiency of edge device workloads and can ultimately enable edgeAI. However, algebraic operations such as Multiply-Accumulate (MAC) require multiple cycles in constrained circuits or area and energy-hungry multipliers, either decreasing performance or increasing area overhead. Both scenarios hamper the integration of BC architectures in low-cost edge devices. To counter this drawback, in this chapter, I show how I enriched BC architectures in two axes: *(i)* Accelerating MAC operations with minimal impact on the area overhead, *(ii)* Providing full support to enable reliable edgeAI core operations.

A summary of this chapter's contributions is presented as follows:

- I present BLADE (Bit-Line Accelerator for Devices in the Edge), the starting point architecture for my optimizations. I have contributed to this work by performing a space design exploration to evaluate the impact of the memory geometry on the area, energy, and memory access time.

- I propose the Associativity-Agnostic BC (AA-BC) architecture supporting signed two's complement fixed-point multiplication. AA-BC integrates several enhancements that simplify the design and enable a data-dependent acceleration of MAC operations. Moreover, a new strategy for parallel SIMD-like operations on a multiple-subarray architecture is presented.

- Finally, I show the MAC accelerator based on BC (MAC-BC) architecture. It encompasses all AA-BC enhancements in an even simpler and more efficient periphery. It uses registers to manage MAC overflow that are compatible with BC operations. I compare MAC-BC with an equivalent digitally implemented

near-memory computing architecture, showing that computing with bit-lines is more energy and area efficient.

Firstly, in Section 2.2, I present the initial architecture, BLADE, published in IEEE Transactions on Computers, 2019 [31]. Next, in Section 2.3, the innovations to enable and accelerate two's complement multiplications directly at the SRAM array are presented. This section gathers works published in IFIP/IEEE International Conference on Very Large Scale Integration - System on a Chip (VLSI-SoC), 2019 [40]; IEEE Computer Society Annual Symposium on Very Large Scale Integration (ISVLSI), 2021 [41]; and IEEE Transactions on Emerging Topics in Computing (TETC), 2023 [34]. Finally, in Section 2.4 I present the strategy to support overflow in both multiplications and accumulations. This work has been submitted to The International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS), 2023 [42].

2.1 Compute memories

The high computational demands of AI applications strain the capabilities of traditional Von-Neumann architectures, especially when considering the tight resource constraints present in edgeAI scenarios. This challenge has fostered a renewed interest in domain-specific computing solutions, inspiring novel approaches based on a variety of systems based on FPGAs, GPUs, systolic arrays, etc. [43] [44], arguably generating a "Cambrian explosion" [45] of heterogeneous solutions in the computing architecture landscape.

In this context, compute memories are particularly promising because they leverage the data-centric nature of AI algorithms, i.e., the fact that applications are composed by manipulations on a massive number of data items. Compute memories can be divided into two subcategories: Near- (NMC) and In-Memory Computing (IMC). While the former only reads the memory and computes the operations independently, the latter computes (fully or partially) operations as part of data access.

In fact, computing memory solutions have been derived in many different ways in recent years, exploiting various technologies, such as resistance-based memories (PCM and RRAM) [2] and dynamic [46] and static random access memories [23] (DRAM and SRAM).

Resistance-based memories, despite their promising potential, still encounter various limitations due to their lack of maturity. Challenges such as writing accuracy, resistance drift, and low endurance persist and need to be addressed for their widespread adoption [19, 20, 21]. On the other hand, established memory technologies like DRAM

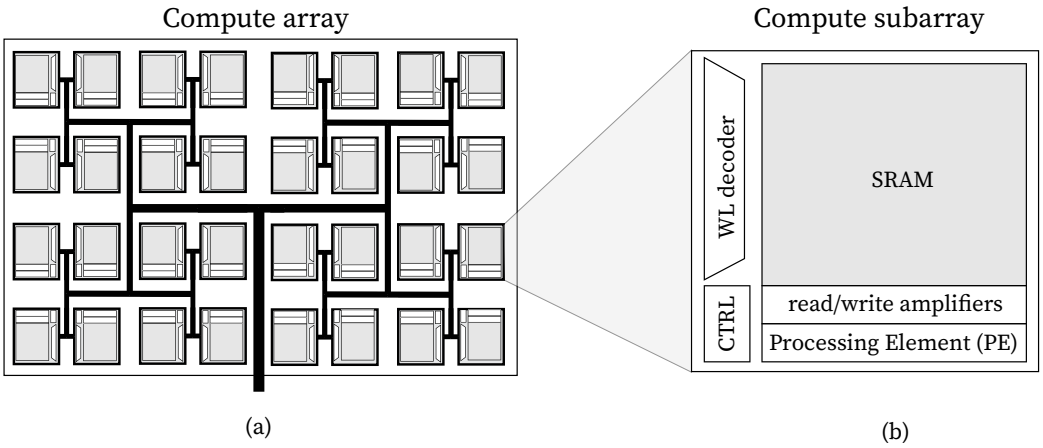


Figure 2.1. (a) H-tree composition of SRAM subarrays that can compute in parallel. (b) Basic elements of a computing subarray.

and SRAM do not face these limitations to the same extent. In addition, their maturity and closer proximity to commercial availability make them attractive options. While SRAM arrays are based on complementary metal-oxide-semiconductor (CMOS) technology, enabling seamless integration with digital logic, DRAM necessitates a specialized fabrication process, which restricts its versatility. Therefore, I herein reference notable works in SRAM-based compute architectures.

SRAM-based near-memory computing architectures take advantage of the regular structure of SRAM arrays. These comprise several subarrays typically interfaced with a system bus via an H-tree interconnect (Figure 2.1-a). As SRAM can be readily co-integrated with CMOS logic on the same die, processing capabilities can be added as digital logic at each subarray's periphery, enabling high parallelism and reducing data transfer requirements. To this end, in [47], the authors use look-up tables to accelerate execution, showing 3× speed-ups with respect to GPUs on AI algorithms.

Further works in this area exploit near-memory processing elements to implement domain-specific functional units. Authors of [48] introduced near-SRAM shifters, rotators, and s-boxes for cryptographic applications, while in [49], a multi-bit product-sum engine is described.

SRAM-based IMC architectures usually rely on the BC strategy. BC performs MAC operations by activating multiple memory rows and passively extracting bit-wise logic operations from the bit-lines. The active logic of the peripheral is then employed to derive arithmetic functions.

In-memory strategies aim to exceed the performance of NMC architectures by adopt-

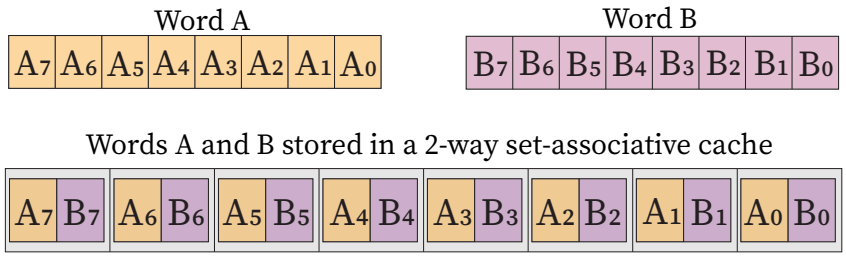


Figure 2.2. Two words bit-interleaved in a 2-way set-associative cache example.

ing modified memory arrays to execute part of the implemented functionality. Most approaches in this field are based on concurrently activating multiple bit-cells in the same memory column, hence implementing the bit-wise AND and NOR operations between two words. Such a mechanism does not depend on the implementation of the memory cell itself, which may be realized as SRAMs, Flip-Flops (Section 2.4), RRAMs [50](Section 3.3) or even a combination of different memory technologies.

2.2 BLADE: bit-line accelerator for devices in the edge

BLADE is an SRAM-based BC architecture designed to accelerate operations inside cache memories. This section will overview BLADE's principle, functionality, and main results. BLADE separates the bit-lines that perform the memory accesses, called Local Bit-Lines (LBL), to the bit-lines in which the computing is carried out. These are called Global Read Bit-lines (GRBL). Thanks to this division, BC operations are performed at high frequencies using commodity 6T SRAM bit-cells. However, computing in the GRBL is only possible when the accessed words are connected to different sets of LBLs, which limits the eligible operands to be computed and may impose run-time overhead due to data movement.

The LBLs isolate the bit-cells into smaller groups, decreasing the parasitic capacitance. This results in an architecture with higher energy efficiency and lower access delay. Moreover, the limited parasitic capacitance reduces bit-line leakage, improving the static noise margin and leading to a more reliable read operation. This is particularly interesting when BLADE operates at lower voltages. The segment of memory that comprehends each set of LBLs is called Local Groups (LG). The LG also has specialized Periphery (LGP) circuitry that interfaces the local and global bit-lines.

BLADE is an in-cache accelerator that uses set associative mapping. Figure 2.2 shows how two words are bit-interleaved and stored in the same line in a 2-way set associative cache memory. This policy enables a higher cache hit rate than direct mapping in

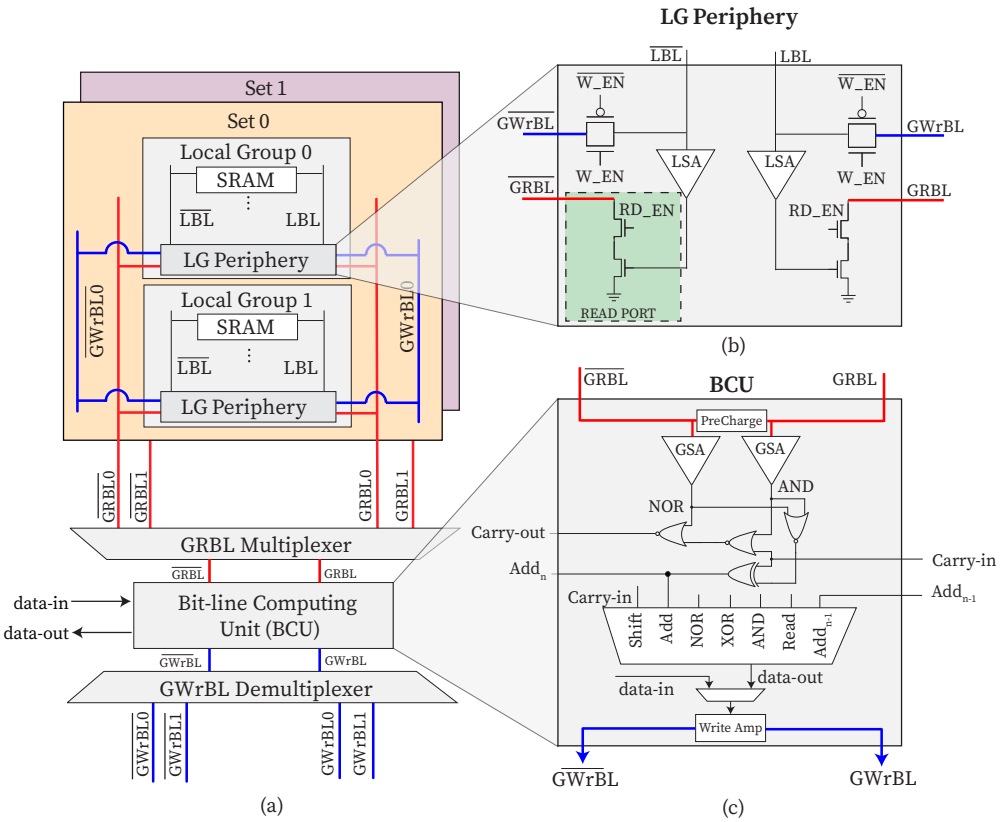


Figure 2.3. (a) BLADE architecture, (b) Local-group periphery circuit, (c) Bit-line computing unit.

conventional cache memory. In BLADE, it greatly impacts the architecture's physical layout. Considering that the LGP and the Bit-line Computing Unit (BCU) need to be aligned and pitched with the SRAMs bit-cells, these blocks' width is limited to $S \times W_{sram}$, where S is the number of sets in the cache and W_{sram} is the width of an SRAM bit-cell. Thus, more significant values of S allow a more relaxed layout, reducing the area overhead. However, increasing the number of sets decreases the performance (fewer bit-wise operations can be executed in parallel). It also limits flexibility since BLADE only performs BC operations between words from the same set.

Figure 2.3-a depicts a block schematic of 1-bit of the BLADE architecture. It has two LGs that share GRBLs and Global Write Bit-Lines (GWrBL). The GRBL multiplexer selects the memory set transmitted to the BCU. The BCU performs bit-wise and arithmetic operations. Its output is either the GWrBL or the data-out. The former is selected if the computed word must be returned to the memory. In this case, the GWrBL demultiplexer selects the correct set of memory. The latter is used when the data must be read from memory.

Figure 2.3-b details the LGP circuit responsible for performing individual SRAM write and read operations inside the LG. For a write, firstly, the data is set on the GWrBLs by the write amplifiers. Then, the GWrBLs are used to set the LBLs through a pair of transmission gates. The transmission gate operates as a switch, transmitting the signal with a very low drop in the voltage. Finally, the write operation is completed by switching on the word-line and allowing the bit-cell to be programmed.

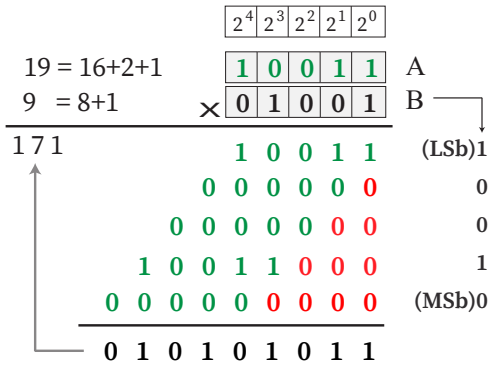
In a read or BC operation, with the LBLs and GRBLs pre-charged to Vdd, the word-line connects the data stored in the bit-cell to the LBL (and $\overline{\text{LBL}}$) through the access transistors. Then, one bit-line discharges to the ground while the other remains at Vdd. This behavior is sensed by the Local Sense Amplifiers (LSA), becoming a logic value. The data sensed at the LGP level needs to be transferred to the GRBLs to complete the operation. To this end, the read port, composed of two NMOS transistors in series, controls the discharge of the GRBLs based on the LSAs output and the read enable (RD_EN) signal. Finally, the Global Sense Amplifiers (GSA) finish the read or BC operation by sensing the GRBLs.

Figure 2.3-c shows the circuit of the Manchester carry adder. The add-and-carry signal computing uses only four logic gates (3 NOR gates and 1 XOR gate) since it is performed based on the AND and NOR carried out on the GRBLs. Moreover, a multiplexer selects the operations to be performed, and these operations can be bit-wise: AND and NOR (from the GRBLs) and XOR (from the adder implementation). Indeed, the pool of operations also counts with add and shift. These are the basis for implementing multiplication. However, depending on the algorithm, the multiplication might need varying cycle counts in the data distribution function, complexifying its control from a system perspective. For this reason, the add-forward operation (Add_{n-1}) concatenates the addition and shift in a single cycle, allowing the multiplication to have a homogeneous cycle count despite the data.

2.2.1 Unsigned integer multiplication in BLADE

Regardless of the base in which the number is represented (e.g., decimal, binary), the multiplication of two operands can be decomposed into partial products, which are summed up to retrieve the entire product. Each partial product is found by multiplying each digit of the multiplier with the multiplicand and shifting the result to the left based on the position of this digit. In binary base, the digits of the multiplier are either zero or one. Thus, the partial products are either zero or the left-shifted multiplicand.

Figure 2.4 illustrates an example that shows a binary integer multiplication of two input operands, A ($10011_2 = 19_{10}$) and the B ($01001_2 = 9_{10}$). Figure 2.4 also shows the



Naive multiplication in BLADE:	Cycle count:
Add(A, C) → Wb(C); LSh(A)→Wb(A)	4
LSh(A)→Wb(A)	6
LSh(A)→Wb(A)	8
Add(A, C) → Wb(C); LSh(A)→Wb(A)	12
LSh(A)→Wb(A)	14

Figure 2.4. Binary integer multiplication example, between the A ($10011_2 = 19_{10}$) and the B ($01001_2 = 9_{10}$), the multiplication is decomposed in shift-add instructions. LSh operations are left shifts.

required operations for a naive implementation, where C is the partial product at each iteration. It can be noticed that operations are only performed with A (left shifted in each step) and product vector C, while each bit of the B (b_n) dictates which operation should be done at each stage. When b_n is equal to one, four cycles are required (one addition, one shift, and two write-backs), while for b_n equal to zero, two cycles are required (one shift and one write-back). In the following, I show how the Add_{n-1} homogenizes the cycle count per b_n.

Unlike the naive implementation depicted by Figure 2.4 illustrates. BLADE uses a multiplication algorithm that shifts C instead of A. Consequently, the inspected bits (b_n) start from the most (MSB) to the least significant bit (LSB). In this algorithm, if $b_n = 1_2$, A is accumulated into C. C, on its turn, is left shifted in every iteration except the last (when processing the LSB). Thus, since the accumulation and shift operations are being performed in C, they can be concatenated and share the same write-back operation. Therefore the add-forward Add_{n-1} is selected when $b_n = 1_2$. Table 2.1 demonstrates this technique via the same example presented in Figure 2.4. The number of cycles for the example is 10 with add-forward, while four more cycles would be required without add-forward (Figure 2.4).

A multiplication cycle counts independent of the binary value is very important for BLADE since it features support for SIMD operation. BLADE subarrays store 64-bit words, however, it supports from 1 operation of 64-bit up to 8 operations of 8-bit. Moreover, BLADE can count on several subarrays to compose the whole architecture. Therefore, since it counts with a homogeneous multiplication cycle count, the instructions can be shared during the multiplication.

Table 2.1. Comprehensive example of a multiplication between $A = 19_{10}$ and $B = 9_{10}$ with detailed intermediate steps.

C Binary	C Decimal	b_n (MSB → LSB)	BC Operation	BLADE Instruction	Cycle Count
0000000000	0	0	LSh (C)	LSh (C) → Wb(C)	2
0000010011 0000100110	19 38	1	Add (C,A) LSh (C)	Add_{n-1}(C,A) → Wb(C)	4
0001001100	76	0	LSh (C)	LSh (C) → Wb(C)	6
0010011000	152	0	LSh (C)	LSh (C) → Wb(C)	8
0010101011	171	1	Add (C,A)	Add(C,A)→ Wb(C)	10

2.2.2 Experimental setup, validation methodology and main results

I've electrically validated BLADE using the 28nm CMOS technology from TSMC. A 256WL × 64BL array has been implemented, divided into 2LGs of 32WLs that store 64-bit words. The array has a 4-way set-associative policy and considers high-density SRAM bit-cell with a pitch of 500nm. This allows the periphery to be pitched at 2μm, totaling an area overhead of 8%. The array operates at a maximum frequency of 2.2GHz at 1V and 416MHz at 0.6V. BLADE requires 24.4 fJ/bit on average to perform read, write, or BC operations. However, these results are heavily dependent on the geometry of the array. In Section 2.2.2.1, I discuss the trade-offs.

BLADE has also been validated at a system level by Dr. Willian Simon, the original author of BLADE. It has been implemented into the gem5-X [51] architectural simulator to measure energy and performance trends for emerging edge device workloads. Three benchmarks were selected: Cryptography [52], HEVC Video Processing [53], and a single convolutional layer of a CNN. Moreover, A NEON coprocessor [33], a SIMD unit found on many edge devices, was also simulated for performance comparison. gem5-X provided traces of all CPU, memory, NEON, and BLADE operations, which have been used to compute application performance and energy consumption for NEON and BLADE benchmarks.

2.2.2.1 Subarray design space exploration

I explored the complex interrelationship between the subarray parameters (e.g., number of word-lines, bit-lines, and local groups) with its performance, energy, and area efficiency. Figure 2.2.2.1 shows the results for a 2KB memory with varying dimensions.

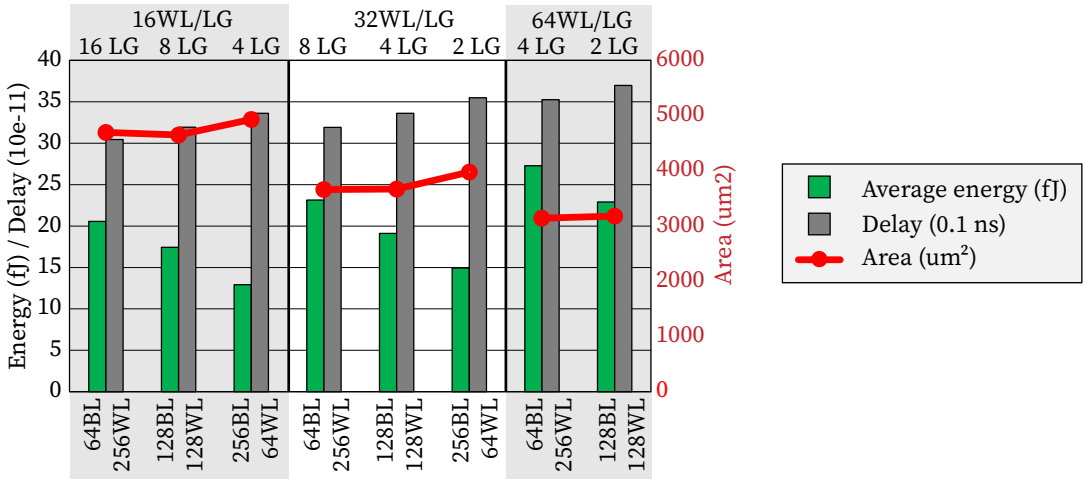


Figure 2.5. Energy, area, and delay variations across subarray geometries.

Varying the number of word-lines in each LG trades-off area by energy and memory access time. On the one hand, the number of word-lines in each LG significantly impacts area efficiency, with larger LGs requiring overall less periphery, resulting in higher efficiency. In a 128BL × 128WL configuration, LG sizes of 16, 32, 64, and 128 result in 55.6, 71, 84.4, and 91 percent area efficiencies, respectively. On the other hand, with smaller LGs, the parasitic capacitance connected to the LBLs decreases, speeding up switching time and increasing energy efficiency. In a 64BL × 256WL configuration, BLADE is 33% more energy and performance efficient with an LG of size 16 than 64.

Regarding the size of the word-line, varying its size results in a divergent trend in delay and energy. Bigger word-lines slow down BLADE but provide cheaper accesses, as switching energy is shared between more bits, averaging down the access energy cost. Considering designs with an LG size of 16 word-lines, BLADE is 37% less energy efficient with small word-lines crossing 64BLs than longer word-lines crossing 256BLs. On the other hand, the configuration with smaller word-lines is 14% faster.

2.2.2.2 System impact evaluation

The first edge device workload, cryptography [52], is based on three bit-wise operations: XOR, Shift, and AND. BLADE presented up to 4×/6× performance/energy improvement concerning the NEON architecture. Secondly, With the 4K HVEC video compressing [53] workload, BLADE yields 4.3× higher efficiency compared to NEON. Otherwise, energy gains are limited to 1.8× since this workload is dominated by multiplications, which are more complex and requires more memory accesses to be performed, hindering the energy efficiency of BLADE.

Finally, CNNs have more than 90% of their operations based on MAC operations. Since BLADE uses unsigned integer binary format, MAC operation will rapidly increase the data bit-width. For this reason, with this workload, BLADE stores data in 32-bit and 8-bit in a 64-bit word as an overflow management strategy. This architecture heavily underutilizes the memory, decreasing potential gains for MAC-based workloads. Consequently, BLADE presents up to only 1.5× better energy efficiency and 3× better performance than NEON.

2.2.3 Main drawbacks and limitations

Although BLADE presents promising results to accelerate edge workloads, it still presents some drawbacks that may hinder its usability. In the following, I list these issues and link them with solutions that I present in this chapter.

- **Data locality**: In Section 2.3.1, I present how I re-structured an LG-based BC architecture to enable associativity-agnostic BC operations, decreasing the data locality issue by a factor equal to the number of sets in the memory.
- **Slow multiplication**: In Section 2.3.1.1, I present how to profit from the memory structure to accelerate multiplication, using the bit-lines to perform multiple in-situ shifts and one addition in a single clock cycle.
- **Multiplication overflow**: In Section 2.3.2, I show how to provide full support to two's complement fixed-point arithmetics, which allows the use of truncation to approximate multiplications without needing more bits to represent the data.
- **MAC overflow**: Finally, in Section 2.4, I present an architecture fully dedicated to accelerating MAC operations. This architecture is compared to an equivalent digital NMC implementation. I show that a BC-based architecture outperforms a digital implementation in the area and energy efficiency while maintaining the same performance.

2.3 AA-BC: associativity-agnostic bit-line computing

The AA-BC architecture accelerates multiplications based on processing specific patterns of the data instead of the bit-by-bit analysis presented in BLADE. The strategy considers that multiplication is performed between operands residing in each subarray, called In-Memory Operands (IMO), and a common Broadcasted Operand (BO) embedded in the subarray instructions. Thus, the Multiple-IMO, Single-BO (MISB)

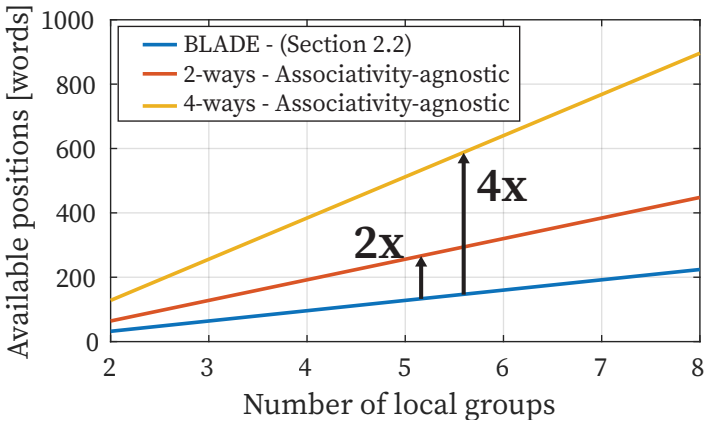


Figure 2.6. Available positions for operands (i.e., misalignment mitigation) versus the number of LGs for the proposed and baseline BL computing memories.

strategy enables several parallel multiplications with the cycle count varying BO-wise instead of subarray-wise. Importantly, this approach allows fine-grained flexibility in the bit-width of BOs, which may assume arbitrary values, leading to fine-grained control of trade-offs between model cost (size, energy, time-per-inference) and accuracy (explored in Chapter 4). Finally, multiple subarrays are connected in an H-tree configuration, as shown in Figure 2.1-a. This organization ensures that all the signals transmitted from/to the array periphery have the same distance to all the subarrays, equalizing the delays and minimizing critical paths.

2.3.1 Modified local group

A BC architecture that is associativity-agnostic allows BC operations to be realized between words that occupy different sets of an associative cache memory. Naturally, the operands still must reside in different LGs.

Associativity-agnostic operations simplify the controller at the system level since it enhances the range of eligible operands. It eases the constraints on data placement and increases overall energy efficiency by avoiding unnecessary data movement. Considering a BLADE architecture of 4 sets, 2 LGs of 32 word-lines, totaling 256 words. Hence, each word has only 32 potential available operands with which it can be multiplied. However, in the same configuration but enabling associativity-agnostic BC operations, 128 positions are available. Indeed, more positions can be made available by increasing the number of LGs, as shown in Figure 2.6.

Figure 2.7 shows the difference between BLADE and the AA-BC. The dashed line colors are related to the set of memory that the data is accessed. Blue is the first set (way-0),

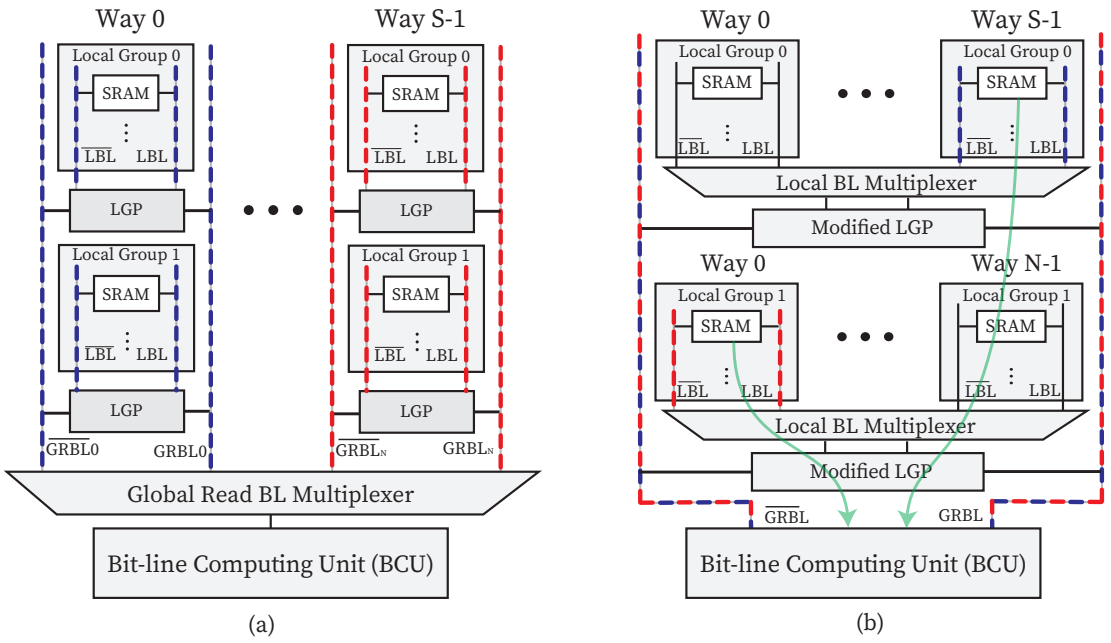


Figure 2.7. (a) BLADE memory organization, (b) Proposed associativity-agnostic memory organization with modified LGP and local BL multiplexer.

while red is the last set (way-$(S-1)$). BLADE (Figure 2.7-a) multiplexes the global bit-lines in the periphery, preventing data that occupy different sets from interacting. In the proposed architecture, by multiplexing the local bit-lines instead of the global bit-lines, all the sets can now share the same LGP. Consequently, by providing independent control for each local BL multiplexer, data belonging to different sets can couple into the same GRBLs, as depicted in Figure 2.7-b, by the blue and red dashed lines.

This solution reduces the circuit complexity and improves energy efficiency. The overall number of global bit-lines for read and write are divided by $S\times$, where S is the number of sets. This decreases the energy of accessing data as fewer bit-lines are pre-charged. Furthermore, this work greatly enhances the area's efficiency. Whereas BLADE needs S LGP blocks per bit-column, the AA-BC employs just one.

2.3.1.1 Embedded shift

The fact that only one LGP block is shared between all the sets of the LGs allows modifications on the LGP design with limited effect on the area efficiency. Indeed, modifying the LGP, especially the read port, brings interesting opportunities to accelerate multiplications, as the operations at the LGP level can be performed in parallel with the adder in the BCU.

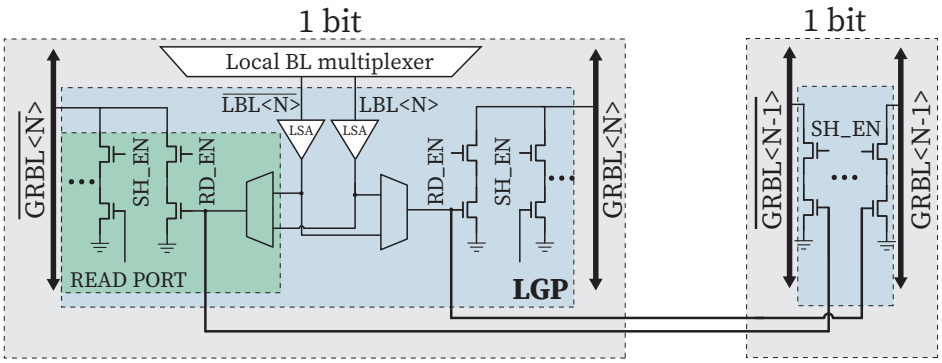


Figure 2.8. Modified LGP circuit with extended read port, enabling shift and negation. The transmission gate responsible for the write operation is omitted from the figure.

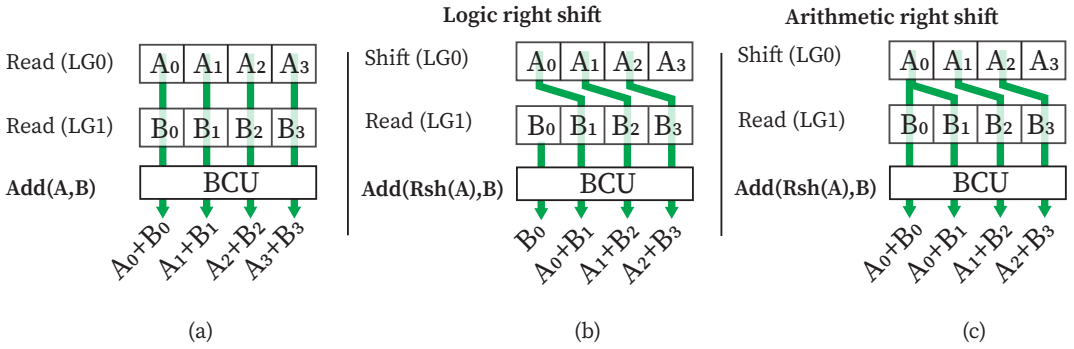


Figure 2.9. Data path of addition with (a) both LGs performing read access, (b) LG1 performing a logic right shift, and (c) LG1 performing an arithmetic right shift.

Figure 2.8 shows the design of the new LGP. There are two additional elements with respect to the read port presented in Figure 2.3-b. First, a new pair of NMOS transistors is connected to the GRBLs. These are responsible for the embedded shift. Second, two multiplexers at the LSA output are linked to the implementation of two's complement arithmetic, and it is discussed in Section 2.3.2.

A conventional read operation uses the path controlled by the Read Enable (RD_EN) signal and the LSA output of the same bit. Embedded shifts instead rely on the path controlled by the Shift Enable (SH_EN) signal and the LSA output of a *neighbour* bit. Figure 2.9 shows the addition of two words and the data path for a 2-LG BC architecture. In the examples, the LG1 is accessed by the read signal. Otherwise, the LG0 performs a read, a logic right shift, and an arithmetic right shift for the first, second, and third examples, respectively. Right shifts are used in signed two's complement multiplication instead of the left shifts used by the unsigned integer multiplication, such as in BLADE (Table 2.1).

	01000000	10001010
Unsigned integer (Uints) 2^7 2^6 2^5 2^4 2^3 2^2 2^1 2^0	$2^6 = 64$	$2^7 + 2^3 + 2^1 = 138$
Two's complement (Q1.7) -2^0 2^{-1} 2^{-2} 2^{-3} 2^{-4} 2^{-5} 2^{-6} 2^{-7}	$2^{-1} = 0.5$	$-2^0 + 2^{-4} + 2^{-6} = -0.921875$

Figure 2.10. 8-bit data encoding represented in unsigned integer and in fixed-point $Q1.7$.

2.3.2 Two's complement arithmetic

Two's complement fixed-point binary representation is a method of coding signed integer and fractional numbers in binary form. In this scheme, the most significant bit (MSB) of the binary number indicates the sign of the number, with 1 indicating a negative number and 0 indicating a positive number. The binary representation of a negative value is calculated through the two's complement operation of the equivalent positive value. It consists in inverting all the bits of the respective positive value and summing up one.

The interest of this notation is that by scaling the operands to have 1 bit for the integer part and n bits for the fractional part (a format commonly indicated as $Q1.n$), the values are always in the range $[-1,1)$. Thus, multiplying two values in this range will limit the result to the exact same range. However, this property does not hold for additions. Therefore, the accumulator may overflow with the $Q1.n$ notation while performing MAC operations. This is discussed and solved in the Section 2.4. In Figure 2.10, two binary values represented in both unsigned integer and signed fixed-point are depicted to show the difference between the two encoding schemes.

Performing an exact multiplication with signed two's complement fixed-point values confined in the $[-1,1)$ range will also require more bits to fully represent the product, such as the unsigned integer multiplication. However, the binary representation of the signed fixed-point multiplication product grows towards the least significant bits since its MSB will always represent the value -1 (in $Q1.n$). Therefore, by truncating the LSBs, the product of the fixed-point multiplication can be approximated using the same bit-width of the operands.

2.3.2.1 BC support to two's complement multiplication

The algorithm of the signed fixed-point multiplication may require the calculation of bit-wise negation if one of the operands is negative. To this end, the LGP also embeds 2-to-1 multiplexers connected to both outputs of the LSA (Figure 2.8). The

multiplexers invert the LBL and $\overline{\text{LBL}}$. Finally, the two's complement operation is accomplished by performing bit-wise negation in the LGP and asserting the carry-in of the least significant bit in the BCU.

Moreover, shifts must be arithmetic while performing signed fixed-point multiplication, meaning that the right shift must extend the value sign. In the proposed BC architecture, this is supported by the embedded shift. At the LGP of the most-significant bit, the output of the multiplexers is connected to both the read path and the shift path. Thus, when a shift is performed, the operand's most significant bit is both read in its own GRBL and shifted to the neighbor GRBL, such as Figure 2.9-c shows.

Finally, considering the proposed read port with one embedded shift, all the iterations of the signed fixed-point multiplication between an IMO and a BO can be executed in a single clock cycle. The multiplication on the BC architecture can be performed by employing sequences composed by the following two instructions:

(i) Addition of two operands, both arithmetically right-shifted (i.e., with sign extension) by one bit. One operand is the partial product ACC, while the second is either 0 or the IMO, depending on the b_n bit.

(ii) Addition of two operands, where one operand is the partial product ACC, and the other is either 0 or the 2's complement of IMO.

Considering that BO's bit-width is N bits, the instruction (i) is performed at every iteration step from bit 0 to bit $N-1$ of BO, while operation (ii) is used for bit N of BO (its most significant bit).

Figure 2.11-a shows an example of multiplication among two two's complement numbers in the Q1.7 and Q1.4 formats, with the result in Q1.7. Figure 2.11-b shows the operations required to perform the multiplication in the case of only supporting shifts of one bit. Notice that the bit-width of the partial products does not increase at each iteration, and no overflow occurs. Instead, truncations induce small errors in the computed product (0.4% in the example).

2.3.2.2 Accelerating multiplications

With a bigger Number of Embedded Shifts (NES), speed-ups in the computation are expected at the cost of added complexity in the read port. Indeed, in the case of NES = 2, 2 bits of the BO can be processed in one iteration, provided that the first

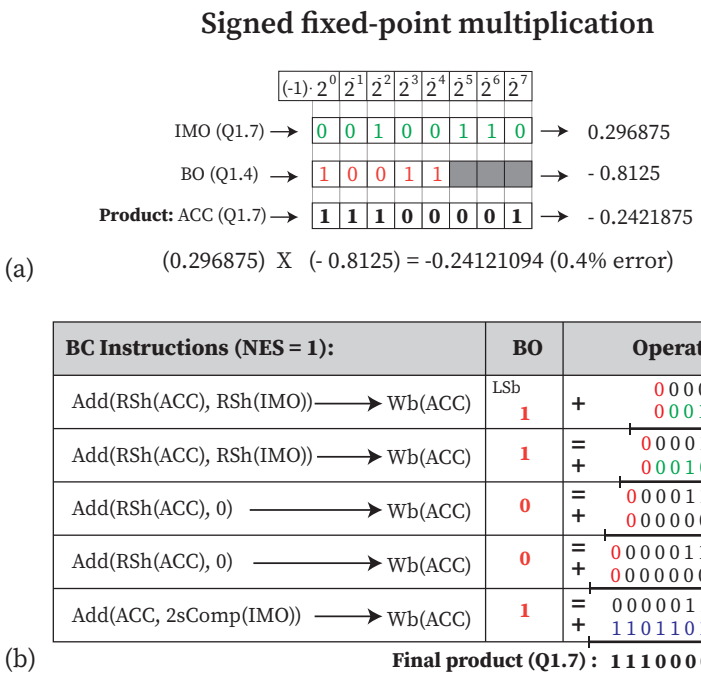


Figure 2.11. (a) Two's complement fixed-point multiplication example between the IMO expressed in Q1.7 and the BO expressed in Q1.4. The BC instructions for (b) NES = 1. RSh operations are right shifts.

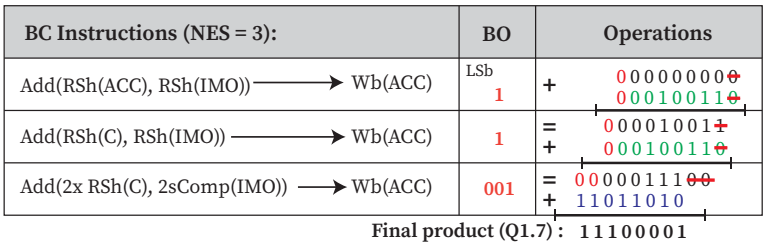


Figure 2.12. BC instructions NES = 3.

one is equal to zero ("00" and "01"), since in this case only one addition, or none, is required after two shifts. For NES = 3, BO sequences of bits with two leading zeroes ("000" and "001") can be processed in a single clock cycle.

Figure 2.12 shows the BC instructions to execute the same multiplication as Figure 2.11-b, but on a BC array with NES = 3. In this configuration, only 3 operations are required (instead of 5 when NES = 1), resulting in a speed-up of 67%.

Finally, Figure 2.13 show the read port connections of the MSBs when considering NES = 3 and a 16-bit word memory. In this figure, only the read port and the multiplexer output are depicted for simplicity. Since the sign is extended in a right shift, the MSB

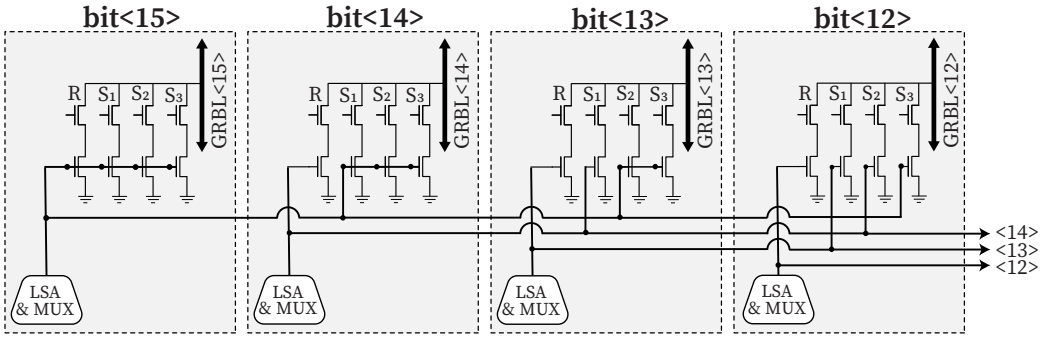


Figure 2.13. Embedded shifts connection of the MSBs for a 16-bits array and NES = 3.

read port (bit<15>) connects the multiplexer output signal to the read path and all the other shift paths. In bit<14>, this same signal is connected to the three-shift path. In bit<13>, the first shift is connected to bit<14>, while the second and third are also connected to bit<15>. The connections follow a regular pattern from the bit<12> onwards.

2.3.3 Experimental setup and results

The electrical validation used the same environment and technology as presented in Section 2.2.2. Otherwise, the systems analysis is performed through analytical behavior models, and the reported gains sum up to the gains reported in Section 2.2.2.2. In this section, I analyze the overheads brought by the extra circuitry necessary to accelerate and enable signed two's complement multiplications. Moreover, I consider the inference of the CNN model AlexNet [7] to explore the effects of the MISB strategy on energy and run-time profiles for 1, 32, and 128 subarrays. The effects on CNN accuracy are covered in the next section in the context of MAC overflow management.

2.3.3.1 Impact of embedded shifts

I designed an analytical, behavioral model of the multiplication algorithm to evaluate the acceleration of the embedded shifts. Since BLADE does not perform signed two's complement fixed-point multiplication, the analytical model considers an unsigned integer multiplication for comparison purposes. The model calculates the number of cycles required to perform the multiplication for a given set of parameters (NES, BO's bit-width, and value). Then, I extracted statistical data considering all the possible values (i.e., from 0 to $2^{16} - 1$ for 16-bit word operands) to assess the cycle count distribution. Figure 2.14 shows the normalized representation.

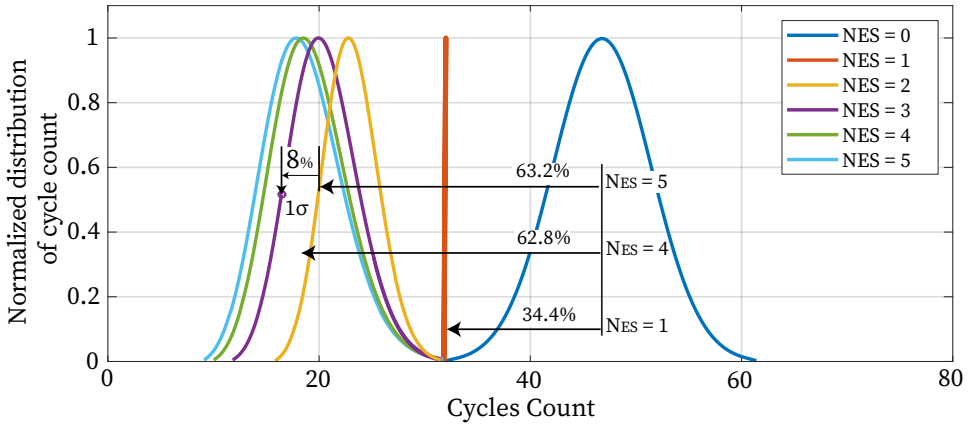


Figure 2.14. Cycle count distribution of 16bits multiplications.

For NES = 0 (equivalent to a naive BLADE implementation without the add-forward Add_{n1}), each BO bit requires 2 or 4 cycles. If b_n is '1', 4 cycles are required (Add, write back, left shift, write back), while if b_n is '0', it only takes 2 (left shift and write back). Thus, the distribution spans from 32 to 64 cycles. For NES = 1, similar to BLADE's performance employing the add-forward at the BCU, the number of operations is equal to the size of the operand regardless of the data structure. Thus, the distribution centers in 32 cycles.

For NES higher than 2, the distribution's right tail always equals 32 cycles, representing the worst case of multiplier ($B = 2^{16} - 1$) when all the bits are equal to 1. The average, however, decreases accordingly to higher values of embedded shift. The accelerated patterns become increasingly scarcer for each successive embedded shift, lessening the potential gain. The difference between the average cycle count for NES = 4 and 5 is less than half a cycle.

While the average gain for NES> 2 exceeds 60%, it must be noted that the computation time is highly data-dependent. For example, some CNNs layers have data distribution very skewed towards zero, thus making the total gain of multiplication higher than the average. As a reference, for NES = 3, considering a data structure where the accumulated cycle count of several multiplications shifts from the average to one sigma left, there is an extra gain of 8%.

2.3.3.2 Electrical characterization and area estimations

Figure 2.15 shows the AA-BC's specifications (area, delay, energy) compared to BLADE. By attaching more transistors in the read port, the overall parasitic capacitance of the GRBL and of the LSA output increases, raising energy consumption and reducing

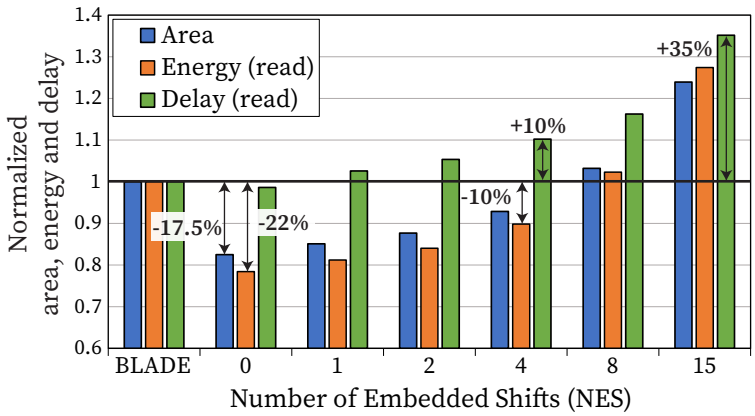


Figure 2.15. Area overhead, read delay, and energy evolution of the proposed work normalized with BLADE (lower is better).

performance. Thanks to the GRBL optimization enabled by using a local BL multiplexer, this effect is compensated. Therefore, for NES = 0, the energy is reduced by 22% for one operation. Otherwise, beyond NES = 7, the energy consumption of AA-BC overcomes BLADE.

Concerning the read delay, for NES = 0, the delay is similar to BLADE (e.g., less than 2% reduction). For higher values of NES, however, AA-BC surpasses BLADE because the path covered by the signal is longer. Each embedded shift increases the signal propagation distance in 2μm (it could be noted that increasing the LSA drive may mitigate this effect while increasing the LG area). The read delay is 10% higher for NES = 4, and it exceeds 35% for NES > 15. Finally, I extract the corresponding area from the layout and show a 17% density improvement for NES = 0, but AA-BC becomes larger than BLADE for NES > 7.

2.3.3.3 System level assessment

Figure 2.16 shows the average cycles gain and area efficiency in the function of NES. For the same multiplication performance as BLADE (NES = 1), I show a 9% higher area efficiency. On the other hand, Figure 2.16 also shows a gain saturation that can be explained by the fact that after a point, more embedded shifts only accelerate a marginal portion of the possibilities of words, that is, saturating the average gain. Compared to BLADE, AA-BC shows a 44% cycle count reduction.

Overall, I show several non-aligned trends: (i) the average multiplication performance gain (i.e., cycle count) tends to saturate with NES. (ii) The area overhead and energy are beneficial for a few shifts (i.e., less than 7) but become disadvantageous beyond.

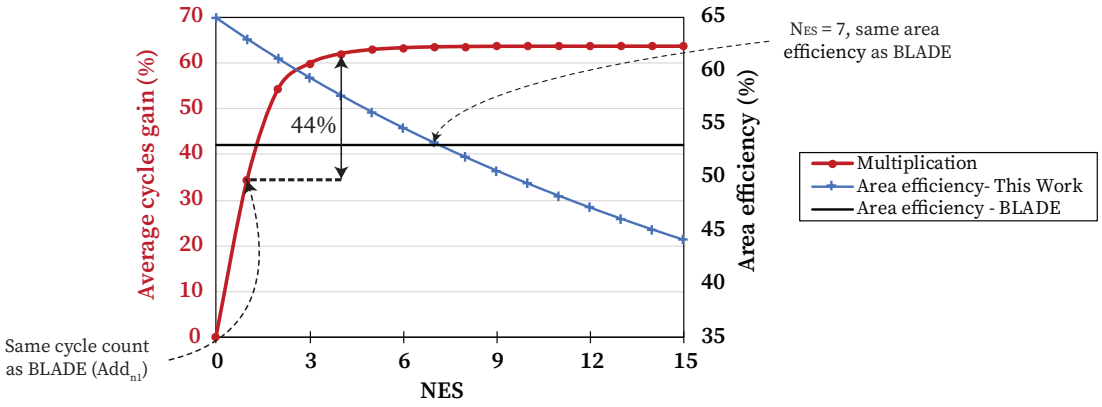


Figure 2.16. Cycle gain and area efficiency per NES.

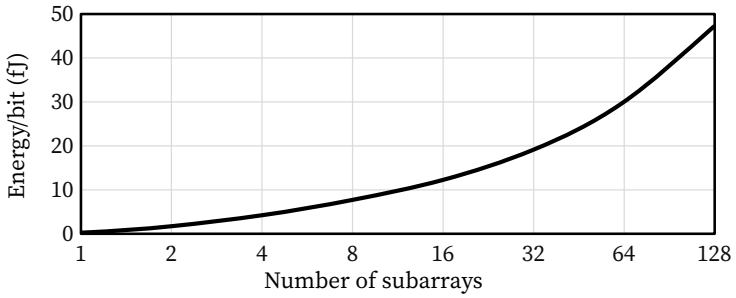


Figure 2.17. Energy of 1-bit data transfer in the function of the amount of the H-tree.

(iii) The operation delay degrades with the number of embedded shifts.

To evaluate the effect of array-level parallelism strategies, I implemented the architecture with NES = 3 since it presents a good trade-off between efficiency, area overhead and complexity to control. With this configuration, the AA-BC is 10%/13% more area/energy efficient but 7% slower than BLADE.

2.3.3.4 Array-level parallelism

The energy cost of data transfers from the array periphery to the subarray grows with bigger H-trees. Since the data has a longer path to cover, the parasitic attached to the line consumes more energy. Figure 2.17 shows the cost of transferring 1-bit data from the array periphery to the subarray. To be able to assess the effect of array-level parallelism correctly, I modeled the size of the H-tree based on the number of subarrays and lines (output, input, and control signals) connected in the subarrays. I extracted the energy cost using RC networks. With 32 subarrays or more, the cost of transferring data surpasses the cost of computing in-memory.

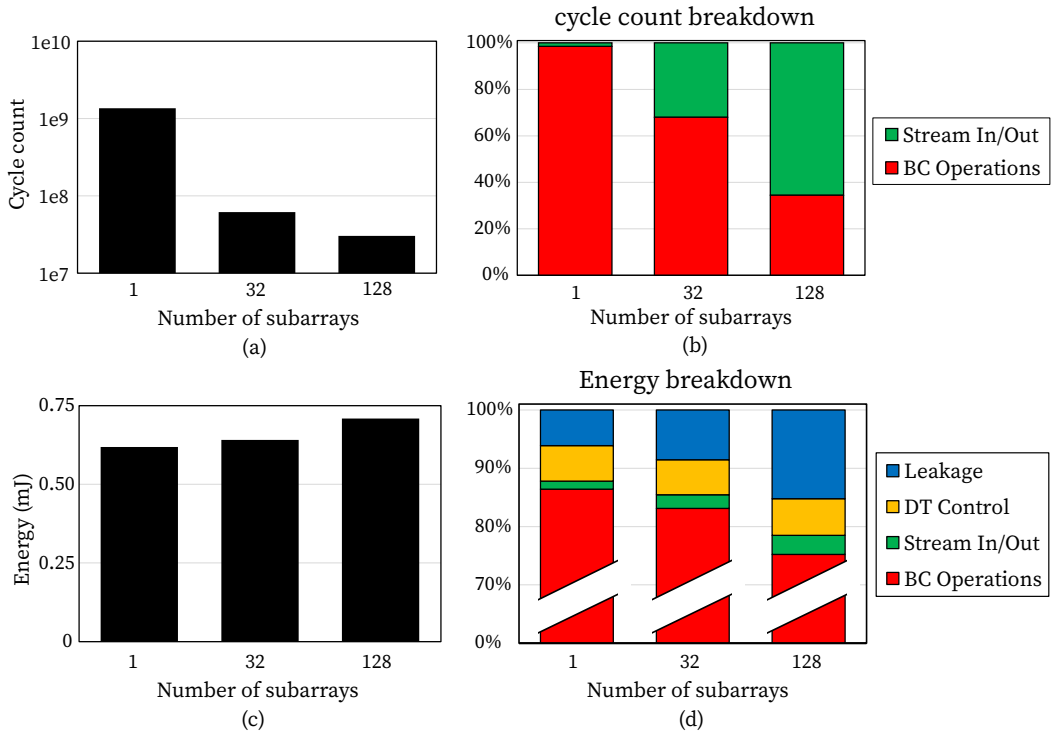


Figure 2.18. (a) Absolute cycle count and (b) its breakdown of multiple subarray architecture for an AlexNet inference. (c) Absolute inference energy and (d) its breakdown.

Figure 2.18-a and -c report absolute cycle count and energy requirements to perform an AlexNet [7] inference. Figure 2.18-b and -d, otherwise, show the proportional breakdowns. Figure 2.18-a indicates the cycle count scales down with the number of subarrays, as the workload is effectively distributed. The CNN run-time is accelerated by 22× (45×) when employing 32 (128) subarrays with respect to a single one. As more subarrays are considered, the potential gain (32× for 32 subarrays and 128× for 128 subarrays) is muted by data transfers. When the workload is entirely run on a single subarray, 99% of the clock cycles are used to perform MAC operations. However, using 32 or 128 subarrays reduces this percentage to 68% and 35% (respectively) because data transfers are performed sequentially. At the same time, the parallelism of BC operations grows linearly with the number of subarrays.

Using a high number of subarrays does not scale the energy as the performance. In fact, it slightly degrades the energy efficiency. The CNN shows 14.4% less energy efficiency with 128 subarray configurations with respect to one subarray. Indeed, the number of subarrays does neither influence the energy cost of BC operations nor the number of BC operations required by an inference. The slight increase in energy consumption is due to leakage energy consumed by subarrays during data transfers and by the larger and more energy-hungry H-tree required to connect them.

Finally, the architecture achieved an energy efficiency of 1.78 TOPS/W while executing Alexnet.

2.4 MAC-BC: MAC accelerator with bit-line computing

The overflow issue was solved in Section 2.3.2.1 in multiplications by truncating the LSBs. However, overflow still may happen in addition. Addition of two values confined in the range $[-1,1)$ may result in a value outside of this range. In this case, the signed-digit representation changes from $Q1.n$ to $Q2.n$. Even though it is still possible to use the right shift to concatenate the data and represent it in $Q2.(n-1)$, specialized circuitry would need to be put in place to keep track of the signed-digit representation. Moreover, each subarray may have a different overflow in a multi-subarray configuration, complexifying the data management.

Since most edge device workloads are based on MAC operations, supporting this operation is crucial to enable edgeAI. To this end, in this contribution, I propose a BC-based architecture that fully supports and manages the overflow in MAC operations. The MAC-BC architecture builds upon all the developed methods presented in the previous sections, such as the array-level parallelism support for the MISB strategy and the modified read port that supports embedded shift and negation. Moreover, it embeds a MAC register that has twice the size of the IMO, occupying two words. Thus, for an array storing 16-bit words, as I consider in this work, the data stored on the MAC registers are represented in $Q16.16$. Moreover, in this contribution, I introduce the support for word-level parallelism, which means that the subarrays support either 16-bit or 8-bit operations. For the latter, the data stored on the MAC accumulator is represented in $Q8.8$. To guarantee the correct operation, the data needs to be pre-processed and scaled to be in the correct range.

The MAC-BC is also easier to integrate into conventional SRAM arrays concerning BLADE and AA-BC. Firstly, MAC-BC does not rely on local groups, avoiding interleaving standard memory cells with the modified LGPs. For this contribution, MAC-BC concentrates all its computing periphery in a single region called the Processing Element (PE). Since the PE performs BC operations with only one word stored in the memory, while the other resides in the PE registers, a single word-line decoder is necessary, differently from its counterparts based on local groups that require two decoders.

2.4.1 Equivalent NMC implementation

The ease of integration of MAC-BC architecture is comparable with NMC which can be digitally implemented and live in the vicinity of an SRAM array. In order to validate that a BC-based architecture can outperform a digital one and justify the effort to design such an architecture, I implemented an equivalent NMC architecture. As presented in Figure 2.19-a, the PEs from the MAC-BC and NMC must have the following elements to be equivalent: MAC registers, process up to 3 bits of BO (NES = 3) in a single clock cycle and be capable of supporting positive and negative overflow. For comparison reasons, I first show the implementation of an equivalent NMC architecture.

Figure 2.19-b shows a block scheme of the equivalent NMC PE. The PE reads the word from the output drivers of the SRAM array and computes the multiplication based on the MISB strategy. The PE embeds a 32-bit adder and the logic for the IMO's bit-wise negation and right shifts. Moreover, it allows right shifting by up to three bits of the partial product value (ACC). Three registers store a copy of an SRAM memory word (IMO reg), the partial product (ACC reg), and the MAC value (MAC reg) being computed.

2.4.1.1 NMC overflow avoidance

At run-time, when performing a multiplication, the first adder operand is selected either as '0', the right-shifted IMO, or its 2's complement (when processing the BO most significant bit[1]). The second adder operand is the content of the ACC register, properly right-shifted by one, two, or three bits. Once a multiply operation is finished, accumulation is performed as an addition between the ACC and MAC registers. Overflow is avoided in accumulations by sizing the MAC register to 32 bits. Finally, when all multiply-accumulate operations required for a dot product terminate, the two parts of the MAC register corresponding to the least significant and most significant 16 bits are read out.

2.4.1.2 NMC data level parallelism

Both 2×8-bit and 1×16-bit IMO operands are supported by configuring the connections in the carry chain of adders so that in 2×8-bit mode, carriers do not propagate in-between subwords boundaries. Moreover, sign extension (instead of shifting) is performed between the 7th and the 8th bit in shifters.

[1] In this case, the carry-in of the adder is set to '1' to perform the arithmetic negation of the IMO value correctly.

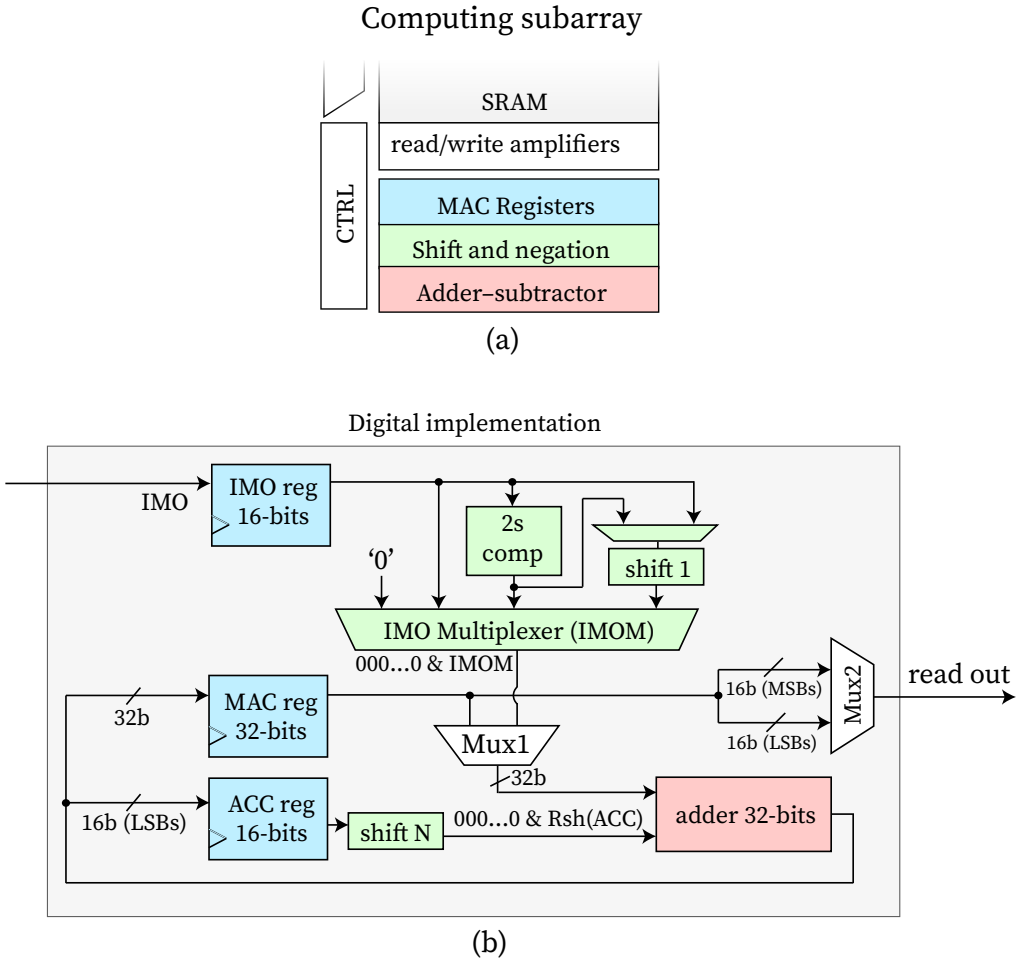


Figure 2.19. (a) Generic basic PE composed of the IMO pre-processing circuit, registers for the multiply and accumulate operation, and the adder. (b) digital implementation of the NMC architecture.

2.4.2 MAC-BC implementation

Bit-line computing can be applied whenever bit-lines (BL and $\overline{BL}$) are employed to access the bit value stored in a memory cell, regardless of the cell implementation. I leverage this opportunity in the BC design to allow computations between in-memory operands (stored in SRAM cells) and dedicated registers containing partial products and accumulation values (employing flip-flops).

The circuit of a 1-bit column of the MAC-BC architecture exploiting bit-line computing is provided in Figure 2.20-a and further detailed in Figure 2.20-b. Similarly to the NMC design, the MAC-BC architecture uses registers to store the ACC and MAC values.

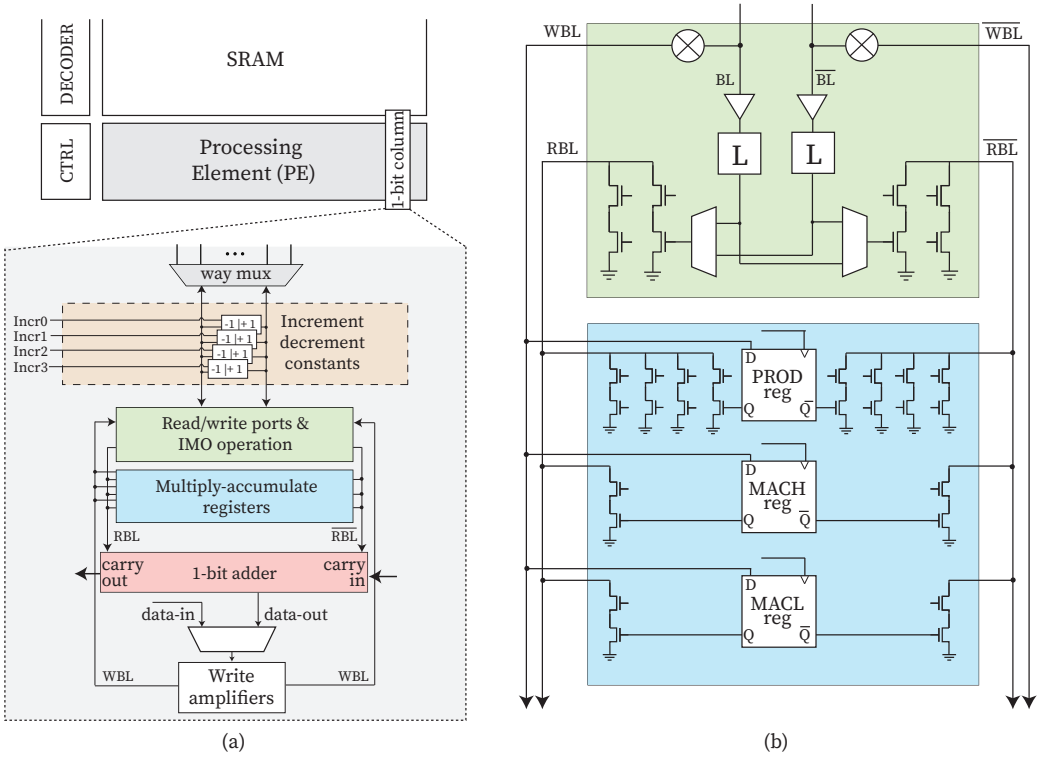


Figure 2.20. (a) Block diagram of the MAC-BC architecture, focusing on a 1-bit column of the PE. (b) Detailed circuit of the read ports and registers. $RBL \backslash \overline{RBL}$, $WBL \backslash \overline{WBL}$ refer to the read and write path of the bit-lines, respectively.

However, since all words must share the same bit-width to perform bit-line computing, the MAC register is split into two parts hosting the most significant bits (MACH) and the least significant ones (MACL), respectively.

When performing multiplications, as the IMO is first read from the memory array, it is also latched after the LSA to avoid repetitive accesses to the same memory words (and the associated energy cost). By using latches (rather than flip-flops), the fetching of an in-memory operand and the first shift-add operation can be performed in the same clock cycle, speeding up execution with respect to the NMC design.

Since results of in-memory operations are stored in registers, writing back to the SRAM memory array is avoided. Therefore, the write bit-lines (WBL) can be asserted in parallel with the pre-charging of read bit-lines (RBLs) since flip-flops' read/write paths are separated. Otherwise, in SRAMs, this strategy would lead to short circuits. This results in a 2× acceleration of multiply operation compared to AA-BC.

The MAC-BC case performs accumulation in two cycles (to update MACH and MACL). Otherwise, the NMC requires only one cycle. However, since the MAC-BC computes

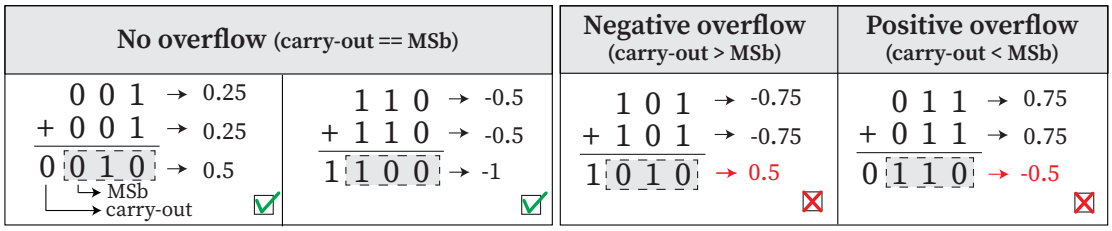


Figure 2.21. Overflows from additions of two values represented in Q1.2 format.

the first multiplication operation as part of data access, the MAC-BC and NMC designs have equal performance.

2.4.2.1 MAC-BC overflow avoidance

In-memory shift-add operations are employed during multiplications, relying on fixed-point representation to avoid overflow, as described in Section 2.3.2. In the case of addition, the result of a product stored in the ACC register is first added to the content of the MACL register using the in-memory adder. Note that this operation may induce positive or negative overflow. As depicted in Figure 2.21, the type of overflow is dictated by the carry-out and sum bits of the input MSbs: when these are equal, no overflow occurs, while, if they differ, a positive (“01") or negative (“10") overflow occurs.

Those bits govern the state of the MACH register. In case of no overflow, its value is retained. Instead, in case of overflows, the sum and carry-out signals of the MSb of MACL registers are used to index two constants, corresponding to the values ‘-1’ and ‘+1’. An in-memory addition is then triggered between the selected constant and MACH, with the result being returned to MACH.

2.4.2.2 MAC-BC data level parallelism

As in the NMC design, the MAC-BC architecture can also support word-level parallelism. In the 2×8-bit word mode, the architecture processes two subwords (subword-1 spanning from bit<15> to bit<8>, and subword-0 spanning from bit<7> to bit<0>). The carry-chain of the adder must be segmented in the edge of the two subwords (between the 8th bit and the 7th bit) in order to enable autonomous additions. Moreover, the carry-in of subword-1 (bit<8>) must be programmable (such as the carry-in of subword-0, bit<0>) in order to perform two’s complement operations. To this end, a multiplexer is added at the edge of the two subwords.

The embedded shift strategy requires special attention when routing to allow the

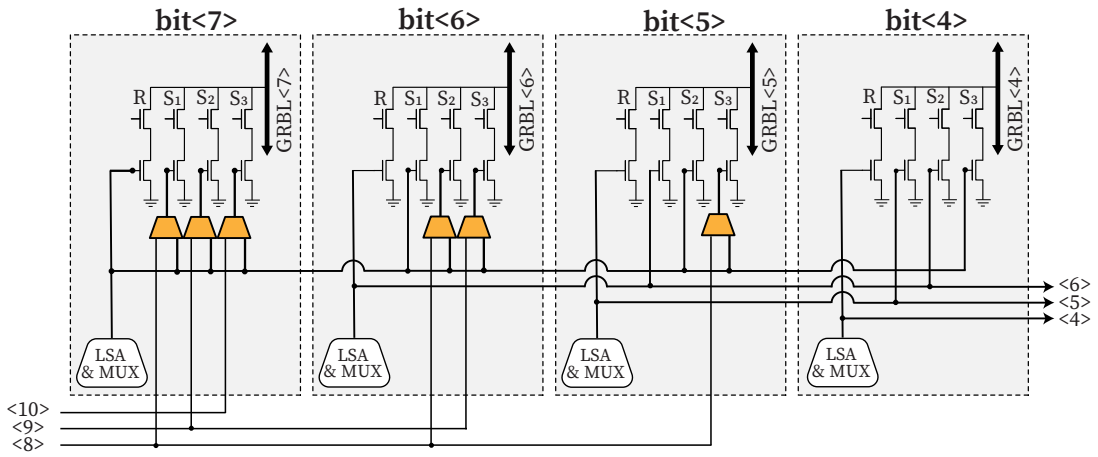


Figure 2.22. Word-level parallelism support for 1×16bit and 2×8bit modes, considering NES = 3.

correct usage in both 1×16-bit and 2×8-bit modes. Such as shown in Figure 2.13, the most-significant bit of a word is connected to several discharging paths of its own and neighbor's read ports to perform sign extension while shifting. Section 2.22 shows the circuit of the subword-2 on the extended read port. Six extra multiplexers are used to control the signals between subword-1 and subword-2.

Finally, when updating the MACH value to account for positive/negative overflows of MACL, MAC-BC provide a mechanism that allows managing two sub-words separately. First, employing a separate read enables for the increment/decrement constants (Figure 2.20) for each subword. These constants are only asserted when a MACH subword increment/decrement must be performed upon an overflow in MACL. Then, providing four constants, encoding the values $\{(+1,+1),(+1,-1),(-1,+1),(-1,-1)\}$ (Table 2.2) to manage all positive/negative MACH update combinations across sub-words.

Table 2.2. Increment/decrement constants.

Increment WL	**subword-1**	**subword-2**
Increment 0	**0 0 0 0 0 0 0 1**	**0 0 0 0 0 0 0 1**
Increment 1	**0 0 0 0 0 0 0 1**	**1 1 1 1 1 1 1 1**
Increment 2	**1 1 1 1 1 1 1 1**	**0 0 0 0 0 0 0 1**
Increment 3	**1 1 1 1 1 1 1 1**	**1 1 1 1 1 1 1 1**

2.4.3 Experimental setup and results

I designed the SRAM array using 28nm TSMC CMOS technology, implementing high-density ($0.127\mu m^2$) memories using 6T SRAM cells. Unlike BLADE and AA-BC, the memory is considered an array of SRAM cells without segmentation (LGs). As a test vehicle for both MAC-BC and NMC, I considered memories composed of a varying number of 2KB subarrays, each organized as 1024 words of 16 bits each. Four words are bit-interleaved in each memory row so that the implemented array circuit presents 64BLs and 256WLs. This configuration allows up to 2.2GHz read and write memory operations. This timing constraint was used to design and optimize MAC-BC and NMC solutions.

The MAC-BC architecture was implemented as a full custom design, the same as presented in Section 2.2.2. On the other hand, the NMC architecture was implemented as a semi-custom design. Its behavior was described in RTL and synthesized using standard digital cells from the TSMC 28nm PDK. Moreover, the energy characterization was performed with realistic test vectors to extract switching activities.

Finally, to evaluate the impact of overflow strategies in the Quality of Service (QoS) of edge device workloads, I consider the accuracy of AlexNet [7] (evaluated on the CIFAR-100 dataset) under the proposed MAC-BC architecture and two baselines: *Saturation* only employs the MACL register. Its content is saturated to the maximum or minimum representable value in the event of positive or negative overflows, respectively. In the second baseline architecture (*8-bit IMOs*), The range of IMOs is limited to prevent overflows when performing accumulations. A similar strategy was considered in BLADE. In this baseline, all IMOs are quantized as 8-bit values but sign-extended to 16 bits, hence preventing the use of 2×8-bit IMOs to parallelize computation (i.e., 8 MSbs remain unused to prevent overflow).

AlexNet was optimized and mapped for the MISB strategy for the presented results, described in Chapter 4.

2.4.3.1 Circuit analysis of area and energy

The area of an ordinary SRAM subarray (i.e., a memory subarray with no computing capabilities) is reported on Table 2.3 and is used as a baseline in the analysis to evaluate the overhead of MAC-BC and NMC implementations. As expected, more than 85% of the memory footprint is due to bit-cells ($2129.9\mu m^2$ out of a total area of $2477.16\mu m^2$), while read/write ports and word-line amplifiers have a minor impact on the full subarray size. An area breakdown of the MAC-BC and NMC implemented

Table 2.3. SRAM subarray area breakdown. Area values are expressed in μm^2.

SRAM Subarray	Area
Bit-cell array	2129.90
Read/write ports	81.02
Word-line amplifiers	266.24
Total	2477.16

Table 2.4. MAC-BC, and NMC Processing Element (PE) area breakdown. Area values are expressed in μm^2.

MAC-BC PE	Area	Overhead	NMC PE	Area	Overhead
Registers	190.70	7.7%	Registers	169.13	6.8%
16-bits adder	39.01	1.6%	32-bits adder	85.29	3.4%
Shift/negation	62.42	2.5%	Shift/negation	111.29	4.5%
Total	292.13	11.8%	**Total**	365.71	14.8%

designs is presented in Table 2.4. The MAC-BC PE has an area 25% smaller than the NMC PE. The left columns show the overhead area for implementing the MAC-BC architecture's described computing capabilities. The area increment is limited to 11.8% and is mainly due to the multiply-accumulate registers illustrated in Figure 2.20. Instead, the 16-bit adder and shift/negation circuits have minimal impact on the total overhead. Similar considerations hold for the NMC implementation, where the slightly higher area footprint of the adder and the shift/negation unit result in a 14.8% overhead concerning the baseline SRAM subarray.

Table 2.5 shows the energy comparison between the SRAM array, the MAC-BC, and NMC PEs. The SRAM subarray requires 491.6 fJ for a 16-bit read operation and 363.6 fJ for a 16-bit write operation while consuming 88.9 fJ as static energy (leakage). The MAC-BC PE consumes 238.6 fJ to compute a shift-add operation, or 14.8 fJ/bit, representing around 50% reduction with respect to BLADE and AA-BC since the cost of memory access is avoided in most of the cycles. Moreover, the energy consumption of the MAC-BC PE is 16% lower than the NMC PE.

It can be seen that the NMC adder and shift/negation logic consumes more than 2× power compared to its MAC-BC peers. The reason is that the MAC-BC PE extracts logic operations passively from the pre-charged bit-lines, reducing energy consumption. However, MAC-BC registers consume 25% more energy than NMC because the MAC-BC solution requires more registers to manage the parallelization of write-back/pre-charge, as explained in Section 2.4.2.

Table 2.5. SRAM subarray (256WL, 64BL), MAC-BC, and NMC Processing Element (PE) energy breakdown.

SRAM Sub	Energy (fJ)	MAC-BC PE	Energy (fJ)	NMC PE	Energy (fJ)
Read	491.6	Registers	155.8	Registers	127.3
Write	363.6	16-bits adder	31.9	32-bits adder	64.2
Leakage	88.9	Shift/negation	51.0	Shift/negation	83.7
		Total	238.6	**Total**	275.2

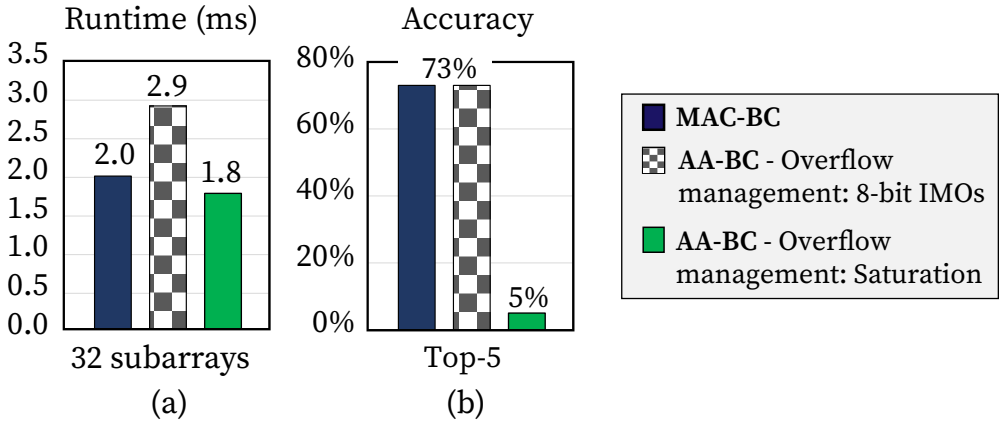


Figure 2.23. (a) Absolute inference run-time and (b) accuracy achieved of AlexNet executed in three overflow management strategies.

2.4.3.2 Application level assessment

Figure 2.23 compares my approach with the two alternative overflow handling techniques discussed in Section 2.4.3. The analysis presented in this section focuses on the performance of the inference and the accuracy achieved. The bar plots on the figure's left show the total run-time, expressed in milliseconds, of the three strategies running in a MAC-BC architecture employing 32 subarrays. In this regard, the results assume that a unique BO is broadcasted to all the arrays at each BC operation (MISB strategy). Finally, the bar plots on the right of Figure 2.23 depicts the Top-5 accuracies achieved by the three different approaches. The results refer to both the MAC-BC and the NMC implementations, as the two designs require the same number of clock cycles for executing MAC operations. The results reflect that both the *Saturation* and *8-bits IMOs* baselines employ only one cycle for both accumulations (since they only employ one MAC register). At the same time, the presented solution requires two (to update MACL and MACH).

The proposed solution is 45% faster than the *8-bit IMOs* baseline. Performance gains are mainly due to the data parallelism possibilities of the MAC-BC solution, which

are instead prevented in the *8-bit IMOs* alternative, as the 8 MSBs of each word must not be used to ensure overflow-free accumulations. This also translates into a non-optimal use of memory for the *8-bit IMOs* implementation, as 50% of memory words cannot be used for computation.

Compared to the *Saturation* solution, MAC-BC results in a 12% increase in run-time. The reason is that saturation operations have no impact on run-time. This assumption puts the *Saturation* baseline at an advantage because, in the proposed implementation, one additional BC operation is required to update the value of the *MACH* register after each MAC operation. However, saturating the accumulator adversely affects accuracy, dropping from 75% to 5%, indicating random predictions in the 100-class classification problem of CIFAR-100. Hence, this approach is not applicable from a QoS perspective.

2.5 Conclusion

In this chapter, I have presented three BC-based architectures designed to accelerate edge AI workloads. I have presented BLADE, the starting architecture for the enhancements presented in this section. BLADE presents very exciting results to accelerate edge AI. In the context of CNNs, BLADE performs up to 3× faster than NEON, a SIMD accelerator found in many edge devices. However, BLADE has two main drawbacks: slow multiplications and data overflow.

I have proposed two architectures (AA-BC and MAC-BC) to tackle these limitations and fully bridge core edgeAI applications on BC-based architecture. AA-BC shows an increase in multiplication efficiency of 44%, on average, with four embedded shifts compared to BLADE. Since BLADE relies on unsigned integer arithmetic, it may induce data overflow and loss of CNN accuracy. To this end, overflow has been solved in multiplication with AA-BC, introducing signed two's complement fixed-point arithmetic and scaling the operands properly. Moreover, the overflow in additions was solved in the MAC-BC architecture, enabling MAC operations to be completely reliable. This architecture was compared with a digitally implemented equivalent NMC architecture and the MAC-BC showed 25% and 16% lower area overhead and energy consumption. Moreover, compared to BLADE and AA-BC, MAC-BC presents 2× faster multiplication since the write-back cycles are avoided and 2× lower energy operations, as it avoids reading from memory in every cycle. Thus, resulting in a 4× increase in energy efficiency.

3 Energy efficiency boost: innovations in bit-line computing architectures

In-memory computing architectures are among the most promising solutions to increase the energy efficiency of edge Artificial Intelligence (edgeAI) thanks to a considerable reduction in data transfers. However, the ever-increasing requirements of AI call for solutions to increase these architectures' energy efficiency even further.

In this chapter, I cover strategies at the architecture and technology level to this end. Firstly, I present a hybrid SRAM-RRAM architecture (Section 3.3) in which the data is mapped and accessed by matching the data behavior with the SRAM and RRAM features, ultimately reducing even more data transfers. Another strategy that I present in this chapter is voltage scaling. Significant energy savings are realized by tuning the voltage since there is a quadratic relation between energy and voltage. However, memories are susceptible to lower voltages as they degrade read, write, and retention margins. To tackle this issue, first, we explore how to minimize the effect of variability, ultimately reducing energy figures and increasing the endurance of the RRAM. Then, I also present how an in-SRAM computing architecture (Section 3.4) can cope with less reliability due to voltage scaling thanks to an error detection and mitigation strategy. Finally, these strategies significantly increase energy efficiency without losing AI's Quality of Service (QoS).

3.1 Introduction

Nowadays, many cloud-based applications, ranging from social media recommendations to business informatics, are AI-driven. However, such applications (at least their inference phase) are not amenable to be executed in the cloud because (a) they must present a high level of responsiveness, (b) they rely on a massive volume of locally-acquired data, whose streaming from device to cloud is extremely energy costly, and (c) they often process personal data, for which security and privacy are major concerns.

Hence, most solutions postulate that algorithms should be executed locally. In turn, the high workloads characterizing state-of-the-art deep learning methods calls for innovative solutions at the hardware and architectural level to boost energy efficiency on constrained edge devices [54]. Research efforts consider a variety of perspectives, as summarized in Section 3.2.2. From a hardware point of view, novel architectural approaches have been proposed to efficiently execute core edge AI operations [55].

Among such strategies, In-Memory Computing (IMC) has recently attracted considerable interest in the research community. IMC effectively exploits data locality and regular computational patterns exposed by Convolutional Neural Networks (CNNs) to compute arithmetic operations in situ, bypassing processor pipelines and drastically reducing memory transfers without altering memory hierarchies [2]. From a vast pool of options, Bit-line Computing (BC) stands out for its simplicity of integration, where very few modifications are required to support energy-efficient CNN applications at ultra-low-power levels. BC operations are based on the concurrent activation of multiple memory words. The output logic values depend on a combination of the voltages stored on the accessed cells, hence performing in-memory bit-wise operations [28]. A few additional logic gates in the memory periphery are then employed to derive arithmetic operations from bit-wise ones.

BC has been explored independently with SRAM [28, 31, 40] and RRAM [22, 56]. As opposed to these efforts, in the first contribution of this chapter (Section 3.3), I present a hybrid BC architecture in which SRAM and RRAM are integrated together and are physically aligned in a single subarray, matching the performance. This design choice is motivated by the different run-time access patterns that characterize the data representing weights and activations of CNNs. On the one hand, since the same kernels are re-used many times, memory locations storing weights are seldom overwritten. Thus, in this case, the small write endurance and high write energy of RRAM [57] are less concerning because weights are stored in RRAM cells to exploit their zero leakage power, reaping significant energy benefits. On the other hand, SRAM stores activations, as they frequently change values, thus, fitting the low write energy of SRAM and its volatility.

However, based on the observation that the number of weights of CNN models can number in the millions, vastly exceeding the RRAM size that can be implemented in practice inside memory arrays, I considered a better data management strategy. For instance, the simple solution would be to overwrite the required weights in the RRAM continuously, but such an approach would rapidly deplete the RRAM write endurance. Consequently, I introduced Weight Data Mapping (WDM), in which all the possible representations of quantized weights are stored and mapped. Hence, in

the presented implementations, weights are directly accessed by their value, reducing data transfers while increasing energy efficiency and performance. Compared to a homogeneous SRAM-based equivalent architecture, the hybrid SRAM-RRAM design results in performance gains of 6.21× and 6.89× when executing inference on Alexnet [7] and Mobilenet [58], respectively, with an increase in the energy efficiency of 93% in both cases. In summary, the highlights of this contribution:

- I present a new hybrid BC architecture that integrates RRAM and SRAM memory cells. Moreover, the resulting circuit-level design was validated, showcasing the compatibility between the two technologies.
- I introduce WDM as a novel strategy to map, access, and perform BC operations in-situ, drastically reducing data transfers.
- I evaluate the proposed BC architectures and show that a 128-subarray design can process up to 22 images/second while requiring only 4.73 mJ/image for MobileNet. For Alexnet, a 128-subarray design requires 9.7 mJ/image and processes up to 11 images/second.

When exploring the hybrid architecture, due to the mandatory bit-line alignment of the RRAM- and SRAM-based LG, I observed that the CMOS transistor connected to the RRAM could be enlarged up to 50% the minimal transistor size. Thus, I explore how this bigger transistor, combined with a read assist technique, improves the RRAM bit-cell stability at lower voltages. Moreover, I show that the proposed solutions are still very appealing even when read margins are within the constraints. Actually, they can provide more relaxed write operations at the RRAM, saving writing energy and increasing memory endurance. The main outcomes of this contribution can be summarized as follow:

- I explore how increasing the width of the bit-cells access transistors mitigates their intrinsic technology variability, and I propose overdriving the memory word-line to reduce the equivalent parasitic resistance of the access transistors.
- I show a reduction in the write energy, at constant Bit-Error-Rates (BER), of up to 3× by employing larger transistor widths and voltage overdrive. Moreover, read access speed-ups of up to 90% are observed.

In the third contribution to this chapter, I present a full SRAM BC architecture that is robust towards ultra-low voltage supply levels, effectively coping with the ensuing

high error rates. The error detection and mitigation strategy operates simultaneously with in-memory computing and borrows most of its components from the BC circuitry, thus requiring minimal hardware overhead. Complementary robustness towards memory upsets can also be achieved with algorithmic-level considerations. In particular, using multiple CNN models (termed ensembles of CNNs in literature) effectively increases resiliency [59]. While such a strategy usually impacts the required workload, the authors of [36] recently proved that, by combining ensembling and pruning, new ensembling solutions can be derived at no additional cost. Therefore, the hardware/software co-optimization strategy ultimately enables a very aggressive scaling of the supply voltage with minimal impact on classification accuracy when executing CNN inferences but with a tangible benefit in terms of energy efficiency. In summary, the main outcomes of this contribution are as follows:

- I present a novel in-memory BC-based architecture that supports transparent in-situ error detection and mitigation while requiring minimal additional circuitry with respect to state-of-the-art BC solutions.
- Targeting five different CNN models, I show that the proposed combination of resilient optimization at the hardware and algorithmic levels can cope with a high rate of memory upsets of 0.01% with accuracy degradations below 1%. In turn, this characteristic opens the opportunity to scale the voltage supply aggressively and ultimately leads to energy efficiency gains of up to 51.3%.

The remainder of this chapter is organized as follows: Section 3.2 provides the necessary background on CNNs, in-memory computing architectures, and emerging non-volatile memories. Section 3.3 presents the hybrid SRAM-RRAM IMC architecture, published at Design Automation and Test in Europe (DATE) conference [50]. Section 3.4 discusses the strategy of voltage scaling on RRAMs, unpublished work. Section 3.5 presents the third contribution, published at Great Lakes Symposium on VLSI (GLSVLSI) [60]. Finally, Section 3.6 concludes the chapter.

3.2 Background and related works

3.2.1 Convolutional neural networks

CNNs have several layers, each abstracting higher-level features from a data source to interpret it [61]. Convolutional layers slide three-dimensional filters on the feature maps at their input to compute sets of three-dimensional outputs. A non-linear activation function typically follows the convolution operation. Pooling layers are

used to realize sub-sampling operations. Finally, fully connected layers combine all the data at their input and are usually employed in the last stages of a CNN.

CNNs are increasingly successful in various AI tasks, from image classification to natural language processing [6]. At the same time, they are becoming deeper (i.e., presenting an increasing number of layers) and more complex, with recent implementations such as ResNet-50 presenting over 23 million trainable parameters [62]. While more compact alternatives, such as MobileNet [58], have been proposed, their execution, even when only considering the inference phase, is still very computing-intensive, making their deployment on edge devices an open research topic [63].

A well-known approach to address this challenge is that of quantization. Indeed, representing weights and features in the fixed-point domain with a reduced number of bits only marginally impacts accuracy in most cases [64, 39] while dramatically decreasing storage and computational requirements. Interestingly, quantized CNNs are very well suited for executing in an IMC architecture for two reasons. First, SIMD parallelization and weight sharing can be easily implemented by adopting multiple memory subarrays. Second, the regular computations characterizing CNN layers only require a small amount of logic, which can be implemented at the memory periphery without incurring huge overheads.

Moreover, pruning also stands out as an excellent method to increase efficiency. It proposes to remove the least critical computation parts, considering granularity ranging from single weights [65] up to entire filters [66], decreasing both workloads at run-time and memory footprints.

Unfortunately, aggressive model compression achieved via quantization and pruning often incurs significant accuracy degradations. Thus, high model robustness is crucial when quantizing, pruning, and considering very low voltage supplies. To this end, combining several CNNs performing inference on the same input data (i.e., ensembling) is especially effective [67]. Nevertheless, ensembling generally incurs large memory and computational overheads because several models are combined to build ensembles. Addressing this issue, the authors of [36] propose a synergic use of quantization, pruning, and ensemble methods resulting in models having very high error tolerance without requiring additional memory and computing resources. In their work, an initial single-instance quantized CNN is first compressed by a factor N via filter pruning. Then, the resulting structure is replicated N times. Each pruned CNN model is individually trained on the target dataset starting from different (random) initial weight values so that N different trained models are obtained and ensembled.

3.2.2 In-memory computing architectures

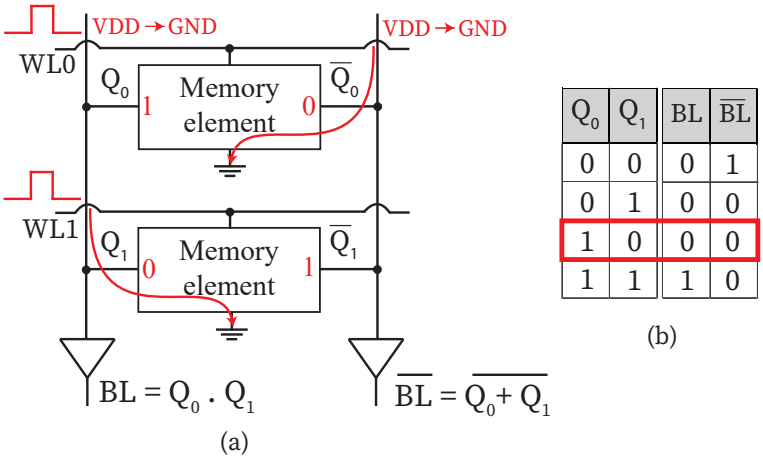
CNNs are both memory and data-intensive. They require a tightly coupled integration of storage and computing elements in order to be efficiently executed. Such features characterize domain-specific architectures such as Neural Processing Units [68] and systolic arrays [69], as surveyed in [55]. Taking this approach even further, IMC proposes strategies to merge computation and storage capabilities entirely.

IMC architectures are extremely promising for supporting edgeAI applications. They enable highly efficient Single Instruction Multiple Data (SIMD) parallelisms directly inside the memory hierarchy while requiring a low area compared to computation-specific accelerators [2]. Two notable IMC avenues are crossbar interconnects and BC. The former store parameter values as programmable resistances in a matrix of Resistive RAM elements [70]. While such an approach has the potential of greatly speeding up the matrix-vector multiplication at the core of CNN computation, it also relies on challenging non-CMOS technologies. It must cope with non-obvious implementation problems, e.g., related to noise rejection [71]. Conversely, BC architectures only require minor modifications to memory structures, making them ideal candidates for integration in existing memory hierarchies, minimizing data transfer while enabling a high degree of parallelism.

As the name suggests, BC employs the discharging mechanism along bit-lines in memory arrays as a basis for deriving arithmetic computations. Therefore, it can be applied in memory arrays whenever two bit-lines are employed to access the bit value stored in a memory cell and its complement value, such as SRAMs. However, such a mechanism does not depend on the implementation of the memory cell itself.

In standard (i.e., non-BC) accesses, bit-lines are pre-charged to Vdd. Then, the word-line connection corresponding to a specified address is assessed, connecting the bit-lines to the memory cell. If the cell stores the '0' value, the bit-line discharges to the ground, while the complement bit-line stays at Vdd. The opposite happens if the cell stores a logical '1'. Word-lines horizontally connect multiple bits, allowing access to an entire memory word at once. The key insight of bit-line computing is that if two word-line signals are activated concurrently, two discharge paths are presented for each bit position, as depicted in Figure 3.1-a. Hence, the bit-lines carry the binary AND and NOR signals of the corresponding bits of two different words, respectively, as shown in Figure 3.1-b.

Among the existing works, [72, 27] only support bit-wise operations. In contrast, [28, 29] propose support for addition and shift, which can be used to perform multiplication when chained. BLADE [31] exhibits the best trade-off between density



Q_0	Q_1	BL	$\overline{BL}$
0	0	0	1
0	1	0	0
1	0	0	0
1	1	1	0

Figure 3.1. BC operation. Bit-lines behave as the logic gates AND and NOR when concurrently accessing two memory cells.

and performance. Thanks to its Local Group (LG) organization, it does not require a word-line underdrive to mitigate data corruption risks [72, 27]. Moreover, using LGs reduces array density by only 10 to 15%, making it more suitable than 8-10T bit-cell arrays [28, 73], which require up to 100% area increase. BLADE also presents the best behavior at lower voltages among the presented architectures. However, the authors didn't consider the effect of lower voltage on memory reliability and how this affects the quality of service (e.g., accuracy) of CNNs. As opposed to that, in the third contribution to this chapter, I present a strategy to cope with errors in memory without degrading CNN accuracy.

The works mentioned above consider SRAM memory cells. Instead, implementations based on emerging Resistive Random Access Memories (RRAM) are proposed in [74] adopting magnetic memory. The authors of [22] and [56] present architectures based on Resistive RAM to implement parallel multiplications and binarized neural networks, respectively. However, all these previous designs employ homogeneous memory structures based on SRAMs or RRAMs. Instead, in the first contribution to this chapter, I show that a hybrid BC architecture, tailored to data access patterns, can leverage the benefits of both technologies (e.g., the high endurance of SRAM and the absence of leakage of RRAMs) while minimizing their downsides and that they can be effectively co-integrated.

3.2.3 Emerging non-volatile memories

RRAM exploits material resistivity control to implement persistent storage, i.e., memories that retain their state without any applied voltage. Different families of RRAM

technologies have been introduced in recent years, such as Magnetic Random Access Memory (MRAMs), Phase Change Memory (PCMs), and filamentary-based RRAMs (ReRAMs). The absence of leakage power would make these technologies ideal candidates for replacing traditional SRAMs and DRAMs at different levels of the memory hierarchy [57, 75].

The conventional way to co-integrate RRAM and CMOS technologies is by creating arrays of 1-Transistor 1-Resistance (1T1R) bit-cells, which enable 3× to 4× higher integration density than 6T SRAM memories [76, 77]. RRAM memories can be programmed between several states called Low Resistance State (LRS) and High Resistance State (HRS) through programming phases that are energy hungry (> pJ/bit). On the other hand, read operations consist of sensing the resistance value and are in the same order of magnitude as SRAM reads. Due to all these reasons, added to their relatively low integration cost compared to conventional embedded non-volatile memories [57], RRAM technologies are very appealing when realizing resource-constrained edgeAI systems.

3.3 Hybrid SRAM-RRAM bit-line computing architecture

The contributions utilizing RRAM technology predominantly concentrate on read and bit-line computing operations. In this context, the presented analysis remains unbiased towards any specific resistive memory, as the general characteristics of resistive memories, involving their behavior as ideal resistance during read operations, are considered. It is essential to note, however, that write operations within RRAM technology are complex due to their intricate nature. Often, these write operations necessitate the utilization of high-current drivers and voltage levels surpassing the Vdd (supply voltage). Consequently, the practical implementation of the proposed architectures and methodologies could potentially incur an increased area overhead. Furthermore, it's worth acknowledging that the complexity of write operations in RRAM technology may lead to a requirement for circuit solutions not explicitly discussed in the current section. As such, addressing these circuit-level challenges would be imperative to achieve the seamless integration of RRAM-based solutions into existing computational architectures.

3.3.1 Hybrid architecture design

The proposed hybrid memory architecture comprises various heterogeneous subarrays, organized similarly to BLADE [31], in Local Groups (LG). However, in this case, two LG types are considered: SRAM- and RRAM-based. Irrespective of their

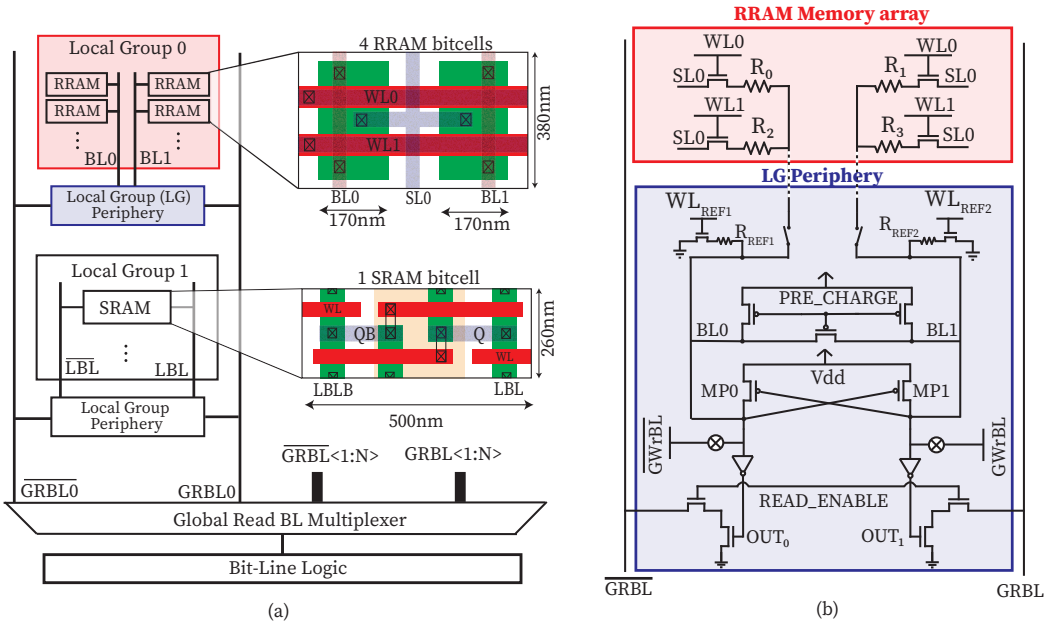


Figure 3.2. (a) Block diagram of the proposed Hybrid SRAM-RRAM BC architecture. It presents the proposed memory organization, the bit-cells layout, and the pitch matching between SRAM and RRAM-based Local Groups (LGs). (b) The architecture of an RRAM-based LG.

implementations, LGs must be physically aligned to share the same set of Global Read Bit-Lines (GRBLs), as shown on Figure 3.2-a. Hence, considering a 28nm CMOS technology node and high-density SRAM rules ($0.127\mu m^2$) as in Section 2.2.2, RRAM LGs must be pitched in a 500nm width, i.e., the width of SRAM cells. Furthermore, RRAM bit-cells must present the same poly-silicon gate orientation as SRAM ones.

3.3.1.1 Pitch matching bit-cells

The minimum width for an RRAM bit-cell in the considered technology is 180nm (considering the access transistor and the additional space for vias). Thus, at maximum, two RRAM bit-cells fit in the same pitch for one SRAM bit-cell. As shown in Figure 3.2-a, the design abides by these constraints by pitch-matching 4 RRAM bit-cells in a 2×2 configuration.

The additional available space is leveraged by increasing the access transistor width beyond the minimum size of 100nm to 170nm, increasing the read margin and easing programming operations. A 380nm height for two bit-cells is achieved by merging the access transistor sources. This design leads to an RRAM bit-cell surface of 0.0475 μm^2, 2.74× denser than the SRAM ones.

3.3.1.2 RRAM local group design

Figure 3.2-b presents a circuit-level view of the proposed RRAM-based LG, including its periphery circuit. To navigate the circuit, the operations of write, read, and BC are considered in the following.

Write operation: RRAM devices can switch between two resistance states: HRS and LRS. The switch from HRS to LRS is called the 'set' operation, while the switch from LRS to HRS is known as the 'reset' operation. Figure 3.2-b shows two transmission gates connecting the Global Write Bit-Lines (GWrBL) to the local bit-lines. Then, with the programming voltage set on the required bit-line, the set operation in a single RRAM bit-cell is performed by activating one word-line and biasing the source-line. The opposite is done for the reset operation: grounding the selected bit-line and biasing the source-line to the programming voltage.

Note that the source-line (SL0) is shared between all four bit-cells, while word-lines WL0 and WL1 are shared between two bit-cells. During the set or reset operation, the switch on the opposite side of the targeted bit-cell is deactivated, preventing any current from passing through the resistive device. This mechanism effectively safeguards against unintended bit-cell writes, ensuring the integrity of the desired write operation.

Read operation: Reading is performed by pre-charging[1] the bit-lines. Then, the targeted RRAM bit-cell is compared with a R_{REF} reference resistance based on their discharge drive of the bit-lines. However, due to the fact that two bit-cells share the same word-line, a comparison is needed between R0 and R2 with R_{REF2}, while R1 and R3 are compared to R_{REF1}, as illustrated in Figure 3.2-b.

To do so, only the selected bit-line is connected to one side of the sense amplifier, while the other one is connected to the opposite R_{REF} through a switch, as in Figure 3.2-b. Finally, to discriminate between the two sides of an RRAM array during the read operation, the data stored in one of the two bit-lines are inversely encoded. Such an approach enables compatibility between the single-ended RRAM LGs and the differential SRAM LGs.

The read operations are completed by the sense amplifier that assesses logic values for the operation. The sense amplifier is made up of two PMOS transistors and two inverters. The transistors MP0 and MP1 are deactivated after pre-charge since their gates are cross-connected with the bit-lines. Then, when the data word-line (WL0 or

[1] 3 PMOS connected to the bit-lines are responsible for the pre-charge phase before every read of BC operation.

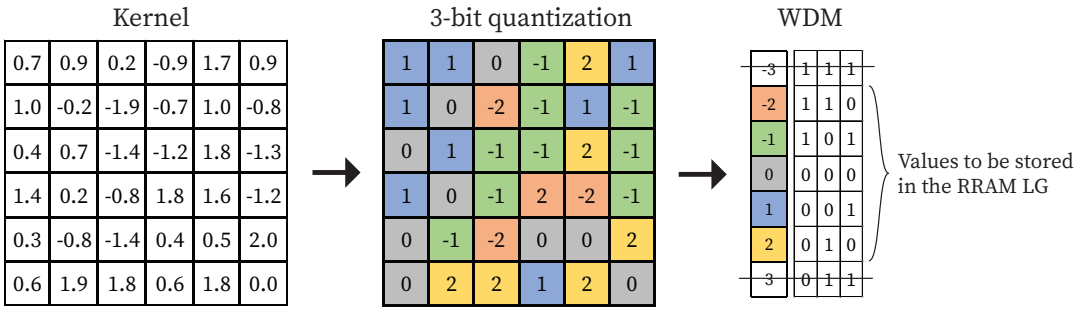


Figure 3.3. Encoding a 6×6 Kernel using 3-bit quantization and Weight Data Mapping.

WL1 in Figure 3.2-b) and the correspondent reference word-line (WL_{REF1} or WL_{REF2}) are activated, the nodes BL0 and BL1 subsequently discharge. The side where the bit-line discharges faster opens the other side's PMOS, connecting it to Vdd and forcing the bit-line to stay activated, latching the sense amplifier.

BC operation: Given that RRAM LG and SRAM LG exhibit compatible read schemes and comparable performance, they can effectively engage in BC operations between them. Consequently, supplementary logic is utilized beneath the array to derive arithmetic operations, including additions and shifts, from bitwise operations. Furthermore, repeated add-and-shift operations are employed to carry out multiplications. For multiplication, the number of required clock cycles is twice the word size: one cycle for the shift-and-add operation and another cycle to write the result back into memory.

3.3.2 Weight data mapping

Storing all weights of a CNN model in RRAM-based LGs would require a huge memory capacity, and it may be unfeasible in practice within the resource constraints of edge devices. Therefore, I present a novel Weight Data Mapping (WDM) strategy that greatly reduces memory requirements to program and access the learned CNN parameters. In quantized CNN models, quantization reduces the admissible weight values to a small set of size 2^q, where q is the post-quantization bit-width. Therefore, instead of storing these values and accessing them by their addresses, WDM proposes to store the admissible values in the RRAM-based LG and read them based on their values rather than their address, which is easily accomplished using a look-up table.

Figure 3.3 exemplifies the approach. The 6×6 kernel, where data is represented in floating-point, is first quantized to a three-bits signed representation. Hence, the elements are cast to integers in the range [-3, 3]. Then, only the resulting post-

quantization values are stored in RRAM. In the Figure 3.3, it can be noticed that the values {-3, 3} are unused. Thus, the WDM scheme must reprogram only the remaining five values. This behavior is also common when considering larger values of q. For $q = 8$ (the quantization level adopted in the rest of this contribution), AlexNet trained on the CIFAR-10 data set employs only 90 of 256 possible values. In comparison, in the case of $q = 16$, such ratio drops to 1097 values out of 65536. Additional energy savings are therefore achieved by only programming in RRAMs the weight values used for inference.

3.3.3 Electrical validation

The functionality and integration of the hybrid architecture described in Section 3.3 was validated by implementing and simulating it using the 28nm bulk CMOS PDK from TSMC. The RRAM LG from Figure 3.2 was simulated using ideal resistance models connected to access NMOS transistors. Such an approach provides technology agnosticity as RRAM technologies usually exhibit almost ideal ohmic behavior while biased under a given critical voltage. Then, a parasitic capacitance was considered for the bit-lines and the RRAM ($10aF$ per bit-cell). Moreover, the reference resistance (R_{ref}) required to perform pseudo-differential read operations was considered a polysilicon resistance connected to a regular access transistor. The reference resistance was set to $R_{ref} = 30K\Omega$ for the simulations as it enables a bit-line discharge time intermediate between $R_{LRS} = 10K\Omega$ and $R_{HRS} = 100K\Omega$, which are commonly reported mean values for RRAM LRS and HRS resistance states distributions [57]. To cover CMOS process variability, RRAM process, and cycle-to-cycle variability, the LRS (HRS) was swept from $30K\Omega$ to $10K\Omega$ (respectively $100K\Omega$), and for each value, 1000 Monte-Carlo simulations were performed. With this approach, the proposed circuit has no read failures down to $10K\Omega$ around the reference value at 1V, thus demonstrating that read operations from an RRAM LG can be reliably performed.

3.3.3.1 Hybrid subarray simulation

To optimize simulation time, I focused on simulating only the critical paths of the subarray. The propagation times of signals were modeled through equivalent circuits containing the extracted RC network and the corresponding gates. To simulate a realistic worst-case condition, I considered the last bit-line and word-line to account for the longest propagation time along the metal lines. The memory array of 256BL × 64WL was implemented and simulated considering one RRAM- and one SRAM-based LG of 32 word-lines each.

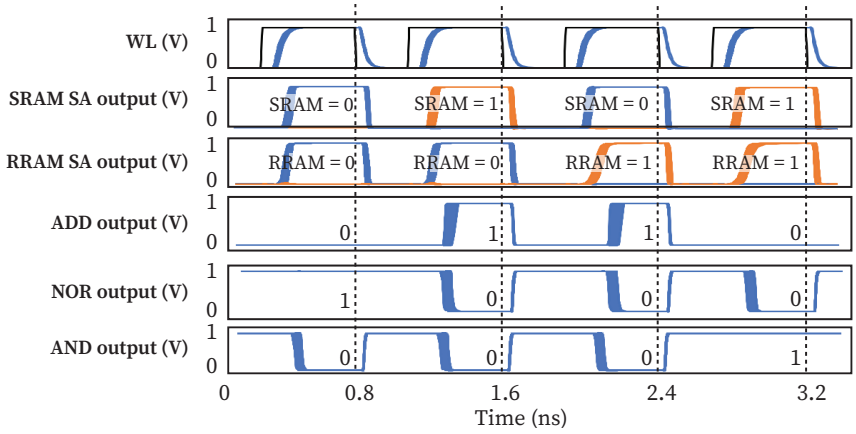


Figure 3.4. Transient simulation of 256×64 memory array featuring two LG, one RRAM-based with 32 WLs and one SRAM-based with 32 WLs. The simulation shows the SRAM and RRAM sense amplifier output envelope.

Figure 3.4 shows one thousand Monte-Carlo transient simulations of the proposed hybrid architecture running BC operations between SRAM and RRAM LGs at 1V. In this simulation, the data accessed by the RRAM- and SRAM-based LGs change in each cycle. The first waveform shows the word-lines signals, the input signal depicted in black, and the propagated signal that arrives in the last bit-cell in blue. SRAM and RRAM sense amplifier outputs are shown in the second and third plots from the top, respectively. The orange and blue lines show the signals transmitted to GRBL and $\overline{GRBL}$. The last three plots depict the output of the operations ADD, NOR, and AND performed on the two read bits, showing that the RRAM and SRAM LGs provide compatible performance ranges and can, therefore, be co-integrated inside the proposed hybrid subarray to perform BC operations reliably. The simulations report that the energy required for a BC operation among a value stored in RRAM and one in SRAM is 16.5 fj/bit. Other system-level energy and performance values are derived from [31], which describes a detailed design space exploration of the BLADE architecture.

3.3.3.2 Ultra-low voltage operation

Figure 3.5 shows a comparison of the RRAM (red) and SRAM (black) LGs read operations while taking into account CMOS variability and considering a $30K\Omega$ reference value for the RRAM LG. As previously described, RRAM LG performances correlate entirely to the R_{ref} value. Considering a smaller R_{ref} could shift the blue curve down at a price of higher energy consumption during an LRS read. Also, it should be noted that R_{ref} is highly related to the parameters of the RRAM technology.

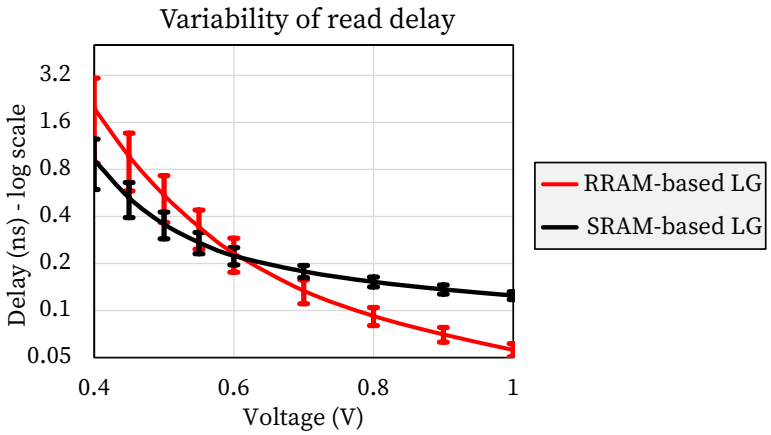


Figure 3.5. Delay of a read operation performed in SRAM (red) and RRAM-based (black) LGs. RRAM-based LG shows higher performances and lower variability than SRAM-based LG below 0.6V.

Overall, two effects are visible in Figure 3.5: (i) the curves are crossing, i.e., below 0.6V, RRAM-based LGs become more profitable than SRAM-based LGs. (ii) Variability-wise, SRAM-based LGs tend to be more volatile than RRAM-based LGs at low voltage. At 1V, the SRAM-based LG is 55% faster than the RRAM-based LG and shows 25% less standard deviation. The LG performances match at Vdd = 0.6V, and the RRAM-based LG shows +53% faster read operation speed with 0.4V and 70% less deviation. These two effects can be explained as follows: (i) poly-silicon reference resistance (which controls the SA switching when reading a HRS) can be accurately controlled and fixed, mitigating the RRAM HRS variability. (ii) RRAM LRS state shows a linear behavior, while CMOS transistors tend to show an extremely non-linear resistance. This effect has been explored and exploited in [78] to introduce RRAM-based transmission gates in sub-Vt FPGAs.

The trends on low voltage presented in this section do not consider the matter of degraded reliability and how this affects read margins and overall quality of service. To this end, in the second and third contributions to this section, I investigate and propose solutions to these drawbacks.

3.3.3.3 Macro memory H-tree

The energy cost associated with the communication between each subarray and the controller residing at the BC array boundary was modeled to perform a system-level exploration of a multi-array architecture. The connection was done through a scalable H-tree interconnect, capable of operations pipelining. The length of each wire was measured, and the parasitic capacitance associated was extracted considering 100nm

Table 3.1. Explored homogeneous and heterogeneous BC designs.

Architecture	SRAM (bytes)	RRAM (bytes)	Area (μm^2)	Bit-density (bit/μm^2)	Leak/op (fj)
BLADE	640	0	1318	3.88	27.8
SRAM_WDM	816	0	1973.6	3.31	35.4
HY_WDM_1	640	176	1556.4	4.19	27.8
HY_WDM_2	512	176	1318	4.18	22.2

pitch Metal 4 buses. The equation $E = CV^2/2$ determines the unitary energy cost of the metal line charge. Each subarray is connected to two address buses, one bidirectional data bus, five control signals, and the subarray decoder bus. BC operations require the activation of the two address buses for the two operands. In contrast, normal read and write operations require the activation of one address bus and one data bus. All these aspects were considered while modeling energy costs related to data transfers and subarray controlling.

3.3.4 Application-level simulation

To assess the performance of applications running on the proposed BC architecture, a cycle-accurate simulator able to model the execution of entire CNN inferences (i.e., pooling, convolutional, and fully connected layers) was developed. To this end, a simple run-time behavior in which the inputs to each CNN layer are streamed to the memory is considered. Only the CNNs activation is transferred to the memory in a configuration that uses WDM. In contrast, without WDM, both activation and weights are transferred [2].

Based on the geometry of the kernel and the subarray capacity, the simulator tiles the input data and distributes the tiles to different subarrays. Moreover, if the number of tiles exceeds the number of subarrays (a common occurrence for large feature maps), multiple rounds are performed for a single layer. Similarly, large convolutions exceeding the available memory capacity are decomposed in partial ones. The data bus is considered to have a data transfer bandwidth of one word per cycle. BC operations require two cycles instead, one to perform the bit-wise operation and the other to write the result. Still, the same operation can be performed in parallel on each subarray on different data.

[2] This work does not consider the mapping strategy presented in Chapter 4. In fact, the observation that weights are not required at the subarray level motivated the innovation discussed in that chapter.

3.3.5 Hybrid architecture evaluation

In the following, I discuss the benefits of the hybrid SRAM-RRAM BC architecture from performance and energy efficiency standpoints when executing two different CNN benchmarks (Mobilenet [58] and Alexnet [7]). Four different memory subarray implementations are evaluated, which incrementally embody the novel features introduced in Section 3.3. Multiple subarrays are integrated by considering the H-tree interconnect described in Section 3.3.3. The subarray characteristics are summarized below and in Table 3.1:

- Baseline: an SRAM-based BLADE subarray [31]. It embeds 5 LGs containing 32 WLs (i.e., 64 words).
- The SRAM_WDM design adds further LGs, still implemented in SRAMs, dedicated to storing weights data using WDM.
- HY_WDM_1 has the same organization than SRAM_WDM, but employs RRAM instead of SRAM for storing weights.
- HY_WDM_2 is a further hybrid implementation which, by only embedding four SRAM LGs, has the same area of the BLADE baseline.

As reported in Table 3.1, SRAM_WDM presents the highest leakage since it features the highest amount of SRAM memory cells. Conversely, HY_WDM_2 has the smallest SRAM capacity among the investigated design points and the smaller leakage energy per operation.

3.3.6 Computation-to-communication analysis

While BC architectures can significantly reduce the required memory operations, data transfers still account for an essential part of run-time when considering designs supporting a high degree of parallelism. Highlighting this effect, Figure 3.6 shows in blue the clock cycles devoted to memory transfers in the BLADE design and the cycles required for in-memory computation in green. A cross-over point is reached for a 16-subarray memory, after which data transfers dominate the overall workload. By instead adopting the WDM approach, BC operations constitute the majority of run-time even for significant memories of 128 subarrays, thanks to a considerable reduction in the amount of data transfers (60× and 27× for the AlexNet and Mobilenet, respectively).

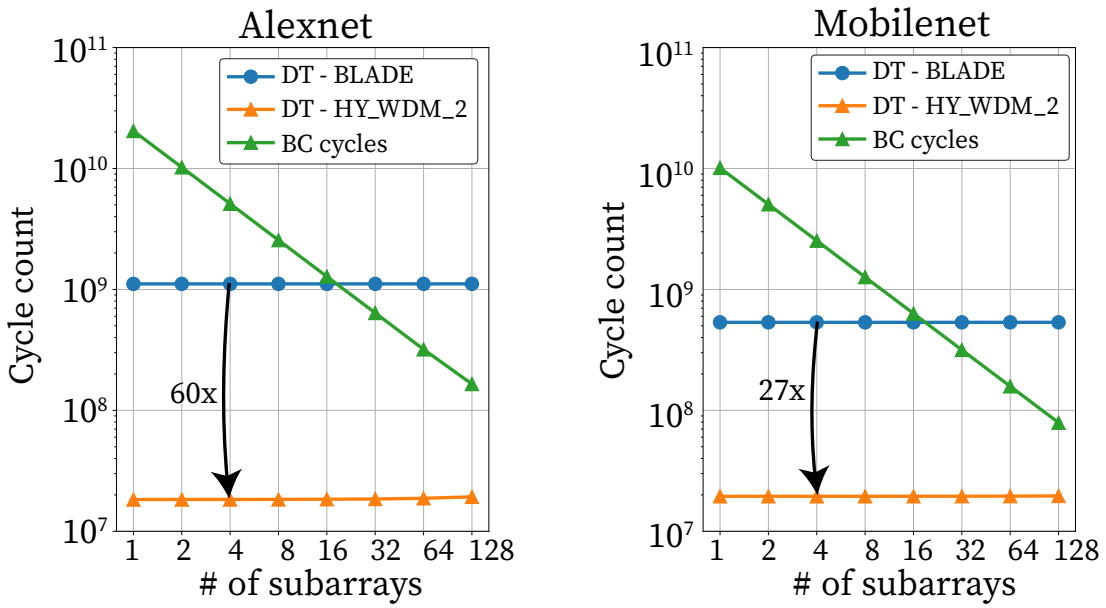


Figure 3.6. Number of Data Transfer (DT) and Bit-line Computing (BC) clock cycles required for inference in (a) Alexnet and (b) Mobilenet.

3.3.7 Energy evaluation

Speedups achieved thanks to WDM positively impact energy efficiency, as showcased in Figure 3.7. This figure reports the energy gains achieved by WDM designs with respect to the equivalent BLADE ones [3] [31] (Section 2.2). Moreover, energy benefits become more relevant with increasing memory sizes because run-time (hence, leakage energy) becomes increasingly dominated by data transfers.

In addition, Figure 3.7 shows that hybrid architectures (HY_WDM_1 and HY_WDM_2) are even more energy-efficient because weights are stored in zero-leakage RRAMs. The importance of minimizing leakage currents is highlighted by the high efficiency of the HY_WDM_2, which presents the least number of SRAM cells, achieving up to 93% energy gains (for both Alexnet and in the Mobilenet) concerning a baseline BLADE implementation.

Finally, the findings are summarized in Table 3.2, which compares the energy-per-inference and the frames per second (FPS) of homogeneous (BLADE) and heterogeneous (HY_WDM_2) designs. Energy and FPS are pretty close in the two cases for simple memory organizations (e.g., when only four subarrays are employed) but highly favor hybrid architectures for more complex cases. Indeed, a 128-subarray

[3] Gains are defined as $(E_{BLADE}/E_{\#} - 1)$, where E_{BLADE} is the energy-per-inference of BLADE and $E_{\#}$ the one of the case under study.

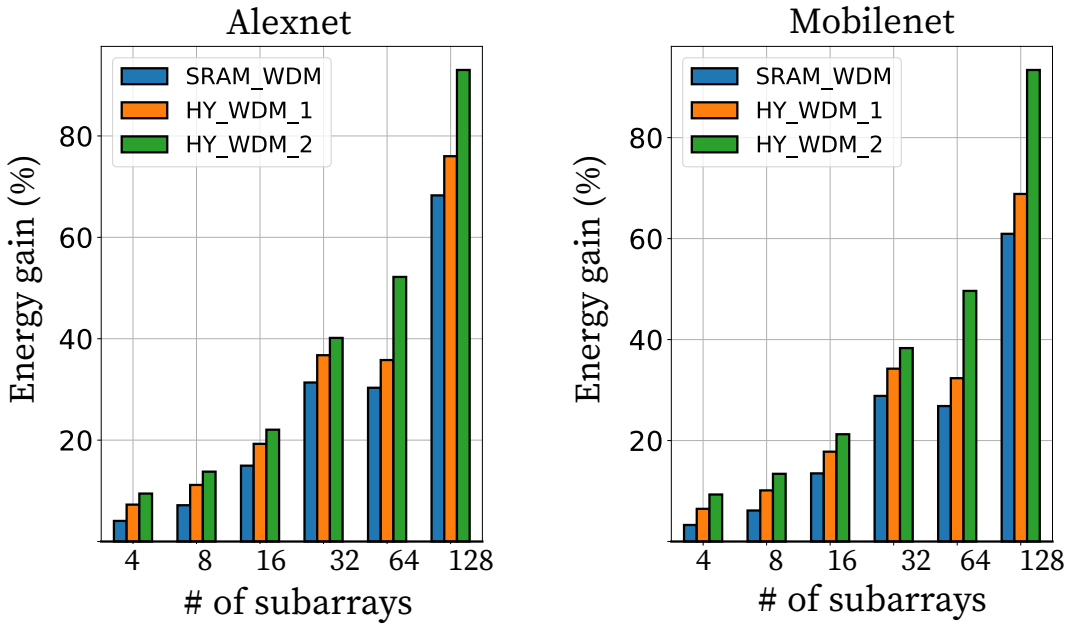


Figure 3.7. Energy gain over baseline BLADE implementations, varying the number of subarrays. (a) Alexnet, (b) Mobilenet.

HY_WDM_2 architecture can execute Mobilenet at more than 22 FPS while only requiring 4.73 mJ per frame. HY_WDM_1, which presents a larger SRAM capacity, achieves a slightly higher performance (23 FPS) at the cost of a decreased area and energy efficiency.

3.4 Exploring density/reliability trade-off in RRAM

Contrasting with the ongoing trend of increasing as much as possible the RRAM integration density, in this contribution, I explore the trade-off deriving from adopting a larger memory footprint for RRAM memory cells in exchange for more dependable low-voltage operations. The investigation is focused on a 1 Transistor - 1 RRAM (1T1R) topology that avoids IR drop issues and demands a less complex peripheral circuitry. Moreover, for the sake of simplicity, I study such trade-offs on entire RRAM arrays rather than hybrid SRAM-RRAM ones.

The reliability of 1T1R RRAMs is tightly linked to the variability of the electrical characteristics employed by access transistors, especially in low-voltage supply regimes. To mitigate this effect, I explore how increasing such transistors' width impacts the read operation. Moreover, I show that by combining a larger transistor with Word Line Overdrive (WLOD), read operations can be reliably performed in ultra-low voltages.

Table 3.2. Runtime performance (frame/second) and efficiency (energy/frame) of homogeneous and heterogeneous designs.

		Subarrays	4	32	128
Energy (mJ)	Alexnet	*BLADE*	10.22	14.00	18.73
		HY_WDM_2	9.34	9.99	9.70
	Mobilenet	*BLADE*	5.06	6.88	9.15
		HY_WDM_2	4.63	4.98	4.73
FPS	Alexnet	*BLADE*	0.35	1.26	1.73
		HY_WDM_2	0.43	3.34	11.93
	Mobilenet	*BLADE*	0.72	2.58	3.58
		HY_WDM_2	0.86	6.55	22.29

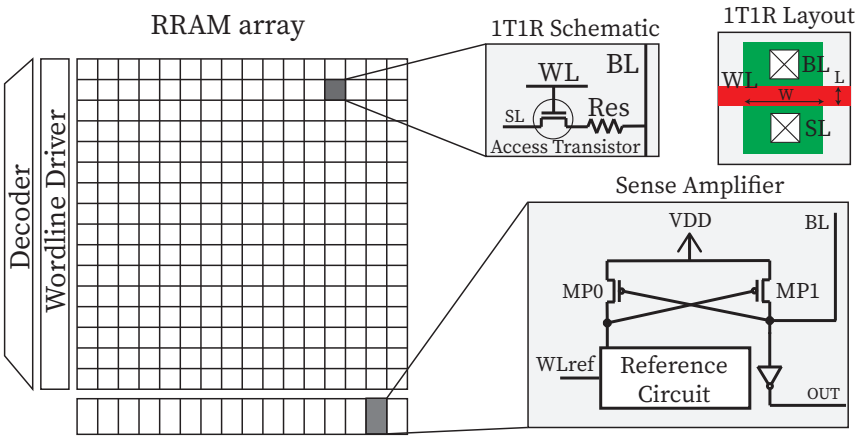


Figure 3.8. Circuit used for validation presenting the bit-cell configuration and the voltage-mode single-ended sensing amplifier.

Finally, I showcase that increased read margins can be exploited to relax constraints on write operations, ultimately increasing energy efficiency and positively impacting the endurance of RRAM memories.

The 1T1R bit-cell is depicted in Figure 3.8, where the variable resistance is placed in series with the access transistor. To assess the value stored in an RRAM bit-cell, I consider a voltage-based sensing as it is faster, more energy, and area efficient than the current-based SAs [79]. The RRAM-based LG sense amplifier presented in Section 3.3.1.2 is similar to the one considered in this work. However, diferently from the hybrid architecture, only one bit-line is sensed, instead of two. Nevertheless, since the exploration focuses on increasing read margins in RRAM arrays by applying solutions at the bit-cell level, the findings are independent of the SA topology.

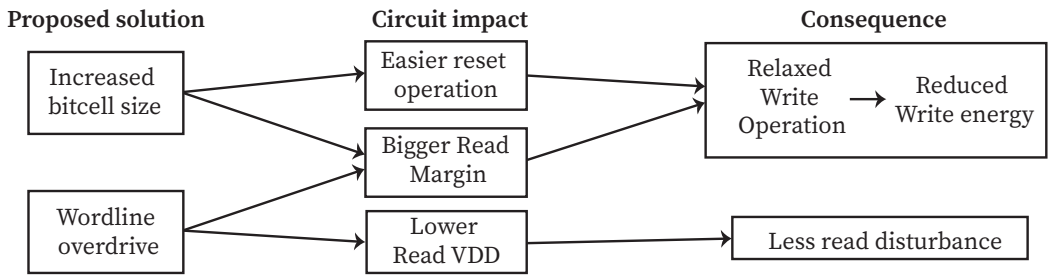


Figure 3.9. Exploration avenues presented in this paper, the circuit impact expected, and how to leverage them in two main axes: Decrease of the write energy (thus increasing the bit-cell lifetime) and a more reliable read operation.

A strong relationship exists between the HRS/LRS ratio and programming energy. Indeed, to ensure reliable low-voltage read accesses, programming must provide high ratios between the resistance states. These can only be achieved by increasing the write voltage, time, and/or current to cope with the intrinsic technology variability [20, 21]. In addition to requiring more energy-per-write, large HRS/LRS ratios also reduce the memory lifetime (endurance to write operation) by introducing additional stress during programming operations.

3.4.1 Tuning 1T1R access transistor and word-line voltage

Figure 3.9 summarizes how increasing the access transistor width and/or the word-line voltage impacts read and write accesses to RRAM bit-cells. By increasing the size of the access transistor, the impact of the process variability on the transistor is minimized, mitigating the influence of the transistor on read errors. The effect of technology variability, and hence the importance of mitigation strategies, are especially crucial at low voltage supply levels. Indeed, the authors of [20] consider 0.6V a minimum viable voltage supply with minimally-sized transistors because of variability-induced read margin reductions.

The WLOD strategy proposes activating the WLs using a voltage higher than Vdd to increase the reliability of bit-cells at sub-threshold voltages. In sub-threshold regimes, the equivalent transistor resistance (R_{ds}) increases, disturbing the read operation as this resistance is in series with the RRAM. By applying WLOD, the R_{ds} decreases, which increases the read margin and allows the read operations to be performed reliably at ultra-low voltages without requiring high HRS/LRS ratios during the write accesses.

One further benefit of WLOD is the reduction of read disturbances on the resistance state of the RRAM. When reading the bit-cell, the voltage between the bit-line and

source-line can flip the cell or drift its resistance. Thus, reading the bit-cell at a lower voltage mitigates the read disturbances. Also, the energy spent during the read operation is decreased as the precharge process energy is quadratically reduced ($E = 0.5CV^2$).

3.4.2 Exploration methodology

I considered a simulation environment that relies on a 28nm CMOS bulk technology industrial PDK. The 28nm node is used as it is the most advanced technology considered for industrial use of RRAM technologies [80, 20, 76]. The read and write performance of the target 256BL x 256WL (65kb) RRAM memory is extracted from three low-voltage settings ranging from 0.4V to 0.6V. Word-line overdrives were set between 50mV and 300mV.

The employed access transistor widths range from 100nm to 200nm, resulting in a bit-cell density reduction of 46%. However, in the context of the hybrid BC architecture, presented Section 3.3, due to the alignment of SRAM and RRAM local groups, the extension of the 1T1R access transistor does not imply area penalty. It instead uses the already available space.

CMOS technology variability is considered through ten thousand Monte Carlo simulations. For simulations during read operations, RRAM cells are considered ideal resistances and consider the values as corner values. When referring to an HRS value (respectively LRS), it is the minimum HRS value (respectively maximum LRS value) to have a successful read. In the same way, when referring to a HRS/LRS ratio, it is the minimum acceptable ratio beyond which the read is failing.

RRAM parasitic capacitance is added along the bit-line ($10aF$) in each bit-cell when simulating complete arrays besides standard RC parasitics. Such modeling strategy can easily be adapted to target read operations in different RRAM technologies. The programming operations of RRAM bit-cells considered the model from [81] (referring to filamentary HfO2-based RRAM), which was calibrated on recent measurement data from [57]. During these programming operations, it was considered the self-terminated set and reset pulses, such as in [82]. As discussed in [57, 21], the HRS value is hardly controllable. For this reason, it is considered a high enough resistance $100k\Omega$ HRS and then finely tune the LRS by controlling the set current to values from $20\mu A$ to $150\mu A$ to control the LRS value from $5k\Omega$ to $50k\Omega$ (i.e., 20 to 2 HRS/LRS ratio).

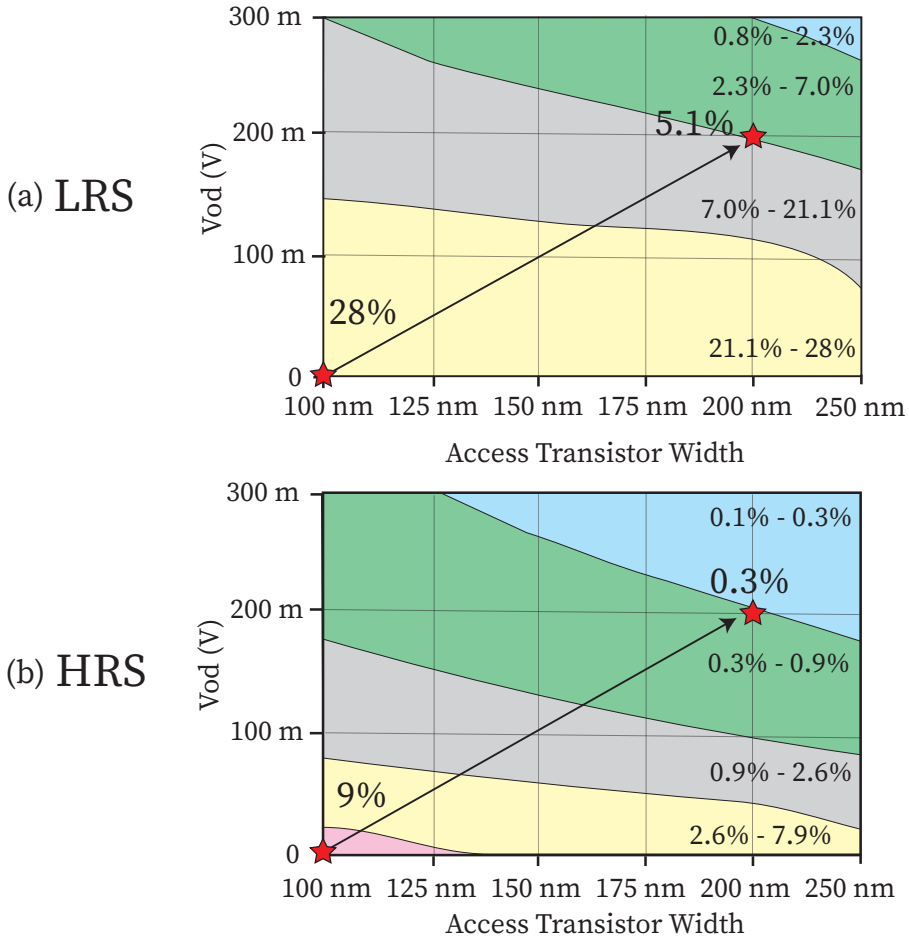


Figure 3.10. Coefficient of variation of the current I_{cell} during an LRS (a) and HRS (b) read operation in the function of the Vod and the size of the access transistor.

3.4.2.1 Analysis of read current, timing and reliability

The variability of the access transistor reduces the read margin as it hinders the sense amplifier precision, especially at lower voltages. The coefficient of variation measures the extension of this variation over the nominal value, and it is defined as the ratio of the average μ_I over the standard deviation σ_i.

Considering the values for HRS and LRS presented on the Section 3.4.2, Figure 3.10 shows the coefficient of variation of the read current I_{cell} versus the WL overdrive (Vod) and the transistor width at Vdd = 0.5V, in LRS and HRS, respectively. For the LRS, using V_{WL} = Vdd and the smallest transistor width allowed on the considered 28nm technology (100nm), the coefficient of variation is 28%, while applying a Vod = 0.2V and doubling the width of the access transistor, this variation drops to 5.1%. For HRS, the same trend is visible. The I_{cell} with no memory enhancements presents a

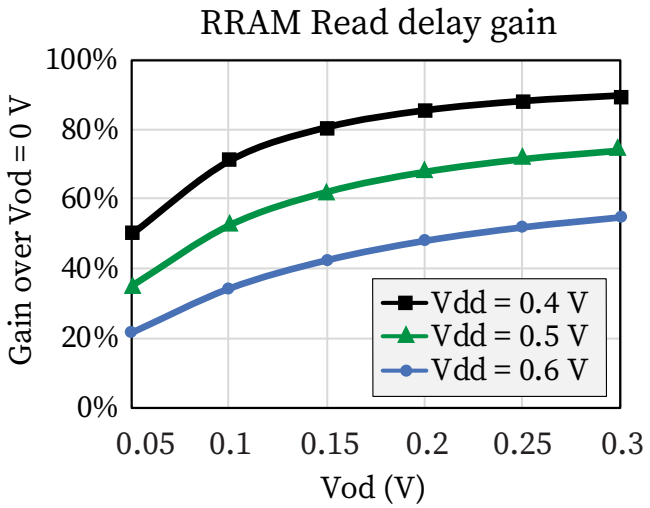


Figure 3.11. Read delay gain with Word-line Overdrive for different values of Vdd.

coefficient of variation of 9%, while with Vod = 0.2V and the W = 200nm, this value drops to 0.3%. For the same width presented in the hybrid architecture, 170nm, the LRS (HRS) presents a variability of 7.2% (0.6%). The effect of CMOS variations is less prominent in HRS as the transistor voltage drop V_{ds} is lower in HRS than in LRS, resulting in lower coefficients of variation.

Figure 3.11 presents the read delay improvement when the WLOD technique is applied. In blue, the performance gain for Vdd = 0.6V, orange for Vdd = 0.5 V, and gray for Vdd = 0.4 V. WLOD has a bigger impact as the Vdd decreases. For Vdd = 0.6V, the read is performed 48% faster when Vod = 0.2V and 55% when Vod = 0.3V, in comparison when no voltage overdrive is applied. While considering a Vdd = 0.4V, the gain with Vod = 0.2V is 85%, and for Vod = 0.3V the gain is 90%. Therefore, WLOD decreases the variability of the I_{cell} during the read operation and increases timing performance at ultra-low-voltage levels.

Figure 3.12 shows the evolution of the BER during read operations versus the HRS/LRS ratio for (a) Vdd = 0.4V and (b) 0.5V. Represented in blue is the failure rate for a read performed without any enhancement, in gray and orange when instead the assists techniques (WLOD and increased width of access transistors, respectively) are applied individually. Finally, the yellow curve shows the BER for the read operation using both techniques. Three main observations can be made from Figure 3.12. (1) At a constant HRS/LRS ratio, introducing the proposed read assist techniques can drastically reduce the bit error rate (e.g., by 30% for an HRS/LRS ratio of 2 at 0.4V) without any technology or programming operations modifications. (2) At constant BER, thanks to the proposed techniques, the HRS/LRS ratio can be dramatically

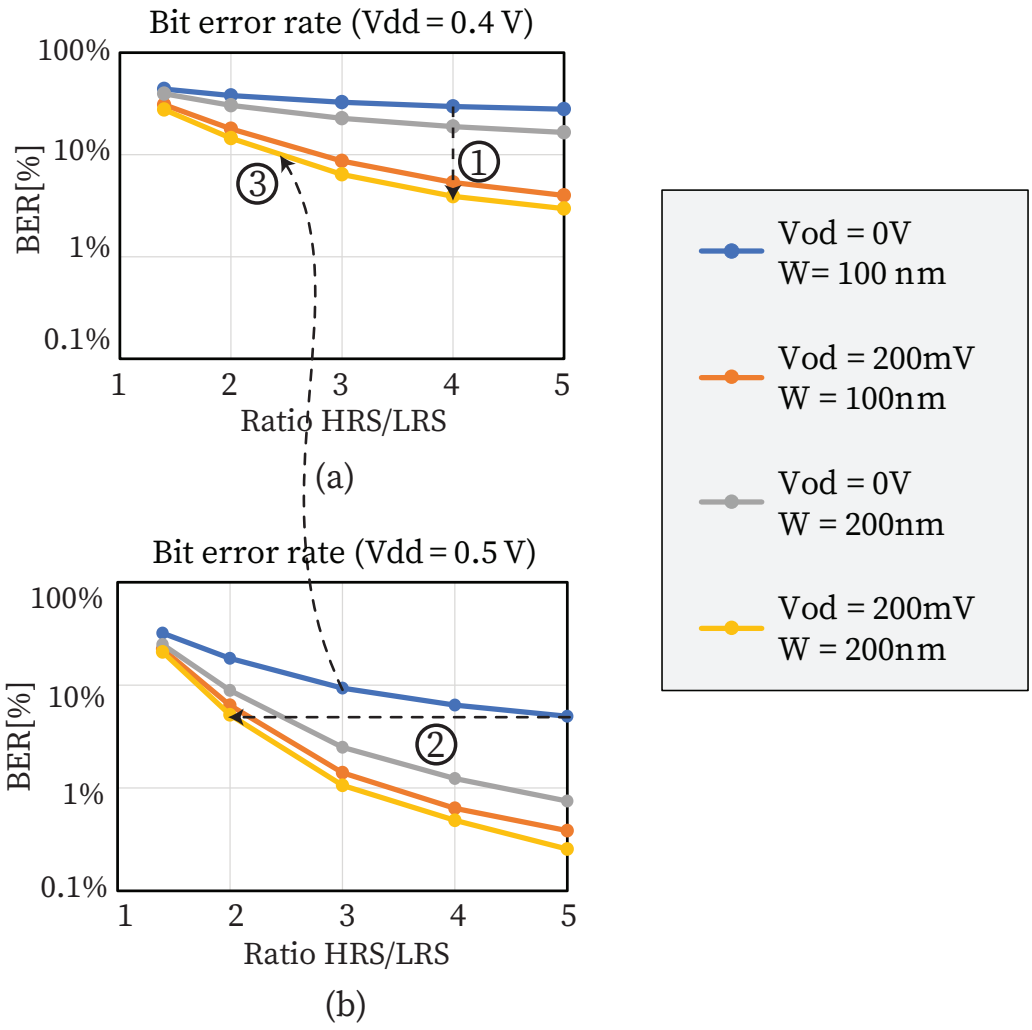


Figure 3.12. Bit-error rate for different configurations, for Vdd = 0.4 and Vdd = 0.5V, highlighting the main trends.

reduced (e.g., from 5 to 2 for a 5% BER at 0.5V). (3) At constant BER or constant ratio, such techniques can enable Vdd reduction, thus saving read energy.

3.4.2.2 Analysis of write access

The proposed read assist techniques facilitate a reduction in the HRS/LRS ratio, which allows for a lower set current [57, 21] to be applied. During a set operation, the programming current remains fixed and is unaffected by the access transistor width. In this case, the current is controlled by adjusting the V_{gs} (gate-source voltage). A larger transistor may trigger a slightly faster set, though. Conversely, the reset operation is triggered by the voltage across the memory and is thereby highly sensitive to the access

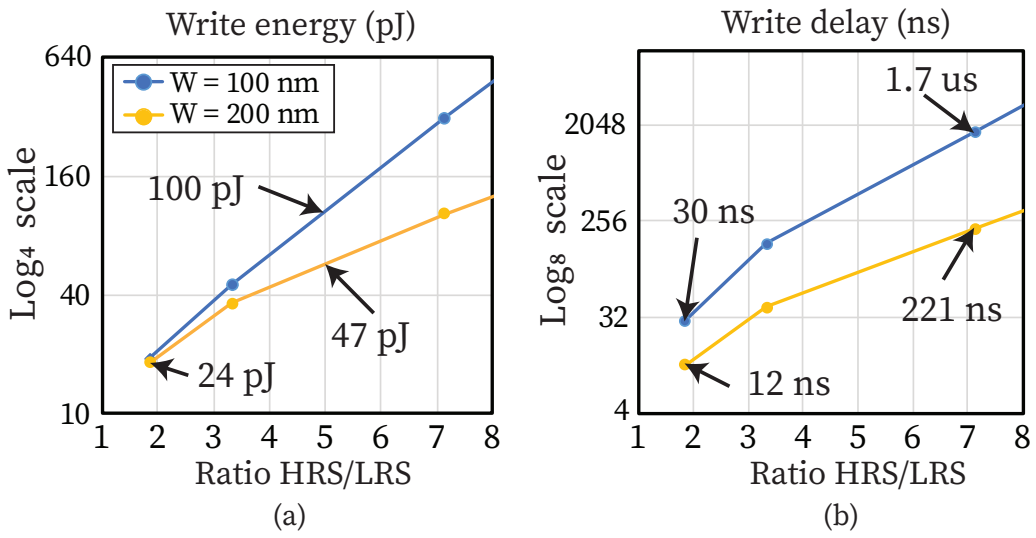


Figure 3.13. (a) RRAM programming energy versus HRS/LRS ratio. By reducing the HRS/LRS ratio from 5 to 2, the programming energy can be decreased by 2 to 3×. (b) Write delay versus HRS/LRS ratio.

transistor's equivalent resistance. In [83], the effect of gate overdrive and transistor width in reset is extensively discussed.

Figure 3.13-a shows the energy consumed during a set+reset cycle versus the HRS/LRS ratio. By doubling the size of the access transistor, it shows that for a 5× HRS/LRS ratio, the programming energy can be reduced from 100pJ down to 47pJ (2×). Then, by considering a 2× HRS/LRS ratio at constant BER, the proposed techniques further reduce the programming energy to 24pJ (3×). In addition, to illustrate the effect on write accesses of increasing the access transistor width, Figure 3.13-b shows the reset time versus the HRS/LRS ratio for 100nm and 200nm wide access transistors. Considering a double-size access transistor, the reset time can be reduced by 7.7× for a 7× HRS/LRS ratio (from 1.7us down to 221ns). Finally, as mentioned earlier, reducing the HRS/LRS ratio enables shorter programming operations at a lower current. Such reduction in the programming conditions reduces the stress received by the RRAM bit-cell. As discussed in [84], a direct relationship exists between the HRS/LRS ratio and memory endurance (i.e., a lower ratio induces a longer lifetime). In that context, the assist techniques in reading may open the way for more relaxed programming operations, thereby enhancing the lifetime of RRAM-enabled systems.

3.5 Error resilient bit-line computing architecture

In this contribution, I explore how combining hardware and software strategies can significantly increase the resilience of BC architectures. Bit-line computing core operations are the bit-wise AND and NOR. Conveniently, these operations also provide the basis for efficient in-memory error detection and mitigation, as I show in this section. While a recent article presented an error correction mechanism for crossbar IMCs [85], this is the first time that an error detection approach targeting BC architectures is presented [60].

The resilient BC architecture is based on local groups (LGs) as depicted in Figure 3.14-a. However, unlike the hybrid BC architecture, the resilient BC architecture employs SRAMs exclusively. Each array row stores a data word and an additional parity bit. Rows are organized in local groups (LGs). Two address decoders are present. They concurrently assert two WL signals simultaneously when performing BC operations requiring two operands. Only one is employed for non-BC memory accesses (reads and writes) and for operations requiring a single operand, such as bit-wise negation and shift. Indeed, two-operands BC operations are possible between any two words as long as they belong to different LGs.

Within the Bit-Line Computing Unit (BCU) located in the periphery of the array (as shown in Figure 3.14-b), the read/write circuitry interfaces with Bit-wise and Arithmetic Logic (BAL). The BAL circuit performs various operations, such as in-memory additions, subtractions, shifts, and bit-wise operations. These in-memory operations can be leveraged to perform multiplications efficiently [41]. The BAL block does not cause memory access overhead when normal memory reads are performed. In addition, the circuitry dedicated to error mitigation is also found in the BCU. Its main components perform parity check and generation, manage the parity bit logic, and the detection and mitigation features. The description of this circuit is presented in detail in Section 3.5.1. External to the BCU, a controller orchestrates its operation, dictating which operation is executed at each clock cycle.

3.5.1 Error detection and mitigation strategy

The BC array outlined above provides three features to enhance error resiliency: parity generation, parity check, and error mitigation. These are performed entirely in the BCU block and are hence transparent from a system and application perspective. Jointly, they counter the effect of single bit-flips both on standard memory accesses and during single- or dual-operands BC operations. When an error is detected, the value "0" is written to memory (in case of an in-memory operation) or presented at

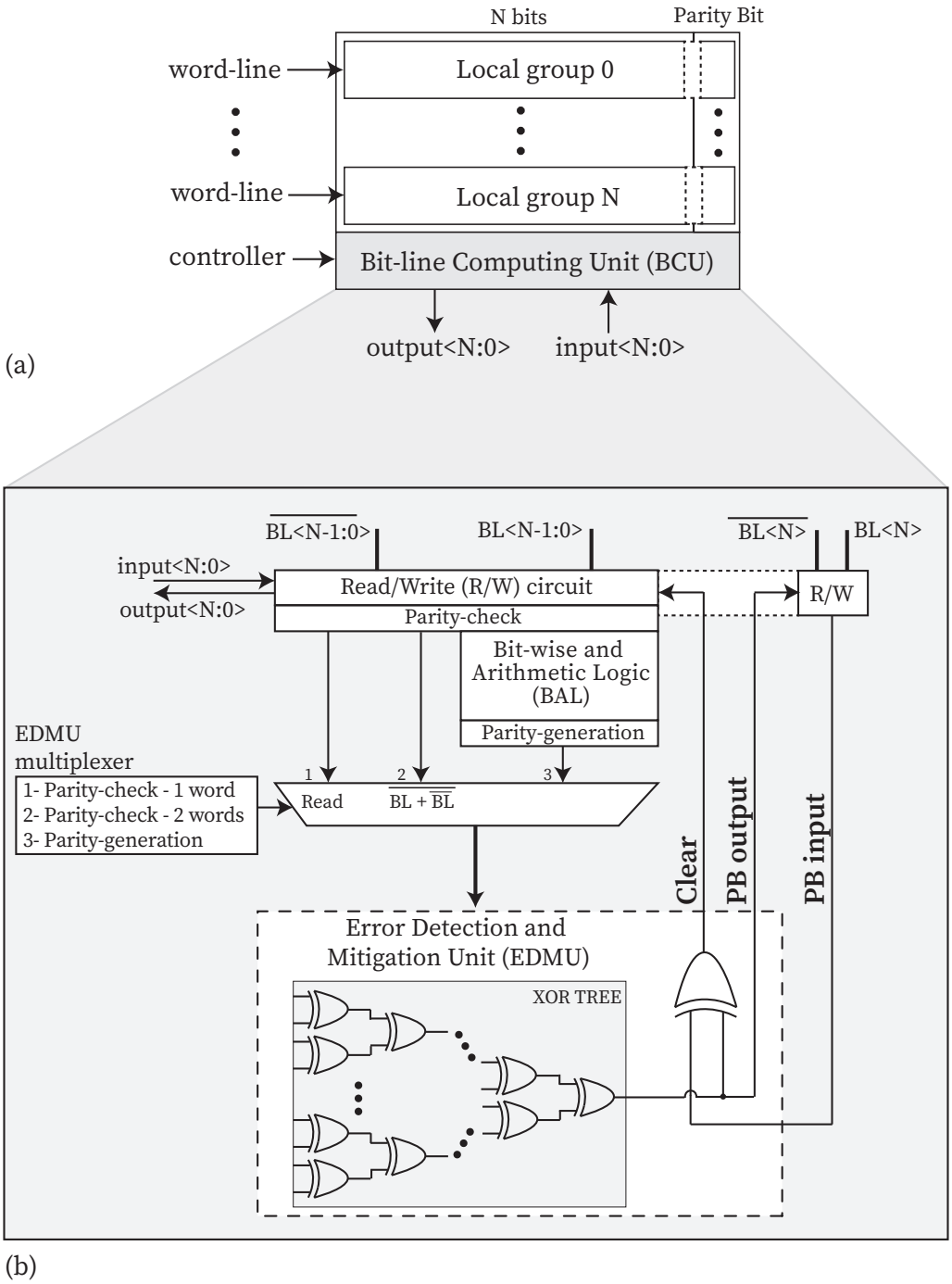


Figure 3.14. (a) BC memory architecture comprising N-bits words and one parity bit. (b) the Bit-line Computing Unit (BCU) comprises the read/write circuit, the Bit-wise and Arithmetic Logic, and the Error Detection and Mitigation Unit (EDMU), which comprises the XOR-tree and an extra XOR gate to compute the Clear signal.

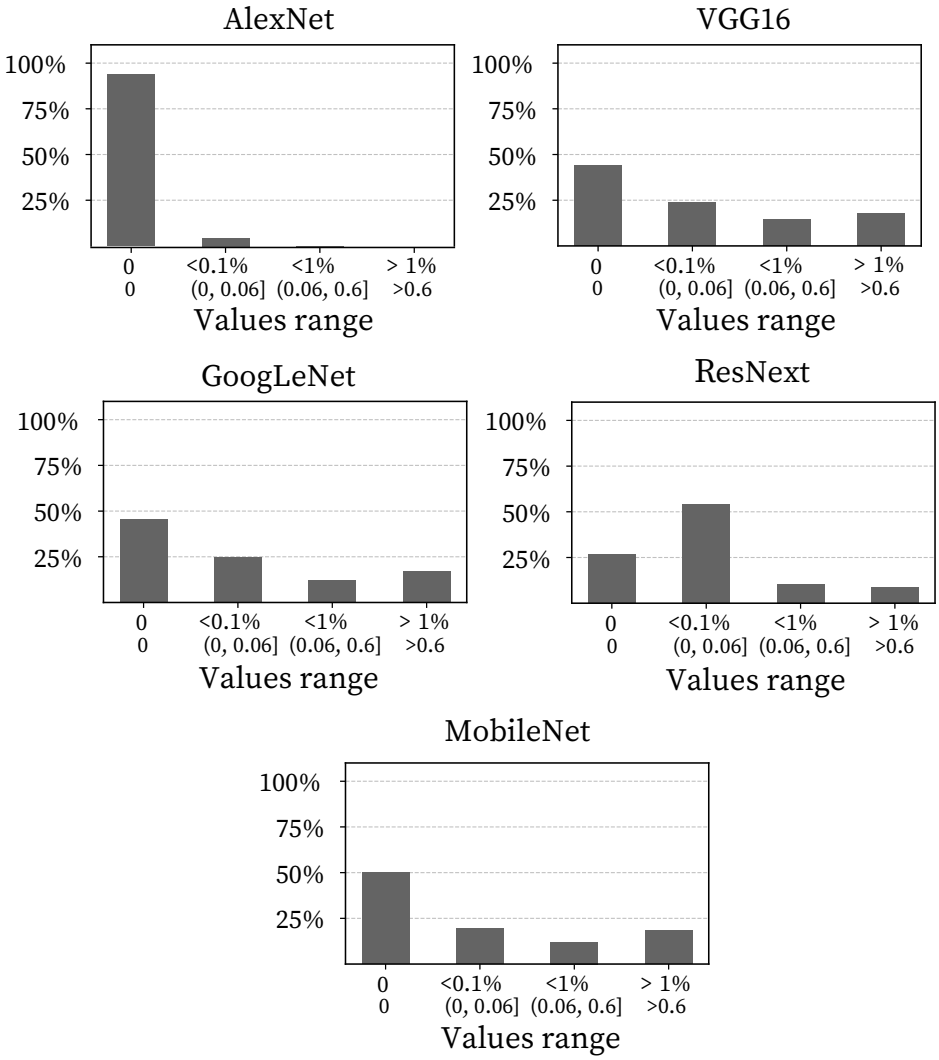


Figure 3.15. Activations' distribution in single-instance CNNs on the CIFAR-100 dataset. More than 75% of activations assume values within 1% of the representable range, with a significant fraction of them being exactly 0.

the memory output (in case of standard memory access). This choice is motivated by the plots in Figure 3.15, which show the statistical distribution of the computed values (activations) in CNNs is highly skewed towards zero, with only a few outliers having a high magnitude. For AlexNet, more than 90% of the activation values are 0, while for RexNext, 75% of them are smaller than 0.1% of the representable range, i.e., values very close to zero.

Error detection is straightforward when only one operand is involved, requiring all bits' XORing ($\oplus$), including the parity. When instead a two-operand BC operation is performed, e.g., between two words $\mathbf{A} = \{A_{n-1}; A_{n-2}; ...; A_0\}$ and $\mathbf{B} = \{B_{n-1}; B_{n-2}$

Table 3.3. BC operation between Q_0 and Q_1. Bit-lines behave as the logic gates AND and NOR. An XOR between the memory cell values is derived using an additional NOR gate on the bit-lines.

Q_0	Q_1	BL	$\overline{BL}$	NOR(BL,$\overline{BL}$) = XOR(Q_0,Q_1)
0	0	0	1	0
0	1	0	0	1
1	0	0	0	1
1	1	1	0	0

;...;B_0}), only the values ($BL_i = A_i \,.\, B_i$) and ($\overline{BL}_i = \overline{A_i + B_i}$) are available, but not A_i and B_i themselves, since they are accessed simultaneously. The expression for $A_i \oplus B_i$ can nonetheless be computed from the bit-line values with a NOR gate (as shown in Table 3.3).

$$A_i \oplus B_i = \overline{BL_i + \overline{BL_i}} \tag{3.1}$$

Hence, parity checking in can be performed based on *both* bit-line signals, as follows:

$$\begin{aligned} Parity &= (A_{n-1} \oplus B_{n-1}) \oplus ... \oplus (A_0 \oplus B_0) \\ &= \overline{(BL_{n-1} + \overline{BL_{n-1}})} \oplus ... \oplus \overline{(BL_0 + \overline{BL_0})} \end{aligned} \tag{3.2}$$

At the center of the detection/mitigation circuitry is a tree of XOR gates (named Error Detection and Mitigation Unit, or EDMU, in Figure 3.14). At each memory access, the EDMU is employed to detect if a parity error occurred. The "Clear" signal is assessed to zero the value at the read/write block output if an error is detected. The EDMU inputs are either the bit-line signals (BL) for standard memory reads and single-operand BC operations or the bit-wise NOR between BL and $\overline{BL}$ signals for two-operands ones, as discussed in the previous section. The selection of proper inputs is dictated by the EDMU multiplexer, governed by the memory controller.

In the case of a BC operation, a new parity bit must be generated. To this end, the EDMU is used again, having the value computed by the arithmetic logic block at its input, i.e., the result obtained by a given in-memory operation. The output parity bit is then written back to memory simultaneously with the data bits. All the actions outlined above are executed in a single clock cycle.

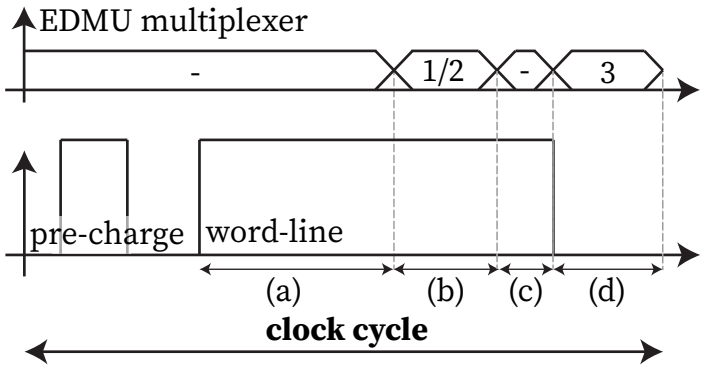


Figure 3.16. EDMU multiplexer states at run-time in one clock cycle. (a) Read one or two words. (b) Parity check and in-memory operation. (c) IMC result is available. (d) Parity calculation of the new IMC output word.

The timing behavior of the resilient BC computing solution is depicted in Figure 3.16. First, the bit-lines are pre-charged to Vdd. Then (a), the word-lines are activated to access the memory cells in the desired words. Once the voltage on the bit-lines is stable, the parity check is performed by the EDMU on the read value (b). During the phase (c), the word-lines are deactivated, and the arithmetic unit computes the desired in-memory operation. Its result is stored in a dedicated register if no parity error occurs. Otherwise, the state of the register is cleared (set to zero). Finally, in phase (d), the EDMU is employed again to generate the output parity bit. In the case of an in-memory operation, the result must be stored back in memory. This operation is performed in a different clock cycle.

While the calculation of parity check is performed in parallel with in-memory arithmetic operation in phase (b), parity generation phase (d) does incur an additional delay. Such delay is marginal, below 10% of the critical path across the experiments presented in Section 3.5.3.

3.5.2 Experimental setup

3.5.2.1 Single-instance and ensemble benchmarks

The evaluation of the proposed architecture and framework methodology was validated on AlexNet [7], VGG16 [86], GoogLeNet [87], ResNext [88] and MobileNet [58] on the CIFAR-100 dataset [89]. Differently from the testbench of the hybrid architecture (Section 3.3.5) in which only quantization was considered, in this contribution, I consider a single-instance, 2- and 4-instances ensemble implementations of E2CNN [36],

as presented in Section 3.2.1. This approach allows obtaining ensemble models that exhibit equal (or smaller) memory and computational requirements concerning the corresponding single-instance CNN. Still, it also shows increased accuracy and error robustness (as shown in Section 3.5.4), ultimately achieving better accuracy/energy trade-offs.

CNNs are trained in PyTorch by Flavio Ponzina, and the details are reported for completeness. The training used a fake-quantization approach [90] for the last 20 training epochs, at a quantization level of 8 bits for weights and 16 bits for activations. Such a setting leads to negligible accuracy drops compared to floating-point implementations. Instances of ensembles are trained independently, resulting in compressed models with similar accuracy but slightly different weight values. Each CNN instance composing an ensemble independently processes the input data at run-time, producing separate classification probabilities. These are then averaged together to compute the ensemble output.

3.5.2.2 Accuracy evaluation

To explore the accuracy achieved by different CNNs in the presence or absence of the error mitigation strategies, the bit-flip probabilities reported in [91] for different supply voltage levels in 40nm technology were considered. The used error model targets stuck-at faults on a bit-level basis, i.e., assuming a non-zero probability that a bit is always set as a '1' or as a '0', irrespectively of its intended value.

Faults cause observable errors if they affect the representation of the accessed data. Assuming an equal probability of stuck-at-0 and stuck-at-1 faults, the probability of having an observable error in a bit in memory access is as follows:

$$P_e = \frac{1}{2} P_{stuck-at} \tag{3.3}$$

Considering a word of n bits (possibly including a parity bit), the probability of having k bit-flips during access is then:

$$P_{(num-err==k)} = \binom{n}{k} P_e^k (1-P_e)^{n-k} \tag{3.4}$$

The Equation 3.4 is part of an inference solver (written in the C language by Flavio Ponzina). When executing multiply-accumulate operations, a non-zero probability

Table 3.4. Energy consumption per access and bit error rate for an SRAM built on a 40nm CMOS process at different voltage levels (fJ/bit).

	Read	Write	BC op.	Error Rate
800mV	62.7	81.1	101.0	
750mV	46.9	50.9	74.1	1e-5
700mV	36.0	34.9	54.3	1e-4
650mV	23.7	24.8	42.3	7e-4
600mV	18.6	18.3	32.1	2e-3

of bit-flips is assumed when computing the result[4]. Without any error mitigation schemes, all bit-flips are propagated to successive computations. When instead simulating the strategy outlined in Section 3.5, results are set to zero in the presence of an odd number of bit-flips. The probability of error detection is:

$$P_{error-detection} = \sum_{k=1,3,5,\ldots}^{k=n-1} \left[\binom{n}{k} P_e^k (1-P_e)^{n-k} \right] \tag{3.5}$$

The probability of having multiple errors (e.g., 2, 3, 4, ...) in the same memory access decreases exponentially, motivating the choice of only addressing single-error mitigation.

3.5.2.3 Energy and area evaluation

To ensure consistency with the bit-error data reported in [91] and presented in Table 3.4, I implemented the resilient BC architecture using 40nm CMOS technology. This choice of technology enables the obtained results to align with the reported data, thus providing a reliable basis for analysis and comparison. The array of 16-bit words is composed of 4 LGs of 128 words. The addresses are interleaved in 4-ways set associative police, and each LG has 32 word-lines. Consequently, each subarray stores 1kB (64BL × 128WL). To account for the parasitic effects of the physical layout, I considered RC networks based on the width, length, and metal layer. The energy required to read, write and perform a BC operation is reported in Table 3.4. As a baseline, I considered an iso-size BC array as in [31], which does not feature error mitigation.

[4]MAC operations are executed as a sequence of shift-adds among two operands, in which each of the two may be affected by stuck-at faults.

To assess energy requirements for different benchmarks, I measured the number of reads, writes, and BC operations required by inference on each of the considered benchmarks through a cycle-accurate simulator, such as in Section 4. However, the mapping strategy used in this contribution diverges from that used in the hybrid architecture (Section 3.3), in which it considers that a full-SRAM BC architecture would write the kernel inside the memory to perform the BC operation. Here, I consider a more advanced mapping strategy that embeds the kernels in the BC instruction and modulates the multiplication performance based on a heterogeneous quantization method, which is discussed in Section 4.4.

3.5.3 Area, energy and performance breakdown

Figure 3.17 shows the floor plan of the subarray implementation. The total area of the subarray is 3448 μm^2, of which 76.9% (2651 μm^2) is occupied by the high-density SRAM bit-cells from TSMC and the sense-amplifiers of the 4 LGs. Each bit-cell has a surface of 0.253 μm^2. The SRAM bit-cells occupy most of the LG surface, with the LG sense amplifier representing 20% of its total area. Moreover, an overhead of 5.9% of the 4 LGs is necessary to store the parity bit. Finally, the overhead of the BCU to enable arithmetic operations and the error mitigation strategy is just 12.4%.

The energy values per bit for the read, write, and BC operations are presented in Table 3.4. Reducing the supply voltage from 800mV to 700mV reduces the energy of each operation by 47%, while at 600mV, the total energy reduction reaches 72%. However, considering the implementation with parity-bit, the read and BC operation energy increases by 15%. This overhead is due to the parity bit access and the EDMU, used twice during the same cycle in these operations, as described in Section 3.5.1.

Operating at sub-nominal voltages forces a reduction of the operating frequency. Indeed, a supply voltage of just 650mV reduces the maximum operating frequency by more than 40% compared with the 800mV baseline. Nevertheless, in this condition, the architecture can still operate at 300MHz, a relatively high frequency in ultra-low power devices. Moreover, in-memory operations are parallelized by employing multiple subarrays to increase throughput, compensating for the frequency reduction.

3.5.4 Accuracy/energy trade-off

The accuracy achieved at different sub-nominal voltages is presented in Figure 3.18, where the energy cost of inference for different benchmarks is shown on the x-axis. Black curves correspond to baseline single-instance CNNs, and blue ones represent

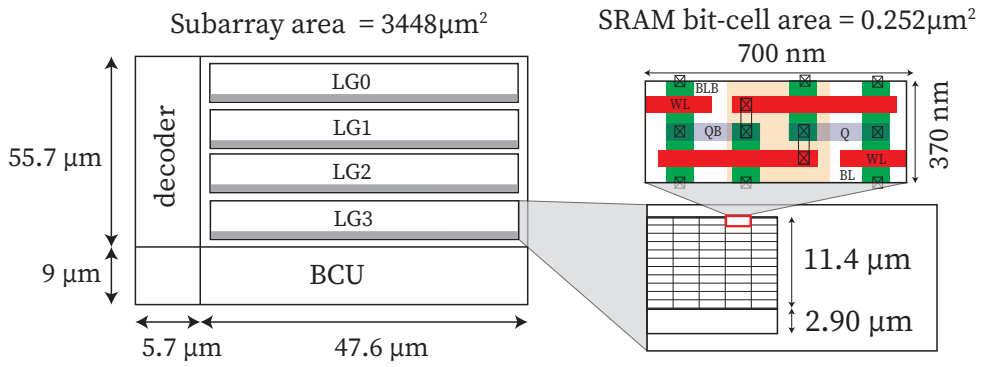


Figure 3.17. Area breakdown of a single BC subarray.

the same models in which the error mitigation strategy is applied. Finally, green and red lines correspond to ensemble-based solutions, including the proposed error mitigation strategy. Different markers highlight different supply voltages (hence error rates) as introduced in Table 3.4. As for voltage supply, it is considered that it ranges from 800 mV (a level in which no error occurs) down to 600 mV. These results indicate that voltage scaling dramatically impacts the accuracy of baseline solutions. In particular, by slightly reducing the voltage from 800mV to 750mV, the accuracy of the considered benchmarks is reduced to 59.8% on average. The experiments report similar effects in ensembles (i.e., where the error mitigation strategy is not implemented), highlighting that, despite their increased accuracy and robustness, errors affecting activations are still critical [36].

Figure 3.18 also demonstrates the accuracy improvements of the error mitigation approach at any evaluated sub-nominal voltage. More precisely, star points show that more aggressive voltage scaling can be applied in the BC architecture, which retains a better accuracy level. The minimal energy overhead due to the additional circuitry performing the parity check is compensated mainly by the possibility of reducing the supply voltage. Thus, it ultimately results in a more favorable accuracy/energy trade-off.

On average, energy savings of 41.2% can be achieved with the presented methodology while preserving the baseline accuracy. Combining the error mitigation strategy with the described ensembles also results in even more advantageous energy/accuracy trade-offs. The error mitigation strategy reduces the inexactness introduced by memory errors in the activations, thus limiting their impact on the accuracy of ensembles that can achieve, on average, 8.2% higher accuracy than single-instance CNNs, when the proposed solution is applied. In this context, ensembles serve two purposes: on one side, they increase the initial inference accuracy at a nominal voltage (i.e.,

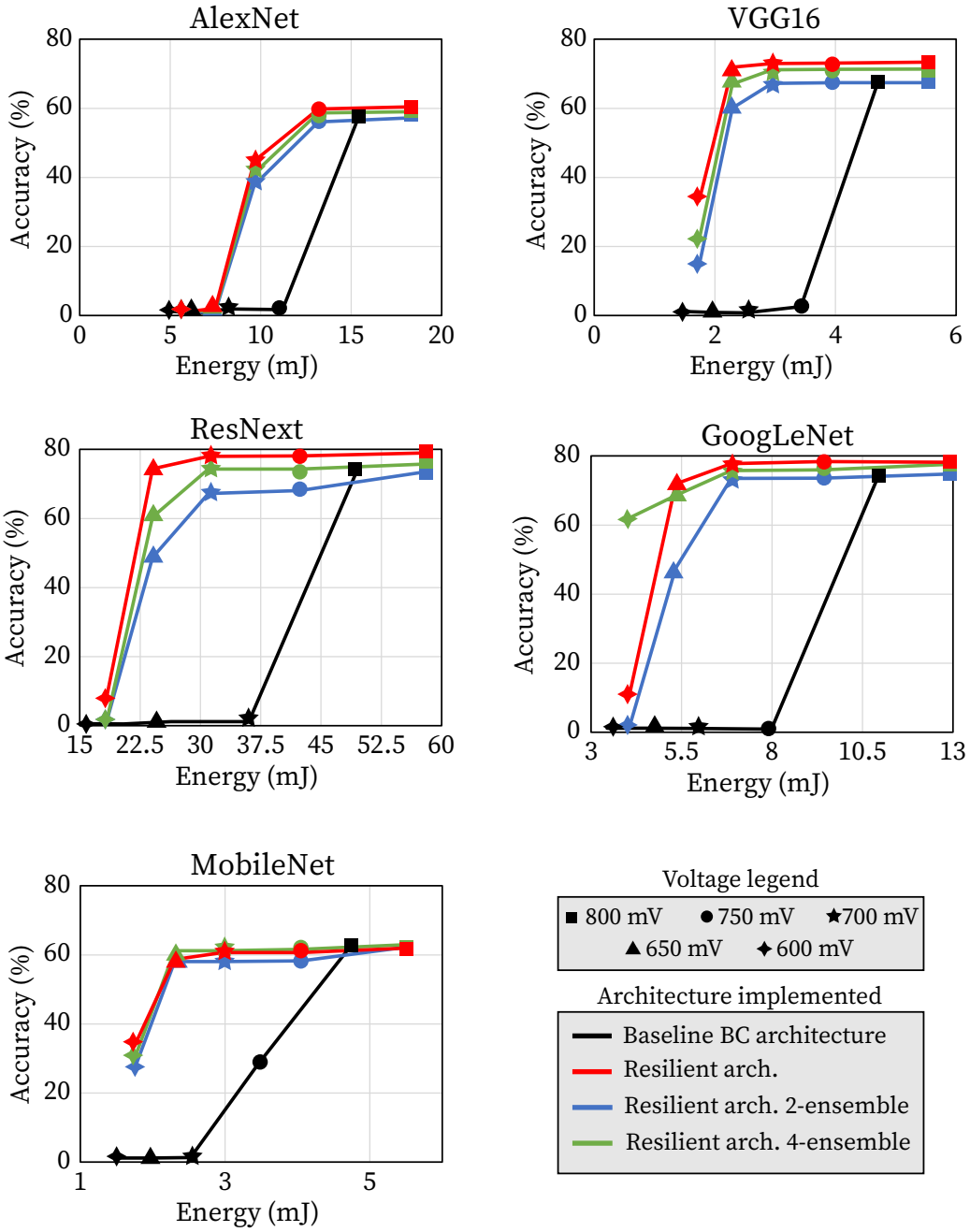


Figure 3.18. Energy-per-inference vs. accuracy, varying the supply voltage. Black line: baseline BC implementation. Blue-green-yellow lines: BC array featuring error detection and mitigation, employing different ensemble sizes.

error-free executions), and on the other side, their additional robustness against errors is exploited to enable more aggressive voltage scaling. This effect is particularly evident in Figure 3.18 for VGG16, GoogLeNet, and ResNext, where the curves of en-

sembles exhibit a smoother accuracy degradation due to voltage reduction compared to single-instance alternatives.

In GoogLeNet, the 2-Ensemble option offers significantly higher accuracy than the 4-Ensemble one at 650mV, while the latter configuration outperforms the former at any other voltage level. This behavior has already been discussed in [36], where the authors underline that larger ensembles require more aggressive pruning on the initial CNN. The result is a lower accuracy of the individual pruned CNNs that the ensemble, in the presence of high error rates, may not recover. Nevertheless, combining the error mitigation technique with ensemble-based solutions reduces the supply voltage to just 650mV, resulting in energy savings of up to 51.3% with minimal impact on the initial CNNs accuracy.

3.6 Conclusion

Energy-efficient computing is critical to unlocking AI's potential at the edge. However, uniform solutions that rely on hardware or software methods do not reach a suitable energy efficiency for edgeAI. Otherwise, the most promising solutions synergically combine hardware and software methods. At the hardware level, I have proposed a BC architecture using new emerging RRAM and studied the effect of voltage scaling in both SRAM and RRAM. Finally, to leverage even more the benefits of the hybrid and resilient BC architectures, software-optimized CNNs were considered.

First, in this chapter, I have proposed a hybrid BC architecture tailored to the workload characteristics of deep neural networks. The proposed BC architecture embeds volatile and non-volatile bit-cells, conforming to the characteristics of data accesses at run-time. I have demonstrated that the proposed integration of SRAM and RRAM technologies can be effectively managed from a layout and electrical perspective. Furthermore, I have shown that WDM and hybrid IMC drastically reduce (up to 60×) the data transfer required to process a complete AlexNet or MobileNet CNN inference. Consequently, up to 93% energy efficiency and 6× performance improvement have been achieved for these workloads.

Next, I have explored the trade-off between density and dependability in RRAM memory architectures. I have shown that increasing the width of the access transistors on a 1T1R topology, such as in the hybrid BC architecture and overdriving the WLs, allows reading accesses at near-threshold voltage levels in RRAMs. On the contrary, higher Vdd levels are required to achieve acceptable BERs when minimal width sizes are employed. These techniques counter the intrinsic CMOS variability, reducing the BER during ultra-low-voltage read operations. Furthermore, I have demonstrated that

such techniques can enable constant BER with a lower HRS/LRS memory ratio, thus paving the way for lower energy programming operations (2 to 3× energy reduction in the proposed experiments).

Finally, I have proposed a new resilient BC architecture designed to support aggressive supply voltage scaling when running CNN inference, thanks to implementing a transparent error mitigation technique. Additionally, I have shown how the synergic use of constrained CNN ensembles and the proposed error mitigation method can improve the robustness against memory errors. By reducing the voltage from the nominal 800mV to just 650mV, the resilient architecture has shown energy savings of up to 51.3% without affecting the initial CNN accuracy.

4 Enabling CNN Inferences for EdgeAI Applications

This chapter proposes strategies to quantize, compress (and decompress), map, and accelerate Convolutional Neural Networks (CNNs) in compute memories. The mapping strategy classifies the workload's data types based on their behavior, and by finding common operands during the execution of the CNN, it avoids unnecessary data transfers. This strategy opens exciting opportunities to bridge CNN software and hardware optimizations, achieving up to a 20× reduction in cycle count, while only experiencing a minimal accuracy degradation of 1%. Moreover, due to the hardware-software optimizations, a lightweight encoding/decoding scheme leverages the deep connection of the optimized CNN model and the compute memories instructions. Considering both the quantization and the encoding, the average bit-width of the CNNs' weights decreased from 8 bits to 2.2 bits. Finally, the mapping strategy presented in this section was used to validate the results of Chapter 2 and Chapter 3.

4.1 Introduction

Thanks to their ability to extract abstract information from raw data acquisitions, deep learning algorithms such as Convolutional Neural Networks (CNNs) are fostering a revolution in multiple and diverse fields, from personal mobility to health care. Nevertheless, the increased accuracy of recent CNN models comes at the cost of massive memory requirements and intense workloads [61].

These downsides are particularly important for edge devices running artificial intelligence algorithms, a scenario named edgeAI in literature [92]. Computational efficiency is key in edge AI because applications must often comply with real-time constraints. Such constraints must be met within tight computing and energy budgets, commonly orders of magnitude lower at the edge when compared to the cloud, thus requiring careful hardware and software optimization. The main avenues towards op-

timizing deep learning workloads leverage these algorithms' high levels of parallelism and robustness.

In CNNs, parallelism is enabled by their structured and repetitive computing patterns based on Multiply-ACcumulate (MAC) instructions, millions of which are employed to implement their convolutional and fully connected layers. Indeed, a high degree of data reuse is present when convolving filters with activations (in convolutional layers) and executing matrix-vector products (in fully connected ones). This characteristic can be harnessed by Single Instruction, Multiple Data (SIMD) strategies to increase efficiency and performance [69].

Moreover, due to their robustness, CNNs can be optimized with very little or no accuracy drop by reducing the required MAC operations or simplifying their computation. For example, using fixed-point arithmetic instead of floating-point arithmetic only requires integer hardware, resulting in more energy-efficient operations. Quantization approaches advocate using fixed-point formats in contrast to floating-point, enabling only a few bits to represent the parameters (weights) and intermediate values (activations). Pruning strategies focus on coarser granularity, seeking to skip weights and MAC computations with little impact on the output quality. As detailed in Section 4.2, pruning and quantization are often combined in state-of-the-art edgeAI strategies.

In addition to software optimization, the rise of edge AI has motivated the computer architecture and hardware research community to introduce dedicated designs. Approaches range from processors-based solutions, such as the ultra-low-power PULP multi-core [54], to custom accelerators [69]. In this context, compute memory architectures are particularly appealing, as computation inside or near memory avoids energy-expensive data movements in-between processing and storage components. In contrast, the parallelism made available by the regular structure of memory banks presents a good opportunity to support the SIMD patterns in CNNs.

SRAM arrays are usually distributed in several banks, called subarrays. In conventional cache memories, this organization is transparent, and the entire array is abstracted as a unit. Memory read and write operations are performed through the system's bus, which usually transfers from one to a few words per clock cycle. Otherwise, compute memories are the architectures that profit from the high bandwidth at the periphery of each subarray. Moreover, since additional logic is attached to each subarray, employing a complex circuit may induce a high area overhead. Thus, as discussed in Chapter 2 and Chapter 3, the processing elements of these architectures are limited to adders and shifters to perform multiplications.

This contribution addresses the fundamental hardware/software co-design challenge

by providing a holistic framework for the optimization, deployment, and execution of CNN models on a compute memory architecture for edgeAI computing. I combine a novel CNN optimization strategy with highly optimized compute memory architectures. Both support fine-grained quantization and pruning in fully connected and convolutional layers. Moreover, leveraging the statistical distribution of weight values in CNNs, the methodology features a novel weight encoding strategy during CNN optimization and in the compute memory hardware implementation. The strategy, named Generic Convolutional Weights (GCW) encoding, uses fewer bits to encode weight values that appear more frequently and a higher number of bits for those that are only rarely used, reducing the quantized model sizes by up to 4× in the experiments. A dedicated pipeline is in charge of decompressing the model representation at run-time without impacting performance, converting it to a sequence of compute memory instructions. Operations are then executed in parallel on multiple subarrays, greatly reducing run-time. In summary, the contributions of this chapter are:

- I present a synergic hardware and software framework that employs fine-grained bit-widths, data compression, and in-memory parallel computing to support edgeAI applications with high energy efficiency.

- I introduce a strategy to map the parameters of convolutional and fully connected layers on subarrays, maximizing operations parallelism and data reuse.

- I present a compression strategy, called GCW encoding, to compress CNN models losslessly. Also, I describe a corresponding decoding circuit operating at run-time and show that this circuit required little area and no timing overhead.

Section 4.2 discusses related works on CNN optimization/compression and compute memory architectures. Section 4.3 introduces an overall view of the proposed framework. Section 4.4 describes the mapping of convolutional and fully connected layers of CNNs on compute memory architectures. Section 4.5 focuses on software optimization by detailing the CNN optimization methodology and the compression approach. Next, Section 4.6 presents the compute memories instructions and the proposed pipeline circuit for GCW decoding. Section 4.7 provides details on the experimental setup and the achieved results. Finally, Section 4.8 concludes the chapter.

4.2 Background

4.2.1 Convolutional neural networks

CNNs have revolutionized the field of computer vision in recent years, enabling significant progress in image classification, object detection, and other related tasks. One of the earliest CNNs to achieve notable success was LeNet [93], proposed in 1998. LeNet was designed for handwritten digit recognition and consisted of several layers of convolutional and pooling operations, followed by fully connected layers for classification. In 2012, AlexNet[7] was introduced, a much deeper and more complex CNN architecture. Its architecture is depicted in Figure 4.1. CNNs, in general, adapt their layers and model sizes to the function of the input figure. In Figure 4.1, it considers a 32×32 RGB figure, which is also considered in the experiments of this chapter.

VGG16 [86] is another influential CNN architecture proposed two years after AlexNet. Its deep structure is characterized by 16 layers of convolutional and pooling operations, followed by three fully connected layers for classification. The VGG16 architecture has a very homogeneous structure and has achieved impressive performance on various computer vision tasks. MobileNet [58] is a CNN architecture designed for efficient mobile and embedded applications, focusing on minimizing the parameters and computation required. Finally, XCeption [94] was introduced in 2017 by Google researchers, and such as Mobilenet, it uses depthwise separable convolutions, which factorize the standard convolution operation into separate depthwise and pointwise convolutions, reducing the number of parameters and computations required while maintaining high accuracy. Overall, these CNN architectures have demonstrated the potential of CNNs to achieve remarkable performance on challenging image recognition tasks. In this chapter, I considered the mentioned CNNs to evaluate the hardware-software co-design strategy since they cover various scenarios for CNNs applications, from lightweight (e.g., LeNet) to heavy (e.g., XCeption).

CNNs process input data employing a layer-based structure, extracting increasingly more abstract features in deeper layers. The compute-intense workload and large memory requirements of CNNs are mostly due to convolutional and fully connected layers. Thus, the optimizations presented in this chapter focus on these two types of layers. In convolutional layers, three-dimensional matrices of CNN weights (named filters) are convolved over three-dimensional input feature maps, producing one output element for each filter position in the input. Conversely, fully-connected layers (usually included after convolutional ones) compute linear transformations, multiplying the input feature vectors by the weight matrixes. Fully-connected and

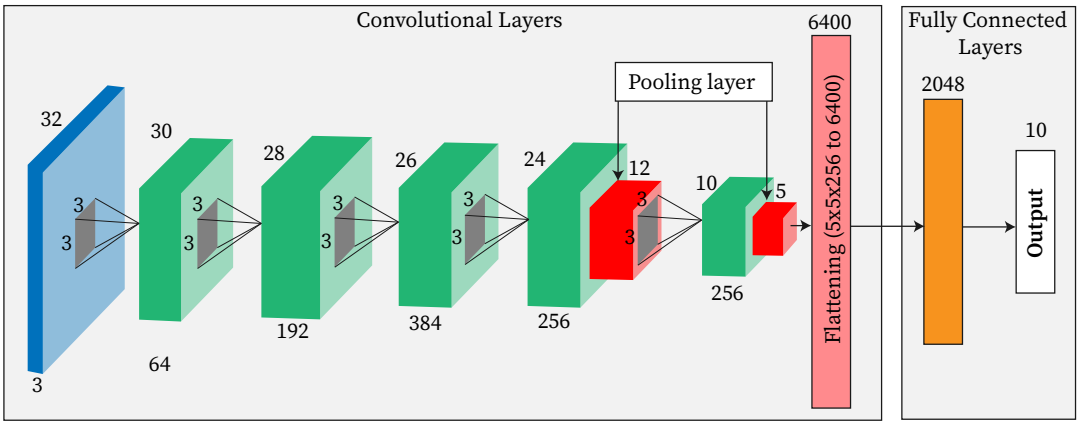


Figure 4.1. AlexNet CNN model for a 32×32×3 input activation.

convolutional layers have different data access patterns. In contrast to convolutions, weights in fully connected layers are used only once during an inference because each column of the weight matrix multiplies the input vector to produce one output element. This difference is key for the data mapping strategy discussed in Section 4.4 because it leads to different parallelization strategies.

The number of multiplications and data transfers required to compute the output of a convolutional layer in a CNN depends on the size of the output activation, the size of the filters (also known as kernels), and their quantity. Assuming an input of size $H \times H \times C$, a filter size of $F \times F \times C$, and K filters, the number of multiplications and data transfers required to compute the output feature map of size $P \times P \times K$ can be calculated as shown in Equation 4.1.

$$\begin{aligned} Multiplication &= F \times F \times C \times P \times P \times K \\ Input &= H \times H \times C \\ Output &= P \times P \times K \\ Kernel &= F \times F \times C \times K \end{aligned} \tag{4.1}$$

4.2.2 CNN models compression

Pruning and quantization are the most common approaches exploiting the inherent redundancy in CNNs to reduce their complexity, hence supporting their deployment in constrained devices. In pruning, either individual weights [65] or entire convolutional filters [66] are removed, achieving model compression higher than 10× [95]. Instead,

in quantization techniques, the weights and activations comprising CNNs models are represented using low bit-widths integer data representations instead of floating-point numbers [96]. Quantized CNN models have a smaller memory footprint than floating-point ones. Furthermore, they use less complex integer hardware to compute MAC operations, reducing energy requirements. It has been shown that 8-bit quantization can usually be implemented without affecting the initial CNN accuracy [97].

A further approach to compress CNN models is weight encoding. It is often applied after quantization, as low bit-width representations constrain the set of admissible values. Encoding can be implemented by employing different strategies. Codebook-based strategies limit the number of unique weights and store them in small look-up tables (i.e., named codebooks), encoding the baseline model into a set of binary indexes that address specific code-words[98]. Another weight encoding approach is to leverage the statistical distribution of weight values to compress their representation, employing shorter code-words for the most used values and longer code-words for rarer ones[39, 99].

Since encoding weight values does not change CNN models but only operates on data representation, it does not degrade accuracy. On the other side of the coin, run-time decoding may introduce overheads in time, area, and energy requirements. The authors of [39] use Huffman codes to index a codebook storing a constrained set of CNN weight values. While the proposed GCW strategy (detailed in Section 4.5) has some similarities with respect to Huffman coding, it does not require explicit look-up tables, minimizing the cost of its hardware implementation.

4.2.3 Compute memories

Compute memories hold great promise in accelerating AI algorithms due to their ability to exploit the data-centric nature of these applications. AI algorithms process a large volume of data. Compute memories enables this data to be stored and manipulated directly in memory without frequent data transfers between the memory and the processor [2]. Static random-access memory (SRAM) is an especially promising technology for implementing compute memories because it can be integrated with complementary metal-oxide-semiconductor (CMOS) logic near (or inside) the memory (Figure 4.2-a), enabling high parallelism and reducing data transfer requirements [100]. SRAM-based compute memory architectures are designed to take advantage of the regular structure of SRAM arrays [23], consisting of multiple subarrays typically connected to a system bus via an H-tree interconnect (Figure 4.2-b).

Compute memories can be divided into in-memory computing [31] and near-memory

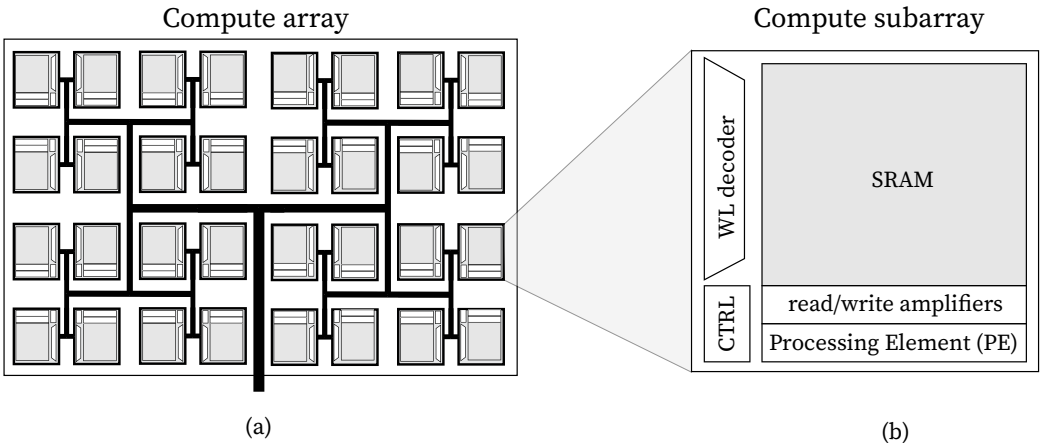


Figure 4.2. (a) H-tree composition of SRAM subarrays that can compute in parallel. (b) Basic elements of a computing subarray.

computing [47]. The former computes data as part of the data access, while the latter reads the memory and computes the data independently. These designs enable the simultaneous processing of multiple data items, which is essential for the high-performance requirements of AI algorithms. Moreover, the regularity of the SRAM array structure makes it easier to design and optimize the compute memory architecture for different AI workloads, leading to improved efficiency and performance. As AI continues to play an increasingly important role in various industries, developing more efficient and high-performance compute memory architectures will be critical in meeting the demands of these workloads.

4.3 Co-design framework

Figure 4.3 offers an overview of the optimization framework, illustrating the pathway from an initial (floating-point, non-optimized) CNN model to a tailored implementation in compute memories. Because of the typically large size of intermediate features in CNNs, the convolutions and the matrix-vector operations in fully connected layers are decomposed into smaller computing blocks (Figure 4.3.C). This tilling process, the focus of Section 4.4, allows large CNN models to be accelerated in limited-sized memories, minimizing the number of data transfers while exploiting parallelism to increase performance.

Application-level optimizations, as discussed in Section 4.5, leverage non-uniform quantization schemes and encoding methods to achieve an optimized model that can be effectively executed in memory, as depicted in Figure 4.3.A. This iterative process involves optimizing the bit-width of weights and activations in convolutional and fully

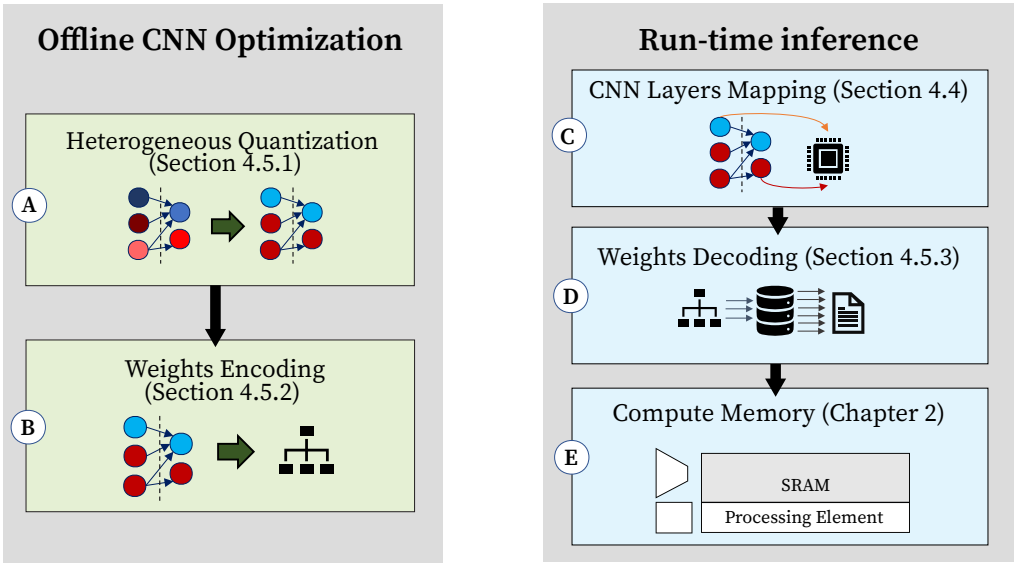


Figure 4.3. Overall view of the HW-SW co-design framework, showing algorithmic optimizations (left), mapping, and execution on compute memory hardware (right).

connected layers while ensuring that the accuracy remains above a specified threshold. By carefully managing the precision of these elements, the optimized model strikes a balance between memory efficiency and computational accuracy.

In addition, the distribution of CNN weights allows for further compression through GCW encoding, as illustrated in Figure 4.3.B. GCW selectively assigns smaller bit-widths to frequently occurring weight values and larger bit-widths to infrequently occurring ones. During the execution of convolutional layers, the weights are decoded in real-time with minimal overhead (Figure 4.3.D), as discussed in Section 4.6.2. The weight decoder translates the parameters into their two's complement representations and generates a sequence of compute memory instructions that control the execution of the memory arrays (Figure 4.3.E). The design and features of the compute memory, including support for heterogeneous quantization, are presented in Chapter 2.

4.4 Mapping CNNs to compute memories

Compute memories can effectively accelerate the execution of convolutional and fully connected layers, whose workload dominates the execution of CNN models (e.g., they constitute more than 98% of the run-time in the benchmarks in Section 4.7). When deploying a CNN layer onto compute memories, the primary goals are reducing data transfers between the subarrays and the periphery and maximizing parallelism when computing MACs. To this end, I developed and applied a series of strategies in a

cycle-accurate simulator. It relies on distributing workloads to the compute memory architecture considering hardware constraints, such as the number of available subarrays and their size, as well as application characteristics, including the type of CNN layer (convolutional or fully connected) and its geometry. Because large CNN layers may not entirely fit the limited memory size of the available compute arrays, the operations scheduler divides each layer into smaller blocks, or tiles, processed in parallel by each subarray. I detail next the specific mapping strategy for convolutional and fully connected layers.

4.4.1 Convolutional layers

In order to gain a deeper understanding of the motivations behind the mapping strategy, it is instructive to consider a hypothetical scenario involving a naive implementation of a convolution operation. The following example is intended to provide a clearer perspective on the underlying factors and challenges associated with the mapping process.

- Input: $8 \times 8 \times 3$
- Weights: 2 kernels of $3 \times 3 \times 3$
- Output: $6 \times 6 \times 2$

Table 4.1 applies the Equation 4.1 in this example. The left column shows the number of data transfers and multiplications for deployment without parallelization (e.g., single-core CPU or one compute subarray). Considering a single subarray, 192 inputs, 54 weights, and 72 outputs are transferred. The run-time, however, is dominated by multiplications, with 1944 occurrences. Therefore, the data transfer is negligible compared to the cost of multiplication, especially considering that 8-bit multiplication may require up to 16 cycles.

The convolution is effectively distributed when deploying this convolution on four subarrays, requiring 486 multiplications per subarray. However, input and kernel transfer operations increase. The input filter is tiled and transmitted to the 4 subarrays, and due to border effects, the tiles must partially overlap. In the example, $5 \times 5 \times 3 = 75$ inputs are transfers for each subarray, totaling 300 words or 56% more input words than with a single subarray. However, since the weights must be moved to all subarrays to calculate the output, the number of data transfers related to weights scales linearly, representing the biggest drawback of parallelizing. In the example with four subarrays, 216 ($4 \times 3 \times 3 \times 3 \times 2$) weights are transfers instead of 54. Indeed, when considering

Table 4.1. Number of operations of data transfers and multiplication in function of the number of subarrays (data transfers and multiplications have different cycle counts) considering a naive convolutional mapping.

Operations	1 subarray	4 subarrays	
		Per subarray	all subarrays
Input	$8 \times 8 \times 3 = 192$	$5 \times 5 \times 3 = 75$	$75 \times 4 = 300$
Output (Op)	$6 \times 6 \times 2 = 72$	$3 \times 3 \times 2 = 18$	$18 \times 4 = 72$
Weights	$3 \times 3 \times 3 \times 2 = 54$	54	$54 \times 4 = 216$
Mult/subarray	$Op \times 3 \times 3 \times 3 = 1944$	$Op \times 3 \times 3 \times 3 = 486$	486

full CNNs inferences, the run-time gets dominated by weight transfers. In [50], I show that for a full AlexNet inference in a 128-subarray configuration, 98% of data transfers are weights.

As seen in Section 2.3.2, multiplications in compute memories (either in- or near-memory computing) are performed by breaking down these operations into shift-add operations, which are based on the bits of one of the multiplication operands. In BLADE [31], the multiplication shift add instructions are generated in the periphery of the compute subarray, while the periphery receives generic instructions from the main controller. In this case, both weights and activation must be transferred to the memory for convolution.

When dealing with convolutions and matrix-vector multiplication performed across multiple subarrays, the regular pattern of operations enables the identification of a shared operand. Therefore, I propose a novel approach: rather than stream this operand to each subarray individually, I embed it directly within the instructions and broadcast them parallel to all subarrays.

Therefore, for convolutions, the input activations are considered to be locally stored in each subarray, and thus, they are called In-Memory Operands (IMO). While weights are embedded in the instructions, and thus, they are called Broadcasted Operands (BO). This strategy is named Multiple-IMO, Single-BO (MISB). Figure 4.4 shows the convolutional layer example with the data mapped to the proposed MISB strategy.

Using the MISB strategy simplifies the mapping phase and reduces issues with data locality. However, for large convolutional layers deployed in small subarrays, even the smallest tile may require more input words to compute a single output value than the subarray storage capacity. Considering a convolutional layer of 64 filters of

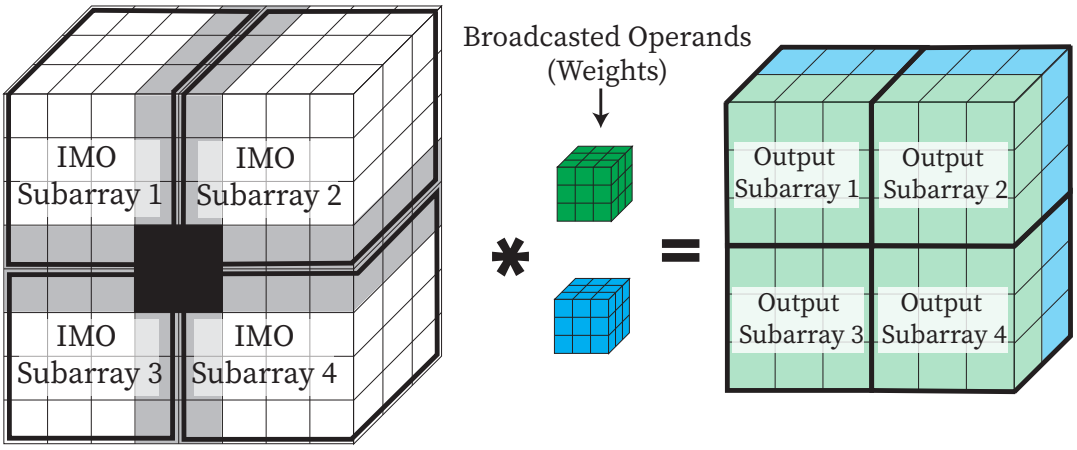


Figure 4.4. Convolutional layer activation tilling and transferred to different subarrays, while weights are converted into compute memory operations.

size 11×11×3 [1], it requires 363 input words to calculate a 1×1 output for all 64 filters (1×1×64), which could surpass the compute subarray capacity. Partial convolutions are performed in these cases, as shown in Figure 4.5. Consequently, filters are decomposed in the depth direction. Then, partial convolution results are retrieved in each compute subarray. Finally, the outputs of the partial convolutions are merged with additional in- or near-memory operations.

The MISB strategy also supports depthwise separable convolutions. This kernel separates the convolutional into two phases. First, the partial convolution, as shown in Figure 4.5, is executed without the output reconstructions. Instead, the output of the subarrays is convoluted with a 1× 1 × C where C is the depth of the output.

Depending on the geometry of the layers, the MISB strategy becomes limited for large numbers of subarrays. One characteristic of the CNNs is that at each convolutional or pool layer, activation layers' height and width decrease while these layers become deeper. For example, the last convolutional layer of VGG16 has an output activation of 4×4×512. For this layer, the maximum number of parallel subarrays operating in parallel is 16 since each computes 1×1×512 outputs. In this case, using more subarrays does not allow to leverage more parallelization.

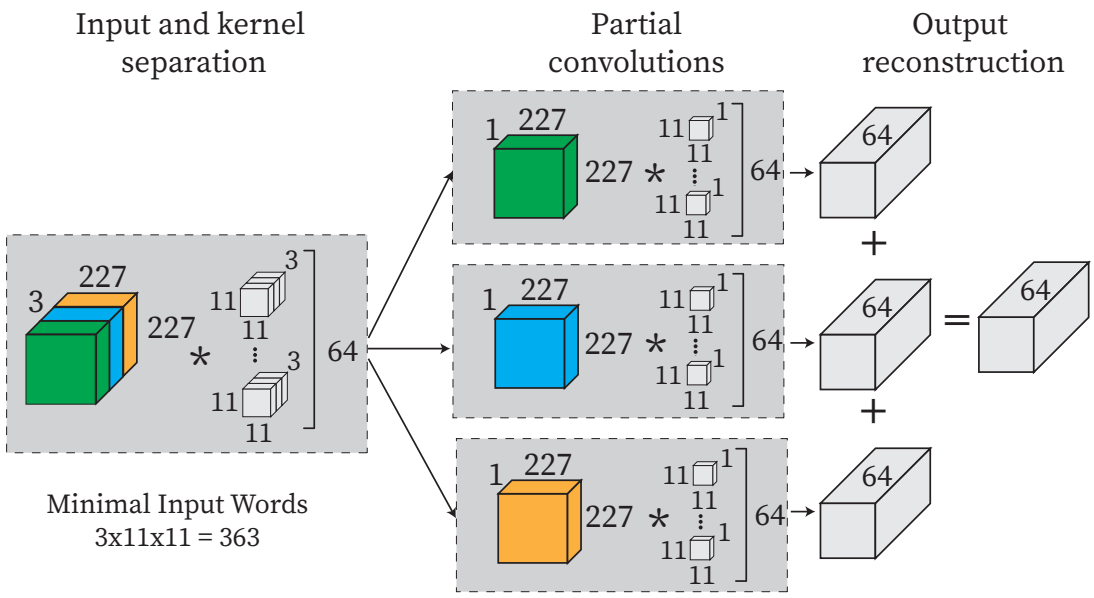


Figure 4.5. Example of convolution layer inputs that surpass the capacity of a subarray, this layer is deployed with partial convolutions, which requires an extra in-memory operation to reconstruct the output activation.

4.4.2 Fully connected layers

Fully connected layers compute each output by multiplying each input element with a corresponding weight. This process can be visualized in Figure 4.6-a. Unlike convolutional layers, where weights are shared and reused across different spatial locations, fully connected layers only use each weight once during each inference. However, each input element is employed to compute every output, resulting in significant data reuse. Therefore, the proposed approach involves storing the weights as IMOs and broadcasting the inputs, as opposed to convolutional layers. In other words, the weights are stored to allow efficient access during computation, while the input elements are broadcasted to all subarrays. This strategy takes advantage of fully connected layers' inherent data reuse property, enabling a more efficient and streamlined memory access pattern.

To illustrate this mapping strategy, consider the example shown in Figure 4.6-b. In this example, the input vector X consists of four elements, and the output Y has three elements. Consequently, twelve weights are required for the layer. The mapping of this layer into three subarrays is depicted in Figure 4.6-c. Each subarray is responsible for calculating one output, and the MAC operations are performed between the common

[1]This is the first layer of AlexNet. However, differently from Figure 4.1, this model considers a 227×227×3 input figure instead of 32×32×3 as I consider in this work.

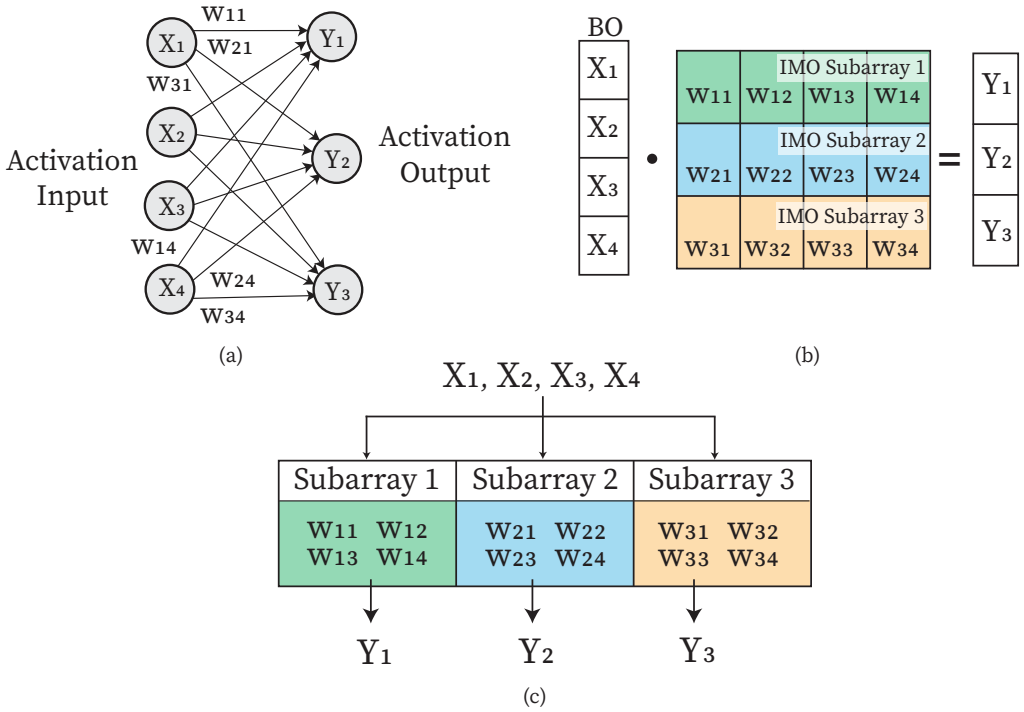


Figure 4.6. Example of fully connected layer represented in (a) graphical form and (b) matrix multiplication. (c) Data mapping to execute this workload on a compute array.

input X, which is broadcasted to the subarrays as compute memory instructions, and the stored weights W. As a result, all three outputs are calculated in parallel, leveraging the parallel computing capabilities of the subarrays.

In summary, the proposed in-memory operands and broadcasting approach effectively address the challenges associated with fully connected layers. By storing weights as IMOs and broadcasting inputs, a more optimized and efficient memory access pattern is achieved. This strategy enables parallel computation of outputs, leading to improved computational performance. Table 4.2 summarizes the MISB mapping of convolutional and fully connected layers.

4.5 Algorithmic-level CNN optimization

Algorithmic optimizations aim to decrease a CNN model's workload and memory requirements with MISB-aware transformations. The optimization process consists of two stages: first, a heterogeneous quantization and pruning step reduces the bit-width and the number of weights and activations in convolutional and fully connected layers. This phase was entirely proposed and executed by Flavio Ponzina in our joint work

Table 4.2. Assignment of operands for fully connected and convolutional layers

Operand \ Layer type	**convolutional**	**fully-connected**
Multiplicand In-Memory Operand (IMO)	Activations	Weights
Multiplier Broadcasted Operand (BO)	Weights	Activations

[42] and reported here for completeness and as a pre-requisite for the next sections. Next, I propose encoding quantized weights using variable-bit-width codes, reducing memory requirements.

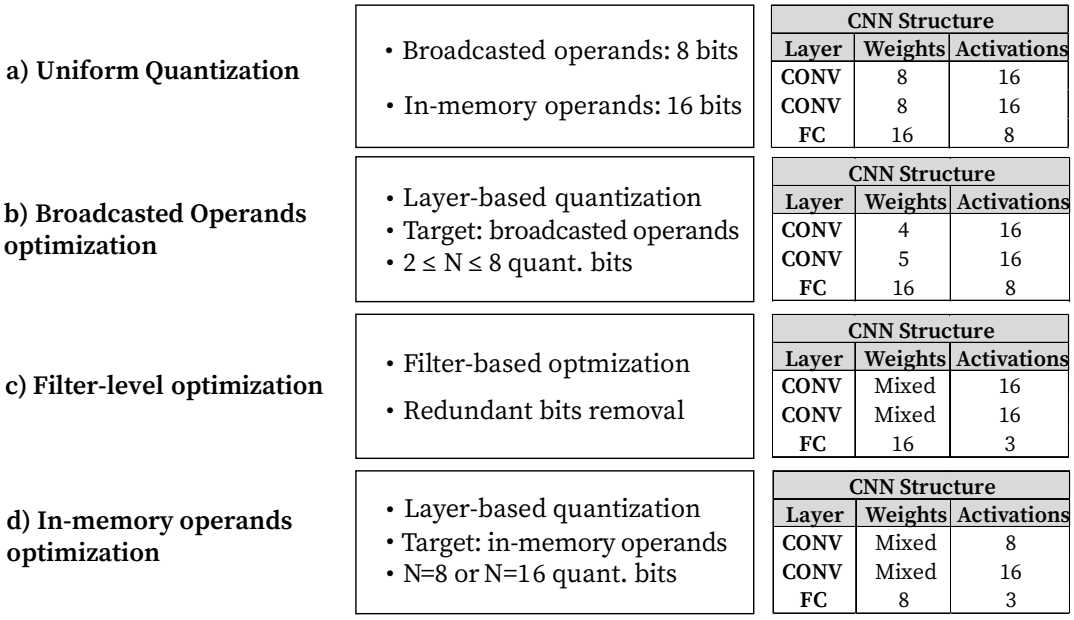
4.5.1 Heterogeneous quantization

Per-layer quantization enables aggressive model compressions, and it is directly leveraged by a compute memory using the MISB strategy. The layers in CNN have different degrees of robustness, with layers more sensitive to quantization requiring larger bit-width for activations and weights. Therefore, heterogeneous schemes reach better trade-offs between accuracy and model size. However, heterogenous quantization has a much higher degree of complexity compared to uniform quantization.

To navigate it, Ponzina introduced an iterative process that reduces the size of in-memory and broadcasted operands (IMOs and BOs, as defined in Table 4.2), according to the scheme in Figure 4.7. As a running example, the figure considers in its rightmost part a running example referring to a CNN with two convolutional and one fully connected layer, which I will follow in the rest of this section.

The input models in the optimization flow are homogeneously quantized CNNs, employing 16-bit IMOs and 8-bit BOs (Figure 4.7-a). As shown in [96], CNNs at these quantization levels have indistinguishable accuracies with respect to floating-point implementations. In the first optimization step (Figure 4.7-b), the bit-width of the BOs is independently tailored for each layer.

To this end, the flow reduces only one bit per layer, starting from the layer having the highest number of MAC operations (and, therefore, the highest potential for savings). After the CNN with the new configuration is retrained with a small number of epochs, the model has its accuracy verified, and if the degradation exceeds a user-defined threshold (1% and 5% maximum degradations are considered in the experiments of Section 4.7.1), the optimization flow backtracks. Interactively, the flow targets the layers having the second, third, etc., most numerous MAC operations. Once all



CNN Structure		
Layer	Weights	Activations
CONV	8	16
CONV	8	16
FC	16	8

CNN Structure		
Layer	Weights	Activations
CONV	4	16
CONV	5	16
FC	16	8

CNN Structure		
Layer	Weights	Activations
CONV	Mixed	16
CONV	Mixed	16
FC	16	3

CNN Structure		
Layer	Weights	Activations
CONV	Mixed	8
CONV	Mixed	16
FC	8	3

Figure 4.7. Workload-aware quantization and pruning methodology (left). Running example (right).

layers have been processed, the flow further tries to reduce again one bit of layers' quantization (respecting the number of MAC operations priority) from which the flow hasn't previously backtracked. The iteration continues until no further bit-width reductions are possible in the BOs of any layer.

This phase is followed by filter-level optimization, which focuses on convolutional layers only. Many CNN filters do not use the entire value range when representing weights, especially after the aggressive quantization mentioned above. As illustrated in Figure 4.7-c, the flow hence drops, without loss of accuracy, the most significant bits (MSbs) on a per-filter base if allowed by weight ranges, correspondingly scaling convolution outputs. For example, if the range of values of the BOs in a filter is $\subset [-0.25, 0.25)$, 2 MSbs can be lowered and the outputs should be divided by $2^{Dropped\_bits} = 4$. In this stage, filters with all weights equal to zero are entirely deleted.

The last step of the optimization flow (Figure 4.7-d) performs the tailoring of the IMOs bit-widths. It takes advantage of the word-level parallelism supported by the compute memories (as described in Section 2.4). Similarly to the approach followed for BOs, the flow attempts to reduce the bit-width of IMOs on a per-layer basis. However, quantization steps are coarser in this case, as they must abide by the sub-word formats supported in hardware. In the implemented compute memories experiments, I admit 1×16-bit and 2×8bit sub-words, with the latter resulting in 2× reduction in execution

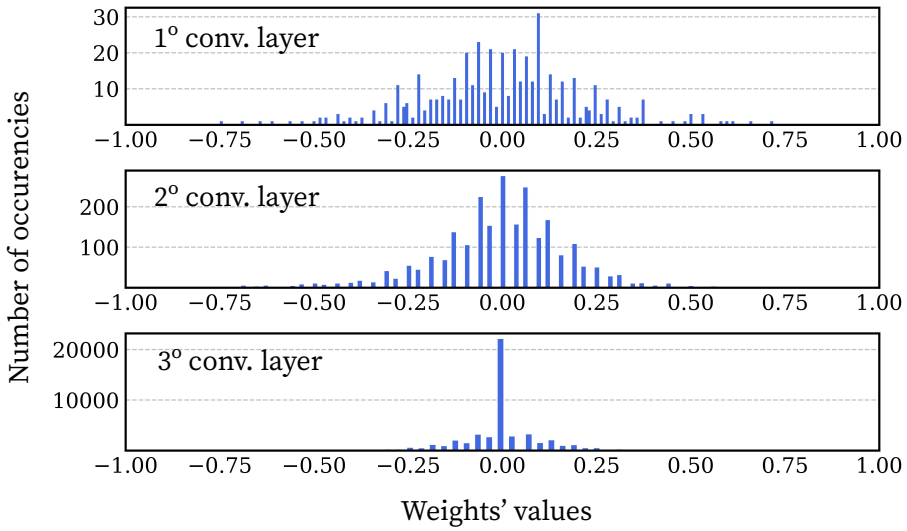


Figure 4.8. Weights distribution in the three convolutional layers of LeNet-5.

time[2].

4.5.2 Generic convolutional weights encoding

The GCW compression technique reduces the memory needed to represent quantized parameters in convolutional layers. This strategy enables efficient implementation of run-time decoding by leveraging the limited set of values that quantized weights can assume and their characteristic statistical distribution. It's important to note that the encoding procedure targets the broadcasted parameters common across multiple computations. Due to the offline nature of the encoding strategy, it cannot be applied to fully connected layers since their BOs (activations) vary for each processed input sample. Consequently, only the weights of convolutional layers are subject to encoding. Nonetheless, convolutional layers represent almost the totality of MAC operations in the benchmarked CNNs in Section 4.7.

Analyzing the weight distribution in the evaluated benchmarks clarifies why this approach is beneficial. For instance, in LeNet-5's three convolutional layers, many weights have a zero value, even after filter-level optimization has removed some of them. An example of this distribution can be seen in Figure 4.8. Furthermore, the figure shows a narrow distribution centered around zero. Smaller (quantized) weight values occur much more frequently than larger-magnitude values.

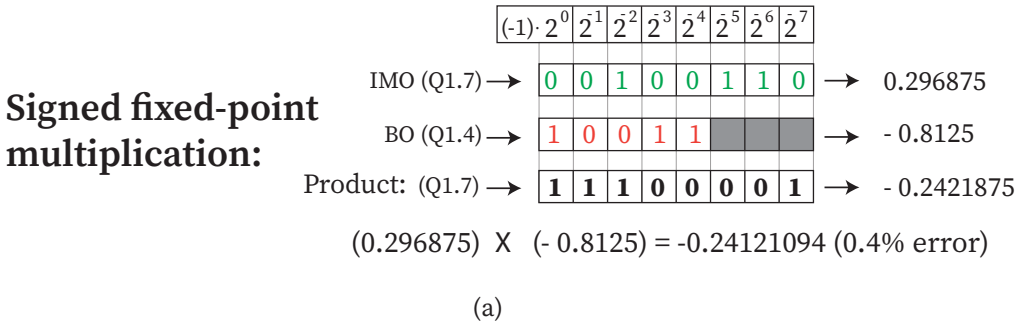
[2]While in principle the approach could be extended to 4 bits or 2 bits per word, such settings would incur in unacceptably large accuracy degradations.

Decimal Value (A)	GCW Coding	
	Code	Size
$\frac{[-9, 2^{(N-1)}]}{2^{(N-1)}}$	10000 & $\text{bin}_N(A)$	N+5
$\frac{[-1, -8]}{2^{(N-1)}}$	1 & $\text{bin}_4(A)$	5
0	0	1
$\frac{[1, 7]}{2^{(N-1)}}$	1 & $\text{bin}_4(A)$	5
$\frac{[8, 2^{(N-1)}-1]}{2^{(N-1)}}$	10000 & $\text{bin}_N(A)$	N+5

Figure 4.9. Generic Convolutional Weights coding scheme, where N indicates the quantization level values before coding. The code uses fewer bits to represent the most frequent symbols.

These findings lay the foundation for an encoding scheme that resembles Huffman coding, employing variable-length code words. Values with higher occurrences are encoded using low-bit-width representations, while less frequent values are mapped to higher bit-width codes. This encoding scheme optimally utilizes the limited memory resources by allocating fewer bits to commonly occurring values while ensuring an accurate representation of less frequent values.

The GCW encoding scheme is illustrated in Figure 4.9, with N being the number of quantized bits. Weights are assumed to be normalized in the $[-1,1)$ range. They are divided into five intervals, symmetric with respect to zero (as shown in the leftmost column). The zero employs the minimal bit-width (1 bit). Values close to zero are represented with 5-bit code words. A 1-bit prefix is appended to the two's complement 4-bit representation of the corresponding value. Other, seldom used, code words are derived by appending a fixed 5-bit prefix to their N-bit representation. For $N = 8$, 13 bits are required to represent large-magnitude values, but only 5 bits for low-magnitude ones. Note that, for $N < 5$, long code words are never generated, and the coding only generates different code-word lengths for zero and non-zero weights.



(a)

Compute Memory Instructions (NES = 3):	BO	Operations
Add(RSh(ACC), RSh(IMO)) → Wb(ACC)	LSb 1	+ 0 0 0 0 0 0 0 0 0 / 0 0 0 1 0 0 1 1 0
Add(RSh(ACC), RSh(IMO)) → Wb(ACC)	1	= 0 0 0 0 1 0 0 1 1 / + 0 0 0 1 0 0 1 1 0
Add(2x RSh(ACC), 2sComp(IMO)) → Wb(ACC)	001	= 0 0 0 0 0 1 1 1 0 0 0 / + 1 1 0 1 1 0 1 0
		Final product (Q1.7) : 1 1 1 0 0 0 0 1

(b)

Figure 4.10. (a) Two's complement fixed-point multiplication example between the IMO expressed in Q1.7 and the BO expressed in Q1.4. The compute memory instructions consider three simultaneous shifts.

4.6 Run-time CNN inference

This section presents the instructions required to perform MAC operations by the compute memories. and in the following, I cover how the pipeline circuit decompresses and generates these instructions.

4.6.1 Compute memory instructions

The considered compute memories are equipped with very constrained logic units. It performs additions, shifts, and negations in one word of 16-bit or two words of 8-bit. These operations are used in the core of MAC operations. The compute memories modulate the MAC cycle count based on the operands' bit-width, directly allowing heterogenous quantization strategy to translate into higher performance and energy efficiency. Moreover, I consider that compute memories can perform multiple shifts in a single clock cycle. In Section 2.3.3.1, I show that 3 embedded shifts accelerate, on average, 40% the multiplications. Hence, I consider this configuration for the compute memories.

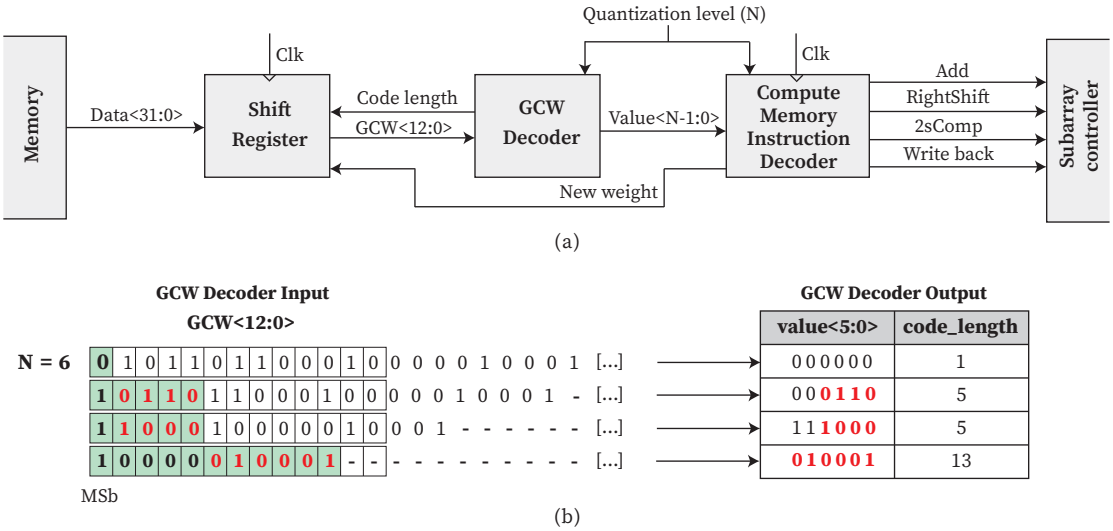


Figure 4.11. (a) Block diagram of the pipeline circuit to extract compute memory instructions from the compressed CNN model. (b) GCW decoding example with N = 6.

The instructions are carried out as an addition between the shifted value of the multiply accumulator with one of the following elements: zero, two's complement of IMO, or right shifted IMO. Figure 4.10 shows a typical multiplication performed by compute memories. With the IMO quantized in 8-bit word and BO in 5-bit word. The multiplication truncates the least significant bit in each step, so the product of multiplication has the same bit-width as the IMO. This slightly impacts the result of the operation, with the example showing only a 0.4% deviation from the expected value.

4.6.2 Real-time data decompression

According to the Generic Convolutional Weights (GCW) representation, convolutional weights are stored in an encoded form to reduce the size of CNN models, hence memory requirements. A pipeline circuit decodes them at run-time, deriving the corresponding compute memory instructions. Figure 4.11-a depicts the pipeline stages of the decoder. First, a shift register reads a memory word containing multiple GCW-encoded weights, possibly of different bit widths. Then, the GCW decoder decompresses the weights into their Q1.n representation. Finally, the compute memory instruction decoder converts these values into instructions broadcast to the subarrays.

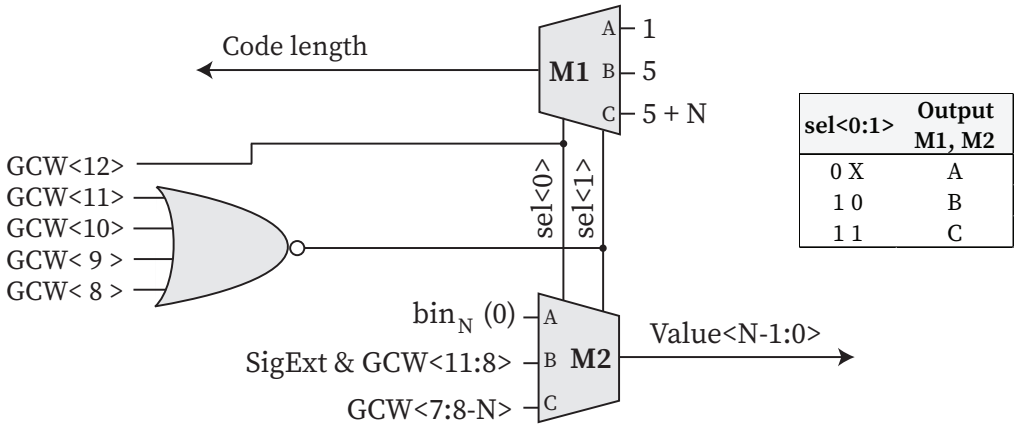


Figure 4.12. GCW decoder circuit-level design.

4.6.2.1 GCW shift register

For each convolutional filter, GCW-encoded weights form a bit-stream where each value is represented with 1 bit (for the "0" value), 5 bits (for values close to 0), or up to 13 bits (otherwise). Hence, the shift register, filled by 32-bit memory words at a time, usually holds multiple GCW code-words. When the decoder requires a new weight, it examines the first 13 bits of the shift register (`GCW<12:0>` in Figure 4.11-a), determining which bit-field (first bit, first 5 bits, or first 5+N bits, where N is the quantization level) contains the next code-word and advances the shift register according to the code length. The shift register is re-filled when less than 13 bits remain in its buffer, concatenating memory words. Code words can cross the boundary of two subsequent memory words, preventing under-utilization.

4.6.2.2 GCW decoder

The GCW decoder decodes the values of the weights according to their quantization level N, as illustrated in Figure 4.9. It analyzes `GCW<12:0>`, searching for specific bit sequences. The circuit-level design of the GCW is depicted in Figure 4.12. It comprises two multiplexers 3-to-1 (M1 and M2), which share the same selection signal (`sel<0:1>`). The signal `sel<0>` is directly connected to `GCW<12>`, while `sel<1>` is the output of a 4-input NOR gate (`GCW<11:8>`). When `sel<0>` = 0, the weight value is zero, encoded using a 1-bit code. Instead, when`sel<0>` = 1, `sel<1>` controls the output of the multiplexers. If the `GCW<11:8>` bits differ from “0000” they represent a small-magnitude weight encoded using a 5-bit code. The corresponding value is GCW<11:8>, sign-extended to N bits (`SigExt`). Finally, if `GCW<11:8>` = “0000”, the code-word represents a large-magnitude weight, whose value is encoded in the

GCW<7:8-N> bits using a variable code length ranging from 9 to 13.

Figure 4.11-b illustrates an example of run-time decoding considering $N = 6$. In the first one, GCW<12> = 0. Consequently, the weight value is zero, encoded using 1 bit. Thus, the shift register shifts one position, allowing the next weight to be decoded. Next, in the second example, GCW<12:8> = "10110". The GCW decoder extracts the last 4 bits and concatenates them with "00" to form a 6-bits binary value. The third example shows a similar case, but as the read value is negative, "11" is concatenated. The last example depicts the case where the GCW decoder finds the sequence "10000". The binary value is then retrieved in GCW<7:2>.

4.6.2.3 Compute memory instruction decoder

This block converts the weights' bits into compute memory instructions, defining the RightShift, the Add, and the 2sComp signals. It also governs the write-back signal to store the result of in-memory operations in the SRAM arrays. The sequence of shift-add operations that implement an MAC is entirely skipped when a "0" weight value is decoded, which results in energy and performance gains, as discussed in Section 4.7.1.

4.7 Experimental setup and results

The experimental results presented in this chapter were obtained through a collaborative effort with Flavio Ponzina, who proposed CNN optimization with heterogeneous quantization. My research focuses on CNN compression and mapping techniques. By combining our expertise, we explored and investigated different aspects of CNN optimization, resulting in a comprehensive analysis of the techniques employed.

Ponzina evaluated the heterogenous quantization framework on several edgeAI benchmarks of different complexity to demonstrate the effectiveness of the approach in a significant range of applications: LeNet-5 [93] on CIFAR-10 [89] and AlexNet [7], VGG16 [101], MobileNet [58] and Xception [94] on the CIFAR-100 dataset [89]. Accuracy values for various CNNs and optimization levels are retrieved using PyTorch [102] and the quantization functions described in [90].

CNNs are first trained using floating-point precision for 200 epochs, obtaining accuracies in line with the state-of-the-art. Similar to [41], models are then homogeneously quantized to 16-bit IMOs and 8-bit BOs, and refined for 20 additional training epochs. This configuration, which has no accuracy loss with respect to employing floating-point weights and activations, is assumed as the starting point for further

Table 4.3. Inferences-per-second in homogeneously quantized implementations executing on the BLADE architecture compared to heterogeneously quantized models executed on optimized compute memories architecture and employing 1, 32, or 128 subarrays. Optimized implementations are obtained for a 1% accuracy degradation threshold.

	Inferences Per Second (IPS)			
	BLADE	**Optimized (1% drop)**		
		1 sub	**32 subs**	**128 subs**
LeNet-5	289	1415	22491	22545
AlexNet	0.11	1.62	36	73
VGG16	0.39	2.89	78	201
MobileNet	0.35	3.76	93	220
Xception	0.0023	0.09	2.18	5.13

optimizations using the proposed methodology.

Repeated homogenous quantization of the BOs (down to 2 bits) and retraining were performed to establish a baseline. Then, models were retrained for five fine-tuning epochs at each step. Five epochs are also run when applying the heterogeneous approach, as described in Section 4.5, after each BOs and IMOs optimization step (phases (b) and (d) in Figure 4.7).

As for the electrical implementation, a subarray storing 5120 bits (640 bytes) was considered a test vehicle for our experiments. It stores 320 words in 1×16bit mode or 640 words in 2×8bit mode. Targeting a 28nm CMOS technology from TSMC, the compute memory can operate at a maximum frequency of 2.2GHz. Otherwise, The GCW decoder is designed as a semi-custom IC. It is synthesized, placed, and routed (again, in 28nm CMOS technology from TSMC) to extract its area, timing, and energy requirements. The circuit has an area of $760\mu m^2$, which represents 61% of the area of a single subarray and less than 1% of the total area in a 128-subarray configuration.

To highlight the gains of the proposed methods, I compare them with BLADE [31]. This architecture was the baseline for most of the contributions in my thesis. Moreover, such architecture has been shown to achieve 3× better performance and 1.5× increased energy efficiency compared with the ARM NEON SIMD accelerator when running inferences on benchmark CNNs. Thus, all the reported gains in the following sections can be extended to the SIMD accelerator.

4.7.1 Cycle count reduction on CNN inferences

Figure 4.13 illustrates the accuracy/performance trade-off in different optimized benchmarks. Black markers report the accuracy of homogeneously quantized models.

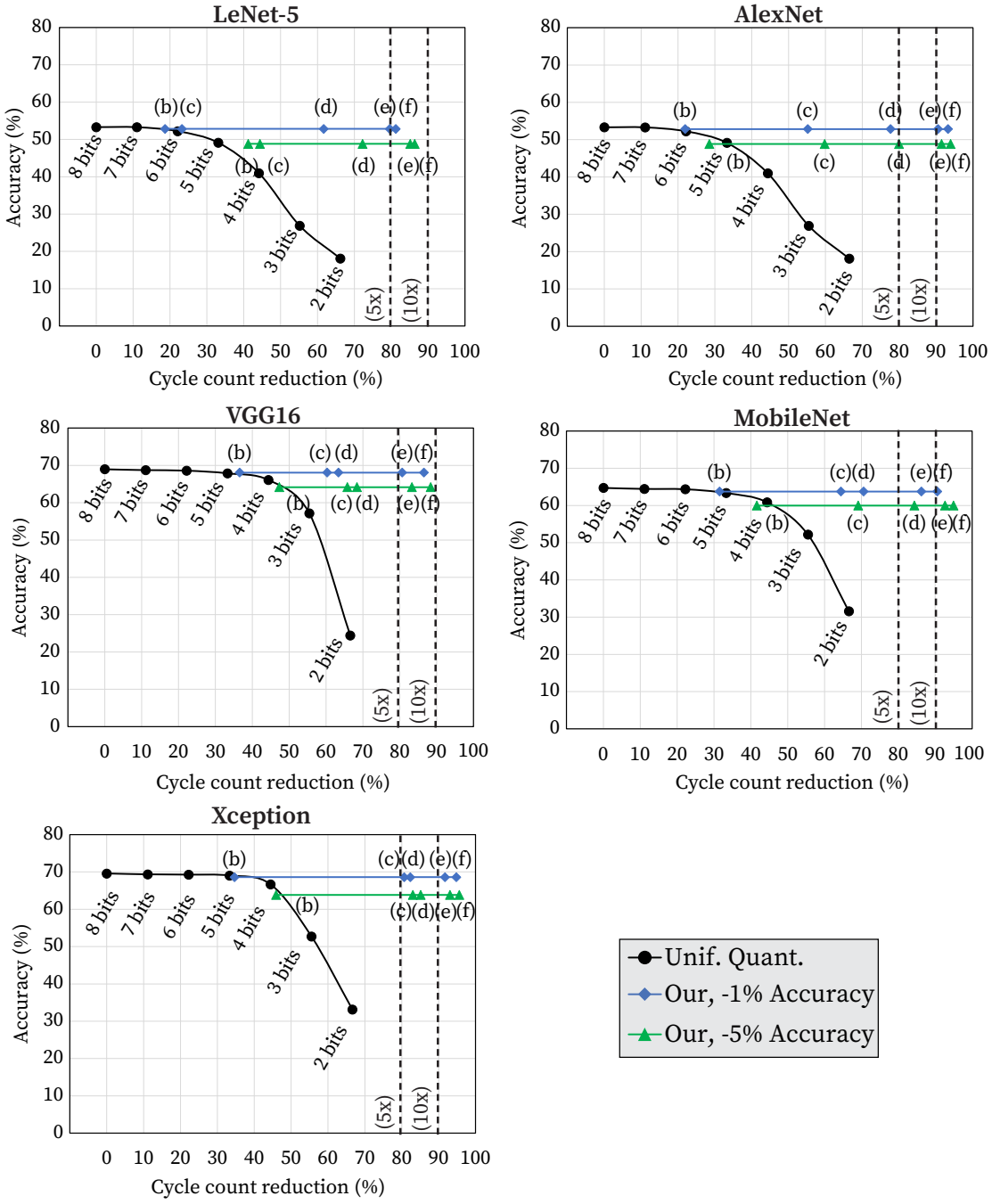


Figure 4.13. Accuracy and cycle count reductions in homogeneously quantized CNNs (black lines) and optimized CNNs (blue and green lines) for a 1% and a 5% accuracy drops. Data refers to single-subarray compute memory architectures. Vertical dashed lines mark speed-up levels of 5× and 10×.

At the same time, blue and green lines show the accuracy achieved in the various steps of the proposed hardware/software co-design methodology for accuracy degradation thresholds of 1% and 5%, respectively, compared to configurations with 8-bit BOs and 16-bit IMOs.

In Figure 4.13, the points (b), (c), and (d) highlight improvements obtained in the different stages of the optimization strategy, as illustrated in Figure 4.7 and detailed in Section 4.5. They report the performance/accuracy of CNNs after (b) BOs, (c) convolutional filters, and (d) IMOs optimization.

Points (e) and (f) report further cycle count reductions obtained by hardware optimizations. In (e), up to three single-cycle bit-shifts are supported. Additionally, in (f), MAC operations involving zero-valued broadcasted operands are skipped (Section 4.6.2).

When employing 5 bits, baseline uniform quantization achieves a 33% cycle count reduction at the cost of an average 1.3% accuracy degradation, which rapidly increases for smaller bit-widths. Conversely, significantly higher performance improvements are obtained with the approach. In particular, heterogeneous quantization alone (steps (b)-(d)) enables alone up to 80% cycle count gains. Compute memory hardware optimizations are also highly effective (steps (e) and (f)). Support three embedded shifts results in an average cycle count reduction of 2.1×, which increases to 2.9× when also skipping multiplications involving zero-valued broadcasted operands. Considering all software and hardware optimizations, the co-design framework achieves an average cycle count reduction of 89.3% (an average speed-up of 11.5×, , with its maximum in Xception, reaching 20×) for 8-bit quantized baselines for 1% degradation thresholds and 91.9% average cycle count reduction (a speed-up of 15×) for 5% accuracy degradation.

Table 4.3 shows the Inferences Per Second (IPS) in 8-bit quantized CNN baselines (BLADE with one subarray) and in optimized models. The IPS of the evaluated benchmarks scales up with the number of subarrays, as the workload is effectively distributed using the CNN mapping strategy proposed in Section 4.4. For AlexNet, VGG16, MobileNet, and Xception, on average, a speed-up of 58× is reached when employing 128 subarrays with respect to a single one. Being a smaller network, LeNet-5 is less amenable to parallelization, reaching in this setting a 15× speed-up.

4.7.2 Accuracy-constrained compression

Figure 4.14 depicts the average bit-width (across layers) of IMOs and BOs in CNN benchmark applications optimized with the proposed methodology. In all cases, the

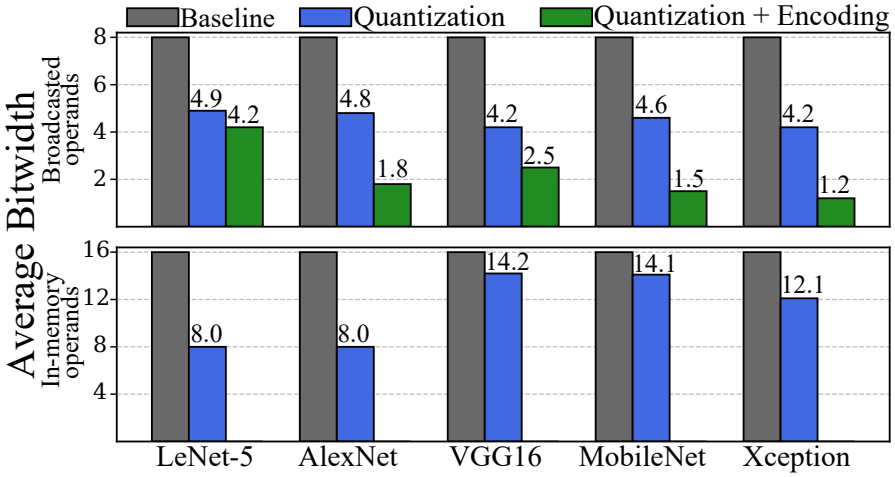


Figure 4.14. Average bit-widths achieved in the evaluated benchmarks by employing a synergic use of heterogeneous quantization (blue bars) and GCW encoding (green bars), for maximum accuracy degradation constraint of 1%.

results are for an accuracy threshold of 1% with respect to implementations having 16-bit IMOs and 8-bit BOs. Blue bars illustrate the average bit-width reduction achieved in convolutional and fully connected layers by mean of the heterogeneous quantization approach (as detailed in Section 4.5.1), resulting in compression ratios of CNN models of 76.8% on average. Then, additional savings are achieved by encoding the weights of convolutional layers (illustrated in Section 4.5.2). Results are shown as green bars in Figure 4.14. They show that GCW encoding effectively reduces storage requirements, resulting in average overall model size savings of 85.3%.

Experimental outcomes show that all activations of convolutional layers of simpler CNNs (LeNet-5 and AlexNet) can be effectively reduced to 8 bits while abiding by the accuracy constraint. Such optimization can only be selectively applied in more complex benchmarks, such as VGG16, MobileNet, and Xception, highlighting the benefit of a heterogeneous approach. Moreover, especially high compression ratios are achieved for larger models because the footprint of convolutional weights largely determines their size.

In Table 4.4, a comprehensive evaluation of the GCW encoding technique is presented, showcasing the remarkable compression achieved across all layers of the XCeption architecture when quantized in 5-bit words. This quantization scheme encompasses approximately 55% of the total MAC operations in the CNN. The results demonstrate a significant reduction in memory requirements, as the memory footprint is reduced from 20.7Mb when using 8-bit fixed-point representation to a mere 4.6Mb with the application of GCW encoding in the layers employing heterogeneous quantization. This substantial decrease in memory usage highlights the effectiveness of the GCW

Table 4.4. Compression of XCeption layers quantized in 5-bit words (1% accuracy drop).

Decimal	Signed Fixed-point (FP)	GCW	Appearances	Memory footprint (bits)		
				FP 8-bit	FP 5-bit	GCW
-1.0000	10000	1000010000	1	8.0E+0	5.0E+0	1.0E+1
-0.9375	10001	1000010001	0	0.0E+0	0.0E+0	0.0E+0
-0.8750	10010	1000010010	1	8.0E+0	5.0E+0	1.0E+1
-0.8125	10011	1000010011	1	8.0E+0	5.0E+0	1.0E+1
-0.7500	10100	1000010100	4	3.2E+1	2.0E+1	4.0E+1
-0.6875	10101	1000010101	5	4.0E+1	2.5E+1	5.0E+1
-0.6250	10110	1000010110	12	9.6E+1	6.0E+1	1.2E+2
-0.5625	10111	1000010111	21	1.7E+2	1.1E+2	2.1E+2
-0.5000	11000	11000	60	4.8E+2	3.0E+2	3.0E+2
-0.4375	11001	11001	102	8.2E+2	5.1E+2	5.1E+2
-0.3750	11010	11010	370	3.0E+3	1.9E+3	1.9E+3
-0.3125	11011	11011	958	7.7E+3	4.8E+3	4.8E+3
-0.2500	11100	11100	4614	3.7E+4	2.3E+4	2.3E+4
-0.1875	11101	11101	11959	9.6E+4	6.0E+4	6.0E+4
-0.1250	11110	11110	61210	4.9E+5	3.1E+5	3.1E+5
-0.0625	11111	11111	174433	1.4E+6	8.7E+5	8.7E+5
0.0000	00000	0	2095312	1.7E+7	1.0E+7	2.1E+6
0.0625	00001	10001	152846	1.2E+6	7.6E+5	7.6E+5
0.1250	00010	10010	65962	5.3E+5	3.3E+5	3.3E+5
0.1875	00011	10011	16337	1.3E+5	8.2E+4	8.2E+4
0.2500	00100	10100	6557	5.2E+4	3.3E+4	3.3E+4
0.3125	00101	10101	1481	1.2E+4	7.4E+3	7.4E+3
0.3750	00110	10110	537	4.3E+3	2.7E+3	2.7E+3
0.4375	00111	10111	145	1.2E+3	7.3E+2	7.3E+2
0.5000	01000	1000001000	91	7.3E+2	4.6E+2	9.1E+2
0.5625	01001	1000001001	28	2.2E+2	1.4E+2	2.8E+2
0.6250	01010	1000001010	16	1.3E+2	8.0E+1	1.6E+2
0.6875	01011	1000001011	5	4.0E+1	2.5E+1	5.0E+1
0.7500	01100	1000001100	5	4.0E+1	2.5E+1	5.0E+1
0.8125	01101	1000001101	0	0.0E+0	0.0E+0	0.0E+0
0.8750	01110	1000001110	1	8.0E+0	5.0E+0	1.0E+1
0.9375	01111	1000001111	0	0.0E+0	0.0E+0	0.0E+0
			Total	**20.7 Mbits**	**13.0 Mbits**	**4.59 Mbits**

encoding approach in achieving efficient compression without compromising the accuracy and performance of the CNN model. The findings presented in Table 4.4 underscore the potential of GCW encoding as a valuable tool in optimizing CNN architectures, enabling resource-constrained systems to leverage the benefits of neural network computations while minimizing memory utilization.

4.8 Conclusion

AI can be enabled on the edge, but this requires optimizing both software and hardware due to limited computing resources. To address this challenge, a comprehensive framework has been introduced in this chapter for optimizing and executing CNNs within limited memory. The framework includes a compression strategy that combines heterogeneous quantization and GCW encoding techniques. By leveraging the unique characteristics of each layer, this strategy achieves significant memory reductions compared to uniformly quantized CNN implementations. The approach has been tested on various CNN benchmarks, and empirical results show an average memory reduction of 85% while maintaining a 1% accuracy degradation constraint.

The framework's execution phase considers optimized compute memories. Unlike other approaches, inference speed-ups have been achieved by integrating compression strategy and intelligent memory utilization. The compute memories achieved up to 20× improvement in inference speed while maintaining the same 1% accuracy degradation constraint. Opportunities for enhanced AI capabilities at the edge have been opened by combining compression and execution optimization strategies through our framework. The significant reduction in memory usage and speed improvements from our software and hardware approach offers great potential to deploy AI applications on resource-constrained edge platforms.

5 Conclusion and future work

This concluding chapter of the thesis summarizes the key contributions of the research on enabling edge AI through co-optimization with bit-line computing (BC) architectures. Furthermore, valuable insights into future research directions are presented, built on the findings and results obtained.

5.1 Summary

State-of-the-art BC architectures have demonstrated impressive results on edge AI workloads. For example, BLADE has outperformed the NEON [33] SIMD accelerator from ARM by up to 6× in speed. However, these architectures still present a high potential for improvements since they suffer from slow multiplications and potential data overflow. Therefore, in Chapter 2, I presented two novel BC-based architectures to enable edge AI workloads by addressing such limitations. Firstly, the associativity-agnostic (AA-BC) architecture [40] presents an innovative memory organization that increases the flexibility of operands placement and simplifies the circuit periphery, allowing operations to be extracted in-situ. The AA-BC architecture increased multiplication efficiency by 44% compared to BLADE by incorporating four embedded shifts. Additionally, I introduced signed two's complement fixed-point arithmetic and operand scaling to address the issue of data overflow. This improved the reliability of the multiplication operations. The second presented architecture addresses the multiply-accumulate (MAC) overflows. The MAC-BC architecture presents the most optimized periphery, and thanks to its enhanced register management, the multiplication operation no longer requires write-back cycles and constant memory access, resulting in a boost of up to 4× energy efficiency compared to AA-BC. Furthermore, strategies for parallelism at the subarray and array level were explored, leading to remarkable reductions in the CNN inference cycle count.

Moving to Chapter 3, the focus shifted to energy-efficient computing for AI at the edge. Firstly, the hybrid BC architecture combines SRAM and RRAM technologies and showcases significant improvements in power performance. A data mapping strategy that matches data behavior with specific characteristics of memory technology allowed the reduction of data transfer by up to 60× for popular CNN models such as AlexNet and MobileNet. This resulted in energy efficiency gains of up to 93% and performance improvements of up to 6×. Additionally, focusing on reducing the voltage to improve the system's efficiency, I explored the effects on the compatibility of SRAM and RRAM at ultra-low voltage. By increasing the width of access transistors on a 1T1R configuration and overdriving word-lines, near-threshold voltage levels could be achieved during RRAM read operations, minimizing the impact of CMOS variability and reducing bit error rates. These techniques enabled a lower high resistance state/low resistance state (HRS/LRS) memory ratio, leading to 2-3× energy reductions in programming operations. Furthermore, I also proposed a resilient BC-based architecture that embedded an error detection and mitigation strategy. This architecture allows for the reduction of voltage from the nominal 800 mV to just 650 mV by implementing a parity-bit checker and calculator based on bit-line computing at the periphery of the memory. The resilient architecture demonstrated energy savings of up to 51.3% without affecting the initial CNN accuracy.

Finally, in Chapter 4, I introduced a comprehensive framework for optimizing, compressing, and executing CNNs with limited memory on edge platforms. The compression strategy achieved remarkable memory reductions by combining heterogeneous quantization and lossless encoding. Empirical results on various CNN benchmarks demonstrated an average memory reduction of 85% while maintaining a 1% accuracy degradation constraint. In terms of execution optimization, the framework achieved up to 20× inference speed-up compared to other approaches while adhering to the same 1% accuracy degradation constraint. These findings highlight the potential of compression and execution optimization strategies in enabling AI applications on resource-constrained edge platforms.

These findings have important implications for AI at the edge, providing insights into designing efficient hardware and software solutions. The substantial reductions in memory usage, the significant speed improvements, and the energy efficiency gains showcased throughout the thesis offer promising avenues for deploying AI applications on edge platforms with limited computing resources.

Figure 5.1. Layout of the implemented subarray on 65nm CMOS technology from TSMC.

5.2 Future work

This thesis opens novel research avenues and the future work can be divided into three main areas. Firstly, validating the proposed architectures in silicon is crucial to ensure their practical implementation. Then, one important aspect is to provide full support for CNN inferences without needing data to hop between the accelerator and the CPU. Developing the required circuitry for a fully independent accelerator would pave the way to enable edgeAI learning directly in the memory. This breakthrough would enhance the efficiency of edgeAI systems and create opportunities for adopting distributed learning approaches, including federated learning. Another direction is to explore the applicability of the proposed methods in other contexts, beyond CNNs, by investigating their effectiveness in different types of neural networks. These future endeavors aim to enhance edgeAI capabilities by expanding CNN support, exploring broader neural network contexts, and ensuring the feasibility and efficacy of the proposed architectures through rigorous silicon validation.

5.2.1 Silicon validation

During my Ph.D., our research group integrated a bit-line computing architecture into three chips. The implementations were executed in 65nm TSMC CMOS technology and included the associativity-agnostic bit-line computing architecture, Figure 5.1 shows the subarray design. This was done in collaboration with the Integrated Systems Laboratory (IIS) at the Swiss Federal Institute of Technology in Zürich (ETHZ) for the Rosetta [103] and Darkside [104] chips. These chips featured 16 subarrays of 2KB each, along with a RI5CY [105] processor.

However, these circuits could not validate the bit-line computing principle with local

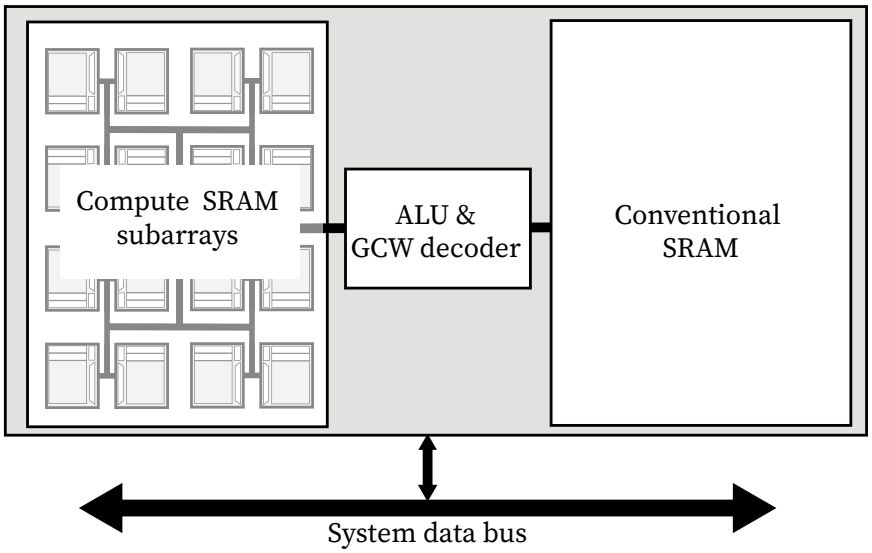


Figure 5.2. Bit-line computing architectural extension to fully support CNN inferences and distributed edge learning.

groups due to various challenges. The Rosetta chip encountered issues due to the short deadline and the transposition of the design from 28nm to 65nm, resulting in faulty operation. The Darkside chip faced problems in integrating the digital controller described in RTL with the in-memory compute instance created using the full custom design methodology. We rectified the circuit and resubmitted it in the HEEPocrates chip, incorporating the knowledge gained from the previous tape-outs.

The HEEPocrates chip was executed entirely in the ESL laboratory and is based on X-HEEP [106], a configurable and extendable single core RISC-V-based ultra-low-power microcontroller. In this chip, we integrated a single subarray with the system's bus and another stand-alone subarray controlled through a shift register and accessed directly from the chip's pins. Once HEEPocrates returns from the foundry, it will be possible to characterize its performance using the integrated subarray and measure energy consumption with the stand-alone subarray. Additionally, once the concept is proven, the next tape-out would include the MAC-BC architecture, enabling the deployment of real CNNs for evaluation and further advancement.

5.2.2 Architecture exploration

The current approach of scaling CNNs' activations on the MAC-BC architecture (Section 2.4) necessitates reading all the outputs, determining the maximum and minimum values, and performing a division of all the activations to complete the lay-

ers scaling. In our experiments with the MAC-BC architecture, we implemented these operations using 32-bit floating-point precision, assuming that the activations are read from memory in each layer. The minimum and maximum values are determined during this process, and the data is then sent back to the compute memories after the division by the CPU.

An architectural extension, as depicted in Figure 5.2, could optimize this process. This extension involves connecting the compute memories to a conventional SRAM array, with a dedicated logic unit in between that would be dedicated to scaling. Additionally, the pipeline circuit presented in Section 4.6.2 for decompressing convolutional weights would also be integrated between the two types of SRAMs. Several aspects need to be explored, such as the number of compute subarrays, the size of the conventional SRAM array, and the required features for the intermediate ALU.

Moreover, incorporating the necessary circuitry within the memory opens up possibilities for performing learning tasks directly within it. With the ability to perform floating-point division, differential equations required for gradient calculation and backpropagation can also be computed. Consequently, this architecture would potentially enable more efficient distributed edge learning and facilitates the application of federated learning in scenarios where privacy and security are essential.

5.2.3 Exploration of applications

In addition to its application in CNNs, innovations in bit-line computing have great potential for other neural network architectures. Exploring the extension of bit-line computing to Recurrent Neural Networks (RNNs) can enhance their performance and energy efficiency. RNNs are commonly used in sequential data analysis tasks, such as natural language processing and speech recognition. By leveraging repetitive computations and data dependencies in RNNs, bit-line computing architectures can accelerate these computations and improve overall efficiency. Another promising avenue for exploration is the integration of bit-line computing with Graph Neural Networks (GNNs). GNNs have gained significant attention for their ability to model complex relationships in graph-structured data. Accelerated graph-based computations could be enabled by adopting the MAC-BC architecture to GNNs, which depend heavily on MAC operations.

Comprehensive system support is essential to ensure widespread adoption and easy programming for bit-line computing architectures. This involves developing user-friendly tools and libraries that abstract complexities and provide high-level APIs tailored to bit-line computing. Moreover, compiler enhancements could enable auto-

matic optimization and efficient code mapping, maximizing performance and energy efficiency. A first step in this direction would be the integration of CNN2BLADE, the cycle-accurate simulator with all the methods to map CNN into the proposed architectures in ONNX (Open Neural Network Exchange). ONNX is an open format that facilitates interoperability between different deep learning frameworks. ONNX defines a common representation of neural network models, including their structure, parameters, and computations. ONNX has gained popularity and is supported by several major deep-learning frameworks, including PyTorch and TensorFlow. Thus, these architectures can be adopted and utilized across various domains and applications by expanding the application scope of bit-line computing innovations and providing robust system support.

5.3 List of publications

Patents:

- **M Rios**, WA Simon, AS Levisse, M Zapater, DA Alonso, “Associativity-agnostic in-cache computing memory architecture optimized for multiplication” US Patent 11,211,115 (2021) [107].
- WA Simon, **M Rios**, AS Levisse, M Zapater, DA Alonso, “Memory chip or memory array for wide-voltage range in-memory computing using bitline technology” US Patent 11,094,355 (2021) [108].

Journal publications:

- F Ponzina, **M Rios**, A Levisse, G Ansaloni, D Atienza, "Overflow-free compute memories for edge ai acceleration" The International Conference on Hardware/-Software Codesign and System Synthesis (CODES+ISSS) (2023) [42] - Journal submitted.
- **M Rios**, F Ponzina, A Levisse, G Ansaloni, D Atienza, “Bit-Line Computing for CNN Accelerators Co-Design in Edge AI Inference” IEEE Transactions on Emerging Topics in Computing (2023) [34].
- F Ponzina, S Machetti, **M Rios**, BW Denkinger, A Levisse, G Ansaloni, D Atienza, “A hardware/software co-design vision for deep learning at the edge“ IEEE Micro 42 (6), 48-54 (2022) [109].

- WA Simon, YM Qureshi, **M Rios**, A Levisse, M Zapater, D Atienza, "BLADE: An in-cache computing architecture for edge devices" IEEE Transactions on Computers 69 (9), 1349-1363 (2019) [31].

Conference publications:

- **M Rios**, F Ponzina, G Ansaloni, A Levisse, D Atienza "Error resilient in-memory computing architecture for CNN inference on the edge" Proceedings of the Great Lakes Symposium on VLSI 2022 [60].
- F Ponzina, **M Rios**, G Ansaloni, A Levisse, D Atienza, "A flexible in-memory computing architecture for heterogeneously quantized CNNs" IEEE Computer Society Annual Symposium on VLSI (ISVLSI), 2021 [41].
- **M Rios**, F Ponzina, G Ansaloni, A Levisse, D Atienza, "Running efficiently CNNs on the edge thanks to hybrid SRAM-RRAM in-memory computing" Design, Automation & Test in Europe Conference & Exhibition (DATE), 2021 [41].
- A Levisse, **M Rios**, WA Simon, PE Gaillardon, D Atienza "Functionality enhanced memories for edge-AI embedded systems" 19th Non-Volatile Memory Technology Symposium (NVMTS), 2019 [110].
- **M Rios**, W Simon, A Levisse, M Zapater, D Atienza "An associativity-agnostic in-cache computing architecture optimized for multiplication" IEEE 27th International Conference on Very Large Scale Integration (VLSI-SoC), 2019 [40].

Bibliography

[1] M. S. Murshed, C. Murphy, D. Hou, N. Khan, G. Ananthanarayanan, and F. Hussain, "Machine learning at the network edge: A survey," *ACM Computing Surveys (CSUR)*, vol. 54, no. 8, pp. 1–37, 2021.

[2] A. Sebastian, M. Le Gallo, R. Khaddam-Aljameh, and E. Eleftheriou, "Memory devices and applications for in-memory computing," *Nature nanotechnology*, vol. 15, no. 7, pp. 529–544, 2020.

[3] T. Periyaswamy and M. Balasubramanian, "Ambulatory cardiac bio-signals: From mirage to clinical reality through a decade of progress," *International journal of medical informatics*, vol. 130, p. 103928, 2019.

[4] A. Shaout, D. Colella, and S. Awad, "Advanced driver assistance systems - past, present and future," in *2011 Seventh International Computer Engineering Conference (ICENCO'2011)*, 2011, pp. 72–82.

[5] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges," *IEEE Internet of Things Journal*, vol. 3, no. 5, pp. 637–646, 2016.

[6] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[7] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Communications of the ACM*, vol. 60, no. 6, pp. 84–90, 2017.

[8] H. Salehinejad, S. Sankar, J. Barfett, E. Colak, and S. Valaee, "Recent advances in recurrent neural networks," *arXiv preprint arXiv:1801.01078*, 2017.

[9] J. Zhou, G. Cui, S. Hu, Z. Zhang, C. Yang, Z. Liu, L. Wang, C. Li, and M. Sun, "Graph neural networks: A review of methods and applications," *AI open*, vol. 1, pp. 57–81, 2020.

[10] S. Mittal and S. Vaishay, "A survey of techniques for optimizing deep learning on gpus," *Journal of Systems Architecture*, vol. 99, p. 101635, 2019.

[11] E. Georganas, S. Avancha, K. Banerjee, D. Kalamkar, G. Henry, H. Pabst, and A. Heinecke, "Anatomy of high-performance deep learning convolutions on simd architectures," in *SC18: International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 2018, pp. 830–841.

[12] S. A. McKee, "Reflections on the memory wall," in *Proceedings of the 1st Conference on Computing Frontiers*, ser. CF '04. New York, NY, USA: Association for Computing Machinery, 2004, p. 162. [Online]. Available: https://doi.org/10.1145/977091.977115

[13] X. Huang, C. Liu, Y.-G. Jiang, and P. Zhou, "In-memory computing to break the memory wall," *Chinese Physics B*, vol. 29, no. 7, p. 078504, 2020.

[14] S. Bavikadi, P. R. Sutradhar, K. N. Khasawneh, A. Ganguly, and S. M. Pudukotai Dinakarrao, "A review of in-memory computing architectures for machine learning applications," in *Proceedings of the 2020 on Great Lakes Symposium on VLSI*, 2020, pp. 89–94.

[15] H. Wu, X. H. Wang, B. Gao, N. Deng, Z. Lu, B. Haukness, G. Bronner, and H. Qian, "Resistive random access memory for future information processing system," *Proceedings of the IEEE*, vol. 105, no. 9, pp. 1770–1789, 2017.

[16] D. Ielmini, "Resistive switching memories based on metal oxides: mechanisms, reliability and scaling," *Semiconductor Science and Technology*, vol. 31, no. 6, p. 063002, 2016.

[17] S. Raoux, W. Wełnic, and D. Ielmini, "Phase change materials and their application to nonvolatile memories," *Chemical reviews*, vol. 110, no. 1, pp. 240–267, 2010.

[18] C. Chappert, A. Fert, and F. N. Van Dau, "The emergence of spin electronics in data storage," *Nature materials*, vol. 6, no. 11, pp. 813–823, 2007.

[19] A. Levisse, B. Giraud, J. Noel, M. Moreau, J. Portal *et al.*, "Architecture, design and technology guidelines for crosspoint memories," in *2017 IEEE/ACM International Symposium on Nanoscale Architectures (NANOARCH)*. IEEE, 2017, pp. 55–60.

[20] Q. Dong, Z. Wang, J. Lim, Y. Zhang, Y.-C. Shih, Y.-D. Chih, J. Chang, D. Blaauw, and D. Sylvester, "A 1mb 28nm stt-mram with 2.8 ns read access time at 1.2

v vdd using single-cap offset-cancelled sense amplifier and in-situ self-write-termination," in *2018 IEEE International Solid-State Circuits Conference-(ISSCC)*. IEEE, 2018, pp. 480–482.

[21] A. Grossi, E. Vianello, C. Zambelli, P. Royer, J.-P. Noel, B. Giraud, L. Perniola, P. Olivo, and E. Nowak, "Experimental investigation of 4-kb rram arrays programming conditions suitable for tcam," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 26, no. 12, pp. 2599–2607, 2018.

[22] A. Haj-Ali, R. Ben-Hur, N. Wald, and S. Kvatinsky, "Efficient algorithms for in-memory fixed point multiplication using magic," in *2018 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 2018, pp. 1–5.

[23] S. Mittal, G. Verma, B. Kaushik, and F. A. Khanday, "A survey of sram-based in-memory computing techniques and applications," *Journal of Systems Architecture*, vol. 119, p. 102276, 2021.

[24] A. Biswas and A. P. Chandrakasan, "Conv-ram: An energy-efficient sram with embedded convolution computation for low-power cnn-based machine learning applications," in *2018 IEEE International Solid - State Circuits Conference - (ISSCC)*, 2018, pp. 488–490.

[25] M. Ali, A. Jaiswal, S. Kodge, A. Agrawal, I. Chakraborty, and K. Roy, "Imac: In-memory multi-bit multiplication and accumulation in 6t sram array," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 67, no. 8, pp. 2521–2531, 2020.

[26] S. Yin, Z. Jiang, M. Kim, T. Gupta, M. Seok, and J.-S. Seo, "Vesti: Energy-efficient in-memory computing accelerator for deep neural networks," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 28, no. 1, pp. 48–61, 2019.

[27] S. Jeloka *et al.*, "A 28 nm configurable memory (TCAM/BCAM/SRAM) using push-rule 6t bit cell enabling logic-in-memory," *IEEE JSSC*, 2016.

[28] K. C. Akyel, H.-P. Charles, J. Mottin, B. Giraud, G. Suraci, S. Thuries, and J.-P. Noel, "Drc 2: Dynamically reconfigurable computing circuit based on memory architecture," in *2016 IEEE International Conference on Rebooting Computing (ICRC)*. IEEE, 2016, pp. 1–8.

[29] C. Eckert, X. Wang, J. Wang, A. Subramaniyan, R. Iyer, D. Sylvester, D. Blaaauw, and R. Das, "Neural cache: Bit-serial in-cache acceleration of deep neural networks," in *2018 ACM/IEEE 45Th annual international symposium on computer architecture (ISCA)*. IEEE, 2018, pp. 383–396.

[30] J. Wang, X. Wang, C. Eckert, A. Subramaniyan, R. Das, D. Blaauw, and D. Sylvester, "A 28-nm compute sram with bit-serial logic/arithmetic operations for programmable in-memory vector computing," *IEEE Journal of Solid-State Circuits*, vol. 55, no. 1, pp. 76–86, 2019.

[31] W. A. Simon, Y. M. Qureshi, M. Rios, A. Levisse, M. Zapater, and D. Atienza, "Blade: An in-cache computing architecture for edge devices," *IEEE Transactions on Computers*, vol. 69, no. 9, pp. 1349–1363, 2020.

[32] W. Simon, J. Galicia, A. Levisse, M. Zapater, and D. Atienza, "A fast, reliable and wide-voltage-range in-memory computing architecture," in *Proceedings of the 56th Annual Design Automation Conference 2019*, 2019, pp. 1–6.

[33] T. R. ARM, "Arm. introducing neon development article." https://developer.arm.com/documentation/dht0002/a/Introducing-NEON, accessed 09-06-2023.

[34] M. Rios, F. Ponzina, A. Levisse, G. Ansaloni, and D. Atienza, "Bit-line computing for cnn accelerators co-design in edge ai inference," *IEEE Transactions on Emerging Topics in Computing*, 2023.

[35] D. Fujiki, S. Mahlke, and R. Das, "Duality cache for data parallel acceleration," in *Proceedings of the 46th International Symposium on Computer Architecture*, 2019, pp. 397–410.

[36] F. Ponzina, M. Peon-Quiros, A. Burg, and D. Atienza, "E2cnns: Ensembles of convolutional neural networks to improve robustness against memory errors in edge-computing devices," *IEEE Transactions on Computers*, vol. 70, no. 8, pp. 1199–1212, 2021.

[37] T. Liang et al., "Pruning and quantization for deep neural network acceleration: A survey," 2021.

[38] J. Wu, C. Leng, Y. Wang, Q. Hu, and J. Cheng, "Quantized convolutional neural networks for mobile devices," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016, pp. 4820–4828.

[39] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," *arXiv preprint arXiv:1510.00149*, 2015.

[40] M. Rios, W. Simon, A. Levisse, M. Zapater, and D. Atienza, "An associativity-agnostic in-cache computing architecture optimized for multiplication," in *2019 IFIP/IEEE 27th International Conference on Very Large Scale Integration (VLSI-SoC)*. IEEE, 2019, pp. 34–39.

[41] F. Ponzina, M. Rios, G. Ansaloni, A. Levisse, and D. Atienza, "A flexible in-memory computing architecture for heterogeneously quantized cnns," in *2021 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*. IEEE, 2021, pp. 164–169.

[42] ——, "Overflow-free compute memories for edge ai acceleration," in *The International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, 2023.

[43] A. Reuther, P. Michaleas, M. Jones, V. Gadepally, S. Samsi, and J. Kepner, "Ai accelerator survey and trends," in *2021 IEEE High Performance Extreme Computing Conference (HPEC)*. IEEE, 2021, pp. 1–9.

[44] S. P. Baller, A. Jindal, M. Chadha, and M. Gerndt, "Deepedgebench: Benchmarking deep neural networks on edge devices," in *2021 IEEE International Conference on Cloud Engineering (IC2E)*. IEEE, 2021, pp. 20–30.

[45] S. R. Sukumar, J. A. Balma, C. Xu, and S. Serebryakov, "Survival of the fittest amidst the cambrian explosion of processor architectures for artificial intelligence," in *2021 IEEE/ACM Programming Environments for Heterogeneous Computing (PEHC)*. IEEE, 2021, pp. 34–43.

[46] Y.-C. Kwon, S. H. Lee, J. Lee, S.-H. Kwon, J. M. Ryu, J.-P. Son, O. Seongil, H.-S. Yu, H. Lee, S. Y. Kim *et al.*, "25.4 a 20nm 6gb function-in-memory dram, based on hbm2 with a 1.2 tflops programmable computing unit using bank-level parallelism, for machine learning applications," in *2021 IEEE International Solid-State Circuits Conference (ISSCC)*, vol. 64. IEEE, 2021, pp. 350–352.

[47] A. K. Ramanathan, G. S. Kalsi, S. Srinivasa, T. M. Chandran, K. R. Pillai, O. J. Omer, V. Narayanan, and S. Subramoney, "Look-up table based energy efficient processing in cache support for neural network acceleration," in *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2020, pp. 88–101.

[48] Y. Zhang, L. Xu, Q. Dong, J. Wang, D. Blaauw, and D. Sylvester, "Recryptor: A reconfigurable cryptographic cortex-m0 processor with in-memory and near-memory computing for iot security," *IEEE Journal of Solid-State Circuits*, vol. 53, no. 4, pp. 995–1005, 2018.

[49] Z. Zhang, J.-J. Chen, X. Si, Y.-N. Tu, J.-W. Su, W.-H. Huang, J.-H. Wang, W.-C. Wei, Y.-C. Chiu, J.-M. Hong *et al.*, "A 55nm 1-to-8 bit configurable 6t sram based computing-in-memory unit-macro for cnn-based ai edge processors," in *2019 IEEE Asian Solid-State Circuits Conference (A-SSCC)*. IEEE, 2019, pp. 217–218.

[50] M. Rios, F. Ponzina, G. Ansaloni, A. Levisse, and D. Atienza, "Running efficiently cnns on the edge thanks to hybrid sram-rram in-memory computing," in *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2021, pp. 1881–1886.

[51] Y. Qureshi et al., "Gem5-x: A gem5-based system level simulation framework to optimize many-core platforms," in *SpringSim*, 2019.

[52] M. J. Dworkin, "Sha-3 standard: Permutation-based hash and extendable-output functions," 2015.

[53] M. Viitanen, A. Koivula, A. Lemmetti, A. Ylä-Outinen, J. Vanne, and T. D. Hämäläinen, "Kvazaar: Open-source hevc/h. 265 encoder," in *Proceedings of the 24th ACM international conference on Multimedia*, 2016, pp. 1179–1182.

[54] D. Rossi, F. Conti, A. Marongiu, A. Pullini, I. Loi, M. Gautschi, G. Tagliavini, A. Capotondi, P. Flatresse, and L. Benini, "Pulp: A parallel ultra low power platform for next generation iot applications," in *2015 IEEE Hot Chips 27 Symposium (HCS)*. IEEE Computer Society, 2015, pp. 1–39.

[55] Y. Chen, Y. Xie, L. Song, F. Chen, and T. Tang, "A survey of accelerator architectures for deep neural networks," *Engineering*, vol. 6, no. 3, pp. 264–274, 2020.

[56] M. Bocquet, T. Hirztlin, J.-O. Klein, E. Nowak, E. Vianello, J.-M. Portal, and D. Querlioz, "In-memory and error-immune differential rram implementation of binarized deep neural networks," in *2018 IEEE International Electron Devices Meeting (IEDM)*. IEEE, 2018, pp. 20–6.

[57] E. Vianello, O. Thomas, M. Harrand, S. Onkaraiah, T. Cabout, B. Traoré, T. Diokh, H. Oucheikh, L. Perniola, G. Molas *et al.*, "Back-end 3d integration of hfo 2-based rrams for low-voltage advanced ic digital design," in *Proceedings of 2013 International Conference on IC Design & Technology (ICICDT)*. IEEE, 2013, pp. 235–238.

[58] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *arXiv preprint arXiv:1704.04861*, 2017.

[59] H. Sagha, H. Bayati, J. d. R. Millán, and R. Chavarriaga, "On-line anomaly detection and resilience in classifier ensembles," *Pattern Recognition Letters*, vol. 34, no. 15, pp. 1916–1927, 2013.

[60] M. Rios, F. Ponzina, G. Ansaloni, A. Levisse, and D. Atienza, "Running efficiently cnns on the edge thanks to hybrid sram-rram in-memory computing," in *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2021, pp. 1881–1886.

[61] A. Khan, A. Sohail, U. Zahoora, and A. S. Qureshi, "A survey of the recent architectures of deep convolutional neural networks," *Artificial intelligence review*, vol. 53, pp. 5455–5516, 2020.

[62] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[63] X. Xu, Y. Ding, S. X. Hu, M. Niemier, J. Cong, Y. Hu, and Y. Shi, "Scaling for edge inference of deep neural networks," *Nature Electronics*, vol. 1, no. 4, pp. 216–222, 2018.

[64] M. Courbariaux, I. Hubara, D. Soudry, R. El-Yaniv, and Y. Bengio, "Binarized neural networks: Training deep neural networks with weights and activations constrained to+ 1 or-1," *arXiv preprint arXiv:1602.02830*, 2016.

[65] S. Han, J. Pool, J. Tran, and W. Dally, "Learning both weights and connections for efficient neural network," *Advances in neural information processing systems*, vol. 28, 2015.

[66] Y. He, Y. Ding, P. Liu, L. Zhu, H. Zhang, and Y. Yang, "Learning filter pruning criteria for deep convolutional neural networks acceleration," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2020, pp. 2009–2018.

[67] M. Amin-Naji, A. Aghagolzadeh, and M. Ezoji, "Ensemble of cnn for multi-focus image fusion," *Information fusion*, vol. 51, pp. 201–214, 2019.

[68] H. Esmaeilzadeh, A. Sampson, L. Ceze, and D. Burger, "Neural acceleration for general-purpose approximate programs," in *2012 45th annual IEEE/ACM international symposium on microarchitecture*. IEEE, 2012, pp. 449–460.

[69] Y.-H. Chen, T. Krishna, J. S. Emer, and V. Sze, "Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks," *IEEE journal of solid-state circuits*, vol. 52, no. 1, pp. 127–138, 2016.

[70] G. W. Burr, R. M. Shelby, S. Sidler, C. Di Nolfo, J. Jang, I. Boybat, R. S. Shenoy, P. Narayanan, K. Virwani, E. U. Giacometti *et al.*, "Experimental demonstration

and tolerancing of a large-scale neural network (165 000 synapses) using phase-change memory as the synaptic weight element," *IEEE Transactions on Electron Devices*, vol. 62, no. 11, pp. 3498–3507, 2015.

[71] T.-H. Yang, H.-Y. Cheng, C.-L. Yang, I.-C. Tseng, H.-W. Hu, H.-S. Chang, and H.-P. Li, "Sparse reram engine: Joint exploration of activation and weight sparsity in compressed neural networks," in *Proceedings of the 46th International Symposium on Computer Architecture*, 2019, pp. 236–249.

[72] S. Aga, S. Jeloka, A. Subramaniyan, S. Narayanasamy, D. Blaauw, and R. Das, "Compute caches," in *2017 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2017, pp. 481–492.

[73] A. Agrawal, A. Jaiswal, C. Lee, and K. Roy, "X-sram: Enabling in-memory boolean computations in cmos static random access memories," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 65, no. 12, pp. 4219–4232, 2018.

[74] S. Jain, A. Ranjan, K. Roy, and A. Raghunathan, "Computing in memory with spin-transfer torque magnetic ram," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 26, no. 3, pp. 470–483, 2017.

[75] D. Apalkov, B. Dieny, and J. M. Slaughter, "Magnetoresistive random access memory," *Proceedings of the IEEE*, vol. 104, no. 10, pp. 1796–1830, 2016.

[76] M.-F. Chang, J.-J. Wu, T.-F. Chien, Y.-C. Liu, T.-C. Yang, W.-C. Shen, Y.-C. King, C.-J. Lin, K.-F. Lin, Y.-D. Chih *et al.*, "19.4 embedded 1mb reram in 28nm cmos with 0.27-to-1v read using swing-sample-and-couple sense amplifier and self-boost-write-termination scheme," in *2014 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*. IEEE, 2014, pp. 332–333.

[77] L. Wei, J. G. Alzate, U. Arslan, J. Brockman, N. Das, K. Fischer, T. Ghani, O. Golonzka, P. Hentges, R. Jahan *et al.*, "13.3 a 7mb stt-mram in 22ffl finfet technology with 4ns read sensing time at 0.9 v using write-verify-write scheme and offset-cancellation sensing technique," in *2019 IEEE International Solid-State Circuits Conference-(ISSCC)*. IEEE, 2019, pp. 214–216.

[78] X. Tang, E. Giacomin, G. De Micheli, and P.-E. Gaillardon, "Post-p&r performance and power analysis for rram-based fpgas," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 8, no. 3, pp. 639–650, 2018.

[79] M. Garg, B. Singh *et al.*, "A comparative performance analysis of cmos and finfet based voltage mode sense amplifier," in *2016 8th International Conference on Computational Intelligence and Communication Networks (CICN)*. IEEE, 2016, pp. 544–547.

[80] F. Arnaud, P. Zuliani, J. Reynard, A. Gandolfo, F. Disegni, P. Mattavelli, E. Gomiero, G. Samanni, C. Jahan, R. Berthelon *et al.*, "Truly innovative 28nm fdsoi technology for automotive micro-controller applications embedding 16mb phase change memory," in *2018 IEEE International Electron Devices Meeting (IEDM)*. IEEE, 2018, pp. 18–4.

[81] M. Bocquet, D. Deleruyelle, H. Aziza, C. Muller, J.-M. Portal, T. Cabout, and E. Jalaguier, "Robust compact model for bipolar oxide-based resistive switching memories," *IEEE transactions on electron devices*, vol. 61, no. 3, pp. 674–681, 2014.

[82] A. Levisse, M. Bocquet, M. Rios, M. Alayan, M. Moreau, E. Nowak, G. Molas, E. Vianello, D. Atienza, and J.-M. Portal, "Write termination circuits for rram: A holistic approach from technology to application considerations," *Ieee Access*, vol. 8, pp. 109 297–109 308, 2020.

[83] A. Levisse, P.-E. Gaillardon, B. Giraud, I. O'Connor, J.-P. Noel, M. Moreau, and J.-M. Portal, "Resistive switching memory architecture based on polarity controllable selectors," *IEEE Transactions on Nanotechnology*, vol. 18, pp. 183–194, 2018.

[84] C. Nail, G. Molas, P. Blaise, G. Piccolboni, B. Sklenard, C. Cagli, M. Bernard, A. Roule, M. Azzaz, E. Vianello *et al.*, "Understanding rram endurance, retention and window margin trade-off using experimental results and simulations," in *2016 IEEE International Electron Devices Meeting (IEDM)*. IEEE, 2016, pp. 4–5.

[85] B. Feinberg, S. Wang, and E. Ipek, "Making memristive neural network accelerators reliable," in *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2018, pp. 52–65.

[86] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[87] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 1–9.

[88] S. Xie, R. Girshick, P. Dollár, Z. Tu, and K. He, "Aggregated residual transformations for deep neural networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 1492–1500.

[89] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," 2009.

[90] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam, and D. Kalenichenko, "Quantization and training of neural networks for efficient integer-arithmetic-only inference," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 2704–2713.

[91] D. Bortolotti, H. Mamaghanian, A. Bartolini, M. Ashouei, J. Stuijt, D. Atienza, P. Vandergheynst, and L. Benini, "Approximate compressed sensing: ultra-low power biosignal processing via aggressive voltage scaling on a hybrid memory multi-core processor," in *Proceedings of the 2014 international symposium on Low power electronics and design*, 2014, pp. 45–50.

[92] X. Wang, Y. Han, V. C. Leung, D. Niyato, X. Yan, and X. Chen, "Convergence of edge computing and deep learning: A comprehensive survey," *IEEE Communications Surveys & Tutorials*, vol. 22, no. 2, pp. 869–904, 2020.

[93] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov 1998.

[94] F. Chollet, "Xception: Deep learning with depthwise separable convolutions," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 1251–1258.

[95] P. Molchanov, S. Tyree, T. Karras, T. Aila, and J. Kautz, "Pruning convolutional neural networks for resource efficient inference," *arXiv preprint arXiv:1611.06440*, 2016.

[96] B. Reagen, U. Gupta, L. Pentecost, P. Whatmough, S. K. Lee, N. Mulholland, D. Brooks, and G.-Y. Wei, "Ares: A framework for quantifying the resilience of deep neural networks," in *Proceedings of the 55th Annual Design Automation Conference*, 2018, pp. 1–6.

[97] B. W. Denkinger, F. Ponzina, S. S. Basu, A. Bonetti, S. Balási, M. Ruggiero, M. Peón-Quirós, D. Rossi, A. Burg, and D. Atienza, "Impact of memory voltage scaling on accuracy and resilience of deep learning based edge devices," *IEEE Design & Test*, vol. 37, no. 2, pp. 84–92, 2019.

[98] A. Jain, P. Goel, S. Aggarwal, A. Fell, and S. Anand, "Symmetric k-means for deep neural network compression and hardware acceleration on fpgas," *IEEE Journal of Selected Topics in Signal Processing*, vol. 14, no. 4, pp. 737–749, 2020.

[99] J. H. Lee, J. Kong, and A. Munir, "Arithmetic coding-based 5-bit weight encoding and hardware decoder for cnn inference in edge devices," *IEEE Access*, vol. 9, pp. 166 736–166 749, 2021.

[100] S. Jeloka, N. B. Akesh, D. Sylvester, and D. Blaauw, "A 28 nm configurable memory (tcam/bcam/sram) using push-rule 6t bit cell enabling logic-in-memory," *IEEE Journal of Solid-State Circuits*, vol. 51, no. 4, pp. 1009–1021, 2016.

[101] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[102] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga *et al.*, "Pytorch: An imperative style, high-performance deep learning library," *Advances in neural information processing systems*, vol. 32, 2019.

[103] M. Eggimann *et al.*, "Rosetta (2020)," http://asic.ethz.ch/2019/Rosetta.html.

[104] A. Garofalo *et al.*, "Darkside (2021)," http://asic.ee.ethz.ch/2021/Darkside.html.

[105] A. Waterman, Y. Lee, D. A. Patterson, and K. Asanovic, "The risc-v instruction set manual, volume i: Base user-level isa," *EECS Department, UC Berkeley, Tech. Rep. UCB/EECS-2011-62*, vol. 116, 2011.

[106] P. D. Schiavone, S. Machetti, M. Peon Quiros, J. A. Miranda Calero, B. W. Denkinger, C. T. Müller, R. Rodríguez Álvarez, S. Nasturzio, and D. Atienza Alonso, "X-heep: An open-source, configurable and extendible risc-v microcontroller," 2023.

[107] M. A. Rios, W. A. Simon, A. S. Levisse, M. Zapater, and D. A. Alonso, "Associativity-agnostic in-cache computing memory architecture optimized for multiplication," Dec. 28 2021, uS Patent 11,211,115.

[108] W. A. Simon, M. A. Rios, A. S. Levisse, M. Zapater, and D. A. Alonso, "Memory chip or memory array for wide-voltage range in-memory computing using bitline technology," Aug. 17 2021, uS Patent 11,094,355.

[109] F. Ponzina, S. Machetti, M. Rios, B. W. Denkinger, A. Levisse, G. Ansaloni, M. Peón-Quirós, and D. Atienza, "A hardware/software co-design vision for deep learning at the edge," *IEEE Micro*, vol. 42, no. 6, pp. 48–54, 2022.

[110] A. Levisse, M. Rios, W.-A. Simon, P.-E. Gaillardon, and D. Atienza, "Functionality enhanced memories for edge-ai embedded systems," in *2019 19th Non-Volatile Memory Technology Symposium (NVMTS)*. IEEE, 2019, pp. 1–4.

Marco Antonio **RIOS**

Research Assistant at EPFL

Avenue d'Epenex 19
1024 Ecublens, Switzerland
+41 (0)78 678 72 33
mrc.atr@gmail.com
linkedin.com/in/mrcatr/

- **Out of box thinking**: curious, autonomous, and hands-on
- **Circuit design**: Integrated circuits and PCB design
- **Low-power design**: analog, digital and mixed signal design
- **Data analytics**: interpretation, characterization and visualization of data

EDUCATION

2019-Pres **École polytechnique fédérale de Lausanne (EPFL)** — Lausanne, Switzerland
PhD in Microsystems and Microelectronics
Thesis: Enabling Data-Intensive Applications on the Edge Thanks to RRAM and SRAM Memory Design for In-Memory Computing

2016-2018 **École polytechnique universitaire de Grenoble-Alpes** — Grenoble, France
Master in Integrated Systems Design - Double Degree Program

2013-2018 **Universidade federal do Paraná (UFPR)** — Curitiba, Brazil
Graduation in Electrical Engineering

CORE EXPERIENCE

2019-Pres **Embedded Systems Laboratory (ESL) - EPFL** — Lausanne, Switzerland
Research Assistant
Development of new architectures that merge memory and processing units (computing in- or near-memory). By bringing complex operations towards the memory, such as convolutions and sparse-matrix multiplications, the results show up 20x increase in energy efficiency and performance for artificial intelligence algorithms on low power edge devices.

- 2 patented architectures and 12 publications (8 conferences and 4 journals)
- Integrated circuits design (full- and semi-custom design down to 28nm, pos-layout, RTL design)
- In-memory acceleration of two's complement multiplications and multiply-accumulate overflow management
- Developed a Convolutional Neural Network (CNN) mapper for in-memory inferences
- Applications and architecture co-design to maximize energy efficiency of edge devices

2018 **Laboratoire d'électronique des technologies de l'information (CEA-Leti)** — Grenoble, France
6-month internship
SRAM cell design in 28nm FD-SOI technology with silicon 3D monolithic integration (3D CoolCube™)

- Statistical data analysis (6 sigma) of performance and stability of SRAM.
- Circuit characterization in schematic and pos-layout level.

2017 **Institute of Planetology and Astrophysics of Grenoble (IPAG)** — Grenoble, France
4-month internship
Development and characterization of Low Frequency Radar (LFR) receiver. Comprising the stages of signal filtering, amplification and demodulation

- Characterization of Butterworth, Chebyshev and Bessel filters
- Entire project carried out independently (PCB design, SMD soldering and circuit functionality verification)

ADDITIONAL EXPERIENCE

2019-2022 **Embedded Systems Laboratory (ESL) - EPFL** Lausanne, Switzerland
Teaching assistant for the Laboratory in Electronic design automation (EDA) based design
Full-custom design, semi-custom design and analog design

2014-2016 **Universidade federal do Paraná (UFPR)** Curitiba, Brazil
2-year undergraduate research project
Design and simulation of RF power amplifiers (PA) in 130 nm CMOS technology, with the stages of amplification and gain controllable, increasing energy efficiency.

2014-2016 **Universidade federal do Paraná (UFPR)** Curitiba, Brazil
Various teaching activities

- Teaching assistant for Electric Devices course
- ELETRIZAR: volunteer work on public schools to teach the fundamentals of electronics with fun and interactive experiments
- Scratch workshop: teaching programing concepts to kids
- Hands-on workshop on DC electronics: design, printing and soldering PCBs

TECHNICAL SKILLS

- **EDA tools:** Hspice, ELDO, Cadence Spectre, Cadence Virtuoso, Synopsys Design Compiler, Cadence Innovus
- **Programming languages:** Python (NumPy, Pandas), C and C++
- **Science communication:** 18 publications and several technical presentations, in addition to all my teaching activities, I have developed solid science communication skills.
- **Inter cultural communication:** from my under-graduation to my PhD, I have pursued to work in a multicultural environment
- Solid knowledge on MS office and Adobe Illustrator, Adobe Photoshop

Patents approved:

1. Rios, M. A., Simon, W. A., et al (2021). Associativity-agnostic in-cache computing memory architecture optimized for multiplication. U.S. Patent No. 11,211,115. Washington, DC: U.S. Patent and Trademark Office

2. Simon, W. A., Rios, M. A., et al (2021). Memory chip or memory array for wide-voltage range in-memory computing using bitline technology. U.S. Patent No. 11,094,355. Washington, DC: U.S. Patent and Trademark Office

LANGUAGES

Portuguese	Mother tongue
French	Masters in France, (B2)
English	High-level of writing and speaking
German	A1 level

PERSONAL DETAILS

28, married, brazilian.
Swiss driver's license (Visa type B)