System-on-Chip Architectures for Event-Driven Computing

Diss. ETH No. 28578

System-on-Chip Architectures for Event-Driven Computing

A dissertation submitted to
ETH ZURICH

for the degree of
Dr. sc. ETH Zürich

presented by
ALFIO DI MAURO
MSc Politecnico Di Torino DET
born April 9th, 1992
citizen of Carlentini, Italy

accepted on the recommendation of

Prof. Dr. Luca Benini, examiner
Prof. Dr. Luciano Lavagno, co-examiner

2022

Abstract

The number of electronic devices deployed around us has followed a steadily increasing trend over the last decade. As the transistor miniaturization and integration processes improved year after year, integrated circuits (IC) became increasingly more energy-efficient and computationally capable. In the Internet-of-Things (IoT) era, the number of interconnected electronic devices is destined to increase at the fast pace of a tenth of billions of new interconnected nodes per year. It is evident that the management of the enormous amount of data being transferred to centralized, highly powerful cloud computing nodes for semantically rich information extraction poses new challenges. Among the most relevant is the need for increasingly higher network bandwidth, lower data transmission latency, and higher energy efficiency and storage capacity. With such a present and future scenario in mind, research communities are exploring innovative solutions to tackle such challenges and reduce the need for a massive, energy-hungry, raw data transmission to cloud computing services. A promising approach that has been established over the last years is to move part of the semantic information extraction as close to the IoT sensor as possible by processing the raw data collected by sensors on-site. To fully achieve this goal, with a synergic effort, sensors must become more intelligent and provide less redundant information; simultaneously, the processing devices must efficiently collect such a stream of rich information and parsimoniously process it on-demand in a low energy budget. This thesis will tackle the quest for energy-efficiency information extraction at the edge from two different sides. On the one hand, we will explore novel event-based audio and video sensors, and we will show how such devices can be used to achieve an energy-proportional, sparse data transfer to conventional, digital system-on-chips (SoC) architectures. On the other hand, we will investigate innovative brain-inspired event-driven machine learning (ML) computing frameworks, such as Spiking Neural Networks (SNN), to build an end-to-end energy-proportional computing pipeline that can profit from the inherently sparse nature of the data acquired from such sensors, and that is deployable at the extreme edge. Throughout the thesis, we will demonstrate, on an ASIC manufactured in a 22nm technology node, how a non Von Neumann data-driven accelerator for SNNs can be efficiently integrated into a more standard RISC-V-centric ML-task-oriented SoC. Eventually, we will conclude our exploration by showing how brain-inspired error-resilient ML algorithms allow operating highly integrated digital processing engines in the notably more energy-efficient near-threshold regime, paving the way for more computationally capable, energy-efficient edge-computing nodes of the future.

Riassunto

Il numero di dispositivi elettronici distribuiti intorno a noi ha seguito una tendenza in costante aumento nell'ultimo decennio. La miniaturizzazione dei transistors e' migliorata anno dopo anno, e i circuiti integrati (IC) sono diventati sempre più efficienti dal punto di vista energetico e computazionalmente capaci. Nell'era dell'Internet delle cose "Internet of Things (IoT)", il numero di dispositivi elettronici interconnessi è destinato ad aumentare ad un ritmo di circa dieci miliardi di nuovi dispositivi interconnessi per anno. È evidente, che la gestione dell'enorme quantità di dati che vengono trasferiti su infrastrutture di cloud computing centralizzate, al fine di estrarne informazioni rilevanti, pone nuove sfide. Tra le più rilevanti c'è la necessità, sempre più elevata, di un'adeguata capacita' di trasmissione dei dati (banda di trasmissione), minore latenza nel trasmettere questi dati e maggiore efficienza energetica e capacità di archiviazione. Tenendo a mente lo scenario presente, ed immaginando, in prospettiva futuri sviluppi dello scenario corrente, le comunità di ricerca stanno esplorando soluzioni innovative per affrontare tali sfide e ridurre la necessità di trasmettere dati grezzi, operazione che richiede ingenti quantita' di energia, ai servizi di cloud computing. Un approccio promettente, che si e' affermato negli ultimi anni, è quello di "portare" parte dell'estrazione di informazioni semantiche il più vicino possibile ai sensori che raccolgono dati dall'ambiente circostante. Per raggiungere pienamente questo obiettivo, con uno sforzo sinergico, c'e' la necessita' che i sensori distribuiti attorno a noi diventino sempre più "intelligenti", e quindi capaci di fornire flussi di dati poco ridondanti, e con un alto contenuto informativo; contemporaneamente, i dispositivi di elaborazione devono essere in grado di sfruttare, in modo efficiente, tali flussi di informazioni poco ridondanti ed elaborarle, solo quando necessario, utilizzando un budget energetico ridotto.

Questa tesi affronterà il tema dell'estrazione efficiente di informazioni provenienti da sensori distribuiti nell'ambiente circostante da due angolazioni diverse. Da un lato, esploreremo nuove categorie di sensori audio e video capaci di produrre eventi asincroni in corrispondenza di stimoli esterni, e mostreremo come tali dispositivi possono essere utilizzati per ottenere un trasferimento di dati sparso, ed un consumo di energia proporzionale al numero di eventi trasmessi. D'altra parte, esploreremo algoritmi innovativi di "machine learning" capaci di processare singoli eventi, come Spiking Neural Networks (SNN), al fine di costruire un flusso di elaborazione dei dati (dal sensore al dispositivo di elaborazione) proporzionale al numero di eventi acquisiti. In questa tesi, dimostreremo, su un circuito integrato progettato per questo specifico scopo, "Application Specific Integrated Circuit" o ASIC, fabbricato in tecnologia a 22nm, come un acceleratore basato su architettura non-Von Neumann per SNN può essere integrato, in modo efficiente, in un "System on Chip" (SoC) per machine learning piu' classico, basato su processori RISC-V.

Alla fine, concluderemo la nostra esplorazione mostrando come alcuni algoritmi di machine learning, siano particolarmente resilienti ad errori introdotti dall'utilizzo dei circuiti integrati, al di fuori delle loro specifiche nominali di funzionamento, e nello specifico in un regime operativo chiamato "near-threshold", che consente ai circuiti integrati di essere più efficienti dal punto di vista energetico.

Contents

Abstract **v**

Riassunto **vii**

1 Introduction **1**
- 1.1 Motivation 2
- 1.2 Energy efficient edge-computing devices 3
- 1.3 Next generation applications requirements 6
- 1.4 Brain-inspired edge-computing 8
- 1.5 Outline 9
- 1.6 Contributions 11
- 1.7 List of publications 12

2 Energy-proportional data acquisition **17**
- 2.1 Introduction 17
- 2.2 Related work 20
- 2.3 Architecture 22
  - 2.3.1 Silicon Cochlea Interface 22
  - 2.3.2 Silicon Cochlea Sensor node 26
  - 2.3.3 Dynamic Vision Sensor Interface 28
  - 2.3.4 Dynamic Vision Sensor Node 32
  - 2.3.5 System-on-chip integration 35
- 2.4 Experimental Results 37
  - 2.4.1 DASI FPGA standalone sensor node results . . 37
  - 2.4.2 DVSI FPGA standalone sensor node results . . 42
  - 2.4.3 Silicon implementation results 45

2.5 Conclusion 49

3 Overcoming Reliability Boundaries 53
3.1 Introduction 53
3.2 Related work 57
3.2.1 IoT End-Node Architectures 57
3.2.2 Heterogeneous and Error Resilient Memory Architectures 59
3.2.3 Dedicated Hardware Accelerators for DNNs and BNNs 60
3.3 Architecture 61
3.3.1 Quentin chip 61
3.3.2 BNN accelerator 62
3.3.3 Memory partitioning 63
3.3.4 BNN error resilience 65
3.4 Results 67
3.4.1 Experimental setup 68
3.4.2 Bit Error Rate analysis 70
3.4.3 Power and energy consumption 71
3.4.4 Power accuracy trade-off 76
3.5 Conclusion 78

4 Energy-proportional data processing 81
4.1 Introduction 81
4.2 Related work 83
4.2.1 Analog and mixed-signal neuromorphic platforms 84
4.2.2 Digital neuromorphic platforms 84
4.3 Architecture 85
4.3.1 Spatial and temporal data representation . . . 86
4.3.2 Execution model and mapping 87
4.3.3 Data transfer 88
4.3.4 Interconnect 90
4.3.5 Computing engines 91
4.3.6 SNE neuron model 94
4.4 Experimental results 99
4.4.1 Physical implementation 99
4.4.2 Area breakdown 100
4.4.3 Experimental setup 101

4.4.4 Power consumption and performance 101
4.4.5 Energy consumption 105
4.4.6 Comparison with the state of the art 107
4.5 Conclusion 110

5 Kraken: An event-driven Brain-Inspired edge computing device 113
5.1 Introduction 113
5.2 Related work 116
5.3 Architecture 117
5.3.1 Power and clock domains 118
5.3.2 Fabric controller 118
5.3.3 IO subsystem 120
5.3.4 Compute cluster 120
5.3.5 Accelerator domain 121
5.4 Physical implementation 121
5.5 Results 123
5.5.1 General purpose computing engines 124
5.5.2 Frequency and Power consumption 125
5.5.3 Performance and Energy consumption 126
5.5.4 Event-driven application scenario 127
5.6 Conclusion 133

6 Event-driven SNN deployment 135
6.1 Introduction 135
6.2 Related work 138
6.2.1 applications for SNNs 139
6.2.2 Unsupervised SNN training 139
6.2.3 Supervised SNN training 140
6.2.4 Fake quantization 141
6.2.5 True-quantization and integerization 142
6.3 Vertical software stack 143
6.3.1 Neural network quantization-aware training . . 143
6.3.2 PULP RISC-V toolchain 147
6.3.3 SNE software primitives 149
6.3.4 Automatic code generation 153
6.4 End-to-end application deployment to Kraken 157
6.4.1 results 158

6.5 Conclusion 159

7 Summary and Conclusion 161
7.1 Main results 161
7.2 Future Work and Outlook 164
    7.2.1 Future Work 164
    7.2.2 Outlook 165

A Chip gallery 167
A.1 Kraken 168
A.2 Marsellus 169
A.3 Vega 170

B Notation and Acronyms 171
Acronyms 171

Bibliography 175

Curriculum Vitae 203

Chapter 1

Introduction

The number of connected electronic devices being deployed around us is rapidly increasing, and this trend is expected to accelerate in the near future. Nowadays, billions of sensing devices are being employed to collect and analyze various data types. For example, electronic devices collect environmental data like temperature and humidity in our homes and offices to provide helpful information about the environment where we live. Also, electronic devices are used to monitor health parameters like our heart rate, level of oxygen in the blood, or stress level to provide helpful insight about our health state or track and record statistics about our sports activities. Moreover, sensors are distributed in our cities to monitor the traffic situation and air quality in real-time, collect information related to water levels of lakes and rivers or monitor critical infrastructures maintenance state [1]. Connected electronic devices are also a fundamental component of any modern industrial manufacturing equipment, tracking critical production line parameters to detect malfunctioning equipment. We can safely state that electronic devices, and more specifically electronic sensors, are very pervasive and are now deeply embedded in our lives and part of our daily routines.

1.1 Motivation

As electronic devices become more portable, better interconnected, and consume lower power, the number of fields where such devices are employed will rapidly increase. One aspect deeply related to the pervasiveness of electronic devices, which is already posing complex challenges today, is the treatment, namely processing, of large volumes of data such devices collect from the environment. Indeed, the ultimate goal of any such device is to extract meaningful information from raw non-Human-readable data, interpret them, and expose them to a user or another machine at a higher abstraction level.

The current approach to extracting semantically meaningful information from raw sensor data streams is to use complex and powerful machine learning (ML) algorithms. Over the last years, thanks to the scientific communities' research efforts, artificial neural networks (ANNs) significantly improved their performances, easily surpassing human ones in tasks like object detection or image classification [2]. This groundbreaking milestone was achieved thanks to the increasing size of the models, which allowed ANNs to reach supreme cognitive capabilities. We started to refer to neural network models as deep neural networks (DNNs) because of their many hidden computing layers.

We can identify two fundamental costs associated with extracting information from raw data streams. The first is the cost of transferring raw data streams from the sensor node to the computing engines where the processing algorithm is executed. Sensors were connected initially over wired data links to centralized host machines. However, because of the high number of interconnected devices, today's preferred communication channel is represented by wireless connections. Despite significant improvements in the field of wireless communication [3], the data transfer still represents a significant fraction of the energy spent to extract information from sensor data [4].

The second aspect is the required computing power to execute ML algorithms. As we mentioned before, DNNs have shown excellent capabilities in solving complex tasks. However, the number of FLOPS required by such algorithms is typically very high. Most of those algorithms can only execute on remote cloud computing facilities or, in the best case, on powerful host machines equipped with dedicated graphics process units (GPUs) or tensor process units (TPUs).

To deploy increasingly complex cognitive tasks, we must design more efficient, smarter, autonomous edge devices that extract semantically meaningful content from raw sensor data streams, process them efficiently on-site, and deliver distilled semantically rich information. In this thesis, this challenge will be tackled from two converging directions. On one side, we will investigate the process of acquiring raw data from emerging sensing devices efficiently. On the other side, we will explore promising data processing algorithms to design more intelligent and efficient edge-computing devices.

1.2 Energy efficient edge-computing devices

The quest to achieve high energy efficiency has profoundly influenced industrial and academic research in digital integrated circuits. As new application fields have emerged, the variety of different platforms operating at the edge has considerably increased over the last decade. Almost all industrial and academic institutions operating in such field nowadays propose edge-computing solutions [5–8] to address one or more market segments. Depending on the application context, which typically enforces constraints in terms of power budget, latency and throughput, edge computing platforms might feature very different architectures.

Low-power applications like near sensor analytic application (NSAA), biosignal, audio/vibration, low-resolution imaging, indoor localization [9] are typically deployed on low-end, resource-constrained devices such as the one presented in [10]. Such platforms often exploit aggressive power consumption reduction techniques to effectively use the limited amount of available energy for active computational phases [11–14]. When sensors are not providing stimuli to be processed, such devices try to reduce as much as possible the power consumption, leading to extremely low energy consumption [15]. Low-power idle capabilities become crucial when edge-computing nodes are supplied by small batteries, with limited or no possibility to replace them during the device life-cycle [16].

Low-end processors operating in these scenarios typically feature a low number of pipeline stages and a 32bit instruction set architecture (ISA). Typical workloads for such platforms involve control tasks or lightweight processing [17]. In some cases, such platforms might be specialized to accelerate low-footprint DNN models through the introduction of dedicated ISA extensions to reduce the energy of frequently executed instructions [18]. Low-end processors can typically provide hardware support to execute low-complexity real-time operating systems [19] that have a reduced set of useful features and application programming interface (API) libraries to support real-time tasks and, at the same time, minimize the firmware code size; this represents one of the main requirements of real-time applications [20].

In more complex, high-performance application scenarios, like high frame-rate visual application, or accurate ML algorithms like deep learning algorithms, computing nodes are typically equipped with high-end processors [21, 22]. Such nodes typically feature 64bits, dedicated high throughput FPU to sustain intense floating point workloads, virtual memory management capabilities, and the possibility to boot complex operating systems like Linux [21].

Edge-computing platforms have experienced a profound transformation in the last two decades. The miniaturization of the transistors has closely followed a well-known trend known as Moore's law [23], which has served as a reference for all semiconductor foundries and EDA tool companies. In the last decade, such a technological scaling has introduced new opportunities for devices' energy efficiency improvements, but that has forced researchers and designers to confront new and unforeseen challenges.

Highly scaled devices have started using supply voltages close to the transistors' threshold voltages. This phenomenon, known as near-threshold computing (NTC), has led to a tremendous reduction in the energy consumption of digital processors. However, it forced devices to operate in a regime where the maximum achievable frequency was significantly reduced because of the lower supply voltage of the transistors [23]. For example, other collateral effects, the temperature effect inversion (TEI), were observed when devices operate near the threshold. To counteract detrimental effects, e.g., performance or energy efficiency loss due to environmental conditions variations, solutions like the adaptive body-biasing (ABB) [24], sophisticated circuit logic delay-related clock adaptations [25, 26] were put in place to reduce sign-off margins and optimize the power consumption and performance of digital circuits.

To compensate for the overall frequency reduction in favor of a higher energy efficiency caused by the modern digital circuits' lower operating voltage, system-on-chip (SoC) architectures have evolved toward the so-called parallel computing framework [15]. The idea behind this approach is straightforward yet very effective, and it consists of executing sections of code whose data have no interdependence on multiple central process units (CPUs). As a result, the entire program executes faster, as multiple parts can run in parallel. The effectiveness of this approach is demonstrated by the large number of parallel processing devices that have reached state-of-the-art (SoA) performance in the application, and high-performance computing domains [27–29] as well as the internet of things (IOT) domain [13,30].

In the same regard, a consensus has built on the fact that hardware specialization typically leads to higher energy efficiency [31]. Modern IOT nodes often feature dedicated accelerators to execute specific workloads efficiently, for example, DNN accelerators [32–34]. Having a dedicated accelerator hosted by a SoC is not, per se, a novel concept. However, thanks to the accessibility to cheaper manufacturing technologies and EDA tools, as well as a richer ecosystem of open-source hardware IPs, like the ones developed in the context of the PULP project[1], research communities and industrial institutions have devoted an increasing effort in deploying specific computational kernel to custom hardware blocks; eventually reducing devices energy consumption.

---

[1] https://pulp-platform.org/

The advances in the ML research community have been a strong incentive for adopting dedicated DNN accelerators. Training environments like Pytorch[2] or Tensorflow[3], combined with Quantization-aware training frameworks that are built on top of those like Quantlab [4] or NEMO[5], have allowed achieving remarkable classification accuracy results on complex datasets like CIFAR10 or Imagenet with constrained resources, hardware-friendly DNN models [35–37] running on low-power resource-constrained resources devices. Such devices implement relatively simple low arithmetic precision hardware computing primitives [38].

1.3 Next generation applications requirements

We expect an increasing number of complex applications to be deployed on edge devices in the following years. Here we provide a representative, non-exhaustive list of application fields developed by research communities that require energy-efficient, highly-specialized edge computing platforms, which will directly benefit from more efficient edge computing devices. Wearable and healthcare-connected devices aim to assist users by extracting and summarizing helpful information and statistics related to daily activities. Such devices can implement applications that range from connected step counters to complex heart rate or oxygen concentration monitoring. Other connected devices like smart assistants can interact with a user through natural language-based voice exchanges or gesture-encoded commands. Recent wearable devices target innovative prosthetic applications to recover lost functionalities through non-invasive EMG muscle signals recording. Also, more recent research on smart glasses aims to implement eye-tracking algorithms for brain-machine interfaces or smart field-of-view rendering for virtual or augmented reality applications.

---

[2] https://pytorch.org/

[3] https://www.tensorflow.org/

[4] https://github.com/pulp-platform/quantlab

[5] https://github.com/pulp-platform/nemo

Robotics is another very active application field where edge-computing platforms will play a significant role, especially on nano and pico-sized robots, where the energy consumed by the actuators is on the same order of magnitude as the one allocated to computing resources. Over the last years, significant interest has built on new generation sensor types like artificial skins capable of reproducing a vast range of human touch feelings. Additionally, steadily growing communities are building around event-based sensors for audio and video applications like silicon cochleas [39], which are devices that reproduce the inner mechanics of the human ear, and silicon retinas [40], reproducing most of the functions of biological mammalian retinas. Such devices have distinguished themselves from their more conventional counterparts for their lower power, higher dynamic range, low information encoding redundancy, and required communication bandwidths. Among robotic applications, a fast-developing field is unmanned vehicles, specifically those focusing on unmanned aerial vehicles (UAVs). Such devices are particularly useful in many real-life scenarios [41]. For example, nano and pico-size drones can provide easy access to critical infrastructures that are difficult to inspect and maintain, like duct or narrow enclosures. Moreover, small autonomous devices can ensure low-risk post-disaster inspection or intervention and critical military applications. Moreover, such devices can also be employed for entertainment applications like aerial footage.

Industrial applications may represent the most prominent field for next-generation sensor and edge-computing platforms. A smart industrial plant can use connected distributed devices to monitor the maintenance state of production lines and promptly detect malfunctioning equipment, reduce the accident rate, track stored manufactured parts, and collect statistics to increase productivity.

Based on the application fields we have listed in this section, we can identify a few common requirements that characterize most of them. On the one hand, there is the need to efficiently combine information arriving from multiple sensors to a single computing platform. Sophisticated cognitive applications often rely on a combination of data streams coming from multiple complementary sensors [42, 43]. This approach, often referred to as “multi-sensor fusion”, allows to combine the data acquired from each connected sensor to achieve better application performance in terms of latency or classification accuracy [43, 44].

A second requirement, directly related to the high amount of collected data, is to distill the information acquired from each sensor to feed cognitive algorithms already with semantically rich data streams. A promising approach going in this direction is to exploit, both at the sensor edge and on the computing engine, the high level of redundancy, or conversely, the high information sparsity, that often characterizes raw data streams, to reduce the sensor data transfer [45] and processing energy cost [46] at the edge.

1.4 Brain-inspired edge-computing

To provide adequate computational support to the applications mentioned above, we need edge devices capable of efficiently connecting with a vast set of peripherals and multiple sensor communication protocols. At the same time, they have to be capable of performing the computation in a highly optimized way without sacrificing flexibility and applicability to yet unforeseen scenarios. A promising strategy to balance such requirements is to use highly efficient platforms equipped with dedicated accelerators optimized to execute model-free brain-inspired algorithms, e.g., ANNs. Such algorithms have become very popular for their high versatility and delivered quality of service; in many tasks, for example, object classification, ANNs, and specifically DNNs, have surpassed human performance.

Modern platforms have been equipped with dedicated hardware accelerators targeting highly regular DNN workloads. However, to efficiently process sparse heterogeneous sensor data streams acquired from next-generation sensors, e.g., silicon cochleas or retinas, we need a significant paradigm shift in how computing engines targeting ML applications are designed. Current accelerators primarily target frame-based inspired computation. Moreover, to profit from the high degree of sparsity that already arises in conventional DNNs [46], and which characterizes novel promising algorithms like spiking neural networks (SNNs), we need hardware accelerators that efficiently handle sparse computation without necessarily relying on the specific model of data sparsity [47] to deliver high energy efficiency; such sparsity model is hardly predictable in many scenarios, and difficult to foresee at design time.

This thesis addresses the energy efficiency challenge at the edge from the technological, hardware, and algorithmic angle.

1.5 Outline

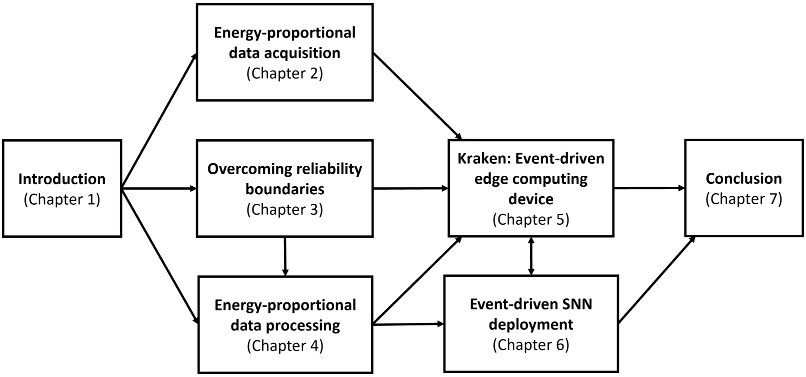

Energy-proportional data acquisition (Chapter 2)
Introduction (Chapter 1)
Overcoming reliability boundaries (Chapter 3)
Energy-proportional data processing (Chapter 4)
Kraken: Event-driven edge computing device (Chapter 5)
Event-driven SNN deployment (Chapter 6)
Conclusion (Chapter 7)

Figure 1.1: Block diagram representation of the structure of the thesis

In the following, we provide an outline of the thesis and give a short overview of the content of the following chapters. The content presented throughout the chapters of this thesis has already been published in conference proceedings and journal papers.

Chapter 2

The second chapter discusses a promising approach for SoA event-sensor data collection. It provides contributions that allow overcoming bandwidth limitations and achieving high data-to-information acquisition and transmission proportionality. The approach presented in this chapter is demonstrated on two event-based sensors acquiring audio and video data, respectively.

Chapter 3

This chapter demonstrates how inherently error-resilient applications like binary neural networks (BNNs) can be effectively deployed on carefully memory-partitioned IOT nodes operating in the NTC regime to achieve high energy efficiency at a negligible quality of service cost.

Chapter 4

The fourth chapter presents a dedicated SNN accelerator meant to be integrated into a system analogous to the one presented in the third chapter and capable of performing energy proportional SNN inference on event streams like those produced by the sensors in chapter two. The proposed accelerator is implemented on silicon in an advanced 22nm technology node.

Chapter 5

This chapter presents Kraken, a complete SoC implemented in a 22nm technology node hosting one of the peripherals presented in chapter two and the accelerator presented in chapter four. The IOT node presented in this chapter represents the first complete SoC featuring a dedicated neuromorphic accelerator targeting low-power embedded applications and an energy-efficient, SoA general-purpose computing cluster of 8 RISC-V cores.

Chapter 6

This chapter discusses and illustrates the complete software stack required to deploy neuromorphic applications on the system presented in chapter five. Chapter six explains the main challenges of training quantized SNNs, showing how such challenges can be effectively addressed to deploy complete end-to-end applications on advanced neuromorphic embedded edge-computing platforms like Kraken.

Chapter 7

Chapter seven provides a concluding discussion and an outlook on the possible future evolution of edge-computing platforms.

1.6 Contributions

The main contributions of this thesis, as well as the related publications, can be summarised as follows:

1. The design of a digital, ultra-low power, asynchronous interface for a novel event-based audio sensor and its field-programmable gate array (FPGA) implementation and characterization in connection with the real sensor. Silicon estimations of the proposed peripheral in an advanced 22nm technology (chapter 2, [45])

2. The design of a digital, ultra-low power interface for a novel visual event-based camera sensor, FPGA implementation, and characterization in a wireless connected visual sensor node. Silicon estimations of the proposed event-based visual sensor interface in an advanced 22nm technology (chapter 2, [48])

3. A study on the energy-efficiency gain opportunities of using error-resilient BNN applications deployed to advanced 22nm ultra-low-power computing node featuring heterogeneous memory subsystems (chapter 3, [49]).

4. The design and silicon prototyping of a SoA novel ultra-efficient event-driven SNN accelerator for 4bit-quantized embedded neuromorphic applications (chapter 4, [50]).

5. The design and silicon prototyping of Kraken; an advanced SoC with power management capabilities integrating an embedded neuromorphic accelerator, a dedicated peripheral for event-based vision sensor, and a SoA general purpose ultra-efficient 8 core RISC-V cluster (chapter 5).

6. A training flow for quantized SNNs, and the related deployment flow for producing code executable on ultra-low power-constrained resources platforms like Kraken (chapter 6).

1.7 List of publications

Most of the content of the thesis has been published in the following international conferences and journals:

[51] A. Di Mauro, M. Scherer, D. Rossi and L. Benini, ”Kraken: A Direct Event/Frame-Based Multi-sensor Fusion SoC for Ultra-Efficient Visual Processing in Nano-UAVs,” 2022 IEEE Hot Chips 34 Symposium (HCS), 2022, pp. 1-19, doi: 10.1109/HCS55958.2022.9895621.

[45] A. Di Mauro, F. Conti, and L. Benini, “An ultra-low power address-event sensor interface for energy-proportional time-to-information extraction,” in 2017 54th ACM/EDAC/IEEE Design Automation Conference (DAC), 2017, pp. 1–6.

[49] A. D. Mauro, F. Conti, P. D. Schiavone, D. Rossi, and L. Benini, “Always-on 674uw@4gop/s error resilient binary neural networks with aggressive SRAM voltage scaling on a 22-nm IoT end-node,” IEEE Transactions on Circuits and Systems I:Regular Papers, pp. 1–14, 2020.

[52] A. D. Mauro, F. Conti, P. D. Schiavone, D. Rossi, and L. Benini, “Pushing on-chip memories beyond reliability boundaries in micropower machine learning applications,” in 2019 IEEE International Electron Devices Meeting (IEDM), 2019, pp. 30.4.1–30.4.4.

[48] A. Di Mauro, M. Scherer, J. F. Mas, B. Bougenot, M. Magno, and L. Benini, “Flydvs: An event-driven wireless ultra-low power visual sensor node,” in 2021 Design, Automation Test in Europe Conference Exhibition (DATE), Feb 2021, pp. 1851–1854.

[50] A. Di Mauro, A. Suravi Prasad, Z. Huang, M. Spallanzani, F. Conti, and L. Benini, “Sne: an energy-proportional digital accelerator for sparse event-based convolutions,” 2022, design, Automation and Test in Europe Conference (DATE 2022); Conference Location: Online; Conference Date: March 14-23, 2022; Conference lecture held on 22 March 2022

The following works published by the author provide additional contributions on topics partially covered in this thesis:

[24] A. Di Mauro, D. Rossi, A. Pullini, P. Flatresse, and L. Benini, “Temperature and process-aware performance monitoring and compensation for an ulp multi-core cluster in 28nm utbb fd-soi technology,” in 2017 27th International Symposium on Power and Timing Modeling, Optimization and Simulation (PATMOS), 2017, pp. 1–8.

[53] A. Di Mauro, F. Zaruba, F. Schuiki, S. Mach, and L. Benini, “Live demonstration: Exploiting body-biasing for static corner trimming and maximum energy efficiency operation in 22nm fdx technology,” in 2020 IEEE International Symposium on Circuits and Systems (ISCAS), 2020, pp. 1–1.

[54] A. Di Mauro, D. Rossi, A. Pullini, P. Flatresse, and L. Benini, “Live demonstration: Body-bias based performance monitoring and compensation for a near-threshold multi-core cluster in 28nm fd-soi technology,” in 2018 IEEE International Symposium on Circuits and Systems (ISCAS), 2018.

[55] A. Di Mauro, D. Rossi, A. Pullini, P. Flatresse, and L. Benini, “Performance-aware predictive-model-based on-chip body-bias regulation strategy for an ulp multi-core cluster in 28 nm utbb fd-soi,” Integration, vol. 72, pp. 194–207, 2020.

[56] A. Di Mauro, H. Fatemi, J. P. de Gyvez, and L. Benini, “Idleness-aware dynamic power mode selection on the i.mx 7ulp iot edge processor,” Journal of Low Power Electronics and Applications, vol. 10, no. 2, 2020.

The author also contributed to the following publications:

[13] A. Pullini, D. Rossi, I. Loi, A. D. Mauro, and L. Benini, “Mr. Wolf: A 1 GFLOP/s Energy-Proportional Parallel Ultra Low Power SoC for IOT Edge Processing,” in ESSCIRC 2018 - IEEE 44th European Solid State Circuits Conference (ESSCIRC), Sep. 2018, pp. 274–277.

[17] P. D. Schiavone, D. Rossi, A. Pullini, A. Di Mauro, F. Conti, and L. Benini, “Quentin: an ultra-low-power pulpissimo soc in 22nm fdx,” in 2018 IEEE SOI-3D-Subthreshold Microelectronics Technology Unified Conference (S3S), 2018, pp. 1–3.

[15] D. Rossi, F. Conti, M. Eggimann, A. D. Mauro, G. Tagliavini, S. Mach, M. Guermandi, A. Pullini, I. Loi, J. Chen, E. Flamand, and L. Benini, “Vega: A ten-core soc for iot endnodes with dnn acceleration and cognitive wake-up from mram-based stateretentive sleep mode,” IEEE Journal of Solid-State Circuits, pp. 1–1, 2021.

[57] D. Rossi, F. Conti, M. Eggiman, S. Mach, A. D. Mauro, M. Guermandi, G. Tagliavini, A. Pullini, I. Loi, J. Chen, E. Flamand, and L. Benini, “4.4 a 1.3tops/w @ 32gops fully integrated 10-core soc for iot end-nodes with 1.7µ w cognitive wake-up from mram-based state-retentive sleep mode,” in 2021 IEEE International Solid- State Circuits Conference (ISSCC), vol. 64, 2021, pp. 60–62.

[58] P. D. Schiavone, D. Rossi, A. Di Mauro, F. K. G¨urkaynak, T. Saxe, M. Wang, K. C. Yap, and L. Benini, “Arnold: An efpga-augmented risc-v soc for flexible and low-power iot end nodes,” IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 29, no. 4, pp. 677–690, 2021.

[59] H. Okuhara, A. Elnaqib, D. Rossi, A. Di Mauro, P. Mayer, P. Palestri, and L. Benini, “An energy-efficient low-voltage swing transceiver for mw-range iot end-nodes,” in 2020 IEEE International Symposium on Circuits and Systems (ISCAS), 2020, pp. 1–5.

[60] M. Hersche, E. M. Rella, A. Di Mauro, L. Benini, and A. Rahimi, “Integrating event-based dynamic vision sensors with sparse hyperdimensional computing: A low-power accelerator with online learning capability,” in Proceedings of the ACM/IEEE International Symposium on Low Power Electronics and Design, ser. ISLPED ’20. New York, NY, USA: Association for Computing Machinery, 2020, p. 169–174.

[18] A. Garofalo, G. Ottavi, A. Di Mauro, F. Conti, G. Tagliavini, L. Benini, and D. Rossi, “A 1.15 tops/w, 16-cores parallel ultra-low power cluster with 2b-to-32b fully flexible bit-precision and vector lockstep execution mode,” in ESSCIRC 2021 - IEEE 47th European Solid State Circuits Conference (ESSCIRC), 2021, pp. 267–270.

Chapter 2

Energy-proportional data acquisition

2.1 Introduction

Over the last decade, we have observed a dramatic increase in complex “smart” applications based on multi-sensor data streams on ultra-low power, resource-constrained edge devices. In such a context, computing devices like micro-controller units (MCUs) have to extract high-level information from noisy, high-bandwidth, heterogeneous data streams. The information content of such date streams is often very sparse or is encoded redundantly. Therefore, to be able to extract relevant information for the designed task, computing devices need to execute computationally intense data analytic algorithms such as principal component analysis [61] for dimensional reduction, k-means [62] for clustering, support-vector machines [63], or neural networks [64] for classification. In many scenarios where a small battery powers the MCU, the desired cognitive task is too complex to be deployed on the same edge-computing device. Therefore, the only alternative is to send raw data streams to a higher-level computing infrastructure in the cloud, which has an enormous energy overhead and requires relatively high-bandwidth communication over a wireless channel. A promising solution to the problem mentioned above is to adopt more intelligent sensors capable of filtering, already at the sensor edge, most of the redundant information contained in raw data streams. With this approach, part of the semantic information extraction burden can be moved to the sensor, directly producing an output data stream that extracts the desired information for the specific application.

Among the “smart” sensors presented in the last years, a particularly interesting class is that of event-based spiking devices [65–67]. Such sensors mimic the functioning of biological neurons in the human retina and cochlea. The output of event-based sensors is often a stream of asynchronous events that are transmitted whenever the content of the underlying sensed analog signal has sufficient energy contained within a set of narrow frequency bands, in the case of silicon cochleas; or spatial coordinates (pixels) in the case of silicon retinas.

Unfortunately, event-based spiking sensors often use asynchronous interfaces to communicate with the external world. This behavior originates from the fact that the information content of the spike stream is contained not only in the spike “address” (i.e., position or frequency) but also in the relative inter-spike time delta [68]. Therefore, an asynchronous representation is naturally suited to encapsulate this information implicitly without adding the cost of converting this information into an explicit form, e.g., synchronization clocks or event timestamps. Indeed, many of these sensors were initially designed to couple with custom-made brain-like interfaces that are inherently asynchronous [69,70]. When such a class of sensors is used in combination with digital, synchronous, off-the-shelf MCUs, the implicit nature of this essential component of the information, which is embedded in a spike stream, makes the combination of these two categories of devices very challenging, and potentially, the information transfer towards ultra-low power edge computing devices very inefficient. Among the issues that can arise when such a class of devices is used in conjunction with synchronous devices, there is a fundamental one: crossing the border between the digital asynchronous and synchronous domains. This operation translates into requirements that need to be satisfied not to lose information. First, it is necessary to sample the spike stream with a sufficiently small sampling period to adequately represent short inter-spike times, as time in synchronous systems is quantized by definition; i.e., a synchronous clock frequency is used to time all the operations. More specifically, to account for the worst-case scenario, the shortest possible inter-spike time forces a high sampling frequency choice even if the average spike rate is low. Moreover, apart from fully streaming ASICs, most synchronous systems (such as off-the-shelf MCU) require temporary data storage in temporary memory to process any data. To this end, data must be transformed in a *latency-insensitive* form, i.e., all time-related information must be explicit to be stored indefinitely. For these reasons, building a link between an asynchronous event-based sensor and a commercial off-the-shelf MCU or a similar synchronous device can be considered a *time measurement* problem, with the additional constraint that power consumption must be kept within a very low-power envelope.

Even in those cases where the sensor already exposes a digital synchronous interface [71], the inherently sparse nature of the data produced by the sensor and the short period at which the active events can be potentially "dumped" from the sensor require a carefully crafted data acquisition pipeline. In such scenarios where the MCU can control the acquisition process, the main challenge is to preserve the proportionality between the information content being transmitted from the sensor and the energy spent to transfer each event, avoiding unnecessary too frequent requests to the sensors.

This chapter will explore two different event-based sensors, and we will propose two digital interfaces to perform energy-to-information proportional event acquisition from event-based sensors. The main goal of both architectures proposed in this chapter is to overcome the inefficiencies when event-based sensors need to be interfaced with commercial MCUs. The key contributions of this chapter can be summarized as follows:

- A fully digital architecture to acquire events from an event-based silicon cochlea in an energy-proportional way. We propose an ultra-low-power FPGA implementation of the circuit for such architecture, showing how silicon cochleas can be efficiently interfaced with off-the-shelf MCU, building a low-power smart audio sensor node.

- A fully digital architecture to acquire events from a silicon retina also called dynamic visual sensor (DVS). Also, for this sensor, we propose a low-power FPGA implementation of the proposed architecture. Additionally, we show how the full system can be interfaced to a remote host device over a Bluetooth Low Energy channel, surpassing the performance of conventional frame-based visual sensor nodes.

- We show how both peripherals can be integrated into a MCU IO peripheral subsystem. We provide power consumption estimates for a silicon implementation of such an IO peripheral subsystem in an advanced 22nm technology node.

2.2 Related work

Event cameras such as DVS are biologically-inspired vision sensors that operate fundamentally differently from frame-based cameras, i.e., they capture light intensity changes from the scene in the field of view [72]. Such delta brightness changes are asynchronously acquired and transmitted as independent pixel-related events, also known as spikes. The DVS events typically have a positive or negative polarity, corresponding to a transition from high brightness intensity to low brightness intensity and vice versa.

Event-based cameras present numerous advantages compared to traditional cameras [73]. Specifically, a spike-based representation of an image contains a high level of information content, reducing the redundancy caused by the transmission of those pixels that did not experience any change since the last frame transmission. Thanks to this innovative approach, the event-frame transmission is considerably more efficient in terms of required bandwidth and power consumption. DVS cameras are inherently data-driven: they only transmit pixels whose intensity variation has changed by a certain amount. Besides the lower bandwidth required for transmitting the scene's contained information, event cameras show several more advantages over conventional frame-based cameras due to their frame-less nature. Among those, event cameras have much faster response times, i.e., they can output single pixels with an inter-pixel time as accurate as a few μs, resulting in an equivalent time granularity achievable only on high frame-rate frame-based cameras. Additionally, they typically feature a very high dynamic range (140 dB vs. 40 dB of standard cameras) [73].

Similarly to DVS cameras, audio sensors such as silicon cochleas [74–76] typically work at tens/hundreds of kevt/s for typical speech scenarios, within a power envelope of less than 15 mW, and down to mere tens of µW for the latest sensor proposed by Yang et al. [39].

These unique features make event-based cameras, as well as silicon cochleas, ideal sensors for a variety of low-power applications; e.g., wearable image or sound acquisition, hand-gesture recognition, robotics, healthcare-oriented, and human-computer interaction (HCI), where the latency and energy efficiency are dominant constraints [72, 77, 78]. In such operating scenarios, the most stringent constraint is the limited energy budget available to acquire the data from the sensor. Therefore, effective use of the energy drained by the battery is consistently among the main concerns when designing such systems, and it becomes the priority when we scale such applications to the extremely low-energy budget application context [79].

Several interfaces for such a class of sensors have been proposed over the last few years. For example, interfaces from AER to PCI or USB have been developed both on FPGA [80–84], achieving sustainable event rates up to 10 Mevt/s in a power envelope in the order of hundreds of mW or more, and ASIC [85,86], with sustainable event rates up to 20Mevt/s. However, the energy spent to acquire the events produced by the sensor has not been investigated deeply. Indeed, the primary goal of most event-based interfaces is to serve as a testbench or debug interface for the sensor or perform the data readout in the highest event-rate scenario. This aspect significantly limits the use of these promising technologies on real application scenarios of wearable and low-power devices for mobile applications and healthcare when the energy budget is very limited [87].

This chapter will take inspiration from the interfaces already presented in the literature. However, as opposed to those, as the main objective, we will target a proportional power consumption scaling with the event rate of the sensor. This behavior allows for a nearly constant energy consumption per event, representing the ideal case at which a proportional amount of energy is consumed per event, regardless of the event rate. At the same time, no event is lost during the acquisition process [45].

2.3 Architecture

This section will describe the architecture of two peripherals that collect data from event-driven sensors. In the first part of the exploration, we will focus on designing an interface for event-driven audio sensors, specifically, a silicon cochlea. This sensor adopts an asynchronous protocol to transmit single audio events. The analysis conducted in this chapter will continue by focusing on a frame-less event-driven visual sensor called Dynamic Vision Sensor (DVS). We will start both investigations with the protocol description that such peripherals might adopt to stream single-pixel activity or individual energy quanta on a specific audio frequency range to a downstream processing device. We will highlight the implications of such an approach on the transmission bandwidth. Then, for both peripherals, we will describe a suitable hardware architecture capable of efficiently collecting the events from such types of sensors and exploiting the energy proportionality between the information acquired from the sensor and the data transferred to a downstream computing unit. Eventually, we will show how such peripherals can be efficiently integrated into the Input/Output peripheral subsystem of a digital MCU-class processing device to design event-driven edge computing nodes. The latter analysis will provide power estimates from post-synthesis and post-silicon implementation for the audio and video peripherals.

2.3.1 Silicon Cochlea Interface

As anticipated in the introduction, the first event-driven sensor we examine in this chapter is a silicon cochlea sensor, which asynchronously emits audio events depending on the sound recorded by dedicated low-power microphones.

The Event-based audio sensor

The energy-proportional audio sensor We targeted for our exploration is the *iniLabs DAS1* cochlea sensor[1], which mounts the *Cochlea AMSC1c* chip [76]. This sensor can mimic a human ear in silicon,

---

[1] `http://inilabs.com/products/dynamic-audio-sensor`

reproducing various internal dynamics and functions of several groups of cells with high fidelity. The data produced by such sensor relate to two input stereo channels, and they are transmitted asynchronously through an Address Event Representation (AER) protocol, which is a digital, asynchronous 4 phases handshake Figure 2.1.

The activity on the AER bus is proportional to the activity at the sensor's input, i.e., the number of audio events produced on both the sensor channels, depending on the intensity, frequency content, and duration of a recorded sound. The input frequency spectrum goes from 50 Hz to 20 kHz, which is approximately the frequency range of a human ear. The events produced by the *Cochlea AMSC1c* are encoded on 12 bits, and an additional polarity signal determines whether an event belongs to the left or right audio channel. The *Cochlea AMSC1c* can output a maximum of 10 Mevt/s, and the number of events produced when recording a human speech is approximately 20 kevt/s.

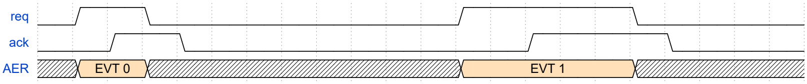


Figure 2.1: AER audio event transmission asynchronous protocol

Peripheral architecture

Three main macro-blocks form the hardware architecture of the dynamic audio sensor interface (DASI): *i)* an AER front-end, which acts as spike stream synchronization block and produces the timestamp-augmented Address-Event-Timestamp Representation (AETR) stream, *ii)* a buffer module, which can be configured to hold the AETR data to create a batch of audio events to be transferred in a block, *iii)* the *Clock Generator*, which provides the recursively divided clock, based on an input reference clock. A point-to-point combinational crossbar interconnects all the blocks that send or receive AETR data, and the configuration values are set through a register-based interface. Except for the request monitor inside the AER front end, all blocks are clock-gated by default and activated only when in active use. All modules use the same global variable frequency clock generated by the clock generator. Figure 2.2 shows a block diagram representation of the DASI peripheral.

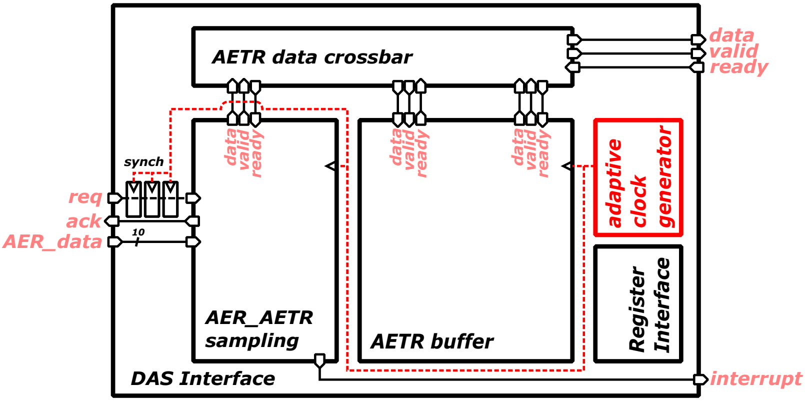


Figure 2.2: High-level block diagram of the DASI peripheral

Adaptive clock relaxation

Every event collected by the DASI does not contain time information; this means that, since the events are buffered in a local buffer, the peripheral must apply a timestamp to each event to reconstruct the exact event arrival time. The cochlea sensor and the DASI operate in a streaming fashion. Therefore the events are transferred in a continuous-time data transmission framework. To reduce the footprint of the timestamp associated with the events and make sure the timestamp has no upper bound, we decided to store only the time difference to the last received events, i.e., only the time delta between successive events is used as a timestamp. We only assume that two events must not be time-separated by more than a specific time delta. This value corresponds to the maximum time value represented in the allocated number of bits. If this value is exceeded, we consider such events as too distant in time to be correlated during any downstream processing.

The DASI receives the events in an asynchronous streaming fashion; therefore, it is impossible to know when and how many events will be received from the silicon cochlea. To ensure the acquisition of all events and still can timestamp such events with the appropriate time granularity, the operating frequency of the DASI must be higher than the maximum event rate produced by the sensor. At the same time, it is evident that operating the DASI at the maximum frequency is sub-optimal in case of low activity, as the events could be collected by operating the interface at a lower frequency, thereby with lower responsiveness.

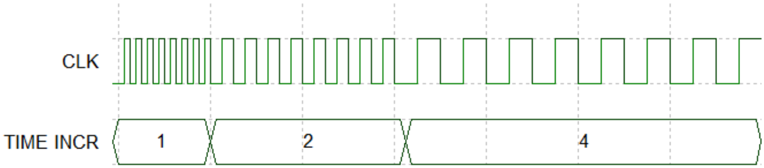


Figure 2.3: Visual representation of the progressively divided clock, along with the incremented timestamp increment, to keep the time count consistent with the original reference, not divided, clock frequency.

To balance both needs and reduce power consumption while ensuring no event loss and accurate timestamping, we developed a progressively-relaxed clock regime, where the operating clock frequency is iteratively halved after a programmable number of clock cycles. Note that the timestamp must be kept coherent with the current clock frequency by doubling the increment step in this operating regime. This approach automatically adjusts the average operating clock frequency of the DASI to the input event rate. The module responsible for generating the variable frequency clock for the whole peripheral is called *Self-adaptive Clock Generator* A finite state machine implements the internal control logic capable of adapting the output clock. The number of clock cycles after which the clock is progressively halved can be programmed via a register interface. Figure 2.3 provides a visual representation of the clock division operated by the *Self-adaptive Clock Generator*. Once a new event is received, the maximum operating frequency is restored to correctly timestamp, i.e., with enough time granularity, a potential event arriving very close in time to the one that has been just acquired.

2.3.2 Silicon Cochlea Sensor node

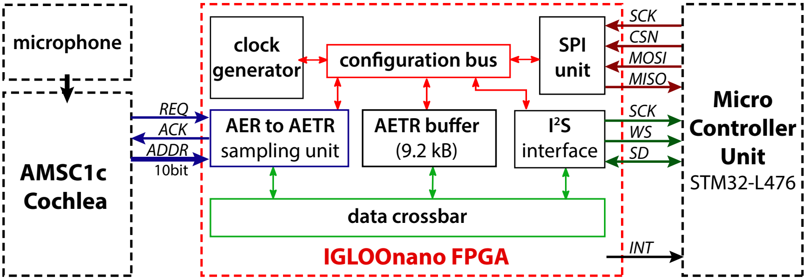


Figure 2.4: Block diagram of the DASI-based standalone FPGA sensor node implementation

This section shows how the DASI architecture can be deployed on a low-power FPGA to build a standalone event-driven energy-proportional audio sensor node. Such a sensor node finds its application in various fields, e.g., environmental audio monitoring, voice-controlled home appliance, industrial equipment audio monitoring, and voice activity detection (VAD) for human-machine interaction (HMI).

To make the DASI operate as a standalone device and to interface such a sensor node with an off-the-shelf MCU, we equipped the DASI-based sensor node with two communication interfaces. The audio event stream has been transferred to the downstream MCU over an Inter-Integrated-Circuit Sound (I^2S) protocol. This choice has been motivated by the fact that the I^2S is one of the de-facto standard protocols to transmit audio streams, and primarily because the I^2S protocol already provides the capability of transferring two parallel audio channels on the same stream of data, which in our case we mapped to the left and right channel of the AETR. All the configuration values of the DASI, accessible over the register-based interface, were mapped to a configuration bus connected to a standard slave SPI. As a MCU receiving the DASI audio streams, we choose the STM32 L476 MCU, a low-power industrial state-of-the-art MCU often adopted in such sensor nodes.

Figure 2.4 shows a high-level block diagram representation of the DASI-based sensor node. The architecture we present here comprises various submodules: *i)* the AER to AETR sampling and conversion module described in the previous section. *ii)* an AETR transmission buffer, which allows creating bursts of audio events and initiates an I^2S transfer only when a certain amount of audio events have been received from the silicon cochlea. *iii)* The I^2S interface that is connected to the downstream MCU. *iv)* The SPI, which is used to receive from the MCU the configuration values for the sub-modules of the sensor node. The AETR data stream travels across a configurable point-to-point crossbar, the main interconnection between the sub-modules composing the sensor node. The crossbar can also bypass the event buffer and directly forward the event stream from the AETR module to the I^2S peripheral.

Sensor node operation

The following steps characterize the DASI-based sensor node operation. Before starting the audio event acquisition, the DASI sensor needs to be configured. During this phase, the configuration values are transferred from the MCU to the DASI-based sensor node over the SPI peripheral and routed to each destination module via the configuration bus represented in Figure 2.4. Then, the audio event acquisition can start. As a first step of the event acquisition, the request coming from the sensor is synchronized by a series of cascaded flip-flops. A clock cycle later, the request is received, and the data lines are sampled and stored in a temporary data register. At the same time the request is received, the counter which traces the timestamp increment is sampled, and a time value is associated with the incoming event sampled by the temporary data register, then the counter is restarted. The reception of a request also triggers the restoration of the maximum operating frequency, such that all the operations above can be completed in the shortest amount of time possible; this ensures that the event stored in the temporary data register is promptly transferred to the AETR event buffer, and a new event acquisition can be performed in the shortest amount of time possible. Once a programmable number of events have been received from the sensor and stored in the AETR buffer, the audio events are transferred to the I^2S peripheral to be transmitted in a burst mode. The AETR buffer behaves like a FIFO; therefore, the order of the events and the precise arrival time is preserved during the acquisition process. The MCU must be configured to accept incoming I^2S audio stream such that the audio events can be stored in the MCU main memory. The processing of the audio events is outside the scope of this demonstration; therefore, we will not enter into the details of how such events are processed once they have been stored in the MCU memory.

2.3.3 Dynamic Vision Sensor Interface

The second event-driven sensor we want to analyze in this chapter is a video sensor. This section will describe how to efficiently acquire video events from such sensors.

The Event-based vision sensor

We selected the DVS132S sensor described in [71] as a case study. The circuit presented throughout the chapters of this thesis operates in the embedded device domain; therefore, we selected the sensor to use in combination with such a system according to the following requirements: low resolution and low-power consumption. In the case of the DVS132, the event camera sensor features a maximum resolution of 132x104 pixels, each with a size of 10 µm × 10 µm. it has a maximum event-rate of 180 Meps, and a minimum reported power consumption of 250 µW at 100 keps. Before describing the hardware architecture of the peripheral, we need to introduce the DVS132 transmission bus protocol. Figure2.5 and Figure 2.6 report the protocol used by the DVS sensor.

From Figure 2.5, it is evident that the sensor behaves like a typical CMOS vision sensor. Indeed, pixels are synchronously read out utilizing digital clocks and control signals to synchronize the rows

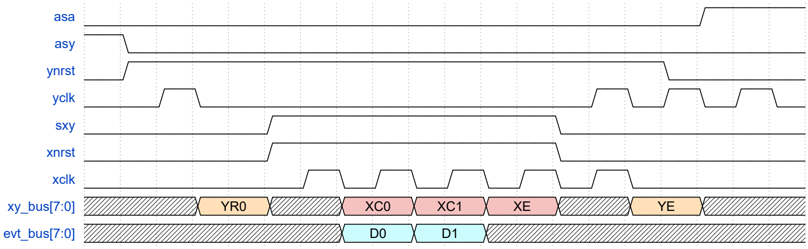


Figure 2.5:
DVS event transmission protocol

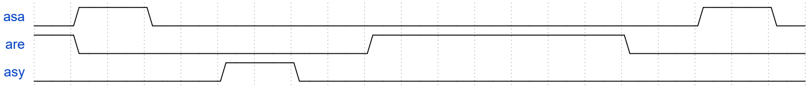


Figure 2.6: DVS event-frame transmission protocol

and columns readout. However, compared to the classic rolling shutter behavior often shown by conventional CMOS sensors, where the entire frame content is streamed out by spanning all column and row pixels of the sensor, the DVS explicitly provides the pixel address along with its value. This modification allows implementing a mechanism where the pixels with low brightness change do not stream out any event while the active ones are explicitly referenced. Such an approach drastically reduces the bandwidth required to transmit the equivalent information content of a standard frame and the space required to store the same information content in the memory.

Besides easy-to-solve technicalities like the higher pin count, this mechanic poses new challenges to hardware designers. At the peripheral level, the highly variable bandwidth requires low power operation but a fast peripheral reaction not to lose any pixel or stop the data stream transmission. At a system level, the data transfer needs to be optimized to guarantee the data transfer towards the main system memory, possibly without speculatively pre-allocating the worst-case scenario occupied memory. This section will provide insights into addressing the first challenge, specifically by providing a detailed description of the architecture of the dynamic visual

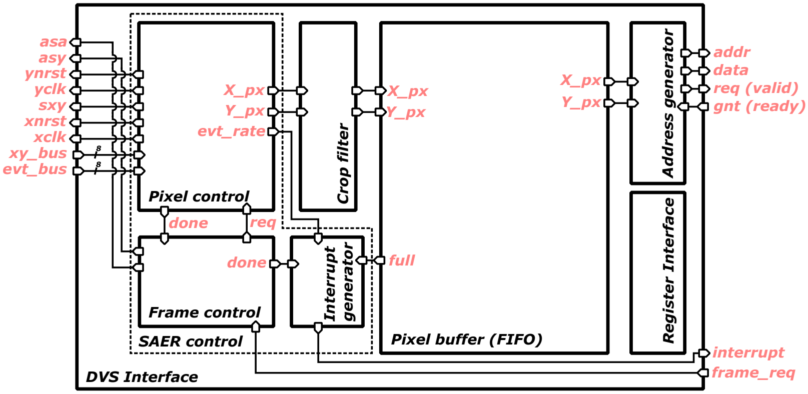


Figure 2.7: DVS high-level block diagram

sensor interface (DVSI) peripheral. System-level optimizations will be covered in a later section of this chapter.

Peripheral architecture

The peripheral architecture, reported in Figure2.7, is divided into two main parts. The Synchronous Address Event Representation (SAER) control, i.e., the protocol driver. This module is responsible for driving the sensor bus to acquire the data. The pixel buffer, this module stores a certain amount of event pixels while waiting for the system to be ready to transfer them to the main memory.

The sensor behavior resembles the one of a standard frame-based CMOS sensor. Therefore, the DVSI behaves as a CMOS sensor controller at a higher abstraction level. However, the SAER Controller is divided into two main sub-modules at a lower level. The first one is in charge of supervising the entire event-frame acquisition. As soon as an external frame request is set on the register interface, the `Frame controller` initiates the event-frame readout. Then, once the sensor is ready to stream out the event pixels, the `Frame controller` hands off the control to the `Pixel controller`. The latter is the module in charge of driving the sensor event pixel bus, acquiring the event pixels, and transferring them to the downstream modules.

The pixels are linearly stored in a temporary buffer during the acquisition, which acts as a first-in-first-out (FIFO) memory. The buffer guarantees that the pixel acquisition continues without interruption when the output bus of the peripheral becomes unavailable, for example, because it is serving the request of another module. Before sending the data to the buffer, a crop filter can be used to artificially reduce the sensor's field of view. Additionally, the peripheral can generate an interrupt when the buffer is full or when the event rate exceeds a certain threshold.

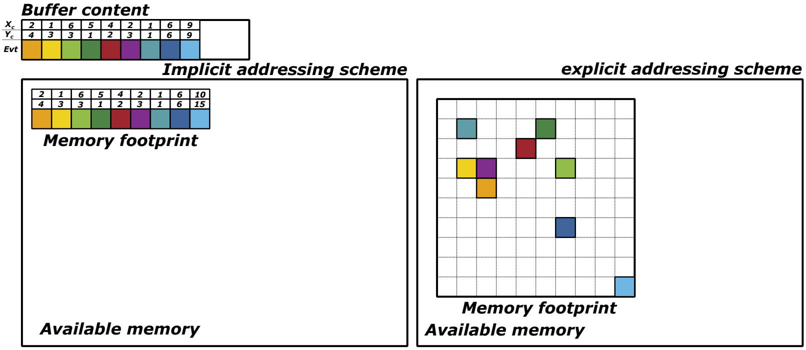


Figure 2.8: Memory footprint of an event-frame in case of explicit and implicit addressing scheme

The address generator is the module that transmits the acquired event pixels from the buffer to a downstream unit or the system memory. Depending on the application's requirements, two addressing schemes can be chosen. In the first one, a starting address is programmed into the address generator, and every time a pixel is sent out, a fixed step increment the address. In this mode, the pixels can be stored in the memory in the same order they are received from the sensor. The memory footprint of this addressing scheme depends on the sensor's activity, i.e., how many active pixels are present in an event frame collected from the DVS. In the second mode, the address of each pixel is composed by adding the base address programmed into the address generator and the `X` and `Y` coordinate of each event pixel. The memory footprint of this addressing scheme is generally higher, as potentially any pixel of the event frame is active; therefore, a memory region as big as an entire frame needs to be provisioned. Figure2.8 provide a graphical representation of the memory footprints of the two addressing schemes.

2.3.4 Dynamic Vision Sensor Node

This section describes the architecture of the DVSI-based node we used as a proof of concept to demonstrate the performance of the DVSI in a real-life operating scenario. Such types of sensor nodes find application in a plethora of use cases. For example, the sensor node described in this section could be employed as a static battery-operated surveillance camera. In that scenario, fast responsiveness to an event happening in the scene is required. At the same time, such a sensor node would need to feature extremely low power consumption when nothing happens in the field of view to avoid draining unnecessary energy from the battery. Other application examples for such types of sensors with similar requirements could be office occupancy monitoring, industrial machinery or production lines monitoring, and traffic monitoring.

We selected a small, low-power, yet reasonably fast FPGA device from the available off-the-shelf commercial Lattice devices to implement an efficient, low-power, and responsive sensor node. The challenging part of such an implementation is to find the right balance between the number of look up tables (LUTs) available on the FPGA, the maximum achievable frequency, and its power consumption. For this demonstrator, we used the Lattice Semiconductor iCE40UP5K [2]. The FPGA was supplied at its nominal voltages, i.e., 1.2 V for the fabric (core), and 3.3 V for the FPGA IO banks. We describe the complete sensor node circuit architecture implemented on the FPGA in the following.

Sensor node architecture

The sensor node is composed of three main sub-blocks. The first module of the proposed sensor node architecture is the DVSI. As described in the previous sections, this module interfaces the physical

---

[2] http://www.latticesemi.com/view_document?document_id=51968

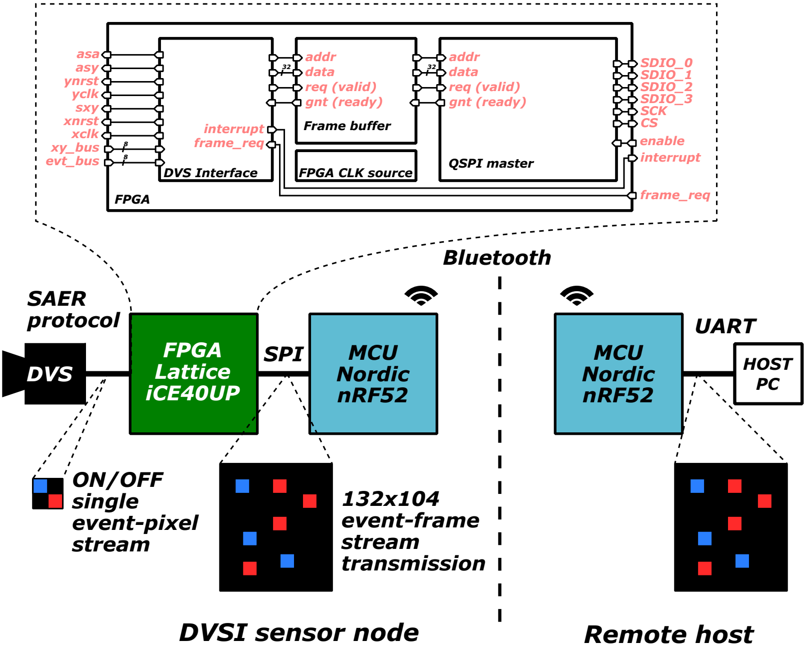


Figure 2.9: Block diagram of the DVSI-based sensor node implemented on FPGA

DVS through the FPGA pads. The DVS driver reads an event frame every time a new request comes from the downstream processing unit. Note that an event-frame request can be generated on-demand, i.e., at a variable event frame rate, by the downstream processing unit, or attached to a timer, i.e., at a fixed event frame rate.

The second module of the DVSI-based sensor node is called `Frame buffer`. This element can be seen as a temporary storage memory where to buffer the event frames read by the DVSI. The role of this module is twofold. On the sensor side, it ensures that the DVSI can continuously transmit an event frame read from the sensor. On the output transmission interface side, it allows packing the data received from the sensor into bursts, triggering size-optimized low-overhead data transmissions. This module has been implemented by mapping the memory element to the FPGA available embedded memories to optimize resource usage.

The third module composing the DVSI-based sensor node is a master quad-serial peripheral interface (QSPI). This module transmits the data stored into the `frame buffer` over a standardized protocol to a downstream processing unit. We chose to use a master QSPI because we wanted to reduce the complexity at the processing unit side. Indeed, the architecture in exam allows issuing an interrupt to signal that the frame buffer has been written with sufficient data. Therefore, a QSPI transaction will be triggered. The processing unit can use this interrupt to exit a sleep or deep-sleep state, prepare the slave QSPI peripheral to receive a known amount of data over the, and activate the `enable signal` when ready to acknowledge the interrupt reception. The maximum allowed size for the QSPI transfer is 256 B, and it is limited by the maximum programmable QSPI transfer size of the downstream processing unit.

We hard-coded the DVSI configuration signals to pre-defined values in this implementation. The DVSI has been configured to store the pixels in the same order they are received. This choice allows for reducing the size of the `frame buffer`, as the probability of receiving a number of active pixels equal to the size of the entire frame is extremely low. DVSI configuration signals could also be connected to QSPI-programmable configuration registers to change such values at run-time. Also, the clock was generated on-chip by exploiting the clock source available on the FPGA.

As peripheral MCU, we chose an nRF52 family SoC from Nordic Semiconductor. The SoC is built around a 64 MHz ARM Cortex-M4F microcontroller and hosts a 2.4 GHz Bluetooth transceiver specifically targeted for low-power Bluetooth applications. this unit was used to stream over a wireless channel to another nRF52 MCU, emulating a central host MCU, the data collected from the DVS. Figure 2.9 shows a block diagram of the DVSI-based sensor node, connected to the host nRF52 MCU, as well as the high-level block diagram of the circuit implemented on the FPGA.

Sensor node operation

The operating principle of the DVSI-based sensor node is the following: *i)* the MCU request an event frame; this operation must be explicitly triggered by raising `frame_req` signal. In our case, the frame read was triggered by one of the timers available on the MCU; *ii)* the DVSI receives the frame request and triggers an event frame acquisition on the DVSI. This operation repeats until enough data have been read from the DVS and an interrupt is issued to the MCU. *iii)* the MCU receives the interrupt, configures the slave QSPI to receive a transaction, then the data are transferred from the DVSI to the MCU. The DVSI-based sensor node behaves as a master frame-based camera sensor; the only difference is that the event frames are not continuously streamed. Instead, the data transfers happen only when enough data have been received from the DVS; multiple event frames can be stored in the frame buffer, depending on the number of active event pixels when the event frame read was triggered.

2.3.5 System-on-chip integration

We presented two event-driven peripherals used to collect audio and video data. This section describes how the two event-driven peripherals presented so far can be integrated into a SoC. Before entering into the implementation details, let us review how input and output peripherals are integrated into a SoA microcontroller-class system. for this example, we take the system presented in [12,15]. All the output peripherals are part of the so-called `uDMA`. This module can be seen as an autonomous subsystem equipped with two master memory ports capable of streaming data from the peripherals to the memory and vice versa. This module is programmed via a register interface, accessible through a system bus. The advantage of using this approach is that during data transfers from the peripherals to the memory, all the parts of the system that are not involved in this operation can be put in a low-power state.

The `uDMA` can host two types of peripherals: *i)* those who do not provide an explicit address where to store the data in the memory. In this case, the `uDMA` controller (uDMA core) generates the memory addresses and stores the data linearly; this is the case, for example,

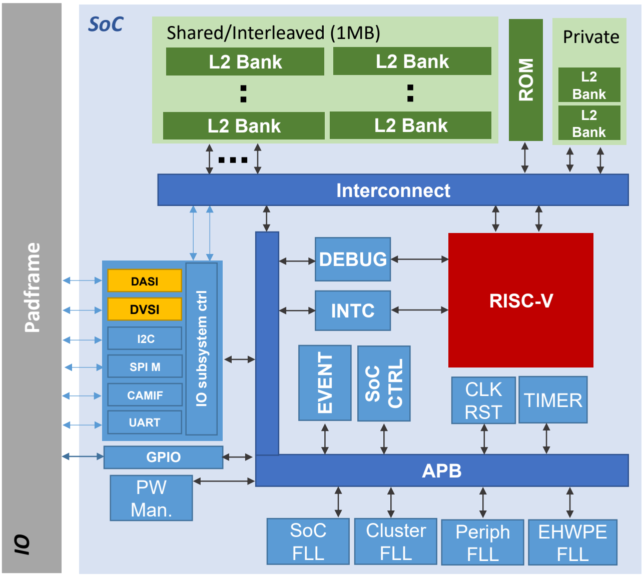


Figure 2.10: Block diagram representation of an SoC architecture where both the DVSI and the DASI are integrated as conventional IO peripherals.

for peripherals like QSPI or Inter-Integrated Circuit (I^2C); these class of peripheral is attached to the so-called *linear channels*. *ii)* the peripherals that can provide an explicit address where to store the data. In this case, this address is forwarded to the `uDMA` controller and used as a memory store address; this class of peripherals is connected to the so-called *external channels*. Since both peripherals already provide the address generation capability, we connected both peripherals to the external channels. Therefore the memory address is generated by the peripheral itself and not by the `uDMA` controller. Figure 2.10 shows a block diagram of the `uDMA` where both the DVSI and the DASI peripherals have been integrated.

2.4 Experimental Results

To demonstrate the effectiveness of the sensor interfaces described in section 2.3, as a first study, we deployed and validated both designs on low-power FPGAs, and we connected each FPGA to a downstream MCU over a standardized protocol to form a standalone sensor node capable of acquiring audio and video sensor data and transfer them to a downstream processing unit. Specifically, in the case of the DVSI, we connected the FPGA to the MCU via a QSPI stream, while in the case of the DASI, we transferred the acquired data from the FPGA to the MCU over a standard I^2S protocol. This preliminary analysis validates the architectural solution described throughout this chapter and serves as a proof of concept to demonstrate that both architectures can be deployed on low-power resource-constrained devices. This exploration also provides insights into the advantages of such sensors, eventually motivating the integration of such sensor interfaces in MCU-class devices IO peripheral subsystems. In the latter part of the analysis, we will show post-synthesis silicon estimates of a sample MCU IO subsystem integrating the DASI peripheral. Then, we will present post-silicon estimates and results measured on an actual silicon implementation of the same IO subsystem integrating the DVSI.

2.4.1 DASI FPGA standalone sensor node results

As audio applications typically have low bandwidth and frequency requirements and operate in an always-on regime, we selected one of the lowest-power FPGAs commercially available. The proposed peripheral has been implemented on an *IGLOOnano AGLN250V2* FPGA device. The FPGA was supplied at the nominal 1.2 V core voltage, and the IOs were supplied at the nominal 3.3 V. The DASI design was synthesized using Synopsys Synplify Pro J2015.03M for logic synthesis and Microsemi Libero SoC 11.7 for FPGA placement & routing. The interface utilizes 31% of the resources available (~ 600 equivalent logic gates). We constrained the design to work with a 30 MHz clock reference frequency generated by a ring oscillator synthesized on the FPGA. The sampling circuitry requires two clock cycles to synchronize and acquire a request from the sensor. Therefore, the highest frequency available for sampling the input audio event is 15 MHz. Thereby, an inter-spike time of 130 ns or more can be sensed by the interface; this value is sufficient to respect the most commonly used standard for the AER protocol, CAVIAR [88], which is used by the silicon cochlea connected to the FPGA. Each audio event transfer must be completed within a 700 ns time window.

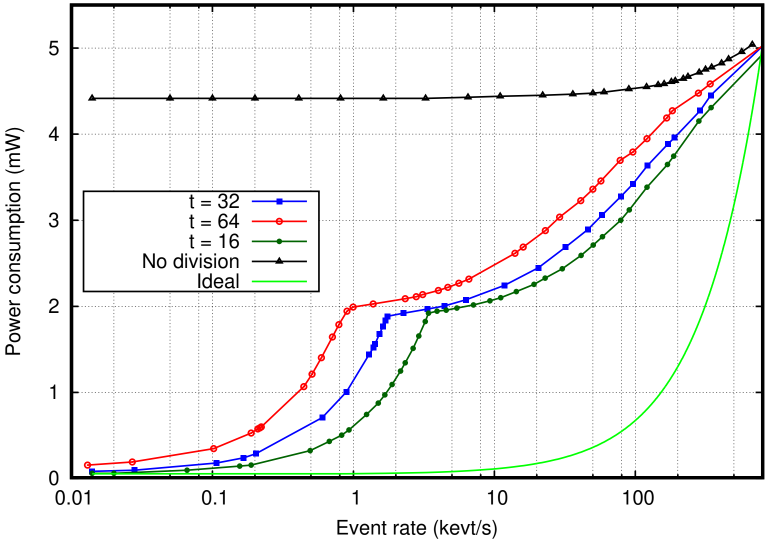


Figure 2.11: Power consumption of the DASI-based sensor node versus the input event rate. Θ_{div} represents the number of cycles to wait before dividing the operating frequency by half. The ideal curve represents the ideal proportionality between the power consumption and the input event rate.

Figure 2.11 reports the total power consumption of the DASI-based sensor node FPGA. To measure the power consumption at a fixed rate, we fed the FPGA with a pseudo-random spike generator producing from 10 evt/s to 800 kevt/s. We compared the power consumption with our approach with a “naïve” constant frequency sampling approach utilizing the same ring oscillator synthesized on the FPGA; in both cases, we clock-gated the unused parts of the circuit to highlight the efficiency improvement introduced by the sole frequency division. In the FPGA implementation, we allocated 12 bits to represent the timestamp value holding the time distance between two successive events. We assumed that two events separated by more than the maximum representable time delta could be considered uncorrelated. Therefore, there is no need to track the time between them. In this condition, the time delta, i.e., the timestamp applied to the incoming event, is clipped to the maximum value, and the circuit is clock-gated to save power.

As shown in Figure 2.11, the proposed solution is vastly more efficient than the naïve clocking approach, at all event rates, except for extremely high rates, when they are on par. Let us consider the ideal power consumption of the interface as a linear function of rate r, i.e.,

$$P_{\text{ideal}}(r) = E_{\text{spike}} \cdot r + P_{\text{static}}, \tag{2.1}$$

where P_{static} is the static power consumed by the FPGA (50 µW) and E_{spike} is the ideal dynamic energy per spike, which we estimated as the one in the high-activity region.

We can see from Figure 2.11 that the power consumption gets farther from ideality as the event rate decreases. However, the clock division technique we propose in Section 2.3.1 drastically improves the situation with respect to the baseline technique with no clock division.

Furthermore, when the event rate drops below ~1 kevt/s the clock is often shut down completely, boosting efficiency up to near-ideal power consumption, particularly at event rates lower than 10 to 100kevt/s. When the sensor's activity is low, the ring oscillator switches off often, determining a steeper decrease in power consumption when successive spikes are uncorrelated. Notice that the switching off of the oscillator can be performed without significantly worsening the acquisition time of the next incoming event since the time to recover from the off state is in the order of 100 ns; which is comparable with a single clock period at the max freq. Therefore, with this clocking methodology, we measured a reduction in power consumption up to 55% in the active region.

The maximum time interval the interface can measure depends directly on the value of θ_{div}, i.e., the number of cycles to wait before halving the operating frequency, as well as the bit-width of the counter tracing the time delta between successive spikes. These parameters can be used as different knobs to match the desired accuracy and maximum time interval the interface can cover before entering the clock-gating low-power state. This time can be computed from Figure 2.11 as the inverse of the event rate in the flex point of the power consumption trends.

Time-to-Information extraction accuracy

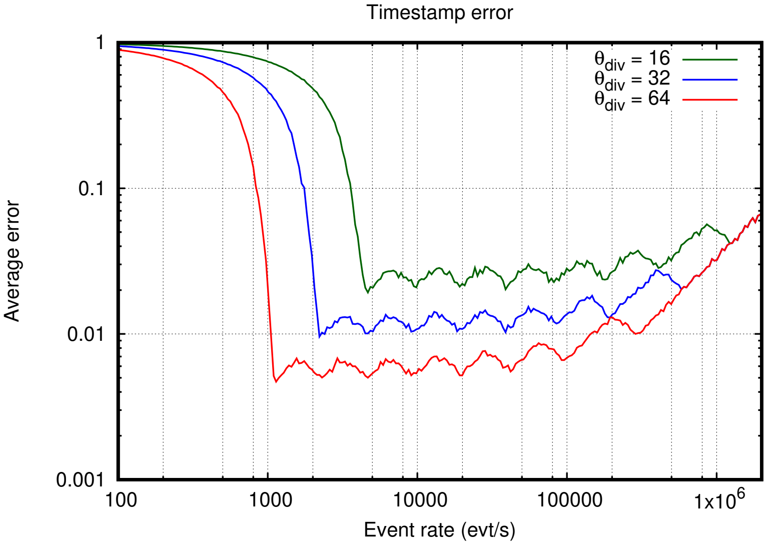


Figure 2.12: Average relative error introduced by the AER-to-AETR conversion.

To evaluate the time accuracy and the error introduced by the time quantization with the adaptive frequency approach, we implemented a Matlab model of the clock generation unit, which can be fed with a configurable event rate Poisson distributed spike stream. This model assumes a perfect clock with constant frequency and a 50% duty cycle. The system has been simulated for different values of θ_{div} and in a range of event rates between 100evt/s and 2Mevt/s. Figure 2.12 shows that in the event rate range of interest (e.g., for $\theta_{div} = 64$, from 1 kevt/s to 550 kevt/s), the average error caused by frequency division can be kept significantly below the analytic 3% bound.

In the graph shown in Figure 2.12, we distinguish three different regions (e.g. for $\theta_{div} = 64$): *inactive region*, from 100 evts/s to 100 kevt/s, corresponding to a very low activity of the sensor; *active region*, from 100 kevts/s to approximately 550 kevt/s, where the divided clock methodology is applied; *high-activity region*, above ~550 kevt/s, where the reference frequency is always the maximum one.

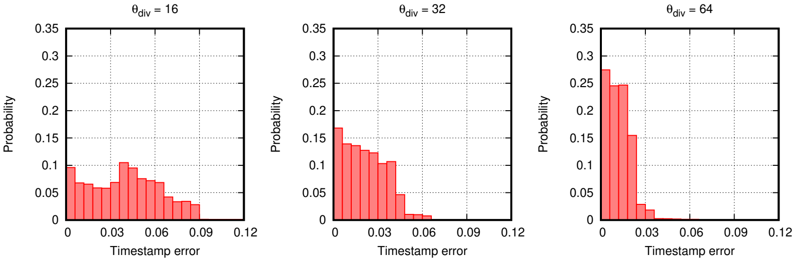


Figure 2.13: Distribution of timestamp errors at different θ_{div}.

In the inactive region, the error is high as the event rate is so low that the interface is essentially always off; therefore, most spike events are tagged with the saturated timestamp: this corresponds to a region in which we are uninterested in the correlation between events. In the high-activity region, the behavior is different: when the event rate is very high, nearing the non-divided sampling frequency, the error increases because an increasing fraction of the spikes are separated by inter-spike times, which are below the Nyquist period, and therefore are tagged incorrectly (this is a limit related to the choice of the non-divided sampling frequency, and not to our frequency division scheme).

In the active region, our main region of interest, the error oscillates between two boundaries; the upper bound is given by a time measurement of the inter-spike time, done just after an iterative frequency division. The lower bound is given when the inter-spike time is measured before a new iterative frequency division. In other words, the peak and valleys in this region's average error are related to the N_{div} successive divisions of the clock. The timestamp error distribution is shown in Figure 2.13

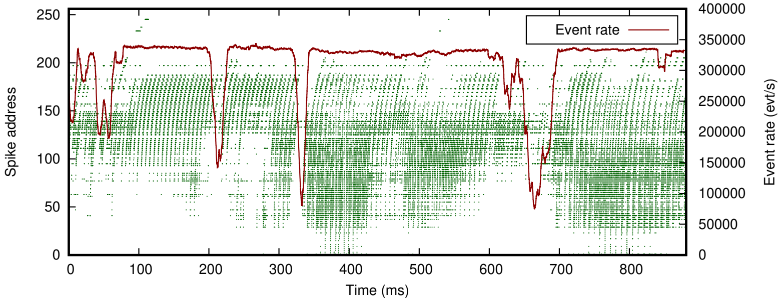


Figure 2.14: Sample phoneme "PULP" acquired with the DASI-based sensor node

Figure 2.14 reports a phoneme acquired with the DASI-based sensor node and transferred to the downstream MCU.

2.4.2 DVSI FPGA standalone sensor node results

This section presents the key results regarding FPGA resources, power consumption, bandwidth, and energy per bit for the DVSI peripheral when implemented as a standalone node. The results here refer to a sample use case, where the sensor node has been used to record data from a DVS in the context of data set acquisition for a gesture classification task.

Table 2.1 reports the FPGA resources utilization. The table shows that the DVSI circuitry occupies a small fraction of the available resources. Specifically, the design occupies only 21% of the available FPGA LUT. The data buffering instead occupy more resources; indeed, 73% of the available memory is used to store the pixels received from the DVSI.

From the static timing analysis (STA), the critical path of the digital circuit implemented on the FPGA is 36 ns. Therefore, the FPGA clock could run at a maximum frequency of 27 MHz. This frequency would be enough to read a number of event frames per second (efps) equal to 2360, considering the maximum amount of data transmitted in a single event frame.

When the whole circuitry implemented on the FPGA is clocked with the same clock source, the main bandwidth limitation through the whole data collection pipeline, i.e., from the DVS through the downstream MCU and eventually to the host MCU connected over Bluetooth, is represented by this latter wireless communication channel.

In the DVSI sensor node we presented here, the FPGA frequency, from which the frequency used to drive the DVSI pixel clocks depends, could be lowered until the input data rate at the DVSI side equals the available bandwidth at the MCU side. Specifically, by accounting for the bandwidth limitation introduced by the wireless channel, we could lower the FPGA clock frequency to 6 MHz. This design choice allows saturating the communication bandwidth at the MCU side and, simultaneously, reducing the power consumption on the FPGA, running the DVSI circuitry at the lowest possible operating frequency.

The maximum end-to-end worst-case sustained event-frame rate that the DVSI-based sensor node can guarantee is 5.2 efps. This value includes the streaming over Bluetooth, and it is obtained when all pixels of an event frame are continuously activated. Each pixel's active events are transmitted through the whole data acquisition pipeline to the host MCU.

Figure 2.15 illustrates the worst-case communication bandwidth between each sub-module of the system. Note that each block, i.e.,

Element	Total	Utilized	Util. %
LUT	5280	1154	21
Flip-Flop	5280	441	8
DSP	8	0	0
IOs	39	38	97
EBR RAM	30	22	73
SPRAM	4	0	0

Table 2.1: FPGA Resource utilization.

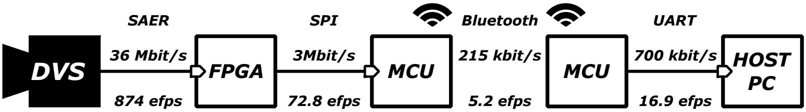


Figure 2.15: Minimum sustained communication bandwidth among system modules in the worst-case operating scenario.

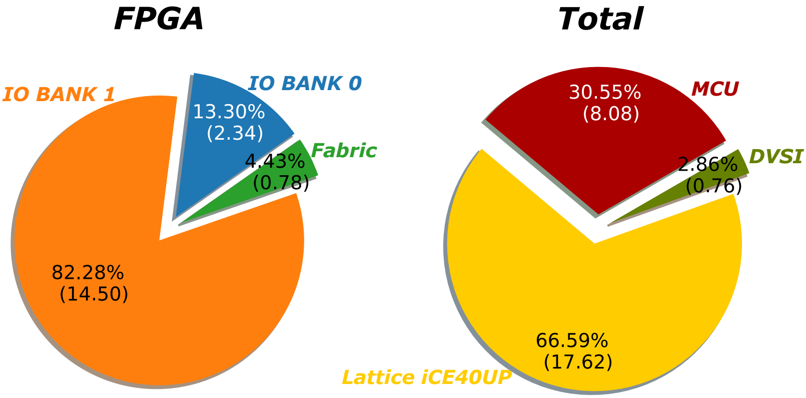


Figure 2.16: System power consumption breakdown. On the left is the power consumption of the Lattice iCE40UP. On the right, the FlyDVS sensor node's total power consumption, including wireless transmission at 200 efps.

each group of 4 pixels, can transmit up to 8 events. However, this represents the worst case, as all the event pixels would be active simultaneously. From the figure, it is evident that the Bluetooth communication channel introduces the main theoretical bandwidth limitation.

The DVSI implemented on the FPGA alone can sustain a worst-case 874 efps from the DVS, which is more than one order of magnitude higher than what conventional cameras can provide in a comparable power budget, enabling on-FPGA event processing when the application needs high responsiveness to the scene change. If we consider the bandwidth limitation introduced by the QSPI, the worst-case efps decreases to 72.8 efps, which is comparable with what is provided by commercial frame-based cameras. The bottleneck of our system is the Bluetooth communication channel, which limits the end-to-end event-frame transmission rate. In the worst-case scenario, the system can still sustain 5.2 efps end-to-end transmission.

In our experiments, we measured the power consumption of the FPGA at 6 MHz, reporting 17.62 mW in the presence of high activity of the DVS, and when the QSPI is transmitting 200 efps. The power consumption of the FPGA decreases to 14.5 mW when no QSPI data transmission is happening, i.e., we are not transmitting event-frames over the QSPI. However, we are still driving the DVSI and receiving event pixels. Note that this power is also including level-shifters to convert the signals from 3.3 V, that is, the nominal voltage to supply the FPGA IO banks, to 1.2 V, that is, the nominal voltage of the DVS IO pads. The system's total power consumption is reported in Figure2.16.

2.4.3 Silicon implementation results

As anticipated in the introduction of this chapter, the last part of this exploration will focus on the silicon implementation of both peripherals. This section aims to show the advantages in terms of power consumption reduction and bandwidth when such sensors are integrated into a MCU-class devices. Therefore, we will report a power performance analysis (PPA) of the IO peripherals subsystem architecture presented before. Both peripherals have been implemented in an advanced silicon technology node. Specifically, in our experiments, we synthesized the peripherals with *Synopsys Design Compiler 2020.09*, in *GlobalFoundries* 22nm FDX process. Specifically, we used 8T, 20, 24, 28, L, and SL voltage threshold cells, SSG corner, 0.72V nominal supply voltage, -40C, 250MHz target clock frequency. Power consumption estimates have been performed at target 250MHz clock frequency, TT corner, 0.8V supply voltage, 25C, using *Synopsys Prime-Power 2019.12.*

DVSI results

This section will show the results for the DVSI peripherals. Specifically, present post-synthesis power estimates for the DVSI peripheral.

Compared to frame-based cameras, the DVSI shows a variable event-frame rate (e-FPS) which depends on the number of active pixels in each event frame. At the maximum target frequency of 250MHz, the DVSI can read a maximum of 9100 event-frames per second (e-FPS) when the activity of each event-frame is 100%; this represents the worst-case scenario. If the frame activity is low, the DVSI reads the frame at the maximum speed and then enters a clock-gated mode, where the power consumption is constituted by only leakage. Figure 2.17 shows the power consumption of the DVSI versus the activity of each event frame.

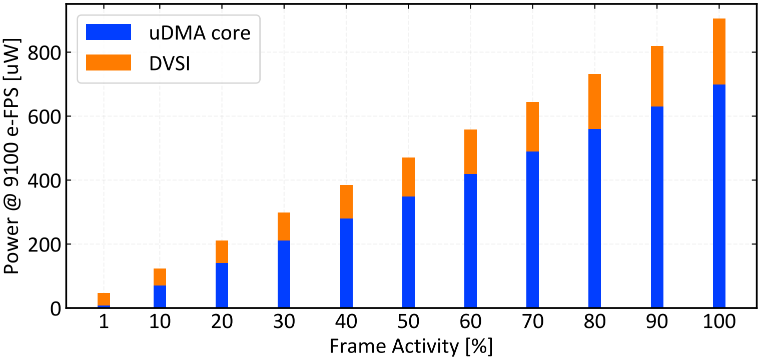


Figure 2.17: Power consumption of the DVSI peripheral versus event-frame activity

As a direct consequence of the result presented in Figure 2.17, it can be noted that the variable event-frame-rate determines the total energy spent by the DVSI to acquire an event frame when the DVSI is requesting frames to the DVS at a fixed rate. Figure 2.18 shows the energy spent to acquire a frame, normalized by the number of events active in a single frame.

As expected, the plot in Figure 2.18 shows that the higher the frame activity, the lower the energy spent to acquire a single event; because an increasing fraction of the fixed frame acquisition period is used to perform useful operations, i.e., acquiring events. On the

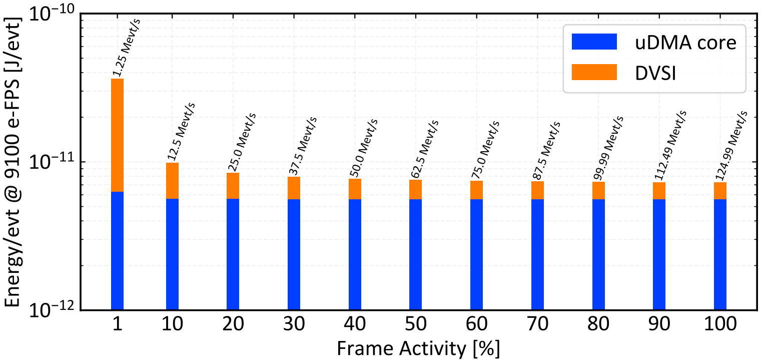


Figure 2.18: Event energy consumption versus event frame pixel activity

contrary, when the frame activity is low, most of the energy spent in a frame acquisition period is constituted by leakage energy. Only a tiny fraction of the period is occupied by the event acquisition. Since it is impossible to know a priori how many pixels will be active in an event frame, the readout frequency can not be lowered to reduce power consumption. Otherwise, the desired frame rate acquisition can not be guaranteed in case of high pixel activity.

DASI results

In this section, we show the results related to the DASI. From the plot in Figure 2.19, it can be noted that the power consumption of the DASI is proportional to the input event activity. The clock relaxation methodology proposed in section 2.3.1 allows mitigating the effect of the high-frequency operation needed to ensure no event loss. Indeed, the DASI is a slave asynchronous interface. Therefore, it is impossible to know when and how many events have to be acquired by the interface in advance. The DASI implementation proposed in this chapter shows a power consumption that decreases by approximately one order of magnitude when the activity decreases from 50 Mevt/s to a few kHz. This result is achieved by adopting a carefully crafted clock gating scheme on the sub-modules of the DASI.

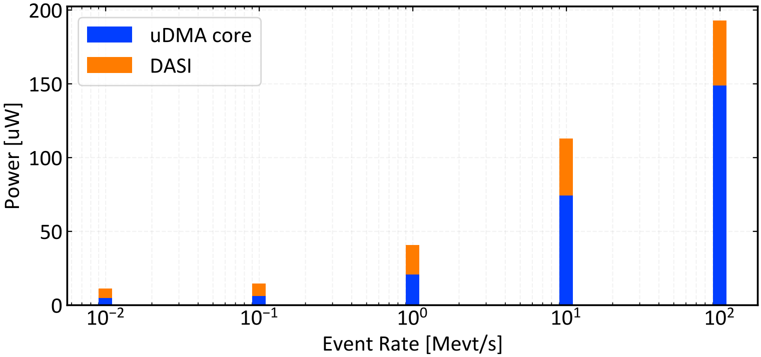


Figure 2.19: Power consumption of the DASI peripheral versus the event rate of the sensor

It is important to note that the average event rate of a DASI peripheral can be as low as a few tens of kevt/s during human speech recording, making it a suitable candidate to be employed in ultra-low power smart sensor nodes.

Similar to what has been done for the DVSI, we analyze the energy per event consumption here. Figure 2.20 shows the energy consumed to acquire a single audio event from a silicon cochlea. Also, in this case, the energy per event decreases as the event rate activity of the sensor increases. This result is expected because the operating frequency of the DASI peripheral has to be high enough to guarantee no event loss. Thanks to the aforementioned adaptive clock regime, the power consumption reduction at a low event rate counterbalance this trend; in the case of the DASI, the activity rate span five orders of magnitude, while the energy consumed per event changes by approximately two orders of magnitude.

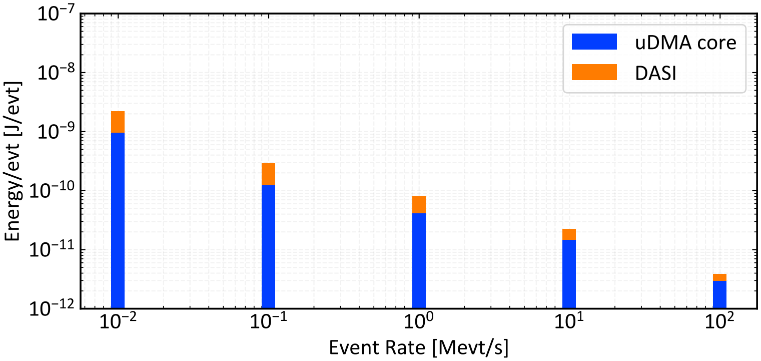


Figure 2.20: Energy consumption per input event of the DASI peripheral versus the sensor event-rate

2.5 Conclusion

This chapter opened the discussion on event-driven data treatment that will develop throughout this thesis by proposing two IO peripheral architectures capable of collecting external data generated by connected sensors in an efficient and energy-proportional way. Energy proportionality is an aspect that becomes crucial when a device is connected to a sensor that is already capable of producing relevant information in a non-redundant way. Therefore, Preserving such proportionality during the data acquisition pipeline becomes crucial when small batteries supply such devices, making effective use of the available energy becomes relevant. The main contributions and results presented in this chapter can be summarized as follows:

- We have observed that event-based sensors, like silicon retinas (DVSI) and silicon cochleas (DASI), pose new challenges in terms of data transfer toward a digital computing device. Despite the significant benefits that come from the lower bandwidth which characterizes this type of sensor, the traffic towards event-based sensing devices is highly irregular and, therefore, complicated to predict. The data retrieval becomes even more challenging when the interface exposed by the sensor is asynchronous. For these reasons, IO peripherals interacting with such sensors need to implement a specific architectural solution that keeps the energy consumption proportional to the sensor data rate.

- Such circuital solutions are effective if the receiving device can directly orchestrate the data reading from the sensor, e.g., in the case of the DVSI. When this is not the case, as we have seen for the DASI, the IO peripheral needs to collect a continuous asynchronous event stream; in this scenario, it is possible to mitigate the energy spent for the data acquisition by dynamically adjusting the frequency of the receiving device in a very fine-grained way, proportionally to the data rate. Specifically, we have seen how the adaptive clock relaxation approach ensures a fast reaction to short event bursts but, at the same time, significantly reduces energy consumption in the absence of input events.

- We have proven on two standalone FPGA implementations that the proposed circuital solutions could effectively exploit the nature of event-based sensors to perform energy-efficient information transfer, thereby demonstrating a complete and effective event acquisition pipeline that can be integrated into a conventional digital microcontroller subsystem.

- In the case of the DVSI, we could demonstrate on a low power “Lattice” FPGA how such peripheral can achieve a high event-frame acquisition rate of 874 $\mathrm{event-framepersecond}$ while consuming only 17.62 mW of power. The whole sensor node consumed 35.5 mW, including the wireless event-frame streaming at 200 efps.

- In the case of the DASI, we could show how the power consumption for time-to-information extraction scales from 4.5 mW at a 550kevt/s rate down to slightly more than 50 µW at rates lower than 10evt/s (a 90× factor) while a naïve constant clock methodology is stuck to the same 4.5 mW power regardless of the event rate.

- In the second part of the results, we have continued the discussion by showing how the proposed IO peripherals can be efficiently integrated into a digital microcontroller autonomous IO subsystem. In this way, we have proven with silicon estimates performed into an advanced technology node that an event-driven energy-proportional data acquisition peripheral is amenable to modern SoCs and that the semantically rich information streams produced by event-based sensors can be easily combined with synchronous digital processing devices.
- In such exploration, we analyzed the energy consumption of both peripherals versus the input event rate. In the case of the DVSI, we observed that the energy spent to acquire an event stays within the same order of magnitude when the sensor event-rate changes from 1% to 100% active pixels. In the case of the DASI, we demonstrate how the peripheral can practically reduce the power consumption by two orders of magnitude when the rate becomes very low (few kevt/s).

This section concludes the analysis we started in the first chapter of this thesis. The discussion carried on throughout the following chapters will start from these findings and build on the ideas presented here.

Chapter 3

Overcoming Reliability Boundaries

3.1 Introduction

In the previous chapters, we have seen how the architecture of modern SoCs can be improved to enable event-driven data processing and interaction with low-power event-driven sensors. As we have seen throughout this thesis, the latest advances in the IOT are changing the nature of edge-computing devices. End nodes have to support, in place, an increasing range of functionality, for example, video and audio sensory data processing and complex systems control strategies. These new capabilities will enable applications such as an entirely new class of biomedical devices [89], autonomous insect-sized drones [90], and cheap smart sensors [91] to continuously monitor the status of bridges, tunnels, and other infrastructures. Machine learning algorithms and DNNs have demonstrated outstanding performance in performing highly complex cognitive tasks. However, deploying such compute-intensive algorithms on battery-powered IoT end-node platforms, characterized by heavily constrained power budgets (typically 100 µW to 100 mW), still constitutes a considerable challenge, as they are expected to be deployed into the operating environment, and process information for time intervals in the orders of few months to several years or even decades without further human intervention. As such, recent research efforts from both industry and academia have focused on enabling the deployment of deep inference on devices operating in the 10 mW to 100 mW power range [30, 34, 92–97].

Duty cycling is a common approach to reducing the average power consumption, widely used in commercial microcontrollers and IoT end nodes. According to this paradigm, the system stays in a deep-sleep, low-power state for most of the time, featuring a power consumption in the range of 100 nW to 10 µW, and wakes up to perform the acquisition and classification task (e.g., with a CNNs) only when explicitly triggered by an external event. Various strategies can be implemented to "wake up" the system and trigger the cognitive task execution. A simpler strategy to determine when the system has to leave the deep-sleep state is to use a time-based triggering mechanism. The applicability of this approach depends on the energy cost of the cognitive task; for example, the computing cost of an accurate CNNs is so high that the active energy becomes rapidly dominant even at low duty cycling rates. As a result, this strategy is highly inefficient whenever a fast reaction time is required at the sensor edge, as it would force a too frequent system activation. A popular strategy for dealing with all those use cases where the cognitive task is very energy hungry is to use the result of lightweight always-on processing as a triggering mechanism. However, this approach requires trigger fine-tuning to reduce the number of false-positive activation, which could be fairly challenging to achieve in a real scenario and would have poor generalization capabilities when the same system is used in other contexts. Alternatively,

Research on semiconductor devices has shown that a common technique to reduce the active power of digital circuits is to operate them near-threshold [23], i.e., by applying a supply voltage slightly higher than the transistor threshold voltage. Scaling voltage and frequency significantly improve digital circuits' energy efficiency by exploiting the quadratic dependency of dynamic power with supply voltage. However, too aggressive voltage scaling significantly impacts the operating frequency of the logic and the reliability of the memory elements of the system, especially those based on SRAMs. To mitigate the performance degradation caused by maximum frequency reduction, modern SoCs can be equipped with a very energy-efficient hardware accelerator [32]. However, the SRAM reliability issue at low voltages remains an unsolved problem, preventing aggressive voltage scaling on all those devices featuring a relatively large on-chip memory needed to execute complex algorithms based on DNNs [95, 98]. A very effective approach is to replace 6T-SRAMs with more resilient, custom solutions such as SRAMs composed of 8T or 10T bitcells supported by reading and writing assist circuits [99,100]. Additionally, among the approaches adopted to improve the resiliency of memory elements at low voltage, usage of standard cell memories (SCM) is particularly convenient since such memory elements are built on top of standard library cells such as flip-flop or latches, which are much more resilient than SRAMs when operating close to the threshold voltage of transistors [101, 102].

In the last years, BNNs [37, 103, 104] showed remarkable performance in terms of both accuracy and memory footprint when executing complex classification tasks. As the gap that separates them from the state-of-the-art fixed point or floating point CNNs narrowed, new and more complex cognitive tasks could be deployed on resource-constrained embedded devices operating at the very edge. BNN implementations show several computational advantages over fixed or floating-point CNN implementations. While the latter relies on convolutions on multiply and accumulate (MAC) operations performed on complex hardware primitives, BNNs are characterized by a very lightweight hardware implementation of the data path. Indeed, when we reduce the multiplication operation to a single bit, this can be performed by a simple logic element such as an XNOR gate. In terms of hardware complexity, BNNs are very convenient, as they do not require multipliers but only a minimal amount of area-hungry adders for partial sum accumulation. Moreover, as opposed to CNNs, BNNs consume and produce single-bit input and output feature maps, reducing the required bandwidth requirements on the interconnection subsystem that allows for communication with the on-chip and off-chip memories, as well as the overall energy consumption of memory load and store operation. These features make BNNs a good candidate for operating scenarios where power consumption is a significant concern. However, at the same time, high responsiveness to the sensor stimuli needs to be ensured (e.g., pico-sized autonomous navigation robots or surveillance nodes). Thanks to their ability to execute reasonably complex cognitive tasks in extremely low-power envelopes, BNNs can be employed in always-on scenarios [105], where the simple triggering mechanisms mentioned before would fail. In this scenario, BNNs would be a suitable algorithm to execute as the first stage of an increasingly more capable staged inference pipeline [106]. As the memory footprint of BNNs is significantly lower than CNNs, larger topologies can be supported in the same power/performance budget, enhancing the generalization capability of the early stages and thereby lowering the false positive triggering occurrence.

A significant advantage of DNNs-like algorithms, specifically BNNs, is the high robustness to noise injected into the input feature maps [107]. An essential aspect of BNNs is that no bit in their activations and weights is inherently more significant than any other. Conversely, no bit is inherently more vulnerable than any other: information processing is spread equally among all bits, and only a very high error rate can bring a dramatic loss in quality-of-results [49]. This chapter will demonstrate how to leverage this intuition to enable a very aggressive energy consumption reduction when an SoC is executing such highly error-resilient algorithms. Moreover, we will demonstrate how an SoC can still operate reliably in such a highly scaled voltage regime by providing architectural guidelines on protecting crucial sub-modules of the SoC from random error occurrence.

The main contribution of this chapter could be summarized as follows:

- We propose a strategy to execute noisy BNNs at ultra-low voltage on a SoC built around a RISC-V core and equipped with a dedicated BNN hardware accelerator.

- We describe and demonstrate a hybrid memory architecture on silicon composed of big SRAMs for error-resilient data and smaller (Standard Cell-based Memories) SCMs to hold vulnerable data such as microcontroller instructions and stacks. This work also provides a methodology to exploit such memory architecture efficiently.

- We present a self-test strategy for Bit Error Rate measurement performed on large SRAM. This approach characterizes SRAM memories at ultra-low voltages, estimating the amount of noise injected into the executed algorithm.

- We demonstrate the validity of this architectural concept on an advanced prototype manufactured in GlobalFoundries 22nm FDX technology, using the safe SCMs to hold a microcontroller program testing SRAM bit error rates with millions of random reads/writes, operating down to 420 mV (50% of the nominal supply voltage) for both logic and memories.

3.2 Related work

In this section, we will review the type of devices and the algorithmic approaches that researchers of machine learning communities have developed to deploy sophisticated artificial intelligence on edge-computing nodes. In this context, the fast-growing *TinyML* research community [108] has introduced significant contributions in the direction of shrinking DNN topologies [109], reducing the amount [110] and numerical precision of network parameters [111], moving from floating-point down to highly quantized numerical representations, e.g., 8 or 4 bits, and ultimately to BNNs [37]. All those improvements have provided a solid theoretical ground to build on for developing lower-power and more energy-efficient hardware computing platforms. Indeed, edge computing platforms have specialized to efficiently accelerate such types of machine learning workloads [98, 112]. In this section, we start our review by describing the newest software-programmable architectures targeting the IoT's end-nodes, going through specialized heterogeneous and error-resilient hardware architectures, and ending with dedicated architectures for CNN inference exploiting extreme quantization and error resiliency.

3.2.1 IoT End-Node Architectures

A desirable feature that ensures high versatility to an IoT end-node is the capability to execute general-purpose operations, i.e., programmable code. Almost all MCU vendors, such as TI [8], STMicroelectronics [7], NXP [6], and Ambiq [5] have commercialized IoT end-nodes based on ARM Cortex-M class processors. Among the various features proposed by such systems, the most appealing for the IoT application domain is the aggressive sleep-walking capability enabled by the sub-10 µW deep-sleep modes, leading to an extremely small average power consumption. On top of this, research communities have demonstrated how the operation in the near-threshold voltage regime further improves the energy efficiency and reduces power consumption during the active computation phase [14, 113–116]. An efficient IoT processor architecture, Mr. Wolf, has been demonstrated by Pullini et al. [13]. The IC comprises an always-on autonomous I/O subsystem coupled with a parallel accelerator with eight floating-point capable RISC-V cores. In [13], it is shown how to combine aggressive deep-sleep capabilities with an energy-proportional architecture, ultimately exceeding the computational capabilities of ULP microcontrollers by two orders of magnitude while offering a competitive energy efficiency also at low and sporadic workloads.

A valid approach is to target specific computation domains such as CNNs. In this scenario, some commercial architectures give up some flexibility to leverage lightweight SW acceleration and optimized DSP libraries to improve performance. A well-known example is CMSIS developed by ARM, a set of libraries to optimize DSP applications on Cortex-M architectures, and CMSIS-NN [117], tuned to the deployment of embedded neural networks. An extension to these libraries has been proposed by Rusci et al. [118] targeting highly quantized networks such as 4-bit, 2-bit, and binarized networks [119]. The main drawback of the approaches based on SW acceleration is represented by the 32-bit parallelism of the IoT nodes where such optimized software libraries are executed. Indeed, while the memory footprint significantly reduces in aggressively quantized embedded deployment, several additional operations are required to pack/unpack activation and weights to arithmetic formats suitable for software processing (e.g., 16-bit or 8-bit) [118], degrading the overall performance. A viable approach is to introduce dedicated ISA extensions to perform sub-word, sub-byte, and SIMD operations efficiently [120, 121], and mitigate such performance degradation.

More specialized approaches have been proposed over the last few years. IoT end node architecture has evolved towards more heterogeneous SoCs, often integrating dedicated hardware processing units to accelerate certain classes of workloads. For example, Intel presented an IoT edge node integrating an x86 processor accelerated by dedicated functional units for CNN cryptography workloads [122]. Conti et al. proposed Fulmine [112], a heterogeneous SoC coupling four general-purpose processors with a convolutional accelerator. While convolutional layers of CNNs run on the accelerator, other functions, such as activations and pooling, execute on the software processing cluster. GAP-8 [123] includes a specialized accelerator for convolutional neural networks supporting 16-bit precision for activations and 16-bit, 8-bit, and 4-bit precision for weights, achieving up to 600 GMAC/s/W within 75 mW of power envelope. Qualcomm Technologies propose a low-power vision sensor node that performs end-to-end always-on visual detection tasks thanks to an ultra-low power QVGA CMOS sensor and a complete digital processor subsystem integrated as a single device [124].

3.2.2 Heterogeneous and Error Resilient Memory Architectures

Memory architecture plays a crucial role in modern IoT nodes operating near-threshold. The power consumption associated with such part of the SoC is typically non-negligible. Therefore, a carefully thought memory architecture can significantly affect the energy efficiency of an IoT node [114, 125, 126]. While many approaches rely on the custom design of low-voltage memories [100, 127], which typically come at the cost of a higher area and power consumption (e.g., 8T or 10T bitcells, read and write assist circuits) [99], an alternative approach relies on approximate SRAMs, often combined with the precision/performance tunability, or in some cases with a heterogeneous memory architecture featuring error-free and error-prone memory regions. A promising approach proposed by Frustaci et al. [128] is to use approximated SRAMs for error-tolerant applications. In this context, energy is saved at the cost of reading/writing errors by exploiting voltage scaling, selective error correction code (SECC), and selective write assist techniques (SNBB). Compared to the voltage scaling at iso-quality, the joint adoption of these techniques can provide more than 2× energy reduction at a negligible area penalty. Other works propose adopting emerging technologies to realize approximate memory cells, such as RRAM [129] and memristors [130].

The main drawback of the abovementioned approaches is that a custom SRAM design is required. In modern technology nodes, SRAM macros can be auto-generated by providing high-level specifications to the memory generators provided with the process development kits (PDK). Therefore, any custom modification to the SRAM circuitry not supported by the toolchain requires a significant additional engineering effort, with a detrimental effect on a potential product time-to-market and total cost. Alternative approaches exploit heterogeneous memory architectures mixing standard SRAMs and latch-based Standard Cell Memories (SCM). While SRAMs can not be considered reliable at relatively low voltages (e.g., below 0.8V or 0.65V in a modern technology node), SCMs can operate in the same operating range as the rest of the logic, which is typically much wider [101]. Tagliavini et al. [131] proposed A memory partitioning-aware HW/SW methodology design energy-efficient ULP systems that can exploit an error-aware allocation strategy. Starting from this intuition, the analysis presented in this chapter leverages standard 6T-SRAM cells memory macros and SCM based on industrially qualified standard cells developed with a conventional digital design flow.

3.2.3 Dedicated Hardware Accelerators for DNNs and BNNs

A very effective approach to maximizing the energy efficiency of specific highly specialized workloads is to execute them on dedicated hardware accelerators. Most such accelerators, especially the ones designed to accelerate machine learning tasks at the extreme edge, employ fixed-point representation, for example, for weights and activations (e.g., Orlando [132] achieving up to 2.9 Top/s/W). As we mentioned before, Binary Neural Networks [37, 103] constitute a particularly interesting niche application due to their properties, as they can be trained to achieve similar results to full-precision counterparts [133] while keeping a smaller footprint, a more scalable structure, and higher resilience to errors. Proof of the effectiveness of BNNs has been provided by *FINN* architecture [134], which can achieve more than 200 Gop/s/W on an FPGA. Reviewing highly specialized architecture implementing BNNs on silicon, remarkable energy efficiencies in the range of 10-50 TOP/s/W have been achieved by *BRein* [135], *XNOR-POP* [136], *Conv-RAM* [93], as well as the BTNN accelerator proposed by Yin et al. [137], the BNN accelerator presented by Wang et al. [138], and Khwa et al. [139]. Similar results have been reported by more “traditional” ASICs such as *UNPU* [34] and *XNORBIN* [140]. Mixed-signal [92], and in-memory mixed-signal approaches [141–144] can achieve up to 10-100× higher efficiency, but paying a very high cost in terms of design time, verification, and scalability to real systems. Yang et al. [107] exploits one such system in their work, where similarly to what we propose, SRAM is aggressively voltage-scaled to achieve a power consumption reduction, with the fundamental difference that the approach proposed in this chapter relies on a fully programmable IoT end node, augmented with a dedicated hardware accelerator [38].

3.3 Architecture

This section will present the Quentin SoC platform we will use as a test bench to demonstrate how to deploy error-resilient applications at aggressively scaled voltage operating corners. The first part of the discussion will describe Quentin's general SoC architecture and memory partitioning. Then we will briefly describe the hardware accelerator where the BNN workload is executed. Eventually, we will provide key insights on the BNN error resilience by evaluating the impact of artificially injected noise on the final accuracy of a sample network topology.

3.3.1 Quentin chip

Quentin is a RISC-V-based microcontroller manufactured in GlobalFoundries 22nm technology node, presented by Schiavone et al. in [125]. The key component of the SoC is a RISC-V processor (RI5CY) [120] optimized for energy-efficient digital signal processing, the core's pipeline features four stages, floating-point and it is fully compliant with the RV32IMFC ISA [145]. The core operates as a “fabric controller” for all the other units of the “pulpissimo” SoC. “The pulpissimo” SoC is publicly available as an open-source open-hardware project[1], part of the Parallel Ultra-Low-Power (PULP) platform[2]. Moreover, Quentin features a complete set of peripherals which include Quad-SPI (QSPI) supporting up to two external devices, I2C, 2x I2S, a parallel camera interface, UART, GPIOs, JTAG, and a DDR HyperBus interface to connect off-chip up to 64 Mbytes of external Dynamic RAM (DRAM) or FLASH memory, and a small ROM used to store the boot-code, as well as an I/O DMA (μDMA [146]) to manage data transfers through peripherals autonomously. The Quentin chip is representative of many IoT nodes implemented in a similar technology node. Therefore, the principles and results demonstrated on such a device also apply to many other IoT edge devices in the same class.

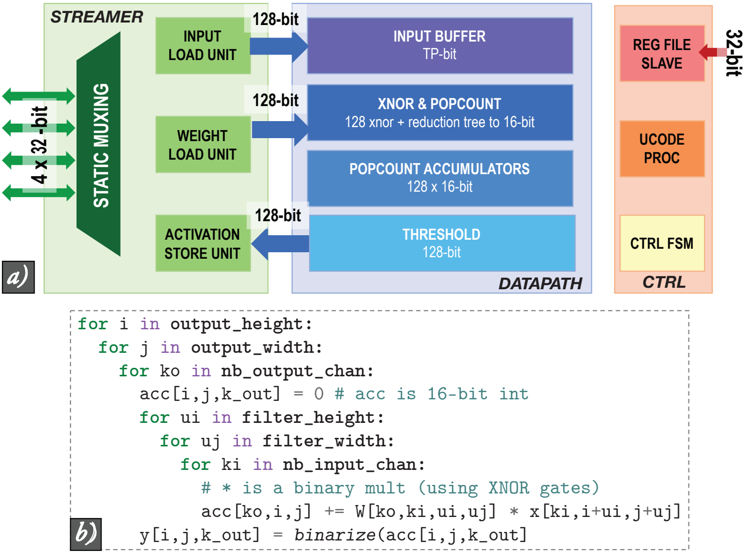
3.3.2 BNN accelerator

Quentin is a heterogeneous architecture device; the SoC is equipped with a dedicated hardware accelerator to execute BNN workloads efficiently. The specific hardware accelerator is called XNOR Neural Engine (XNE), and it has been presented by Conti et al. [38]. Such accelerators often feature high data parallelism towards the memory; XNE is connected to the main interconnect through four memory ports. This design choice optimizes the accelerator data throughput, achieving a sustained data rate of 128 bits per cycle. From a system perspective, the accelerator behaves as an autonomous DMA engine. Specifically, the configuration registers of the accelerator can be accessed for programming by the core through a memory-mapped register interface. Once XNE is programmed, it can autonomously fetch the data from the main data memory (L2) and execute convolutional and linear neural network layers. Figure 3.1 schematizes the internal architecture of the XNE.

---

[1] https://github.com/pulp-platform/pulpissimo

[2] https://pulp-platform.org



```
for i in output_height:
  for j in output_width:
    for ko in nb_output_chan:
      acc[i,j,k_out] = 0 # acc is 16-bit int
      for ui in filter_height:
        for uj in filter_width:
          for ki in nb_input_chan:
            # * is a binary mult (using XNOR gates)
            acc[ko,i,j] += W[ko,ki,ui,uj] * x[ki,i+ui,j+uj]
      y[i,j,k_out] = binarize(acc[i,j,k_out])
```

Figure 3.1: a) XNE internal architecture, showing the *streamer* (green shades), *control* (orange) and *datapath* (blue) submodules; *b)* BNN layer execution pseudo-code highlighting microcoded loops (orange) and datapath execution (blue).

3.3.3 Memory partitioning

As we have introduced in the first section of this chapter, the memory hierarchy can significantly impact the energy efficiency of an IoT end node. The Quentin SoC presents a highly flexible, heterogeneous, error-resilient, and error-prone partitioned memory architecture. Figure 3.2 shows a block diagram representation of the main Quentin L2 memory architecture. The figure also reports the different supply nets at which each memory region is connected and the specific physical implementation flavor, i.e., whether a memory block has been implemented with conventional SRAM macros or implemented with logic standard cells. The various memory regions of Quentin reported in Figure 3.2 serve different purposes. The Interleaved L2 memory region is meant to host data that the RISC-V,

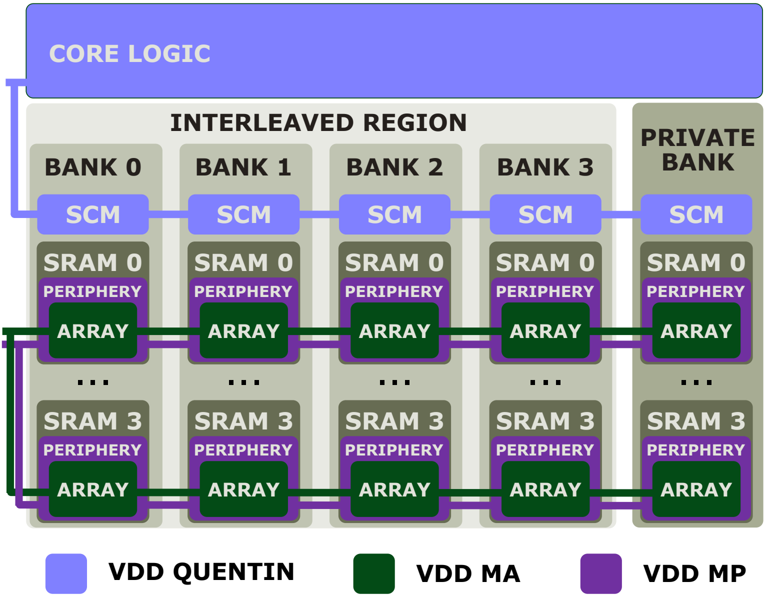


Figure 3.2: Quentin chip memory partitioning

or the BNN accelerator, i.e., XNE core, will process. The Private 64kB memory space is dedicated to the RISC-V core program, stack, and core-private data. From a performance viewpoint, this memory partitioning enables transparent sharing of the L2, increasing by 4x the system memory bandwidth compared to the traditional single-port memory architecture typical of AHB-based MCUs [7], without the usage of power-hungry dual-port memories.

A peculiar feature of the Quentin memory architecture is that the memory banks are internally subdivided into two heterogeneous parts, i.e., they are implemented as a hybrid mix of SRAM and standard-cell-based memory cuts (SCM). Each interleaved bank has 112 kB of SRAM and 2 kB of SCM, while the private banks have 8 KB of SCM as shown in Figure 3.2; the rest is implemented as SRAM macros. The advantage of such a memory partitioning is that the portion of the memory implemented as an SCM-based memory can operate reliably at a much lower operating voltage than an SRAM macro. Moreover, such memories typically feature significantly smaller read and write energy than traditional SRAMs, up to 4X, depending on the configuration (i.e., leakage dominated vs. dynamic dominated) [147]. Such flexibility and higher voltage operating range of the SCM-based memories comes at the cost of a significantly higher memory area. Therefore, finding the right balance between the amount of memory implemented as an SCM and SRAM becomes a crucial optimization known to improve the system-level energy efficiency.

3.3.4 BNN error resilience

As described in this chapter's introductory section, BNN has proven to be an algorithm class relatively robust to noise injected into the input and intermediate features. Yang et al. [107] used a statistical model to quantify the accuracy drop of an application-specific BNN architecture when bit errors occur. In their work, the highest accuracy drop reported by the authors is approximately 5% in the presence of a bit error rate (BER) of 10^{-4}, which is significantly higher than what is commonly reported for SRAM manufactured in advanced technology nodes.

This section presents the results of a similar analysis performed on three different BNNs. To ensure consistency among the results presented for different neural network topologies, we trained three BNNs using the PyTorch 1.0.1 framework. The network we used for our experiments was inspired by those presented by Hubara et al. in [103], and from Yang et al in [107]. We also proposed a size-scaled version of the well-known "VGG" network topology; in the following sections, we will refer to such network topology as "uVGG". The network topology is shown in Figure3.3. Table 3.1 reports the salient characteristics of the three networks.

Similar to what is shown by Yang et al. [107], this experiment aims to evaluate the final BNN classification under various conditions of BER. The "uVGG" network was trained on the CIFAR-10 classification data set, widely considered the smallest one with enough complexity to provide meaningful error results. To model the noise injected into the BNN, we identified the possible noise source when a BNN is executed on the dedicated BNN hardware accelerator. We

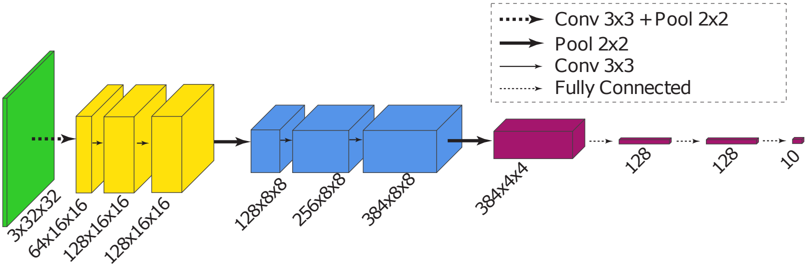


Figure 3.3: *uVGG* BNN topology.

identified three possible sources of errors potentially degrading the final classification accuracy:
i) A bit flipping during the weights reading from the main memory.
ii) A bit flipping during the input features reading.
iii) A bit flipping during the activations storage into the main memory.
In the Quentin SoC, we assume all errors injected into the BNN occur during memory transactions because the logic standard cells can operate reliably at a significantly lower voltage than the SRAM macros. Additionally, no SRAM macros are present inside the XNE accelerator; therefore, we assume that all the operations performed inside the accelerator do not introduce errors. The BNN activations are binarized by comparing the final accumulation value y with a safe 8-bit threshold value τ, stored in the error-free internal register. In our tests, we added uniformly distributed errors to input, weights, and activations of all network layers according to the target BER values to evaluate. The noise was added to the network only at inference time. Therefore no noise injection was experienced by the network at training time. We repeated the same experiments by using two additional network topologies, i.e., the network topology presented by Hubara et al. in [103], and from Yang et al [107], achieving comparable results. Table 3.1 reports the nominal classification accuracy obtained without any bit-error injected noise.

We report, in Figure 3.4, the plot of the BNN classification accuracy for the same networks under different levels of noise (BER) injected into the network. Each result point represents the classification accuracy over the CIFAR-10 test set, averaged for 100

BNN topology	**Nominal accuracy**	**Mem. footprint**
Based on Yang et al. [107]	78.6%[a]	319 kB
Hubara et al. [103],	90.9%	4545 kB
uVGG	85.6%	312 kB

[a] Including both activations and weights.

Table 3.1: Parameters of BNNs used in the resilience experiment.

randomized experiments. We observed that the standard deviation of the results over this sample is always less than 1% of the reported value. Note that the same experiments were repeated with errors occurring without recurring patterns. In this scenario, we did not observe any significant difference in the final classification accuracy with respect to the case where errors were uniformly distributed.

The exploration presented in this section serves as a reference for the analysis shown in the result section, where the BER will be correlated to the voltage scaling. Thanks to the evaluation of several BNN network topology error resilience, it is possible to establish a relationship between the tolerable classification accuracy drop that a specific BER introduces and the energy saving that can be achieved by operating an SoC beyond its reliability boundaries, i.e., significantly below the nominal voltage conditions for its SRAM memories.

3.4 Results

This section presents the error resilience results collected on the Quentin SoC when the supply voltage is scaled significantly below the nominal operating conditions. Such results allow us to directly link the energy saving achieved through the very low voltage operation of the chip to the final classification accuracy of a BNN, which is operating in a bit error-prone framework.

We performed two different evaluations. First, we measured the BER on the SRAM memories caused by operating such memories at

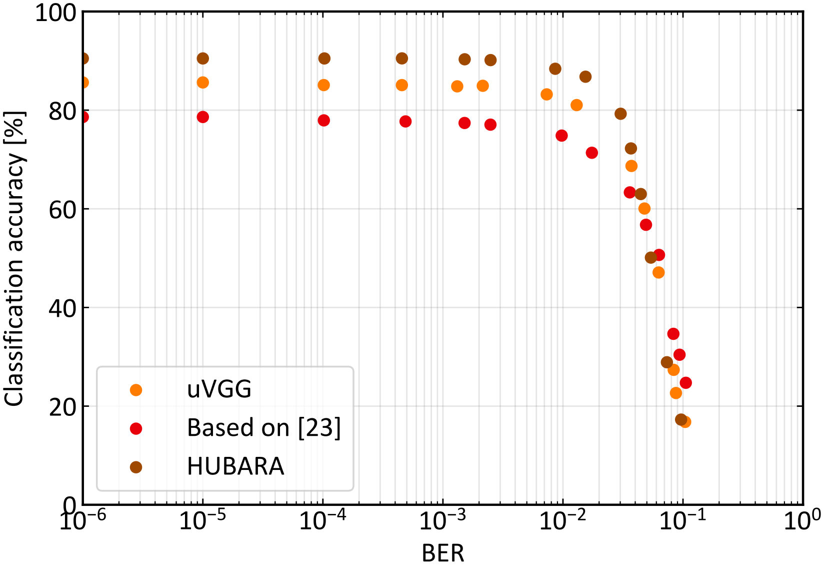


Figure 3.4: Classification accuracy under different levels of bit-error injected noise on input features, weights, and output features.

very low voltages. Then, we measured the current drained by each power domain of the Quentin SoC. Finally, we computed the energy efficiency of the SoC. We evaluated the power saving when the supply voltage of the SRAM is scaled, and the quality of results (i.e., top1 network accuracy) is degraded by less than 1%.

3.4.1 Experimental setup

The results presented in this section have been obtained by testing the Quentin SoC on an Advantest SoC hp93000 integrated circuit testing device. The experimental setup is reported in Figure Figure3.5, the internal tester voltage supply channels have precisely regulated supply voltages; such channels can also precisely measure the current delivered to each power domain with an error lower than 1%.

Testing the BER in a microcontroller featuring several hundreds of kB of system memory, e.g., the Quentin SoC described in this chapter, can require a significant testing time. The main bottleneck in such systems is represented by the relatively low bandwidth provided by the debug interfaces, e.g., the Quentin JTAG test access port (TAP), which requires serializing the data to observe. To detect a single bit-error on the SRAM operating at nominal voltage, we observed that up to 10^9 bits need to be written and successively read from the SRAM memory under test. When the JTAG is operating at 1MHz, which is a reasonable operating frequency for a debug interface, observing a single data point can require a time in the order of several thousand seconds. To overcome this limitation and speed up by approximately 100X the data BER acquisition process, we composed a dedicated software test running on the host RISCY microprocessor. To ensure the correctness of the instructions executed on the core, the microprocessor program was stored in the error-free section of the L2 memory (i.e., the SCM memory banks).

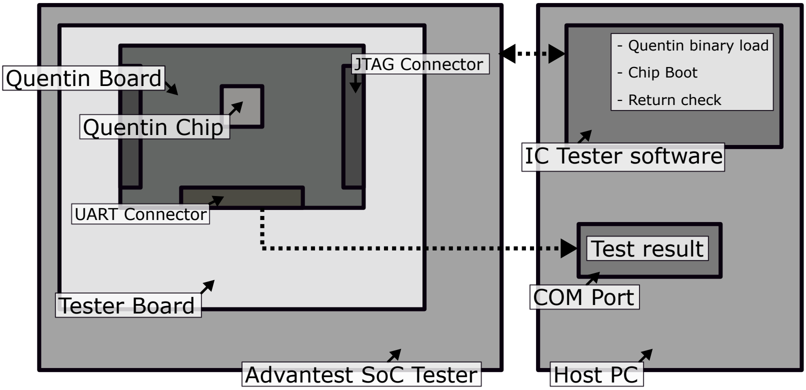


Figure 3.5: High-level block diagram of the experimental setup.

3.4.2 Bit Error Rate analysis

The BER analysis was performed by issuing a writing memory transaction with randomized, uniformly distributed 1 and 0 valued bits to the SRAM memory banks. Since a single data point measurement was repeated approximately 1800 times, data were generated with a software-implemented 32 bits Linear Feedback Shift Register (LFSR). The test application sequentially covers the entire SRAM shared address space. Errors are counted by comparing, bit-wise, the data read at each memory location with the ground-truth value generated by the LFSR generator using the same initial seed. The test is repeated multiple times at each supply voltage point to have a reliable measurement of the BER. Note that this approach could generate artifacts in the error statistics when a memory location is filled in successive iterations with the same test vector. To avoid this problem and make our measurement more robust, the software LFSR uses a different seed to generate test data at each new iteration.

In our tests, we measured only the BER related to SRAM banks. SCM, which is hosted by the same power domain as the circuit logic, was reserved for storing the core instructions of the self-test application and test results (i.e., the number of errors). Note that the storage of the software instruction on an error-free memory space is mandatory for the application to be able to run. In SoCs featuring single-power-domain memory subsystems (i.e., not having the possibility to store core instructions in a separate error-free memory), SRAM errors could also affect core instructions – making aggressive voltage scaling infeasible, as a single corrupted bit on a core instruction could cause errors in the core control flow, making the entire SoC entering unpredictable states, and ultimately the system to fail. For each operating point in our experiments, we performed 1800 on-chip test runs, writing 448kB at each iteration.

Figure 3.6 reports the BER at each SoC operating voltage. By construction, our test could not observe more than $8 * 10^8$ bits. Therefore, the reciprocal of this value represents the lower bound of the on-chip test application, i.e., $1.25 * 10^{-9}$. The results of the BER analysis versus the supply voltage are reported in Figure3.6. When the supply voltage is higher than 0.6 V, no BER is observable by our tests.

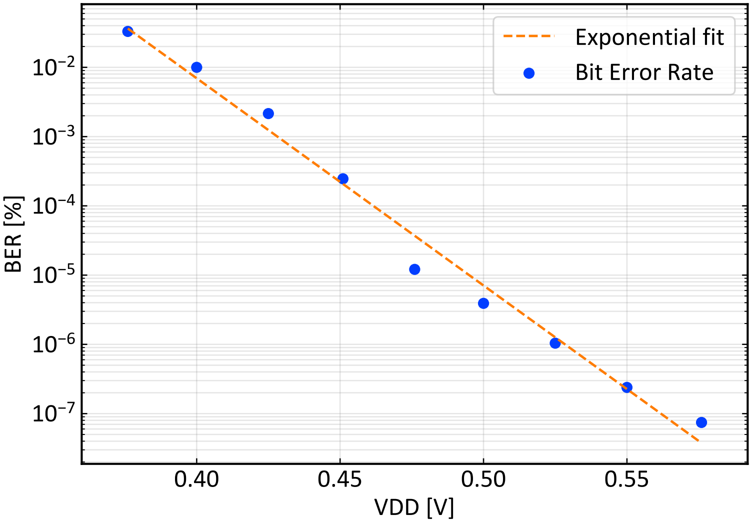


Figure 3.6: Bit Error Rate.

Below a supply voltage of 0.6 V, as expected, we observed a BER increasing with the memory supply voltage decrease, reaching a BER of 10^{-2} at the lowest supply voltage point where the memory was still accessible. The BER measurements confirm that SRAM supply voltage can be scaled at the cost of a higher number of errors. Noise-tolerant applications can be deployed on Quentin SoC; there is enough margin for trading off the amount of noise injected into the data and the potential energy efficiency gain deriving from the voltage scaling.

3.4.3 Power and energy consumption

In this section, we report the power consumption of the Quentin SoC. Moreover, we show the maximum frequency characterization of the Quentin SoC. Both results allow calculating the overall energy efficiency of the system. The system's critical path is in the paths going from the core to the memory system. Power measurements were performed during the execution of a sample test application exercising both the RISC-V core of the system and the XNE hardware accelerator. To precisely control the supply voltages and clock frequencies of the SoC, therefore, to measure the energy consumption of individual SoC power domains with enough accuracy, all the measurements were performed on the Advantest SoC IC tester mentioned in 3.4.1.

The application we used as a benchmark was designed to emulate realistic working conditions, using the XNE accelerator over a randomized pattern of bits. By using a synthetic uniformly distributed and uncorrelated set of binary inputs, we maximized the switching activity both on the XNE input circuitry and the XNOR-based datapath, practically obtaining a worst-case power consumption. The core executes the main application while the XNE accelerates the BNN layer output computation. At the same time, the accelerator is active, the core is in clock-gating, and it wakens up at the end of each XNE job. Instruction code was stored in the error-free SCM, supplied at the same voltage as the core. Data (i.e., binary weights, activation, and partial results) were stored in the error-prone SRAM supplied at the scaled voltage, except for critical 8-bit threshold data stored in the interleaved SCM. Table 3.3 reports more details on how data are stored in memory.

The measurements reported in this section refer to three relevant operating conditions of the SoC. The nominal operating point (namely Nominal) refers to a supply voltage of 0.8 V. This is the operating point for which we performed the Static Timing Analysis (STA). The High Efficiency (HEFF) point is the operating condition at which the chip reported the highest energy efficiency. The Ultra-Low Power

OP mode	$Vdd_{ma/mp/quentin}$	**Freq.**
Nominal	0.8V	565 MHz
HEFF	0.5V	145 MHz
ULP	0.42V	18 MHz

Table 3.2: Supply voltage range of Memory Array (MA), Memory Periphery (MP) and Quentin power domains at Nominal, High Efficiency (HEFF) and Ultra-Low Power (ULP) modes.

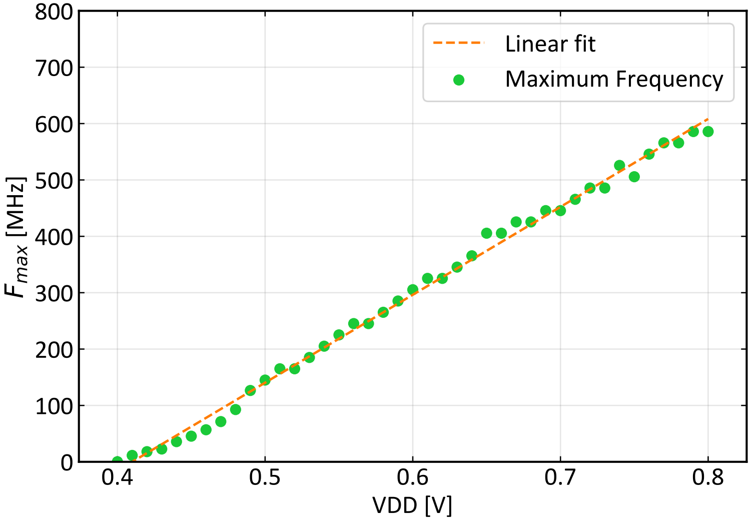


Figure 3.7: SoC maximum operating frequency.

(ULP) point is the operating condition where the chip reported the minimum power consumption. Table 3.2 provides more details about those three points.

Figure 3.8 reports the power consumption breakdown of the Quentin SoC. We measured contributions from the three power domains by observing each power domain's total current drained from the power supply. We performed all power measurements at three different frequencies (fmax, fmax/2, 10 MHz) and used a simple least-squares model to detect static and dynamic power. As shown by the plot in Figure3.8, in ULP mode (i.e., at a supply voltage of 0.42 V), leakage power dominates the dynamic power. In this operating point, the SoC reliably works at its lowest power consumption, 674 µW, yet achieving a sustainable frequency of 18 MHz, and energy efficiency of 6.2 Tops/s/W, which is not lower than the one reported at the nominal operating condition. In ULP mode, the leakage power contributes to approximately 80% of the total power consumption. In this operating

	Weights	**Activation thresholds**	**Input features**	**Output features**	**Instructions and Stack**
SRAM Exec.	*SRAM*	*SCM*	*SRAM*	*SRAM*	*SCM*
SCM Exec.	*SCM*	*SCM*	*SCM*	*SCM*	*SCM*

Table 3.3: Network parameters and core instruction memory storage

point, Quentin can perform 15.4 Inference/s, reaching an energy efficiency of 22.8 Inference/s/mW. Compared to the same workload executed on the embedded RI5CY core, hardware-accelerated execution achieves 21.6× better energy efficiency in the Nominal operating point (167 fJ/op using the XNE, 3.6 pJ/op performing it in software at VDD=0.8V). Whereas the overall power consumption drops when using the core, the performance achieved is significantly lower (6.6 op/cycle) [119]. Note that the software baseline used as a comparison already represents a significant improvement (more than 10X) to the performance reported by leading microcontroller architecture (e.g., arm cortexM4).

Overall, we observed that the most significant leakage contribution originates from the SRAM arrays, working 380 mV below the nominal specifications. In HEFF mode (i.e., at a supply voltage of 0.5 V), the SoC can sustain a clock frequency of 145 MHz, consuming 2.5 mW. We report the highest energy efficiency achieved by the system in this operating point, i.e., 12.7 Tops/s/W (49.2 Inference/s/mW). The leakage represents 32% of the total power consumption in the HEFF operating mode. Figure3.7 reports the SoC maximum frequency used for the energy efficiency computation.

Figure3.9 shows the energy per binary operation when the SoC is executing either from SCMs only or SCMs plus SRAMs; The absolute lowest energy per binary operation is 76 fJ OP, which is achieved at 0.46 V when executing from SCM only. Note that to execute a complete neural network from SCM only is unrealistic since those memories are generally too small because of the low area density. The lowest energy per operation achievable when

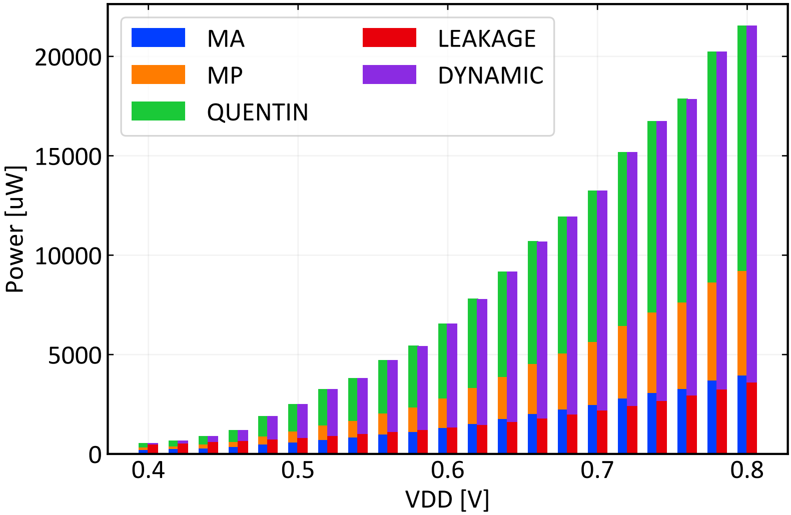


Figure 3.8: Breakdown of independent power contributions when operating from supply voltage scaled SRAM.

executing from SCMs and SRAMs is 78 fJ OP and is reached at 0.5 V. This plot demonstrates that our approach achieves comparable energy per operation on SCMs and SRAMs. The aggressive voltage scaling performed in our experiments, together with a carefully crafted memory partition, significantly improves the energy efficiency when executing from dense SRAMs, ultimately relaxing the memory constraints for error-resilient application deployment. The energy per operation reached at 0.5 V represents an improvement of 2.2X compared to the energy per operation measured at nominal condition 170 fJ.

Table 3.4 compares our work to BNN accelerator implementations that, similarly to our case, exploit BNN error-resilience to maximize energy efficiency. To our best knowledge, this work represents the first complete general-purpose microcontroller architecture capable of

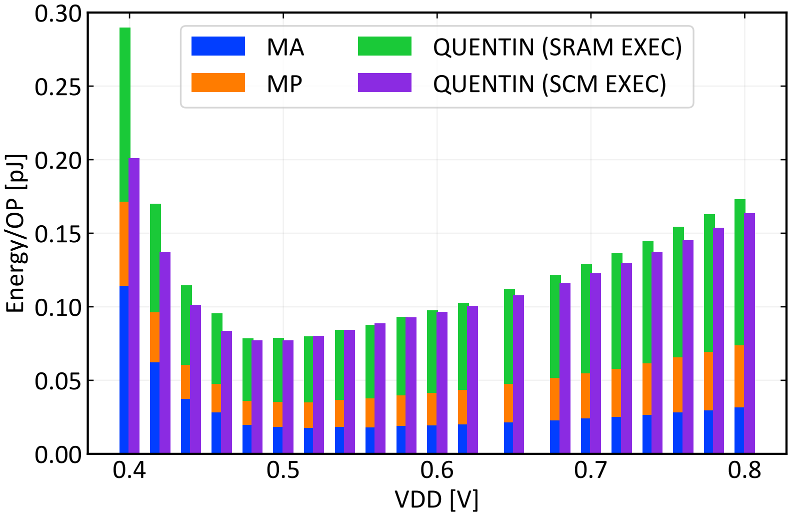


Figure 3.9: SoC energy efficiency comparison when operating with supply voltage scaled SRAM, and when executing from SCM.

exploiting a heterogeneous memory hierarchy to execute error-resilient applications at the highest achievable energy efficiency.

3.4.4 Power accuracy trade-off

From the analysis discussed in the previous section, we concluded that the final classification accuracy could be traded in spite of higher energy efficiency or lower power. Figure3.10 shows the accuracy loss versus the supply voltage on both memories and core logic. Figure3.11 reports the accuracy loss versus the power consumption reduction enabled by the supply voltage scaling.

We did not observe any accuracy loss when the SoC operates in HEFF mode. Thereby, we can conclude that if performance (e.g., in terms of inference per second) is not a critical constraint, the energy efficiency can be improved by 2.2X (from 170 fJ OP to 78 fJ OP)

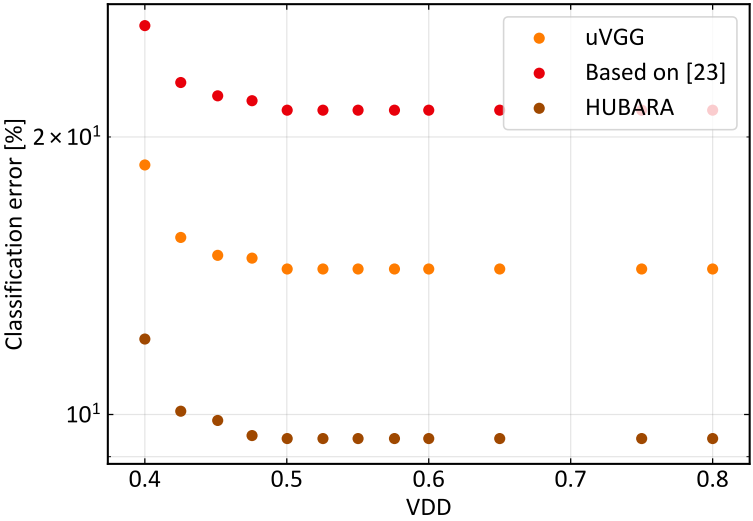


Figure 3.10: SoC supply voltage / Accuracy tradeoff.

without any appreciable penalty on the quality of the result, i.e., the classification accuracy of a BNN.

Furthermore, suppose the application can tolerate a small classification accuracy loss, smaller than 1% in our sample target network topologies analysis. In that case, the power consumption can be further pushed down, reducing it by 3.7X with respect to the HEFF operating mode. In this operating condition, the energy efficiency degrades compared to the HEFF operating point. This is caused by the fact that the total power consumption becomes leakage-dominated as the supply voltage is reduced (Figure3.8). A proportional power reduction does not follow the performance reduction caused by voltage scaling. The ULP mode, where the chip consumes 674 µW, is suitable for always-on operating scenarios or IoT end-node with an expected lifetime in the order of months or years, as well as applications where the peak power dissipation is a critical concern (e.g., implantable devices).

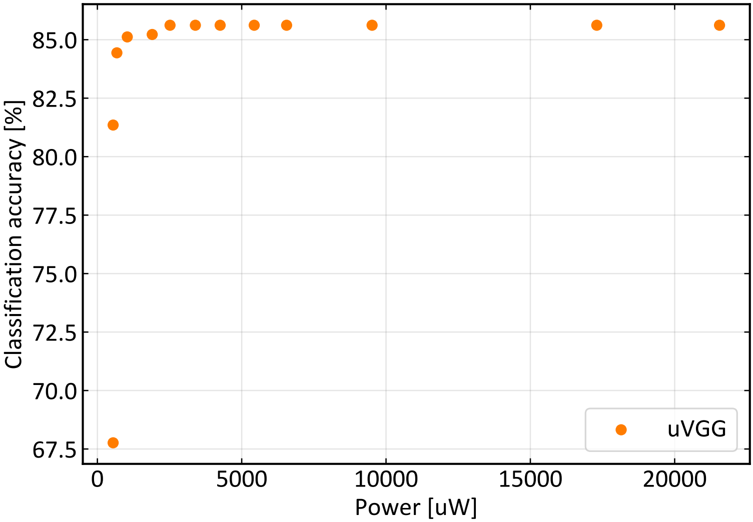


Figure 3.11: Power accuracy tradeoff evaluation for the uVGG BNN topology.

3.5 Conclusion

This chapter presents a methodology to achieve high energy efficiency on edge-computing devices by exploiting the bit-error resilience of cognitive applications like BNNs. Such a strategy relies on a strategic partitioning of the SoC memory into error-free and error-prone regions when the chip is operated under extremely low voltage conditions. The contributions of this chapter can be summarized as follows:

- Our analysis and results demonstrated on actual silicon how to trade-off the energy consumption of an FDX 22nm SoC with the final classification accuracy of Binary Neural Networks, executed on a dedicated hardware accelerator. The proposed approach exploits the intrinsic noise robustness of BNN, i.e., the fact that a significant amount of noise on network parameters, quantified in terms of BER, marginally degrades the final classification accuracy.

- Our measurements show that thanks to a wise L2 memory partitioning, the system can operate reliably at very low voltages (i.e., down to 0.42 V). Therefore, we show that by over scaling the supply voltage of the SRAMs of the SoC significantly below the nominal specifications, the energy per binary operation can be reduced by a factor of 2.2X compared to the nominal supply voltage. In this voltage over-scaled regime, we demonstrate that the reported energy efficiency gain does not affect the end-to-end classification accuracy of the BNN when the voltage is scaled down to 0.5 V.

- We show that if a small penalty on the final classification accuracy is tolerable, e.g., within 1%, the SoC can be operated in an ultra-low power mode, further reducing the overall power consumption (674 µW at 18 MHz, 0.42 V) without exceeding the energy consumption per binary operation shown at nominal operating conditions.

Name	**Technology**	**Core area** $[mm^2]$	**Power** $[mW]$	**Energy eff.** $[TOP/s/W]$	**On-chip memory** $[kByte]$	**Peak perf.** $[GOP/s]$	**Type**	**BNN**
BRein [135]	65nm (Digital)	3.9	600	6	-	1380	DNN ASIC	Configurable
UNPU [34]	65nm (Digital)	16	7.37	51	256	7372	DNN ASIC	Configurable
Bankman et al. [92]	28nm (Mixed signal)	4.84	0.094	772	329	72	DNN ASIC	Fixed topology
BinarEye [148]	28nm (Digital)	1.4	2.2	230	328	90	DNN ASIC	Configurable
Yin et al. [137]	28nm (Digital)	4.8	3.4-20.8	765	224	3270	DNN ASIC	Configurable
This work (0.8V)	22nm (Digital)	2.3	21.6^a	$5.98^a/14^b$	520	129^a	Heterog. SoC core + periph + mem + BNN acc.	SW defined
This work (0.8V) SW Impl.			16^a	0.276^a		3.73^b		
This work (0.5V)			2.52^a	$13^a/23.9^b$		33^a		
This work (0.42V)			0.674^a	$6.2^a/14^b$		4^a		

Table 3.4: Comparison of silicon-proven Application-Specific ICs for Binary Neural Networks. a - full SoC, b - core domain

Chapter 4

Energy-proportional data processing

4.1 Introduction

In the previous chapter, we could observe that A key advantage introduced by event sensors is the proportionality between the primary sensor input and the number of output events generated by it [48]. Specifically, we have seen how to efficiently acquire data from such a class of sensors thanks to two dedicated digital peripheral interface architectures. Such architectures achieve high energy proportionality when receiving data from the sensor, eventually enabling the connection between event sensors and conventional MCU devices. The energy to information proportionality needs to be preserved across the whole processing pipeline to efficiently exploit event-based data streams' inherently sparse nature.

Over the last years, we have seen the tendency to move more and more computation towards the extreme edge. Indeed, many applications are nowadays deployed on embedded IoT nodes operating at the sensor edge. The main challenge of processing data produced by event sensors is represented by the high degree of unstructured sparsity. CPU and GPU class devices can not profit from unstructured data sparsity, as they often target very regular workloads. Similarly, several works have shown how to design deep neural networks (DNN) accelerators operating on sparse data; however, to be able to exploit the data sparsity, they often rely on dedicated architectural features tailored to the specific type of data sparsity [47]. To overcome this limitation, we need a significant paradigm shift in processing time distributed sparse data featuring a high degree of unstructured sparsity.

Neuromorphic algorithms, i.e., algorithms inspired by how the biological nervous system work, are promising candidates to solve the unstructured data processing problem. Such algorithms have been developed to target "asynchronous" time-distributed data streams as a primary algorithmic input. Neuromorphic systems ultimately try to emulate how the information propagates from the peripheral nervous system to a biological brain, where semantically rich information is extracted efficiently.

Among neuromorphic algorithms, Spiking Neural Networks (SNNs) represent the leading model-free algorithmic approach for EVSs data processing [149]. Like many artificial neural networks (ANNs), SNNs rely on elementary computational units, i.e., neurons. Such neurons can be arranged into clusters to form neural network "layers". The interconnection between neurons belonging to different layers and between neurons of the same layer is implemented through synaptic weights. Since SNNs often operate on time-distributed data streams, synaptic connections between neurons can also show a time behavior, i.e., introduce a time delay. Similar to what happens for conventional DNNs, a layer of spiking neurons can be concatenated to form computational networks [150]. A distinctive feature of SNN neurons is the presence of a *neuron internal state*, which evolves over the entire inference process. Spiking neurons, therefore, show a "memory" behavior similar to what can be observed on other neural networks' elementary computational units, e.g., LSTM units [151].

Recent advances in SNNs show that such a class of networks can achieve accuracy levels comparable to state-of-the-art (SoA) deep learning networks while significantly reducing the number of required computational operations [152], therefore making them a suitable candidate to process highly sparse data and to ultimately achieve high energy-to-information processing proportionality.

This chapter will go through the steps required to design a hardware processing engine to accelerate the SNN execution at the extreme edge. The discussion will focus on designing a fully digital hardware accelerator for SNNs workloads. We will start our discussion by listing the main challenges introduced by the nature of the data to be processed. Then, we will present the architectural solutions for addressing such challenges. Moreover, we will see how to efficiently make such a design modular and capable of executing various SNN processing-related jobs in parallel. Eventually, we will show silicon results, demonstrating the inherent energy-proportionality of the proposed architecture and the fact that such architecture approaches classical DNN accelerators' energy efficiencies [153]. The key contributions of this chapter are:

- A data framework to efficiently represents sparse data on edge devices. We show that such data representation facilitates data processing in streaming data-driven computing engines.
- The design of SNE: a dedicated modular fully digital hardware accelerator for spiking neural networks.
- Experimental results form an actual silicon implementation in GlobalFoundries 22nm technology node of the proposed architecture, demonstrating the inherent energy-proportionality of the accelerator.

4.2 Related work

Over the last years, research and industry have proposed various deep learning engines to accelerate inference at the edge, achieving extreme energy efficiencies [154]. Thanks to the progress made at the algorithmic level, novel low-precision, highly-quantized yet very accurate networks were proposed [155]. The hardware platform kept the pace by proposing new processing architectures capable of efficiently exploiting low memory footprints and performing low-precision operations [49, 156].

As mentioned previously, neuromorphic algorithms have been attracting increasing attention as a more energy-efficient alternative to conventional deep learning approaches [157], especially in those contexts where input feature maps are produced at a non-constant rate and are also characterized by high unstructured sparsity [158].

As for conventional DNN tasks, the most efficient strategy to deploy SNNs at the edge is to execute them on dedicated hardware engines. We will refer to hardware platforms capable of accelerating neuromorphic algorithms as "neuromorphic platforms". Neuromorphic platforms can be classified into two main categories: Analog or mixed-signal and digital accelerators. Both classes of accelerators implement a certain number of SNN elementary computational units.

4.2.1 Analog and mixed-signal neuromorphic platforms

A key advantage of analog and mixed-signal implementation over digital implementation is the typically higher energy efficiency that they can achieve. Such energy efficiency is often achieved by operating individual transistors in the sub-threshold regime to form complex analog computing primitives. This approach allows us to achieve a very small neuron area footprint for complex neuron models [159]. However, these designs require a significant engineering effort to scale, as their functionality is often technology-dependent, requiring laborious tuning to the technology node. Additionally, analog and mixed-signal computational primitives require many biases generated on-chip, often degrading the system-level energy efficiency.

4.2.2 Digital neuromorphic platforms

Digital neuromorphic platforms implement SNN computational primitives on digital hardware data paths. To reduce the circuit complexity, such platforms typically adopt less complex neuron models [160–162]. This is due to the fact that non-linear functions that are often required to implement the elementary neuron dynamic are more difficult to implement with digital hardware [163]. On the other hand, digital neuromorphic platforms do not require a significant re-design when moving to a more scaled technology node. Therefore, digital neuromorphic platforms enable fast integration into newly designed efficient digital edge computing platforms.

4.3 Architecture

The accelerator we present in this chapter takes inspiration from the architectures mentioned in the previous sections. This section will go through the key features that allowed us to achieve high energy efficiency and enable the integration of our accelerator into an embedded edge computing device.

The `SNE` architecture (Figure 4.1) is composed of a set of independent parallel processing engines called slices (`SLs`). Each `SL` is connected to a synaptic crossbar (`C-XBAR`), which also connects two autonomous direct memory access engines (`DMA`) used to transfer events from the memory to the `SLs` and vice versa. Output event streams produced by the `SLs` are joined in a single stream using a `collector`, which is also connected as a master to the `C-XBAR`. `SNE` can be integrated as a memory-mapped peripheral into a system on chips (SoC) and programmed through a register interface. The following subsections provide a more detailed description of each `SNE` top-level module.

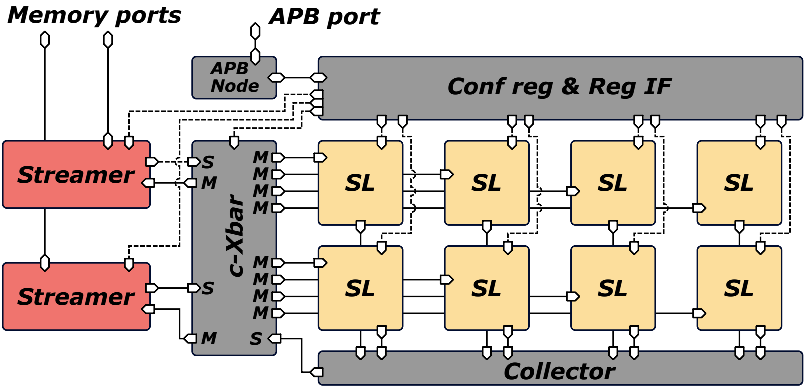


Figure 4.1: SNE accelerator top-level architecture

4.3.1 Spatial and temporal data representation

In sparse neural engine (SNE), we took inspiration from well-known SoA methodologies to efficiently represent sparse data. We adopted a similar data representation for the highly sparse data traversing the internal accelerator modules; specifically, we used the coordinate list (COO) representation. Compared to even more compact data representations like CSR or CSC, the COO results in lower hardware complexity as no computation is required to extract the absolute coordinates of each event, and each active event is referenced independently from the whole matrix. Therefore, this choice allows for representing the active events that need to be processed selectively. Therefore, computational operations are parsimoniously performed to update only the output neuron whose receptive field includes a specific input event.

A peculiarity of SNNs is that the membrane potential of each output neuron decays over time. Therefore, theoretically, each output neuron needs to update its value at every step regardless of the fact that an input event falls into each output neuron's receptive field. In SNE, we also adopted the same approach to represent the active events in time. By explicitly representing the timestamp of each event and by storing the "last update" time of each output neuron, we could postpone each output neuron membrane potential update to the time when a new event falls into the output neuron receptive field. To calculate the correct membrane potential value, it is sufficient to compute a cumulative decay based on the current time of the input event and the time of the last update of each output neuron. In other words, from the data representation point of view, in SNE, the time information associated with each active event is considered simply as a fourth coordinate. Equation (4.2) reports the SNE representation for the sample stream of events distributed over four timesteps, presented in equation (4.1).

$$\begin{pmatrix}1 & 0 & 0 & 1\\ 0 & 0 & 0 & 0\\ 0 & 0 & 0 & 0\\ 0 & 0 & 0 & 0\end{pmatrix} \begin{pmatrix}0 & 0 & 0 & 0\\ 0 & 0 & 0 & 0\\ 0 & 0 & 0 & 0\\ 0 & 0 & 0 & 0\end{pmatrix} \begin{pmatrix}0 & 0 & 0 & 0\\ 0 & 0 & 0 & 0\\ 0 & 1 & 0 & 0\\ 0 & 0 & 0 & 0\end{pmatrix} \begin{pmatrix}0 & 0 & 0 & 0\\ 0 & 0 & 0 & 0\\ 0 & 0 & 1 & 0\\ 0 & 0 & 1 & 0\end{pmatrix} \tag{4.1}$$

$$\begin{aligned} row &= [0,0,2,2,3]\\ col &= [0,3,1,2,2]\\ time &= [0,0,2,3,3] \end{aligned} \tag{4.2}$$

Combining the “compressed” spatial and temporal event representation allows for efficiently consuming a highly sparse stream of input data and processing them with a computationally “dense” sequence of operations executed by the output neuron data paths where nearly optimal utilization is achieved.

4.3.2 Execution model and mapping

The SNE accelerator can be considered a non-von Neumann data flow architecture. During the computation phase, the input feature maps are fetched from the main memory and streamed inside the engines of the accelerator. Specifically, SNE implements the so-called “output stationary” computational model. Indeed, the output neurons of a neural network layer or a smaller patch (tile) are statically mapped on the available computing engines. Weights are statically loaded into local buffers, and input features are fetched from memory and dispatched to all engines in a broadcast fashion.

The SNE can be used in two modes. If the neurons of an SNN can be mapped entirely on the `SNE` available `Clusters` along the spatial dimensions, each `SL`can be used to implement a different layer of the network, and the synaptic connections between neurons of consecutive layers are achieved through the `C-XBAR`. In this mode, events from the `collector`can be redirected to any `SL`. Output events are produced simultaneously with the input event processing, and all the network layers can execute in parallel. Alternatively, if the network needs to

```
# SW managed loops ---------------------------------
for k_o in range(0,C_o):           #output Ch events
 program_sne(W)                      #change weights
#---------------------------------------------------
  # SNE managed loops ------------------------------
  for t in range(0,T):               #time dimension
   for evt_i in events_in[t]      #explicit evt repr
    k_i,e_x,e_y = get_address(evt_i)
    for i in range(0,H_o):         #output h neurons
     for j in range(0,W_o):        #output v neurons
      w_ij = weight(i,j,k_i,e_x,e_y,W)  #weight calc
      evt_o = neuron_dynamics(i,j,w_ij)
      events_out[t].append(evt_o)      #push evt out
  #-------------------------------------------------
```

Listing 4.1: spiking neural network layer execution.

allocate more neurons than available in the `SNE`, intermediate feature maps (output events) must be stored in the external memory. In this case, the `SNE`can be used in a time-multiplexed way to execute only a tile of the network. In this operating mode, synaptic connections are implemented by both the `C-XBAR`and the `DMAs`through the external memory.

Listing 4.1 reports the SNE input event processing pseudo-code.

Figure4.2 shows the execution pipeline of operations to compute an SNN layer. An input event is fetched and made available to all `Clusters`. Then, the `SNE` updates all the neurons of each `Cluster` that are sensitive to the current input event, this operation is performed in 48 clock cycles. The state of each output neuron is held across multiple input event processing, and as soon as a firing operation is received, all the neurons having the membrane potential above the threshold fire an output event.

4.3.3 Data transfer

SNE autonomously transfers the input feature maps and the weights needed for computation. This task is offloaded to two `DMAs` that behave as direct memory accesss (DMAs) on the main interconnection

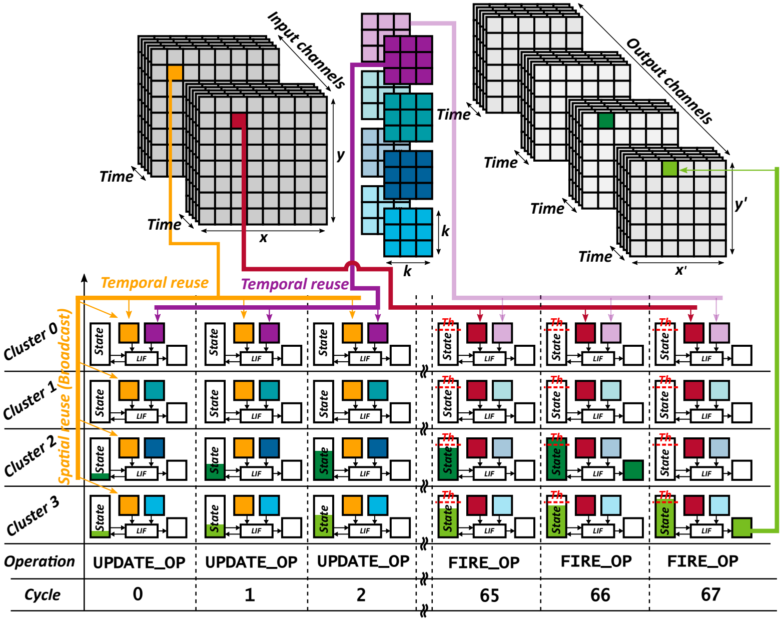


Figure 4.2: SNE convolution operation execution: an input event is sent to all `Clusters` in parallel, weights are stored in the kernel buffer inside the `SL`, and fetched depending on the input address and output neuron position. The yellow and red events belong to the same time step, therefore the first event (yellow) updates the neuron state, while the second (red) allows the neurons to fire.

bus. As SNE is a streaming architecture, the most efficient configuration is obtained when one of the streamers is configured to fetch data from the main memory while the other is in charge of storing the output events produced by the accelerator back to the main memory. In this configuration, SNE can implement a streaming computing pipeline.

Because of the explicit event encoding used for the event to be processed in SNE, the input feature maps can be stored in the main memory as a simple linear sequence of 32bits-wide data. Therefore, `DMAs` implement a simple 1D data movement scheme. SNE implements an "output stationary" computing scheme. In this situation, the weight can be stored in the main memory and loaded into SNE as a deterministic linear sequence of data. Therefore, the same streamers are also used to load the weights inside the accelerator.

Memory requests are granted by the main interconnect bus with a non-deterministic latency. Therefore, to mitigate the impact of delayed memory load grants and avoid stalls caused by a delayed memory store operation, the `DMA` contains a 16-words First-In-First-Out (FIFO) event memory. In other words, the FIFO allows for absorbing memory latency cycles (e.g., due to memory banks access contention) and does not propagate such latency cycles inside the accelerator pipeline.

Output data are produced in parallel by each processing engine of the accelerator. A module, the `collector`, allows packing the output event streams from each `SL` into a single time-synchronized stream and sending it to the output streamer through the main `C-XBAR`. Since the activity of the `SLs` is sparse, a single `DMA` can provide significantly more bandwidth than required on a single `SL` output port. Therefore, the collector arbitrates between the `SLs` output ports and multiplexes them into a single event stream toward the memory.

4.3.4 Interconnect

The SNE accelerator is composed of multiple engines. Input events need to be efficiently transferred from the main memory to each engine and vice versa by the streamers. To provide high flexibility, the main interconnect of SNE is implemented as an all-to-all combinational crossbar (`C-XBAR`). Such configuration allows reconfiguring the interconnection scheme between the input and output ports of the crossbar very easily. The configuration of the `C-XBAR`is controlled through a memory-mapped register interface. Such an interconnection scheme represents a good trade-off between the latency introduced by the interconnect, which is a single cycle, and the possibility to implement a broad spectrum of interconnection schemes, few possible interconnection scheme examples are provided here:

- Broadcast connection from a `DMA` to all `SLs`, the event stream is replicated for each destination port.

- Broadcast connection from a `DMA` to a subset of the available `SLs`.
- Loop-back connection from a `DMA` to the other `DMA` .
- Chained connection from one `DMA` to a single `SL`, and from the output of such `SL`to all the remaining ones.

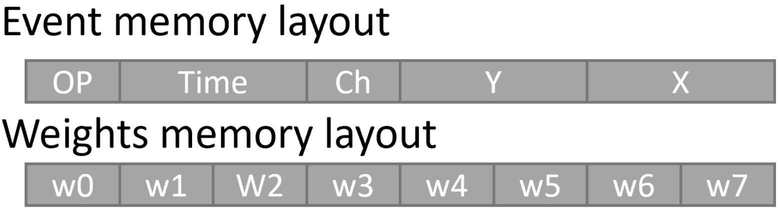


Figure 4.3: SNE internal data representation for events and weights

The data format used for the internal event representation is described in Figure 4.3. Such event representation is used for all the streams traversing the interconnection `C-XBAR`. Each `SL` is connected to the `C-XBAR` with communication protocol using a ready-valid (RV) handshake for flow control. To implement the configuration listed before, the `C-XBAR` can operate in two distinct modes: *i)* single master to single slave port (point-to-point); this configuration is also used to both transfer events and load configuration parameters. *ii)* single master to multiple slave ports (broadcast); in this configuration, the `C-XBAR` can perform flow control and pause the transaction until all slave ports have received the event.

4.3.5 Computing engines

The entire SNE accelerator is built around a computing engine called `SL`. The number of `SLs` is parametric, i.e., the number of internal computing engines can be configured at design time. This section will describe the internal architecture of a single `SL`. Figure 4.4 provides a block diagram representation of an SNE `SL` ,where all the main components are highlighted.

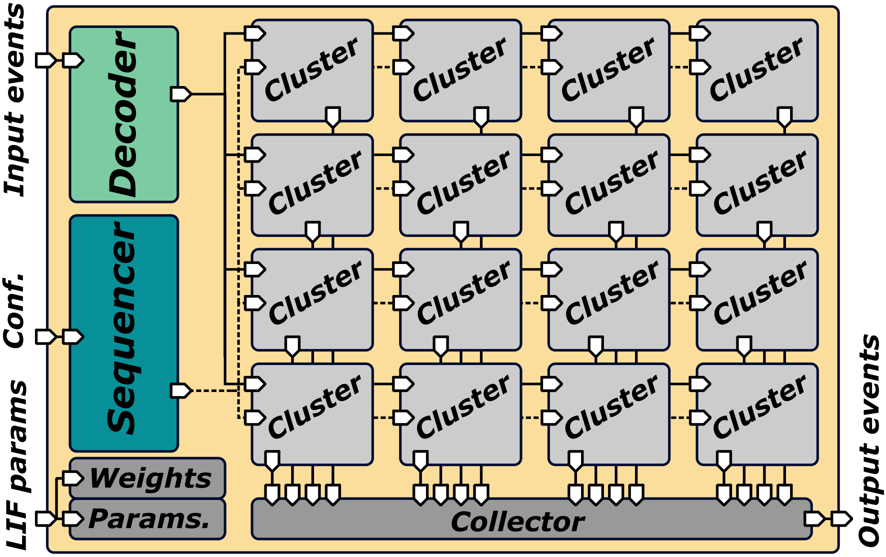


Figure 4.4: SNE slice architecture

SNE has been designed to perform parallel computation. To achieve this goal, each `SL`is further divided into sub-units. Specifically, Each `SL` instantiates 16 parallel computational units, called `Clusters`. A `Cluster` is the elementary computing element of SNE, and it contains a single adaptive leaky-integrate and fire (ALIF) neuron data path. The `Cluster` data-path is designed to compute a neuron state update in a single clock cycle: the implementation of multiple neurons in `Cluster` is achieved by time-domain multiplexing (TDM) such data path. The data associated with each output neuron mapped on that `Cluster`is retrieved from a local latch-based memory element and stored at the next cycle; once that output neuron update is finished.

In section 4.3.4 we listed the possible configuration modes implemented by the crossbar. As the input events are transferred from the `DMA` to one or multiple `SLs`, each `SL` internally implements a filtering mechanism to react only on those input events directed to the output neuron mapped on that `SL`. At a lower level, i.e., the `Cluster` unit level, the filtering mechanism allows to clock-gate all the

`Clusters` that do not need to update the state of any of the mapped output neurons. Each `Cluster` implements 64 TDM neurons using 4 bits for synaptic weights and 8 bits for the internal state computation. Figure 4.5 shows a block diagram representation of a neuron `Cluster`.

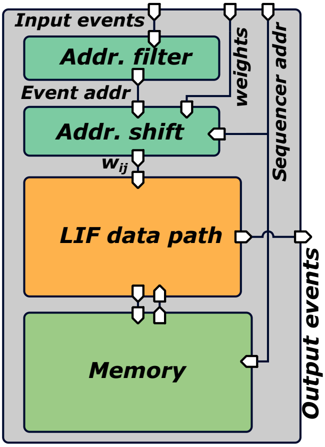


Figure 4.5: SNE cluster architecture

Execution on all `Clusters` happens simultaneously, and the TDM sequence of operations are orchestrated by a module called `Sequencer`. The `Sequencer` provides the relative address of the current TDM neuron being currently updated on each `Cluster`. Based on this relative address and the input event address, the correct weight is retrieved from a local weight buffer at each successive neuron update. Note that all the 64 TDM neurons of each `Cluster` can potentially produce an output event that needs to be written to the main memory or redirected toward a different `SL`during the output neuron update. During this process, a stall caused by the unavailability of the `C-XBAR` to accept an output event would have a detrimental effect on the energy efficiency of the accelerator. Therefore, o avoid stalling the TDM neurons update, each `Cluster` is connected to an output event FIFO, and all FIFOs are connected to a `collector` module which can redirect the stream of events towards a `C-XBAR` input port. This mechanism is similar to the one implemented for absorbing the main memory latency possibly occurring at the `DMAs`. It allows guaranteeing to perform the neuron updated with the maximum throughput achievable by each `Clusters`.

All computed output neurons across the various `Clusters` of a `SL` have the same relative position, i.e., the same triplet of output coordinates (x,y, and output channel position). The absolute spatial mapping of the output neurons is achieved by shifting each address with respect to a `Cluster` base address. To summarize, to achieve high throughput and energy efficiency, the following measures were put in place at the `SL` and `Cluster` architectural level:

- The ALIF neuron dynamic data path is combinational; this allows to perform a complex set of operations in a single cycle.
- Multiple neurons, 64, are implemented by TDM on a single data path.
- To overcome the high memory bandwidth that characterizes SNN inference, the states of all the neurons implemented by a `Cluster` are stored locally into a dedicated state memory.
- Output event write stalls are mitigated by introducing output first-in first-outs (FIFOs), ensuring maximum throughput during the TDM output neuron update.

4.3.6 SNE neuron model

In SNE, we implemented an ALIF neuron model [164]. Such a model is a variant of the simple leaky-integrate and fire (LIF) neuron commonly adopted in many digital accelerators for SNNs [160, 161]. The main difference between the LIF and the ALIF is that the second can adapt the firing threshold whenever a spike is generated. This mechanism introduces more complex dynamics, which reduces the probability of firing an output spike in the presence of high activity of the neuron.

The elementary neuron membrane potential update is given by the equation 4.3. In this equation, V^t represents the output neuron membrane potential at time t. The $e^{\frac{-\delta t}{\tau}}$ term is the iterative formulation of the exponential decay, which is multiplied by the membrane potential at each time step, the $\sum_j W_{ij} S_i[t]$ represent the synaptic contribution added by each input event directed to the receptive field of the output neuron.

$$V^{t+1} = V^t \cdot e^{\frac{-\delta t}{\tau}} + \sum_j W_{ij} S_i[t] \tag{4.3}$$

Membrane potential exponential decay

As we mentioned in the previous section, SNE parsimoniously performs the minimum operation of operations to update an output neuron's membrane potential. To efficiently implement such a feature at the neuron data path level, we observed that the output neuron membrane update does not need to be performed recursively at every time step t. It is possible to combine the iterative decay performed across multiple times steps t into a single operation. Such an idea is proven in the following discussion. We start by calculating the membrane potential at time $t = 0$.

$$V^t|_{t=0} = V_0 \tag{4.4}$$

By replacing this term into equation 4.3, and assuming, for simplicity, that no synaptic contribution is added to the membrane potential, we obtain that the membrane potential at time $t = t+1$ can be computed with the following formula.

$$V^{t+1} = V^t \cdot e^{-\frac{\delta t}{\tau}} = V_0 \cdot e^{-\frac{\delta t}{\tau}} \tag{4.5}$$

Suppose we iteratively unroll the exponential decay up to $t = 2$. In that case, we observe that the membrane potential at this time can be expressed as a function of the initial membrane potential V_0 and a new decay coefficient $e^{-\frac{2\delta t}{\tau}}$ that can be calculated as a function of t.

$$V^{t+2} = V^{t+1} \cdot e^{-\frac{\delta t}{\tau}} = (V_0 \cdot e^{-\frac{\delta t}{\tau}}) \cdot e^{-\frac{\delta t}{\tau}} = V_0 \cdot e^{-\frac{2\delta t}{\tau}} \tag{4.6}$$

therefore, we can express equation 4.6 in a more general form, where $\alpha^N = e^{-\frac{N\delta t}{\tau}}$.

$$V^{t+N} = V_0 \cdot \alpha^N \tag{4.7}$$

From the equation (4.7) we can derive two generic values of V^t calculated at $t = t + N$ and $t = t + M$ respectively.

$$V^{t+N} = V^t \cdot \alpha^N \qquad N > 0 \tag{4.8}$$

$$V^{t+M} = V^t \cdot \alpha^M \qquad M > N \tag{4.9}$$

$$\tag{4.10}$$

We can express the time difference between M and N as Δt, and derive M as a function of N.

$$M - N = \Delta t \tag{4.11}$$

$$M = \Delta t + N \tag{4.12}$$

Therefore, we can write equation 4.7 at the time step M as a function of N:

$$V^{t+M} = V^t \cdot \alpha^M = V^t \cdot \alpha^{\Delta t + N} \tag{4.13}$$

From the equation 4.8 we calculate V^t as a function of N:

$$V^t = V^{t+N} \cdot \alpha^{-N} \tag{4.14}$$

Eventually, we can substitute equation 4.14 in equation 4.13. In this way we can calculate V^{t+M} as a function of V^{t+N}, and the time difference between N and M, namely, Δt:

$$V^{t+M} = V^{t+N} \cdot \alpha^{\Delta t} \tag{4.15}$$

Figure 4.6 shows the ideal exponential decay and the one calculated since the first membrane potential update.

Neuron data path implementation

The SNE ALIF data path implements the exponential decay employing a LUT. The LUT stores the pre-computed fixed-point decay coefficient for the exponential decay for 32 successive time steps since the last update of the membrane potential. Figure 4.7 shows a block diagram representation of the neuron data path.

The data path is divided into 5 main parts:

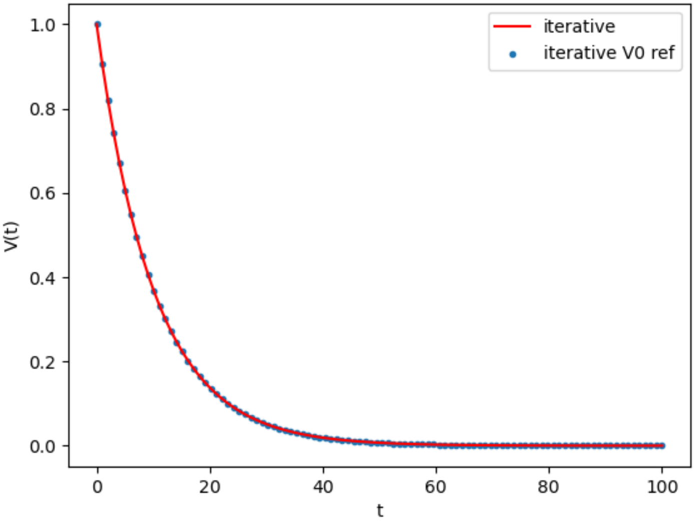


Figure 4.6: Membrane potential exponential decay. The red curve shows the step-by-step iterative calculation, blue dots are calculated by knowing the initial membrane potential value, and the time delta since the beginning (a special case of eq. (4.15))

- The synaptic contribution calculation. This part allows receiving the input synaptic contribution, adding it to the current membrane potential value, and saturating the output value if it exceeds the representable range.

- The spike generation circuitry. This part performs the adaptive threshold calculation and compares the up-to-date value of the membrane potential to determine whether an output spike can be generated.

- The membrane potential decay. Such circuitry includes the LUT with the pre-computed decay coefficients multiplied by

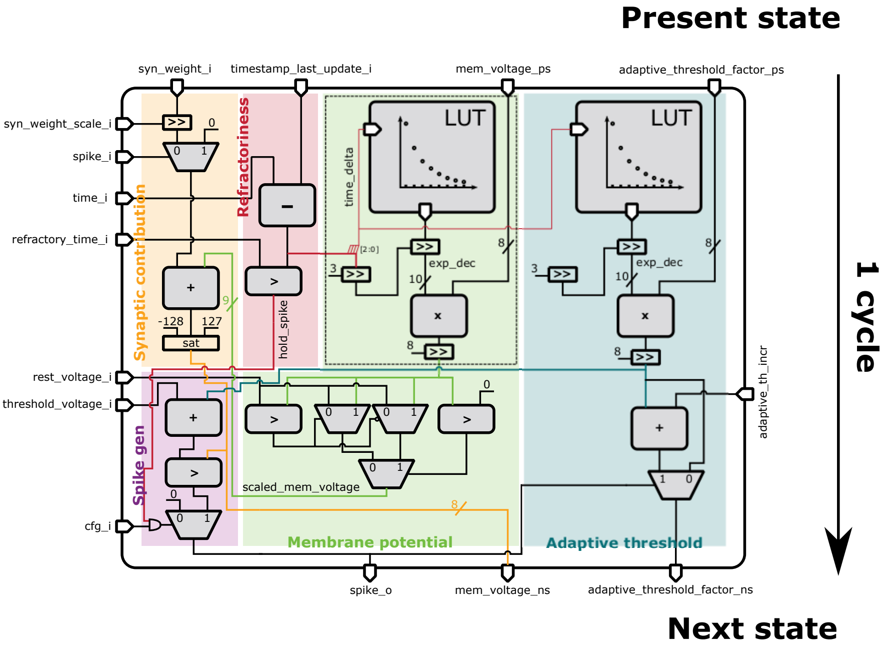


Figure 4.7: SNE ALIF neuron data path combinational block diagram. At the top the current state inputs are fetched from the state memory and fed into the data path, signals at the bottom are the outputs that are stored in the state memory. signals on the side represent the neuron data path memory mapped configuration values.

the current state value of the membrane potential to produce the next state value.

- The adaptive threshold decay. Similar to the membrane potential update sub-circuitry, such circuitry includes the LUT with the pre-computed decay coefficients multiplied by the current state value of the adaptive threshold to produce the next state value.

- The sub-circuitry that compares the last pike's time with the current time to determine whether the neuron is in its refractory period and spike generation has to be inhibited.

4.4 Experimental results

This chapter will present the results related to the SNE accelerator implemented in the Kraken chip. A more extensive description of the Kraken chip, as well as the results that are not directly related to SNE, will be presented and discussed in chapter 5.

4.4.1 Physical implementation

The SNE accelerator presented in this chapterhas been implemented in the GlobalFoundries 22nm technology process. To synthesize the RTL description of the SNE accelerator, which was instantiated inside the Kraken chip architecture, into the respective gate netlist, we used the "Synopsys Design Compiler 2020.09" tool. The design's physical implementation has been performed with the "Cadence Innovus Implementation System 19.1" tool. We used 8T, 20, 24, 28, L, and SL voltage threshold cells for the proposed design. The setup time has been optimized for the slow (SSG) 0.59 V −40 °C process corner, while the hold time has been optimized for the (FFG) 0.72 V 125 °C and −40 °C process corners. The SNE has been constrained to operate at a maximum clock frequency of 200 MHz in the slow process corner.

The SNE accelerator has been placed on a 1.8 mm^2 floorplan area. The standard cell area density of the design in such an area is 39.9%. A switchable power domain hosts SNE. Therefore, when not in use, the voltage supplying the accelerator standard cells can be switched off to reduce the leakage power consumption of the chip. Moreover, SNE is clocked by an independent on-chip clock generator, clock domain crossings (CDCs) FIFOs are placed at the boundaries between the SNE `DMAs`, which operate at the system clock frequency and the `C-XBAR` interconnect. The SNE accelerator is not connected to any external pad, and all accelerator configurations are applied through the main system bus. Figure 4.8 highlights the SNE accelerator in the Kraken chip floorplan.

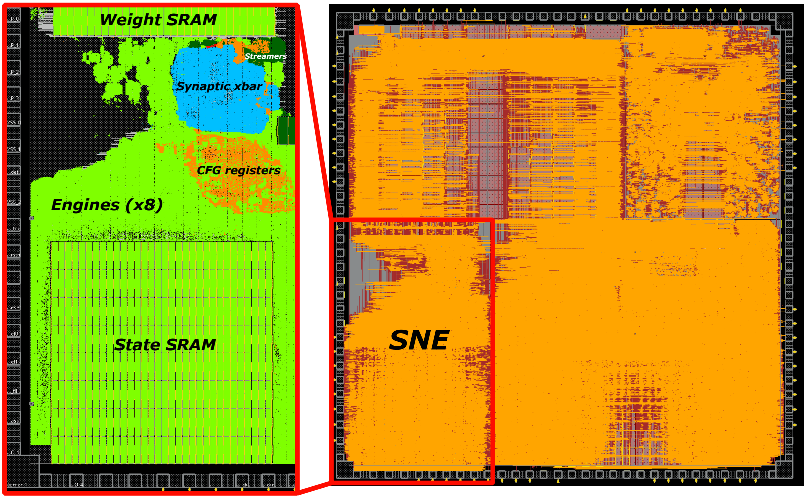


Figure 4.8: On the right, the SNE accelerator placement in the Kraken chip floorplan. On the left, a zoom in into the SNE accelerator floorplan which highlights the various sub-modules of the accelerator.

4.4.2 Area breakdown

The plot presented in figure 4.9 reports the area breakdown of the SNE accelerator. The numbers reported in the plots have been estimated by the actual physical layout of the Kraken chip implemented with the “Cadence Innovus Implementation System 19.1” tool. The SNE slices occupy the main part of the area. Inside each slice, most of the area is occupied by the state memories, where the state of 8192 neurons can be stored, 1024 per `SL`. The remaining area of the SNE top module is occupied by the configuration registers and the interconnection subsystem, the `C-XBAR`. Inside each `SL`, the remaining area is occupied by the ALIF neuron data path, 16 per `SL`, and 16 output FIFOs that store the output event produced by each neuron data path.

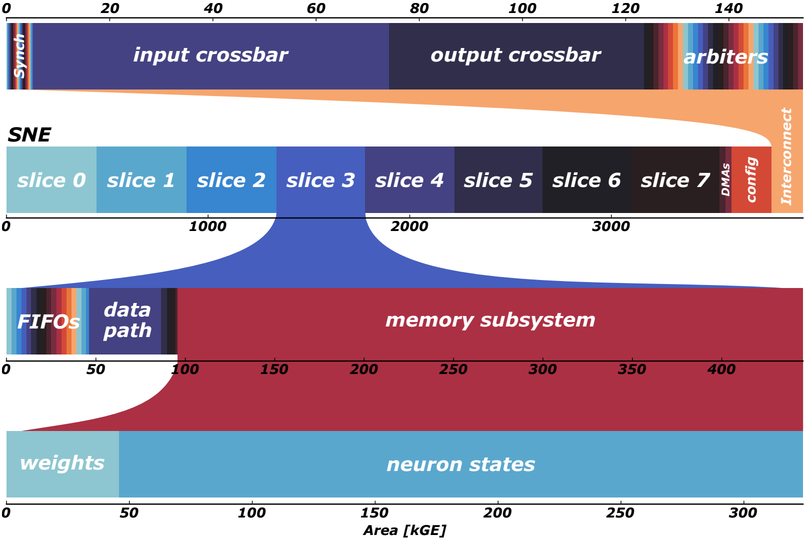


Figure 4.9: Area breakdown of internal components of the SNE accelerator.

4.4.3 Experimental setup

The results presented in this section have been obtained by testing the Kraken SoC on an “Advantest SoC hp93000” integrated circuit testing device. The internal tester voltage supply channels have precisely regulated supply voltages; such channels can also precisely measure the current delivered to each power domain with an error lower than 1%. The results presented in this section will refer to the SNE accelerator domain only. All the measurements have been performed at an ambient temperature of 25 °C.

4.4.4 Power consumption and performance

This section reports the maximum achievable frequency versus the supply voltage and the power consumption of SNE at such frequency and voltage operating points. The power consumption of SNE depends mainly on the number of active `SLs`. To provide an upper bound for the power consumption, we executed a sample application capable of exercising all the `SLs` simultaneously, emulating the case where a workload occupies all the available resources of the SNE accelerator. We built the sample workload as a spiking convolutional neural network layer patch of 14x14 input pixels, 64 input channels, and 32 output channels. This workload can fully saturate the resources of the SNE accelerator, computing, in parallel, the output of 6272 output neurons when SNE is configured in convolutional mode. All the `SLs`are active, and each slice is computing 4 different output channels related to the same input convolutional layer patch. In each `SL`, 196 output neurons are mapped on 4 of the available 16 clusters. Therefore, each `SL`implements up to 784 output neurons. We emulated an infinite time dimension on the input layer patch to measure a sustained steady state, performance, power consumption, and energy efficiency. SNE performed the computation mentioned above in an infinite loop to achieve this goal.

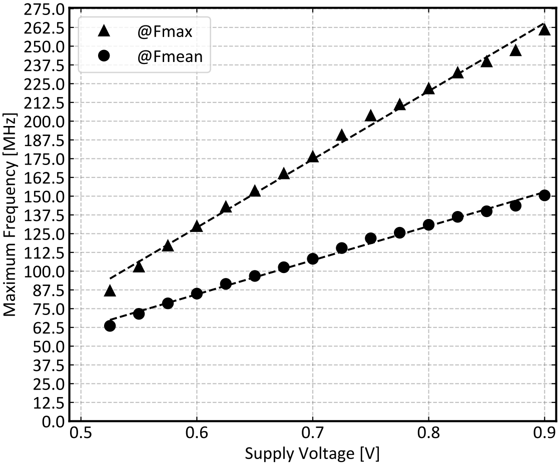


Figure 4.10: SNE maximum frequency when executing a spiking neural network convolutional workload with average 5% output activity

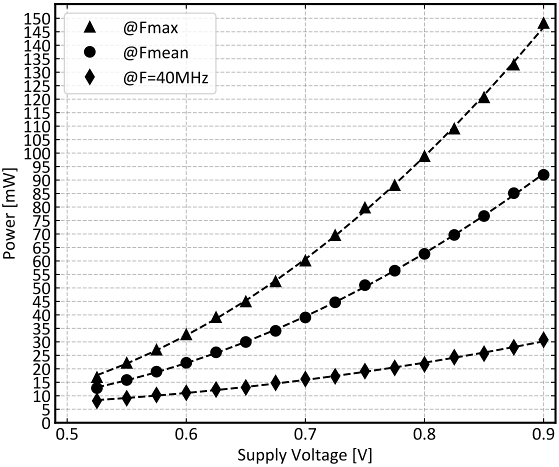


Figure 4.11: Power consumption of SNE at the same voltage and operating frequency points for the spiking convolutional benchmark

The first plot, presented in Figure 4.10, reports the maximum achievable frequency of the SNE accelerator versus the supply voltage when executing the sample computation. From the STA performed during the physical implementation, we know that SNE critical path for the sub-circuitry clocked with the main system clock resides in the connection that goes from the `DMAs`to the main system interconnect. Moreover, as SNE is a data-driven architecture, the number of outputs produced by the accelerator significantly depends on the accelerator input. In our tests, the benchmark is producing an output event at an average firing rate that is consistent with an actual spiking neural network workload [152]. To write the output events to the main memory without stalling the internal data paths, the ratio between the system clock and the internal SNE clock frequency needs to be higher than 0.3, assuming no contention caused by other agents interacting with the same system memory banks. In the second plot presented in Figure 4.11, we report the power consumption associated with the voltage and frequency measurement points reported in 4.10, as well as the power consumption versus the supply voltage when the frequency remains constant at 40 MHz.

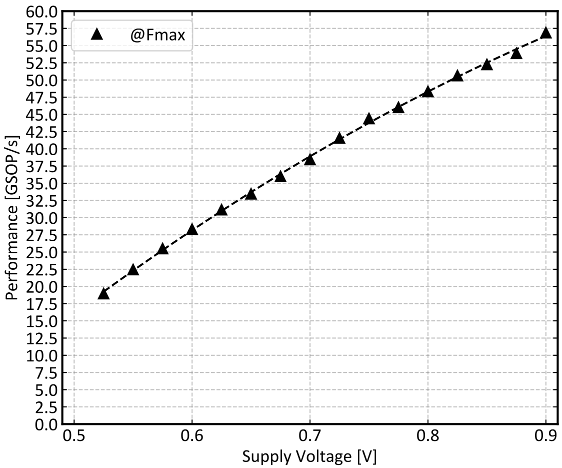


Figure 4.12: Maximum performance of SNE when executing the spiking convolutional benchmark reported in terms of number of synaptic operations per second

The plot shown in Figure 4.12 reports the maximum number of operations per second (SOP/s) that SNE can perform when clocked at the highest achievable frequency. These points represent the maximum performance achievable by SNE. The performance reported in Figure 4.12 refers to a "sustained" input event stream processing, as well as the continuous output event storage in the main memory in the condition reported at the beginning of this section. Note that the performance reported in Figure 4.12 is deterministic and does not depend on the input's " sparsity " SNE. The time to acquire and process an input event is fixed by design. Suppose a spiking neural network produces less than 5% output spikes on average. In that case, the measured performance reported in the plot can be used to estimate the inference time of a SNN layer, or a layer patch, depending on the number of active input events that need to be processed by SNE. SNE can sustain higher output activity for a short time without stalling the internal data path; this condition occurs if the internal output FIFOs reach their maximum capacity. Alternatively, to mitigate the effect of the output bandwidth bottleneck, the `DMAs`can be clocked at a higher frequency, reaching a clock ratio between the SNE internal clock and the system clock of 1.

4.4.5 Energy consumption

Another critical aspect of a digital SoC, other than the maximum performance and power consumed by the circuit, is the amount of energy required to perform a certain number of operations. Indeed, making effective use of the available energy becomes crucial when a battery provides such energy; this is true in most of the IOT devices deployed around us. In this section we present the SNE related energy metrics. Based on the results of maximum frequency and performance reported in Figure 4.11, 4.10 and Figure 4.12 respectively.

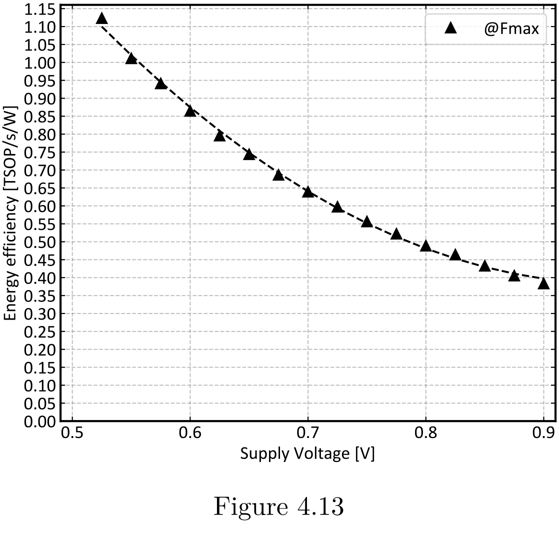


Figure 4.13

Figure 4.13 shows the number of synaptic operations per second, normalized by the total power consumption of SNE. This metric indicates how efficiently the power consumed by the accelerator is used to perform a useful operation. This metric drastically degrades as the static power consumption of a digital circuit increases; for example, static contributions caused by the circuit leakage or any other dynamic power consumption that can not be removed, e.g., the power consumption associated with on-chip clock frequency generators or always-on control logic. As observed many times in all highly scaled technology nodes, as it is the GlobalFoundries 22nm in which SNE has been manufactured, the highest energy efficiency, i.e., the highest number of operations per unit of power and time, is achieved at low operating voltages, in the so-called NTC regime. This phenomenon is well explained by [49]. In our experiments on SNE, we could confirm this trend, reaching the highest energy efficiency at 0.5 V.

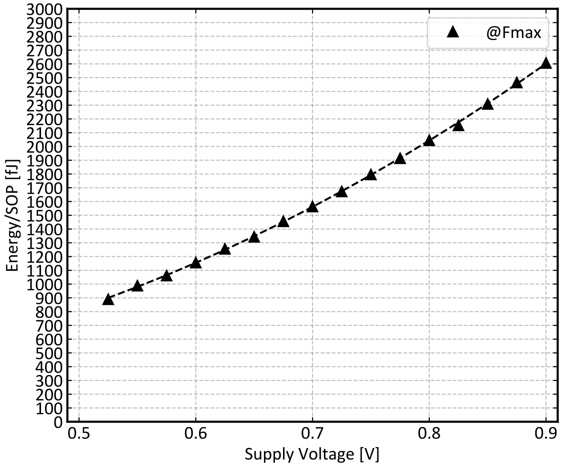


Figure 4.14: Energy per synaptic operation of SNE when executing the spiking convolutional neural network workload.

Figure 4.14 reports the energy cost of a single synaptic operation (SOP), By SOP, we intend the total number of operations that a data path performs in a single cycle, initiated by an input event falling into the receptive field, i.e., the 3x3 kernel in the case of convolutional mode, of an output neuron. Specifically, it refers to:

- Retrieval of the membrane potential from the state memory.

- Extraction of the coefficient from the LUT and multiplication of the membrane potential by such coefficient to perform the decay.
- Extraction of the weight corresponding to the synaptic connection between the input event and the output neuron and accumulation of such synaptic contribution to the decayed membrane potential.
- Comparison of the membrane potential value to the firing threshold and spike generation if the value exceeds such threshold.
- Reset of the membrane potential to a known value in the case of a spike generation.
- Storage of the up-to-date membrane potential, "the time of the last update" and "time of the last spike" to the state memory.

To obtain the energy per synaptic operation consumed by SNE, we calculated the total energy consumed by the accelerator in a single clock cycle. We then divided this value by the number of operations performed in the same clock cycle. Note that the amount of operation can not be directly observed on the actual chip, as SNE is not equipped with performance counters that can trace this information. However, the total number of operations can be easily computed by simulating, with a cycle-accurate simulator, the same sample application benchmark on the description of the SNE accelerator, i.e., the same RTL description used to synthesize the circuit. The same trend we have observed for the energy efficiency reported in Figure 4.13 is also confirmed in the case of the energy per synaptic operation. SNE reaches the lowest energy per synaptic operation at 0.5 V.

4.4.6 Comparison with the state of the art

Table4.1 reports a comparison between SNE and the SoA digital architectures implementing comparable accelerators for SNNs. Compared to the architecture presented by Akopyan et al. in [161], SNE shows lower power and higher energy efficiency. A similar consideration can be done for the platforms presented by Deng et al. in [165]. Compared to the accelerator presented by Davies et al. in [160], Kim et al. in [166] and Höppner et al. in [167], SNE shows the lowest energy per synaptic operation and the highest absolute performance.

Overall, we can summarize this comparison among the neuromorphic platforms presented in the comparison table by stating that SNE shows the lowest energy consumption (sub-pJ) per synaptic operation, approaching the energy consumption of conventional neural network digital accelerators [153] while maintaining or even improving the power consumption, as well as the operating frequency and the absolute performance against other accelerators.

	neuron type	neurons	synapses	network type	implementation technology [nm]	bits	voltage [V]	frequency [MHz]	power [mW]	energy efficiency [GSOP/s/W]	operation energy [pJ/SOP]	performance [GSOP/s]
Truenorth [161]	LIF+	1000000	-	SNN	Digital 28	9	0.75 [0.7-1.05]	N/A	65 [42-323]	400	-	58
Loihi [160]	LIF	1024	1000000	SNN	Digital 14	1-9	-	N/A	-	-	23	-
Spinnaker2 [167]	Prog	250	20000	SNN CNN	Digital 22	Var	0.45 [0.45-0.6]	200	-	3260	1700	-
Tianjic [165]	LIF	-	-	SNN CNN	Digital 28	8	0.9	300	950	649	-	-
Kim et al. [166]	-	256	83000	SNN	Digital 65	4-14	0.45 [0.45-1]	40	3.65 [3.65-268]	-	5.7	10.1
Chen et al. [168]	LIF	4096	1000000	SNN	Digital 10	7	0.52 [0.45-0.9]	105	-	-	3.8	5.2
SNE [50]	**ALIF**	**8192**	**18432**	**SNN**	**Digital** **22**	**4**	**0.52** **[0.52-0.9]**	**175** **[90 - 262]**	**17** **[17-147]**	**1100**	**0.9**	**19**

Table 4.1: State of the art comparison

4.5 Conclusion

In this chapter, we presented a novel, fully digital accelerator for spiking neural network workloads. SNE has been designed to inherently exploit the unstructured sparsity of data produced by event-based sensors by performing a number of operations proportional to the input stream activity. The most important contribution of this chapter can be summarized as follows:

- The unstructured sparsity, which characterizes data produced by event-driven sensors, or that can also arise in the intermediate features of an ANN, can be tackled at the circuital and architectural level. To efficiently process such data, we can design accelerators that can profit from such an irregular data sparsity to reduce the number of operations that, for example, characterize the inference process in artificial neural networks. This was the case in SNE, where the computing engine explicitly consumes spatial and temporal-encoded input events to selectively update the internal output neuron states of those output neurons having a particular input event in their receptive field.

- Accelerator architectures like the one presented for SNE are not specific to a small subset of ANNs, for example, SNNs; instead, the proposed architecture can find applicability in all those contexts where the input data feature a high unstructured sparsity. Indeed, the SNE architecture can be easily re-purposed to perform a different type of sparse computation simply by changing the specific function implemented by the data path sub-module, which, in the case of SNE, is theALIF neuron dynamics.

- We could demonstrate how digital neuromorphic platforms, notably less energy efficient than digital accelerators for CNNs that perform very regular computation, can be improved to achieve a consumed energy per operation that approaches the one shown by more conventional SoA digital accelerator architectures. This result narrows the gap between neuromorphic platforms and classical DNN accelerators [153].

We want to conclude our discussion by remarking that the SNE accelerator presented throughout this chapter can be easily integrated into any digital SoC featuring a main interconnect subsystem with direct memory access and the possibility to memory-map a register-based interface. SNE was successfully integrated on a real SoC manufactured in 22nm advanced technology node and achieved the lowest energy per synaptic operation reported on a flexible digital neuromorphic platform.

Chapter 5

Kraken: An event-driven Brain-Inspired edge computing device

5.1 Introduction

The Internet of Things, e-Health, Smart Sensors, and wearable consumer gadgets are expected to drive the electronic market in the following decades. These applications rely on the capability of the research community to provide devices that couple ultra-low-power (ULP) behavior with a reasonable level of performance. Indeed, these applications are characterized by an increasingly tighter power budget and an increasing demand for computation capabilities. Traditional low-end IoT devices have been designed to serve as the main computing device in low-bandwidth sensor analytic applications, i.e., inertial measurement unit (IMU) data processing, ambient temperature or humidity monitoring, and low rate control tasks. However, the emerging trend is to embed more and more cognitive capabilities into edge computing platforms. Such devices nowadays have to be able to execute small to medium size [169] deep learning tasks on sensor data to limit the data transfer towards cloud computing infrastructure.

Technological advances have driven the quest for energy efficiency. However, the pace dictated by Moore's law has slowed down, and CMOS scaling, which drove semiconductor growth during the past decades, is now delivering only modest energy gains [23]. Therefore, researchers have redirected their efforts towards alternative solutions, taking a step forward from mere technological advances and improving electronic devices at higher abstraction levels.

In this "Moore's law twilight era", further energy gain can be achieved by moving to the near-threshold computing (NTC) domain [170]. In chapter 3, we have observed that operating a digital device in the NTC domain can lead to significant energy efficiency improvements. However, such energy efficiency improvement comes at the cost of lower absolute performance of the device; indeed, high energy efficiency is often achieved at low operating voltages, where a digital circuit is notably slower. This approach needs to be combined with more architectural solutions capable of restoring the desired level of performance, especially in those contexts where significant computational capabilities are required, e.g., when executing DNN workloads.

A viable solution, which exploits the very regular nature of the computation typically performed on such IOT platforms, is to parallelize the application on multiple computing engines. This approach has demonstrated to be highly effective [15] and has opened the way to a significantly more "intelligent" class of applications to be deployed on battery-powered IOT nodes. The platforms that perform edge computation are typically CPU-centric general-purpose "Von Neumann" architectures. Researchers have improved the performance of those devices by increasingly specializing, at a minimal hardware complexity increase cost, the ISA to favor those operations that are repeated many times [119].

The dynamic management of the performance and power consumption of a SoC represents another effective knob that allows meeting the requirements in terms of power budget and desired level of performance [171]. In the past years, approaches like dynamic voltage and frequency scaling (DVFS) and ABB have shown their potential mostly on complex high-end systems. However, their general applicability to almost any digital circuit made it possible to employ such techniques to reduce the power consumption of a broad spectrum of digital devices [56], including low-end ultra-low power (ULP) devices [24].

A big energy efficiency improvement step has been introduced by adopting dedicated computing engines to accelerate specific operations. As we have already observed in chapter chapter 4 and 3, there are significant advantages, in terms of energy consumption, in highly specializing a computing engine for the execution of a reduced set of operations. In this regard, we can observe that the co-existence of general-purpose computing engines, i.e., CPUs and workload-specific accelerators, is nowadays a well-consolidated architectural solution to design energy-efficient SoC [172].

In chapter 2 we have demonstrated how an emerging category of sensors, i.e., the event-based ones, can significantly improve the energy efficiency of extracting semantically meaningful information from environmental stimuli and effectively transfer them to a digital SoC for further data analysis and processing. In chapter 4 we have shown that emerging computing frameworks like the event-drive one are a viable alternative to deal with such highly sparse sensor data. In this chapter, we present “Kraken”, an event-driven SoC capable of directly interacting with event-based sensors and efficiently processing them in an event-driven way. The main contribution of this chapter can be summarized as follows:

- We present a SoC manufactured in Globalfoundries 22nm technology which integrates a small microcontroller subsystem to execute lightweight control tasks, a general-purpose RISC-V based computing cluster for intense machine learning workloads, and a dedicated SoA neuromorphic accelerator for low-precision event-driven computation.

- We describe how to compose an event-driven computing pipeline by outlining a possible staged inference strategy where the system is progressively switched on, depending on the input data to process.

5.2 Related work

Research communities and industries have tackled the challenge of achieving high energy efficiency at the extreme edge from many directions over the last years. Among the most promising strategies to reduce the energy consumption of cognitive tasks deployed at the extreme edge, we could identify two general trends that showed the best results. The first approach goes in the direction of parallel workload executed on energy-efficient computing clusters composed of general-purpose cores [15, 173]. Such an approach has proven to be particularly effective when the final application is difficult to foresee at design time.

The second approach goes more toward a high specialization of the modules composing the SoC. This goal is achieved by integrating highly energy-efficient fixed-function accelerators [34] to compose heterogeneous systems [174]. This approach is particularly effective when the type of workload can be predicted at design time, e.g., in the case of DNN-oriented edge computing devices. Some architecture moved a step forward, cleverly combining both strategies [123], thereby achieving a well-balanced trade-off between application generality and overall system energy efficiency.

Throughout this thesis, we have seen that SNNs could provide significant advantages in terms of energy consumption because of their inherent sparse nature, especially when combined with sensors that can represent information with sparse streams [158]. The aforementioned architectural solutions are designed to deal with a very regular type of workloads, which characterize previous and current generations of DNNs. Therefore such architectures are not efficient when performing sparse computation, like in the case of SNNs workloads.

Several neuromorphic SoCs have been proposed to address this challenge and efficiently execute brain-inspired cognitive tasks [160–162]. Despite the outstanding performance achieved by those solutions, they did not provide the same level of flexibility and "programmability" as shown by the SoA DNNs-oriented platforms presented before. We believe that to facilitate the adoption of neuromorphic platforms at the embedded level and fully express their potential as a next-generation artificial intelligence (AI) platforms, we need to move a step forward and integrate such class of device into ULP SoCs that embedded developers are more inclined to use as a deployment platform for their applications. In this work, we aim to achieve this goal by proposing a heterogeneous microcontroller-like SoC equipped with a dedicated neuromorphic accelerator and a multi-core general-purpose RISC-V-based computing cluster, which can operate on a power budget of 10-100mW. The ultimate goal of Kraken is to bring neuromorphic computing into the digital embedded computing domain, combining it with a more traditional general-purpose approach.

The following sections of the chapter are organized as follows: In section 5.3 we will describe the architecture of the Kraken SoC [51]. Specifically, we will provide details related to the memory hierarchy and interconnection among the various component, as well as system-level aspects like clock domains and power domains. In section 5.5 we will describe the results related to system-level performance and energy consumption, and we will compare such results with the SoA. Eventually, we will provide concluding remarks and an outlook for future development directions.

5.3 Architecture

The architecture of the Kraken chip has been derived from the base “Quentin” SoC architecture presented in 3. The system is divided into multiple clock and power domains. Figure 5.1 shows a block diagram representation of such domains. Compared to the Quentin chip, Kraken features two additional power and clock domains. The first is the esternal hardware processing engine (EHWPE), which hosts two dedicated accelerators, SNE and CUTIE. The former has been already presented in chapter 4, while the latter will not be discussed in this thesis, as it is outside the scope of the analysis presented here. The second domain is a general-purpose accelerator domain called “cluster”, a subsystem composed of eight RISC-V cores.

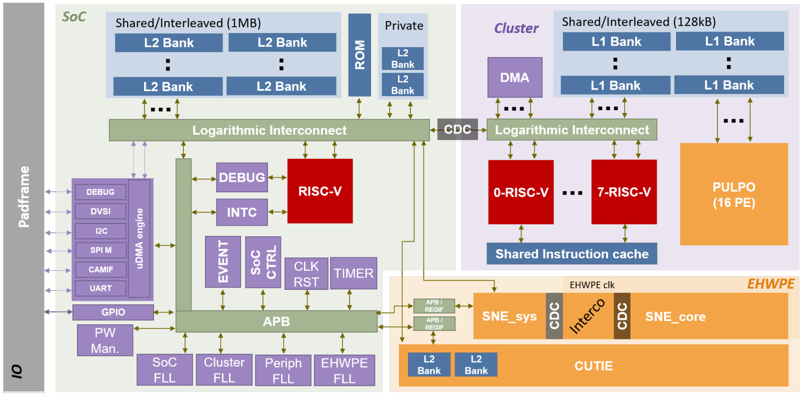


Figure 5.1: Kraken SoC block diagram

5.3.1 Power and clock domains

To carefully control the chip's overall power consumption and enable a versatile application-specific power management scheme, Kraken is subdivided into three independent power domains. Such power domain partitioning allows to switch off unused parts of the system that the running application might not require. The power-up/down of the controllable domains can be performed at runtime. Figure 5.1 also shows the chip power domain partitioning, SoC, Cluster, and EHWPE respectively. The fabric controller (FC) domain is always on because it hosts essential components needed for SoC management tasks, for example, the boot procedure, the clock generators initialization, or the accelerator reset and programming.

5.3.2 Fabric controller

The Kraken FC is built around a 32bits RISC-V core which acts as a main programmable control unit for the whole SoC. The FC hosts the main interconnection busses towards the main L2 memory and the advanced peripheral bus (APB) bus, which controls all the SoC peripherals. As part of the FC domain, we also find a compliant RISC-V debug unit accessible via joint test action group (JTAG), the event unit, which collects interrupt events generated by the peripherals and redirects them toward the core interrupt controller, and four real-time counters (timers) that can be used to generate pulse width modulation (PWM) signals or internal time references. Moreover, the FC hosts the power management unit, accessible through a memory-mapped register interface by the FC core.

Memory subsystem

The main L2 memory of the FC domain is divided into a small core-private section and a bigger shared section. The first one is reserved for storing the RISC-V executable binary and all the core-private variables, e.g., the core stack or critical application variables. The private memory is divided into two 32 KiB memory banks that are accessible through the main interconnect with a linear addressing scheme. The L2 shared memory has a 1 MiB capacity, and it is composed of eight banks. The interleaved section is also accessible through the same main interconnection, and the data stored in this memory are organized in an interleaved fashion. More specifically, a linear 4-byte-wide word memory access is mapped to successive banks of the L2 memory. This interleaved addressing scheme has been chosen to minimize the memory access contention during parallel memory access from different peripherals of the SoC.

Clock generators

The Kraken chip hosts four internal frequency locked loop (FLL) clock generators [175]. Such clock generators use an external 32.768 kHz clock reference, which is provided through an external pad. To configure the frequency output of each FLL, the FC core can access the configuration registers through a memory-mapped register interface via the APB. The FLLs allow generating a clock frequency in a range from a few tenths of kHz to approximately 1.5 GHz, covering both the ULP and high-performance application scenarios. The first two FLLs provide the clock for the FC domain; one is dedicated to the RISC-V core and the interconnection and memory subsystems. The other one provides the reference clock for the IO peripheral subsystem. Such a choice has been made to decouple the internal FC operating frequency from the one used to clock the IO peripherals, which operate at much lower clock frequencies. The third FLL is used to generate the clock frequency of the general-purpose accelerator domain, while the fourth one is dedicated to the EHWPE accelerator domain.

5.3.3 IO subsystem

The FC domain hosts a vast set of IO peripherals commonly adopted on microcontroller-like SoC. Specifically in Kraken, we find the following programmable IO peripherals:

- (4X) I^2C
- (4X) QSPI
- (4X) universal asynchronous receiver transmitter (UART)
- (1X) camera parallel interface (CPI)
- (1X) DVSI
- (44X) general purpose input output (GPIO)

The chip features 76 digital pads, 44 of which can be used to arbitrarily map any pin of the peripherals mentioned above to a chip pad in an all-to-all crossbar configuration, providing high IO mapping flexibility. The pin mapping is configurable via a memory-mapped register interface connected to the APB. Each peripheral can generate interrupts depending on data transmission events; such events can then be routed to the interrupt controller of the RISC-V core. The GPIOs can be programmed as either input or output, and their value can be set or read by the software running on the FC. An autonomous IO subsystem, the “uDMA” [12], hosts all the peripherals. Such a system can be programmed to autonomously orchestrate data transfers from the peripherals to the L2 memory and vice versa, freeing the RISC-V core from micro-managing such data transfers.

5.3.4 Compute cluster

The cluster domain hosts eight RISC-V cores that implement dedicated extensions like hardware loops, multiply and accumulate, and vectorial instructions for low-precision ML workloads. Each RISC-V core of the cluster accelerator is equipped with a private floating point unit (FPU) implementing common floating-point operations, including FMAC, an essential operation for near-sensor tasks such as filtering and neural networks. The cluster also features a 128 KiB L1 tightly coupled data memory (TCDM). The L1 memory can serve all memory requests in parallel with single-cycle access latency and a low average contention rate (¡10% even on the most data-intensive kernels). Fast event management, parallel thread dispatching, and synchronization are supported by a dedicated hardware block (HW Sync), enabling very fine-grained parallelism and high energy efficiency in parallel workloads. The cluster can be clock-gated with a single core granularity, reducing the dynamic power consumption while waiting for cores synchronization.

5.3.5 Accelerator domain

The EHWPE domain hosts two accelerators. One is the SNE neuromorphic accelerator presented in chapter 4, and the other one is a ternary weight neural network accelerator. Both accelerators can be independently power-gated to reduce the leakage current when not in use. The EHWPE domain operates on two clock domains. One clock is the frequency generated by the FC FLL, which is used to clock the interface logic at the boundary between the accelerators and the main system interconnect. The other clock is generated by a dedicated EHWPE FLL, which is used to clock the accelerator's computing engines. Like many peripherals of the SoC, also the accelerators are programmed via memory-mapped register interfaces.

5.4 Physical implementation

In this section, we provide the physical implementation details of the Kraken chip. Table 5.1 reports the salient characteristics.

The Kraken chip presented in this chapterhas been implemented in the GlobalFoundries 22nm FDX technology process. We used the "Synopsys Design Compiler 2020.09" tool to synthesize the design. The design's physical implementation has been performed with the

Technology	GlobalFoundries 22nm FDX
Chip area	$9mm^2$
L2 memory	1MiB (SRAM) + 64kiB
L1 Memory	128KiB (SRAM)
VDD Range	0.5V - 0.9V
Cluster Frequency	380MHz
EHWPE Frequency	330MHz
FC Frequency	380MHz
Power Range	2mW-200mW

Table 5.1: Physical implementation details

“Cadence Innovus Implementation System 19.1” tool. We used 8T, 20, 24, 28, L, and SL voltage threshold cells for the proposed design. The setup time has been optimized for the slow (SSG) 0.59 V −40 °C process corner, while the hold time has been optimized for the (FFG) 0.72 V 125 °C and −40 °C process corners.

The FC domain has been constrained to operate at 160 MHz in the “worst-case” process corner. At the same process corner, the SNE domain and the CLuster domain have been constrained to operate at a maximum clock frequency of 200 MHz. Figure 5.2 shows the chip micrographs where the different power domains have been highlighted. Moreover, the figure also shows the memory and core area occupation.

Power switching

As mentioned in the previous sections, the Cluster, SNE, and CUTIE domains can be independently power-switched to reduce leakage consumption. This functionality has been implemented by exploiting dedicated power switching cells provided with the GlobalFoundries 22nm process development kit (PDK). Each switchable domain hosts power switched, placed on the standard cell’s row, and supplied from an always-on power supply, allowing disconnecting of the domain-specific cells from the power supply. Only the power switches leakage current contributes to the static consumption of a switched-off power domain. Power switches are controlled by the power management unit (PMU) hosted in the FC domain. Figure 5.3 shows the Kraken

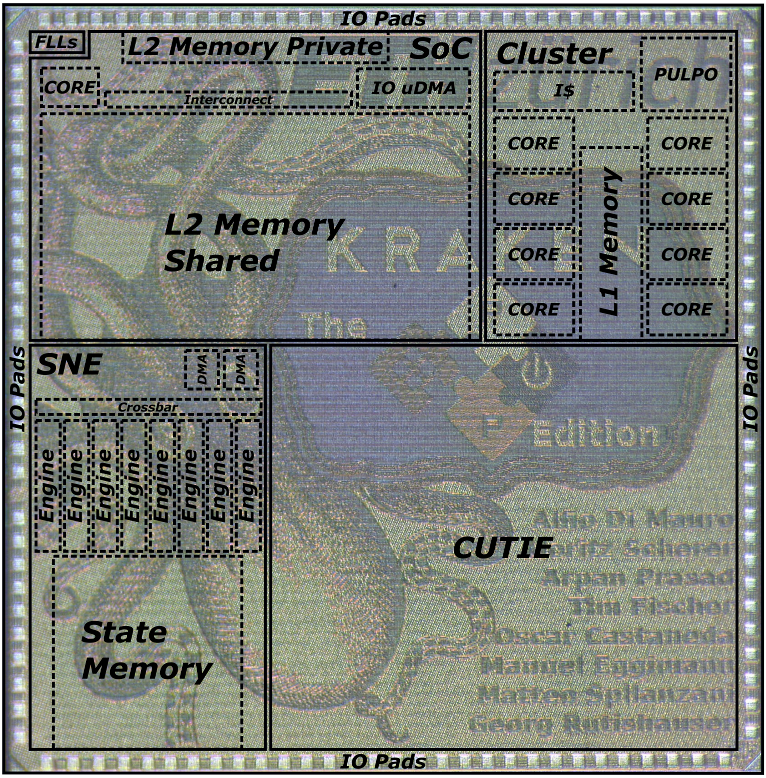


Figure 5.2: Kraken micrograph

power distribution scheme, the power grid structure, and the power switching components.

5.5 Results

In this section, we will complete the discussion on the SNE accelerator performance and energy metrics initiated in chapter 4 by analyzing the performance of the general-purpose cores of the Kraken chip. We will start the presentation of our results by showing the maximum

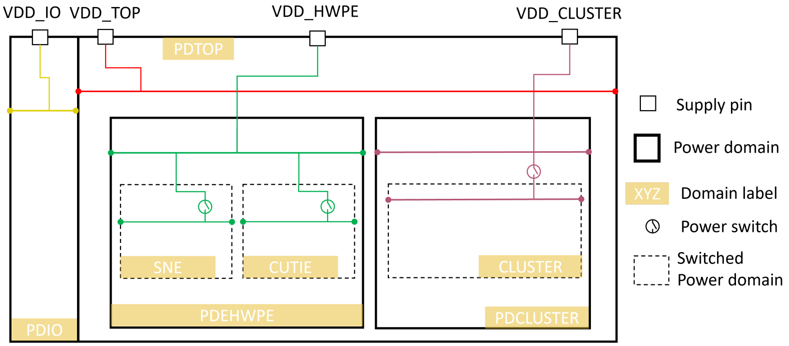


Figure 5.3: Kraken chip power distribution scheme

frequency and power consumption for both the FC and cluster domains. Then we will present a characterization of the event-driven peripheral hosted by the Kraken SoC. Eventually, we will combine those results to provide insights on the total consumption of a complete event-driven pipeline.

5.5.1 General purpose computing engines

The computing engines described in this section are optimized to execute regular, general-purpose intense workloads. To achieve this goal, we identified a few benchmarks representing real-world workloads on the general-purpose RISC-V computing cores. Precisely, we executed a matrix-to-matrix multiplication as a computing primitive capable of simultaneously stressing the interconnect towards the memory and the arithmetic logic unit (ALU) of the cores by performing multiply-accumulate (MAC) operations. To increase the generality of the cores benchmarking operation presented in this section, we executed the matrix-to-matrix multiplication benchmark in a 32bits integer and 32bits floating point variant to cover multiple scenarios. A detailed overview of the benchmarks executed to characterize the power and performance of the Kraken general-purpose cores are reported in Table 5.2.

benchmark	number of operation	CPU cycles	cores
MMUL int32	256*16	1024	8
MMUL fp32	256*16	1192	8
MMUL int8	18*64*4	280	8
MMUL int4	18*64*8	590	8
MMUL int2	18*64*16	1180	8

Table 5.2: Benchmarks executed on the Kraken PULP chip cores

5.5.2 Frequency and Power consumption

This section shows the maximum frequency of the Kraken chip when executing the benchmarks presented in the previous section. We identified the maximum frequency as the frequency at which 10/10 matrix-to-matrix multiplication is executed correctly, at 25 °C. To validate the operation's correctness, the operation results are compared against a reference pre-computed result matrix, pre-loaded into the SoC memory. The maximum frequency of the Kraken chip is reported in Figure 5.4 for different operating voltage points.

The power consumption of the Kraken chip has been measured at the corresponding maximum frequency points. Such frequency operating condition represents the chip's most efficient operating point achievable. By operating the chip at the maximum achievable frequency related to each operating voltage, the constant leakage constitutes the smallest fraction of the total power consumption. the kraken power consumption is presented in Figure 5.5. Both benchmarks show comparable operating frequencies; this result is expected, as the system's critical path does not involve the ALU of the chip; instead, it is identified in the circuitry transmitting the address and data to the memory through the main interconnect. Therefore, there is no fundamental dependency of the maximum frequency with the type of workload executed on the RISC-V cores.

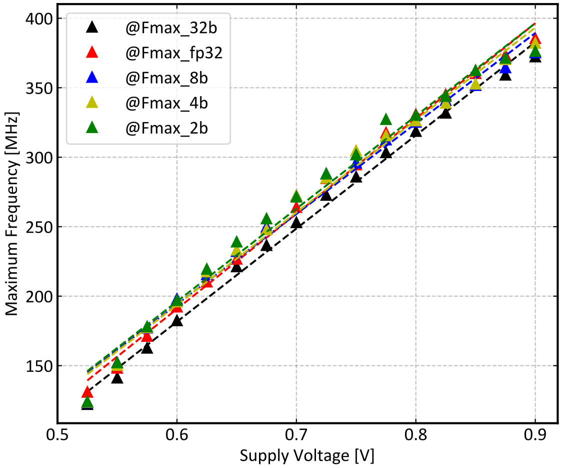


Figure 5.4: Cluster maximum frequency when executing a matrix-to-matrix multiplication for both int32 and float32 benchmarks

5.5.3 Performance and Energy consumption

In this section, we combine previous results to obtain insights into the actual performance of the chip, which are often expressed in terms of the number of operations delivered in a second (OP/s). Such a result is obtained by counting the number of cycles that the cores spend to compute the whole matrix-to-matrix multiplication benchmark, composed of 256x256 MAC operations, and by taking the ratio between the two quantities. Normalizing the result by the time duration of a single cycle makes it possible to derive the number of operations normalized by a reference time of a second. Such metric is reported in Figure 5.6. These results can be interpreted as a proxy to estimate the execution time of a specific application by knowing the total number of operations that compose the application.

Figure 5.7 reports the chip energy efficiency, expressed in terms of the number of operations per second, per unit of power. This metric provides insights on the delivered computational capabilities per unit of power. This metric can be used as a reference to calculate the

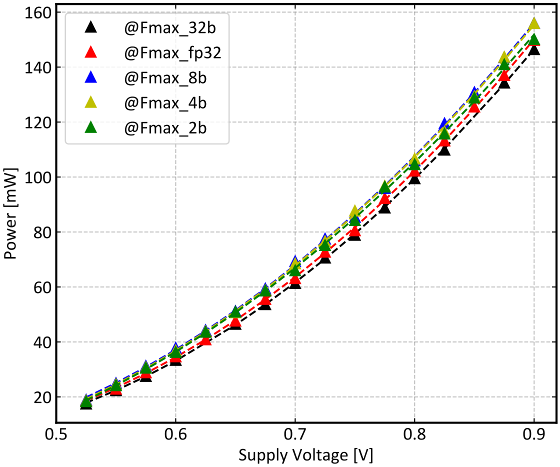


Figure 5.5: Cluster power when executing a matrix-to-matrix multiplication for both int32 and float32 benchmarks

approximate power consumption of an application, starting from the total number of operations.

The last result related to the general-purpose core characterization is the energy per operation (32bit integer of floating-point MAC). Such a result is reported in Figure 5.8 and provides an indication of the total energy cost of a single operation executed on the general-purpose computing cluster. Note that this result includes the cost of the L1 scratchpad memory accesses performed during the benchmark execution.

5.5.4 Event-driven application scenario

In this section, we propose a general application scenario where the various components of the Kraken chip are used to achieve an increasingly higher level of cognitive/computational capabilities to form a "staged inference pipeline". Thanks to the power management capabilities of the chip, the various sub-modules of Kraken can be dynamically turned off to save energy.

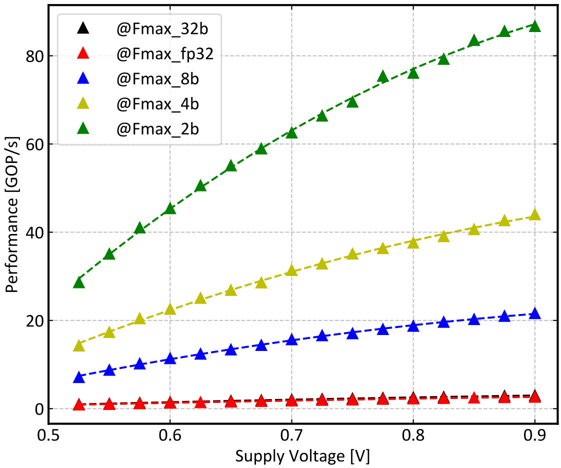


Figure 5.6: Cluster maximum performance when executing a matrix-to-matrix multiplication for both int32 and float32 benchmarks

A typical requirement for many IOT nodes deployed in real application scenarios is the always-on sensor monitoring capability. The Kraken chip can be put in a mode that we call “Always-on energy-proportional” (AOEP) to address this use case. In this mode, the Kraken chip operates in an always-on mode to autonomously collect data from the IO peripherals. As we have seen in chapter 2, in this mode, an IO subsystem like the one hosted by Kraken can consume power in the order of a few hundreds of µW, and the entire FC domain can consume less than 2 mW. In this mode, all the domains of the chip are power-gated, the FC domain is active, but the core spends most of the time in a “wait-for-interrupt” state, where it is clock gated.

The autonomous IO subsystem, i.e., the uDMA, can generate interrupts based on complex conditions. For example, in the case of visual event acquisition from the DVSI, interrupts can be generated if events occur in a specific region of interest of the input event frame or if a specific event rate is exceeded in a programmable time interval. If such conditions occur, the RISC-V core of the FC domain exits

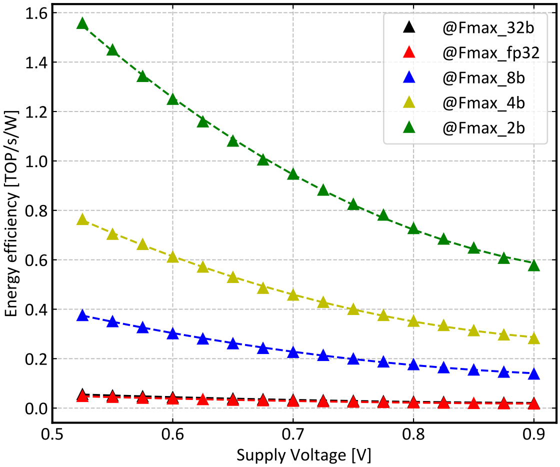


Figure 5.7: Cluster energy efficiency when executing a matrix-to-matrix multiplication for both int32 and float32 benchmarks

the clock-gated state and handles the interrupt. In this mode, which we will call “Smart trigger” (ST), lightweight cognitive algorithms can be executed at the FC domain in a power budget of 15 mW. This architecture allows a smart triggering mechanism to power up the accelerator domains (Cluster and EHWPE) and to offload complex cognitive tasks to both SNE and the RISC-V-based compute cluster. Algorithms running at the level of the FC, serving as smart triggering mechanisms, aim to reduce the false positive rate, avoid the unnecessary accelerator domain power-up, and the consequent energy consumption overhead.

If the smart triggering criteria are met, the complex cognitive task can be executed on the incoming data by offloading a more powerful algorithm, e.g., DNNs or SNNs inference, on the accelerator domains. We named this mode “On-demand Cognitive Computing” (ODCC). In this power mode, the Kraken chip can execute complex DNN, SNN, or a combination of them in a power budget of 320 mW. This mode aims at performing highly accurate and energy-hungry predictions, or more generally, high quality of service. Figure 5.9 reports a visual

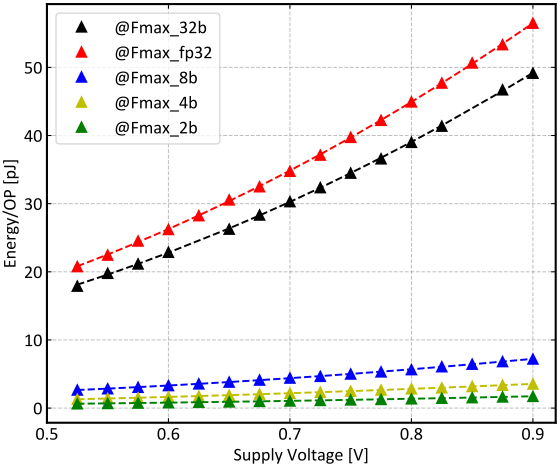


Figure 5.8: Cluster energy per operation when executing a matrix-to-matrix multiplication for both int32 and float32 benchmarks

representation of the modes described in the proposed application scenario

Voltage and frequency scaling considerations

It is important to remark that across the whole "staged cognitive pipeline", conventional power reduction techniques like DVFS can be easily adopted. All the domains are equipped with CDC logic and clocked by independent on-chip clock generators. Moreover, the Kraken chip is equipped with IO peripherals that can be employed to control external I^2C voltage regulators. Therefore it is possible to couple the cognitive pipeline stages activation with software-controlled power management strategies to carefully tailor the power profile of a deployed application to the desired one.

Comparison with the state of the art

In this section, we compare the Kraken chip with similar IoT platforms. Table 5.3 reports SoA chips that target the same

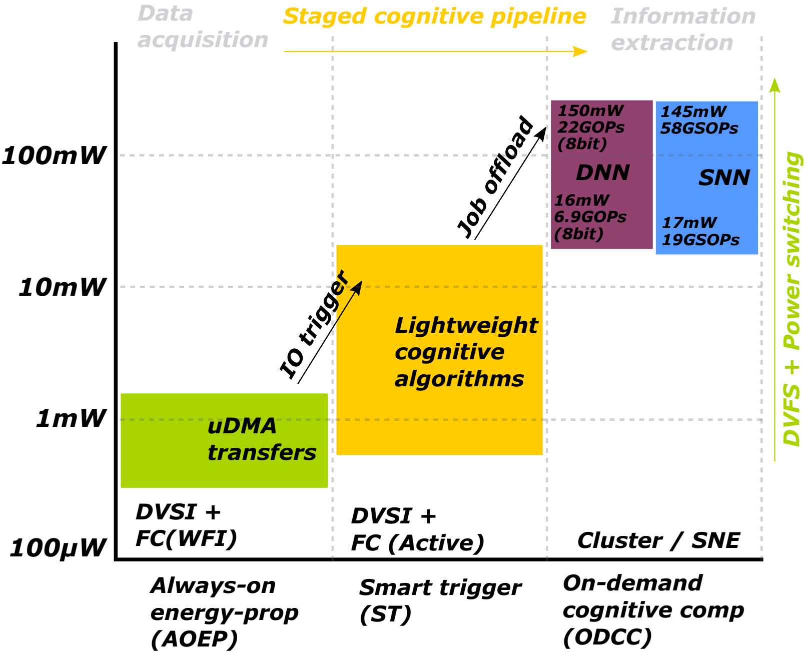


Figure 5.9: Kraken staged inference pipeline

application space. Among those architectures, Kraken shows the highest floating-point energy efficiency. Also, the reported integer energy efficiency is comparable with the SoA devices having a very similar architecture. Additionally, compared to all the presented architectures, Kraken natively supports a direct connection with event-based cameras, enabling high energy saving opportunities in video applications. Moreover, Kraken is equipped with an embedded neuromorphic accelerator, SNE. To the best of our knowledge, Kraken represents the first general-purpose IoT prototype hosting a SoA neuromorphic low-precision computing engine on the side of a powerful general-purpose computing cluster, practically enabling the highly efficient execution of both DNN workloads and neuromorphic tasks like SNNs at the edge.

	RISC-V VP Schmidt et al. ISSCC 2021	SleepRunner Bol et al. JSSC 2021	SamurAI Miro-Panades et al. VLSI 2020	Mr. Wolf Pullini et al. JSSC 2019	Vega Rossi et al. ISSCC 2021	**Kraken (This work)**
Technology	FinFET 16nm	CMOS 28nm FD-SOI	CMOS 28nm FD-SOI	CMOS 40nm	CMOS 22nm FD-SOI	CMOS 22nm FD-SOI
Area	24mm^2	0.68mm^2	4.5mm^2	10mm^2	12mm^2	9mm^2
Type	Vector processor	MCU	Heter. MCU	Parallel MCU	Parallel + Heter. MCU	Parallel + **Neuromorphic** MCU
Applications	DSP	IoT GP	IoT GP, NSAA, DNN	IoT GP, NSAA	IoT GP, DNN, NSAA	IoT GP, NSAA, **Mixed-prec. DNN, EDC, SNN**
CPU/ISA	RV64GC	CM0DS Thumb-2 subset	1xRISCY RVC32IMFXpulp	1xRISCY + 8xRISCY RVC32IMFXpulp	1xRISCY+9x RISCY RVC32IMFXpulp + SF	1xRISC-V + 8xRISC-V-NN RVC32IMFXpulpNN
SRAM	4.5MB	64kB	464 kB 40kB s.r.	64 kB (L1) 512 kB s.r. (L2)	128 kB (L1) 1600 kB s.r. (L2)	128 kB (L1) 1MiB (L2)
Voltage	0.55V-1V	0.4V-0.8V	0.45V-0.9V	0.8V-1.1V	0.5V-0.8V	0.5V-0.9V
Max frequency	1.44GHz	80MHz	350MHz	450MHz	450MHz	390MHz
Power range	n.a - 4W	5.4-320uW	6.4uW-96mW	72uW-153mW	1.7uW-49.4mW	2mW-320mW FC(WFI) to Cluster + SNE (Active)
int Perf. int Eff @ Perf.	-	31 MOPS 97 MOPS/mW @18.6 MOPS	1.5 GOPS 230 GOPS/W @110 MOPS	12.1 GOPS 190 GOPS/W @ 3.8 GOPS	15.6 GOPS (8b) 614 GOPS/W (8b) @7.6 GOPS (8b)	**21.5 GOPS (8bit) @ 0.9V/390MHz** 380 GOPS/W (8bit) @6.9 GOPS (8bit) @ 0.52V/120MHz **1558 GOPS/W (2bit)** **@87 GOPS (2bit) @ 0.52V/120MHz**
float32 Perf. float32 Eff. @Perf.	n.a 92.3 GFLOPS/W n.a	-	-	1 GFLOPS 18 GFLOPS/W @350 MFLOPS	2 GFLOPS 79 GFLOPS/W @1 GFLOPS	**2.7 GFLOPS (0.9V/390MHz)** 50 GFLOPS/W @1 GFLOPS (0.52V/120MHz)

Table 5.3: State of the art comparison table

5.6 Conclusion

This chapter presented Kraken, a SoC targeting cognitive application at the edge. We showed how to integrate successfully, in the same SoC architecture, a general-purpose computing cluster, and a neuromorphic computing accelerator. Moreover, the Kraken chip also hosted the DVSI peripheral presented in chapter 2, enabling an end-to-end energy proportional event-driven computing pipeline when using the accelerator presented in chapter 4. Our analysis characterized the general-purpose computing core capabilities in terms of integer and floating-point performance and energy efficiency at multiple operating points. Kraken showed a floating-point and integer energy efficiency and performance comparable with the SoA. In a further analysis, we used the results of the first part of the analysis to outline a possible application scenario where Kraken is used to implement an event-driven computing pipeline, practically showing how the different parts of the system can be dynamically switched on to execute energy-efficient applications.

Chapter 6

Event-driven SNN deployment

6.1 Introduction

This chapter represents the last piece of the puzzle toward a full event-driven digital platform utilization at the extreme edge. It aims to walk the reader through the required software abstraction layers that need to be developed to build a "vertical software stack" that allows to deploy a complete application. Specifically, throughout this chapter, we will discuss how an application can be deployed on the target hardware platform that we presented in chapter5 and that hosts a dedicated event-driven computing engine like the one presented in chapter4.

In the previous chapters, we have seen that a few peculiar features characterize the data produced by event-driven sensors, namely the high data sparsity coupled with the extremely low data redundancy when encoding useful information [45]. As we have demonstrated in chapter2 and 4, exploiting such data sparsity could lead to significant advantages in terms of system-level energy efficiency; however it posed new challenges both from the point of view of the data acquisition from event-based sensors and the event-driven data processing hardware architecture. This thesis has proposed a few architectural solutions for data acquisition and data processing that can efficiently solve such challenges.

Similar to what we have presented in terms of hardware architectural solutions, this chapter will go through the challenges of deploying event-driven applications from a software point of view. Our analysis will be restricted to deploying SNNs operating on sparse input data streams. We will start our analysis by discussing the general challenges associated with training SNNs. The first step of deploying a machine learning application on any platform starts with selecting a specific neural network model and training such a model on a data set that represents the actual target task to be solved. Notably, SNNs require a set of mathematical measures to circumvent the non-differentiable nature of the spiking neuron membrane potential expression. In this regard, machine learning communities have made significant progress in the very last few years. Therefore, we will explore the most promising approaches to train SNNs, ideally achieving the performance of the DNN counterpart.

Then, we will continue our discussion on constructing a vertical software stack for event-driven applications by focusing on those data precision reduction strategies commonly adopted when deploying machine learning applications on embedded platforms. On such platforms, similar to what we have seen in chapter5, the amount of memory is limited to several hundreds of kilobytes or a few megabytes. Therefore it is crucial to reduce the memory footprint of the deployed algorithms, both in the trained parameters and intermediate results, that ideally should be kept on-chip to reduce the energy associated with the algorithm execution. To tackle this challenge, we will discuss a possible quantization approach suitable for application based on SNNs. Reducing the data representation precision of the deployed algorithm is a well-known yet very effective approach adopted in most embedded applications deployed on low-power hardware platforms. The main goal of such a low-precision data representation is to reduce the memory footprint of the parameters and the intermediate results of the algorithm while ideally preserving the same “quality of service” or, possibly, degrading it by an acceptable margin which depends on the application; we have discussed this aspect in more in-depth in chapter3.

The main challenging aspect when a quantization strategy is applied to SNNs is represented by the complex internal neuron dynamic, as opposed to the simple accumulation resulting from the convolution performed in classical DNNs. Besides the usual synaptic weight and output features quantization, in SNNs, the internal neuron dynamics need to be also represented in low arithmetic precision. As we have seen in chapter4, the arithmetic precision of the synaptic weights and the neuron dynamics are not necessarily the same; in SNE, we had 4 bits represented synaptic weights and 8 bits to represent the internal neuron membrane potential. As can be observed for classical DNNs, also in the case of SNNs, we expect better results in final classification accuracy when the network is trained in a quantization-aware training framework. To achieve this goal and preserve the original classification accuracy of an SNN through the quantization process, it is essential to try to accurately model, already at training time, the exact low precision dynamics of the neural network neurons. This aspect is key to minimizing the mismatch between the algorithm execution performed at training time to evaluate the accuracy of the low-precision network and the one executed on the actual platform. The ultimate goal of accurate modeling of the low-precision algorithm execution is to give the optimization process the chance to compensate for the accuracy degradation caused by the reduced numeric precision.

After discussing the strategy for reducing the data precision, we will examine how to exploit the current available deep learning framework to perform quantization-aware training of SNNs. Specifically, we will describe how to use the existing deep learning libraries and training framework running GPUs, e.g., PyTorch[1]., for the purpose mentioned above. Then, we will discuss the remaining steps for exporting a trained and quantized SNN into a standardized format and deploying it on the target embedded platform through executable code auto-generation. Eventually, we will present a sample SNN deployment to the Kraken platform presented in 5, as well as a short analysis of the inference energy and latency on such a platform.

The main contribution presented in this chapter can be summarized as follows:

---

[1] Open-source framework primarily developed at Facebook's AI labs (www.pytorch.org).

- We developed a complete supervised training framework for quantized SNNs, by combining a SoA back-propagation learning strategy and SoA quantization approaches like PACT [111]. Moreover, we integrated into such a framework a bit-accurate model of the quantized neuron dynamics implemented by the SNE accelerator, such that quantization-aware training targeting the platform presented in chapter4 could be performed. Such training framework relies on existing open-source, well-established training frameworks for conventional DNNs like PyTorch.

- We developed the toolchain that allows deploying quantized SNNs on the embedded platform presented in chapter5 and equipped with the dedicated accelerator presented in chapter4. Such deployment SNN toolchain builds on top of an open-source RISC-V compiling toolchains and hardware abstraction layer (HAL) publicly available, and auto-generates code for the Kraken platform.

- We evaluated the energy consumption and latency of an SNN trained with the methodology presented in this chapter and deployed on the Kraken platform.

6.2 Related work

Over the last years, SNNs have emerged as a potential low-energy model-free algorithm candidate to solve complex tasks [176]. Indeed, such ANN category has often been depicted as the “third generation” AI. Among the most relevant features that make such neural network class a viable substitute for current DNN algorithms is the intrinsic sparse nature of SNNs [177]. As we have already discussed throughout this thesis, data sparsity represents one of the most appealing features whose exploitation promises to reduce the energy consumption of a data processing algorithm by several orders of magnitude [46]. To be able to deploy SNNs to solve a real-world problem, we need to establish a portfolio of robust training strategies, optimization methodologies, as well as a series of sparse benchmarks to be used as a reference to evaluate SNN performance [178], similarly to what has happened for conventional DNNs on well-known problems [2].

6.2.1 applications for SNNs

When discussing potential applications for SNNs, we have to distinguish between what we will call “conventional” well-known DNN problems and a series of “emerging” problems where pioneering studies have shown that SNNs could deliver significant improvement in terms of energy efficiency and latency against the SoA algorithmic approach. In the first category, we can undoubtedly mention classification problems like MNIST, CIFAR10/100, and ImageNet and the more event-based oriented like IBM-DVSGesture, NMNIST NCIFAR10, and many others. In the second category, we can mention simultaneous localization and mapping (SLAM) least absolute shrinkage and selection operator (LASSO), and fast iterative shrinkage-thresholding algorithm (FISTA).

This chapter will focus on the first category of problems for which well-established performance assessment metrics are already available. Rather than achieving SoA performance in solving such algorithms employing SNNs, this chapter aims to demonstrate how to compose an end-end deployment pipeline for SNN on a fully custom event-driven platform.

6.2.2 Unsupervised SNN training

A popular way of training SNNs is to follow the “unsupervised training” approach. A well-known algorithm to perform unsupervised learning is the spike time dependent plasticity (SDBP) [179, 180], and many variants of such algorithms have been proposed over time [181, 182]. Such strategies often rely on neural networks’ self-organizing principles [183, 184]. Despite the effort made by research to emulate and reproduce biologically plausible learning strategies, such approaches do not reach the same SoA performance reported when supervised approaches are used.

6.2.3 Supervised SNN training

Training CSNNs in a supervised way poses new challenges compared to standard DNNs like feedforward CNNs. In DNNs, the information is conveyed from one layer to the next using differentiable activation functions. This differentiability property enables the application of a gradient descent-based algorithm to perform supervised learning. Instead, in SNNs, the information is described by temporal distributions of spikes, which are inherently non-differentiable due to the spike generation dynamics, which prevents the application of gradient descend-based learning.

This subsection will describe some of the most known training frameworks for SNNs that use backpropagation. This discussion generally highlights the main feature of each framework and has not aimed to extensively cover all the details of each framework.

The spike layer error reassignment (SLAYER) [185] algorithm has shown promising results in overcoming the non-differentiability issue featured by SNNs. Weights are updated by exploiting a gradient surrogate that enables back-propagation for SNNs. Therefore, such a framework would be an excellent candidate for training SNNs in a supervised way. However, the neuron dynamics adopted in SLAYER is the spike response model (SRM), which is considerably complex to be implemented on an embedded hardware platform. Hence, despite the good performance of the SLAYER approach, its applicability as a training framework for SNNs to deploy on ultra-low-power computing platforms remains limited.

An interesting alternative is represented by the “Nengo” framework, a simulation and training framework for SNNs composed of various sub-modules. This feature makes “Nengo” very interesting in terms of versatility. Indeed the “Nengo” framework is composed of a front-end module, which exposes the API to describe the structure of the network and its parameters, and a back-end module, which is the module in charge of practically running the SNN simulation on the actual platform. Such structure is very effective, as it allows to transparently run the same network on a plethora of different hardware platforms in a transparent way and with minimal changes. However, the main limitation of the “Nengo” framework is the strategy to circumvent the non-differentiability of the SNN neuron dynamics.

In such a framework, at the training phase, the neurons are converted into rate-based neurons [186], practically restricting the application field to those cases where such spike encoding can be adopted. Moreover, such a strategy has several other limitations, as correctly demonstrated by [187].

A more generalized strategy to overcome the limitation originating from the LIF spiking neuron non-differentiability is the spatio-temporal back propagation (STBP) algorithm proposed by Shi et al. in [188]. Such an approach uses a gradient surrogate around the non-differentiable point of the neuron membrane potential. These mathematical artifacts practically allow using commonly adopted gradient-descent-based optimization techniques to train a neural network and feed-forward SNNs with convolutional or linear synaptic connections. By far, the approach presented by Shi seems the most effective and promising to ultimately solve the SNN supervised training issue, practically making SNN training not very different from conventional DNN training.

In the framework proposed in this chapter, we will adopt the gradient surrogate proposed in [188], and we will build on top of this approximation to construct an end-end training framework capable of training quantized SNNs.

6.2.4 Fake quantization

The second ingredient for deploying a quantized neural network is a quantization strategy. Current neural network topologies can feature billions of parameters typically represented as single-precision floating-point values. This choice is notably dictated by the need to achieve the numerical stability of the algorithms used to train the network. In this framework, the memory footprint of the neural network might not be compatible with the usual amount of memory with which embedded platforms are typically equipped. Contrarily to TPUs or GPUs, the embedded platform has low available memory, i.e., typically a few hundred of kB or a few MB.

To reduce the memory footprint of DNNs deployed to the embedded platform, quantization is a common technique that leads to a low or zero quality of result degradation. Quantization can be performed in many different flavors; more straightforward approaches can reduce

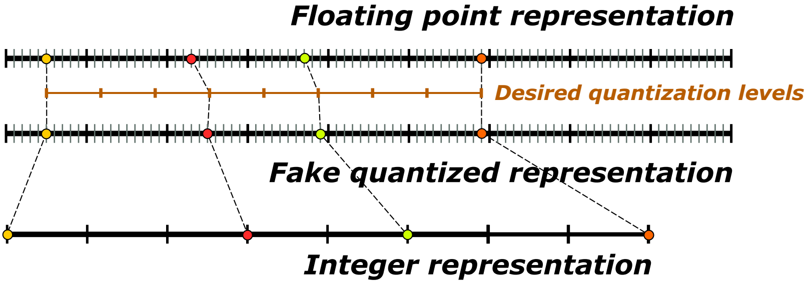

Floating point representation
Desired quantization levels
Fake quantized representation
Integer representation

Figure 6.1: graphical representation of a fake quantized value interval range

the precision from a 32bits or 64bits floating-point representation to a smaller 16bits float representation. More aggressive quantization strategies can start from the full precision representation of the network parameters and target a final 32bits or even lower bitwidth integer representation. As we have seen in chapter3, the extreme case is represented by BNNs.

In this process, “fake quantization” represents a transitional step between the full precision parameter representation and an integer representation of the desired bit-width. In the “Fake quantized” representation, the network parameters are still represented by full precision values. However, each parameter is rounded to a discrete number of values corresponding to the representable integer values in the target bitwidth. Such representation allows quantization-aware training by employing the same optimization algorithms to train full precision networks.

6.2.5 True-quantization and integerization

Once the network has been trained in the “fake-quantized” regime, parameters, i.e., the weights and neuron threshold values, can be converted into true integer values, ideally with no loss with respect to the fake quantized representation of the network parameters.

6.3 Vertical software stack

This section describes the steps required to enable quantization-aware training for SNNs. Our focus is to deploy neural networks to SNE. A block diagram representation of the deployment flow for such platforms is reported in figure 6.2.

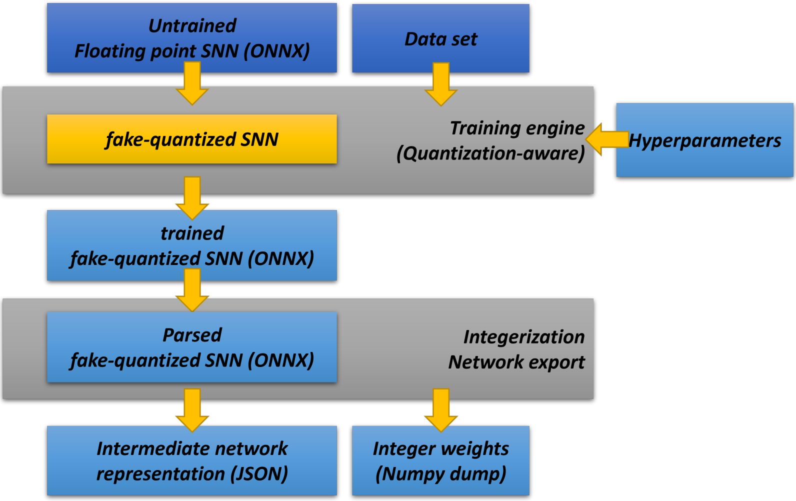


Figure 6.2: block diagram representation of the vertical deployment software stack for SNNs

6.3.1 Neural network quantization-aware training

The first component of the vertical software stack is a quantization-aware training engine. The ultimate goal of this module is to produce a trained network in a standard format like “ONNX” where the weights and the neuron dynamic parameters are represented with fake-quantized values. Here we need to remark that, contrarily to what happens for conventional DNNs, where the accumulation process is trivial, and the inference reduces to a fixed amount of “MAC” operations, in the case of SNNs, the neuron dynamic is considerably more complex; it involves non-linear operations such as exponential decay. Therefore it is crucial to model, as accurately as possible, the neuron dynamic of the chosen elementary neuron, the ALIF in our case, such that the final network behaves ideally in the same way when executed on a GPU and the actual target neuromorphic platform.

The main challenge when accurately modeling the ALIF neuron dynamic is represented by the different ways operations are performed in the GPU running the SNN software implementation used during training and how operations will be executed on the real accelerator. The most convenient and PyTorch-compatible way of performing the SNN "forward pass" during the training phase is to represent the time-distributed input, intermediate, and output features as binary-valued 4-dimensional tensors (T,C,H,W). This representation is not different from a canonical features representation used in DNNs, with the only exception that in SNNs, we find an additional time dimension. In this representation, each non-zero value represents a spike occurring at a specific position in the H, W, and C dimensions and a specific time in the T dimension. In the case of a convolutional synaptic input connection, we can consider the NxN convolutional kernel, which in the case of the SNE accelerator is limited to 3x3 kernels, as the receptive field of each output neuron. The convolutional kernel is repeated and applied for each 1-time step-wide 2D plane of the input tensor along the T dimension. From one step to the successive one, the membrane potential of the neurons is retained, and synaptic contribution is added to the previous value. This algorithm limits the possibility of parallelizing the computation over the T dimension, as the order of the time steps is relevant and can not be changed. Therefore the "forward pass" of an SNN remains a sequential calculation over the time dimension.

Convolutional layer

The "inference" process operations happen radically differently on the SNE accelerator. Specifically, in SNE, there is no fundamental notion of tensor. Instead, single events are broadcasted to all output neurons and, utilizing a filtering mechanism in front of each output neuron, each input event is weighted with the corresponding kernel weight and accumulated on the target output neurons membrane potential. No multiplication operations are executed, as the corresponding weight is already the total synaptic contribution associated with an input event, and the event is represented with a binary value. Moreover, unneeded operations are skipped in the SNE accelerator, and a LUT is used to compute the aggregated delay factors related to multiple time steps. The PyTorch implementation of the inference process accurately reproduces the LUT-based decay mechanism implemented in the real accelerator. This is achieved by storing an additional tensor to trace the time of the last update of each neuron, such that the correct aggregated decay coefficient is retrieved for each output neuron.

Linear layer

The considerations we have done so far for the PyTorch convolutional layer implementation also hold for the linear (fully connected) layers. The neuron dynamic implementation used for this layer is the same as the one used for the convolutional layer. The only difference is the synaptic connection scheme.

Pooling layer

A third layer commonly constituting a neural network is the Pooling layer. Because of the binary nature of input and output feature maps of SNNs, the more straightforward implementation is represented by a ”Max Pooling” layer. This layer can be implemented by simply performing a step-wise boolean “OR” operation between the output feature maps falling into the boundaries of the pooling kernel size.

Fake-quantized implementation

All the building blocks presented so far were designed so that fake quantization could be applied to both the neuron dynamic and the weights. The neuron dynamics of the SNE ALIF neurons use 8bits signed integer values to represent the membrane potential variable and 4bits signed integers for the synaptic weights. The quantization interval for both the neuron membrane potential and the weights is obtained by enforcing an almost-symmetric interval of representable values centered around 0. 255 intervals bounded by floating-point numbers can be defined for the membrane potential. Similarly, for the synaptic weight, 16 values are defined

Network training

Once all the building blocks that compose an SNE compliant implementation of the SNN inference are defined, the SNN network training can be implemented by exploiting the functionalities exposed by the PyTorch framework for conventional DNNs training. For example, an important one is the flexible data loading capability and the automatic differentiation of most of the tensor operators used to compose the neuron dynamics. The algorithm followed for the training essentially reproduced what was presented by presented in the work of Wu et Al. [188].

Neural network integerization

Thanks to the simplicity of the constraints enforced during the quantization-aware training, i.e., the almost symmetric range of values, the integerization process is trivial. It can be performed by simply linearly scaling the floating-point interval range obtained during the fake-quantized training to the integer bounds representable on the accelerator. In the case of the membrane potential, this translates into an interval of [-128, 128). Similarly, the floating-point bounded range translates to [-8,8) for the synaptic weight.

The quantization-aware training was validated on several network topologies and data sets. In figure List of Figures6.3 we provide an example of a simple network topology trained and quantized with the proposed approach. The plot compares the floating-point implementation, the fake quantized network, and the integer-valued parameters network after the integerization process.

The data set we used as a proof of concept for this experiment is the IBM-DVS-Gesture data set[2]. We used 80% and 20% of samples for the training and test set. The goal of this experiment was not to reach the SoA performance on such a dataset but rather to demonstrate the feasibility of the proposed approach.

---

[2] https://www.research.ibm.com/dvsgesture/

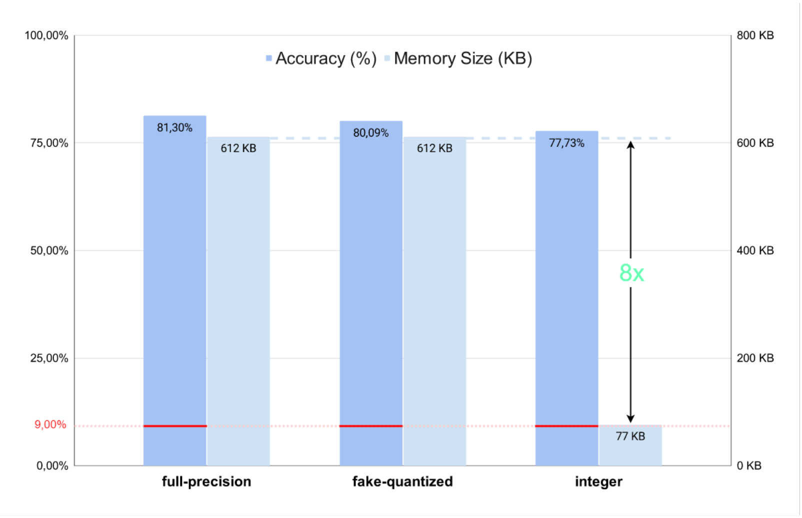


Figure 6.3: Accuracy comparison of a floating point, fake quantized and true quantized implementation of a sample 5-layers convolutional neural network topology SNN

6.3.2 PULP RISC-V toolchain

As we have observed in the chapter5, all the operations of the Kraken chip are orchestrated by the RISC-V processor around which the SoC is built. This module is the main controller for all the peripherals, including the RISC-V-based general-purpose computing cluster and the accelerator subsystem that hosts the SNE accelerator. The deployment of any application somehow relies on the platform's capability to execute general-purpose tasks like data acquisition from external peripherals, data pre, post-processing, or data movement. A common approach to deploying a hardware-accelerated application is to rely on a low-level HAL, which allows abstracting some of the complexity introduced by dedicated hardware platforms and providing simple and intuitive functions to the user or higher layers of the software stack.

In the Kraken platform, this task is offloaded to the "pulp-runtime"[3], which can be considered a minimal set of low-level API which allows governing the entire SoC. A user can compose a complex application manually by explicitly describing the desired algorithm in the "C" language, which calls the aforementioned API functions. Similarly, the very same API functions can be used as an interface for an auto-generated application, like in the case of the deployment flow described in this chapter.

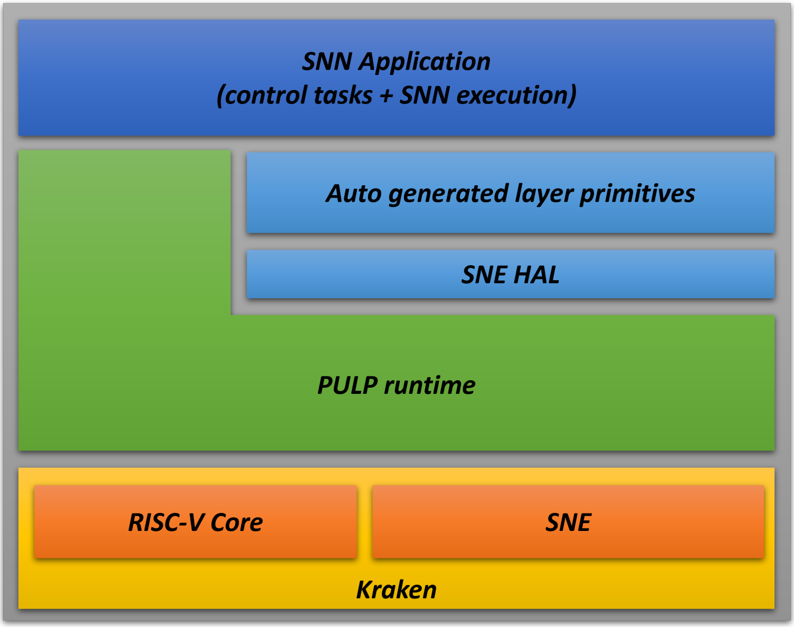

SNN Application
(control tasks + SNN execution)
Auto generated layer primitives
SNE HAL
PULP runtime
RISC-V Core
SNE
Kraken

Figure 6.4: visual representation of the low-level modules composing the software stack SNNs

The programming code, which constitutes the "firmware" application executed by the RISC-V core, is automatically compiled into executable machine instruction using a compile toolchain, i.e., the "PULP RISC-V toolchain". Figure6.4 shows a representation of

---

[3] link to the PULP runtime

the software stack executed on the Kraken platform. This module represents the lowest layer of the vertical stack for deploying SNNs on the Kraken platform.

The conjunction point between the upper layers of the software stack, i.e., the ones in charge of training and quantizing the network, and the lower layers, i.e., the ones in charge of converting the application into a set of executable machine instructions, is constituted by two intermediate software layers that expose a dedicated SNE software interface.

The first intermediate software layer, the “SNE HAL”, contains the definition of all the accessible configuration registers of the SNE accelerator and the wrapper function to program, i.e., read and write the content of such registers. This layer directly uses the “pulp-runtime” API function calls to access the system bus and physically perform the accelerator register configuration. The second intermediate software layer builds on top of the previous one to compose a set of SNE specific primitives that allow configuring the accelerator in the desired mode.

6.3.3 SNE software primitives

This section provides a few relevant examples of such SNE primitive functions. This section aims to show the reader some practical examples of the SNE programming. The SNE software primitives can be divided into three categories described in the following sections.

DMAs API

Each `DMA`can be configured to move data from/to Kraken’s L2 memory. The streamers can load both the layer’s weights and inputs and store the events generated by the engines back in the memory. Listing 6.3.3 reports a sample function that allows setting up a memory transfer. The function serves as a wrapper for a series of “pulp-runtime” register write function calls, configuring the registers of the SNE accelerator.

```c
static inline void hal_sne_init_streamer(
  uint32_t streamer,
  uint32_t l2saddr,
  uint32_t l2step,
  uint32_t l0saddr,
  uint32_t l0step,
  uint32_t transize) {

  pulp_write(SNE_SYSTEM_BASE_ADDR +
    SNE_SYSTEM_CLOCK_CFG_MAIN_CTRL_I_0_OFFSET       + (
    streamer) * 4, 0);

  pulp_write(SNE_SYSTEM_BASE_ADDR +
    SNE_SYSTEM_CLOCK_CFG_TCDM_START_ADDR_I_0_OFFSET + (
    streamer) * 4, l2saddr);
  pulp_write(SNE_SYSTEM_BASE_ADDR +
    SNE_SYSTEM_CLOCK_CFG_TCDM_ADDR_STEP_I_0_OFFSET  + (
    streamer) * 4, l2step);
  pulp_write(SNE_SYSTEM_BASE_ADDR +
    SNE_SYSTEM_CLOCK_CFG_TCDM_END_ADDR_I_0_OFFSET   + (
    streamer) * 4, 0x00000000);
  pulp_write(SNE_SYSTEM_BASE_ADDR +
    SNE_SYSTEM_CLOCK_CFG_TCDM_TRAN_SIZE_I_0_OFFSET  + (
    streamer) * 4, transize);

  pulp_write(SNE_SYSTEM_BASE_ADDR +
    SNE_SYSTEM_CLOCK_CFG_SRAM_START_ADDR_I_0_OFFSET + (
    streamer) * 4, l0saddr);
  pulp_write(SNE_SYSTEM_BASE_ADDR +
    SNE_SYSTEM_CLOCK_CFG_SRAM_ADDR_STEP_I_0_OFFSET  + (
    streamer) * 4, l0step);
  pulp_write(SNE_SYSTEM_BASE_ADDR +
    SNE_SYSTEM_CLOCK_CFG_SRAM_END_ADDR_I_0_OFFSET   + (
    streamer) * 4, 0x00000000);

}
```

C-XBAR API

The `C-XBAR`connects SNE `SLs` with the two `DMAs`. Functions like the one reported in Listing 6.3.3 configure the internal configuration scheme among the various modules of SNE. For example, in this case, the crossbar configures the two internal sub-crossbars to redirect the output of one streamer to the input of all the SNE `SLs`. This function is handy during weights loading, where, for example, the two streamers can be used simultaneously to load two different sets of weights to the engines.

```
static inline void hal_sne_crossbar_all_engine(){

  // configuration for the crossbar stage 0
  pulp_write(SNE_BASE_ADDR +
    SNE_BUS_CLOCK_CFG_XBAR_STAGE_0_0_OFFSET, 0x03def7bc);
  pulp_write(SNE_BASE_ADDR +
    SNE_BUS_CLOCK_CFG_XBAR_STAGE_0_1_OFFSET, 0x7bdef7bc);
  pulp_write(SNE_BASE_ADDR +
    SNE_BUS_CLOCK_CFG_XBAR_STAGE_0_2_OFFSET, 0x7bdef10c);
  pulp_write(SNE_BASE_ADDR +
    SNE_BUS_CLOCK_CFG_XBAR_STAGE_0_3_OFFSET, 0x214c7424);
  pulp_write(SNE_BASE_ADDR +
    SNE_BUS_CLOCK_CFG_XBAR_STAGE_0_4_OFFSET, 0x84653a54);
  pulp_write(SNE_BASE_ADDR +
    SNE_BUS_CLOCK_CFG_XBAR_STAGE_0_5_OFFSET, 0x7bdef7bc);
  pulp_write(SNE_BASE_ADDR +
    SNE_BUS_CLOCK_CFG_XBAR_STAGE_0_6_OFFSET, 0x7bdef7bc);
  pulp_write(SNE_BASE_ADDR +
    SNE_BUS_CLOCK_CFG_XBAR_STAGE_0_7_OFFSET, 0x7bdef7bc);

  // configuration for the crossbar stage 1
  pulp_write(SNE_BASE_ADDR +
    SNE_BUS_CLOCK_CFG_XBAR_STAGE_1_0_OFFSET, 0x7d800000);
  pulp_write(SNE_BASE_ADDR +
    SNE_BUS_CLOCK_CFG_XBAR_STAGE_1_1_OFFSET, 0x000007bc);
  pulp_write(SNE_BASE_ADDR +
    SNE_BUS_CLOCK_CFG_XBAR_STAGE_1_2_OFFSET, 0x7bdef844);
  pulp_write(SNE_BASE_ADDR +
    SNE_BUS_CLOCK_CFG_XBAR_STAGE_1_3_OFFSET, 0x94e957bc);
  pulp_write(SNE_BASE_ADDR +
    SNE_BUS_CLOCK_CFG_XBAR_STAGE_1_4_OFFSET, 0x7bdef7bc);
  pulp_write(SNE_BASE_ADDR +
    SNE_BUS_CLOCK_CFG_XBAR_STAGE_1_5_OFFSET, 0x7bdef7bc);

  //config barrier and synch
  pulp_write(SNE_BASE_ADDR +
    SNE_BUS_CLOCK_CFG_XBAR_BARRIER_I_OFFSET, 0x00001555);
  pulp_write(SNE_BASE_ADDR +
    SNE_BUS_CLOCK_CFG_XBAR_SYNCH_I_OFFSET, 0x3FFF);
}
```

PE API

API functions like the one reported in Listing 6.3.3 control the configuration of the number of channels or neurons, firing threshold, resting voltages, time scaling, and other configurable parameters of the engines.

```c
static inline void hal_sne_init_sequencer(
  uint32_t slice,
  uint32_t saddr,
  uint32_t eaddr){

  pulp_write(SNE_BASE_ADDR + SNE_ENGINE_CLOCK_CFG_ADDR_STEP_I_0_OFFSET  + slice * 4, 0x1);
  pulp_write(SNE_BASE_ADDR + SNE_ENGINE_CLOCK_CFG_ADDR_START_I_0_OFFSET + slice * 4, saddr);
  pulp_write(SNE_BASE_ADDR + SNE_ENGINE_CLOCK_CFG_ADDR_END_I_0_OFFSET   + slice * 4, eaddr);

}

static inline void hal_sne_init_engine(
  uint32_t engine,
  volatile uint32_t cids[4],
  uint32_t channels,
  uint32_t layer,
  uint32_t kernel_mode) {

  pulp_write(SNE_BASE_ADDR + SNE_ENGINE_CLOCK_CFG_CID_I_0_OFFSET   + 4 * engine, (((cids[0])<<24)+((cids[1])<<16)+((cids[2])<<8)+((cids[3])<<0))); //engine cids
  pulp_write(SNE_BASE_ADDR + SNE_ENGINE_CLOCK_CFG_SLICE_I_0_OFFSET + 4 * engine, (kernel_mode << 12) | (channels << 3) | (layer) ); // slice config, layer, number of channels etc
  pulp_write(SNE_BASE_ADDR + SNE_ENGINE_CLOCK_CFG_ERROR_I_0_OFFSET + 4 * engine, 0x06);
}
```

With a function like the one reported in Listing 6.3.3, the input event address filtering modules of each `Cluster`are configured; this function allows to map each engine to a specific portion of the network.

```c
static void hal_sne_set_filter(
    uint32_t slice,
    uint32_t group,
    uint32_t left,
    uint32_t right,
    uint32_t bottom,
    uint32_t top,
    uint32_t xoffset,
    uint32_t yoffset){


    uint32_t reg_val;
    uint32_t lbound;
    uint32_t ubound;
    uint32_t offsets;
    uint32_t address_offset;

    lbound  = (left)  | (bottom<< 16);
    ubound  = (right) | (top   << 16);

    offsets = (xoffset << 1) | (yoffset << 4) | 0x1;

    // address computed manually for each group:
    // clusters have a unique ID, that can be calculated as slice_id * CLUSTERS + group_id
    // The registers are mapped to consecutive addresses (32bit aligned) for consecutive clusters.
    address_offset = 4*(slice*(SNE_CLUSTERS)+group);

    pulp_write(SNE_BASE_ADDR + SNE_ENGINE_CLOCK_CFG_FILTER_MAIN_I_0_OFFSET   + address_offset, offsets);
    pulp_write(SNE_BASE_ADDR + SNE_ENGINE_CLOCK_CFG_FILTER_LBOUND_I_0_OFFSET + address_offset, lbound);
    pulp_write(SNE_BASE_ADDR + SNE_ENGINE_CLOCK_CFG_FILTER_UBOUND_I_0_OFFSET + address_offset, ubound);

}
```

6.3.4 Automatic code generation

API functions like the ones presented in the previous section constitute the building blocks for composing an application capable of executing a SNN network on the SNE accelerator. As we have mentioned in the introductory section of this chapter, the ultimate goal of the deployment toolchain is to facilitate the execution of an SNN workload on a platform like the Kraken chip. Ideally, to achieve this goal, we want to hide as many low-level hardware-related technicalities as possible from the user.

A fair assumption would be to consider a full network inference as the lowest level of detail that we want to expose to the user. This assumption sets the goal for that part of the toolchain that is in charge of generating the code to perform such an operation. For convenience, from now on, we will call this module “snn-code-generator”.

Several challenges are associated with the automatic code generation for executing a neural network inference pass. A prevalent scenario is where a single neural network layer already needs to allocate more neurons than the available ones executable on the target accelerator. To still be able to run a full inference in this scenario, a viable strategy is to “tile” the network layer and execute only a portion of it. In this case, the accelerator can be used in a time-domain multiplexed way to compute the output of a “tile” at the time. The intermediate features can be temporarily stored in the system memory.

The same approach is used to deploy SNN on the SNE accelerator. It is important to note that the accelerator architecture assumes an “output stationary” execution of the SNN. Moreover, the SNN additional time dimension poses strict constraints to how input feature maps, i.e., spike events, are processed by the accelerator. As the neuron state is held inside the accelerator, to avoid unnecessary memory traffic that would be caused by the membrane potential storage and retrieval from the main memory, it is convenient to feed all the output neurons of a tile with the entire input stream of events and produce, simultaneously, the complete output stream of event related to such output neurons. This section does not have the aim to cover the optimal mapping on the SNE accelerator extensively, as this was outside the scope of demonstrating a vertical software deployment stack; however, such investigation is left for future work and definitely would contribute to a more efficient system memory utilization and lower inference latency.

To achieve a good mapping, we identified a balanced yet efficient accelerator configuration, which suits the network topology presented before. Such a configuration is the one that uses all the engines to compute a 14x14 pixels patch of the input and computes four different output channels in parallel on each engine. This configuration of the SNE is used as a target configuration for the snn-code-generator.

Note that in this scenario, such configuration allows achieving a full occupation of the data path when computing all the layers of the sample network.

The generated code is organized as per-layer execution ANSI C auto-generated functions. The inference process then calls all the generated functions sequentially from the main application code. Each layer execution is sub-divided into a layer tile execution series on SNE. A sample "tiled" layer execution is reported in listing 6.3.4.

```c
int conv1_layer(){

  uint32_t channels = 2;
  uint32_t layer = SNE_LAYER_OPT_CONV;
  uint32_t kernel_mode = SNE_ENGINE_4x4MODE;

  // network hyperparameters
  uint8_t refractory_time = 0;
  uint8_t timescale_shift = 4;
  uint8_t adaptive_threshold = 0;
  uint8_t threshold = 16;
  int8_t  rest_voltage = 0;

  uint32_t streamer_done;

  // write network specific hyperparameters to
    configuration registers
  sne_config_network_params(timescale_shift,
    adaptive_threshold, refractory_time, threshold,
    rest_voltage);

  // write engines configuration registers
  sne_init_engines(cids[i], 256, layer, 0);

  // load weights from L2 to engines' SRAMs
  load_weights(i);

  // set kernel mode reflect the SNE_ENGINE_4x4MODE
    configuration
  for(uint32_t j = 0; j < SNE_ENGINES; j++){
      pulp_write(SNE_BASE_ADDR +
    SNE_ENGINE_CLOCK_CFG_SLICE_I_0_OFFSET + (4 * j), (
    channels << 3) | (layer) | (kernel_mode << 12) );
    }

  // connect streamers to engines (broadcast)
  hal_sne_crossbar_all_engine();
  printf("Weights loaded\n");

  // iterate over the tiles dimension
  for(uint8_t i = 0; i < TILES; ++i){

    printf("Starting tile %d\n", i);

    // map engines and clusters wrt input
    sne_set_params(crops[i], overlaps[i]);

    // configure streamers for execution

    // set streamer 1 for output
    pulp_write(SNE_BASE_ADDR + SNE_BUS_CLOCK_CFG_COMPLEX_I_OFFSET, 0x4);
    pulp_write(SNE_BASE_ADDR + SNE_SYSTEM_CLOCK_CFG_MAIN_CTRL_I_1_OFFSET, 0x00);  // reset?
    hal_sne_init_streamer(1, (uint32_t) spikes_out, 4, 0, 1, 0xEFFFFFFF);
    pulp_write(SNE_BASE_ADDR + SNE_SYSTEM_CLOCK_CFG_MAIN_CTRL_I_1_OFFSET, 0x03); // streamer 1, assert: arm, trigger

    // trigger inputs loading
    pulp_write(SNE_BASE_ADDR + SNE_SYSTEM_CLOCK_CFG_MAIN_CTRL_I_0_OFFSET, 0x04);
    hal_sne_init_streamer(0, (uint32_t) spikes_in, 4, 0, 1, N_SPIKES);
    pulp_write(SNE_BASE_ADDR + SNE_SYSTEM_CLOCK_CFG_MAIN_CTRL_I_0_OFFSET, 0x07); // streamer trigger

    printf("Loading inputs...\n");

    // wait for streamer 0 to finish to load the inputs
    streamer_done = 0;
    while(!streamer_done){
      streamer_done = sne_streamer_done(0);
    }

    printf("Waiting...\n");

    // wait end of computation
    waitc(2000);

    for(uint32_t j = 0; j < 100; ++j){
      printf("%08X\n", spikes_out[j]);
      spikes_out[j] = 0;
    }
  }

  pulp_write(SNE_BASE_ADDR + SNE_SYSTEM_CLOCK_CFG_TCDM_END_ADDR_I_1_OFFSET, (uint32_t) spikes_out);
  pulp_write(SNE_BASE_ADDR + SNE_SYSTEM_CLOCK_CFG_MAIN_CTRL_I_1_OFFSET, 0x08); // streamer 1, assert: arm, trigger

  printf("DONE!\n");
  return 0;
}
```

The executable code is generated by a "Python[4] 3.8" command-line tool which internally adopts a popular "templating" strategy for easy code auto-generation. Specifically, the tool relies on the features exposed by the "mako [5]" library to compose the ANSI C layer execution functions.

6.4 End-to-end application deployment to Kraken

This section provides a conclusive study that evaluates the end-to-end, i.e., sensor-to-algorithm result, energy consumption, and latency of an SNN executed on the Kraken platform. The network was trained on the IBM DVS Gesture event data set [72], which contains 11 hand gestures from 29 subjects under three illumination conditions from a DVS128 event camera. Six subjects' data are used for testing and the remaining for training. The SNN used for such example deployment is reported in Table 6.1. The network is composed of two convolutional layers and two fully-connected layers. We used the training flow described in the first part of this chapter to train the network.

We break down the results of our study into two distinct parts: data acquisition on the FC through the dedicated DVSI interface, and data processing, which consists of a spike preprocessing step in

---

[4] https://www.python.org

[5] https://www.makotemplates.org/

the cluster and a spike train inference step in the SNE. An embedded application typically has the ultimate goal of controlling actuators, in this context, based on the output of the SNN processing the sensor data. Note that the main system clock of Kraken can vary from a few kHz to 200 MHz, and the typical operating frequency is 50 MHz. In typical operating conditions, the latency of changing an output actuation signal, e.g., a PWM signal, is a few system clock cycles, i.e., below 1 µs. Therefore, we considered this latency negligible compared to data acquisition and processing.

Table 6.1: DVS Gesture Network Parameters

	Type	Size	Feature Size	Features	Stride
0	Input	128x128x2	-	-	-
1	Pool	32x32x2	4x4x1	2	4
2	Conv	32x32x16	3x3x2	16	1
3	Pool	16x16x16	2x2x1	16	2
4	Conv	16x16x32	3x3x1	32	1
5	Pool	8x8x32	2x2x1	32	2
6	Full	512	2048	512	-
7	Full	11	512	11	-

6.4.1 results

For advanced processing tasks, e.g., when the neural networks exceed SNE output neuron capacity, like in this case, the inference is executed on the accelerator in a tiled way, and the SNE is used in a time-domain-multiplexing fashion. This process requires a software-based inter-layer spike manipulation performed on the Kraken cluster. Specifically, such a preprocessing step performed on the cluster is necessary to assemble a single input event stream from multiple output tiles and create the tiled input streams for the tiles of the successive layer.

6.2 reports Kraken's energy and latency metrics for a complete gesture recognition task execution. Using a 300 milliseconds window input, Kraken requires 164.5 ms and 7.7 mJ to perform a DVS-to-label prediction, meeting real-time processing constraints. The inference execution on SNE takes only 32 ms and 1.4 mJ.

Table 6.2: Performance metrics of Kraken on DVS-HandGesture Dataset with an accuracy of 83%

Proc. Stage	Time (ms)	Power (mW)[a]		Energy (mJ)[c]
		Idle	Active	
Data Acquisition (FC)	1.5	3.5	3.8	0.006
Preprocessing (Cluster)[c],	131	6.5	34	4.6
SNN Inference (SNE)	32	7.7	44	1.4
Total	164.5	17.7	35.6 [c]	**7.7**

All power numbers measured at $V_{DD} = 0.65\,\mathrm{V}$

[a] Idle power measured with respective components clock-gated.

[b] Total energy is computed as the sum of active energy contributions and idle energy of inactive components.

[c] Average total power consumption during inference

6.5 Conclusion

In this chapter, we have presented the required components to build a complete SNN deployment vertical software stack for a custom RISC-V-based hardware SoC which features a dedicated hardware accelerator, like SNE, capable of executing event-driven energy proportional computation. We can summarize the contribution of this chapter as follows:

- We proposed a viable approach to tackle the challenges posed by the non-differentiable nature of SNNs, practically employing SoA techniques like STBP to circumvent the LIF neuron model non-differentiable formulation. Then, we have shown that SNNs can be effectively trained like conventional DNNs, using the same software substrate, i.e., the Pytorch framework running on CPU or GPU, to perform the neural network training. Combining these two elements constitutes the first layer of the proposed vertical software stack.

- We have shown that the SNNs execution is characterized by a significant reduction in the number of required operations, demonstrating that a hardware architecture that can profit from data and operation sparsity significantly benefits from event-driven computing algorithms like SNNs.

- We indicated how a trained network must be treated to be efficiently executed on an embedded platform. We adopted SoA precision reduction techniques, like PACT, to quantize the network parameters and reduce the overall memory footprint to achieve this goal. The quantization process was embedded into the SNN deployment toolchain to perform quantization-aware training of the network. We empirically show that the quality of the result is only marginally affected by the quantization process, practically showing the validity of the approach in the embedded context.

- We listed the required additions to the open-source "PULP runtime" to support the execution of SNNs on SoC like the one presented in chapter 5. We then used such API function calls as a substrate interface on which the SNN application can build. We demonstrated how SNN layer execution code could be auto-generated and compiled into an executable binary application.

- We presented preliminary accuracy, energy, and latency results of an SNN trained with the abovementioned supervised learning and quantization framework and deployed to the Kraken platform with the code auto-generation flow described in this chapter.

Chapter 7

Summary and Conclusion

7.1 Main results

This section summarizes the main results achieved in this thesis.

Efficient, energy-proportional event-based sensor interfaces

We have presented two IO peripheral architectures capable of collecting external data generated by connected sensors in an efficient and energy-proportional way. Energy proportionality is an aspect that becomes crucial when a device is connected to a sensor that is already capable of producing relevant information in a non-redundant way. The two interfaces exploit specific circuital solutions to mitigate the energy spent during sparse event stream acquisition. Both peripherals were demonstrated on ULP FPGA devices, in real working scenarios. In the case of the DVSI, we could demonstrate on a low power "Lattice" FPGA that such peripheral can achieve a high event-frame acquisition rate of 874 $event - framepersecond$ while consuming only 17.62 mW of power. The whole sensor node consumed 35.5 mW, including the wireless event-frame streaming at 200 efps. In the case of the DASI, we could show that the power consumption for time-to-information extraction scales from 4.5 mW at a 550kevt/s rate down to slightly more than 50 µW at rates lower than 10evt/s (a 90× factor) while a naïve constant clock methodology would consume 4.5 mW power regardless of the event rate. In the second part of the results, we have continued the discussion by showing how the proposed IO peripherals can be efficiently integrated into a digital microcontroller autonomous IO subsystem.

Improving edge-devices energy efficiency on error-resilient applications

We have shown that, by relying on a strategic partitioning of the SoC memory into error-free and error-prone regions, an error-resilient application like BNNs can be efficiently deployed on edge computing devices operated under extremely low voltage conditions. Our analysis and results demonstrated on real silicon how to trade-off the energy consumption of an FDX 22nm SoC with the final classification accuracy of Binary Neural Networks, executed on a dedicated hardware accelerator. We demonstrate that the reported 2.2X energy efficiency gain does not affect the end-to-end classification accuracy of the BNN when the voltage is scaled down to 0.5 V. Additionally, we show that, if a small penalty on the final classification accuracy is tolerable, e.g. within 1%, the SoC can be operated in an ultra-low power mode, further reducing the overall power consumption (674 µW at 18 MHz, 0.42 V)

An Energy-proportional, Sparse, Event-Driven accelerator architecture for SNNs

We presented a novel fully digital accelerator for spiking neural network workloads capable of inherently exploiting the unstructured sparsity of data produced by event-based sensors. In SNE, we demonstrated how an efficient input encoding can be exploited to reduce the total number of engine operations during the inference process. SNE inherently handles sparsity by consuming explicitly spatial and temporal-encoded input events to selectively update the internal output neuron states of those output neurons having a certain input event in their receptive field. with SNE, we could demonstrate how digital neuromorphic platforms, notably less energy efficient than a digital accelerator for CNNs that perform very regular computation, can be improved to achieve a consumed energy per operation that approaches the one shown by more conventional SoA digital accelerator architectures. This result narrows the gap between neuromorphic platforms and classical DNNs.

An event-driven prototype manufactured in FDX22nm technology

As a conclusive study, we presented Kraken, a SoC targeting event-driven cognitive application at the edge. In Kraken, we successfully demonstrated how to integrate, in the same SoC architecture, a general-purpose computing cluster, a neuromorphic computing accelerator, and event-based peripherals, enabling an end-to-end energy-proportional event-driven on-chip computing pipeline. In our analysis, we characterized the general-purpose computing core capabilities in terms of integer and floating-point performance and energy efficiency at multiple operating points. Kraken showed a floating point and integer energy efficiency and performance comparable with the SoA. Silicon characterization of the neuromorphic engine also showed SoA results.

A complete framework for SNN deployment on edge neuromorphic devices

A software environment is a key element in enabling the deployment of applications on any hardware platform. Therefore, we have presented a complete SNN deployment vertical software stack for the Kraken prototype chip. we presented a viable strategy to train and quantize SNNs, practically employing SoA techniques like STBP to circumvent the LIF neuron model non-differentiable formulation, PACT quantization algorithm, and commonly adopted software substrate like the Pytorch framework running on CPU or GPU. Moreover, we showed the required additions to the open-source “PULP runtime” supporting the execution of SNNs on Kraken. We then used such API function calls as a substrate interface on which the SNN application can build on, practically demonstrating how SNN layer execution code can be auto-generated and compiled into an executable binary application. Moreover, we showed preliminary results of an SNN inference deployed to Kraken by using the methodology described in the second part of this thesis.

7.2 Future Work and Outlook

In the following sections, we give an overview of possible future work based on the results achieved in this thesis. At the end of this section, we also provide a concluding outlook.

7.2.1 Future Work

A smarter integration of event-based sensors

The two event-based peripheral interfaces presented in this thesis are simply integrated into an autonomous IO subsystem, already providing great energy savings at the sensor edge. To enable further energy savings at a system level, better integration of such peripherals with the power management unit can be explored, such that a system can autonomously enter and exit low-power states depending on the sensor activity.

The execution of a broader set of neural network families

The SNE accelerator has been designed to execute SNN inference. However, the architecture has no intrinsic limitation that prevents the execution of other neural network families, e.g., conventional DNNs. We believe that an evolution of SNE can target a broader set of neural networks with minor adjustments to the design.

A more effective memory usage

The accelerator proposed in this thesis uses a significant amount of local memory to hold the states of the neuron. An improvement that would make the architecture even more scalable would be to reduce such local memories and let SNE fetch and store data from the main system memory as the computation happens. How to feed a large number of parallel computational elements in presence of high unstructured sparsity remains very challenging, and it is partially an open question.

The support for low-precision neuromorphic computing

Neuromorphic computing is a relatively new field, and currently, there is rather limited support for SNN training. When such a computing framework is adopted in the embedded domain, it becomes crucial to adopt all the “standard” memory footprint and computational complexity reduction techniques, e.g. quantization. We believe that similarly to what has been observed for conventional DNNs, SNNs can also provide very good accuracy end energy efficiency, at very low precision. Therefore, there is a need for SNN training frameworks supporting quantization-aware training to unlock the full SNN potential at a large scale, as happened for conventional DNNs.

7.2.2 Outlook

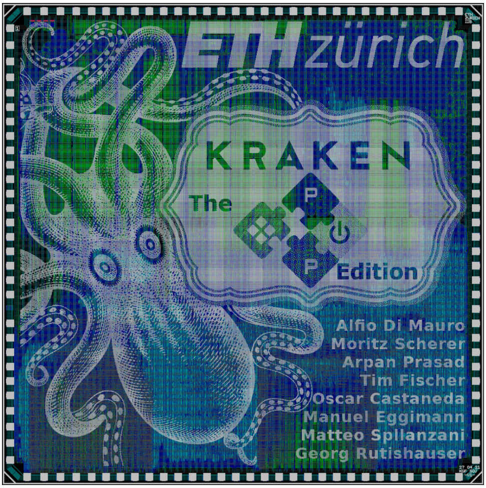
In the next step, we aim at addressing the open points presented above and design an SoC which features better cooperation of event-based sensors and the power management unit, more flexible and scalable, sparse computational engines. We believe also that more extensive software support for automatic application deployment will favor the development of novel, embedded, event-driven efficient applications.

Appendix A

Chip gallery

This appendix lists all chips that have been fabricated and are related to this thesis. A complete, up-to-date list can be found online at: `http://asic.ethz.ch/authors/Alfio_Di_Mauro.html`

A.1 Kraken



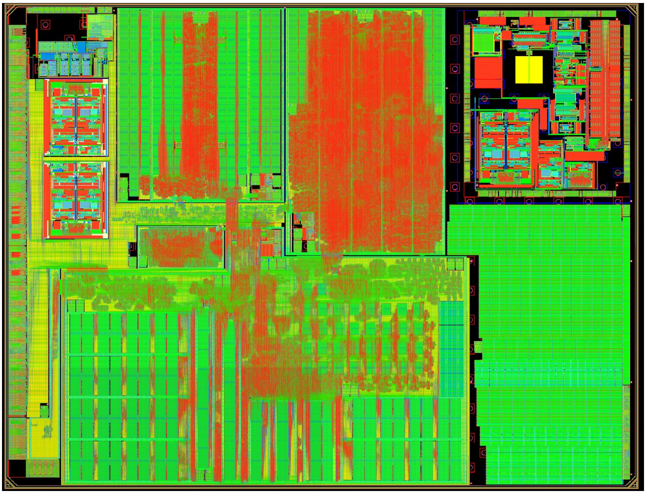
Type	IoT processor
Technology	GlobalFoundries FDX 22nm
Dimensions	3000 µm × 3000 µm
Gate count	25MGE
Supply voltage	[0.5V-0.8V]
Clock frequency	300 (Typical) MHz

Figure A.1: Kraken chip layout and main details

Description: Kraken is a PULP-based IoT processor. It features one fabric controller (a 32bit RI5CY/CV32E40P core) and a computing cluster with eight 32b RISC-V cores. Kraken has a dedicated accelerator domain hosting SNE and CUTIE.

Designed and tested by: Alfio Di Mauro, Moritz Scherer, Arpan Prasad, Tim Fischer, Oscar Castaneda, Manuel Eggimann, Matteo Spallanzani, Georg Rutishauser.

A.2 Marsellus



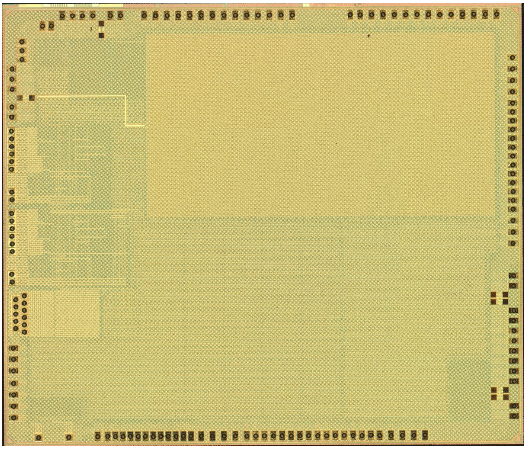
Type	IoT processor
Technology	GlobalFoundries FDX 22nm
Dimensions	4000 µm × 3000 µm
Gate count	25MGE
Supply voltage	0.8V
Clock frequency	400 (Typical) MHz

Figure A.2: Marsellus chip layout and main details

Description: Marsellus is a co-operation between ETH Zurich and Dolphin Design and includes a PULPopen instantiation as well as a low power PLL from the Energy Efficient Circuits and IoT Systems

Designed and tested by: Davide Rossi, Alfio Di Mauro, Francesco Conti, Gianna Paulin, Angelo Garofalo, Gianmarco Ottavi, Georg Rutishauser, Hayate Okuhara, Manuel Eggimann.

A.3 Vega



Type	IoT processor
Technology	GlobalFoundries FDX 22nm
Dimensions	4000 µm × 3000 µm
Gate count	100MGE
Supply voltage	[0.5V-0.8V]
Clock frequency	450 (Typical) MHz

Figure A.3: Vega chip layout and main details

Description: Vega is a 10 (9+1) RISC-V cores, PULP-based, always-on IoT end-node SoC capable of scaling from a 1.7uW fully retentive COGNITIVE sleep mode up to 32.2GOPS (@49.4mW) peak performance. It features 1.6MB of state- retentive SRAM, and 4MB of non-volatile MRAM.

Designed and tested by: Davide Rossi, Francesco Conti, Manuel Eggimann, Stefan Mach, Alfio Di Mauro, Marco Guermandi, Giuseppe Tagliavini, Antonio Pullini, Igor Loi, Jie Chen, Eric Flamand.

Appendix B

Notation and Acronyms

Acronyms

ABB	adaptive body-biasing
AER	Address Event Representation
AETR	Address-Event-Timestamp Representation
AI	artificial intelligence
ALIF	adaptive leaky-integrate and fire
ALU	arithmetic logic unit
ANN	artificial neural network
APB	advanced peripheral bus
API	application programming interface
BNN	binary neural network
CDC	clock domain crossing
CPI	camera parallel interface
CPU	central process unit

DASI	dynamic audio sensor interface
DMA	direct memory access
DNN	deep neural network
DVFS	dynamic voltage and frequency scaling
DVS	dynamic visual sensor
DVSI	dynamic visual sensor interface
EHWPE	esternal hardware processing engine
FC	fabric controller
FIFO	first-in first-out
FISTA	fast iterative shrinkage-thresholding algorithm
FLL	frequency locked loop
FPGA	field-programmable gate array
FPU	floating point unit
GPIO	general purpose input output
GPU	graphics process unit
HAL	hardware abstraction layer
I^2C	Inter-Integrated Circuit
I^2S	Inter-Integrated-Circuit Sound
IMU	inertial measurement unit
IOT	internet of things
ISA	instruction set architecture
JTAG	joint test action group
LASSO	least absolute shrinkage and selection operator
LIF	leaky-integrate and fire
LUT	look up table
MAC	multiply-accumulate

MCU	micro-controller unit
ML	machine learning
NSAA	near sensor analytic application
NTC	near-threshold computing
PDK	process development kit
PMU	power management unit
PPA	power performance analysis
PWM	pulse width modulation
QSPI	quad-serial peripheral interface
SLAM	simultaneous localization and mapping
SNE	sparse neural engine
SNN	spiking neural network
SoA	state-of-the-art
SoC	system-on-chip
SOP	synaptic operation
SRM	spike response model
STA	static timing analysis
STBP	spatio-temporal back propagation
SDBP	spike time dependent plasticity
TCDM	tightly coupled data memory
TEI	temperature effect inversion
TPU	tensor process unit
UART	universal asynchronous receiver transmitter
ULP	ultra-low power

Bibliography

[1] A. Arif, F. A. Barrigon, F. Gregoretti, J. Iqbal, L. Lavagno, M. T. Lazarescu, L. Ma, M. Palomino, and J. L. L. Segura, "Performance and energy-efficient implementation of a smart city application on FPGAs," *Journal of Real-Time Image Processing*, vol. 17, no. 3, pp. 729–743, jun 2018. [Online]. Available: https://doi.org/10.1007%2Fs11554-018-0792-x

[2] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. S. Bernstein, A. C. Berg, and L. Fei-Fei, "Imagenet large scale visual recognition challenge," *CoRR*, vol. abs/1409.0575, 2014. [Online]. Available: http://arxiv.org/abs/1409.0575

[3] M. N. Tehrani, M. Uysal, and H. Yanikomeroglu, "Device-to-device communication in 5g cellular networks: challenges, solutions, and future directions," *IEEE Communications Magazine*, vol. 52, no. 5, pp. 86–92, 2014.

[4] W. Yu, F. Liang, X. He, W. G. Hatcher, C. Lu, J. Lin, and X. Yang, "A survey on the edge computing for the internet of things," *IEEE Access*, vol. 6, pp. 6900–6919, 2018.

[5] "Ambiq Apollo Data Brief."

[6] "NXP LPC185x/3x/2x/1x Datasheet."

[7] "STMicroelectronics STM32L476xx Datasheet."

[8] "Texas Instruments MSP430 Low-Power MCUs," http://www.ti.com/lsds/ti/microcontrollers_16-bit_32-bit/msp/overview.page.

[9] O. B. Tariq, M. T. Lazarescu, and L. Lavagno, "Neural networks for indoor human activity reconstructions," *IEEE Sensors Journal*, vol. 20, no. 22, pp. 13 571–13 584, Nov 2020.

[10] M. Gautschi, P. D. Schiavone, A. Traber, I. Loi, A. Pullini, D. Rossi, E. Flamand, F. K. Gurkaynak, and L. Benini, "Near-Threshold RISC-V Core With DSP Extensions for Scalable IoT Endpoint Devices," *IEEE TVLSI*, vol. 25, no. 10, pp. 2700–2713, 2017.

[11] P. Prabhat, B. Labbe, G. Knight, A. Savanth, J. Svedas, M. J. Walker, S. Jeloka, P. M.-Y. Fan, F. Garcia-Redondo, T. Achuthan, and J. Myers, "27.2 m0n0: A performance-regulated 0.8-to-38mhz dvfs arm cortex-m33 simd mcu with 10nw sleep power," in *2020 IEEE International Solid- State Circuits Conference - (ISSCC)*, 2020, pp. 422–424.

[12] A. Pullini, D. Rossi, I. Loi, G. Tagliavini, and L. Benini, "Mr.wolf: An energy-precision scalable parallel ultra low power soc for iot edge processing," *IEEE Journal of Solid-State Circuits*, vol. 54, pp. 1970–1981, 7 2019.

[13] A. Pullini, D. Rossi, I. Loi, A. D. Mauro, and L. Benini, "Mr. Wolf: A 1 GFLOP/s Energy-Proportional Parallel Ultra Low Power SoC for IOT Edge Processing," in *ESSCIRC 2018 - IEEE 44th European Solid State Circuits Conference (ESSCIRC)*, Sep. 2018, pp. 274–277.

[14] A. D. Mauro, D. Rossi, A. Pullini, P. Flatresse, and L. Benini, "Independent body-biasing of p-n transistors in an 28nm utbb fd-soi ulp near-threshold multi-core cluster," in *2018 IEEE SOI-3D-Subthreshold Microelectronics Technology Unified Conference (S3S)*, Oct 2018, pp. 1–3.

[15] D. Rossi, F. Conti, M. Eggimann, A. D. Mauro, G. Tagliavini, S. Mach, M. Guermandi, A. Pullini, I. Loi, J. Chen, E. Flamand,

and L. Benini, “Vega: A ten-core soc for iot endnodes with dnn acceleration and cognitive wake-up from mram-based state-retentive sleep mode,” *IEEE Journal of Solid-State Circuits*, pp. 1–1, 2021.

[16] P. Mayer, M. Magno, and L. Benini, “Self-sustaining acoustic sensor with programmable pattern recognition for underwater monitoring,” *IEEE Transactions on Instrumentation and Measurement*, vol. 68, no. 7, pp. 2346–2355, 2019.

[17] P. D. Schiavone, D. Rossi, A. Pullini, A. Di Mauro, F. Conti, and L. Benini, “Quentin: an ultra-low-power pulpissimo soc in 22nm fdx,” in *2018 IEEE SOI-3D-Subthreshold Microelectronics Technology Unified Conference (S3S)*, 2018, pp. 1–3.

[18] A. Garofalo, G. Ottavi, A. di Mauro, F. Conti, G. Tagliavini, L. Benini, and D. Rossi, “A 1.15 tops/w, 16-cores parallel ultra-low power cluster with 2b-to-32b fully flexible bit-precision and vector lockstep execution mode,” in *ESSCIRC 2021 - IEEE 47th European Solid State Circuits Conference (ESSCIRC)*, 2021, pp. 267–270.

[19] N. Bruschi, G. Haugou, G. Tagliavini, F. Conti, L. Benini, and D. Rossi, “Gvsoc: A highly configurable, fast and accurate full-platform simulator for risc-v based iot processors,” in *2021 IEEE 39th International Conference on Computer Design (ICCD)*, 2021, pp. 409–416.

[20] B. Forsberg, M. Solieri, M. Bertogna, L. Benini, and A. Marongiu, “The predictable execution model in practice: Compiling real applications for cots hardware,” *ACM Trans. Embed. Comput. Syst.*, vol. 20, no. 5, jul 2021. [Online]. Available: https://doi.org/10.1145/3465370

[21] F. Zaruba and L. Benini, “The cost of application-class processing: Energy and performance analysis of a linux-ready 1.7-ghz 64-bit risc-v core in 22-nm fdsoi technology,” *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 27, no. 11, pp. 2629–2640, 2019.

[22] R. Christy, S. Riches, S. Kottekkat, P. Gopinath, K. Sawant, A. Kona, and R. Harrison, “8.3 a 3ghz arm neoverse n1 cpu in 7nm finfet for infrastructure applications,” in *2020 IEEE International Solid- State Circuits Conference - (ISSCC)*, 2020, pp. 148–150.

[23] R. G. Dreslinski, M. Wieckowski, D. Blaauw, D. Sylvester, and T. Mudge, “Near-threshold computing: Reclaiming moore’s law through energy efficient integrated circuits,” *Proceedings of the IEEE*, vol. 98, no. 2, pp. 253–266, Feb 2010.

[24] A. Di Mauro, D. Rossi, A. Pullini, P. Flatresse, and L. Benini, “Temperature and process-aware performance monitoring and compensation for an ulp multi-core cluster in 28nm utbb fd-soi technology,” in *2017 27th International Symposium on Power and Timing Modeling, Optimization and Simulation (PATMOS)*, 2017, pp. 1–8.

[25] J. Cortadella, L. Lavagno, P. López, M. Lupon, A. Moreno, A. Roca, and S. S. Sapatnekar, “Reactive clocks with variability-tracking jitter,” in *2015 33rd IEEE International Conference on Computer Design (ICCD)*, Oct 2015, pp. 511–518.

[26] M. Cannizzaro, S. Beer, J. Cortadella, R. Ginosar, and L. Lavagno, “Saferazor: Metastability-robust adaptive clocking in resilient circuits,” *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 62, no. 9, pp. 2238–2247, Sep. 2015.

[27] F. Zaruba, F. Schuiki, and L. Benini, “Manticore: A 4096-core risc-v chiplet architecture for ultraefficient floating-point computing,” *IEEE Micro*, vol. 41, no. 2, pp. 36–42, 2021.

[28] P. Vivet, E. Guthmuller, Y. Thonnart, G. Pillonnet, G. Moritz, I. Miro-Panadès, C. Fuguet, J. Durupt, C. Bernard, D. Varreau, J. Pontes, S. Thuries, D. Coriat, M. Harrand, D. Dutoit, D. Lattard, L. Arnaud, J. Charbonnier, P. Coudrain, A. Garnier, F. Berger, A. Gueugnot, A. Greiner, Q. Meunier, A. Farcy, A. Arriordaz, S. Cheramy, and F. Clermidy, “2.3 a 220gops 96-core processor with 6 chiplets 3d-stacked on an active interposer

offering 0.6ns/mm latency, 3tb/s/mm¡sup¿2¡/sup¿ inter-chiplet interconnects and 156mw/mm¡sup¿2¡/sup¿@ 82*2020 IEEE International Solid- State Circuits Conference - (ISSCC)*, 2020, pp. 46–48.

[29] S. Davidson, S. Xie, C. Torng, K. Al-Hawai, A. Rovinski, T. Ajayi, L. Vega, C. Zhao, R. Zhao, S. Dai, A. Amarnath, B. Veluri, P. Gao, A. Rao, G. Liu, R. K. Gupta, Z. Zhang, R. Dreslinski, C. Batten, and M. B. Taylor, "The celerity open-source 511-core risc-v tiered accelerator fabric: Fast architectures and design methodologies for fast chips," *IEEE Micro*, vol. 38, no. 2, pp. 30–41, 2018.

[30] E. Flamand, D. Rossi, F. Conti, I. Loi, A. Pullini, F. Rotenberg, and L. Benini, "GAP-8: A RISC-V SoC for AI at the Edge of the IoT," in *2018 IEEE 29th International Conference on Application-Specific Systems, Architectures and Processors (ASAP)*. Milano, Italy: IEEE, Jul. 2018, pp. 1–4.

[31] D. C. Daly, L. C. Fujino, and K. C. Smith, "Through the looking glass-2020 edition: Trends in solid-state circuits from isscc," *IEEE Solid-State Circuits Magazine*, vol. 12, no. 1, pp. 8–24, 2020.

[32] F. Conti and L. Benini, "A ultra-low-energy convolution engine for fast brain-inspired vision in multicore clusters," in *2015 Design, Automation Test in Europe Conference Exhibition (DATE)*, 2015, pp. 683–688.

[33] M. Scherer, G. Rutishauser, L. Cavigelli, and L. Benini, "Cutie: Beyond petaop/s/w ternary dnn inference acceleration with better-than-binary energy efficiency," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 41, no. 4, pp. 1020–1033, 2022.

[34] J. Lee, C. Kim, S. Kang, D. Shin, S. Kim, and H. Yoo, "UNPU: An Energy-Efficient Deep Neural Network Accelerator With Fully Variable Weight Bit Precision," *IEEE Journal of Solid-State Circuits*, vol. 54, no. 1, pp. 173–185, Jan 2019.

[35] M. Courbariaux and Y. Bengio, "Binarynet: Training deep neural networks with weights and activations constrained to +1 or -1," *CoRR*, vol. abs/1602.02830, 2016. [Online]. Available: http://arxiv.org/abs/1602.02830

[36] A. Zhou, A. Yao, Y. Guo, L. Xu, and Y. Chen, "Incremental network quantization: Towards lossless cnns with low-precision weights," *CoRR*, vol. abs/1702.03044, 2017. [Online]. Available: http://arxiv.org/abs/1702.03044

[37] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, "XNOR-Net: ImageNet Classification Using Binary Convolutional Neural Networks," *arXiv:1603.05279 [cs]*, Mar. 2016.

[38] F. Conti, P. D. Schiavone, and L. Benini, "XNOR Neural Engine: A Hardware Accelerator IP for 21.6-fJ/op Binary Neural Network Inference," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 11, pp. 2940–2951, Nov. 2018.

[39] M. Yang, C.-H. Chien, T. Delbruck, and S.-C. Liu, "A 0.5V 55μW 64$\times$2 Channel Binaural Silicon Cochlea for Event-Driven Stereo-Audio Sensing," in *2016 IEEE International Solid-State Circuits Conference (ISSCC)*. IEEE, 2016, pp. pp. 388–389.

[40] C. Li, L. Longinotti, F. Corradi, and T. Delbruck, "A 132 by 104 10um-pixel 250uw 1kefps dynamic vision sensor with pixel-parallel noise and spatial redundancy suppression," in *2019 Symposium on VLSI Circuits*, 2019, pp. C216–C217.

[41] A. Bachrach, "Skydio autonomy engine: Enabling the next generation of autonomous flight," in *2021 IEEE Hot Chips 33 Symposium (HCS)*, 2021.

[42] F. Alam, R. Mehmood, I. Katib, N. N. Albogami, and A. Albeshri, "Data fusion and iot for smart ubiquitous environments: A survey," *IEEE Access*, vol. 5, pp. 9533–9554, 2017.

[43] X. Li, D. Neil, T. Delbruck, and S.-C. Liu, "Lip reading deep network exploiting multi-modal spiking visual and auditory

sensors," in *2019 IEEE International Symposium on Circuits and Systems (ISCAS)*, 2019, pp. 1–5.

[44] Z. Jiang, P. Xia, K. Huang, W. Stechele, G. Chen, Z. Bing, and A. Knoll, "Mixed frame-/event-driven fast pedestrian detection," in *2019 International Conference on Robotics and Automation (ICRA)*, 2019, pp. 8332–8338.

[45] A. Di Mauro, F. Conti, and L. Benini, "An ultra-low power address-event sensor interface for energy-proportional time-to-information extraction," in *2017 54th ACM/EDAC/IEEE Design Automation Conference (DAC)*, 2017, pp. 1–6.

[46] T. Hoefler, D. Alistarh, T. Ben-Nun, N. Dryden, and A. Peste, "Sparsity in deep learning: Pruning and growth for efficient inference and training in neural networks," 2021. [Online]. Available: https://arxiv.org/abs/2102.00554

[47] S. Dave, R. Baghdadi, T. Nowatzki, S. Avancha, A. Shrivastava, and B. Li, "Hardware acceleration of sparse and irregular tensor computations of ml models: A survey and insights," *Proceedings of the IEEE*, pp. 1–47, 2021.

[48] A. Di Mauro, M. Scherer, J. F. Mas, B. Bougenot, M. Magno, and L. Benini, "Flydvs: An event-driven wireless ultra-low power visual sensor node," in *2021 Design, Automation Test in Europe Conference Exhibition (DATE)*, Feb 2021, pp. 1851–1854.

[49] A. D. Mauro, F. Conti, P. D. Schiavone, D. Rossi, and L. Benini, "Always-on 674uw@4gop/s error resilient binary neural networks with aggressive sram voltage scaling on a 22-nm iot end-node," *IEEE Transactions on Circuits and Systems I: Regular Papers*, pp. 1–14, 2020.

[50] A. Di Mauro, A. Suravi Prasad, Z. Huang, M. Spallanzani, F. Conti, and L. Benini, "Sne: an energy-proportional digital accelerator for sparse event-based convolutions," 2022, design, Automation and Test in Europe Conference (DATE 2022); Conference Location: Online; Conference Date: March 14-23, 2022; Conference lecture held on 22 March 2022.

[51] A. Di Mauro, M. Scherer, D. Rossi, and L. Benini, “Kraken: A direct event/frame-based multi-sensor fusion soc for ultra-efficient visual processing in nano-uavs,” in *2022 IEEE Hot Chips 34 Symposium (HCS)*, 2022, pp. 1–19.

[52] A. D. Mauro, F. Conti, P. D. Schiavone, D. Rossi, and L. Benini, “Pushing on-chip memories beyond reliability boundaries in micropower machine learning applications,” in *2019 IEEE International Electron Devices Meeting (IEDM)*, 2019, pp. 30.4.1–30.4.4.

[53] A. Di Mauro, F. Zaruba, F. Schuiki, S. Mach, and L. Benini, “Live demonstration: Exploiting body-biasing for static corner trimming and maximum energy efficiency operation in 22nm fdx technology,” in *2020 IEEE International Symposium on Circuits and Systems (ISCAS)*, 2020, pp. 1–1.

[54] A. Di Mauro, D. Rossi, A. Pullini, P. Flatresse, and L. Benini, “Live demonstration: Body-bias based performance monitoring and compensation for a near-threshold multi-core cluster in 28nm fd-soi technology,” in *2018 IEEE International Symposium on Circuits and Systems (ISCAS)*, 2018, pp. 1–1.

[55] A. Di Mauro, D. Rossi, A. Pullini, P. Flatresse, and L. Benini, “Performance-aware predictive-model-based on-chip body-bias regulation strategy for an ulp multi-core cluster in 28 nm utbb fd-soi,” *Integration*, vol. 72, pp. 194–207, 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167926017307952

[56] A. Di Mauro, H. Fatemi, J. P. de Gyvez, and L. Benini, “Idleness-aware dynamic power mode selection on the i.mx 7ulp iot edge processor,” *Journal of Low Power Electronics and Applications*, vol. 10, no. 2, 2020. [Online]. Available: https://www.mdpi.com/2079-9268/10/2/19

[57] D. Rossi, F. Conti, M. Eggiman, S. Mach, A. D. Mauro, M. Guermandi, G. Tagliavini, A. Pullini, I. Loi, J. Chen, E. Flamand, and L. Benini, “4.4 a 1.3tops/w @ 32gops fully integrated 10-core soc for iot end-nodes with 1.7μ w cognitive

wake-up from mram-based state-retentive sleep mode," in *2021 IEEE International Solid- State Circuits Conference (ISSCC)*, vol. 64, 2021, pp. 60–62.

[58] P. D. Schiavone, D. Rossi, A. Di Mauro, F. K. Gürkaynak, T. Saxe, M. Wang, K. C. Yap, and L. Benini, "Arnold: An efpga-augmented risc-v soc for flexible and low-power iot end nodes," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 29, no. 4, pp. 677–690, 2021.

[59] H. Okuhara, A. Elnaqib, D. Rossi, A. Di Mauro, P. Mayer, P. Palestri, and L. Benini, "An energy-efficient low-voltage swing transceiver for mw-range iot end-nodes," in *2020 IEEE International Symposium on Circuits and Systems (ISCAS)*, 2020, pp. 1–5.

[60] M. Hersche, E. M. Rella, A. Di Mauro, L. Benini, and A. Rahimi, "Integrating event-based dynamic vision sensors with sparse hyperdimensional computing: A low-power accelerator with online learning capability," in *Proceedings of the ACM/IEEE International Symposium on Low Power Electronics and Design*, ser. ISLPED '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 169–174. [Online]. Available: https://doi.org/10.1145/3370748.3406560

[61] H. Abdi and L. J. Williams, "Principal component analysis," *Wiley Interdisciplinary Reviews: Computational Statistics*, vol. 2, no. 4, pp. 433–459, 2010.

[62] T. Kanungo, D. M. Mount, N. S. Netanyahu, C. D. Piatko, R. Silverman, and A. Y. Wu, "An efficient k-means clustering algorithm: Analysis and implementation," *IEEE transactions on pattern analysis and machine intelligence*, vol. 24, no. 7, pp. 881–892, 2002.

[63] C.-C. Chang and C.-J. Lin, "LIBSVM: A Library for Support Vector Machines," *ACM Transactions on Intelligent Systems and Technology*, vol. 2, no. 3, pp. 27:1–27:27, May 2011.

[64] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “ImageNet Classification with Deep Convolutional Neural Networks,” in *Advances in Neural Information Processing Systems 25*, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, Eds. Curran Associates, Inc., 2012, pp. 1097–1105.

[65] C. Posch, T. Serrano-Gotarredona, B. Linares-Barranco, and T. Delbruck, “Retinomorphic Event-Based Vision Sensors: Bioinspired Cameras With Spiking Output,” *Proceedings of the IEEE*, vol. 102, no. 10, pp. 1470–1484, Oct. 2014.

[66] M. Yang, C. H. Chien, T. Delbruck, and S. C. Liu, “A 0.5V 55 μW 64x2-channel binaural silicon cochlea for event-driven stereo-audio sensing,” in *2016 IEEE International Solid-State Circuits Conference (ISSCC)*, Jan. 2016, pp. 388–389.

[67] M. Gottardi, N. Massari, and S. A. Jawed, “A 100 μW 128x64 pixels contrast-based asynchronous binary vision sensor for sensor networks applications,” *IEEE Journal of Solid-State Circuits*, vol. 44, no. 5, pp. 1582–1592, May 2009.

[68] D. A. Butts, C. Weng, J. Jin, C.-I. Yeh, N. A. Lesica, J.-M. Alonso, and G. B. Stanley, “Temporal precision in the neural code and the timescales of natural vision,” 2007.

[69] S. Moradi and G. Indiveri, “An event-based neural network architecture with an asynchronous programmable synaptic memory,” *IEEE Transactions on Biomedical Circuits and Systems*, vol. 8, no. 1, pp. 98–107, Feb. 2014.

[70] P. a. Merolla, J. V. Arthur, R. Alvarez-Icaza, a. S. Cassidy, J. Sawada, F. Akopyan, B. L. Jackson, N. Imam, C. Guo, Y. Nakamura, B. Brezzo, I. Vo, S. K. Esser, R. Appuswamy, B. Taba, A. Amir, M. D. Flickner, W. P. Risk, R. Manohar, and D. S. Modha, “A million spiking-neuron integrated circuit with a scalable communication network and interface,” *Science*, vol. 345, no. 6197, pp. 668–673, Aug. 2014.

[71] C. Li, L. Longinotti, F. Corradi, and T. Delbruck, “A 132 by 104 10um-pixel 250uw 1kefps dynamic vision sensor with

pixel-parallel noise and spatial redundancy suppression," in *2019 Symposium on VLSI Circuits*, 2019, pp. C216–C217.

[72] A. Amir, B. Taba, D. Berg, T. Melano, J. McKinstry, C. Di Nolfo, T. Nayak, A. Andreopoulos, G. Garreau, M. Mendoza *et al.*, "A low power, fully event-based gesture recognition system," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 7243–7252.

[73] G. Gallego, T. Delbruck, G. Orchard, C. Bartolozzi, B. Taba, A. Censi, S. Leutenegger, A. Davison, J. Conradt, K. Daniilidis *et al.*, "Event-based vision: A survey," *arXiv preprint arXiv:1904.08405*, 2019.

[74] B. Wen and K. Boahen, "A silicon cochlea with active coupling," *IEEE Transactions on Biomedical Circuits and Systems*, vol. 3, no. 6, pp. 444–455, Dec. 2009.

[75] S. C. Liu, A. van Schaik, B. A. Mincti, and T. Delbruck, "Event-based 64-channel binaural silicon cochlea with q enhancement mechanisms," in *Proc. IEEE Int. Symp. Circuits and Systems*, May 2010, pp. 2027–2030.

[76] S. C. Liu, A. van Schaik, B. A. Minch, and T. Delbruck, "Asynchronous Binaural Spatial Audition Sensor With 2 64 4 Channel Output," *IEEE Transactions on Biomedical Circuits and Systems*, vol. 8, no. 4, pp. 453–464, Aug. 2014.

[77] W. Song, Q. Han, Z. Lin, N. Yan, D. Luo, Y. Liao, M. Zhang, Z. Wang, X. Xie, A. Wang *et al.*, "Design of a flexible wearable smart semg recorder integrated gradient boosting decision tree based hand gesture recognition," *IEEE Transactions on Biomedical Circuits and Systems*, vol. 13, no. 6, pp. 1563–1574, 2019.

[78] M. Vandersteegen, W. Reusen, K. Van Beeck, and T. Goedemé, "Low-latency hand gesture recognition with a low-resolution thermal imager," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, 2020, pp. 98–99.

[79] V. K. Sarker, M. Jiang, T. N. Gia, A. Anzanpour, A. M. Rahmani, and P. Liljeberg, "Portable multipurpose bio-signal acquisition and wireless streaming device for wearables," in *2017 IEEE Sensors Applications Symposium (SAS)*, 2017, pp. 1–6.

[80] A. Linares-Barranco, R. Paz, A. Jimenez-Fernandez, C. D. Lujan, M. Rivas, J. L. Sevillano, G. Jimenez, and A. Civit, "Neuro-inspired real-time USB & PCI to AER interfaces for vision processing," in *Proc. Int. Symp. Performance Evaluation of Computer and Telecommunication Systems SPECTS 2008*, Jun. 2008, pp. 330–337.

[81] G. García, C. Jara, J. Pomares, A. Alabdo, L. Poggi, and F. Torres, "A Survey on FPGA-Based Sensor Systems: Towards Intelligent and Reconfigurable Low-Power Sensors for Computer Vision, Control and Signal Processing," *Sensors*, vol. 14, no. 4, p. 6247–6278, Mar 2014. [Online]. Available: http://dx.doi.org/10.3390/s140406247

[82] R. Berner, T. Delbruck, A. Civit-Balcells, and A. Linares-Barranco, "A 5 Meps USB2.0 Address-Event Monitor-Sequencer Interface," 2006.

[83] R. Paz-Vicente, A. Linares-Barranco, D. Cascado, M. A. Rodriguez, G. Jimenez, A. Civit, and J. L. Sevillano, "PCI-AER interface for neuro-inspired spiking systems," in *2006 IEEE International Symposium on Circuits and Systems*, May 2006, pp. 4 pp.–.

[84] S. O. Cisneros, J. J. R. Panduro, D. T. A. Bretón, and J. R. R. Barón, "Space-time aer protocol receiver asynchronously controlled on fpga," *Computing Science and Automatic Control (CCE)*, 2014.

[85] M. Hofstätter, P. Schön, and C. Posch, "A SPARC-compatible general purpose address-event processor with 20-bit l0ns-resolution asynchronous sensor data interface in 0.18um CMOS," *IEEE International Symposium on Circuits and Systems*, 2010.

[86] C. Brandli, R. Berner, M. Yang, S. C. Liu, and T. Delbruck, "A 240×180 130 dB 3 μs Latency Global Shutter Spatiotemporal Vision Sensor," *IEEE Journal of Solid-State Circuits*, vol. 49, no. 10, pp. 2333–2341, Oct. 2014.

[87] A. Banerjee, C. Chakraborty, A. Kumar, and D. Biswas, "Emerging trends in iot and big data analytics for biomedical and health care technologies," in *Handbook of data science approaches for biomedical engineering.* Elsevier, 2020, pp. 121–152.

[88] *CAVIAR Hardware Interface Standards, Version 2.01.*

[89] H. Greenspan, B. van Ginneken, and R. M. Summers, "Guest Editorial Deep Learning in Medical Imaging: Overview and Future Promise of an Exciting New Technique," *IEEE Transactions on Medical Imaging*, vol. 35, no. 5, pp. 1153–1159, May 2016.

[90] D. Palossi, A. Loquercio, F. Conti, E. Flamand, and L. Benini, "A 64mW DNN-based Visual Navigation Engine for Autonomous Nano-Drones," *IEEE Internet of Things Journal*, vol. 6, no. 5, pp. 8357–8371, Oct. 2019.

[91] M. Manic, K. Amarasinghe, J. J. Rodriguez-Andina, and C. Rieger, "Intelligent Buildings of the Future: Cyberaware, Deep Learning Powered, and Human Interacting," *IEEE Industrial Electronics Magazine*, vol. 10, no. 4, pp. 32–49, Dec. 2016.

[92] D. Bankman, L. Yang, B. Moons, M. Verhelst, and B. Murmann, "An Always-On 3.8μJ/86% CIFAR-10 Mixed-Signal Binary CNN Processor with All Memory on Chip in 28nm CMOS," in *Proceedings of 2018 IEEE International Solid-State Circuits Conference.*

[93] 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 *Proceedings of 2018 IEEE International Solid-State Circuits Conference.*

[94] A. A. Bahou, G. Karunaratne, R. Andri, L. Cavigelli, and L. Benini, “XNORBIN: A 95 TOp/s/W Hardware Accelerator for Binary Convolutional Neural Networks,” *arXiv:1803.05849 [cs]*, Mar. 2018.

[95] 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, Apr. 2018.

[96] S. Sharify, A. D. Lascorz, K. Siu, P. Judd, and A. Moshovos, “Loom: Exploiting Weight and Activation Precisions to Accelerate Convolutional Neural Networks,” in *Proceedings of the 55th Annual Design Automation Conference*, ser. DAC ’18. New York, NY, USA: ACM, 2018, pp. 20:1–20:6.

[97] 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, Jan. 2017.

[98] V. Sze, Y.-H. Chen, T.-J. Yang, and J. Emer, “Efficient Processing of Deep Neural Networks: A Tutorial and Survey,” *arXiv:1703.09039 [cs]*, Mar. 2017.

[99] N. Verma and A. P. Chandrakasan, “A 65nm 8T Sub-Vt SRAM Employing Sense-Amplifier Redundancy,” in *2007 IEEE International Solid-State Circuits Conference. Digest of Technical Papers*, Feb. 2007, pp. 328–606.

[100] N. Verma and A. P. Chandrakasan, “A 256 kb 65 nm 8T Subthreshold SRAM Employing Sense-Amplifier Redundancy,” *IEEE Journal of Solid-State Circuits*, vol. 43, no. 1, pp. 141–149, Jan 2008.

[101] A. Teman, D. Rossi, P. Meinerzhagen, L. Benini, and A. Burg, “Controlled placement of standard cell memory arrays for high density and low power in 28nm FD-SOI,” in *Design Automation Conference (ASP-DAC), 2015 20th Asia and South Pacific*, Jan. 2015, pp. 81–86.

[102] D. Esposito, A. G. M. Strollo, and M. Alioto, “Power-precision scalable latch memories,” pp. 1–4, May 2017.

[103] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, “Quantized Neural Networks: Training Neural Networks with Low Precision Weights and Activations,” *arXiv:1609.07061 [cs]*, Sep. 2016.

[104] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, “Binarized neural networks,” in *Advances in Neural Information Processing Systems 29*, D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, Eds. Curran Associates, Inc., 2016, pp. 4107–4115. [Online]. Available: http://papers.nips.cc/paper/6573-binarized-neural-networks.pdf

[105] M. Yang, C. Yeh, Y. Zhou, J. P. Cerqueira, A. A. Lazar, and M. Seok, “A 1uW voice activity detector using analog feature extraction and digital deep neural network,” in *2018 IEEE International Solid - State Circuits Conference - (ISSCC)*, Feb 2018, pp. 346–348.

[106] K. Goetschalckx, B. Moons, S. Lauwereins, M. Andraud, and M. Verhelst, “Optimized Hierarchical Cascaded Processing,” *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 8, no. 4, pp. 884–894, Dec 2018.

[107] L. Yang, D. Bankman, B. Moons, M. Verhelst, and B. Murmann, “Bit Error Tolerance of a CIFAR-10 Binarized Convolutional Neural Network Processor,” in *Proceedings - IEEE International Symposium on Circuits and Systems*, 2018.

[108] “tinyML Summit,” https://tinymlsummit.org/.

[109] S. Daniel and W. Pete, *TinyML*. O’Reilly Media, Inc., 2019.

[110] S. Han, J. Pool, J. Tran, and W. Dally, “Learning both Weights and Connections for Efficient Neural Network,” in *Advances in Neural Information Processing Systems*, 2015, pp. 1135–1143.

[111] J. Choi, Z. Wang, S. Venkataramani, P. I.-J. Chuang, V. Srinivasan, and K. Gopalakrishnan, “PACT: Parameterized Clipping

Activation for Quantized Neural Networks," *arXiv:1805.06085 [cs]*, May 2018.

[112] F. Conti, R. Schilling, P. D. Schiavone, A. Pullini, D. Rossi, F. K. Gürkaynak, M. Muehlberghuber, M. Gautschi, I. Loi, G. Haugou, S. Mangard, and L. Benini, "An IoT Endpoint System-on-Chip for Secure and Energy-Efficient Near-Sensor Analytics," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 64, no. 9, pp. 2481–2494, Sep. 2017.

[113] J. Myers, A. Savanth, D. Howard, R. Gaddh, P. Prabhat, and D. Flynn, "An 80nW retention 11.7pJ/cycle active subthreshold ARM Cortex-M0 subsystem in 65nm CMOS for WSN applications," in *2015 IEEE International Solid-State Circuits Conference - (ISSCC) Digest of Technical Papers*, Feb 2015, pp. 1–3.

[114] D. Bol, J. De Vos, C. Hocquet, F. Botman, F. Durvaux, S. Boyd, D. Flandre, and J.-D. Legat, "SleepWalker: A 25-MHz 0.4-V Sub-mm2 7-uW/MHz Microcontroller in 65-nm LP/GP CMOS for Low-Carbon Wireless Sensor Nodes," *IEEE Journal of Solid-State Circuits*, vol. 48, no. 1, pp. 20–32, Jan. 2013.

[115] F. Frustaci, D. Blaauw, D. Sylvester, and M. Alioto, "Approximate SRAMs With Dynamic Energy-Quality Management," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 24, no. 6, pp. 2128–2141, June 2016.

[116] A. Roy, P. J. Grossmann, S. A. Vitale, and B. H. Calhoun, "A 1.3uW, 5pJ/cycle sub-threshold MSP430 processor in 90nm xLP FDSOI for energy-efficient IoT applications," in *2016 17th International Symposium on Quality Electronic Design (ISQED)*, March 2016, pp. 158–162.

[117] L. Lai, N. Suda, and V. Chandra, "CMSIS-NN: Efficient Neural Network Kernels for Arm Cortex-M CPUs," *arXiv:1801.06601 [cs]*, Jan. 2018.

[118] M. Rusci, A. Capotondi, F. Conti, and L. Benini, "Work-in-Progress: Quantized NNs as the Definitive Solution for Inference

on Low-Power ARM MCUs?" in *2018 International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, Sep. 2018, pp. 1–2.

[119] A. Garofalo, M. Rusci, F. Conti, D. Rossi, and L. Benini, "PULP-NN: Accelerating Quantized Neural Networks on Parallel Ultra-Low-Power RISC-V Processors," *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, vol. 378, no. 2164, p. 20190155, 2020. [Online]. Available: https://royalsocietypublishing.org/doi/abs/10.1098/rsta.2019.0155

[120] M. Gautschi, P. D. Schiavone, A. Traber, I. Loi, A. Pullini, D. Rossi, E. Flamand, F. K. Gürkaynak, and L. Benini, "Near-Threshold RISC-V Core With DSP Extensions for Scalable IoT Endpoint Devices," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 25, no. 10, pp. 2700–2713, Oct 2017.

[121] Y. Joseph, "Introduction to Armv8.1-M Architecture," https://pages.arm.com/rs/312-SAX-488/images/Introduction{_}to{_}Armv8.1-M{_}architecture-3.pdf.

[122] T. Karnik, D. Kurian, P. Aseron, R. Dorrance, E. Alpman, A. Nicoara, R. Popov, L. Azarenkov, M. Moiseev, L. Zhao, S. Ghosh, R. Misoczki, A. Gupta, M. Akhila, S. Muthukumar, S. Bhandari, Y. Satish, K. Jain, R. Flory, C. Kanthapanit, E. Quijano, B. Jackson, H. Luo, S. Kim, V. Vaidya, A. Elsherbini, R. Liu, F. Sheikh, O. Tickoo, I. Klotchkov, M. Sastry, S. Sun, M. Bhartiya, A. Srinivasan, Y. Hoskote, H. Wang, and V. De, "A cm-scale self-powered intelligent and secure IoT edge mote featuring an ultra-low-power SoC in 14nm tri-gate CMOS," in *2018 IEEE International Solid - State Circuits Conference - (ISSCC)*, Feb 2018, pp. 46–48.

[123] E. Flamand, D. Rossi, F. Conti, I. Loi, A. Pullini, F. Rotenberg, and L. Benini, "Gap-8: A risc-v soc for ai at the edge of the iot," in *2018 IEEE 29th International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, July 2018, pp. 1–4.

[124] R. Sivalingam, E. Park, and E. Gousev, "Ultra-low Power Always-On Computer Vision," https://rebootingcomputing.ieee.org/images/files/pdf/iccv-2019{_}edwin-park.pdf.

[125] P. D. Schiavone, D. Rossi, A. Pullini, A. D. Mauro, F. Conti, and L. Benini, "Quentin: An Ultra-Low-Power PULPissimo SoC in 22nm FDX," in *2018 IEEE SOI-3D-Subthreshold Microelectronics Technology Unified Conference (S3S)*, Oct. 2018, pp. 1–3.

[126] D. Rossi, A. Pullini, I. Loi, M. Gautschi, F. K. Gürkaynak, A. Bartolini, P. Flatresse, and L. Benini, "A 60 GOPS/W, -1.8V to 0.9V body bias ULP cluster in 28nm UTBB FD-SOI technology," *Solid-State Electronics*, vol. 117, pp. 170 – 184, 2016. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0038110115003342

[127] B. H. Calhoun and A. P. Chandrakasan, "A 256-kb 65-nm Sub-threshold SRAM Design for Ultra-Low-Voltage Operation," *IEEE Journal of Solid-State Circuits*, vol. 42, no. 3, pp. 680–688, March 2007.

[128] F. Frustaci, M. Khayatzadeh, D. Blaauw, D. Sylvester, and M. Alioto, "SRAM for Error-Tolerant Applications With Dynamic Energy-Quality Management in 28 nm CMOS," *IEEE Journal of Solid-State Circuits*, vol. 50, no. 5, pp. 1310–1323, May 2015.

[129] B. Li, P. Gu, Y. Shan, Y. Wang, Y. Chen, and H. Yang, "RRAM-Based Analog Approximate Computing," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 34, no. 12, pp. 1905–1917, Dec 2015.

[130] B. Li, Y. Shan, M. Hu, Y. Wang, Y. Chen, and H. Yang, "Memristor-based approximated computation," in *International Symposium on Low Power Electronics and Design (ISLPED)*, Sep. 2013, pp. 242–247.

[131] G. Tagliavini, D. Rossi, A. Marongiu, and L. Benini, "Synergistic Architecture and Programming Model Support for Approximate Micropower Computing," in *VLSI (ISVLSI), 2015*

IEEE Computer Society Annual Symposium On. IEEE, 2015, pp. 280–285.

[132] G. Desoli, N. Chawla, T. Boesch, S. P. Singh, E. Guidetti, F. De Ambroggi, T. Majo, P. Zambotti, M. Ayodhyawasi, H. Singh, and N. Aggarwal, "A 2.9TOPS/W deep convolutional neural network SoC in FD-SOI 28nm for intelligent embedded systems," *Digest of Technical Papers - IEEE International Solid-State Circuits Conference*, vol. 60, pp. 238–239, 2017.

[133] X. Lin, C. Zhao, and W. Pan, "Towards Accurate Binary Convolutional Neural Network," in *Advances in Neural Information Processing Systems 30*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds. Curran Associates, Inc., 2017, pp. 345–353.

[134] Y. Umuroglu, N. J. Fraser, G. Gambardella, M. Blott, P. Leong, M. Jahre, and K. Vissers, "FINN: A Framework for Fast, Scalable Binarized Neural Network Inference," in *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, ser. FPGA '17. New York, NY, USA: ACM, 2017, pp. 65–74.

[135] K. Ando, K. Ueyoshi, K. Orimo, H. Yonekawa, S. Sato, H. Nakahara, S. Takamaeda-Yamazaki, M. Ikebe, T. Asai, T. Kuroda, and M. Motomura, "Brein memory: A single-chip binary/ternary reconfigurable in-memory deep neural network accelerator achieving 1.4 tops at 0.6 w," *IEEE Journal of Solid-State Circuits*, vol. 53, no. 4, pp. 983–994, April 2018.

[136] L. Jiang, M. Kim, W. Wen, and D. Wang, "XNOR-POP: A processing-in-memory architecture for binary Convolutional Neural Networks in Wide-IO2 DRAMs," in *2017 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*, Jul. 2017, pp. 1–6.

[137] S. Yin, P. Ouyang, J. Yang, T. Lu, X. Li, L. Liu, and S. Wei, "An energy-efficient reconfigurable processor for binary-and ternary-weight neural networks with flexible data bit width," *IEEE*

Journal of Solid-State Circuits, vol. 54, no. 4, pp. 1120–1136, 2019.

[138] Y. Wang, J. Lin, and Z. Wang, “An energy-efficient architecture for binary weight convolutional neural networks,” *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 26, no. 2, pp. 280–293, 2018.

[139] W.-S. Khwa, J.-J. Chen, J.-F. Li, X. Si, E.-Y. Yang, X. Sun, R. Liu, P.-Y. Chen, Q. Li, S. Yu, and M.-F. Chang, “A 65nm 4Kb Algorithm-Dependent Computing-in- Memory SRAM Unit-Macro with 2.3ns and 55.8TOPS/W Fully Parallel Product-Sum Operation for Binary DNN Edge Processors,” in *Proceedings of 2018 IEEE International Solid-State Circuits Conference.*

[140] A. A. Bahou, G. Karunaratne, R. Andri, L. Cavigelli, and L. Benini, “XNORBIN: A 95 TOp/s/W hardware accelerator for binary convolutional neural networks,” *21st IEEE Symposium on Low-Power and High-Speed Chips and Systems, COOL Chips 2018 - Proceedings*, pp. 1–3, 2018.

[141] H. Valavi, P. J. Ramadge, E. Nestler, and N. Verma, “A 64-Tile 2.4-Mb In-Memory-Computing CNN Accelerator Employing Charge-Domain Compute,” *IEEE Journal of Solid-State Circuits*, vol. 54, no. 6, pp. 1789–1799, June 2019.

[142] J. Zhang and N. Verma, “An In-memory-Computing DNN Achieving 700 TOPS/W and 6 TOPS/mm2 in 130-nm CMOS,” *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 9, no. 2, pp. 358–366, June 2019.

[143] D. Bankman, L. Yang, B. Moons, M. Verhelst, and B. Murmann, “An Always-On 3.8 μ J/86% CIFAR-10 Mixed-Signal Binary CNN Processor With All Memory on Chip in 28-nm CMOS,” *IEEE Journal of Solid-State Circuits*, vol. 54, no. 1, pp. 158–172, Jan 2019.

[144] A. Agrawal, A. Jaiswal, D. Roy, B. Han, G. Srinivasan, A. Ankit, and K. Roy, “Xcel-ram: Accelerating binary neural

networks in high-throughput sram compute arrays," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 66, no. 8, pp. 3064–3076, 2019.

[145] A. Waterman, Y. Lee, D. A. Patterson, K. Asanovic, V. I. U. level Isa, A. Waterman, Y. Lee, and D. Patterson, "The risc-v instruction set manual," 2014.

[146] A. Pullini, D. Rossi, G. Haugou, and L. Benini, " μDMA: An autonomous I/O subsystem for IoT end-nodes," in *2017 27th International Symposium on Power and Timing Modeling, Optimization and Simulation (PATMOS)*, Sept 2017, pp. 1–8.

[147] A. Teman, D. Rossi, P. Meinerzhagen, L. Benini, and A. Burg, "Controlled placement of standard cell memory arrays for high density and low power in 28nm FD-SOI," in *The 20th Asia and South Pacific Design Automation Conference*, Jan 2015, pp. 81–86.

[148] B. Moons, D. Bankman, L. Yang, B. Murmann, and M. Verhelst, "Binareye: An always-on energy-accuracy-scalable binary cnn processor with all memory on chip in 28nm cmos," in *2018 IEEE Custom Integrated Circuits Conference (CICC)*, 2018, pp. 1–4.

[149] N. Rathi, A. Agrawal, C. Lee, A. K. Kosta, and K. Roy, "Exploring spike-based learning for neuromorphic computing: Prospects and perspectives," in *2021 Design, Automation Test in Europe Conference Exhibition (DATE)*, Feb 2021, pp. 902–907.

[150] P. Panda, S. A. Aketi, and K. Roy, "Toward Scalable, Efficient, and Accurate Deep Spiking Neural Networks With Backward Residual Connections, Stochastic Softmax, and Hybridization," *Frontiers in Neuroscience*, 2020.

[151] K. Greff, R. K. Srivastava, J. Koutník, B. R. Steunebrink, and J. Schmidhuber, "Lstm: A search space odyssey," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 28, no. 10, pp. 2222–2232, 2017.

[152] A. Sengupta, Y. Ye, R. Wang, C. Liu, and K. Roy, “Going Deeper in Spiking Neural Networks: VGG and Residual Architectures,” *Frontiers in Neuroscience*, 2019.

[153] D. C. Daly, L. C. Fujino, and K. C. Smith, “Through the looking glass-2020 edition: Trends in solid-state circuits from isscc,” *IEEE Solid-State Circuits Magazine*, vol. 12, no. 1, pp. 8–24, 2020.

[154] A. Reuther, P. Michaleas, M. Jones, V. Gadepally, S. Samsi, and J. Kepner, “Survey and benchmarking of machine learning accelerators,” in *2019 IEEE High Performance Extreme Computing Conference (HPEC)*, 2019, pp. 1–9.

[155] M. Spallanzani, G. P. Leonardi, and L. Benini, “Training quantised neural networks with ste variants: the additive noise annealing algorithm,” 2022. [Online]. Available: https://arxiv.org/abs/2203.11323

[156] H. Qin, R. Gong, X. Liu, X. Bai, J. Song, and N. Sebe, “Binary neural networks: A survey,” *Pattern Recognition*, vol. 105, p. 107281, 2020.

[157] M. Bouvier, A. Valentian, T. Mesquida, F. Rummens, M. Reyboz, E. Vianello, and E. Beigne, “Spiking neural networks hardware implementations and challenges: A survey,” *J. Emerg. Technol. Comput. Syst.*, vol. 15, no. 2, Apr. 2019.

[158] A. R. Young, M. Dean, J. S. Plank, and G. S. Rose, “A Review of spiking neuromorphic hardware communication systems,” *IEEE Access*, 2019.

[159] A. Rubino, C. Livanelioglu, N. Qiao, M. Payvand, and G. Indiveri, “Ultra-low-power fdsoi neural circuits for extreme-edge neuromorphic intelligence,” 2020.

[160] M. Davies, N. Srinivasa, T. Lin, G. Chinya, Y. Cao, S. H. Choday, G. Dimou, P. Joshi, N. Imam, S. Jain, Y. Liao, C. Lin, A. Lines, R. Liu, D. Mathaikutty, S. McCoy, A. Paul, J. Tse, G. Venkataramanan, Y. Weng, A. Wild, Y. Yang, and

H. Wang, “Loihi: A neuromorphic manycore processor with on-chip learning,” *IEEE Micro*, vol. 38, no. 1, pp. 82–99, 2018.

[161] F. Akopyan, J. Sawada, A. Cassidy, R. Alvarez-Icaza, J. Arthur, P. Merolla, N. Imam, Y. Nakamura, P. Datta, G. Nam, B. Taba, M. Beakes, B. Brezzo, J. B. Kuang, R. Manohar, W. P. Risk, B. Jackson, and D. S. Modha, “Truenorth: Design and tool flow of a 65 mw 1 million neuron programmable neurosynaptic chip,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 34, no. 10, pp. 1537–1557, 2015.

[162] C. Frenkel, J.-D. Legat, and D. Bol, “A 28-nm convolutional neuromorphic processor enabling online learning with spike-based retinas,” in *2020 IEEE International Symposium on Circuits and Systems (ISCAS)*, 2020, pp. 1–5.

[163] E. M. Izhikevich, “Which model to use for cortical spiking neurons?” *IEEE Transactions on Neural Networks*, vol. 15, no. 5, pp. 1063–1070, 2004.

[164] G. Bellec, F. Scherr, A. Subramoney, E. Hajek, D. Salaj, R. Legenstein, and W. Maass, “A solution to the learning dilemma for recurrent networks of spiking neurons,” *Nature Communications*, vol. 11, no. 1, jul 2020. [Online]. Available: https://doi.org/10.1038%2Fs41467-020-17236-y

[165] J. Pei, L. Deng, S. Song, M. Zhao, Y. Zhang, S. Wu, G. Wang, Z. Zou, Z. Wu, W. He, F. Chen, N. Deng, S. Wu, Y. Wang, Y. Wu, Z. Yang, C. Ma, G. Li, W. Han, H. Li, H. Wu, R. Zhao, Y. Xie, and L. Shi, “Towards artificial general intelligence with hybrid Tianjic chip architecture,” *Nature*, 2019.

[166] J. K. Kim, P. Knag, T. Chen, and Z. Zhang, “A 640m pixel/s 3.65mw sparse event-driven neuromorphic object recognition processor with on-chip learning,” in *2015 Symposium on VLSI Circuits (VLSI Circuits)*, 2015, pp. C50–C51.

[167] S. Hoppner, Y. Yan, A. Dixius, S. Scholze, J. Partzsch, M. Stolba, F. Kelber, B. Vogginger, F. Neumarker, G. Ellguth, S. Hartmann, S. Schiefer, T. Hocker, D. Walter, G. Liu,

J. Garside, S. Furber, and C. Mayr, “The spinnaker 2 processing element architecture for hybrid digital neuromorphic computing,” 2021. [Online]. Available: https://arxiv.org/abs/2103.08392

[168] G. K. Chen, R. Kumar, H. E. Sumbul, P. C. Knag, and R. K. Krishnamurthy, “A 4096-neuron 1m-synapse 3.8-pj/sop spiking neural network with on-chip stdp learning and sparse weights in 10-nm finfet cmos,” *IEEE Journal of Solid-State Circuits*, vol. 54, no. 4, pp. 992–1002, 2019.

[169] 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,” 2017. [Online]. Available: https://arxiv.org/abs/1704.04861

[170] D. Markovic, C. C. Wang, L. P. Alarcón, T. Liu, and J. M. Rabaey, “Ultralow-power design in near-threshold region,” *Proceedings of the IEEE*, vol. 98, no. 2, pp. 237–252, 2010.

[171] A. Alimonda, S. Carta, A. Acquaviva, A. Pisano, and L. Benini, “A feedback-based approach to dvfs in data-flow applications,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 28, no. 11, pp. 1691–1704, 2009.

[172] F. Conti and L. Benini, “A Ultra-Low-Energy Convolution Engine for Fast Brain-Inspired Vision in Multicore Clusters,” *2015 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 683–688, 2015.

[173] C. Schmidt, J. Wright, Z. Wang, E. Chang, A. Ou, W. Bae, S. Huang, A. Flynn, B. Richards, K. Asanović, E. Alon, and B. Nikolić, “4.3 an eight-core 1.44ghz risc-v vector machine in 16nm finfet,” in *2021 IEEE International Solid- State Circuits Conference (ISSCC)*, vol. 64, 2021, pp. 58–60.

[174] I. Miro-Panades, B. Tain, J.-F. Christmann, D. Coriat, R. Lemaire, C. Jany, B. Martineau, F. Chaix, A. Quelen, E. Pluchart, J.-P. Noel, R. Boumchedda, A. Makosiej, M. Montoya, S. Bacles-Min, D. Briand, J.-M. Philippe, A. Valentian,

F. Heitzmann, E. Beigne, and F. Clermidy, “Samurai: A 1.7mops-36gops adaptive versatile iot node with 15,000× peak-to-idle power reduction, 207ns wake-up time and 1.3tops/w ml efficiency,” in *2020 IEEE Symposium on VLSI Circuits*, 2020, pp. 1–2.

[175] D. E. Bellasi and L. Benini, “Smart energy-efficient clock synthesizer for duty-cycled sensor socs in 65 nm/28nm cmos,” *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 64, no. 9, pp. 2322–2333, 2017.

[176] M. Davies, A. Wild, G. Orchard, Y. Sandamirskaya, G. A. F. Guerra, P. Joshi, P. Plank, and S. R. Risbud, “Advancing neuromorphic computing with loihi: A survey of results and outlook,” *Proceedings of the IEEE*, vol. 109, no. 5, pp. 911–934, 2021.

[177] C. Stöckl and W. Maass, “Optimized spiking neurons can classify images with high accuracy through temporal coding with two spikes,” *Nature Machine Intelligence*, vol. 3, no. 3, pp. 230–238, mar 2021. [Online]. Available: https://doi.org/10.1038%2Fs42256-021-00311-4

[178] M. Davies, “Benchmarks for progress in neuromorphic computing,” *Nature Machine Intelligence*, vol. 1, no. 9, pp. 386–388, sep 2019. [Online]. Available: https://doi.org/10.1038%2Fs42256-019-0097-1

[179] H. Shouval, S. Wang, and G. Wittenberg, “Spike timing dependent plasticity: A consequence of more fundamental learning rules,” *Frontiers in Computational Neuroscience*, vol. 4, 2010. [Online]. Available: https://www.frontiersin.org/article/10.3389/fncom.2010.00019

[180] M. Hartley, N. Taylor, and J. Taylor, “Understanding spike-time-dependent plasticity: A biologically motivated computational model,” *Neurocomputing*, vol. 69, no. 16, pp. 2005–2016, 2006, brain Inspired Cognitive Systems. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0925231206001731

[181] B. Nessler, M. Pfeiffer, L. Buesing, and W. Maass, "Bayesian computation emerges in generic cortical microcircuits through spike-timing-dependent plasticity," *PLOS Computational Biology*, vol. 9, no. 4, pp. 1–30, 04 2013. [Online]. Available: https://doi.org/10.1371/journal.pcbi.1003037

[182] S. Guo, Z. Yu, F. Deng, X. Hu, and F. Chen, "Hierarchical bayesian inference and learning in spiking neural networks," *IEEE Transactions on Cybernetics*, vol. 49, no. 1, pp. 133–145, 2019.

[183] N. Srinivasa and Y. Cho, "Self-organizing spiking neural model for learning fault-tolerant spatio-motor transformations," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 23, no. 10, pp. 1526–1538, Oct 2012.

[184] S. R. Kheradpisheh, M. Ganjtabesh, S. J. Thorpe, and T. Masquelier, "Stdp-based spiking deep convolutional neural networks for object recognition," *Neural Networks*, vol. 99, pp. 56–67, 2018. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0893608017302903

[185] S. B. Shrestha and G. Orchard, "SLAYER: Spike layer error reassignment in time," in *Advances in Neural Information Processing Systems 31*, S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, Eds. Curran Associates, Inc., 2018, pp. 1419–1428.

[186] W. Guo, M. E. Fouda, A. M. Eltawil, and K. N. Salama, "Neural coding in spiking neural networks: A comparative study for robust neuromorphic systems," *Frontiers in Neuroscience*, vol. 15, 2021. [Online]. Available: https://www.frontiersin.org/article/10.3389/fnins.2021.638474

[187] S. Park, S. Kim, B. Na, and S. Yoon, "T2fsnn: Deep spiking neural networks with time-to-first-spike coding," 2020. [Online]. Available: https://arxiv.org/abs/2003.11741

[188] Y. Wu, L. Deng, G. Li, J. Zhu, and L. Shi, "Spatio-temporal backpropagation for training high-performance spiking neural

networks," *Frontiers in Neuroscience*, vol. 12, 2018. [Online]. Available: https://www.frontiersin.org/article/10.3389/fnins.2018.00331

Curriculum Vitae

Alfio Di Mauro

Integrated system laboratory (IIS), ETZ J65, ETH Zurich, 8092
Mobile: +(41) 762462469 Email: adimauro@ethz.ch

Born: 09 April, 1992, Catania, Italy Nationality: Italian

Current position

Postdoctoral researcher, Integrated system laboratory (IIS), ETH Zurich

Areas of specialisation

SoC architectures, ASIC design, Event-driven Computing

Education

2014 BACHELOR'S DEGREE IN ELECTRONIC ENGINEERING Politecnico di Torino, Italy
2016 MASTER'S DEGREE IN ELECTRONIC ENGINEERING,

ELECTRONICS SYSTEMS Politecnico di Torino, Italy
2016 MASTER'S THESIS (EXCHANGE), ELECTRICAL AND ELECTRONICS ENGINEERING, Eidgenössische Technische Hochschule Zürich, Switzerland.
2022 DR. SC. ETH ZÜRICH,, Eidgenössische Technische Hochschule Zürich, Switzerland.

Publications

- A. Di Mauro, M. Scherer, D. Rossi and L. Benini, "Kraken: A Direct Event/Frame-Based Multi-sensor Fusion SoC for Ultra-Efficient Visual Processing in Nano-UAVs," 2022 IEEE Hot Chips 34 Symposium (HCS), 2022, pp. 1-19, doi: 10.1109/HCS55958.2022.9895621.

- A. Di Mauro, F. Conti, and L. Benini, "An ultra-low power address-event sensor interface for energy-proportional time-to-information extraction," in 2017 54th ACM/EDAC/IEEE Design Automation Conference (DAC), 2017, pp. 1–6.

- A. Di Mauro, F. Conti, P. D. Schiavone, D. Rossi, and L. Benini, "Always-on 674uw@4gop/s error resilient binary neural networks with aggressive SRAM voltage scaling on a 22-nm IoT end-node," IEEE Transactions on Circuits and Systems I:Regular Papers, pp. 1–14, 2020.

- A. Di Mauro, F. Conti, P. D. Schiavone, D. Rossi, and L. Benini, "Pushing on-chip memories beyond reliability boundaries in micropower machine learning applications," in 2019 IEEE International Electron Devices Meeting (IEDM), 2019, pp. 30.4.1–30.4.4.

- A. Di Mauro, M. Scherer, J. F. Mas, B. Bougenot, M. Magno, and L. Benini, "Flydvs: An event-driven wireless ultra-low power visual sensor node," in 2021 Design, Automation Test in Europe Conference Exhibition (DATE), Feb 2021, pp. 1851–1854.

- A. Di Mauro, A. Suravi Prasad, Z. Huang, M. Spallanzani, F. Conti, and L. Benini, “Sne: an energy-proportional digital accelerator for sparse event-based convolutions,” 2022, design, Automation and Test in Europe Conference (DATE 2022); Conference Location: Online; Conference Date: March 14-23, 2022; Conference lecture held on 22 March 2022

- A. Di Mauro, D. Rossi, A. Pullini, P. Flatresse, and L. Benini, “Temperature and process-aware performance monitoring and compensation for an ulp multi-core cluster in 28nm utbb fd-soi technology,” in 2017 27th International Symposium on Power and Timing Modeling, Optimization and Simulation (PATMOS), 2017, pp. 1–8.

- A. Di Mauro, F. Zaruba, F. Schuiki, S. Mach, and L. Benini, “Live demonstration: Exploiting body-biasing for static corner trimming and maximum energy efficiency operation in 22nm fdx technology,” in 2020 IEEE International Symposium on Circuits and Systems (ISCAS), 2020, pp. 1–1.

- A. Di Mauro, D. Rossi, A. Pullini, P. Flatresse, and L. Benini, “Live demonstration: Body-bias based performance monitoring and compensation for a near-threshold multi-core cluster in 28nm fd-soi technology,” in 2018 IEEE International Symposium on Circuits and Systems (ISCAS), 2018.

- A. Di Mauro, D. Rossi, A. Pullini, P. Flatresse, and L. Benini, “Performance-aware predictive-model-based on-chip body-bias regulation strategy for an ulp multi-core cluster in 28 nm utbb fd-soi,” Integration, vol. 72, pp. 194–207, 2020.

- A. Di Mauro, H. Fatemi, J. P. de Gyvez, and L. Benini, “Idleness-aware dynamic power mode selection on the i.mx 7ulp iot edge processor,” Journal of Low Power Electronics and Applications, vol. 10, no. 2, 2020.

- A. Pullini, D. Rossi, I. Loi, A. D. Mauro, and L. Benini, “Mr. Wolf: A 1 GFLOP/s Energy-Proportional Parallel Ultra Low Power SoC for IOT Edge Processing,” in ESSCIRC 2018 - IEEE

44th European Solid State Circuits Conference (ESSCIRC), Sep. 2018, pp. 274–277.

- P. D. Schiavone, D. Rossi, A. Pullini, A. Di Mauro, F. Conti, and L. Benini, “Quentin: an ultra-low-power pulpissimo soc in 22nm fdx,” in 2018 IEEE SOI-3D-Subthreshold Microelectronics Technology Unified Conference (S3S), 2018, pp. 1–3.
- D. Rossi, F. Conti, M. Eggimann, A. D. Mauro, G. Tagliavini, S. Mach, M. Guermandi, A. Pullini, I. Loi, J. Chen, E. Flamand, and L. Benini, “Vega: A ten-core soc for iot endnodes with dnn acceleration and cognitive wake-up from mram-based stateretentive sleep mode,” IEEE Journal of Solid-State Circuits, pp. 1–1, 2021.
- D. Rossi, F. Conti, M. Eggiman, S. Mach, A. D. Mauro, M. Guermandi, G. Tagliavini, A. Pullini, I. Loi, J. Chen, E. Flamand, and L. Benini, “4.4 a 1.3tops/w @ 32gops fully integrated 10-core soc for iot end-nodes with 1.7µ w cognitive wake-up from mram-based state-retentive sleep mode,” in 2021 IEEE International Solid- State Circuits Conference (ISSCC), vol. 64, 2021, pp. 60–62.
- P. D. Schiavone, D. Rossi, A. Di Mauro, F. K. G¨urkaynak, T. Saxe, M. Wang, K. C. Yap, and L. Benini, “Arnold: An efpga-augmented risc-v soc for flexible and low-power iot end nodes,” IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 29, no. 4, pp. 677–690, 2021.
- H. Okuhara, A. Elnaqib, D. Rossi, A. Di Mauro, P. Mayer, P. Palestri, and L. Benini, “An energy-efficient low-voltage swing transceiver for mw-range iot end-nodes,” in 2020 IEEE International Symposium on Circuits and Systems (ISCAS), 2020, pp. 1–5.
- M. Hersche, E. M. Rella, A. Di Mauro, L. Benini, and A. Rahimi, “Integrating event-based dynamic vision sensors with sparse hyperdimensional computing: A low-power accelerator with online learning capability,” in Proceedings of the ACM/IEEE International Symposium on Low Power Electronics and Design,

ser. ISLPED '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 169–174.

- A. Garofalo, G. Ottavi, A. Di Mauro, F. Conti, G. Tagliavini, L. Benini, and D. Rossi, "A 1.15 tops/w, 16-cores parallel ultra-low power cluster with 2b-to-32b fully flexible bit-precision and vector lockstep execution mode," in ESSCIRC 2021 - IEEE 47th European Solid State Circuits Conference (ESSCIRC), 2021, pp. 267–270.