Diss. ETH No. 27094

Design of energy-efficient RISC-V-based edge-computing devices

A thesis submitted to attain the degree of

DOCTOR OF SCIENCES of ETH ZURICH
(Dr. sc. ETH Zurich)

presented by

PASQUALE DAVIDE SCHIAVONE
MSc CE Polytechnic of Turin, Italy

born on September 18th, 1989
citizen of Italy

accepted on the recommendation of

Prof. Dr. Luca Benini, examiner
Prof. Dr. Luca Carloni, co-examiner

2020

Acknowledgments

I do not know whether there is a "template" for the Ph.D. *Acknowledgments*, but it does not matter. I take this space to let *Davide* express his gratitude in whatever shape it takes. Written at night as in a Romantic movie, but with artificial lights and no candles. Well, a bit of a template must be followed. I think I have to start with the Ph.D.-related people.

A very big "thank you" goes to my boss during the last 5 years. Prof. Luca Benini. Thanks a lot, Luca. You believed in me since the beginning of my adventure. You made me feel part of the group from the first to the last day. Your valuable advice made me grow up as an engineer, a researcher, and a man. I learned not only the technical and writing aspects of this job, but also how to handle stress periods, prioritize tasks, and take my responsibilities, resulting in a higher self-confidence.

Another big "thank you" must go to Davide Rossi. Since the beginning, you have been part of my path, helping me overcome obstacles, boosting my motivations, and sharing funny moments, trips, experiences. I cannot say how much you helped me during the Ph.D. I wonder whether I could make it without you! Thanks!

I could not be here writing these pages if Prof. Massimo Poncino did not accept me for the Master Thesis. I spent it at the Integrated System Laboratory at ETH Zurich, where all started. Thanks, Massimo.

Thanks, Antonio Pullini, you gave me so much advice and inspired me deeply.

Thanks to my office mates and friends, Florian, Fabian, and Stefan. Having a pleasant working environment is crucial for everyday working life. I will not forget that our office was the funniest.

Thanks to M. Gautschi, F. Conti, F. Gurkaynak, M. Schaffner, B. Muheim, G. Haugou, E. Flamand, M. Magno, A. Di Mauro, A. Traber, I. Loi, P. Vogel, and all the other colleagues with whom I could work closely in several projects. I learned a lot from you, and I hope it was a pleasant journey for you. Thanks also to the lunch-crew Giorgio, Alessandro, Gabriele, and Sara.

I also had the privilege to make true friends to share not only the Ph.D. life but also the life outside. Endless conversations, dinners, nights out, trips, etc. Thanks, Antonio, Giovanni, and Nello. And thanks, Andrea, you even had to live with me for 18 months! Not an easy task, he can tell you.

I could spend part of my Ph.D. at Imperial College London at the Centre for Bio-Inspired Technology, hosted by Prof. Timothy Constandinou. I want to thank the whole group for making me feel part of the crew since the first day. It was significant for me, and I liked the experience so much. Even in London, I could meet new friends. A special thanks to Dorian, Federico, and Bruno.

...And now it is time to go out of office.

I have always been firmly convinced that to perform in life, you must be surrounded by people that love you and that you love. They give you meanings in life. I feel so honored and lucky to have many people to thank. Now it is your turn.

Thanks, Jacopo, far for ten years but the closest and best friends one can imagine. A brother from another mother. Endless Whatsapp audios, deep conversations, and funny moments gave me so much energy to go through every moment of my Ph.D. and life. I love you so much. There is not enough space to tell you how lucky I am for having you always present for me.

Grazie Stefano, credi sempre in me e mi rallegri sempre con la tua spensieratezza. Hai sempre creduto in me sin dall'inizio. Spesso credi in me più di quanto lo faccia io stesso. Ti voglio bene zio. Ma dove si trovano altri amici così?

Thanks, Andrea. We share similar interests in life and work. I admire your ambitions and passions, which often inspire me. I love you, bro.

Jacopo, Stefano, Andrea, the friends I chose when I still had baby teeth, and yet here with me at the age of 31.

Thanks, Alessio, Claudio, Patrick, David, Ricky, Alex, and Cristian. Sorted from the one I know for the longest. Close friends to share everyday life, social moments, trips, good and bad periods. Friends I count for everything. Thanks for making my life so rich and full of moments.

Thanks, Marco, Petra, Ersilia, Matthew, Jorge, Martha, and Greta, for simply being my friends, sharing social life, dancing, dinners, and giving me feelings.

Thanks to all the other people I did not mention here but part of my life.

To conclude, I devote the thesis to my parents. Lucia and Luigi. This is not probably the place to express all my gratitude, but I will never be able to repay the love you gave me. Thanks a lot, mamma e papà for everything you are. Whatever I became, it's all for you. You are my model. I love you so much. And thanks to Marco, my brother and best friend, that was a young man when I started the Ph.D. and now a father of two amazing guys, Alberto and Beatrice. Thanks to all the members of my big and close family.

(In Italian for them)

Questa tesi è dedicata ai miei genitori. Lucia e Luigi. Non è questo il posto giusto per esprimere la mia gratitudine nei vostri confronti, ma non potrò mai ripagare l'amore che mi avete sempre dato. Grazie mamma e papà per tutto quello che siete per me. Se sono diventato quello che sono è grazie a voi. Siete il mio esempio. Vi voglio tanto bene. E grazie Marco, fratello e migliore amico, che eri un ragazzo quando ho cominciato e ora padre di due bellissimi bambini, Alberto e Beatrice. Grazie a tutta la mia famiglia.

Abstract

The "Era of Connection" is the current historical period where an increasing number of people are connected with other people and with many different everyday objects augmented by artificial intelligence. Such objects are computing systems connected in the Internet-of-things (IoT), and share common features as: they are typically battery-powered; they receive data from sensors, elaborate them, and transmit the output of the elaboration to the network wirelessly.

Elaborating information already allows reducing data to transmit and thus traffic from the millions of edge-nodes to the central servers, saving resources in the whole network infrastructure.

The advance on technology scaling is enabling miniaturization of such edge-computing devices, which become more complex by aggregating more sensors information and by elaborating them with smarter algorithms. For these reasons, energy efficiency became the most critical constraint, as it bonds performance, needed by smart algorithms, and power, constrained by the batteries' capacities and life-time expectations.

Near-threshold Computing (NTC) conclusively demonstrated that digital system-on-chips (SoCs) are more energy-efficient when supplied "near" the transistor threshold-voltage, where the compound of leakage and dynamic energy achieves the minimum point. Although effective for what concerns power due to the quadratic dependency with the

supply voltage, frequency is degraded. Thus, parallel computing is used to recover performance by dividing the task among different computing engines that run at a slower speed.

Edge-computing SoCs that are programmable are usually preferred over fixed-function circuits as they have shorter time-to-market and higher versatility, which makes the product life-time longer and adaptable to fast-changing pattern recognition algorithms. Open-source IPs are also preferred due to lower costs, trust, and re-usability.

Software programmable devices are based on microcontroller units (MCUs), which are typically composed of a central processing unit (CPU), memories, and peripherals.

In this thesis, we propose Instruction-Set extensions to an open-source CPU based on RISC-V to boost energy efficiency on data processing algorithms, and on the other hand, we propose a new CPU to boost energy efficiency on control processing tasks on always-on-domain that are typically waiting for events.

When performance requirements are beyond the capabilities of single-CPU systems, MCUs are extended with accelerators. As for the SoCs, these accelerators can be fixed-function circuits or programmable.

In this work, we use a programmable multicore accelerator to process physiological data efficiently; we extend a neural-network accelerator to work with single-instruction-multiple-data operations by leveraging packed vectors; and we implement an MCU extended with a binary neural-network accelerator in 22nm technology node. The latter can leverage ultra-low-voltage to push even further error-resilient applications. We show how we build accelerators to boost energy-efficiency on power-constrained devices thanks to optimized software, computer architecture design, and implementation on advanced technology nodes.

Embedded Field-Programmable Gate-Arrays (eFPGAs) are programmable logic that can be integrated into SoCs to augment their functionalities. They are programmable with soft-hardware and reside between software and fixed-function accelerators. They can implement new custom peripherals or specific-function accelerators. Their reconfigurability enables versatility and longer product life-time as software, and at the same time, their parallel nature allows to implement functions faster than software accelerators.

Finally, in this work, we propose an MCU extended with an eFPGA implemented in 22nm technology node. We show that in the power budget typical of edge-computing devices, the system can achieve higher performance and energy efficiency without giving up versatility and the advantages of reconfigurability.

Riassunto

L'Era in cui l'uomo vive è oggi è caratterizzata da millioni di dispositivi connessi in una rete (Internet-delle-Cose) che scambiano dati. Tali dispositivi elettronici sono spesso alimentati a batteria, la cui durata è limitata dalla continua trasmissione di tali dati a centri di elaborazione.

Anticipare parte dell'elaborazione sui dispositivi stessi permette di comprimere i dati, riducendo la mole di pacchetti da trasmettere, e quindi il consumo di energia. Questo paradigma è chiamato *edge computing*, dove i dati vengono elaborati vicino ai sensori che li ha prodotti.

Spostare parte dell'elaborazione sui dispositivi vicino ai sensori richiede che questi eseguano algoritmi complessi, in un tempo relativamente breve, in un contesto dove la potenza è limitata dalla batteria (pochi mWs).

I microcontrollori sono tra i dispositivi più utilizzati in questo contesto in quanto programmabili via software, che rende più semplice l'implementazione di tali algoritmi. Inoltre la loro versabilità permette il loro riutilizzo, diminuendo così i costi di produzione.

La maggior parte dei microcontrollori disponibili sul mercato sono basati su un singolo processore con un consumo limitato di potenza. Il limite principale di questo tipo di dispositivi è la loro capacità di calcolo, che non permette l'esecuzione di algoritmi più complessi che

elaborano segnali proveniente, ad esempio, da una camera o da un insieme numeroso di segnali fisiologici.

Questo lavoro esplora alcune tecniche per estendere la capacità di calcolo dei microcontrollori pur mantenendo un consumo di potenza limitato.

Essendo il processore l'elemento principale dei microcontrollori, questo lavoro presenta alcune estensioni ad un processore open-source per renderlo piu efficiente energenticamente durante l'esecuzione di algoritmi dominati da operazioni di processamento di dati. Molte applicazioni dell'Internel-delle-Cose sono caratterizzate da una fase attiva, tipicamente breve, e una fase inattiva dominante. I microcontrollori quindi sono dotati di stati-di-potenza dove consumano il minimo durante la fase inattiva, e dove invece ottimizzano le performance in fase attiva, ottenendo cosi un consumo di potenza medio ridotto. Durante la fase inattiva, non è sempre possibile minimizzare il consumo di potenza spegnendo completamente il microcontrollore. È dunque necessario ridurre al minimo il consumo di potenza di perdita ottimizzando l'area della parte sempre attiva. In questo lavoro sono presentati due nuovi processori open-source ottimizzati per l'area e potenza di perdita al costo di performance minori.

I risultati mostrano che il processore piu efficiente energicamente è quello che ha l'architettura piu adeguata per le applicazioni per cui è stato pensato. Quindi un processore molto piccolo e limitato al massimo in performance per l'esecuzioni di compiti puramente di controllo, un processore con un set di istruzioni esteso per l'esecuzione di algoritmi di elaborazione di segnali, e in fine, un processore con capacita di calcolo limitate per l'esecuzioni di algoritmi che hanno sia una parte di controllo che di computazione. Vedremo anche che in un contesto dove il processore è sempre alimentato dalla tensione, il processore con la migliore efficienza energetica dipende anche dal periodo con cui tale applicazione viene eseguita.

Per finire, questo lavoro presenta un sistema di virifica innovativo che sfrutta un ottimizzatore per creare programmi in linguaggio assembly che massimizzato la copertura del codice del dispositivo da verificare. Tale sistema è stato in grado di trovare 10 non-funzionalità nel processore testato.

Oltre al processore, per ottenere efficieze energetiche molto alte è importante ottimizzare anche il sistema attraverso un'architettura

efficiente, ma anche l'utilizzo di tecnologie avanzate per sfruttare un ampio spettro di tensione di alimentazione e il body-biasing. Questo lavoro propone un microcontrollore implementato nella tecnologia Globalfoundries GF22FDX (GF22). Il microcontrollore proposto sfrutta un ampio spettro di tensione e sfrutta la tensione di bulk per aumnetare le perfromance a dispetto del consumpo di potenza. L'efficienza energetica e le performance sono misurate sia a tensioni nominali, sia a tensioni prossime a quella di soglia, dove i transistori ottengono la loro efficienza massima. I risultati mostrano anche l'effetto della tensione di substrato. Inoltre, il microcontrollore proposto offre un'architettura di memoria eterogeneo, che puo' essere sfruttato per aumnetare performance ed efficienza energetica in cambio della capacita' massima di memoria grazie alla diminuizione di energia di perdita. Questo lavoro mostra anche come un applicazione resiliente agli errori possa sfruttare un aggressivo calo della tensione sulle memorie statiche (al di sotto della soglia di funzionamento) per ottenere consumini di potenza ridottissimi in contesti dove il microcontrollore e' sempre attivo.

Quando la potenza di un solo processore non basta, il microcontrollore puo essere esteso con degli accelrratori. Tali acceleratori sono utilizzati per aumentare le performance durante un compito specifico, per cui tali acceleratori sono altamente specializzati per raggiungere la massima efficienza energetica e performance. In questo lavoro vengono esaminati gli acceleratori integrati in dispositivi per l'Internet-delle-Cose progettati con funzionalita' fissa o come IP programmabili.

Un acceleratore specifico per le Convolutional Neural Network (CNN) e' stato esteso per supportare l'esecuzione di convoluzioni in parallelo su dati raprresentati con dimensionalita' ridotta. Tale acceleratore e' stato integrato in un microcontrollore con altri quattro processori in tecnologia a 65 nm. I risultati mostrano che tale acceleratore e' piu efficiente e piu performante anche di quattro processori che eseguono lo stesso compito in parallelo. Quando la precisione dei pesi della reta viene ridotta da 16 a 8 o 4 bit, l'acceleratore raggiunge effiienze energetiche ancora piu alte senza compromettere significamente l'accuratezza dell'applicazione.

Quando invece i task da accelerare possono cambiare velocemente, un acceleratore programmabile e' una soluzione migliore per ridurre i tempi e costi di implementazione. Nel campo delle neuro-tecnologie ad

esempio, i segnali proventienti dal cervello hanno frequenze e risoluzione spaziale diversa, per cui gli algoritmi per estrarne le informazioni differenscono di molto. Questo lavoro propone un microcontrollore esteso con un acceleratore costituito da otto processori per comprimere i dati proventienti da un sensore di potenziali d'azione provenitne dai neuorini del cervello. Tale microcontrollore sfrutta i diversi stati di potenza del microcontrollore, usando tutto la potenza durante la compressione di tali dati, mentre per l'acquisizione e processamento di eventi, il microcontrollore minimizza il consumpo di potenza durante la loro attesa, per poi attivare un singolo processore per classificare tali eventi. La soluzione proposta ha cosi un potenza proporzionale ai dati acquisiti. Inoltre, per processare segnali da un elettroencefalogramma, un microcontrolore con quattro processori e' stato utilizzato per estrarre informazioni nel dominio della frequenza ottenendo un efficienza energetica migliore rispetto ai microcontrollori standard.

Per completare l'analisi delle possibili soluzioni per implementare acceleratori in ambito dei dispositivi di processamento dei dati vicino ai sensori, questo lavoro propone un microcontrollore esteso con una FPGA integrata. L'utilizzo di una FPGA integrata e' utile in quanto combina il vantaggio della programmabilita' con la naturale efficienza dell'hardware. Applicazioni dove l'utilizzo di tali risorse e' ottimale, sono ad esempio, periferiche personalizzate o acceleratori di media complessita per reti binarie o algoritmi di sicurezza. Questo lavoro propone un microcontrollore implementato in tecnologia GF22 composto da un processore e una FPGA connessa al sistema attraverso diverse interfacce come per comunicare con le memorie e il sistema di input/output per offire la massima flessibilita' al programma-utente. Inoltre, tale FPGA puo sfruttare la tensione di substrato per diminuire la potenza quando non e' utilizzata, mantenendo comunque attivo il suo stato interno. Questo lavoro mostra il consumo di energia, potenza e performance del microcontrollore in un ampio spettro di tensione, dimostrandosi il migliore in performance ed efficienza energetica rispetto ad altre soluzioni proposte in letteratura.

Contents

1 Introduction **1**
1.1 Contribution and Publications 19
1.2 Parallel Ultra-low Power Platform 22

2 RISC-V CPUs **27**
2.1 RISC-V Microarchitectures 30
2.1.1 *Riscy* 30
2.1.2 *Zero-riscy* 42
2.1.3 *Micro-riscy* 43
2.2 Toolchain Support 44
2.3 Experimental results 45
2.3.1 Area 45
2.3.2 Workloads 47
2.3.3 Performance 49
2.3.4 Power estimation 51
2.3.5 Energy calculation 54
2.3.6 Always-On activity 56
2.4 *Riscy* Verification Framework 58

3 FDSOI MPU **65**
3.1 Quentin Architecture 66
3.1.1 Memory Subsystem 66

3.1.2 I/O subsystem 67
3.1.3 Clock subsystem 68
3.1.4 Accelerator subsystem 68
3.1.5 Chip Implementation and Results 68
3.1.6 Error-resilient application use-case 74

4 Fixed-Function Accelerators 79
4.1 Hardware Convolution Engine SIMD Extensions . . . 80
4.2 Fulmine chip 82
4.3 Experimental evaluation 84
4.3.1 System-on-Chip Operating Modes 84
4.3.2 HWCE Performance and Power Evaluation . . 86

5 Software Accelerators 89
5.1 *Neuro-PULP* 92
5.2 *Neuro-PULP* Case Study Applications 93
5.2.1 *Compress-Stream* application 95
5.2.2 *Spike-Sorting* application 97
5.3 Drowsiness Detection on the PULP architecture 100
5.4 *Neuro-PULP* results 102
5.5 Drowsiness acceleration results 103

6 Soft-Hardware Accelerators 107
6.1 Arnold Architecture 108
6.1.1 Memory Subsystem 111
6.1.2 I/O subsystem 111
6.1.3 Clock subsystem 112
6.1.4 eFPGA subsystem 112
6.2 eFPGA Software and Tools 115
6.3 Arnold Physical Design 116
6.3.1 Performance and Energy Efficiency 118
6.4 Use Cases 122
6.4.1 I/O subsystem accelerator 122
6.4.2 Custom I/O interface 123
6.4.3 CPU subsystem accelerator 124

7 Summary and Conclusion **129**
7.1 Overview and main results 132
7.1.1 RISC-V CPUs 132
7.1.2 FDSOI MPU 133
7.1.3 Fixed-Function Accelerators 134
7.1.4 Software Accelerators 135
7.1.5 Soft-Hardware Accelerators 136
7.2 Outlook 137

A Notation and Acronyms **139**
Acronyms 139

Bibliography **143**

Curriculum Vitae **165**

Chapter 1

Introduction

In the *Era of Connection*, every object collects data and sends them to the central servers where hidden information is extracted, creating the Internet-of-things (IoT).

Such objects, or end-nodes of the IoT, are for example: measuring instruments like weight-scales, thermometers, speed controllers; data concerning online shopping items for food or any kind of good; cameras for surveillance, face recognition or to capture important moments of life; microphones to record the voice; watches to count the number of steps or the heart-rate; mobile phone keyboards that collects written words in a text; mobile phone applications that count the Calories eaten by the users; health tracker for Electrocardiography (ECG), Electromyography (EMG) or Electroencephalography (EEG); etc.

Information that can be extracted from these raw data is much. From simple statistics like averages or distributions (e.g., the probability that a male person of 180 cm is 80 kg, or the probability that a 34 years old woman buys a new book if she just bought a new pair of sunglasses), to automatic face tag in social networks like Facebook, voice control of objects like Alexa or Google Home, the likelihood of

the word "is" when preceded by the word "what" in a text, or when a ECG stream contains an arrhythmia.

Data-mining and artificial intelligence are tools to extract such information from raw data. Usually, raw data are processed to extract the features that are correlated to the information the most. For example, bio-physical data such as ECG, EMG or EEG, carry information in their frequency-domain representation. Whereas video or images need more complex features extracted by Convolutional Neural Networks (CNNs).

As these objects are usually battery-powered, it is essential to preserve energy for extended battery life and limit their peak power consumption below the maximum battery capacity (mW range). Size is also important, as such objects are usually wearable. Finally, performance is also crucial as data needs to be sent to the main smart servers without losing them.

When many raw data are sent, most of the end-node power consumption is spent on the transmission antenna, which is active most of the time, limiting the battery life of such nodes [1].

One way to overcome this limit is introduced by the edge-computing paradigm, where end-nodes evolve from simple data collectors into smart devices able to extract features, thus sending to the IoT denser information [2,3]. Sending denser information makes the time spent on transmission shorter, thus the battery-life time longer. However, as the end-nodes need computing capabilities, this paradigm holds if the extra power consumed by the computing part of the object is smaller than the antenna transmission power. Edge-computing is beneficial not only for the energy efficiency of the end-node per se but also across the whole IoT infrastructure, where switches, routers, and finally the servers, need to process less, pre-selected data, saving power, and bandwidth.

Denser information can be extracted by applying simple cropping, like, for example, reducing the size of images or applying more complex transformation like extracting pre-selected features, all the way to apply on-the-edge classification and send to the servers simple tags to collect statistics.

However, edge-computing devices require the execution of arithmetic-intensive algorithms typical of the data-mining and artificial intelligence domain. Such computations have to be performed relatively fast to meet performance and quality of service requirements (e.g., images/s to

achieve high-quality real-time face recognition, prediction of a seizure in 1 second in epileptic patients), and in a limited power budget (1-100 mW). Last but not least, a device with higher energy efficiency (operations/ mW) is desirable to increase the density of information extracted or the accuracy and prolong the battery lifetime.

These devices tend to look more and more like complete systems-on-node, including sensors, microprocessors, specialized Hardware (HW), memories, and wireless transceivers capable of operating autonomously for several years [4].

Depending on the constraints of the application such as flexibility, performance, power, and cost, IoT computing platforms can be implemented as hardwired, fixed-function Application-Specific Integrated Circuits (ASICs), programmable HW (or soft-hardware) on field-programmable gate arrays (FPGAs), or as Software (SW) programmable on Micro Controller Units (MCUs).

1. hardwired fixed-function ASICs;
2. soft-hardware on FPGAs;
3. software programmable on MCUs.

Hardwired fixed-function ASICs are devices highly-optimized for a specific application and usually achieve the best energy efficiency. However, they lack versatility and require long time-to-market [5]. For example, it may be difficult, if not impossible, to use a hardwired fixed-function ASIC for face recognition done with CNN inferences to detect sleep stages in EEG data streams. Hence, their usage is preferred in highly standardized applications or specialized single-function products with tight Power-Performance-Area (PPA) and energy constraints.

For example, as CNNs require Billions of operations to be computed in short time, many fixed-function ASICs for CNNs at ultra-low power budget have been proposed. Orlando [6] achieves up to 2.9 Top/s/W; Origami [7] 803 Gop/s/W; whereas YodaNN [8], *BRein* [9], *XNOR-POP* [10], *Conv-RAM* [11] and Khwa et al. [12] achieves energy efficiency in the range of 10-50 TOP/s/W using in-memory computing.

Due to the wide applications of the edge-computing IoT domain, programmable platforms are preferred for longer product lifetime and time-to-market.

FPGAs offer versatility after fabrication via hardware programmability, and they allow exploiting spatial computations typical of ASICs designs, as opposed to sequential execution. For these reasons, FPGAs are used in a wide range of applications, from machine learning [13–15], sorting [16], and cryptography accelerators for data centers [17], to smart instruments [18], Analog-to-Digital converters (ADCs) [19], to low-power systems for wearable applications [20], control-logic systems [21], and for implementing smart-peripherals connected to System-On-Chips (SOCs) [22, 23].

FPGAs range from high-end FPGAs used to accelerate high-performance workloads to ultra-low-power, small, and low-cost technology implementations.

High-end FPGAs, such as the Xilinx Virtex Ultrascale devices [24] and the Intel Cyclone 10 GX device [25], have millions of LUTs, flip-flops, Digital Signal Processing (DSP)-blocks, and Static Random Access Memory (SRAM) macros containing Mbytes of memory. To extended their capabilities in the embedded application domain running software, such FPGAs are often programmed with soft-CPUs [26]. The users can implement a deeply pipelined core with multiple issues to achieve high performance, or a tiny soft-core with a small area footprint for control applications [27,28], and offload part of the control functionalities executed in SW to the soft-CPU. For example, Choi et al. [29] presented an FPGA-based 20 k-Word speech recognizer using a Xilinx Virtex-4 FPGA where the computationally less demanding tasks are executed in SW, whereas the rest of the algorithms is accelerated in HW.

As soft-cores are limited in performance [30] and occupy resources, FPGAs are often extended with hard-CPUs as application processors (usually ARM-based embedded processors such as the Xilinx Zynq-7000 SoC [31] and Intel Arria V SoC [32], in the case of Microsemi PolarFire [33] RISC-V processors). As a result, high-end FPGAs have typical power consumption in the order of tens of Watts [34], and they are usually used as high-performance accelerators on servers connected via Ethernet or PCI interfaces [35].

In the low-power domain, FPGAs are typically realized with a less aggressive process than high-end FPGAs. They are usually smaller, cheaper, and as a result, have lower performance than the others. Examples are the Microsemi IGLOO nano [36], which has up to 3 k

logic elements[1], or the Lattice Semiconductor iCE40 UltraLite [37], which has more than 1K of LUTs+flip-flops. Both consume from a few uW to hundreds of mW. These FPGAs are used to extend the Input/Output (IO) subsystem of embedded controllers [38], even with simple data pre-processing engines to lower the bandwidth coming from sensors [20, 23]. FPGAs can also be extended with CPUs to leverage HW/SW co-designed IoT nodes in the low-end space. An industrial RISC-V based soft-core is provided by the Microsemi Mi-V RV32, ready to be integrated into the SmartFusion2 SoC [39] or in the IGLOO FPGA [40] in an area footprint of 10 k-26 k LEs. Other RISC-V based solutions have emerged during the *RISC-V SoftCPU Contest* in December 2018, with the VexRiscv soft-core as the winner. Hard-CPUs are also used as in the Microsemi SmartFusion2 SoC in 65nm [39], which proposes an MCU-class (ARM Cortex-M) core running at 166 MHz and an FPGA with DSP blocks and up to 150 k logic elements, 656 kB[2] of memory, and power consumption in the order of hundreds of mWatts. Examples that use the Microsemi SmartFusion2 SoC can be found in Gomes at al. [41], which proposes a system where most of the tasks are executed by the ARM core, whereas the FPGA is used for accelerating critical network kernels. In Fournaris et al. [42], the operating system, and user interfaces run in software, whereas the FPGA is used to collect sensor data, extract features, and to calculate the nearest neighbor on the extracted information. The system runs at 160 MHz consumes 4.96 mW on the CPU part and 153.97 mW on the FPGA side. While their power consumption is within the range of IoT applications, these FPGAs are limited in performance and thus not suitable for computationally intensive applications.

Even if they are usually less energy-efficient than hardwired fixed-function ASICs and FPGAs, MCUs are often chosen as the baseline for IoT devices as they offer high versatility, requiring only software revisions, enabling short time-to-market and IP cheaper re-usability costs.

To close the gap with ASICs and FPGAs PPA and energy results, different optimizations at IP, architecture, and implementation level

[1] One logic element is composed of one 4-input LUT and one flip-flop
[2] 512 Bytes of Non-Volatile Memory

can be made on MCUs to leverage their advantages without giving up performance.

In this Thesis, we show optimizations at IP level on Central Processing Units (CPUs) and accelerators, as well as at architectural level by efficiently integrating accelerator or building memory subsystems, and at implementation level, by exploiting advanced technology to optimize PPA and energy results.

The majority of IoT MCUs use a single-core. Depending on the applications target domain, some MCUs host a simple, small, ultra-low-power CPU for controlling and light-weight processing, or an enhanced CPU for high computation demanding applications. Single-issue in-order cores with a high Instruction Per Cycle (IPC) are typically more energy-efficient as no operations have to be repeated due to mispredictions and speculation [43].

Most Off-the-Shelf (OTS) MCUs use energy-efficient CPUs based on ARM Cortex-M family of cores. For example, the Cortex-M0 [44], which is meant for running applications that need minimal power and area, or the Cortex-M4 [45], which provides Instruction Set Architectures (ISAs) to target signal processing algorithms and many more. The Cortex-M0 is a single-issue in-order core with three pipeline stages and optional single-cycle 32 bit multiplier unit, whereas the Cortex-M4 is a single-issue in-order core with three pipeline stages and a branch speculation engine, single-cycle 32 bit Multiply And Accumulate (MAC) unit, 8/16 bit Single Instruction Multiple Data (SIMD) instructions and an optional floating-point unit. Examples of such systems are the NXP i.MXRT1050 [46], the STMicroelectronics STM32L476xx family [47], or the Silicon Labs EFM32 Giant Gecko 11 [48], all featuring a power budget within a few tens of mW. Synopsys provides the DesignWare ARC configurable processors [49], based on the ARCv2 ISA to target power- and area-constrained embedded systems, as well as the ARC EM DSP for ultra-low-power embedded applications requiring advanced signal processing capabilities. The latter support an extended ARCv2DSP ISA and provide MAC, fixed-point and SIMD instructions.

Solutions based on the open-source RISC-V ISA [50] are also becoming popular. An open-source ISA is a desirable starting point for an IoT core, as it can potentially decrease dependency from a single IP provider and cut cost, while at the same time allowing freedom

for application-specific instruction extensions. In this Thesis, all the solutions proposed are based on open-source ISA CPUs, particularly on RISC-V.

In the high-performance side of IoT applications, the CVA6 (formerly Ariane [51]) has recently joined OpenHW Group from ETH Zurich, Boom [52] and the Rocket [53] come instead from UC Berkely. High-performance cores are out of the scope of this Thesis and are comparable to Application class cores from ARM, for example, those able to run a complex application such as operating systems like Linux. In the low-power edge devices domain we can find CPUs for control purposes, like the PicoRV32, a small configurable RISC-V core with high frequency but also low IPC [54], the Z-Scale, a small 32 bit three stages single-issue in-order core, [55], and the LowRISC Ibex core (formerly *Zero-riscy* from ETH Zurich [56]). Whereas in the computing domain we can find the OpenHW Group CV32E40P (formerly *Riscy* [57]). Both *Zero-riscy* and *Riscy* are discussed in Chapter 2 and are part of the contribution of this Thesis.

Single-core systems can be used for both control and signal processing applications. For signal-processing systems we can find for example: Benatti et al. [58], who proposes an EMG gesture recognition system based on Cortex-M4 to extract salient features out of the EMG signals and classify them with a Support-Vector-Machine (SVM); Imtiaz et al. [59] proposes a system to detect epileptic seizures using an ultra-low-power Texas Instrument MSP430 MCU [60] as the main computation device; Barsakcioglu et. al. [61] instead used a Cortex-M0+ to extract features from 32 brain Action Potential (AP) channels and classify them. For control applications, where the CPU is used to coordinate and control several peripherals or other computational engines, we can find for example: Konijnenburg et al. [62], where a multi-sensor acquisition system with a Cortex-M0 is used to interact with a sensor readout chip and a system of hardware accelerators; Ickes et al. [63], where a 16 bit core controls memory-mapped accelerators for Finite-Impulse-Response (FIR) filters and fast Fourier transform (FFT); Pullini et al. propose Mr.Wolf [64], where the *Zero-riscy* core coordinates the peripherals in the always-on domain, and a software-programmable accelerator made of 8 *Riscy* cores.

However, the application profile (data-intensive or control-oriented) is not the only variable that matters when selecting a CPU.

Code-size, for example, is directly influenced by the ISA selected, thus saving the area at CPU level may not transfer at system level due to an increase of instruction memory requirements. An analysis of RISC-V code-size on different applications of the IoT domain has been presented in [65].

On duty-cycled systems, the event frequency that triggers the computation is also an essential variable to select the most efficient CPU. In particular, when events are rare in always-on domains where power-gating techniques are not implemented or wake-up times are too long, the leakage power plays the most significant contribution to the final energy efficiency.

In Chapter 2, an analysis of three different cores is presented, showing that smaller cores are better than DSP-enhanced core even on data-intensive applications when such computation does not have tight performance constraints and has rare trigger events.

CPUs used for data-intensive applications are usually more performant and feature more complex ISA and datapath components.

As it is discussed in Chapter 2, DSP and SIMD ISA extensions enable higher computational capabilities still maintaining the target power consumption of the mW range. With respect to pure DSPs, ISA extensions keep the CPU compatible with the standard ISA and the programming model is kept unchanged, allowing for SW re-usability and compatibility. In particular, SIMD extensions are interesting as in the IoT domain most of the sensors (like cameras or ADCs) uses 16 bit or less, to represent the data, thus multiple of them can be embedded to increase computational capabilities. For example, the ARM Cortex-M4 [45] supports both DSP and SIMD functionalities while remaining energy-efficient (32.8 µW/MHz in 90 nm low power technology [45]).

In Chapter 2, DSP, SIMD, as well as bit manipulations ISA extensions are implemented in the *Riscy* core, showing their advantages on computational-intensive algorithms typical of the edge-computing domain. However, such extensions increase the CPUs area and power. Thus such processors should be used when computations are highly demanded. Techniques to reduce the area and power consumption of such CPUs exploit resource sharing whenever possible and to use strategies as power-gating (used more often in high-performance CPUs), clock-gating and operands isolation.

The complexity and heterogeneity of digital devices used in embedded systems are increasing every day, and delivering a bug-free design is still a very complicated task. The interest for open-source hardware in real products demands tools and advanced methodologies for verification to provide high reliability to open and free IPs.

Some surveys state that validation, verification, and testing (VV&T) require about 60% [66] of the total production costs. Companies approaching open-source IPs usually verify their functionality internally and can provide reports or fixes to the IP designers to improve the quality of the free hardware. For instance, recently, the *Riscy* core has been compared and chosen to be a valid candidate in an industry project by Google [67]. For that reason, it has also been extensively verified using STING [68], a versatile design verification platform.

In Chapter 2, we exploited an open-source evolutionary optimizer called μGP [69] to create assembly programs that optimize the code coverage of the *Riscy* core. The final programs are then evaluated both on Device Under Verification (DUV) and on the reference model. We used the *Riscy* Instruction-Set Simulator (ISS) as reference model. To increase the code coverage, we split the assembly program generation by specializing every evolution to a particular subset of the DUV. An external module that randomly generates external events that cannot be triggered in software, such as interrupts or memory stalls, has been added to push the code coverage higher. Finally, the generation and evaluation phases of the tandem verification framework have been merged so that all the individuals of each generation of the evolution are evaluated on both the DUV and the reference model. All of these optimizations helped uncover crucial bugs in the *Riscy* multiplier and the forwarding and stall logic related to the load-and-store unit. The bugs have been reported and fixed, proving the usefulness of open-source hardware in the industry context and the needs of advanced verification strategies.

The architecture and the implementation of the MCUs are crucial to reach a high level of energy efficiency. To be interfaced with different external components, MCUs usually offer a large variety of peripherals such as I2C, UART, SPI, and GPIOs. The architecture and runtime software should allow for techniques to reduce the energy consumed at a single IP level and system level [70]. The most common approach to reduce the average power consumption, widely used in commercial

MCUs, is duty cycling, where the system is kept in off/idle/sleep mode whenever is waiting for an event to happen, and it switches to on/run/active mode when such event triggers, to start the computation. The idle states are characterized by a dominating leakage power consumption, while the active part can be dominated either by the dynamic or static power, depending on technology, running frequency, and supply voltage. Power-gating techniques allow to shut down part of the system that are not needed during idle states, while the retentive parts of the system (like memories with instructions or sensitive data) are supplied with minimum voltage, to reduce as much as possible the leakage power. In the active mode, the system can exploit techniques such as Dynamic-Voltage-Frequency Scaling (DVFS) and clock-gating and operands isolation to reduce power consumption. For example, in Pullini et al. [64], the Mr.Wolf system ranges from tens of μW in the idle states to hundreds of mW in the active states. In Bol et al. [71], where the system consumes hundreds of nW per kB in idle mode, and 144 mW in the active state at the minimum energy point. However, duty-cycle requires triggers to reduce the number of false-positive activations, which could be somewhat challenging to achieve in a real scenario in applications such as pattern recognition. For example, Liu et al. [72] proposes an ADC that generates even only when detects spikes in brain APs, with a true-positive detection rate of 93%. Thus, such sensors can be used to wake up systems that process such spikes, as we show in Chapter 5.

In active mode, the system should run at its best energy-efficient point, characterized by the minimum voltage supply that allows the system to run at the target frequency. Scaling voltage together with frequency allows improving the energy efficiency of computation significantly, by exploiting the quadratic dependency of dynamic power with supply voltage. In general, Near-Threshold Computing (NTC) is the best energy-efficient point [73]. However, aggressive voltage scaling has a significant impact on performance and SRAM reliability. Accelerators are often used to recover performance degradation. Examples of MCUs that exploit NTC and DVFS can be found in [74, 75]. Technology plays a significant role in the energy consumed by the MCUs. Transistor selection can be used to trade power-performance results at design-time by selecting low, regular, or high threshold voltage transistor flavors. Fully Depleted Silicon-On-Insulator (FDSOI)

technology offers promising power-performance trade-offs and the possibility to adapt the energy consumption at runtime by leveraging adaptive body-bias to lower the power consumption (reverse body-bias - RBB) or increase performance (forward body-bias - FBB). Examples of MCUs that exploit FDSOI and body-bias can be found in [71,76–78].

In Chapter 3, an MCU implemented in Globalfoundries GF22FDX (GF22) technology is proposed. Such MCU efficiently embeds the *Riscy* core presented in Chapter 2, it is optimized to reach high energy efficiency coupling NTC and body-bias to select a wide power-performance operating range. It features an autonomous IO subsystem optimized to deal with the wide variety of sensors available in IoT end-nodes, and a heterogeneous (Standard Cell based Memory (SCM) and SRAM) architecture to better exploit the low-voltage capabilities of GF22 technology and to trade leakage power and memory capacity both in active and idle states. In addition, the MCU hosts a Binary Neural-Network (BNN) accelerator to boost performance and energy efficiency beyond the capabilities of a single CPU. BNNs are robust to random bit-level noise, making aggressive voltage scaling attractive as a power-saving technique for both logic and SRAMs. The system is the most performant and energy-efficient among State-of-the-art (SOA) related solutions.

Albeit CPUs performance coupled with power-saving and energy efficiency strategies such as duty-cycling and DVFS allow for software programmable end-nodes for the IoT, there are still applications that need higher performance and energy efficiency than what a single CPU can offer.

For example, deep CNNs are leading feature extraction and classification algorithms in fields such as computer vision (e.g. object detection [79], scene parsing [80], and semantic segmentation tasks [81]) and audio signal analytics [82]. CNNs are characterized by many billions of MAC operations, which is often too high for a single CPU.

For this reason, edge-computing devices can be augmented with accelerators used to compute the high compute-intensive tasks. In MCUs, they are used to alleviate the CPU from specific kernels, and they can be used either in parallel (on different data), or in a time-multiplexed fashion.

Accelerators can be coupled in IoTs devices as external or integrated devices. The MCU sees external accelerators as peripherals, and

communication and data exchange between the two engines is usually not efficient as it requires slow and power-hungry communication. The focus of this Thesis is instead on integrated tightly-coupled accelerators, that share with the CPU the MCU on-chip memories to minimize the time and energy spent in data exchange.

Similarly to edge-computing devices, accelerators can be implemented as:

1. fixed-function circuits;
2. software programmable.
3. embedded FPGAs (eFPGAs);

Accelerators are usually bigger, faster, and more power-consuming, but more energy-efficient than a single CPU. To mitigate their power overhead, they are typically used in a duty cycling fashion, where they become active on the specific part of the application (e.g., during the computation of the kernel they need to accelerate), and kept in sleep mode the rest of the time. When possible, NTC is used during the active time of the accelerated kernel to reduce the power further and increase energy-efficiency.

Fixed-functions circuits are kernel-specific functional blocks. They can be programmed by memory-mapped operations or by specific CPU instructions (co-processors). There exist several MCUs extended with custom accelerators, for example: the BNN accelerator [83] integrated in the MCU discussed in Chapter 3; the cryptography engine integrated into [48]; or GAP8 from GreenWaves Technologies [84] that embeds a CNNs accelerator, the chip proposed by Intel [85], where a x86 processor cooperates with a dedicated functional units for CNN and cryptography workloads, and many others like frequency-domain-transforms [86], linear algebra [87], security engines [88]. In the biomedical context for seizure detection, it has been shown that it is possible to speed up FFT by a dedicated hardware block, which is controlled by an MCU [89].

As CNN are de-facto standard feature extractor and classifier of image and video processing pattern-recognition tasks, and as they are characterized by tens of billions of operations of a net such as ResNet-18 [90] or Inception-v3/v4 [91, 92] on devices with a power

budget of a few mW, particular interest is spent on accelerators to speed up CNN kernels.

To meet such constraints, researchers focused on reducing: the *number of elementary operations*, with smaller networks [93] and techniques to prune unnecessary parts of the network [94]; the *cost of an elementary compute operation*, by lowering the complexity of elementary operations [95,96]; and the *cost of data movement*, again by reducing the size of CNNs and taking advantage of locality whenever possible [97].

On solution to reduce the cost of elementary computations and the cost of data movement is reducing the number of bits to represent both the CNN weights and activations to 32, 16, 8, all the way to 1 bit in BNNs [98, 99].

The most computational dominating part of CNNs is the linear transformation, i.e., the 2D-convolution, where million of weights are multiplied by the input layers. In Chapter 4, customizations to a fixed-function accelerator integrated in an MCUs have been implemented to push energy efficiency during CNN applications further. In particular, SIMD extensions are implemented in a CNN accelerator to exploit the error-tolerant nature of CNNs, thus enabling such engine to implements four parallel 16x4 bit, or two parallel 16x8 bit, or one 16x16 bit convolution. The datapath of the CNN can thus exploit operations on multiple weights in parallel, achieving 4x higher performance when 4 bit weights are used instead of 16. This approach also avoids the requirement of specific training for binary connected networks, while accuracy, throughput, and energy efficiency can be traded using different weight formats. Such accelerator has been integrated into an MCU implemented in 65nm, next to four DSP-enhanced cores. Results show that higher energy efficiency and performance are achieved when compared to a cluster of four cores at near-threshold.

Instead, software-programmable accelerators are not specialized for a specific kernel, but rather optimized for a given domain. These are processing engines that execute functions described in software with a high-level language such as C or C++. Thus the functions they accelerate are not known ahead at design time. These accelerators are implemented as DSP units or more efficient CPUs in MultiProcessor

SoCs (MPSoCs). For example, DSP units are special-purpose processors designed to accelerate general signal processing algorithms as: time series processing (FFT, Discrete Wavelet Transform (DWT)); pattern recognition algorithms (SVM, Logistic Regression, Neural-Networks); or data-compression algorithms (compressed sensing, Huffman coding). DSPs are processors with instruction-set, microarchitecture, and memory model highly specialized for signal processing. When integrated into an MCU, users have to deal with the different toolchains used to compile code that runs in the CPU and code that runs in the DSP. Instead, when different CPUs co-exist, they can either run the same ISA with a different microarchitecture, or implement different ISA extensions of the main one. In the first case, for example, a CPU can be optimized for performance and another for low-power. In the context of high-performance IoT devices, this is, for example, the case of the ARM big.LITTLE architecture [100], where a cluster of high-performant cores is used when speed is required; otherwise, a cluster of low-power cores are used.

In edge-computing devices for the IoT, the always-on parts are characterized typically by control-flow execution and do not require to run data-intensive algorithms, that can be offloaded to accelerators, as their main purpose is to wait for events coming from the peripheral subsystem or to schedule tasks. Architectural heterogeneity provides a possible solution to harmonize these competing constraints as fast execution achieved by more efficient accelerators, and low leakage cores for always-on domains; the availability of different cores optimized for diverse tasks, but able to run the same code is advantageous for IoT devices. In the context of heterogeneous ISA for instance, the Texas Instrument CC2650 [101] has one ARM Cortex-M3 to control the system and execute applications and one dedicated ARM Cortex-M0 to control only the wireless transmission. NXP, TI, and other vendors offer a core optimized for control task like the Cortex-M0+ and a Cortex-M3 or -M4 for more computationally demanding tasks to achieve the best energy efficiency in a broader set of tasks [101, 102]

For more data-intensive algorithms, which require higher performance or energy efficiency, multicore architectures can be used, especially on kernels that are highly parallelizable like the data-mining and pattern-recognition ones. Neves et al. proposed a multicore system with SIMD capabilities for biomedical sensors [103]. Benatti et al. [104]

use a multicore cluster to extract complex features from EEG data to detect seizures. The Greenwave's GAP8 [84] uses a single *Riscy* core to handle peripherals and the computation of light-processing tasks, whereas a cluster of 8 *Riscy* cores is used to accelerate complex computations. Differently from GAP8, in Pullini et al. [64], Mr.Wolf leverages a heterogeneous architecture that uses one *Zero-riscy* to control tasks in the always-on domain, and a cluster of 8 *Riscy* cores as an accelerator. These systems can choose to divide the workload as a subset of processors to meet the performance target at the lowest energy budget [56].

These systems have both the flexibility of MCUs and competitive performance and efficiency when dealing with computationally intensive tasks. Software programmable systems are usually less performant than their ASICs or FPGAs counterparts due to the sequential nature of software-defined functions against the parallel nature of hardware implementations. However, the wide variety of applications in the IoT scenarios requires programmable systems for short time-to-market, versatility costs, and product lifetime.

For example, in the context of neuronal technology, the breadth of interfacing techniques and experimental methods is reflected in the different electrode types and signals acquired from the nervous system (e.g., intra-/extra-cranial, frequency bands, amplitude, etc.), the number of signals recorded (spatial resolution) and their associated processing requirements. Local Field Potential (LFP) and EEG signals contain macro-scale activity over many neurons and are widely used in applications such as seizure or drowsiness detection [104–106]. These signals are typically sampled at $\leq 1\,\text{kHz}$, and information is extracted by time-frequency analysis. Extracellular APs are used to study individual neuronal (or circuit) activity, acquired with intra-cranial probes at frequencies $\geq 10\,\text{kHz}$. Spike sorting [107] is the process that follows the acquisition phase to differentiate between action potentials from each of the neurons sensed by an electrode. This task requires implantable neural acquisition devices that have to be ultra-low power to avoid tissue damage ($<80\,\text{mW/cm}^2$ [108]), and for extended battery lifetime. Moreover, wireless transmission of brain activity to enable free movements and a high number of channels is desirable to analyze single-neuron activity across a wide region of the brain [109].

The variety of signal processing kernels, as well as intra and inter subjects variability [110, 111], push for highly versatile platforms.

In Chapter 5, we exploit a RISC-V platform derived from the Mr.Wolf MCU presented in Pullini et al. [64] coupled with an event-based 64-channel ADC for APs presented in [72]. The event-based ADC allows the MCU to exploit its power-state in a duty-cycled fashion. Also, the pre-computation of the ADC allows the system to use a single-core only to sort the detected AP spikes. Whereas when the system is used in stream mode, the higher number of data to process is offloaded to a cluster of 8 cores. In addition, to show the versatility of software-programmable accelerators, another task in the context of EEG signal is accelerated in a cluster of 4 cores derived by another ultra-low-power MCU [77]. The software-accelerated solution allows for reaching higher energy efficiency compared to SOA systems that try to solve the same problem.

Increased integration density of modern SOCs allowed a reasonably sized FPGAs array to be integrated as part of an on-chip system. eFPGAs are FPGA IP cores specifically meant to be integrated into SOCs to extend them with programmable logic. Unlike the FPGAs, eFPGAs are not meant to be used standalone but are designed to enhance the capabilities of the SOCs. Vendors provide tools to allow eFPGAs to be customized to the SOCs and properties like the number of arrays, with a given number of LUTs, DSP blocks, flip-flops, IO pins, etc. can be configured. They can be used to enable post-silicon soft-hardware programmable functions in SOCs or MCUs to make updates on accelerators or custom peripherals. Hardwired accelerators or peripherals outperform their eFPGA-based implementations, but lack flexibility and post-fabrication reconfigurability. The benefit of integrating eFPGAs into SoCs is the possibility to increase performance by specializing the SOCs for one particular domain that can change over time, increasing the product lifetime and application span.

eFPGAs can be provided as soft-IP [112, 113], described in RTL and synthesized with the rest of the system, or hard-IP [114–116] as hard-macros with pre-determined physical layout, featuring a different trade-off between performance and cost. Although soft eFPGA macros are easily portable from different technology nodes as they are made by standard cells, hard-macro eFPGAs, which are usually custom-designed at layout level, feature significantly better PPA figures.

For example, in Renzini et al. [112], a soft-IP is complementing a MCU for power control applications is implemented using a 90 nm Bipolar CMOS DMOS (BCD) technology. This eFPGA is relatively small (only 96 4-input LUTs and 192 flip-flops) and connected exclusively to the IO subsystem to implement low-latency and flexible control tasks such as Pulse Width Modulation (PWM). Several companies are providing hard-IP blocks, as Achronix [117], which provides 7nm FinFET eFPGAs, Flex-Logix [118], which provides from 12 nm to 180 nm eFPGAs macros, QuickLogic Corporation [119], which provides from 22 nm to 65 nm core IPs, and Menta [113], which provides IPs from 10 nm to 90 nm. Several heterogeneous reconfigurable SOCs have been presented in the last years, ranging from high-performance systems to low-power embedded systems. Whatmough et al. presented a 25 mm^2 SOC implemented in 16 nm FinFET technology featuring two ARM A53 cores, a quad-core datapath accelerator, 4 MBytes on-chip SRAM, and a 2x2 FlexLogic eFPGA macro featuring hardwired DSP slices [114].

In the embedded domain, several solutions have been proposed in different technology nodes. Borgatti et al. [115] implemented a 180 nm 2 0mm^2 SOC, where eFPGA is integrated with the CPU pipeline to implement a reconfigurable Application Specific Instruction Processor (ASIP) SoC, with the eFPGA implementing custom instructions. Also, the eFPGA is connected to the system bus and IO pads. The system reports up to 10x performance gain using instruction extensions to accelerate face-recognition algorithms and 2x for IO intensive tasks when dealing with camera peripherals with pre-processing. Lodi et al. [116] implemented a 42 mm^2 SOC in 130 nm, where the CPU pipeline is directly connected with the eFPGA to implement custom instructions, whereas a second eFPGA is connected to the system bus and IO pads. The system reports up to 15x performance gain and 89% energy saving by exploiting the eFPGAs to accelerate a set of data processing algorithms. However, as a consequence of using a mature technology node, the eFPGAs (~15 kGE) presented in the proposed SoCs feature limited capabilities and performance.

To boost signal processing workloads, both hard and soft eFPGAs can have DSP-blocks included in the IP itself, or they can have pins dedicated to communicating with external blocks, featuring, once again, a different trade-off between time to market for DSP-blocks

customization at design time. The first ones can be used by eFPGA synthesis tools to map user-designs in DSP-blocks implicitly, whereas in the second case, the user explicitly designs logic in the eFPGA to interact with the external blocks.

In Chapter 6, we propose a RISC-V SOC featuring an advanced microcontroller (based on *Riscy* and on the MCU presented in Chapter 2) augmented by an eFPGA for IoT applications in GF22 process technology. The MCU is based on the one presented in Chapter 3, but it replaces the BNN accelerator with an eFPGA. We demonstrate the flexibility of the SOC to tackle the challenges of many emerging IoT applications, such as *(i)* interfacing sensors and accelerators with non-standard interfaces, *(ii)* performing on-the-fly pre-processing tasks on data streamed from peripherals, and *(iii)* accelerating near-sensor analytics, encryption, and machine learning tasks. A unique feature of the proposed SOC is the exploitation of body-biasing to reduce leakage power of the eFPGA fabric, achieving SOA state bitstream-retentive sleep power for the eFPGA fabric. Figure 1.1 shows how the related work on IoT devices is split, and topics faced in the various Chapters of this Thesis.

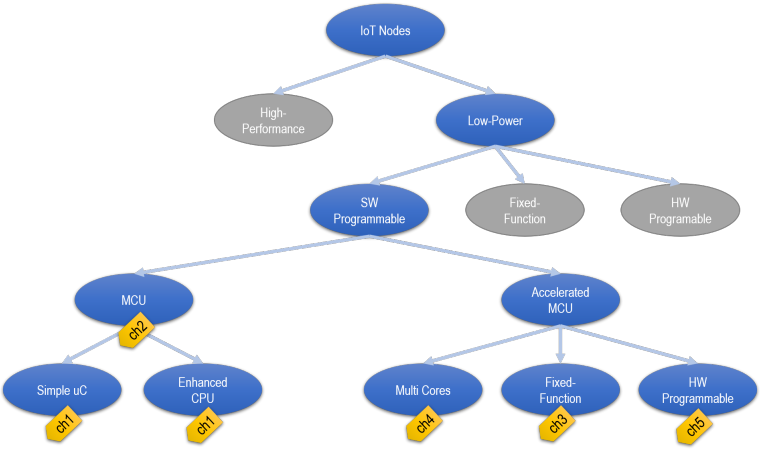


Figure 1.1: Map of the Thesis in the context of the IoT devices.

1.1 Contribution and Publications

The contribution of this Thesis can be summarized as follow.

In Chapter 2, we extended a RISC-V CPU to be more performant and energy-efficient, within a limited power and area overhead, when executing data-analytics algorithms typical of the IoT domain. We optimized two RISC-V CPUs to be smaller and less power consuming that are ideal when used to execute control-flow tasks (e.g., peripheral and power managers) typical of MCUs. We evaluated the CPUs under different timing-constraints and operating voltage, showing the energy efficiency of the three cores executing three different kernels. We show that optimized area cores are preferred for always-on domains that are seldomly in active states. Finally, we leveraged a tandem verification framework that uses an evolutionary optimizer to create assembly programs that aim to optimize the code coverage of the DUV, and a reference model (the ISS) to evaluate the correctness of the DUV. In this framework, the test creation and evaluation phases are merged to increase the probability of finding bugs.

In Chapter 3, we implemented a RISC-V based MCU that hosts a DSP-enhanced CPU, it exploits a heterogeneous memory subsystem, and it embeds a hardwired BNN accelerator. Such MCU has been implemented in GF22 FDSOI technology; it has three different operational modes to trade memory capacity and energy efficiency in active mode or to optimize power consumption in idle mode in a duty-cycled scenario. Thanks to the FDSOI, the MCU can exploit a full voltage operational range to enable DVFS and NTC. Also, FBB is used for trading performance and power consumption. When error-tolerant applications such as BNNs are executed on the accelerator, aggressive voltage scaling can be leveraged to trade accuracy and power consumption in tight constrained or always-on scenarios. We show that thanks to the compound of CPU performance, MCU architecture, and technology that can exploit efficiently DVFS and FBB, the proposed chip outperform single-core MCU in terms of energy efficiency and performance, withing a low-power budget.

In Chapter 4, we extended a hardwired CNN accelerator to work on parallel data of packed weights of 4, 8, and 16 bit. As CNNs are error-tolerant, negligible accuracy loss can be traded to increase

performance and energy efficiency. We integrated it in a heterogeneous multicore MCU together with another accelerator implemented in UMCL 65nm technology. We evaluated the CNN accelerator energy and performance against a DSP-enhanced CPU and a cluster of four DSP-enhanced CPUs on a wide range of supply voltage. We show that the hardwired accelerator's best energy efficiency is achieved as the specialized architecture is lightened of software overheads.

In Chapter 5, we exploit an MCU extended with a software accelerator to achieve high performance, energy efficiency, and versatility in bio-applications that deal with *i)* high-frequency, high-dimension brain APs to perform spike detection and compression, and *ii)* low-frequency, low-dimension EEG signals to extract complex features to perform drowsiness detection. In the first case application, we leverage an ADC that can generate events and an MCU implemented in TSMC 40nm that has different power states to enable efficient duty cycling, and a cluster of eight DSP-enhanced CPUs to perform efficient computations. We show that thanks to the software accelerator and power states, MCU-based implementations of neuro-applications are outperforming MCU based solutions, offering at the same time high-versatility, in a low-power budget. Besides, we exploit a software accelerator implemented in ST FDSOI 28nm technology made of four DSP-enhanced CPUs to extract frequency components of EEG signals at near-threshold to achieve the highest energy efficiency among SOA related solutions.

Finally, in Chapter 6, to evaluate all the possible integrated accelerator solutions in the low-power domain of edge-computing devices, we implemented an MCU augmented with an eFPGA implemented in GF22 technology. Such MCU hosts a DSP-enhanced CPU and an eFPGA fabric. The eFPGA can be connected to the rest of the system with flexibly with different plugs as: by shared memory, IO Direct Memory Access (DMA), or General-purpose Input/Output (GPIO). The system exploits FBB to achieve higher performance on the MCU, and RBB to achieve lower power consumption on the eFPGA. We evaluated the system on a wide range of supply voltage to evaluate DVFS and NTC. We show that it is possible to embed an eFPGA in a MCU within a low-power budget for battery-powered edge-computing devices. The MCU achieves SOA performance and energy-efficiency,

and thanks to RBB, it is possible to minimize the power overhead of the eFPGA macro, still maintaining its state.

(a) [56] P. D. Schiavone et al., "Slow and steady wins the race? A comparison of ultra-low-power RISC-V cores for Internet-of-Things applications." 2017 27th International Symposium on Power and Timing Modeling, Optimization and Simulation (PATMOS). IEEE, 2017.

(b) [57] M. Gautschi et al. "Near-threshold RISC-V core with DSP extensions for scalable IoT endpoint devices." IEEE Transactions on Very Large Scale Integration (VLSI) Systems 25.10 (2017): 2700-2713.

(c) [120] P. D. Schiavone et al., "An Open-Source Verification Framework for Open-Source Cores: A RISC-V Case Study," 2018 IFIP/IEEE International Conference on Very Large Scale Integration (VLSI-SoC), Verona, Italy, 2018, pp. 43-48, doi: 10.1109/VLSI-SoC.2018.8644818.

(d) [121] P. D. Schiavone et al., "Quentin: an Ultra-Low-Power PULPissimo SoC in 22nm FDX," 2018 IEEE SOI-3D-Subthreshold Microelectronics Technology Unified Conference (S3S), Burlingame, CA, USA, 2018, pp. 1-3, doi: 10.1109/S3S.2018.8640145.

(e) [83] F. Conti et al. "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 37.11 (2018): 2940-2951.

(f) [122] A. D. Mauro et al., "Pushing On-chip Memories Beyond Reliability Boundaries in Micropower Machine Learning Applications," 2019 IEEE International Electron Devices Meeting (IEDM), San Francisco, CA, USA, 2019, pp. 30.4.1-30.4.4, doi: 10.1109/IEDM19573.2019.8993434.

(g) [123] A. D. Mauro et al., "Always-On 674uW @ 4GOP/s Error Resilient Binary Neural Networks with Aggressive SRAM Voltage Scaling on a 22nm IoT End-Node." arXiv preprint arXiv:2007.08952, 2020.

(h) [124] F. Conti et al. "An IoT endpoint system-on-chip for secure and energy-efficient near-sensor analytics." IEEE

Transactions on Circuits and Systems I: Regular Papers 64.9 (2017): 2481-2494

(i) [125] P. D. Schiavone et al., "Neuro-PULP: A Paradigm Shift Towards Fully Programmable Platforms for Neural Interfaces," 2020 2nd IEEE International Conference on Artificial Intelligence Circuits and Systems (AICAS), Genova, Italy, 2020, pp. 50-54, doi: 10.1109/AICAS48895.2020.9073920.

(j) [105] Kartsch, Victor Javier, et al. "A sensor fusion approach for drowsiness detection in wearable ultra-low-power systems." Information Fusion 43 (2018): 66-76.

(k) [126] P. D. Schiavone et al., "Arnold: an eFPGA-Augmented RISC-V SoC for Flexible and Low-Power IoT End-Nodes." arXiv preprint arXiv:2006.14256, 2020.

Figure 1.2 shows the energy optimizations faced in the and topics faced in the various Chapters of this Thesis.

1.2 Parallel Ultra-low Power Platform

In this Section, the PULP architecture is described as it has been used as a baseline to build the technical part of this Thesis.

The PULP architecture is an MCU that extensively exploits parallel computing at near-threshold voltage to improve energy efficiency. The energy efficiency is increased thanks to the quadratic dependency of dynamic power with supply voltage, while performance degradation due to *low-voltage* operation is compensated by using multiple processor cores [127, 128].

A brief description of its architecture is following. Interested readers are referred to [64, 97, 129, 130] for more details about the PULP platform.

The last version of PULP is based on RISC-V, and it is composed of a single-core MCU, acting like a fabric controller that handles the IO subsystem, and executes controls tasks. Such single-core MCU is referred to as *PULPissimo*. PULPissimo can be extended with a software accelerator composed of a multi-core cluster with an indipendent voltage and frequency domain. The PULP cluster supports a configurable number of cores with a shared instruction cache and

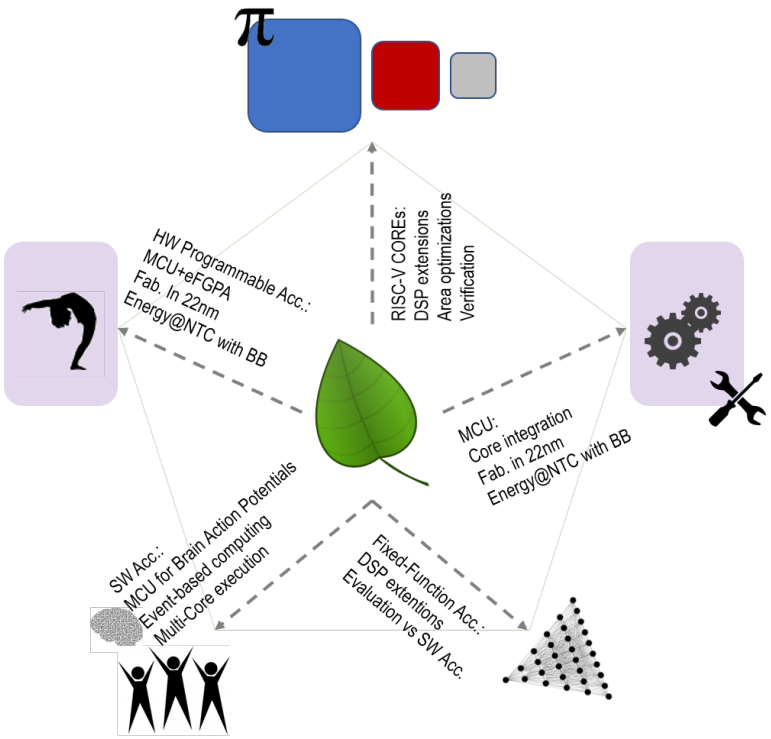


Figure 1.2: Energy efficiency optimizations faced in the Chapters of this Thesis.

scratchpad data memory. Figure 1.3 shows a generic PULP MCU with a cluster in a configuration with eight cores and 16 TCDM-banks. Different implementations have been taped out since the PULP project started. They differ in the number of cores, instruction cache, L1 tightly coupled memory composition, etc. A shared I$ is used to reduce the cost per core, leveraging the single-program-multiple-data nature of most parallel near-sensor processing application kernels. A tightly coupled DMA engine manages transfers between IO and L2 memory and the shared TCDM. Data access pattern predictability of

key application kernels, the relatively low clock frequency target, and the tightly constrained area and power budget make a shared TCDM preferable over a collection of coherent private caches [131].

The data memories can be configured to be split in area-efficient SRAM, and energy-efficient SCM blocks. Since SCMs are built of standard cells, it is possible to scale the supply voltage and operate near the threshold voltage of transistors [132].

Both the PULP cluster and PULPissimo can be extended with hardwired fixed-function accelerators or eFPGAs, as discussed in this Thesis.

The single-core part of the PULP MCU hosts an autonomous IO subsystem described in [133]. The core runs the runtime or a light operating system as freeRTOS or zephyrOS, it hosts the debug modules, and it manages the power and frequency of the whole MCU. The core that runs on PULPissimo does not feature any instruction cache, and it is directly connected to the L2 memory.

The L2 memory is composed of an interleaved part made of 4 banks and a non-interleaved part made of 2 banks. The interleaved part is generally used by the autonomous IO subsystem to store data acquired from sensors, that are then processed by the cluster or dedicated accelerators.

The non-interleaved part (private) is used to store the code of the runtime (or light operating system) and private core data (as the stack, and runtime data structures). In this way, bank conflicts are minimized as the stack-data, and instruction code of the core are stored in the two private banks, whereas the shared data are in the interleaved part of the system.

The PULP MCU has been successfully taped out with OpenRISC, and RISC-V cores [64, 97, 129, 130], and in the version proposed in [130] achieves a top energy efficiency at Near-Threshold (NT) of 193 MOps/mW in 28 nm FDSOI technology, whereas, in the version proposed in [64], it achieves a top performance of 7000 MOps in 40 nm technology.

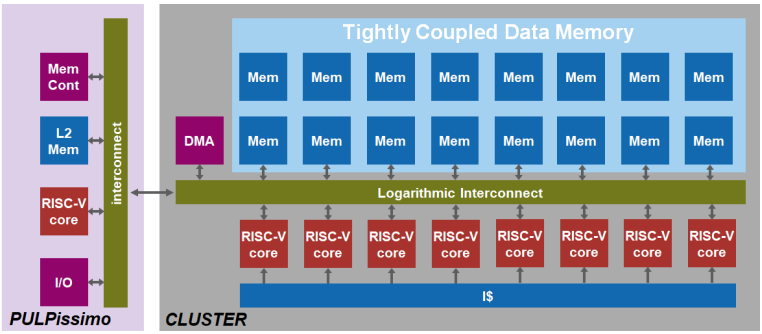


Figure 1.3: The PULP MCU with 8 cores, 16 shared TCDM-banks and a shared I$.

Chapter 2

RISC-V CPUs

In this Chapter, the Parallel Ultra Low Power Platform (PULP) Instruction Set Architecture (ISA) extensions and the pipeline architecture of two RISC-V cores (*Riscy* and *Zero-riscy*) are described. The two RISC-V cores are optimized for different applications and working domain.

The ISA extensions are specifically targeting data-intensive computations at near-threshold voltage operation in tightly-coupled memory systems of multiple cores. Several ideas have been borrowed from the Digital Signal Processing (DSP) domain to enhance the *Riscy* core, but still maintaining complete compatibility with the streamlined RISC-V ISA.

On one hand, *Riscy* is optimized for data-intensive applications, benefiting of the aforementioned ISA extensions, working in a cluster of multi-cores (4/8 cores), operating at near-threshold voltage as software-accelerator, thus used in a duty-cycle fashion. On the other side, *Zero-riscy* targets mixed arithmetic/control applications and control-oriented tasks, working as a microcontroller in an always-on domain.

In addition, in this Chapter, the verification level of the DSP-enhanced core is increased to provide a higher quality core.

The main contributions of this Chapter can be summarized as follows:

- An optimized execution stage supporting flexible fixed-point and saturated arithmetic operations as well as Single Instruction Multiple Data (SIMD) extensions, including dot-product and shuffle instructions on a performant core, meant to run in a parallel cluster.

- Two flavors of an area-optimized core to improve the energy efficiency on always-on domains with limited computation requirements. *Micro-riscy* is optimized to have a minimal area and power. *Zero-riscy* is optimized to have a small area and power, but higher computation performance than *Micro-riscy*. They consist of one single RTL description with parameters to tune the area resources. Furthermore, to target high energy efficiency and ultra-low power in battery-powered Internet-of-things (IoT) devices, the cores are evaluated in Near-Threshold (NT), where the transistors achieve their maximum energy efficiency [73].

- Creating an automatic high-code coverage verification stimuli for the *Riscy* core to widely explore the verification research space by combining the generation of the test-programs with their evaluations.

The three cores (two RTL descriptions with parameters) provide an open-source family of heterogeneous cores ready to be used in different contexts, all implementing the RISC-V ISA supporting compressed instructions.

We show that with the help of the ISA extensions and micro-architectural enhancements, signal processing kernels, such as filters, convolutions, etc. can be completed faster and more energy-efficient on *Riscy*.

We show that thanks to an area-aware design with small datapath, an optimized core for control tasks consumes less energy when executing *Coremark*. An even smaller core with extreme area optimizations is the most energy-efficient in pure control-oriented code.

We provide in-depth-comparative analysis of the three cores in terms of area, power, performance, and energy efficiency, with a wide

range of synthesis constraints and workloads. We analyze their energy consumption in an always-on context, where the cores are waiting for an event to start the computation and finally go back to sleep. We show that when the interval time between events is long enough, the leakage power contribution becomes crucial. Hence small cores overwhelm the fast cores in energy efficiency.

Finally, a simulation-based framework based on μGP[1] is developed to increase the verification level of the *Riscy* core (extended with a Floating Point Unit (FPU)) with automatic test program generation, reaching on average about 90% on a set of high-level code coverage metrics while unveiling ten different bugs still present in the processor description.

Given a large number of external users of such open-source cores, and as companies require continuous support, the cores graduated from an academic project to an industrial product. The small core, formally called *Zero-riscy*, has been moved to a non-for-profit organization called *LowRISC* [2] under the name of Ibex. The other core, formally called *Riscy*, has been moved to another non-for-profit organization called *OpenHW Group*[3] under the name of CV32E40P. Both the cores will be kept open-source and maintained by the two companies.

As the targeted domain of the proposed cores is close to the one of the Cortex-M cores, Table 2.1 shows an area comparison between the proposed RISC-V cores and their corresponding Cortex-M family implementations. Furthermore, the verification level of the *Riscy* extended with an FPU has been increased by generating pseudo-random test programs that try to optimize the code coverage of its HDL description. The simulation-based approach unveiled bugs present in the RTL description by combining the generation and testing of such programs.

[1] μGP is open-source and freely downloadable at http://ugp3.sourceforge.net/
[2] https://www.lowrisc.org/
[3] https://www.openhwgroup.org/

TABLE 2.1
COMPARISON BETWEEN THE PROPOSED RISC-V CORES AND ARM CORTEX-M FAMILY IMPLEMENTATIONS.

Core	*Area [KGE]*	*Core*	*Area [KGE]*
Riscy	40.7	Cortex-M4/Cortex-M3	53.0/37.9
Zero-riscy/Micro-riscy	18.9/11.6	Cortex-M0/M0+	13.3/12.5

Cortex-M0 area from synthesis in UMC 65nm with 32 cycles mult, 16 interrupts, no wake-up, and debug controllers. Cortex-M0+, M3, and M4 area estimated from publicly available information [134].

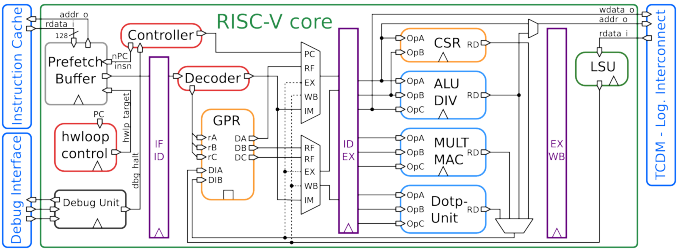


Figure 2.1: Simplified block diagram of the *Riscy* core architecture showing its four pipeline stages and all functional blocks.

2.1 RISC-V Microarchitectures

In this Section, the RISC-V architectures proposed in this work are described. We detail the extensions made to the RISC-V ISA and micro-architectural optimizations for increasing the efficiency of the *Riscy* core. The pipeline architecture, the ISA extensions, and the individual components of the core are discussed in Subsection 2.1.1. The area-optimized cores and their RISC-V ISA extensions implemented, as well as their pipeline architecture, are described in Subsection 2.1.2 and Subsection 2.1.3.

2.1.1 *Riscy*

Riscy is an open-source 32 bit RISC-V core, in-order with four pipeline stages [57].

The number of pipeline stages of the core is one of the key design decisions that determines the speed and the Instruction Per Cycle (IPC). A higher number of pipeline stages allows for higher operating frequencies but reduces the IPC due to a higher number of data and control hazards, which can be alleviated with branch predictions, prefetch-buffers, multiple-issues and out-of-order execution. However, such optimizations increase the area and power consumption. Thus such cores are usually used in high-performance systems rather than IoT devices. Ultra-low-power systems host instead simpler microprocessor with 1 to 5 pipeline stages, in-order, single-issue. Such cores usually achieve higher IPC than higher frequency, consume less power, and are smaller than high-performance Central Processing Units (CPUs).

The *Riscy* core presented in this Chapter is composed of 4 pipeline stages. Its architecture, shown in Figure 2.1 underlines the Instruction Fetch (IF), Instruction Decode (ID), Execution (EX), and Write Back (WB) stages, with a 3-read-2-write ports register-file.

Since the memory interface mainly determines the critical path, it is possible to extend ISAs with single-cycle enhanced arithmetic operations without incurring additional timing penalties.

All three cores implement the RVC RISC-V standard to support compressed-instructions (16 bit wide instructions). Therefore, the IF stage has been designed to handle compressed instruction decoding, cross-word instructions, and misaligned memory accesses.

Following, *General-Purpose ISA Extentions* implemented in the *Riscy* core are presented.

General-Purpose ISA Extentions

Load/Store extensions to the basic RISC-V architecture have been designed to support register+register address generation and post-increment operations; the latter used to update pointers automatically without explicit *add* operations. The 2-write register-file ports are used to implement automatic post-increment memory operations in a single-cycle, reserving a write port for the EX stage that performs the increment of the address, and the second port for the upcoming memory data from the WB stage. In addition, the *Riscy* core supports misaligned memory accesses, which frequently happen during vector operations such as convolutions. The core emits two word-aligned

memory operations and recombines the data internally, completely transparent to software.

Table 2.2
Extended Load/Store Instructions.

Instruction format	Description
`p.l{b,h,w} rD, {rB,I}(rA)`	Load a value from address (rA+{rB,I})[c]
`p.l{b,h,w} rD, {rB,I}(rA!)`	Load a value from address rA and increment rA by {rB,I} [c]
`p.s{b,h,w} rB, {rD,I}(rA)`	Store a value to address (rA+{rD,I}) [c]
`p.s{b,h,w} rB, {rD,I}(rA!)`	Store a value to address rA and increment rA by {rD,I} [c]

[c] b, h, w specific the data lenght of the operands: byte (8 bit), halfword (16 bit), word (32 bit).

Zero-overhead loops are a common feature in many DSP processors. They are used to overcome the overhead of instructions to handle counters and branches to benefit IPC and code-size. The *Riscy* can handle up to 2 nested loops. The ISA has been extended to set the beginning, end, and the number of iterations of the loop. Once the core reaches the end of the loop, no cycles are lost to jump back to the first instruction of the loop, making iterative-software more efficient.

Table 2.3
Hardware Loop Instructions.

Instruction format	Description
`lp.starti L, I`	Set the HW loop start address
`lp.endi L, I`	Set the HW loop end address
`lp.count L, rA`	Set the HW loop number of iterations
`lp.setup L, rA, I`	HW loop setup with registers
`lp.setupi L, I1, I2`	HW loop setup with immediate

Other instructions useful for general purpose applications implemented in the *Riscy* core include: minimum and maximum between two operands, sign-extensions, rotation, and absolute value.

In addition, the *Riscy* core implements bit manipulation instructions. Such instructions operate on sets of bits to *extract* (read set of bits), *insert* (write to a register a set of bits), *clear, set* (clear/set a set of bits), *count* (count number of bits that are 1), *find-first ,find last* (find index of first/last bit that is 1 in a register), and *count leading* (count leading bits in a register).

Packed-SIMD support

As the *Riscy* has been optimized for software acceleration, the processor's datapath has been modified to work on four bytes and two half-words in parallel. Such operations are also known as subword parallelism [135], packed-SIMD (pSIMD), or micro-SIMD [136] instructions.

In the IoT domain, pSIMD operations can be used on data sampled from sensors that are typically <32 bit data (e.g., 8 bit for low-power cameras, or 12 bit on Analog-to-Digital converters (ADCs), etc.).

Benefits of pSIMD operations are: reducing the data bandwidth, thanks to parallel load and store operations; and increasing performance, thanks to parallel data processing. However, area and power overhead have to be kept small to gain energy efficiency at application level, i.e., the execution time improvement has to be higher than the power increase due to the more complex core architecture.

pSIMD operations have been implemented in three addressing variations. The first variation uses two registers, the second uses an immediate value, and the third replicates the scalar value in a register as the second operand for the vectorial operation.

Vectorial operations like additions, subtractions, and comparisons have been implemented by splitting the datapath (adder and comparator) into four sub-operations. The Arithmetic and Logic Unit (ALU) decides whether to propagate the carry signals based on the vector mode. For example, the adder does not propagate the three intermediate carry-out signals to produce the four parallel addition; it does not propagate the second carry-out signal for the two half-word additions; whereas for the full 32 bit result, the carry-out chain is propagated throughout all the four adders. Others like shift-operations or dot-product instructions use parallel datapath instead without resource sharing of sub-modules.

The proposed multiplier can multiply two vectors and accumulate the result in a 32 bit value in one cycle. A vector can contain two 16 bit elements or four 8 bit elements. To perform signed and unsigned multiplications, the 8 bit/16 bit inputs are sign-extended. Therefore each element is a 17 bit or 9 bit signed word. A common problem with an N bit multiplier is that its output needs to be $2 \cdot N$ bit wide to cover the entire range. In some architectures, an additional register is used to store part of the multiplication result. Such dot-product (*dotp*)

operations can be implemented in hardware with four multipliers and a compression tree and allow to perform up to four multiplications and three additions in a single operation as follows:

$$d = a[0] \cdot b[0] + a[1] \cdot b[1] + a[2] \cdot b[2] + a[3] \cdot b[3],$$

where a[i], b[i] are the individual bytes of a register and d is the 32 bit accumulation result. The multiply-accumulate (MAC) equivalent is the Sum-of-Dot-Product (*sdotp*) operation, which can be implemented with an additional accumulation input at the compression tree. With a vectorized ALU, and *dotp*-operations, it is possible to significantly increase the computational throughput of a single core when operating on reduced bit-width data.

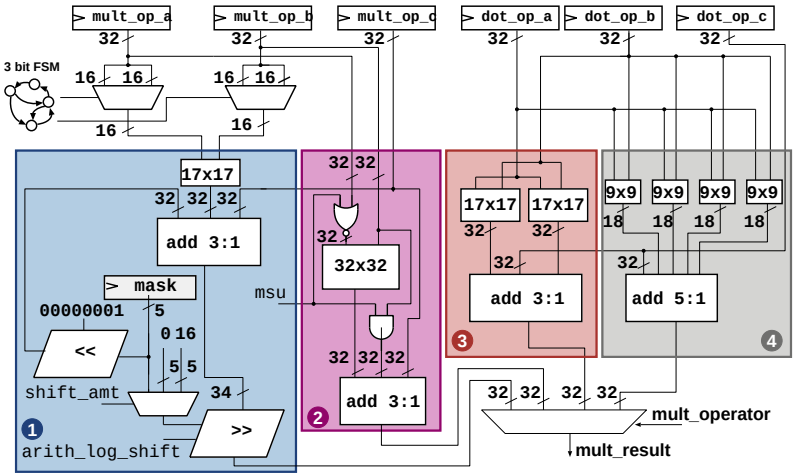


Figure 2.2: Simplified block diagram of the Multiplier in the RISC-V core implementation reflecting the behavioural implementation.

The dot-product unit's implementation has been designed such that its longest path is shorter or equal to the critical path of the

overall system, which in the PULP architecture is between the core and the memory subsystem. The dot-product unit has been designed to compute the result in a single-cycle. To achieve such a result without decreasing the maximum frequency, sub-modules units are not shared, resulting in a bigger area overhead. Additional pipeline registers would have resulted in additional stalls when computing back-to-back *dotp* operations. Figure 2.2 shows the 16 bit dotp-unit (region *3*) and 8 bit dotp-unit (region *4*) which have been implemented by one partial product compressor which sums up the accumulation register and all partial products coming from the partial product generators. The operations have been described as employing SystemVerilog operators to keep the design flexibility in terms of multipliers datapath selection, leaving the synthesizer the freedom to choose the best architecture that best meets the timing and area constraints. For example, a tight timing constrained design would exploit a carry-save format multiplier, whereas a less critical design could choose smaller multiplier architecture optimized for the area. The stand-alone multiplier has been analyzed in detail to minimize the area-delay product. Area can be saved when sharing compression-trees of the two dotp multipliers.

Additional packed-operand manipulation instructions are needed to prepare vector operands [137]. For example a *shuffle* instruction has been added to combine operands inside a vector based on a mask operand. Other instructions as *pack*, *insert*, and *extract* are also useful to manipulate vectors.

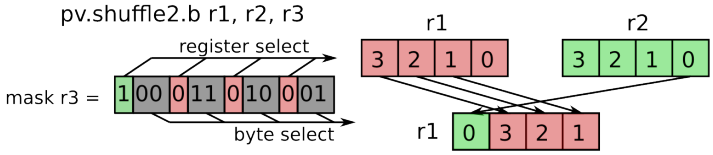


Figure 2.3: The Shuffle instruction allows to efficiently combine 8 bit, or 16 bit elements of two vectors in a single one. For each byte the mask encodes which byte (index) is used from which register (select).

Table 2.4
Vectorial Instructions.

Instruction format	Description
`pv.inst.{b,h} rD, rA, rB`	General vectorial instruction between two registers [a]
`pv.inst.{b,h} rD, rA, I`	General vectorial instr. between a register and an immediate [a]

[a] b, h, w specific the data lenght of the operands: byte (8 bit), halfword (16 bit), word (32 bit).

Fixed-Point support

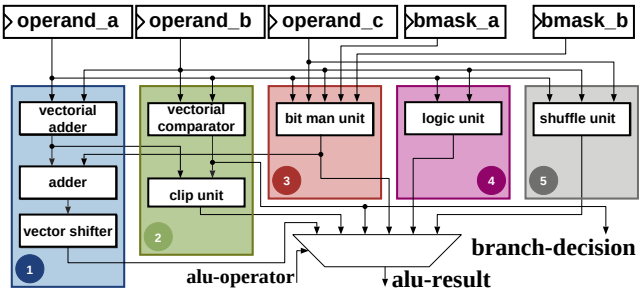


Figure 2.4: Simplified block diagram of the *Riscy* ALU.

In the IoT domain, many applications (speech processing,Convolutional Neural Networks (CNNs), Electroencephalography (EEG) processing) achieve enought performance with fixed-point accuracy [138]. Fixed-point arithmetic uses the integer datapath, as additions followed by a shift to adjust the fixed-point format (normalization), or comparisons with boundaries to saturate the numbers (saturation).

Fixed-point numbers are often given in the Q-Format where a Q$n.m$ number consists of n integer bits and m fractional bits. Some processors support a set of fixed-point numbers encoded in 8 bit, 16 bit or 32 bit, and provide dedicated instructions to handle operations with these numbers. For example, the ARM Cortex-M4 ISA provides

instructions as QADD or QSUB to add two numbers and then saturate the results to 8 bit, 16 bit or 32 bit.

Table 2.5
Addition of four Q1.11 fixed-point numbers with and w/o fixed-point instructions.

Without Add Norm Round	With Add Norm Round

`add r3, r4, r5`
`add r3, r3, r6`
`add r3, r3, r7`
`addi r3, r3, 2`
`srai r3, r3, 2`
	`add r3, r4, r5`
	`add r3, r3, r6`
	`p.addRN r3, r3, r7, 2`

The proposed ISA extensions implemented in *Riscy* support fixed-point arithmetic operations on any Q-format with the only limitation that $n + m < 32$. Normalization after additions, subtractions, or multiplications is implemented by shifting the result of such operations by a given amount that depends on the format. Rounding is implemented by adding to the result of such operations a number of the least precision before shifting. The two code examples in Table 2.5 show how the combined add-round-normalize (*p.addRN*) instruction can save both code-size (3 instead of 5 instructions) and execution time (2 cycles less). In this example, four numbers represented by Q1.11 are summed up. The result, if not normalized, will be a 14 bit long Q3.11 number. To keep the result in 12 bits, rounding can be achieved by adding 2 to the result and shift the number right by two places. The result can then be interpreted as a 12 bit number in Q3.9 format. The final *p.addRN* instruction achieves this rounding in a single step by first adding the two operands using the adder, then adding $2^{(I-1)}$ to the intermediate result, and finally shifting the result by I bits utilizing the shifter of the ALU. An additional 32 bit adder was added to the ALU to help with the rounding operation, as seen in the highlighted region 1 of Figure 2.4.

Fixed-point support for multiplications has also been extended. Differently from ALU functions, 32 bit multiplications have not been extended with shifting operations as this would result in a higher frequency penalty, not only for the shifter operation per se but also due to a larger multiplication result (64 bit) that should have been produced. We preferred instead to extend the ISA with fixed-point multiplication operations only to 16 bit operands. These new instructions accept

Table 2.6
Element-wise multiplication of n Q1.11 elements with round and normalization.

Without Mul Norm Round		With Mul Norm Round	
addi	r3, r0, *n*	addi	r3, r0, *n*
lp.setup	r0, r3, endL	lp.setup	r0, r3, endL
p.lh	r4, 0(r10!)	p.lh	r4, 0(r10!)
p.lh	r5, 0(r11!)	p.lh	r5, 0(r11!)
mul	r4, r4, r5	p.mulsRN	r4, r4, r5, 12
addi	r4, r4, 0x800	endL: sw	0(r12!), r4
srai	r4, r4, 12		
endL: sw	0(r12!), r4		

two 16 bit values (signed or unsigned) as input operands and calculate a 32 bit result. This result can also be accumulated (mac) with a third 32 bit operand, shifted to normalize the result and rounded. The two additional 32 bit values need to be added to the partial-products compressor, which does not increase the number of the compressor-tree levels and, therefore, does not add delay to the circuit [139]. The *p.mac* multiply-add instruction allows to perform 32 bit-operands multiplications and adding the result to an extra 32 bit value. The multiplier architecture is shown in Figure 2.2 where the fractional multiplier is shown in region *1*.

All these instructions are performed in one cycle, benefiting both the code-size and the number of cycles.

The code example given in Table 2.6 demonstrates the use of fixed-point multiplication support. In this example, two vectors of n Q1.11 elements are multiplied with each other (a typical operation in the frequency domain to perform convolution). The multiplication results in a Q2.22 number, and a subsequent rounding and normalization step will be needed to express the result with 12 bits as a Q1.11 number. It is important to note that, for such operations, performing the rounding operation before normalization reduces the error.

Similarly to the dot-product, the fixed-point multiplier design has been done in parallel, as adding single-cycle shift operation to the 32 bit multiplier increased the critical path. Thus, a parallel 16 bit multiplier has been implemented to handle the operations above. The simplified final multiplier architecture is shown in Figure 2.2 contains four modules: A 32x32 bit multiplier, a fractional multiplier, and two dot-product (dotp) multipliers.

Table 2.7
Addition of n elements with saturation.

Without clip support		With clip support	
addi	r15, r0, 0x800	addi	r3, r0, *n*
addi	r14, r0, 0x7FF	lp.setup	r0, r3, endL
addi	r3, r0, *n*	p.lh	r4, 0(r10!)
lp.setup	r0, r3, endL	p.lh	r5, 0(r11!)
p.lh	r4, 0(r10!)	add	r4, r4, r5
p.lh	r5, 0(r11!)	p.clip	r4, r4, 12
add	r4, r4, r5	endL: sw	0(r12!), r4
blt	r4, r15, lb		
blt	r14, r4, ub		
j	endL		
lb: mv	r4, r15		
j	endL		
ub: mv	r4, r14		
endL: sw	0(r12!), r4		

For fixed-point operations, a *clip* instruction has been implemented to check if a number is between two values and saturates the result to a minimum or maximum bound otherwise. No significant hardware has been added to the ALU to implement the clip instruction. In fact, the *greater than* comparison is done using the existing comparator, and the *less than* comparison is done in parallel by the adder. The *clip* instruction relies on the input data; therefore, handling overflows is left to the programmer. Unlike the ARM Cortex-M4 implementation, our implementation requires an additional *clip* instruction but has the added benefit of supporting any Q-number format and allows to round and normalize the value before saturating, which provides higher precision.

Table 2.7 shows an example of compiler-generated code where two arrays, each containing n Q1.11 signed elements, are added together, then the result is normalized between -1 and 1 represented in the same Q1.11 format. The example clearly illustrates the difference between the RISC-V ISA with and without clip support. Table 2.8 shows the added instructions for fixed-point support. Note that the code to the right is not only shorter; it also does not have control-flow instructions, thereby achieving better IPC.

Table 2.8
Fixed Point Instructions.

Instruction format		Description
`p.add[R]N`	`rD, rA, rB, I`	Addition with round and normalization by I bits [a]
`p.sub[R]N`	`rD, rA, rB, I`	Subtraction with round and normalization by I bits [a]
`p.mul[hh][R]N`	`rD, rA, rB, I`	Multiplication with round and normalization by I bits [a b]
`p.mac[hh][R]N`	`rD, rA, rB, I`	MAC with round and normalization by I bits [a b]
`p.clip`	`rD, rA, I`	Clip the value between -2^{I-1} and $2^{I-1}-1$

[a] If R is not specified, there is no round operation before shifting.
[b] If hh is specified, the operation takes the higher 16 bit of the operands.

Complex numbers support

Complex numbers are numbers that can be expressed as $(a, b) = a + ib$, where a is called real-part, and b imaginary part. As signal theory algorithms are often based on frequency-domain transformations built on complex domains, having hardware support for complex arithmetic increases performance when executing such algorithms. Leveraging the existing hardware infrastructure for the pSIMD support, complex instructions between 16 bit data can be added with negligible overhead. A complex number d is represented as two 16 bit packed data in a register using the higher part to represent the imaginary number $d[1]$, and the lower part for the real number $d[0]$. As the multiplication d between two complex numbers $a = (ra, ia)$ and $b = (rb, ib)$ is defined as:

$$d = (ra \cdot rb - ia \cdot ib, ra \cdot ib + ia \cdot rb),$$

two instructions that use the dotproduct have been added. The first instruction calculates the real part of the complex multiplication as:

$$d[0] = a[0] \cdot b[0] - a[1] \cdot b[1].$$

The second instruction calculates the imaginary part as:

$$d[1] = a[0] \cdot b[1] + a[1] \cdot b[0].$$

In addition to complex multiplications, subtraction with rotation by 90 degrees and complex and conjugate instructions have been added.

Furthermore, a bit-reverse instruction has been included to calculate the index in the first or last stage of fast Fourier transform (FFT) algorithm butterfly. This instruction takes as input the index X of an FFT made on 2^p points (up to 2^{31}) in radix 2^r ($r = 1$, 2 or 3), and it returns a bit reversed representation. For example, for an FFT of 8 points ($p = 3$) and radix-2 ($r = 1$) implementation, the index $x = 6 = 110b$ is translated by the bit reverse instruction into $bitrev(x) = 3 = 011b$. The area overhead of all these extensions is only 1% of the whole processor, consisting of 9 flip-flops and multiplexers. Figure 2.5 shows a simple example where two arrays of bytes are added in a third array. The example shows the baseline implementation with pure RISC-V only instructions and the performance achieved by the *Riscy* when running it. Then it shows the contribution of the instruction extensions on performance and code-size.

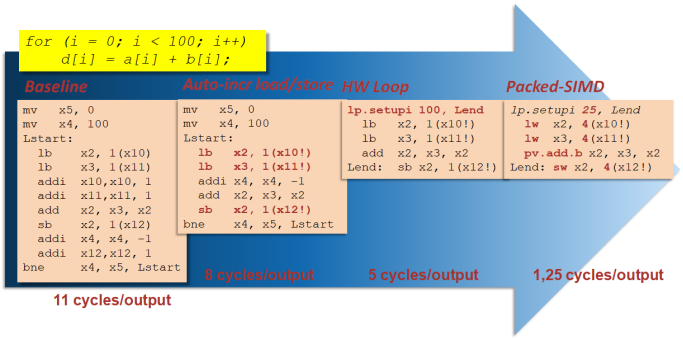


Figure 2.5: *Riscy* ISA extensions for performance. This example sums two arrays of bytes.

Power-saving Architecture

The proposed ISA-extensions are realized with separate execution units in the EX-stage, which have contributed to an increase in area (8.3 kGE ALU, 12.6 kGE multiplier). To keep the power consumption at a minimum, switching activity at unused parts of the ALU must be kept at a minimum. Therefore, all separate units: the ALU, the integer and fractional multipliers, and the dot-product unit all have separate

input operand registers that can be clock gated. The instruction decoder controls the input operands and can be held at constant values to eliminate the propagation of switching activity in idle units further. The switching activity reduction is achieved by additional flip-flops at the input of each unit (192 in total). These additional flip-flops allow reducing the switching activity, which decreases the power consumption of the core by 50%.

2.1.2 *Zero-riscy*

Zero-riscy is an area-optimized RISC-V core implementing the RVC32IM instruction set architecture; Figure 2.6 shows a simplified version of its micro-architecture. It has two pipeline stages that are the IF and Instruction Decode and Execute (IDE) stage. Its RTL description is heavily based on the *Riscy* core has it started as a fork of that project. Similarly to *Riscy*, the IF stage interacts with the instruction memory subsystem. It contains a prefetch-buffer that collects data from the instruction memory, and it handles compressed instructions.

The prefetch-buffer generates the instruction address and the program counter value (which can be different due to misalignment given by compressed instructions), and it contains a FIFO to store instructions also when the next stage is not ready to process them. The IDE stage decodes the instructions, reads the operands from the register file, prepares the operands for the ALU and the multiplier unit, and executes the instructions. The register file is a 2-read-1-write latch-based register file. The ALU contains the minimal hardware resources to implement the RVC32IM ISA: one 32 bit adder, one 32 bit shifter, and the logic unit. The 32 bit adder is used to compute additions, subtractions, and comparisons. It is shared with the data address generation unit, the branch engine, and the divider. Branch, multiplication, division, load, and store instructions are computed iteratively, stalling the next instruction until they have finished.

Figure 2.7 shows the simplified ALU architecture. The top multiplexer selects the addition operands, whereas the shifter and the logic unit always compute their function on operands coming from the decoder or register-file. Branch instructions, which need to compute both the comparison and the branch target, use the adder in two different cycles: the first cycle is used to compute if the branch is taken

or not. In case the branch is taken, in the next cycle, the adder is used to add to the program counter the offset to build the target address. Load and store instructions are executed in two cycles: the first cycle is used to calculate the address and the second cycle to receive the data from memory. The multiplier unit contains one Multiply And Accumulate (MAC) unit, which is able to sequentially multiply two 16 bit operands and accumulate the result in a 32 bit register. Divisions are implemented with minimum resources with the unsigned serial division algorithm using the adder in the ALU in all the steps. No forward-path and data-dependency logic are present in the core, which allows to save control-logic, multiplexers, and comparators. *Zero-riscy* implements a minimum set of control-status registers defined by the privileged RISC-V 1.9 spec, debug, and up to 32 interrupt requests.

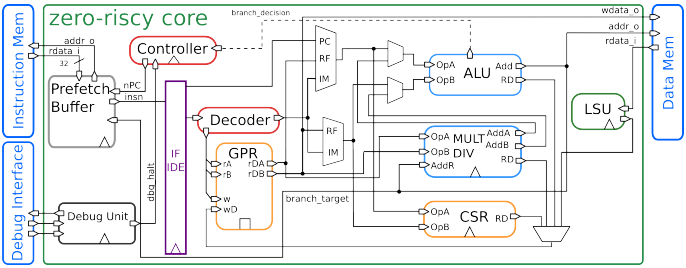


Figure 2.6: Simplified block diagram of *Zero-riscy*. The critical path is along the branch-address from the IF-IDE pipeline stage to the ALU, all the way along to the branch-target in the prefetch-buffer and finally to the instruction memory. It is about ∼280 FO4 NAND2 gate in UMC 65nm.

2.1.3 *Micro-riscy*

Micro-riscy is further optimized for area with respect to *Zero-riscy* by removing the RVM RISC-V extensions. *Micro-riscy* does not have any Hardware (HW) support for multiplications and divisions. To further reduce the area footprint, it implements the RVE RISC-V specification, which allows to use only 16 general-purpose registers. This core is

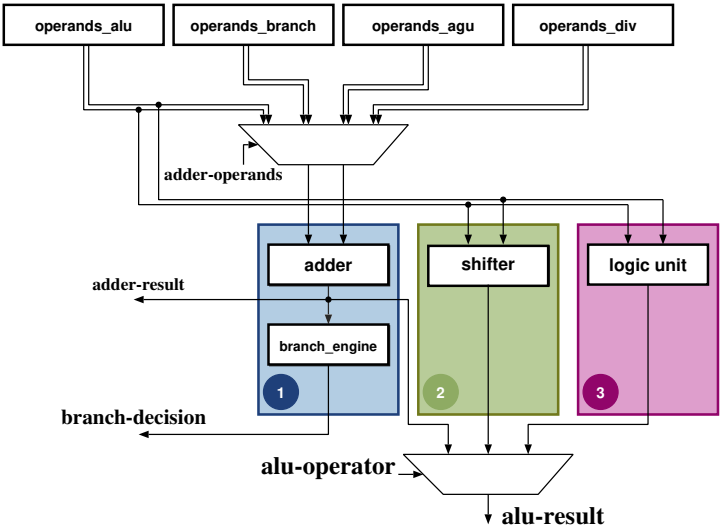


Figure 2.7: Simplified block diagram of the *Zero-riscy* ALU. The adder is shared between additions/subtractions, branches, address generation for load/store operations and divisions.

suitable for control-oriented code like Finite State Machines (FSMs), runtime functions, schedulers, etc.

2.2 Toolchain Support

The PULP-compiler used in this Chapter has been derived from the original GCC RISC-V version, which is itself derived from the MIPS GCC version. GCC-5.2 release and the latest Binutils release have been used. The Binutils have been modified to support the extended ISA as well as a few new relocation schemes.

The PULP-compiler automatically generates Hardware loops, post-increment load and store instructions, fixed-point supports instructions, bit manipulations, and packed-SIMD expect for specific *shuffle* instructions and dot-products, for which specific built-in functions need to be used.

An example of using *dotp* instructions is given below:

```
// define vector data type and dotp instruction
typedef short PixV __attribute__((vector_size(4)));
#define SumDotp16(a,b,c) __builtin_sdotsp2(a,b,c);

PixV VectA, VectB; // vectors of shorts
int S;
...
S = 0;
// each iteration is computing two mult and 2 accum
for (int k = 0; k < (SIZE>>1); k++) {
  S = SumDotp16(VectA[k], VectB[k], S);
}
C[i*N+j] = S;
...
```

2.3 Experimental results

In this Section, the area, performance, power, and energy consumption of the three cores at different operating points running different workloads are analyzed. Each core has been synthesized, placed, and routed targeting the UMC 65nm technology. How the energy consumption changes to different workloads and operating frequencies and voltages are also discussed in this Section.

2.3.1 Area

Figure 2.8 shows the area distribution of the three cores in terms of kgates-equivalent. The *Riscy* area is 40.7 *KGE*[4], whereas the *Zero-riscy* and *Micro-riscy* areas are 18.9 and 11.6 *KGE* respectively when all the cores are synthesized at relaxed timing-constraints. All the ISA extensions increased the *Riscy* core area by 14% due to the dot product unit, the extra register-file read and write port and all the other instructions that made the decoder and execution units more complex.

The smaller and less complex prefetch-buffer saves 2.7 *KGE* overall in both the area optimized cores. Thanks to the non-extended ISA

[4] Equivalent minimum-size NAND2 gate area. In UMC 65nm, one gate equivalent (GE) is 1.44 μm^2. The complex instruction support are not included.

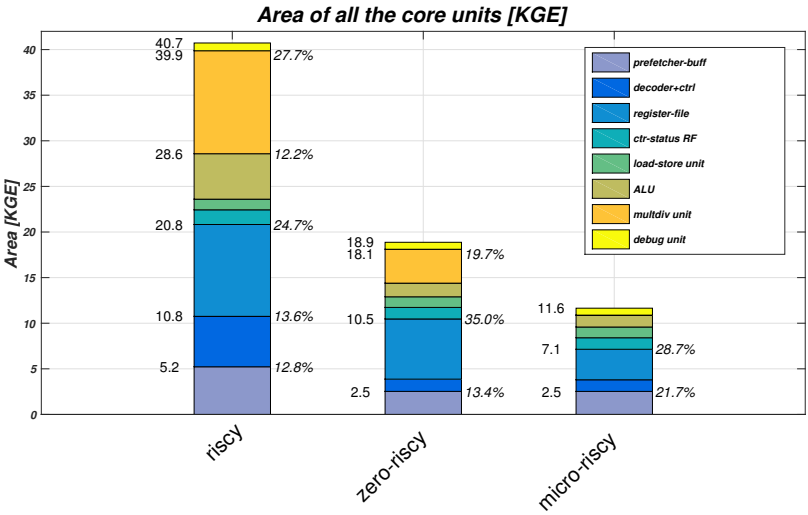


Figure 2.8: Area distribution of the different cores. Values are taken from the post-synthesis slow-netlist instances.

and less complex pipelines, the decoder, and controller are also much smaller ($\sim$7*KGE*). Also, the ALU has been optimized to use as few resources as possible, preferring time-multiplexing for instructions that need the same datapath concurrently. The *Zero-riscy* multiplier has also been optimized to use multi-cycle implementations of the product instructions, and it does not contain any dot-product unit nor fixed-point support. *Riscy* has a costly 3-read-2-write ports latch-based register-file (10 *KGE*), almost as much as the entire *Micro-riscy* core. By substituting it with a 2-read-1-write register-file, *Zero-riscy* saves 3.4 *KGE*; furthermore, *Micro-riscy* reduces its size to only 16 entries, down to only 3.3 *KGE* of footprint.

Finally, the simpler special-purpose registers reduces the area of the control-status register-file from 1.5 *KGE* to 1.2 *KGE*. The major contribution to the *Riscy* area footprint is the multiplier and division unit (27.7%), due to the complex dot-product unit and fixed-point multipliers. In *Zero-riscy*, the core area is dominated by the register-file (35%), as the multiplier unit implements only the RVM specifications

in a multi-cycle fashion. Even if the number of entries has been reduced to 16, in *Micro-riscy*, the top contribution to the total area is again the register-file (28.7%).

2.3.2 Workloads

We chose three micro-benchmarks selected from a large set as representative of three different classes of workloads.

The *2D-Convolution* is a standard kernel in image and signal processing applications. This kernel has been chosen to benchmark signal processing capabilities of the three cores. We chose to implement a 5x5 filter and consider a 32x32 input image. Both the filter and the input and output images are 16 bit fixed-point. Furthermore, before storing the result, the output is saturated between -1 and 1. In the case of *Riscy*, the code has been optimized to use pSIMD, dot-product, fixed-point, and vector manipulation instructions.

Figure 2.9 explains how a 5x5 2D-convolution can be computed with vector instructions. It can be seen that for each convolution step, 25 data values have to be multiplied with 25 constants and accumulated. If 8 b values are being used, registers can be used to hold vectors of four elements each. Once this calculation is completed, for the next step of the iteration, the five values of the first row will be discarded, and a new row of five values will be read. If these vectors are not aligned to word boundaries, an unaligned word must be loaded from memory, which can be supported either in hardware or software. A software implementation requires at least five instructions to load two words and combine the pixels in a vector. In addition, it blocks registers from being used for actual computations, which is why we support unaligned memory accesses directly in the load-store-unit by issuing two subsequent requests to the shared-memory. Hence, unaligned words can be loaded in only two cycles. We also implement the *shuffle* instruction that can combine sub-words from two registers in any combination. Figure 2.9b) shows how *move* and *shuffle* instructions are used to recombine the pixels in the right registers instead of loading all elements from memory. This allows to reduce register file pressure and the number of loads per iteration from 5 to 2.

The second micro-benchmark is Coremark, a system-independent benchmark from the Embedded Microprocessor Benchmark Consortium used to test embedded system cores [140]. It contains both integer arithmetic and control code operations. This kernel has been chosen to show arithmetic-control mixed processing capabilities of the cores.

Finally, the *Runtime* workload implements a simple set of embedded-system Runtime functions, as, e.g., a non-preemptive task-scheduler and a driver to interact with peripherals. The functions are taken from the PULP Runtime.

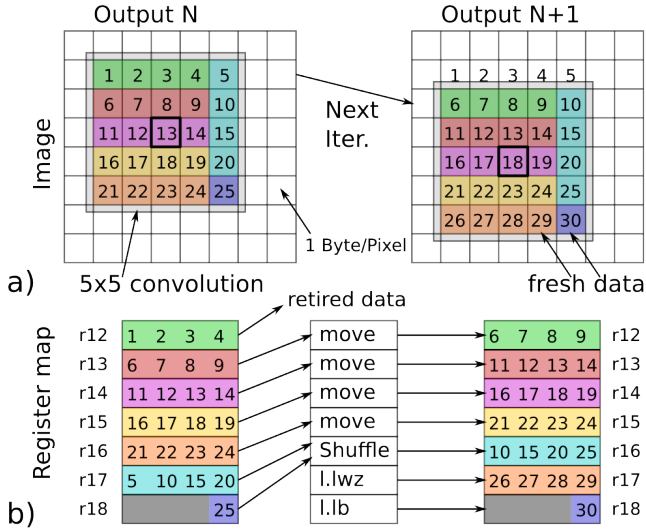


Figure 2.9: a) Example of a 5x5 convolution to compute output N and N+1 in the image domain and b) how the register content is efficiently updated using the shuffle instruction. One 5x5 convolution requires exactly 4 *move*, one *shuffle*, and 2 loads to prepare the operands and 1 *dotp*-, and 6 *sdotp*-operations to compute one output pixel.

2.3.3 Performance

Experiments have been conducted by integrating the three cores within three separate instances of the open-source PULPino micro-controller platform [141], an older and simpler version of PULPissimo. The micro-controller has one core, one data and one instruction memory, an event-unit for handling interrupts and standard peripherals as SPI, UART, I2C, GPIO, and timers. PULPino is a simplified version of PULPissimo, where there are only two private banks. In Figure 2.10, we show an overview of the PULPino architecture.

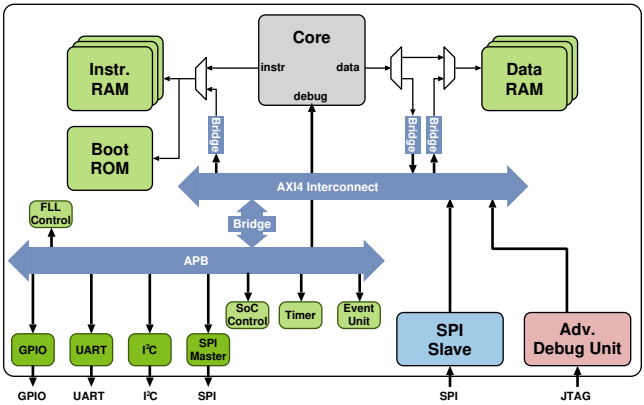


Figure 2.10: PULPino architecture overview. Software and RTL are open-source and available at www.pulp-platform.org

The three PULPino instances have been fully synthesized, placed, and routed at the 1.08V worst-case corner using Synopsys © Design Compiler 2015.06 for the synthesis and Cadence © Innovus 15.20 for the place and route phase. For each core, two different netlists have been generated to show the timing constraints' effects on power consumption, as discussed in the next Section 2.3.4. The first netlist has been constrained with a relaxed clock period of 10 ns, whereas the second netlist with a tighter clock period of 3 ns. We refer to the "relaxed" design as slow-netlist and to the "tighter" design as fast-netlist. In typical conditions of the targeted technology at 1.2 V, the slow-netlist can reach 185 MHz and the fast-netlist 560 MHz, whereas at 0.8 V, the

slow-netlist can reach 55 MHz and the fast-netlist 160 MHz. Table 2.9 shows the micro-benchmark [5] execution cycles, the IPC and code-size. Execution cycles and code-size have been normalized with respect to *Riscy*. In this experiment all the cores run at the same frequency and can reach almost the same maximum frequency. As expected, *Riscy* is the fastest core in all the kernels thanks to its enhanced ISA and more complex pipeline, whereas *Micro-riscy* is the slowest. When running DSP applications, *Riscy* is 6.1x faster than *Zero-riscy* and 53.4x faster than *Micro-riscy*.

Such a difference in performance is mostly given by the ISA extensions rather than the core architecture. In fact, when data-analytic applications are compared running on the same core with and without ISA extensions, the *Riscy* is on average 3.5x faster and 3.2x more energy efficient as presented in [57]. Whereas on *Coremark*, it is 8.5% faster.

The code-size for the DSP kernel changes less than 10%, whereas for *Coremark*, the code-size increases by 10% and 30% for *Zero-riscy* and *Micro-riscy* respectively. As expected, in the control-code micro-benchmark case, the code-size does not change significantly.

Zero-riscy is 8.8x faster than *Micro-riscy*. The two cores have the same pipeline architecture, but *Zero-riscy* implements the HW multiplication instruction, which is heavily used in the *2D-Convolution* kernel. *Riscy* provides the best performance also on arithmetic-control mixed code (3.19 CoreMark/MHz), but the benefits with respect to *Zero-riscy* (2.44 CoreMark/MHz) are much less significant than the signal processing algorithms. *Coremark* does not contain much DSP code. Therefore the difference in performance does not come from the DSP extensions of *Riscy*, but mainly from its deeper pipeline that enables single-cycle data memory access, single-cycle multiplication-accumulation, and general-purpose instruction extensions as, e.g., bit-manipulation. *Zero-riscy* targets arithmetic-control mixed code: its area is small, but it still has the hardware resources necessary to support multiplications and divisions, which are required for signal processing algorithms. Similarly to what happens in the DSP kernel, *Micro-riscy* is the slowest core (0.91 CoreMark/MHz), primarily due to the lack

[5] The three benchmark kernels have been compiled with GCC5.2 whose back-end has been modified to instantiate the extended instructions for *Riscy* when appropriate.

of multiplication instructions, which are emulated in software. The reduced number of registers is only associated with a 3% performance drop. In the Runtime kernel, the three cores show negligible differences in performance: the code implementing these routines do not benefit from multiplications nor DSP extensions.

In terms of performance and targeted domain, the *Zero-* and *Micro-riscy* are comparable with the ARM Cortex-M0/M0+ (2.33/2.46 Coremark/MHz), whereas *Riscy* is comparable with Cortex-M4 (3.40 Coremark/MHz [57]). Whereas in terms of code-size, the proposed ISA extensions still have 7.3% inflation over the ARM-Thumb2 when compared on standard benchmarks of the IoT, mainly due to push/pop instructions, conditional branches, etc. [65].

The IPC of *Riscy* is the highest for all the micro-benchmarks. As the PULPino data and instruction memories have one cycle access, the IPC is 0.8 due to misaligned memory accesses (that require two cycles), data-hazard after load operations, and branches/jumps. Furthermore, in the control-oriented kernel, the control-status write operations require the core to flush the pipeline. For *Zero-riscy*, the IPC in the DSP micro-benchmark is only 0.5 as the most common instructions are multi-cycle operations as load, store and multiplications. In addition, the multi-cycle operations penalize to 0.7 the IPC also in the other kernels. Finally, *Micro-riscy* has IPC 0.7 in all the kernels.

Table 2.9
Number of cycles, IPC and code-size for each kernel.

KERNEL	*Riscy*			*Zero-riscy*			*Micro-riscy*		
	Cycles	IPC	Cod. Size	Cycles	IPC	Cod. Size	Cycles	IPC	Cod. Size
2D-Convolution	1.0 (43.1K)	0.82	1.0 (1080B)	6.1	0.52	1.0	53.4	0.66	1.0
Coremark	1.0 (313.5K)	0.79	1.0 (15.3KB)	1.3	0.67	1.1	3.5	0.67	1.3
Runtime	1.0 (37)	0.76	1.0 (232B)	1.0	0.68	1.1	1.0 (36)	0.67	1.1

2.3.4 Power estimation

The switching activity for each one of the versions of the platform described in Section 2.3.3 has been extracted employing simulation on post-layout netlists. The switching activity has been then analyzed using Synopsys © PrimeTime 2012.12 at 1.2 and 0.8 V in typical condition.Table 2.10 shows the dynamic power density and the leakage power estimated.

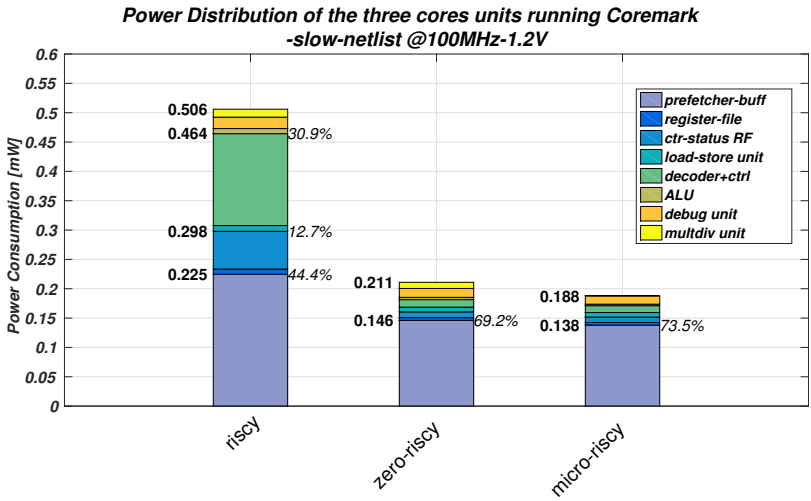


Figure 2.11: Power distribution of the different cores units measured from post-layout simulations at 1.2V, 100MHz.

The fast-netlists have higher leakage power and dynamic power density, since it has been synthesized, placed, and routed to target higher frequencies. On average, the leakage power of the fast-netlists is 14x the leakage power of the slow-netlists, while the dynamic power density increases by 1.2x at 1.2 V. At 0.8 V, the leakage power of the fast-netlists increases by 16x, whereas the dynamic power density by 1.3x.

Table 2.11 shows the total power consumption of all the four instances at the maximum frequency achievable at 1.2 V and 0.8 V and the ratio between the power consumption at the two voltages. As is visible in the plot, the power at 0.8 V is up to 12.2x smaller than in 1.2 V, whereas the speed decreases only up to ∼3.5.

Figure 2.11 shows how power is spent in the various components of each core. The main contribution to the total power consumption is provided by the prefetch-buffer, which interacts with the instruction memory every cycle, produces the instruction fetch address, and the program counter value and stores the instructions in a FIFO. In the *Riscy* core, it also handles hardware-loop instruction requests. As the

Table 2.10
Core power estimations.

Core	PDynD- PLkg @1.2V	PDynD - PLkg @0.8V
Netlist with relaxed timing constraints		
Riscy	5.07 - 1.91	1.40 - 0.22
Zero-riscy	2.08 - 0.73	0.59 - 0.07
Micro-riscy	1.88 - 0.45	0.52 - 0.04
Netlist with tight timing constraints		
Riscy	6.70 - 24.90	2.10 - 2.96
Zero-riscy	2.30 - 11.0	0.68 - 1.24
Micro-riscy	2.03 - 6.25	0.60 - 0.70

Dynamic power density is reported in µW/MHz and leakage power in µW.

Table 2.11
Core total power estimations at maximum speed.

Core	Ptot @1.2V	Ptot @0.8V	Power Ratio
Netlist with relaxed timing constraints			
	@185 MHz	*@55 MHz*	
Riscy	940	77	12.2
Zero-riscy	386	33	11.7
Micro-riscy	348	29	12.0
Netlist with tight timing constraints			
	@560 MHz	*@160 MHz*	
Riscy	3777	339	11.1
Zero-riscy	1299	110	11.8
Micro-riscy	1143	97	11.8

Power is reported in µW.

decoder and controller of the two smaller cores have been simplified, their power consumption has also been reduced significantly.

2.3.5 Energy calculation

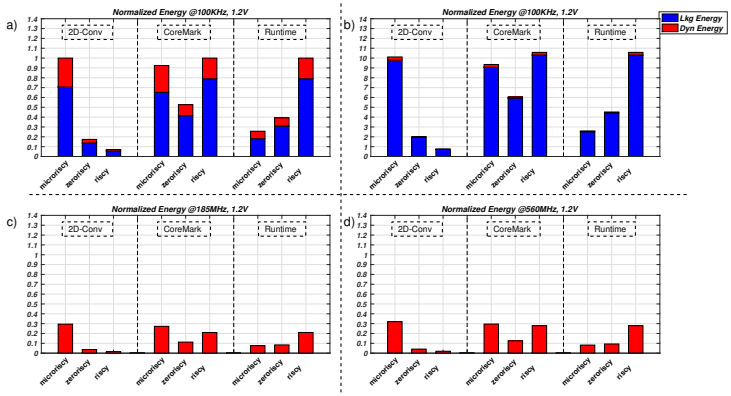


Figure 2.12: Energy consumed by the three cores while running different workloads. In blue, the leakage energy, in red, the dynamic. In the left column, the slow-netlist results are shown (a and c), whereas the fast-netlist is shown in the right column (b and d).

In this Section, the energy is calculated for each combination of core and kernel at different operating frequencies and voltages. In Figure 2.12, the normalized energy consumption for each core synthesized at the two different clock targets at 1.2 V is shown. Figure 2.12a and Figure 2.12c show the energy consumption of all the slow-netlist instances at two different frequencies, whereas in the right side, Figure 2.12b and Figure 2.12d show the energy consumption of all the fast-netlist instances. The energy consumption of each kernel has been normalized with respect to the worst-case using the slow-netlist at 100 kHz. For example, the *2D-Convolution* has been normalized with respect to the energy consumed by *Micro-riscy* as it is the highest. Figures 2.12a and 2.12b show the effect of the tighter timing-constraints at a very slow frequency; this underlines the effect of leakage.

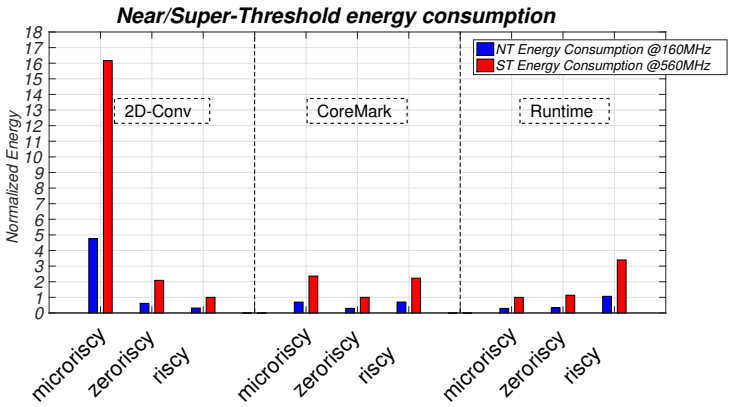


Figure 2.13: Energy consumed by the threecores while running different workloads at two different voltages. In blue the energy at 0.8 V, in red at 1.2 V.

As reported in Table 2.10, the fast-netlist has 1.2x the dynamic power density and 14x the leakage power of the slow-netlist. At the same operating frequency, the execution time of the two netlists is identical, whereas the total energy consumption of the fast-netlist increases by ∼10.7x. 97% of this energy consumption is due to leakage in the fast-netlist, whereas it is 76% in the slow-netlist. In Figure 2.12c and Figure 2.12d, the slow and the fast netlists are compared at their maximum frequency in typical conditions (185 MHz and 560 MHz respectively). The energy consumption of the fast-netlist is only 1.2x the energy of the slow-netlist due to the combination of the reduced execution time (∼3x) and the higher power consumption (∼3.6x). Leakage energy is negligible in both the netlists.

For DSP applications, the core equipped with DSP instruction extensions consumes the minimum energy with respect to the other two cores in all the operating conditions. The ratio between the execution time of *Riscy* and *Zero-riscy* (6.1 at the same frequency) is smaller than the ratio of powers. For example, in the fast-netlist at 560 MHz, the *Riscy* total power consumption is 2.9x the power of *Zero-riscy*, whereas in the slow-netlist at 185 MHz it is 2.4x. *Zero-riscy* and *Micro-riscy* consume on average 2.4x and 15.9x the energy of *Riscy*,

respectively. For arithmetic-control mixed applications as *Coremark*, *Zero-riscy* is the one which consumes the least energy. The execution time of *Zero-riscy* is 1.3x longer than *Riscy* but it consumes much less power. On the other hand, it is faster than *Micro-riscy* (2.7x) but it consumes more power. *Riscy* and *Micro-riscy* consume on average 1.9x and 2x the energy of *Zero-riscy* respectively. Finally, *Micro-riscy* is the core that consumes the least energy for control-code, as any benefit in performance does not compensate for the extra power of the other cores. Hence, *Zero-riscy* and *Riscy* consume on average 1.4x and 3.5x the energy of *Micro-riscy* respectively. The same experiment has been repeated at 0.8 V using only the fast-netlist, whose maximum frequency is 160 MHz. At 0.8 V, the cores consume, on average 0.3x the energy consumed in the super-threshold region. Figure 2.13 shows the energy consumption of the cores for each kernel in the super-threshold region 1.2 V, 560 MHz and at near-threshold 0.8 V 160 MHz. For each kernel, the energy consumption has been normalized by the most energy-efficient core for that application. It is possible to notice that *Riscy* is still the most energy-efficient core for DSP applications, *Zero-riscy* for arithmetic-control mixed applications and finally *Micro-riscy* for control-oriented code at NT.

2.3.6 Always-On activity

To evaluate the effect of duty-cycled activities in always-on conditions, the final experiment takes into account the time frame where the core is active and when the core is in idle-mode waiting for an event. In many cases, this happens in an always-on domain where the core cannot be completely switched-off with power gating. Thus they always consume leakage power. The longer the time between two events, the bigger the leakage contribution to the total amount of energy consumed.

Figure 2.14 shows the energy consumption of the three slow-netlist instances at 55 MHz when executing the *2D-Convolution* at 0.8 V, 55 MHz in the y-axis and the time between two events that trigger the computation in x-axis. The three curves show the energy consumption starting from the end of the execution of the benchmark. The energy consumption is normalized with respect to the energy consumed by *Riscy*. The inter-event time is normalized with respect to the execution time of *Riscy* (43.1 Kcycles/55 MHz = 784 µs. As previously discussed,

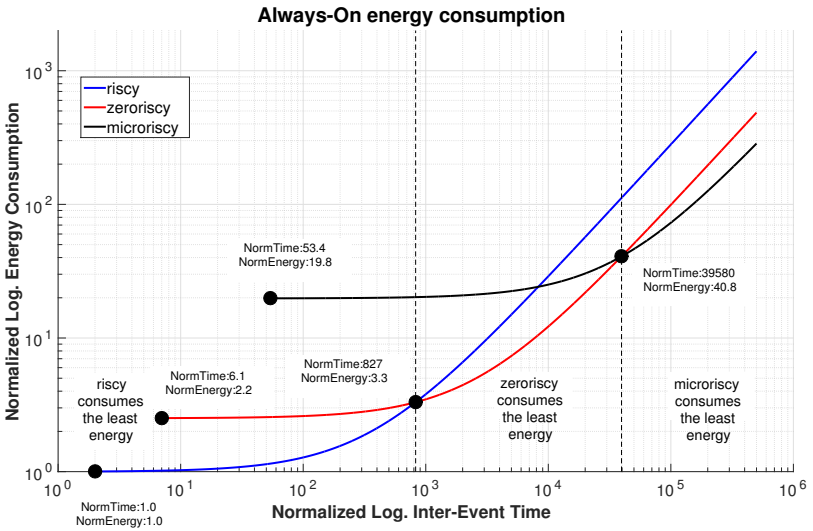


Figure 2.14: Always-On normalized energy consumption at 55 MHz 0.8 V: the core is active for a portion of time and sleeps for the rest of the time

when only the active period is taken into account, *Riscy* is the fastest and the most efficient core for signal-processing applications. However, when the idle period starts being longer, the leakage contribution overwhelms the active energy, thus smaller cores are less penalized. In the second region (from 649 ms to 31 s) *Zero-riscy* consumes less energy than the others, hence it is the most energy-efficient core. Such inter-event time is, for instance, in the EEG processing range, where the processing tasks wait for 1 or 2 s before they can start processing the acquired signals [59,104]. *Micro-riscy* becomes the most energy-efficient core when idle periods become extremely long. This analysis shows that leakage-optimized cores are well-suited for sporadic data-intensive applications if the throughput requirement is sufficiently low.

2.4 *Riscy* Verification Framework

In this Section, a new strategy to automatically generate a verification test-set that optimize code coverage is described.

We developed a simulation-based framework that relies on an evolutionary test program generator, the Device Under Verification (DUV), a reference model to evaluate the correctness of the DUV, and an independent evaluator that promotes programs that cover most of the DUV HDL description.

The framework developed in our case study is depicted in Figure 2.15. The setup includes a generation step (*stimuli generation*) combined with subsequent checking (*response checking*).

On the left part of the figure, the evolutionary optimizer (called μGP) [69] creates the verification program. μGP receives the description of the processor assembly syntax through the so-called *Instruction Library*, which describes which kind of instructions can be generated. The *Instruction Library* differs from ISA to ISA, and this Chapter, the developed library implements the RISC-V RV32IMFC extensions, plus the PULP extensions discussed above in this Chapter. In addition, μGP is configured to sets the number of test programs created in the population during the evolution, the number of instructions included on every test program, the maximum number of test programs to simulate, etc. Then, the evolutionary process starts by creating a set of random programs, so-called individuals, that are evaluated externally by a fitness function. At every evolutionary step or *generation*, the best individuals are improved using genetic operators such as *crossover* and *mutation*, while the worst ones are discarded.

To support complex sequence of instructions, the *Instruction Library* has been extended to describe finite loops, illegal instructions, and special environmental parameters to throw exceptions. Also, divisions by zero, infinite, and not-a-number operations are described to cover special cases in the microprocessor.

In our experiments, the verification programs are evaluated, resorting to a checking scheme reported in the right part of Figure 2.15.

In particular, the framework setup followed these steps:

1. The Instruction Library was created according to the target ISA.

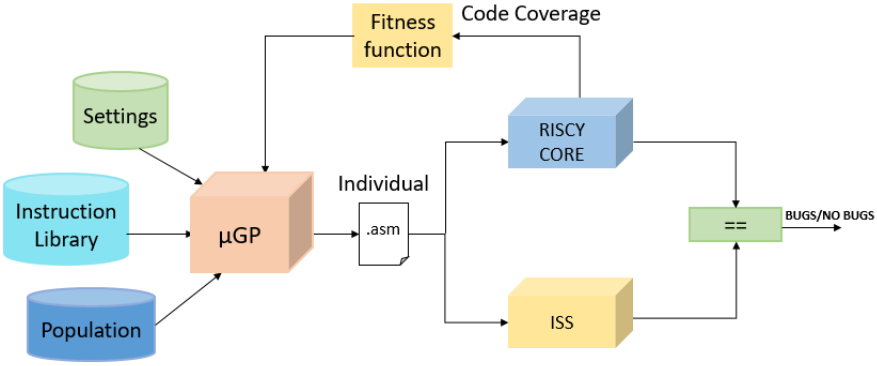


Figure 2.15: Verification framework

2. The main μGP settings were defined.
3. A script was produced to automate the flow. Its purpose was to pick up the individuals produced by μGP generation by generation and provide it to the simulator and the Instruction-Set Simulator (ISS). Then, to create the fitness function, the code coverage percentages were collected and fed to the evolutionary algorithm again.

The fitness of every verification program is evaluated by measuring its capacity to maximize the code coverage metrics on the processor model [142] [143]. There are different code coverage metrics, the one used in this Chapter are: *Statement Coverage*, *Branch Coverage*, *Condition Coverage*, *Expression Coverage*, *FSM State Coverage*, and *FSM Transition Coverage*.

The fitness function is computed as following: for each metric in use, the *arithmetic average*, the *variance* and the *sum* is computed accounting for all the core hierarchical modules single metrics. As equation 2.1 highlights, this results in a vector of 18 elements, which is used by μGP to drive the evolution, with the first element of the fitness function vector being the one with the highest priority.

The order of elements has been decided empirically, with the first 6 elements being the average (a), the second 6 elements the variance

(v) and last 6 elements the sum (s) of the *Statement Coverage* (1), *Branch Coverage* (2), *Condition Coverage* (3), *Expression Coverage* (4), *FSM State Coverage* (5), and *FSM Transition Coverage* (6).

$$fitness = \{a_1, a_2, a_3, a_4, a_5, a_6, v_1, v_2, v_3, v_4, v_5, v_6, s_1, s_2, s_3, s_4, s_5, s_6\}. \tag{2.1}$$

Where for instance:

$$a_1 = \frac{\text{Sum of Statement Coverage of all units}}{\text{Number of units}}$$
$$a_2 = \frac{\text{Sum of Branch Coverage of all units}}{\text{Number of units}} \tag{2.2}$$

To stimulate conditions such as stalls on the core memory interfaces, as well as interrupt requests, a *perturbation* module able to randomly simulate such external requests has been embedded in our framework. Such events cannot be triggered using only instructions; therefore, external devices are involved in stimulating those conditions [144]. The *perturbation* module contains memory-mapped registers to configure a stall and interrupt mode (random number of stalls/interrupts or fixed). The initialization of the perturbation module is set random before executing the generated individual. However, one can consider future works to program the perturbation registers as part of the evolutionary process.

Together with the fitness value that is used to guide the evolutionary process, any program is also used to compare the current processor model outcome (*Riscy+FPU* in Figure 2.15) against a high-level and reliable model (the ISS in Figure 2.15).

The ISS functional model is an accurate model of the *Riscy* ISA described in C. For every instruction, the HDL simulator pushes the instruction word to the ISS via a DPI-C wrapper. At the end of the execution, it compares the result computed by the HDL description with the one computed by the ISS.

In this way, it is possible to find differences in execution between the compared models during the evolutionary generation phase rather than only the optimized final individuals. Once a difference is found, a report is created, allowing the verification engineer to evaluate the bug of the DUV.

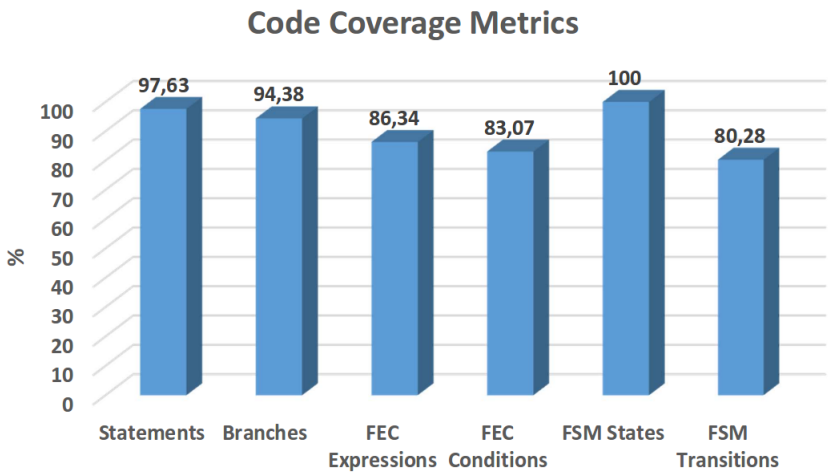


Figure 2.16: Code Coverage Results

The previously described framework was implemented, and at the end of the experiments, the verification test set obtains a high code coverage results reported in Figure 2.16.

In the performed experiments, the code coverage metrics have been extracted resorting to Modelsim® HDL Simulation and used as variables of the fitness function to evaluate the program's fitness. However, any free logic simulator tool that supports code coverage estimation could be used (e.g., Verilator).

The *Instruction Library* contains hundreds of rules to describe the aforementioned ISA extensions and special cases, and it is split into three sub-libraries to allow a better exploration of the processor description and maximizing the final code coverage at the same time. The first library generates test programs that only contain RV32IMC instructions (no floats) and PULP extensions plus constraints to generate special cases as illegal instructions and functions to enter sleep mode.

To stress more complex units and corner cases, a second library has been designed to focus on FPU instructions without polluting the individuals with the generation of integer instructions. This approach

Table 2.12
Code Coverage Results.

Library	Code Coverage
RV32IMFC Random	52%
Single General-Purpose	64%
Proposed Optimized Splitted	85%
Proposed Optimized Splitted + Perturbation	90%

Code coverage results for different individuals.

allows to further stress the FPU with high-density float instruction sequences. This library also contains special cases as RV32F illegal instructions and corner cases (NaN, infinite, division by zero). Finally, the third library has been used to generate two different experiments: one to optimize the coverage of the float multiply-and-accumulate unit and the float division-and-square root unit isolated from the rest of the DUV; and one still targeting the FPU but with focus on its conditional part. This is achieved by giving more relevance to the *Condition Coverage* metrics in the aforementioned fitness function instead of the *Statement Coverage*.

Splitting the evolutionary phases is useful to specialize the verification functions to face those parts of the DUV code challenging to cover. Indeed, a single general-purpose library to generate test programs based on the RV32IMFC plus extensions (but without special conditions like explicit NaN operands or illegal instructions) was able to cover only an average of 64% of the HDL code. One of the main reasons was that the generated program did not contain enough RV32F instructions keeping the FPU coverage low. A pure random program generated by the first generation of individuals from the same library had instead only 52%.

This modular experiment produced a final test program generation that results in a final code coverage of 85%. Finally, by adding random stalls on both data and instruction memory and random interrupt requests, the final code coverage increased to more than 90%.

Table 2.12 summarizes the code coverage discussed above.

The remaining 10% was related, for example, to FSM transitions, which never happened, the special case of NaN, infinite numbers, and their combinations. It is important to note that the main drawback

when dealing with evolutionary-based tools is the simulation time. In fact, to generate one of the best individuals included in the final test set, it takes about 8 hours of simulation.

The fitness function has been designed trying to boost all the coverages metrics to grow uniformly, and if two individuals have the same coverage results, the smaller code-size program is preferred.

The final test set uncovered ten design bugs during the verification process. These errors belong to different types; for example, the computation performed by the instruction *p.clip* and *p.extract* were discovered to be incorrect in a particular case. For instance, due to an intermediated overflow, the *p.clip* instruction failed to compute the correct value with specific cases, whereas the logic shift instead of the arithmetic one was mistakenly used for the *p.extract* instruction. Another example involves bugs related to FPU square root operations with rounding modes or with NaN operations in the RISC-V $fclass$ instruction.

The proposed method performs better than pure random and manual approaches, as well as using a single test program that covers the highest coverage as proposed in [145].

In fact, having the generation and verification phases split means using only the best individual to test the core's correctness, whereas combining the two phases means testing all the programs generated during the evolutionary steps. This allows to explore a more extensive space of programs and increasing the probability of success.

For example, the bug related to *p.clip* instruction previously mentioned was not part of the highest coverage program found by the optimizer. Still, it has been generated among the individuals of a prior generation to be then discarded during the evolutionary process. By combining generation and verification, every individual is not only used to calculate the code coverage, but it has the possibility of being executed and compared against the reference model (the ISS) to find design bugs. This approach shows that it is essential to maximize the code coverage and leverage less important individuals during the generation phase.

Chapter 3

FDSOI MPU

In this Chapter, a single-core instance of the Parallel Ultra Low Power Platform (PULP) architecture is implemented in Globalfoundries GF22FDX (GF22) 10 Metal technology. The chip, called *Quentin*, it is based on the PULPissimo platform. It hosts the *Riscy* core presented in Chapter 2 extended with an Floating Point Unit (FPU), a Binary Neural-Network (BNN) accelerator presented in [83], and a wide set of peripherals. In this Chapter, we discuss the size, performance, power, and energy results of the chip's implementation. In particular, performance, power, and energy are discussed in a wide range of supply voltage, exploiting FBB to achieve higher performance. The system has a heterogeneous memory subsystem to trade memory capacity with performance and energy efficiency. Such a subsystem is also exploited to trade accuracy and energy efficiency when running error-resilient applications on the BNN accelerator.

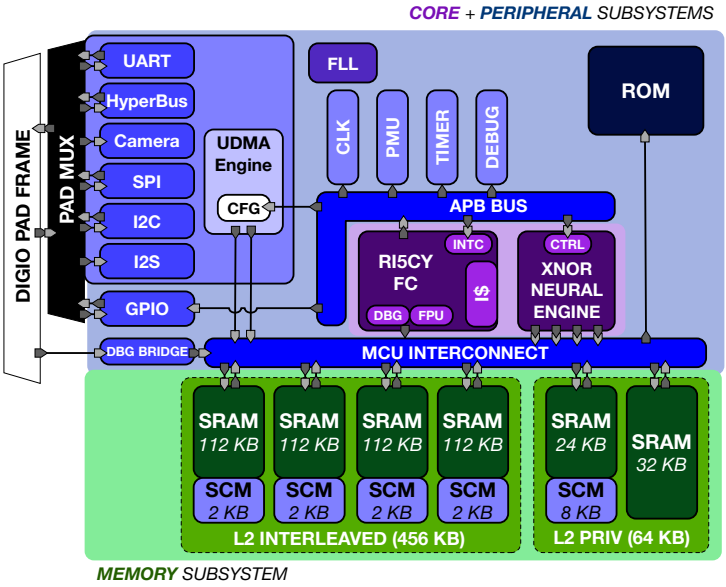


Figure 3.1: Quentin System-On-Chip (SOC) Architecture.

3.1 Quentin Architecture

3.1.1 Memory Subsystem

The SOC includes 520 kB of L2 memory and a Read-only-memory (ROM) with the boot-code. The L2 memory layout of Quentin is organized as four 114 kB word-level interleaved banks to minimize conflicts during parallel accesses of different masters (e.g., core and accelerator), plus two banks of 32 kB that can be used privately by the core (e.g., program, stack, private data) without incurring in banking conflicts with the other masters. The memories are slaves of the system bus, which is based on a single-cycle latency logarithmic interconnect [146] (Micro Controller Unit (MCU) interconnect bus in Figure 3.1). In case of two or more masters request to access the same slave, a round-robin arbiter selects the one that first communicates with

the slave. Both memory regions are implemented as a heterogeneous memory architecture composed of a mix of Static Random Access Memory (SRAM) and Standard Cell based Memorys (SCMs) [147].

In particular, each of the interleaved banks has 2 kB of the 114 implemented as SCM, and one of the private banks has 8 kB of SCM as shown in Figure 3.1, for a total of 504 kB of SRAM and 16 kB of SCM.

The SCM portion of the private bank is implemented as a 3-read 2-write ports register file: 2 of the three read-ports and 1 of the two write-ports are dedicated to the data and instructions interfaces of the *Riscy* core and one read- and one write-port are used by the interconnect arbiter for any other master node of the system. From a performance viewpoint, this memory organization enables transparent sharing of the L2 memory, increasing by 4x the system memory bandwidth with respect to a traditional single-port memory architecture without the use of high area overhead multiple ports memories.

The MCU interconnect also has one master and one slave port towards a cluster with an application class processor based on the CV64A core [51]. Such a cluster, called *Kerbin*, has an independent voltage and frequency domain, and it is not part of this Thesis.

3.1.2 I/O subsystem

The SOC includes a full set of peripherals: Quad SPI supporting up to two external devices, I2C, 2 I2S, a parallel camera interface, UART, GPIOs, JTAG, and a DDR HyperRam interface to extend the size of the on-chip memory. An I/O DMA (μDMA [133]) manages data transfers through peripherals to minimize the workload of the processor. To improve the efficiency of I/O communications, the μDMA has a dedicated connection to all the peripherals through 2 dedicated 32 bit ports on the L2 memory interconnect, granting an aggregated bandwidth sufficient to satisfy the requirements of all the peripherals (up to 1.6 Gbit/s) with a frequency of 57 MHz. Debug of the Quentin MCU is possible via read- and write- operations to memory-mapped registers of the core using JTAG.

Table 3.1
Quentin SOC features.

Technology	CMOS GF22
Chip Area	2.3 mm^2
Memory Transistors	520 kB
Equivalent Gates (NAND2)	1.8 Mgates
Voltage Range	0.5 V – 0.8 V
Body Bias Range	0.00 V – 1.4 V
Frequency Range	32 KHz – 670 MHz
Frequency Range (with FBB)	32 KHz – 938 MHz
Power Range	300 μW – 10.4 mW
Power Range (with FBB)	300 μW – 66.2 mW

3.1.3 Clock subsystem

Quentin includes three Frequency-locked loops (FLLs) that take as input an external 32 kHz reference clock and provide internal clocks up to 2.1 GHz. One FLL is used to provide the clock to *Kerbin*, one for the peripheral subsystem, and the remaining modules as CPU, memories, busses, etc.

3.1.4 Accelerator subsystem

The Quentin SOC hosts a BNN accelerator (called XNOR Neural Engine (XNE)) presented in [83] to increase performance and energy efficiency in tasks such as for example, image classifications. The XNE has four master ports towards the L2 memory for an overall memory bandwidth of 128 bits per cycle. All the configuration registers are memory-mapped and accessible by the core. The XNE can execute both convolutional and fully connected layers, autonomously from the core, once all data reside in L2.

3.1.5 Chip Implementation and Results

Figure 3.2 shows the floorplan of the Quentin SOC, while Table 3.1 summarizes its main features. The SOC was implemented using a flip-well (LVT) standard cell library. The design has been synthesized with Synopsys Design Compiler 2016.12, while Place & Route has been

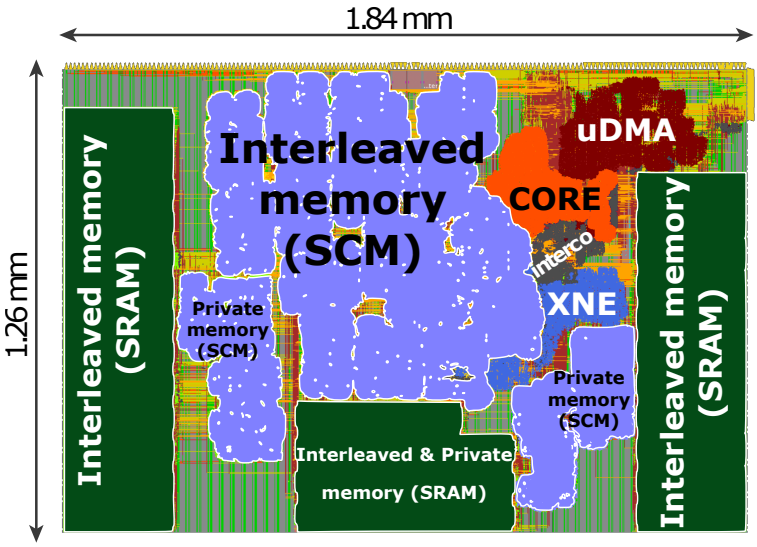


Figure 3.2: Quentin SOC floorplan.

TABLE 3.2
QUENTIN AREA BREAKDOWN IN MM^2

CPU subsystem	0.020
SRAM (504 kB)	0.817
SCM (16 kB)	0.292
ROM	0.009
I/O subsystem	0.056
XNE	0.014
Interconnect	0.009

performed with Cadence Innovus 16.10. The design has been closed at 200 MHz, worst-case conditions at 0.59 V for setup constraints, and best-case conditions at 0.88 V for hold constraints between -40°C and 125°C have been used to guarantee performance across the process, voltage, and temperature variations.

The floorplan area of the SOC is 2.31 mm^2 and its effective area is 1.22 mm^2 (6154 kGE). Its main modules are highlighted in Figure 3.2.

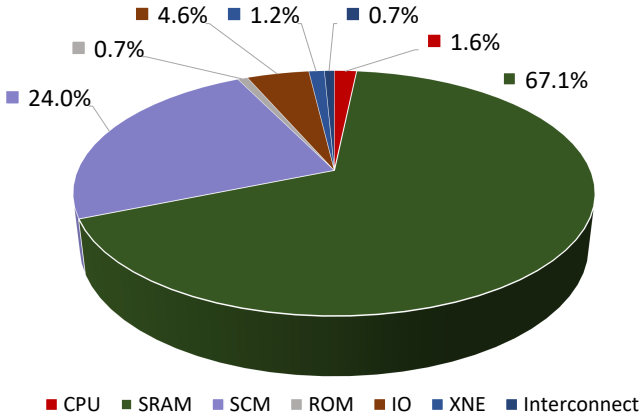


Figure 3.3: Quentin SOC area breakdown.

The two largest components of the SOC are the SRAM banks of the L2 memory subsystem (i.e., 504 kB), and by the 16 kB of SCM banks. Although the latch based implementation features approximately a 11x area overhead compared to approaches based exclusively on SRAMs (Table 3.2, Figure 3.3, 616 kB/mm^2 versus 54 kB/mm^2), it allows significant energy savings [147], and it enables more flexible power management strategies that can be played at the system level.

To exploit both the energy advantage of SCMs and area density advantage of SRAMs and to enhance the power/performance/precision tuning capabilities of Quentin, the chip was implemented as a multi power-domain system.

The SRAM cuts have separate power connections from the rest of the logic for both periphery and array, as shown in Figure 3.4. However, the periphery supply voltage must be equal to the rest of the logic supply voltage when operating, as no level-shifter has been implemented in the proposed SOC.

It is thus possible to configure the memory array, periphery, and logic circuit supply voltages using external power managers. The SRAM cuts can be completely power-gated, limiting the system to operate only with the 16 kB SCM, that can be scaled down as low

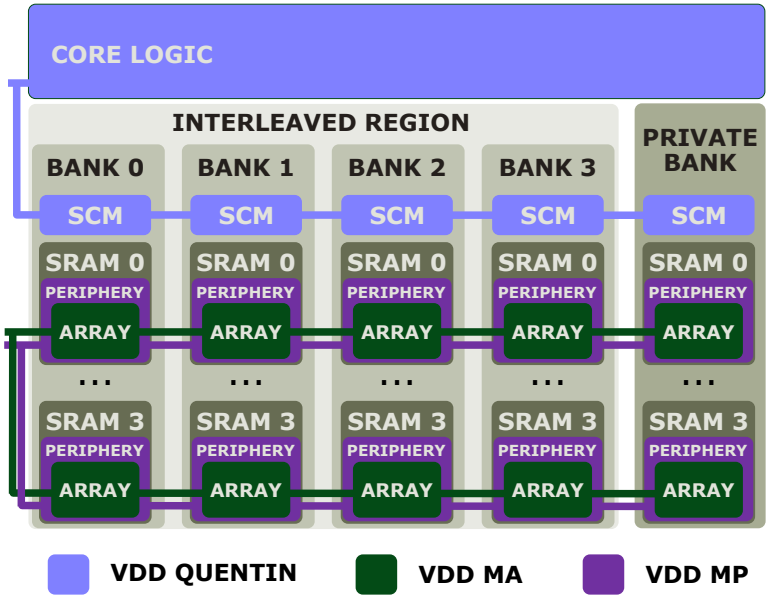


Figure 3.4: Quentin SOC Power Domains.

as the logic. In addition, FBB can be applied up to 1.4 V, to boost performance at every supply-voltage operating point.

To measure the chip performance, power, and energy consumption, an 8x8 32 bit matrix multiplication running on the *Riscy* core has been compiled and execute on Quentin. The chip has been tested on the Advantest SOC V93000 ASIC tester.

To characterize the system in different operational modes, three different setup have been tested for every measurements:

1. *SCM with SRAM gated.*
2. *SCM with SRAM on*
3. *SRAM*

In the *SCM-* experiments, data and code have been allocated on SCMs; otherwise, they are allocated on SRAMs.

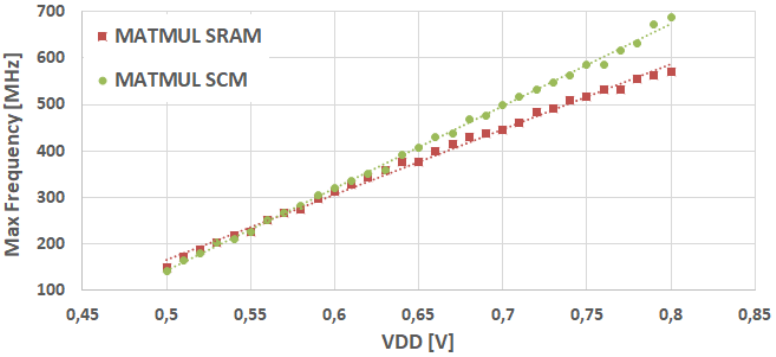


Figure 3.5: Maximum frequency against supply voltage when code and data reside on SRAMs or SCMs and no body-bias applied.

In the first case, the power connections of SRAM are power-gated (i.e., the supply voltage of periphery and array are at 0 V). In the second case, the SRAM are powered-on, but not used, for example, in case the SRAMs hold data (retentive-state) or the time to power-on/off is too long. In the *SRAM* setup, SRAMs power connections of array and periphery are connected to the same voltage level as the rest of the logic.

Figure 3.5 shows the maximum operating frequency of Quentin when running the matrix multiplication on SCMs or SRAMs. Note that the maximum frequency of the *SCM* setup is the same whether the SRAMs are switched on or off as no access to the SRAM cuts during the test happens. The chip reports no error when computing the matrix multiplication starting from 0.5 V, running at 148/156 MHz, and achieves the peak frequency of 570/670 MHz at 0.8 V when running on SRAMs and SCMs respectively with no body-bias.

Note that the matrix multiplication tests have not been performed as many times as to incur in SRAM failures due to the low-voltage, as explained in [123].

When applying FBB, the frequency can increase to more than 60% (at 0.6V), and it achieves 938 MHz at 0.8 V when 1.4 V are applied to the body-gate. The lower the supply voltage, the higher the effect of FBB. The effect of the magnified impact of body biasing at low

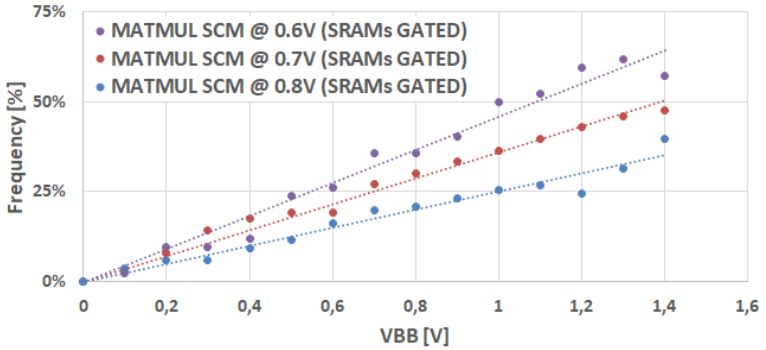


Figure 3.6: Performance benefits from forward body-bias. FBB impacts more low-voltage points.

voltage is a well-known effect seen in near-threshold Fully Depleted Silicon-On-Insulator (FDSOI) chips [129].

Figure 3.6 shows how the maximum frequency changes for three supply voltages when FBB is applied on the *SCM* setup. It is interesting to note that the lower the supply voltage, the higher the benefit of FBB.

The chip lowest power configuration uses only SCMs while SRAMs are power-gated. In this setup, it consumes only 0.95 mW at 0.5 V and no FBB, running at 156 MHz, and it consumes up to 32.1 mW at 0.8 V with 1.4 V FBB running at 938 MHz. When SRAMs are switched-on but not used, the leakage power increases by ~2 mW at 0.8 V and no FBB applied. Figure 3.7 shows how the leakage power increases at three different voltage levels when FBB is applied from 0 to 1.4 V, and data and instructions are in the SRAMs. It is possible to note that the leakage power increases faster at lower supply voltages. The voltage range used to test this chip ranges from 0.5 V to 0.8 V for both SRAMs and SCMs setup. However, SCMs and logic can be supplied at lower voltage, as explained in Subsection 3.1.6. Regardless the voltage supply, the system can exploit an *idle* mode consuming only leakage power (that depends on the mode as mentioned earlier) by not performing any I/O transaction, not using any accelerator, and executing the "wait for interrupt" (WFI) RISC-V instruction to clock-gate the CPU.

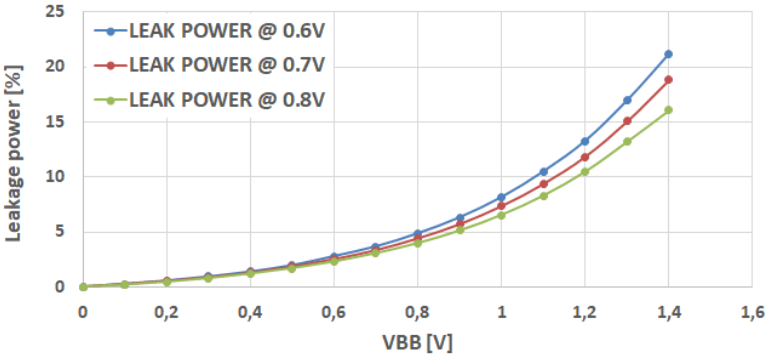


Figure 3.7: Power increase due to forward body-bias. The lower the supply voltage the higher the penalty.

Finally, the energy efficiency of the system measured in μW/MHz is shown in Figure 3.8. At every point, the three setups are measured at their maximum efficiency. Given the higher frequency and lower power consumption, the *SCM* setup with SRAMs gated is the most energy-efficient at every point as expected. However, in this configuration, the system has a limited memory capacity of 16 kB. In the configuration where operations are executed only on the SCMs, but with the whole memory available (SRAMs on), the system has higher energy efficiency than operating on SRAMs for frequencies >400 MHz. This can also be observed in Figure 3.5, as the supply voltage needed to reach such frequencies is higher on the *SRAM* setup, thus the power consumption increases.

3.1.6 Error-resilient application use-case

In this Subsection, we show how the proposed heterogeneous memory-subsystem implemented in *Quentin* can be exploited by an error-resilient application employing aggressive voltage scaling.

When the SRAMs are supplied at a lower value than the minimum working voltage, stored data can be corrupted with a remarkable Bit-Error-Rate (BER) (i.e., BER>$10^{-}5$ when voltage <0.5 V in this chip) as discussed in [123]. The lower the voltage, the higher the BER.

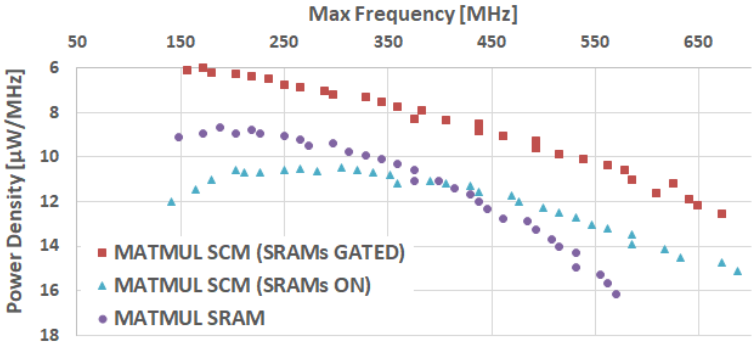


Figure 3.8: Quentin power density. Using only SCMs is more energy efficient than SRAMs due to better performance.

For our analysis, we use BNNs as they are partially error-resilient to random high error rates, (∼5% of accuracy loss with respect to the nominal accuracy under a BER of 10^{-4} [148]) as no bit in their activations and weights is inherently more significant than any other. Therefore, very aggressive voltage scaling can be applied to reduce the SRAMs power consumption.

The *Quentin* SOC memory subsystem can thus be exploited to put non-corruptible data (e.g., instructions executed by the core and core private data) to error-free SCMs, that are supplied at the same voltage level of the logic, whereas BNN weights, activations and partial results of internal layers in SRAMs.

In this scenario, the BNN has three potential sources of errors affecting the final BNN classification accuracy: *i)* weights reading *ii)* input features reading *iii)* activations storage. Threshold values are instead stored in the error-free SCMs. Partial-results are not affected by errors, as they are held inside the local buffer of the accelerator, implemented in SCMs.

Appreciable SRAM failures due to voltage scaling are visible already starting from 0.575 V. However, no accuracy loss is observable until 0.5 V (as for the matrix multiplication case). Accuracy can be traded for power when the voltage is further scaled down. Indeed, power

TABLE 3.3
PERFORMANCE COMPARISON WITH STATE-OF-THE-ART MCU.

	SleepWalker [149]	**REISC** [150]	**GAP8** SoC only [84]	**Mr-Wolf** SoC only [64]	**This Work**
Technology	CMOS 65nm LP	CMOS 65nm LP	CMOS 55nm LP	CMOS 40nm LP	CMOS 22nm FDSOI
CPU	16 bit MSP430	32 bit	32 bit RV32IMCXpulp	32 bit RV32IMC	32 bit RV32IMCXpulp
FPU	NO	NO	NO	NO	YES
I$/D$/L2	64 B n.a. 18 kB	8 kB 8 kB n.a.	4 kB n.a. 512 kB	n.a. n.a. 512 kB	n.a. n.a. 520 kB
Voltage range (SRAMs)	0.4 V (1.0 V)	0.54 - 1.2 V (0.4 - 1.2 V)	1.0 - 1.2 V	0.8 - 1.10.4 V	0.5 - 0.8 V
Frequency range	25 MHz	82.5 MHz	32 kHz - 250 MHz	32 kHz - 450 MHz	32 kHz - 938 MHz
Best Power Density (SRAM on)	6.1 μW/MHz[1]	10.2 μW/MHz	180.2 μW/MHz 1.0 V, 150 MHz	33.3 μW/MHz 0.8 V, 170 MHz	8.7 μW/MHz 0.52 V, 187 MHz
Best Power Density (SRAM off)	5.3 μW/MHz[1] [2]				6 μW/MHz 0.51 V, 171 MHz
Best Performance	25 MOPS	82.5 MOPS	650 MOPS	234 MOPS	2400 MOPS
Best Energy Efficiency (SRAM on)	164 MOPS/MHz [1] at 25 MOPS	98 MOPS/MHz at 0.54 MOPS	14.4 MOPS/MHz at 390 MOPS	35.1 MOPS/MHz at 88.4 MOPS	300 MOPS/MHz at 486 MOPS
Best Energy Efficiency (SRAM off)	188 MOPS/MHz [1] [2] at 25 MOPS				433 MOPS/MHz at 445 MOPS

[1] Without accounting for DC/DC overhead
[2] Assuming 100% hit rate on the I$ and PMEM switched-off

can be reduced to 674 μW at 0.42 V for an accuracy drop below 1%. This result allows for always-on Internet-of-things (IoT) end-node with an expected long lifetime (in the order of months or years) and in applications where the peak power dissipation is a critical concern (e.g., implantable devices). However, energy efficiency is worse due to the leakage power dominating the overall power consumption and performance degradation. In fact, the maximum frequency at 0.42 V is only 18 MHz.

Multicore PULP systems based on RISC-V have already been implemented in [84] and [64]. A comparison against their fabric controller and two additional efficient processors [149, 150] is shown in Table 3.3. Quentin shows the highest energy efficiency and performance as a single-core MCU thanks to the compound of advanced architecture design and technology. With respect to [64, 84] and [149], it does not implement any on-chip power manager and does not have any state retentive memory. Thus it has to rely on external memories. With respect to [64], the fabric controller adopts a more performant

core, whereas Quentin is implemented in more advanced technology with respect to [84].

Chapter 4

Fixed-Function Accelerators

In this Chapter, we cover how computations on fixed-function accelerators can boost the energy efficiency of Internet-of-things (IoT) end-nodes during the execution of standard functions as 2D-convolutions, widely used in Convolutional Neural Networks (CNNs). We extend a CNN accelerator with SIMD operations to increase the energy efficiency of applications where CNN weights can be represented with less precision. We extend the original engine presented in [151] that computes convolutions on 16 bit for both weights and data, to performs two parallel convolutions on 8 bit weights, or four parallel convolutions on 4 bit weights, whereas data are kept to 16 bit format. Such accelerator has been tightly-coupled integrated into a cluster of four DSP-enhanced cores. We show that the accelerator is faster and more energy-efficient than four DSP-enhanced cores. The presented accelerator has also been used as a baseline for the accelerator integrated into the GreenWaves GAP8 [84] MCU.

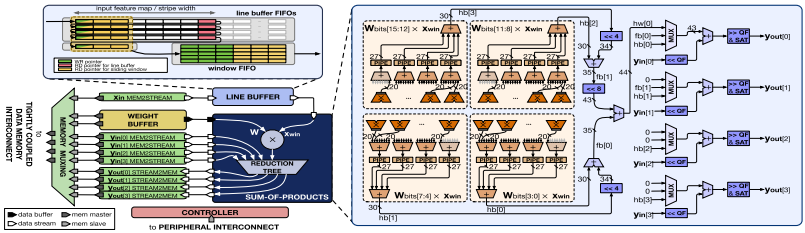


Figure 4.1: *Fulmine* HWCE architecture, with the *controller* shaded in red, the *wrapper* in green, and the *datapath* in blue. The diagram also shows details of the line buffer and sum-of-products sub-modules microarchitecture.

4.1 Hardware Convolution Engine SIMD Extensions

In this Section, we describe the SIMD extensions to the CNN accelerator and how those extensions achieve higher energy efficiency than a cluster of four DSP-enhanced RISC processors based on the OpenRISC Instruction Set Architecture (ISA) [152].

The Hardware Convolution Engine (HWCE) is based on a precision-scalable extension of the design proposed by Conti [151].

The main purpose of this engine is to provide a performance and efficiency boost on CNN kernels. It equips power- and area-aware design techniques such as clock-gating and datapath sharing in a time-multiplexed fashion.

CNN layers are processed by computing the 2D-convolution, which maps N_{if} input feature maps into N_{of} output feature maps utilizing a set of filters W. Filters are usually as large as 3x3, 5x5, or 7x7. The output feature maps are then processed by pointwise non-linear activation functions, often a rectifier (ReLU), or a hyperbolic tangent, or a Sigmoid function. The linear part of convolutional layers is usually the most dominant operation in a CNN model, and it is represented by:

$$\mathbf{y}(k_{\text{of}}) = \mathbf{b}(k_{\text{of}}) + \sum_{k_{\text{if}}=0}^{N_{\text{if}}-1} \Big(\mathbf{W}(k_{\text{of}}, k_{\text{if}}) * \mathbf{x}(k_{\text{if}}) \Big), \tag{4.1}$$

for the $k_{\text{of}} \in 0 \cdots N_{\text{of}} - 1$ output layer.

The HWCE has been designed to implement efficiently the task described in Equation 4.1 and to support 5x5 and 3x3 2D-convolutions, whereas any other arbitrary filter size has to be handled with software helper functions by combining 5x5 and 3x3 filters.

The internal datapath of the accelerator can perform one 16 bit, two 8 bit or four 4 bit parallel convolutions on different output k_{of} feature maps, while input feature maps are always represented in 16 bit.

In these scaled precision modes, a similar level of accuracy to the 16 bit full precision CNNs can be maintained by proper training [153, 154], with access to significantly improved performance, memory footprint, and energy efficiency, as is shown in Section 4.3.

Figure 4.1 shows the HWCE architecture, which can be divided into three main components: a *datapath* performing the main part of the data plane computation in a purely streaming fashion, relying on an AXIStream-like handshake for back-pressure; a *wrapper* that connects and decouples the datapath streaming domain from the memory-based cluster; and a *controller* that provides a control interface for the accelerator.

In the full-precision 16 bit mode, the HWCE datapath performs a dot-product between a pre-loaded filter $\mathbf{W}$ (stored in a weight buffer) and a 5x5 $\mathbf{x}_{\text{win}}$ window extracted from a linear $\mathbf{x}$ input feature map stream. The output of the dot-product is accumulated into an input $\mathbf{y}_{\text{in}}$ value; in other words, the accelerator needs no internal memory to perform the feature map accumulation component of Equation 4.1 but uses the shared-memory of the cluster directly.

The weight buffer can also host packed-values of two 8 bit or four 4 bit weights, that correspond to two or four different filters $\mathbf{W}$. The datapath has been designed hierarchically to maximize resource sharing between the three SIMD modes. Four sub-modules (shown in orange in Figure 4.1) compute the sum-of-products of $\mathbf{x}_{\text{win}}$ with a 4 bit slice of $\mathbf{W}$ each, using a set of signed multipliers and a first-stage reduction tree. A second-stage reduction tree and a set of multiplexers are used to combine these four partial sums-of-products to produce one, two, or four concurrent $\mathbf{y}_{\text{out}}$ outputs; fractional part normalization and saturation are also performed at this stage. As multiple accumulations of convolutions are performed concurrently, the $\mathbf{y}_{\text{in}}$ and $\mathbf{y}_{\text{out}}$ streamers

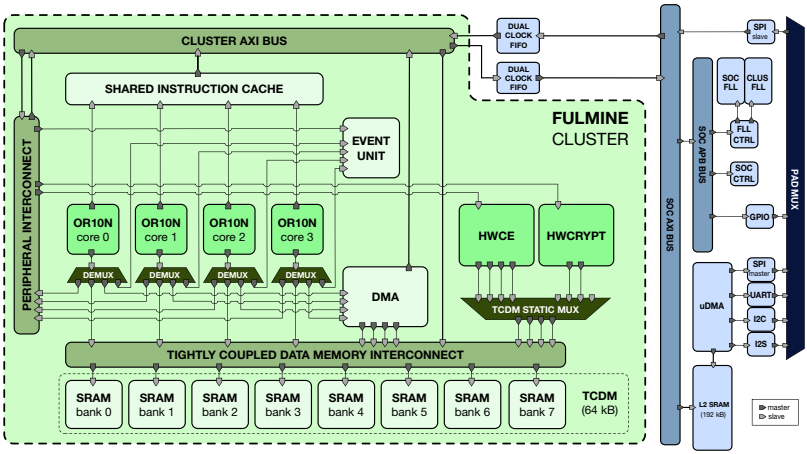


Figure 4.2: *Fulmine* SoC architecture. The SOC domain is shown in shades of blue, the CLUSTER domain in shades of green.

are replicated four times. All HWCE blocks are aggressively clock gated so that each component consumes power only when in active use.

4.2 Fulmine chip

The HWCE accelerator has been integrated into a chip called *Fulmine*, a four DSP-enhanced cores MCU based on the Parallel Ultra Low Power Platform (PULP) architecture. The chip hosts two accelerators: the HWCE and a cryptographic engine, which is not part of this Chapter as it is not a contribution of this Thesis. The cores are based on the OpenRISC ISA [155], and they have been extended with similar instructions discussed in Chapter 2. Figure 4.2 shows the *Fulmine* architecture.

The CLUSTER domain is built around six *processing elements* (four general-purpose OpenRISC processors and two flexible accelerators) that share 64 kB of level 1 Tightly-Coupled Data Memory (TCDM), organized in eight word-interleaved Static Random Access Memory

(SRAM) banks. A low-latency logarithmic interconnect [146] connects all processing elements to the TCDM memory, enabling fast and efficient communication among the cluster players. The two hardware accelerators, *Hardware Cryptography Engine* (HWCRYPT) and *Hardware Convolution Engine* (HWCE), can directly access the same TCDM used by the cores.

To avoid a dramatic increase in the area of the TCDM interconnect, as well as to keep the maximum power envelope in check, the two accelerators share the same set of four physical ports on the interconnect, thus they are used in a time-interleaved fashion.

The cores are based on an in-order, single-issue, four-stage pipeline similar to the *Riscy* core described in Chapter 2.

The cluster features a set of peripherals, including a Direct Memory Access (DMA) engine, an event unit, and a timer.

A sophisticated power management architecture distributed between the SOC and CLUSTER domains can completely clock-gate all the resources when idle. The power manager can also be programmed to put the system in a low power retentive state by switching down the FLLs and relying on the low-frequency reference clock (*low freq* and *idle* mode). Finally, it can be used to program the external DC/DC converter to fully power-gate the CLUSTER domain.

The event unit is responsible for automatically managing the transitions of the cores between the active and idle states. This happens when the processors execute an explicit *Wait For Event* instruction, for example, during a synchronization barrier or after a DMA transfer. The event unit then stalls the processors, and once all pending transactions (e.g., cache refills) are complete, they are clock-gated.

As the CLUSTER and SOC power domains are managed independently, it is possible to put the CLUSTER in *idle* mode transparently, where it consumes less than 1 mW, when waiting for an event such as the end of an I/O transfer to L2 or an external interrupt that is expected to arrive often.

The accelerator's shared-memory nature enables efficient zero-copy data exchange with the cores and the DMA engine, orchestrated by the cluster event unit. This architecture enables complex computation patterns with frequent transfers of data set tiles from/to memory.

A typical application running on the *Fulmine* SoC operates conceptually in the following way. First, the input set (e.g., a camera

frame) is loaded into the L2 memory from an external I/O interface using the μDMA. The cluster can be left in sleep mode during this phase and woken up only at its conclusion. The input set is then divided into tiles of appropriate dimension so that they can fit in the L1 shared TCDM; one tile is loaded into the cluster, where a set of operations are applied to it either by the SW cores or the HW accelerators. These operations can include 2D-convolutions (on the HWCE), plus any SW-implementable kernel. The output tiles are then stored back to L2 memory using DMA transfers, and computation continues with the next tile. Operations such as DMA transfers can typically be overlapped with computation by using double buffering to reduce the overall execution time.

The cores are used both for actual computation on the data set and for control; to avoid inefficient busy waiting, events are employed by the HW accelerators to notify completed execution. Accelerator events trigger an appropriate interrupt in the controller core while it is either in sleep and clock-gated, or executing a filter of its own in parallel to HW-accelerated computation.

For example, while the HWCE is executing the dot-product of on the CNN layer L_{k}, the four cores can be used to execute the pooling and non-linear activations functions of the previous layer $L_{\mathrm{k}-1}$.

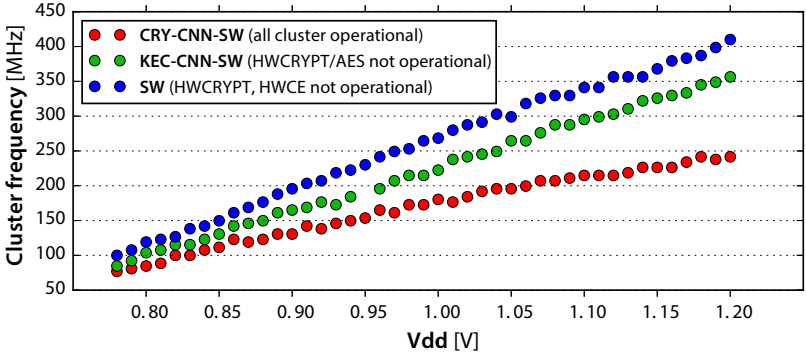
4.3 Experimental evaluation

In this Section, we analyze measured performance and efficiency of our platform on the manufactured *Fulmine* prototype chips, fabricated in UMC 65 nm LL 1P8M technology in a 2.62 mmx2.62 mm die (6.86 mm^2).

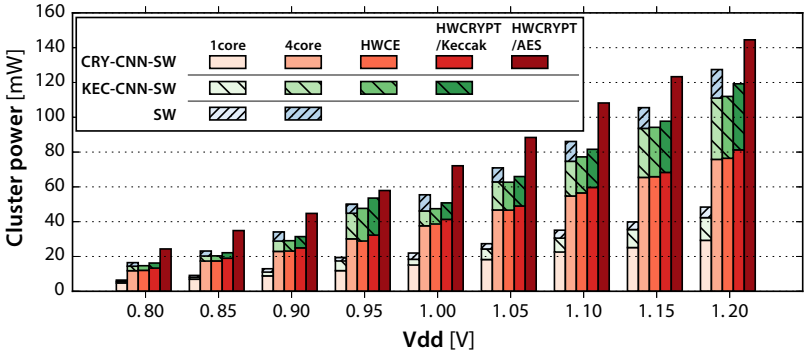
4.3.1 System-on-Chip Operating Modes

Fulmine can operate in many different conditions: in pure software, with part of the accelerator functionality available, or with both accelerators available. These modes are characterized by very different average switching activities and active power consumption.

There are three operating modes available for *Fulmine*:



(a) Cluster maximum frequency in three operating modes.



(b) Cluster power at maximum frequency in three operating modes.

Figure 4.3: Cluster maximum operating frequency and power in the CRY-CNN-SW, KEC-CNN-SW, and SW operating modes. Each set of power bars, from left to right, indicates activity in a different subset of the cluster. KEC-CNN-SW and SW bars show the additional power overhead from running at the higher frequency allowed by these modes.

1. CRY-CNN-SW, where both the accelerators in their full capabilities and the cores can be used;
2. KEC-CNN-SW, where both the accelerators and the cores can be used, but the cryto engine is limited to KECCAK-f[400]

primitives. This allows the system to run faster as only the non-timing critical parts of the crypto engine can be used;

3. SW, where only the cores can be used to maximize frequency.

Figure 4.3 shows frequency scaling in the three operating modes while varying the cluster operating voltage V_{DD}. The three modes were designed so that at V_{DD} =1.2 V, current consumption under full load is close to 100 mA (i.e. 120 mW of power consumption), as can be seen in Figure 4.3b.

4.3.2 HWCE Performance and Power Evaluation

The *Fulmine* SoC includes many distinct ways to perform the basic operation of CNNs, i.e., 2D convolutions. In software, a naïve single-core implementation of a 5x5 convolution filter has a throughput of 94 cycles per pixel. Parallel execution on four cores can provide almost ideal speedup reaching 24 cycles/px. Thanks to the SIMD extensions described in Chapter 2, an optimized multi-core version can be sped up by almost 2x down to 13 cycles/px on average.

With respect to this baseline, the HWCE can provide a significant additional speedup by employing its parallel datapath, the line buffer (which saves input data fetch memory bandwidth), and weight precision scaling. We measured average throughput by running a full-platform benchmark, which therefore takes into account the overheads for real-world usage: line buffer fill time, memory contention from cores, self-contention by HWCE inputs/outputs trying to access the same TCDM bank in a given cycle. Considering the full precision 16 bit mode for the weights, we measured an average inverse throughput of 1.14 cycles per output pixel for 5x5 convolutions and 1.07 cycles per output pixel for 3x3 convolutions - the two sizes directly supported by the internal datapath of the HWCE. This is equivalent to a 82x speed up to the naïve single-core baseline, or 11x to a fully optimized 4-core version.

As described in Section 4.1, the HWCE datapath enables application-driven scaling of arithmetic precision in exchange for higher throughput and energy efficiency. In the 8 bit precision mode, average inverse throughput is scaled to 0.61 cycles/px and 0.58 cycles/px for the 5x5 and 3x3 filters, respectively; in 4 bit mode, this is further improved to 0.45 cycles/px and 0.43 cycles/px, respectively. In the 4 bit precision

mode, the HWCE is fully using its 4-port memory bandwidth towards the TCDM in order to load 4 $\mathbf{y}_{\text{in}}$ partial results and store back 4 $\mathbf{y}_{\text{out}}$ ones. Further performance scaling would, therefore, require an increase in memory bandwidth.

Figure 4.4 reports time and energy per pixel, running the same set of filters in the KEC-CNN-SW operating mode while scaling the V_{DD} operating voltage. At 0.8 V, the energy to spend for an output pixel can be as low as 50 pJ per pixel, equivalent to 465 GMAC/s/W for a 5x5 filter. At 0.8 V, the four-core cluster can run up to 120 MHz, consuming 12 mW. Thus, the energy to spend for an output pixel can be as low as 1316 pJ per pixel, equivalent to 19 GMAC/s/W, 26x less efficient than the HWCE.

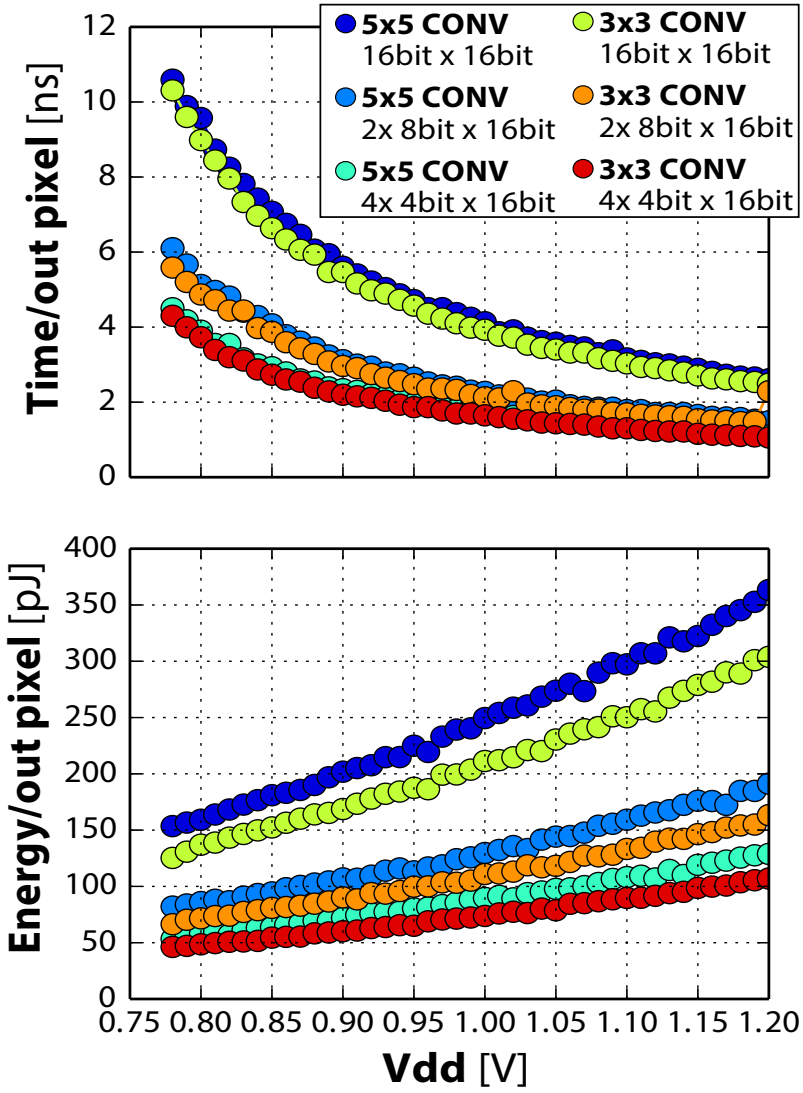


Figure 4.4: Performance and efficiency of the HWCE accelerators in terms of time/energy for elementary output.

Chapter 5

Software Accelerators

In this Chapter, we cover how computation on software-defined accelerators can boost the energy efficiency of edge-computing devices that deal with the wide-range of bio-applications. We propose a novel, fully programmable platform, based on the Parallel Ultra Low Power Platform (PULP) architecture (Micro Controller Unit (MCU) plus a software-accelerator of four/eight DSP-enhanced cores) for neural interfaces that can efficiently: *i)* perform on-node 64-channel spike sorting on detected spikes thanks to their event-based nature; and *ii)* compressing full bandwidth Action Potential (AP) streams as well as extracting EEG features thanks to the powerful multicore architecture.

The system is composed of two state-of-the-art devices: an MCU that acquires data and processes them with a software-programmable accelerator made of multicores, and an AFE optimized for neural signals.

As more complex artificial intelligence and data-analytic algorithms enable the extraction of hidden information from neurons at higher accuracy, the MCUs need to be high performant and energy-efficient to allow on-the-edge computation. For this reason, Mr.Wolf [64] has been selected as the MCU candidate for the proposed system as its

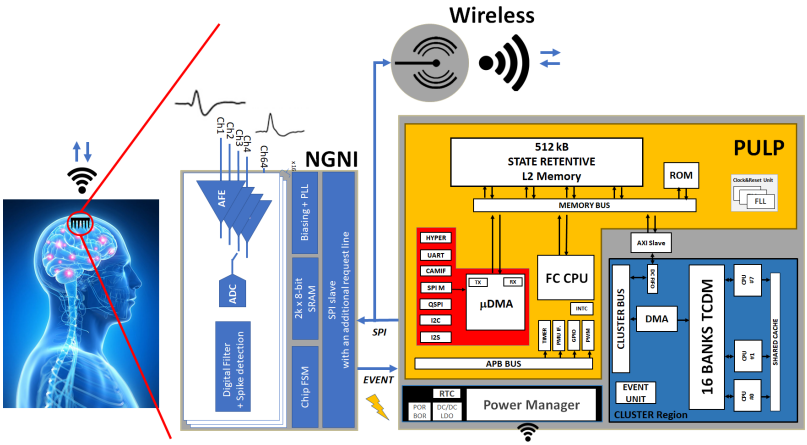


Figure 5.1: *Neuro-PULP* architecture. The Analog front-end (AFE) called NGNI senses 64 probes and sends to the Digital Signal Processing (DSP) (PULP) the neural activity. Spikes' clusters and time-stamps or compressed data streams are sent wirelessly to a PC, that can send a new processing chain or updates to the current SW.

computational capabilities on bio-signals have already been shown to be top performer [106, 156–159].

The AFE selected is the programmable event-based NGNI presented in [72] as it is ultra-low-power, and it enables different levels of programmability. We believe that having two different devices, rather than a single mixed-signal device is more versatile as different AFEs can then be plugged to the MCU when the number of channels or the signal features is different. Also, different technology nodes can be chosen for the two parts of the system to optimize the performance-cost trade-off.

The high energy efficient and proportional power consumption of the proposed system allows both the execution of computationally hungry algorithms (as 64-channels compression) leveraging the multicore cluster accelerator of Mr.Wolf [64], and the execution of light algorithms on detected spikes in an ultra-low-power budget leveraging the event-based nature of both the subsystems NGNI-Mr.Wolf.

In addition, we exploit the PULP cluster to boost the energy efficiency of an EEG drowsiness device presented in [105]. Instead of Mr.Wolf, this application has been implemented in the PULPv3 MCU [77]. However, the cluster of the two MCUs has a very similar architecture as both based on the PULP architecture. The analysis of EEG signals is one of the most common method to detect drowsiness. It has been demonstrated that the variation of the brain rhythms on the alpha waves band (7.5–13 Hz) indicates a drowsy state [160]. The system uses an EEG signal to detect alpha wave activities by applying the Short-Time fast Fourier transform (FFT) (STFT) over 512 samples (1.024 s) with sliding windows of 32 samples (64 ms). Here, the power spectrum is calculated on the frequencies of interest (7.5–13 Hz), and the maximum value within this band is selected for each time step (64 ms). With respect to a low-power ARM Cortex-M4 microcontroller implementation, the proposed optimization exploits the PULPv3 MCU to extract the EEG features on four Central Processing Units (CPUs).

The major contributions of this Chapter are:

- the design of new neural computing platform based on the integration of two devices to enable performant yet low-power and highly-programmable brain interfaces;
- an optimized SW library to acquire data from the 64-channel AFE leveraging the autonomous PULP I/O subsystem. Such library

handles both high-bandwidth of data streams and event-based information.

- implementation of two example-applications in C language for neural signal processing: *i)* one application leverages streams of high-bandwidth data and the PULP cluster to compress them (*streaming* mode). This feature is useful to reduce the output bandwidth of the acquired raw signals. *i)* an application that leverages the event-based nature of brain action potentials. Such application exploits the event-based hierarchical-chain of the AFE, the I/O subsystem, and the processing part done on a single core (*event* mode).

- implementation of the STFT transform on the PULP cluster to boost the energy efficiency of an EEG drowsiness detection device.

5.1 *Neuro-PULP*

In this Section, the architecture of the proposed system to process high-frequency neural activity is described. The block diagram of *Neuro-PULP* is shown in Figure 5.1, consisting of three main parts: the (external) electrode array, the 64-channel neural-interface *NGNI* and the Mr.Wolf PULP SoC. The NGNI is a programmable neural recording system that can be configured to select the number of channels and whether to send data streams or only the detected spikes. Spikes are detected with the absolute value threshold crossing detection. Mr.Wolf has been adopted as the MCU in the proposed system as it offers proportional power consumption and high-performance thanks to its power manager and software-accelerator. It consists of an advanced microcontroller architecture called fabric-controller featuring an autonomous I/O subsystem (called μDMA) that reaches a peak aggregated bandwidth for transmission and reception of 14 Gbit/s [133], 512 kByte of L2 memory, a cluster of 8 cores, and it implements six power modes. The NGNI and Mr.Wolf are connected via SPI running at 50MHz. The NGNI issues a request (*req*) signal every time a new packet has to be sent. First, it sends the *header* packet containing information about the upcoming samples (data valid, the channel

number, etc.). The next packet contains a 16-bits data or 16x16-bits data in *streaming* or *spike* mode respectively. In *event* mode, the optimized SW driver keeps PULP waiting most of the time in sleep mode for the NGNI AFE to detect a spike. This means that when no spikes are detected in any of the 64 channels, the system consumes ∼7 mW, 6.49 mW for the AFE and 0.55 mW for Mr.Wolf.

The *req* signal, mapped on a general-purpose IO of Mr.Wolf, acts as a wake-up event that starts the acquisition and processes the spike. When the request signal is raised, the power-manager of Mr.Wolf triggers the *µDMA*, which autonomously acquires the packets, and then it awakes the core that processes the spike, and enqueues the next commands to the *µDMA*. Note that this implementation consumes power proportionally to the spike-activity as it is active only on-demand. Figure 5.2(a) shows the data packet from the NGNI and the FSM implemented in the fabric-controller as well as a temporal diagram of the application operating in *event* mode. In *streaming* mode, PULP continuously acquires signals from the NGNI sharing the protocol mentioned above, and it processes them. This means that for every enabled channel, the AFE sends a request containing the *header* and the 16bit sample. When 64 channels are sampled at 15 kHz, the frequent interaction with the I/O subsystem requires a highly optimized code run by the fabric-controller at high-frequency (300 MHz). This requires the core to take control every 2 SPI transactions (808 ns) as shown in Figure 5.2.(b). Once there is sufficient data stored, the computation is offloaded to the multicore cluster accelerator in a pipelined fashion. The cluster processes the data stream collected, as shown in Figure 5.2(b).

5.2 *Neuro-PULP* Case Study Applications

The versatility of *Neuro-PULP* as its suitability for multi-modal neural interface processing tasks is demonstrated by running two example applications that represent the two different modes (*streaming* and *event*): called *Compress-Stream* and *Spike-Sorting*. Both the applications start by configuring the AFE to enable 64 channels, to load the signal conditioning parameters (amplifier gain, sample frequency of 15 kHz) and to select the operational mode. The computation starts once enough samples have been acquired.

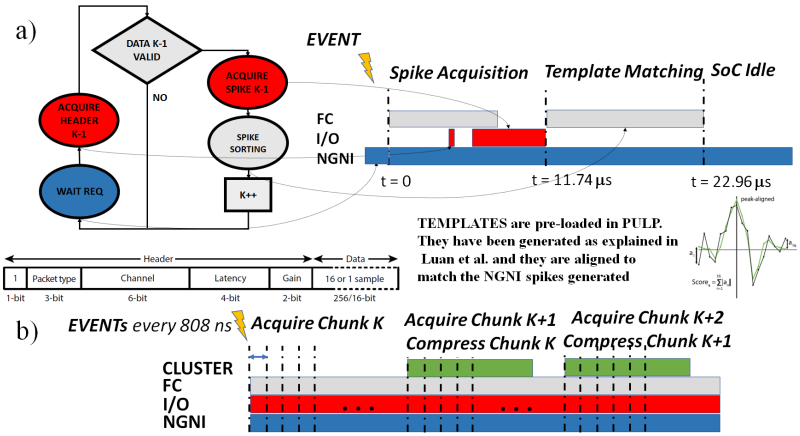


Figure 5.2: (a) Spike sorting mode: FSM implemented in the MCU. For every new request, a new spike is acquired and templates matching is performed. (b) Streaming mode: 2 SPI packets are received until 128 samples per channel have been acquired. The cluster is turned-on to compute the Discrete Wavelet Transform (DWT) and to compress the data, meanwhile the rest of the system collects the new chunk.

5.2.1 *Compress-Stream* application

The *Compress-Stream* application shows how the MCU can be used to compress raw neural AP signals. This is useful for reducing the data rate of transmitted signals if neuroscientists are interested in the raw data stream. Another useful scenario is the one presented in [161], where AP streams collected by the neural-interface are sent to a PC to perform off-line training and to generate spikes' templates. Once PULP acquired 128 samples per channel, (in 8576 μs), the DWT is calculated to map the signals in a sparse domain and compressed sensing is used to reduce the dimension of the acquired chunk. Both the kernels are executed in the cluster running at 40 MHz. Even if this work focuses on authors' previously studied processing chains, future work can leverage the cluster capabilities to implement higher performance spike sorting algorithms or even activity recognition from spike trains as it can run up to 350MHz. Double buffering is used such that the cluster operates on the *kth* stream, while the fabric-controller acquires the *k+1th* stream as shown in Figure 5.2.(b). The estimated power consumption of the whole system (NGNI, communication, and PULP) is 44.14 mW (690 μW/channel), and the break down is shown in Figure 5.3. It is interesting to note that such a complex application has never been implemented in an MCU device tightly coupled to the AFE, as it would have required too high performance or consumed too much power. This result demonstrates not only the high power efficiency of *Neuro-PULP*, but also that efficient MCU and AFE can be used in the next generation of neural interfaces without giving up the high flexibility offered by programmable devices.

Fixed number of channels case In *streaming* mode, the core has to be awakened for every data coming from every enabled channel. However, the channels in the NGNI are grouped in 16 blocks, so is the transmission. The NGNI sends all the enabled "channel 1" from all the blocks, then the second, etc. When 64 channels are active, channels from the same block are sent every 16 transactions in a time-multiplex fashion. For this reason, an optimized version of the *Compress-Stream* has also been implemented. In this flavor, the PULP μ*DMA* is programmed to acquire 16 data independently without the need for the core supervision between transfers. Thus the interaction

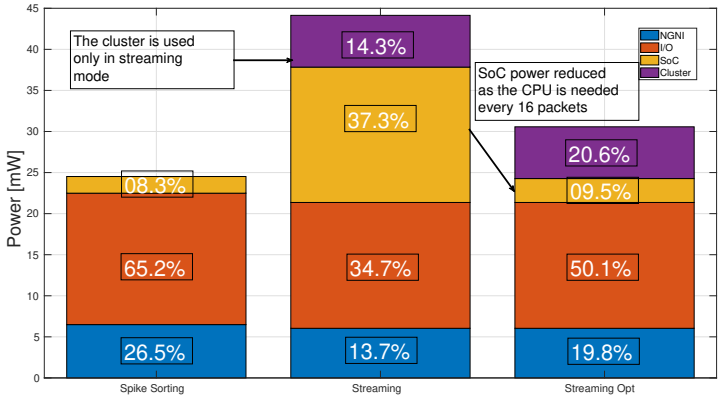


Figure 5.3: In *sorting* mode, the active power consumption is reduced as the cluster is not used and the fabric-controller runs at lower frequency (50 MHz against 300 MHz) . The power consumption is reduced by 2.24x wrt the *streaming* mode. The *streaming* Opt. bar shows the fixed number of channels case where the fabric-controller run at lower frequency due to the less frequent interaction with the I/O subsystem.

between the core and the I/O subsystem is reduced to every 16750 ns, which allows the frequency to be reduced to 80 MHz and the power to 30.58 mW (478 μW/channel).

5.2.2 *Spike-Sorting* application

Table 5.1
Comparisons of SoA Closed-Loop Neural Interfaces

Ref	AFE	Digital Processor	Application	Max Samp. Freq. [kS/s]	Power/Ch. [mW]
[162]	128-ch. COTS	Blackfin BF532 DSP	Template matching spike sorting	31.25	2.063
[163]	4-ch. COTS	32-bit AVR® AT32UC3C1512C	Unsupervised spike detection, sorting	21	27.935[1]
[164]	32-ch. Intan	FPGA	Spike detection and Compression	21	5.000
[61]	32-ch. Intan	32-bit ARM® Cortex-M0+	Waveform-derivative features spike sorting	15–31	0.975[2]
[161]	32-ch. Intan	FPGA	Template matching spike sorting	15	0.454[3]
This work	64-ch. custom SoC [72]	32-bit RISC-V Multicore PULP [64]	64-ch. Template matching spike sorting on AFE-detected spikes	15	0.114[4]
This work	64-ch. custom SoC [72]	32-bit RISC-V Multicore PULP [64]	64-ch. AP Compression with DWT (Wavelet)	15	0.478[4]

COTS=Commercial off-the-shelf, [1]Estimated as 3.3V×(4-channels×380 μA/channel+CPU 490μA/MHz×66 MHz)=111.738 mW=27.935 mW/channel. [2]Power of AFE not included [3]Spike event mode without MCU and SDCard power. [4]Spike sorting with 10 spikes/s/channel on average. [5]Discrete Wavelet Transform on 128x64 samples.

The second operating mode demonstrates the processing chain from the NGNI on-site detected spikes to the PULP template matching for real-time systems. At the beginning of the application, the NGNI is programmed to work in AP snippets mode, and threshold values for spike detection are loaded. For each acquired spike, the sum of absolute differences against all the four templates per channel [161] is executed on the fabric-controller running at 50 MHz. In this case, the multicore cluster is not used as the complexity of the algorithm, and the input bandwidth is low enough to be handled by only one core. However, thanks to the advanced power manager of Mr. Wolf, the software accelerator is kept is sleep mode with negligible contributions to the total power consumption.

At a nominal spike rate of 10 spikes/s [165], the event-based protocol is particularly suited to save power as Mr. Wolf is most of the time in sleep mode consuming only 0.55 mW. Once a spike is detected, the *req* signal connected to the Mr. Wolf GPIO triggers the acquisition and sorting task, as shown in Figure 5.2(a). This mode is typically attractive for ultra-low-power consumption as it is reduced by 2.24x with only 383 μW/channel during the active sorting phase, consuming only 6.49 mW from the ADC and 0.55 mW from PULP when no spikes are detected as shown in Figure 5.3. For further investigations, two ideals systems operating in *streaming* mode are compared to *Neuro-PULP* wrt the spikes rate, as shown in Figure 5.4. The two PULP based systems assume double buffering and zero-overhead on top of the template matching algorithm and operate at the minimum voltage-frequency required. The first system *Always On Streaming mode* assumes that every 66.7μs (15kHz sampling rate), one new sample per channel is ready in the main memory to be processed. The second system *Always On Packed Streaming mode* assumes that data are ready every 16 samples. The minimum voltage-frequency is selected to estimate the power consumption in both the ideal scenarios. For the *Always On Streaming mode*, the multicore cluster is activated, as doing a computation every 66.7μs would require too high performance for only one core. As these two new ideal systems operate in *streaming* mode, the power is not proportional to the spikes rate. It is quite interesting to note that for a typical spike rate of 10 spikes/s per channel (or 640spikes/s for all the channels), the event-based approach is 4.15x more power-efficient than the *Always On Streaming mode*

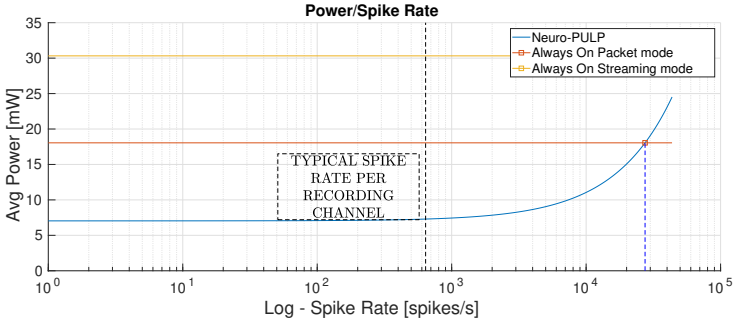


Figure 5.4: In the x-axis, the aggregate spike rate of the 64 channels is used to estimate the power proportional power of *Neuro-PULP*. In the typical 10 spikes/s per channel (~640spikes/s for all the channels) the event-based approach is easily outperforming the ideal *streaming* mode.

system and 2.47x than the *Always On Packet mode* system, even with zero-overheads.

5.3 Drowsiness Detection on the PULP architecture

This Section describes the implementation of the 5-levels drowsiness detection alarm system on a 4-cores PULP architecture [77]. Although a typical MCU with single ARM Cortex-M4, as the STMicroelectronics STM32F407 microcontroller used in [105], has enough resources to execute the drowsiness detection at the required performance, using a software accelerator made of 4 DSP-enhanced cores boosts the energy efficiency of the whole system, so the battery lifetime is longer. As the most compute demanding part of the drowsiness detection algorithm based on alpha waves is the FFT, a great effort has been spent to optimize this kernel on the PULP architecture, exploiting a fine-grained data-parallel scheme supported by the programming model. The FFT algorithm requires to compute a set of butterflies on the N input samples (where N is the size of FFT, 512 for this application) for

each stage of computation, and the number of stages is equal to the 2-base logarithm of the number of input samples (9 in this case). In the baseline radix-2 algorithm, after each stage, data generated by the butterflies of the previous stage has to be shuffled to compute the butterflies on the following stage.

On the PULP architecture, the FFT is computed by splitting the butterflies' calculation homogeneously among the four cores and synchronizing the cores using hardware barriers after each butterfly stage to maintain the data consistency. Two different implementations have been evaluated, described in the following.

As explained above, the baseline radix-2 FFT requires a synchronization barrier after each stage of butterflies (i.e., nine barriers), leading to a relevant synchronization overhead not amortized by the small computational load required by each stage of butterflies. A more optimized approach relies on the radix-8 algorithm. Exploiting this implementation, each butterfly performs a single Discrete Fourier Transform (DFT) among eight samples instead of 2, as in the radix-2 implementation. This reduces the number of butterflies to be computed at each stage, but it increases the butterfly's computational complexity. The proposed implementation is composed of 3 stages, each with 64 butterflies (16 for each core). Therefore three barriers are triggered to accomplish the full 512 samples FFT. Hence, this approach increases the available parallelism and reduces the synchronization overhead with respect to the radix-2 algorithm. The computation of the magnitude of the FFT is parallelized by dividing the signal by 4. Thus each core works independently without synchronizations. The FFT and magnitude parallel implementations feature a speed-up greater than 3.9 with respect to the single-core version on PULP, showing a quasi-ideal parallel speed-up in performance.

Regarding the computation part related to the IMU signal, the RMS envelope is computed on the last 512 samples instead of 32 since it offers a more efficient use of cores. A parallel version was implemented by splitting the signal again into four parts. Each core computes the summation of the square of the signal assigned, and finally, the cores are synchronized. In the last phase, a single core is in charge to sum the four results and to compute the square root.

5.4 *Neuro-PULP* results

Table 5.1 shows a set of programmable neural interfaces implementations. In [163] and [61], the MCUs are used to detect and calculate spikes' features. The simple MCUs have enough capabilities to handle 32 channels while running low complexity algorithms to extract features, but power consumption has not been optimized, and the systems are always-on operating in *streaming* mode. Besides, the simple architecture of such MCUs can handle a relatively low input bandwidth while processing, as explained in [133]. Therefore, they cannot be used with scaled systems, and more complex on-chip processing as their performance is limited. With respect to *Neuro-PULP* operating in *streaming* mode, they can handle a lower number of channels and less complex algorithms.

In [161] and [164], the acquisition and processing system are implemented in FPGAs. Such solutions offer higher scalability with better power consumption and processing capabilities. However, FPGAs require HW re-design for every change in the system from the protocol to interface with the AFE all the way to the algorithm to process data. Furthermore, when compared with our solution, they consume significantly higher power (x4) [161]. Finally, a higher number of channels and performance capabilities are shown in [162]. Such a system offers limited programmability as it requires low-level SW programming and higher power consumption (3x) when compared with the proposed work running in *streaming* mode. In addition, the acquisition system is implemented outside the MCU, and it has to be re-adapted for different neural-recording systems. *Neuro-PULP* relies on SPI to communicate with the neural-interface as implemented in commercial systems like [166]. It is scalable in performance from low complexity algorithms executed in the single-core domain up to more complex ones leveraging the multicore cluster accelerator, and it is ultra-low-power and scalable, as shown in Figure 5.3.

The event-based approach means the system power consumption is proportional to the level of spiking activity, consuming only 114 μW/channel when 10 spikes/s per channel are detected and *sorting* is executed to classify neural activities, allowing a battery lifetime >100h. The power proportionality of *Neuro-PULP* is further shown

Table 5.2
Number of cycles required to compute each function.

Kernel Func	Single Core ARM CMSIS	Pulp 1 Core	Pulp 4 Cores	ARM/ SCPulp	ARM/ 4CPulp	SCPulp/ 4CPulp
FFT RADIX8	42.90	64.03	16.69	0.67	2.57	3.84
Magnitude	17.87	15.17	3.83	1.18	4.66	3.96
RMS (32s)	0.26	0.30	0.16	0.86	1.67	1.94
RMS(512s)	2.52	2.95	0.83	0.86	3.05	3.56

in Figure 5.4, where two ideal *streaming* mode scenarios are compared with the event-based power consumption of the proposed system. Compared with the two ideal systems, the power consumption is lower, even in excess 10,000 spikes/s in total, which suggests that future high channel count systems should be event-based. In addition, new implementations of neural interface algorithms can be easily implemented in C code in *Neuro-PULP* thanks to the system's high versatility, opening the possibility for embedded C code spike sorting libraries. Finally, as the PULP performance has not yet been fully exploited, more computationally demanding algorithms than the one presented can be implemented, e.g., compression algorithms or feature extraction scenarios [107].

5.5 Drowsiness acceleration results

The PULP architecture has been taped-out in several technologies, the one used for the implementation of this task is the PULPv3, implemented in UTBB FD-SOI 28 nm technology. To estimate the power consumption of the architecture, data have been extracted from measurements on the PULPv3 silicon prototype and adapted to the configurations actually employed in the exploration (i.e., a 4-core architecture enhanced with floating-point units).

To evaluate the performance and energy consumption of the computing platforms adopted in the system (i.e., PULP and Cortex-M4-based MCUs), only the compute-intensive kernels, responsible for more than 99% of the overall computational load of the algorithm have been analyzed. Table 5.2 shows the number of cycles needed to perform the FFT, Magnitude, and RMS functions in both platforms, as well as a

comparison of the execution time of the different solutions. While the code running on the Cortex-M4 architecture relies on heavily optimized CMSIS libraries, the implementation on the PULP platform is based on an ANSI C implementation of the algorithms with OpenMP extensions for parallelization.

Nevertheless, although the ARM Cortex-M4 core performs slightly better with respect to the single-core PULP architecture for some of the kernels, a single-core PULP platform provides an almost 20% speed-up with respect to the Cortex-M4 for the Magnitude function, while relying on a fully flexible C implementation of the algorithm. The situation dramatically changes when executing the algorithms exploiting parallel processing over the four cores of the PULP platform. In this case, the execution time with respect to the Cortex-M4 processor reduces by up to 4.66x. It can be noted that for the kernels with high parallelism, like FFT and Magnitude, that account for more than 95% of the overall computational load during sequential execution, the speed-up is nearly ideal. The only function that is not easily parallelizable is the RMS 32s, due to a small dataset, and hence parallelism, but it has a negligible impact on the application's overall execution time.

An important factor to consider for the calculation of energy efficiency is the minimum latency required to achieve real-time constraints. Indeed, to avoid sample loss, all the signal processing must be finished in time no longer than 2ms. This constraint was taken into account to adjust the clock frequency of the PULP platform to compare the execution with minimal energy consumption. Table 5.3 shows the real-time frequency (RT Freq) of the analyzed computing platforms, which includes one high-end MCU STM32F407x and one ultra-low-power MCU Ambiq Apollo, both based on Cortex-M4 processor, and PULP executing on a single core and four cores. This task cannot be accomplished by a low-power MCU like Ambiq Apollo, due to its limited maximum operating frequency (24 MHz) as explained in [105]. The real-time frequency allows to select the minimum supply voltage needed to meet the performance constraints.

More interesting is the exploitation of parallel near-threshold computing on the PULP platform, leading to a further improvement of 3.4x in performance with respect to sequential processing, and improvement of 12.1x and 63.3x in terms of energy consumption with respect to commercial MCUs.

From an application perspective, these results show that the optimization of the parallel processing tailored for a highly efficient HW/SW platform allows extending the whole system's battery life to 46 hours, leading to an improvement of 7 times with respect to a solution based on a commercial MCU.

Table 5.3
Comparison between different platforms.

MCU	A. Apollo	STM32F407	1C PULP	4C PULP
No. of Cores	1	1	1	4
RT Freq [MHz]	31.68[1]	31.68	45.59	11.76
Vdd (V)	1.80	2.50	0.48	0.45
Pw Dens [µW/MHz]	115	600	10.27	27.64
Power [mW]	3.64	18.99	0.42	0.30
Energy[µJ]	7.28	37.97	0.84	0.59

[1] Ambiq Apollo does not achieve the required frequencies (i.e. max frequency is 24 MHz)

Chapter 6

Soft-Hardware Accelerators

In this Chapter, we present *Arnold*: a RISC-V based MCU extended with an eFPGA, implemented in Globalfoundries GF22FDX (GF22) technology. The contribution of the presented heterogeneous SoC design and silicon demonstrator with respect to the other Micro Controller Units (MCUs) augmented with an eFPGA are summarized as follows.

1. *Architectural Flexibility*: to enable architectural flexibility that fully exploits the configurable logic. The eFPGA is connected with the rest of the system with different interface options on the data-plane: *i*) a direct connection to the I/O Direct Memory Access (DMA) engine on the SoC - to process and filter data streams on their way from/to on-chip shared-memory buffers in memory; *ii*) a high-bandwidth, low-latency interface to the memory of the RISC-V core - to interleave with zero-copy FPGA-accelerated parallel processing and sequential processing by the core; *iii*) a direct GPIO interface to implement master or slave peripheral ports for non-standard off-chip digital sensors or actuators. On the control plane we provide: *i*) an AMBA Advanced Peripheral

Bus (APB) interface to allow the user to configure the mapped soft-hardware; *ii*) sixteen interrupts to notify the CPU.

2. *Power Management*: thanks to RBB enabled by conventional-well Fully Depleted Silicon-On-Insulator (FDSOI) technology used for the physical implementation of the eFPGA fabric, leakage power can be reduced by 18x to 20.5 µW (featuring a fully state retentive bitstream) when eFPGA functionality is not required.
3. *Leading Edge Performance and Energy Efficiency*: the System-On-Chip (SOC) achieves SoA performance and efficiency, leveraging a voltage and frequency scalable architecture from 0.5 V to 0.8 V, with a peak energy efficiency of 46.83 µW/MHz at 0.52 V and a maximum frequency of 600 MHz at 0.8 V. The proposed SoC achieves 3.4x better performance and 2.9x better energy efficiency than State-of-the-art (SOA) MCUs augmented with eFPGA built for the same power target applications [112, 115, 116].

Figure 5.1 shows the simplified architecture diagram of the *Arnold* SOC to underline the eFPGA connections with the rest of the system.

6.1 Arnold Architecture

The proposed system is built around the *Riscy* core presented in Chapter 2 augmented with an FPU. Differently from the *Quentin* MCU described in Chapter 3, we extended the CPU with a RISC-V compliant Physical Memory Protection (PMP) unit that can control read, write, and execute permissions on regions of the physical memory. This allows for protecting sensitive parts of the system from corrupted user applications. The implemented RISC-V PMP supports all address matching schemes as: naturally aligned power of 2 regions *NAPOT* (including 4 bytes alignment *NA4*); and the top boundary of an arbitrary range *TOR*. The PMP occupies only 14% of the total CPU area due to the extra registers and comparators needed to implement the specifications and provides much-needed security features for user-applications in the Internet-of-things (IoT) domain. In the proposed SoC, the CPU is responsible for executing the runtime to manage the

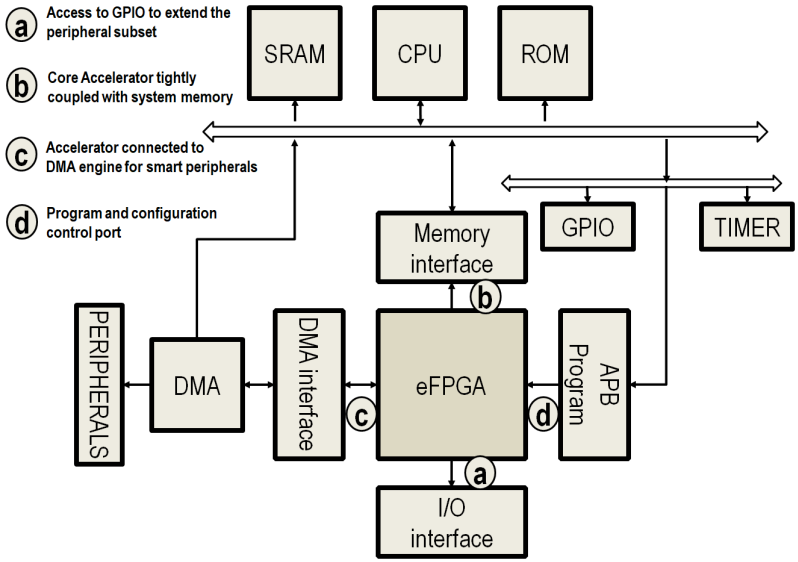


Figure 6.1: MCU-eFPGA SoC architecture. eFPGA connections towards the MCU and to the external peripherals are highlighted.

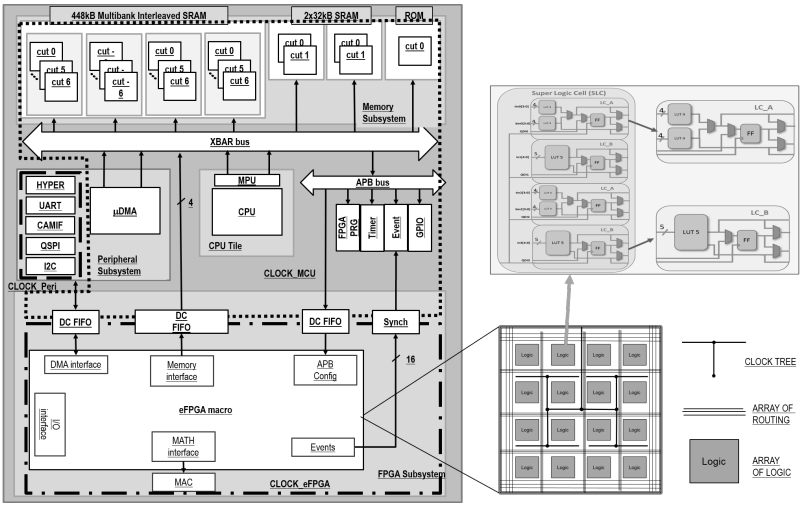


Figure 6.2: Detailed block diagram of the proposed design. The eFPGA (bottom) connected with the MCU and its private MAC units in a clock domain (CLOCK eFPGA). Peripherals (center – left) are directly connected to the µDMA in the Peripheral subsystem and operate on the CLOCK Peri clock domain. The rest of the system works in the CLOCK MCU domain. The CPU runs the SW and orchestrates the whole system.

system and execute user applications to process data or to control external peripherals and configure and control the eFPGA itself.

6.1.1 Memory Subsystem

The memory system, composed of 512 kB of Static Random Access Memory (SRAM), is shared among the CPU (instruction and data), the I/O DMA (μDMA) (RX and TX), the JTAG, and the eFPGA masters. The shared-memory consists of four word-level interleaved memory banks, each with 112 kB each, and two memory banks of 32 kB featuring a non-interleaved address scheme. Every memory bank is a composition of single-port 4096 by 32 bit words (16 kB) memory cuts optimized for density and power. The size chosen for the memory cuts allows to place them comfortably during the physical implementation as described below, and concurrently to meet the frequency target.

The chosen interleaving scheme for the four 112 kB (448 kB) memory portion approximates multi-port memory access, and it increases the bandwidth up to 4x when multiple masters are loading or storing data sequentially, which is the typical case for most Digital Signal Processing (DSP) applications. When low-latency single-cycle accesses with no contention are needed, the two private banks can be used to offer a bandwidth of 19.2 Gbps each. In the proposed MCU, they are used to store private CPU data such as the stack and instruction binary. In this way, the interleaved part can be used by the other masters with no conflicts. This solution avoids the use of power and area hungry multi-port memory cuts, still providing low-latency access to memory and increasing total energy efficiency. A Read-only-memory (ROM) has also been implemented to store the boot instructions responsible for setting the system upon reset.

6.1.2 I/O subsystem

The I/O subsystem is composed of a broad set of peripherals that include JTAG, HyperRam, UART, Camera Interface, quad-SPI, and I2C, which communicate with the shared-memory system through an autonomous µDMA based on [133]. The µDMA is a smart-engine that allows peripherals to control transfers to/from memory without the need for the CPU continuous control. The HyperRam peripheral is

particularly interesting as it allows access of off-chip memory with a bandwidth of 800 Mbps, extending the MCU with larger memory capacity, useful for holding several eFPGA bitstreams.

The µDMA has two ports towards the main memory, one to transmit and one to receive data from peripherals. At 600 MHz, the µDMA has an aggregated bandwidth equal to 38.4 Mbps. Except for the JTAG, which is directly connected to a master port of the system bus, the other peripherals are controlled by the µDMA core, which handles memory requests in a time-multiplexed fashion. The µDMA control registers are used to select the active peripheral, the peripheral clock frequency, number of transfers, etc. Other peripherals, such as SoC control registers, timers, GPIOs, and event units, are also included in the proposed MCU and accessible through the APB bus.

6.1.3 Clock subsystem

Arnold includes three Frequency-locked loops (FLLs) that take as input an external 32 kHz reference clock and provide internal clocks up to 2.1 GHz. One FLL each is used to provide the clock to the eFPGA, the peripheral subsystem, and the remaining modules as CPU, memories, busses, etc. The eFPGA has access to six clock sources: four from external GPIOs; one from the eFPGA FLL block; and one from an integer frequency divider from the same FLL.

6.1.4 eFPGA subsystem

The eFPGA is tightly coupled to the system to minimize the overhead of communications with the CPU. It has 3712 pins to be used to connect the IP with the rest of the SOC. In this work, we designed a novel, highly flexible 4-mode SOC interface to:

(a) an I/O interface with direct connections toward the pad frame of the system, enabling the implementation of custom off-chip interfaces;

(b) a memory interface suitable for shared-memory accelerators implemented on the FPGA logic and tightly coupled with the CPU;

(c) an I/O DMA interface suitable for implementing I/O filtering functions for data streamed into the system from the standard I/O;

(d) an APB configuration and control interface suitable for controlling the programmable logic.

The I/O interface is made of 41 sets of three signals (input, output, direction) from the eFPGA to the GPIOs. This interface is used for custom I/O protocols, which are challenging to implement efficiently in SW due to latency constraints. Each I/O pad can be either used by a peripheral (quad-SPI, Camera Interface, etc.), or by software (Core GPIO), or by the eFPGA. Multiplexers controlled by SoC registers drive the functionality mode of each pad.

The memory interface implements the protocol presented in [146]. The proposed SOC has four interfaces connected as master ports in the bus, providing up to 128 bit memory operations (load or store) per transaction. Access to the on-chip SRAM is provided through four 32 bit 4 words dual-clock FIFOs to allow the MCU and the eFPGA subsystem to operate at independent frequencies. This is a crucial feature since the eFPGA usually runs at a lower frequency than the rest of the SoC and its frequency depends on the user design. For security reasons, the eFPGA memory interface has only access to SRAM banks and not to APB peripherals and boot ROM.

The I/O DMA interface is composed of one receive (RX), and one transmit (TX) bus featuring a ready/valid handshaking, plus one 32 bit configuration bus as described in [133]. The configuration bus allows controlling the peripherals mapped into the eFPGA with external registers which can avoid the use of the APB interface described below, and thus save resources. In addition, this interface can be used to stream data through the µDMA without using eFPGA resources for the address generation logic as it would with the memory interface. In this case, the µDMA transfers data from the eFPGA to memory (and vice versa) linearly. Communication between the µDMA and the eFPGA happens using two 32 bit 4 words dual-clock FIFOs.

Designs mapped into the eFPGA (as accelerators or peripherals) can be controlled by registers through the APB configuration and control interface. Such an interface is made of a 7 bit address, 32 bit data read, and data write, write-enable, ready, peripheral select and

enable signals (75 pins). One 32 bit 4 words dual-clock FIFO is used for communications between the MCU and the eFPGA.

In addition to the four interfaces mentioned above, the eFPGA can generate sixteen events to interact asynchronously with the CPU, avoiding inefficient polling operations and saving power. In fact, the eFPGA event pins are connected to dual-clock event-propagators that notify the events to the CPU as dedicated interrupts requests. The interrupt service routines are user-defined, and they can be used to handle the eFPGA requests, for example, starting a new I/O transaction, or programming the new acquired data pointers to start processing them in case of accelerator design.

To improve computational arithmetic density, two synthesizable parallel-vectorial Multiply And Accumulate (MAC) accelerators are connected to the eFPGA to compute four 8 bit, two 16 bit, or one 32 bit MAC operations for each unit. The two MAC blocks are connected via 310 pins each, which control the MAC blocks, whether data comes from the eFPGA or the MAC buffers, the input and output data, and the vector mode (8, 16, or 32).

The CPU programs the eFPGA through another APB interface. Such a master interface is connected to the eFPGA Fabric Configuration Block (FCB), which is responsible for controlling the eFPGA, managing the power procedures, and report the actual status of the eFPGA. The eFPGA binary is 225.5 kB, small enough to be contained in the on-chip SRAM. To program the macro, the CPU reads the binary from external memory and writes it to the on-chip memory, then the CPU reads the binary array and writes its content to the APB FCB via non-critical load and store instructions.

The eFPGA fabric is organized in four quadrants with dynamic reconfiguration capabilities, each one composed of an array of 16x16 Super Logic Cells (SLCs). Each SLC has four logic cells that are organized in two sub-logic clusters: two instances of logic cell A (LCA) and two instances of logic cell B (LCB), as shown in Figure 6.2. Both LCA and LCB also include one register and multiple multiplexers that enable the logic cell to perform different functions (e.g., combinatorial, sequential, or both). If a logic cluster or a highway network within the SLC is not used, it is powered off to save static power. A shared register clock, set, and reset signals for all four logic cells helps reduce

routing congestion. If the logic cluster or highway network within the SLC is not used, it is powered off to save static power.

6.2 eFPGA Software and Tools

To use the eFPGA in the *Arnold* SoC, the user writes HDL code (VHDL, Verilog or SystemVerilog) and synthesizes it with Mentor Graphics Corporation © Precision RTL Synthesis OEM Quicklogic tool. The synthesized design is then placed and routed with the QuickLogic Aurora Software Tool Suite (Aurora). The user must map each of the soft-module interface pins to the corresponding pin of the eFPGA hard-macro. For example, the user may define the memory interface request signal as "MemREQ_output", in the Aurora tool, the user may specify that the signal is connected to the 3rd memory interface of the eFPGA specifying that "MemREQ_output" is connected to "tcdm_req_p3_o" pin. The eFPGA pin has been assigned to its interface functionality at SoC design time to optimize the place and route phase.

Once the constraints and the pin mapping have been defined, Aurora performs logic optimization on the synthesized design, places, and routes it. It also generates static timing analysis and the bitstream containing the binary of the user-design. The binary is then loaded into the main memory by the CPU. The CPU stores each binary word into the bitstream registers. Once the eFPGA has been programmed, the CPU can control the design with user-defined registers mapped into the eFPGA APB interface described above to start the design, to check the status, etc. application Programming Interfaces (APIs) have been developed to provide C procedures for the user. In particular, functions to RESET the eFPGA, to load the bitstream, and to wait for the end of the eFPGA computation (*wait_fpga_eoc*) have been implemented for fast integration into the user application. The *wait_fpga_eoc* routine leverages the WFI instruction to clock-gate the CPU to save dynamic power.

6.3 Arnold Physical Design

The proposed SOC fabricated in GF22 10 Metal technology occupies 3x3 mm^2. The synthesis tool used for this project is Synopsys © Design Compiler 2017.09, whereas the place and route tool used is Cadence © Innovus 18.11. The design has been closed at 430 MHz for the MCU side, and for up to 100 MHz for the eFPGA soft-designs. Worst-case conditions at 0.72 V for setup constraints, and best-case conditions at 0.88 V for hold constraints between -40°C and 125°C have been used to guarantee performance across the process, voltage, and temperature variations.

The die picture and floorplan of the chip are shown in Figure 6.3. The eFPGA macro is 2x2 mm^2, and it has been placed in the bottom left of the design. The memory cuts have been placed to the right of the eFPGA. The eFPGA memory interface pins have been assigned to the right part of the eFPGA to minimize routing efforts and to minimize the congestion issue as the path towards the memory is the most critical. The core has also been automatically placed close to the memory to minimize timing penalties. The eFPGA pins for the MAC blocks accelerators have been placed to the top part, where the local math accelerator SRAM buffers have been placed. On the left part of the eFPGA, the pins towards the µDMA, the user APB interface, and the 16 events pins have been assigned. GPIOs pins are spread along the four sides of the eFPGA. The six clock pins of the eFPGA are located three on the top and three on the bottom side. The three FLLs have been placed on the top part of the chip, whereas the standard cells have been automatically placed by the place and route tool.

The effective area occupied by the chip is 5.11mm^2, of which the eFPGA macro occupies 78% (4mm^2) and the MCU 22% (1.11mm^2). The main memory occupies 14.46% of the system area, whereas the I/O subsystem and the CPU take only 0.43% and 0.54%, respectively. The eFPGA subsystem components occupy 1.26% of the MCU area. The eFPGA subsystem is a set of modules that interact directly with the eFPGA macro, dual-clock FIFOs, the FCB, the MAC accelerators (including memory buffers), and clock multiplexing logic. Table 6.1 shows the area distribution of the chip.

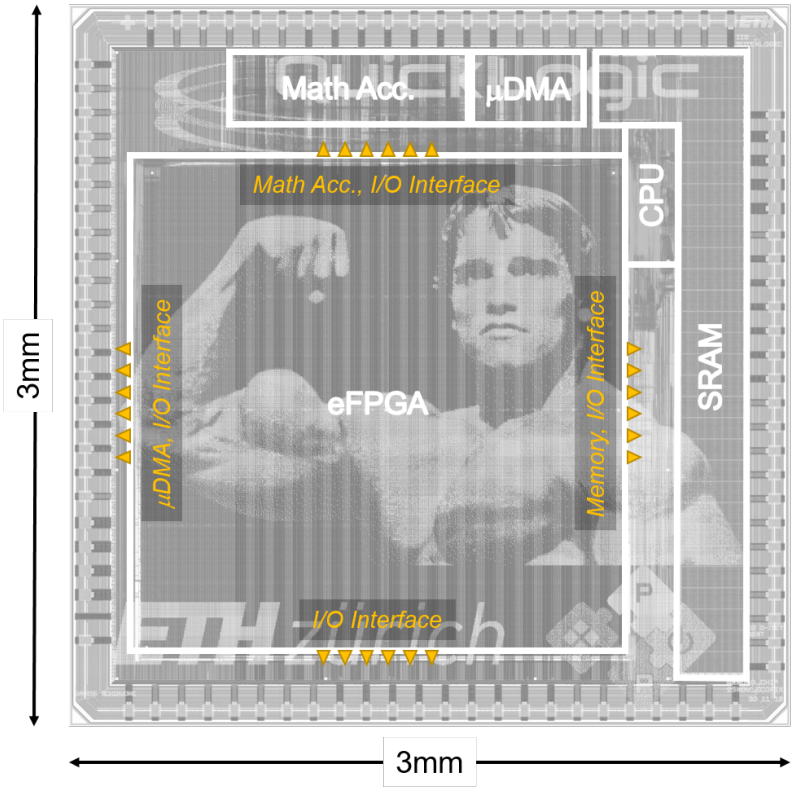


Figure 6.3: Die photo of the proposed design with the main components and eFPGA pins highlighted.

Table 6.1
Area distribution of the main components of Arnold.

Module	Area [µm^2]	Percentage
CPU	27'186	0.54%
Main Memory	734'232	14.46%
I/O DMA	21'755	0.43%
eFPGA subsystem	63'946	1.26%
PAD Frame	229'519	4.52%
eFPGA Macro	4'000'000	78.79%

The MCU and the eFPGA operate at the same supply voltage, but the eFPGA can be switched off from external power managers. The range of operation is between 0.5 V to 0.8 V. To reduce the leakage power while preserving the eFPGA configuration during state-retentive deep sleep states, RBB is applied from an external generator to minimize on-chip implementations overheads. On the other hand, FBB is applied to the CPU, memory, and the rest of the logic to increase performance [121, 122].

6.3.1 Performance and Energy Efficiency

In this subsection, measured results at room temperature from the implemented chip are reported and discussed. Performance and power results have been measured using an Advantest SoC V93000 ASIC tester. Figure 6.4 (left) shows the maximum frequency (a), power consumption (b), and power density (c) of the MCU during the execution of a matrix multiplication at different supply voltages. Measured results at ambient temperature show a maximum frequency of 135 MHz, and power consumption 11.88 µW/MHz at 0.49 V, up to a maximum of 600 MHz at the nominal 0.8 V while consuming 26.18 µW/MHz. The maximum frequency at 0.49 V is comparable with commercial single-core MCUs performance while achieving very low power consumption thanks to voltage scaling. When high performance is needed, 600 MOPS can be achieved at a maximum power consumption of 16 mW. The leakage power of the whole MCU ranges from 0.53 mW (33%) to 2.39 mW (15%) at 0.49 V and 0.8 V

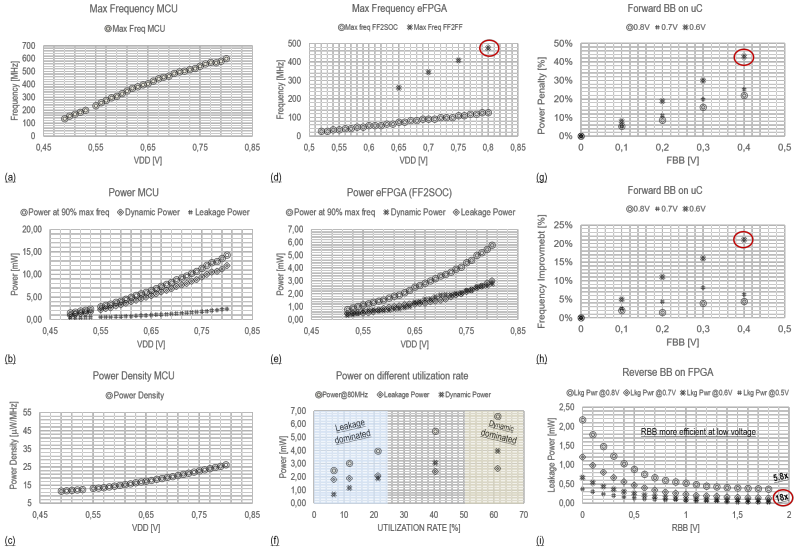


Figure 6.4: Frequency (a), power consumption (b), and energy-efficiency (c) with respect to the supply voltage of the MCU part of the proposed design. In the center, frequency (d) and power of the eFPGA macro with respect to the supply voltage (e) and power with respect to the utilization rate (f). The effect of the FBB on power (g) and frequency (h) on the MCU. The effect of RBB on the eFPGA leakage power during state-retentive deep-sleep mode (i).

respectively. Figure 6.4(g) shows the effect of the FBB on the MCU power consumption, and Figure 6.4(h) on the frequency. The MCU can run up to 20% faster at 0.6 V at the price of 43% higher power consumption, whereas the effect of FBB is smaller when applied at 0.8 V (only 5% faster) for a maximum frequency of 630 MHz.

Figure 6.4 (center) shows the eFPGA measured results. Figure 6.4(d) shows the maximum frequency of two different designs: *FF2SOC* is an eight-way parallel 32 bit accumulator that reads values from the SoC memory and accumulates them in eight different registers. The signature can be read with the APB interface; *FF2FF* is a nine bit counter that divides the eFPGA clock by 512 and drives

a GPIO with the divided clock. The designs are different as the *FF2SOC* communicates with synchronous elements in the SoC (dual-clock FIFOs). Thus its maximum frequency is bounded by the internal delays of the eFPGA and the logic outside its boundary, whereas *FF2FF* has been designed to measure only the flip-flop to flip-flop delay, without taking into account the propagation and setup timing of the eFPGA and the external logic at its boundary. The output of the Q-pin of the MSB flip-flop of the nine bit counter is directly connected to the GPIO, and the frequency is measured with an oscilloscope. From measurements, we determined a maximum frequency of 475 MHz at 0.8 V and 260 MHz at 0.65 V. *FF2SOC* occupies 15% of the internal eFPGA resources and it can run from 26.38 MHz, consuming 34.34 µW/MHz at 0.52 V, to 126.88 MHz at 0.8 V consuming 47.98 µW/MHz (Figure 6.4(e)).

The eFPGA *FF2SOC* leakage power is 0.38 mW at 0.5 V, up to 2.18 mW at 0.8 V. The power has been measured separately from the rest of the system as the power grid stripes of the eFPGA are different from the MCU ones. The power overhead added by the eFPGA is affordable in the IoT domain, making the integration of such programmable arrays a viable option for the next generation of edge-computing nodes. The eFPGA leakage power consumption is reduced via state-retentive deep sleep states applying RBB, resulting in a minimum leakage power of 20.5 µW at 0.5 V and 374.2 µW at 0.8 V and 1.8 V reverse body-bias as shown in Figure 6.4(i), i.e., a 5.8x(at 0.8 V) to 18x (at 0.5 V) reduction can be achieved thanks to RBB. This result makes the eFPGA power consumption significantly reduced when not used, minimizing the integration cost and overhead. Figure 6.4(f) shows how the power consumption changes with respect to the utilization rate. A design with a parametrizable number of adders has been implemented in the eFPGA to measure the power consumption with respect to the utilization rate. When running at 80 MHz, 0.75 V, results show an energy-efficiency of 0.40 µW/MHz/SLC, being leakage dominated when <20% of resources are utilized. The best energy-efficient point of the whole system is 46.83 µW/MHz (eFPGA consumes 28% of total power) achieved in near-threshold at 0.52 V, when the core and the eFPGA are running at 183.6 MHz and 26.38 MHz respectively. This result has been measured when the eight parallel 32 bit accumulators are mapped on the eFPGA.

Table 6.2
Performance comparison with state-of-the-art MCU and eFPGA systems.

	Borgatti [115]	**Lodi** [116]	**Renzini** [112]	**Fournaris** [42]	**Whatmough** [114]	**Bol** [71]	**This Work**
Technology [nm]	180	130	90	65	16	28	22
I$/D$/SRAM [kB]	8/8/48	8/8/256	-/-/32	8/-/656	2K[1]/-/4K	-/-/64	-/-/512
Voltage Range [V]	1.8	1.2	1.2	1.2	0.5 - 1.0	0.4 - 0.8	0.5 - 0.8
FPGA IP macro	Hard	Hard	Soft	Hard	Hard	-	Hard
FPGA Area [mm^2]	8.2	6.0	0.347	-	1.0	-	4.0
FPGA #LUT	15 kGE	15 kGE	96 5/4:2	12084 4:1 [2]	8800 6:2 [3]	-	6018 4:1
FPGA #FF	-	-	192	12084 [2]	22656 [4]	-	4096
FPGA #DSP	-	-	-	22 [5]	80 MACs [6]	-	2 vecMACs
Access Mode to SoC	GPIOs m/s mmap RX DMA	GPIOs s mmap TX/RX DMA	s mmap	GPIOs m/s mmap TX/RX DMA	m/s mmap	- - -	GPIOs m/s mmap TX/RX DMA
FPGA Lkg Power *	-	-	-	-	12000 [7]	-	20.5 - 2178
FPGA Max Freq.**	175	166	50	160	734	-	475
FPGA Power Density ***	-	-	34.72@1.2V [8]	962@1.2V [9]	-	-	31.98@0.6V [10]
MCU Lkg Power *	-	-	-	7000[11]	-	1 - 30	532 - 2386
MCU Max Freq. **	175	166	50	166	-	80	600
MCU Power Density ***	-	-	101.22@1.2V[8]	31@1.2V[12]	-	3@0.4V, 48MHz	11.88@0.49V, 135MHz
MCU+FPGA Power Density ***	-	1807.23@1.8V	135.94@1.2V [8]	993@1.2V [9,12]	-	-	46.83@0.52V [13]

* Power numbers are in µW ** Frequency numbers are in MHz *** Power density numbers are in µW/MHz
[1] Two 64 kB of L1 cache shared between Instructions and Data for each core, plus 2 MBytes of L2 cache.
[2] SmartFusion2 M2S010S data available in the product brief.
[3] 2520x2 LUTs for the two logic tile and 1088x2 for the two DSP tiles [167].
[4] 6304x2 flip-flops for two logic tile. 5024x2 for the two DSP tiles [167].
[5] Signed multiplication, dot product, and built-in addition, subtraction, and accumulation units.
[6] 40x2 MACs for the two DSP tile [167].
[7] 3 mW reported in the datasheet [167]. Assuming it is for a 1x1 tile, [114] uses a 2x2 tile, thus 12 mW have been reported in the Table.
[8] Average measurements.
[9] Estimated from [42]. It assumes the FPGA runs at 160 MHz. [10] When *FF2SOC* design is synthesized on the eFPGA [11] Includes FPGA leakage power as well.
[12] Number taken from [42]. The authors use the ARM Cortex-M3 power consumption from the datasheet reported in 90 nm LP.
[13] When *FF2SOC* design is synthesized and running on the eFPGA and the MCU is computing a matrix multiplication at the same time

6.4 Use Cases

To demonstrate the flexibility and efficiency of our heterogeneous reconfigurable SoC, three different use cases have been implemented, highlighting the versatility of embedded programmable logic.

Table 6.3
Resource utilization, power consumption and overall energy savings for implementing different use-cases on the eFPGA.

Use Case	GPIO	FF	LUT	Power [mW]	Energy Saving [x]
Custom I/O	36	205	289	6.0	2.5
BNN	0	854	1229	12.5	2.2
CRC	0	20	47	7.5	42.2

6.4.1 I/O subsystem accelerator

In the context of applications for bio-signal processing, it is common to extract features in the frequency domain to classify activities sensed from skeletal muscles or the brain [105]. Wavelet or Fourier transforms used to convert the signal from the time to the frequency domain, then features like the spectral power, are extracted and used by a pattern recognition algorithm. For this reason, a peripheral that extracts relevant information of the signal acquired from the sensors has been developed and mapped to the eFPGA to alleviate the pre-processing part of the CPU, which then classifies the activity starting from the extracted features. The peripheral accelerator mapped on the eFPGA consists of an SPI module extended with computational capabilities to calculate the Haar Discrete Wavelet (HDWT), an attractive algorithm to implement in an eFPGA as it does not require multipliers [168].

The accelerator is configured to acquire N samples of 16 bit of raw data coming from ADCs, and to store the Approximated and Detailed Wavelet Transform coefficients in the main memory. Also, coefficients can be stored in an 8 bit format to compress information in the main memory. The accelerator is programmed at the beginning with the number of samples to acquire and the output vector pointers. The

eFPGA autonomously loops over SPI transactions and stores to the main memory, either the raw data or the Approximated and Detailed coefficients of the HDWT. When all the N data have been stored into the memory, an interrupt notifies the core at the end of the acquisition.

Moreover, a second function has been mapped to the custom SPI peripheral, namely, to extract 4 bits local binary patterns from a stream of data coming from sensors, as an algorithmic approach presented in [169]. In this case, for each data acquired, the eFPGA reuses the subtractor instantiated for the HDWT to compare the last two samples. If the last sample is greater than the previous one, it stores 1 in a 4 bit shift register, otherwise 0. The accelerator stores into memory a 16 bit value every four samples, each representing four single sample overlapping windows. The core takes eight cycles for each tuple approximate-detail coefficient to compute the HDWT, whereas it takes 16 cycles for the local binary pattern. The eFPGA instead computes the features during the acquisition of the signal from SPI without adding latency overheads.

The design utilizes 20% of the available SLCs, and it uses a memory interface port, the APB interface, four GPIOs (3 output pins and 1 input pin), and it generates one event.

6.4.2 Custom I/O interface

IoT devices are often connected to custom peripherals that need more control pins that the usual peripherals as SPI, UART, I2C, I2S, etc. In this case, off-chip FPGAs are selected to implement the control part of the custom peripheral on one side and to communicate with the MCU with a standard protocol (e.g., SPI) to the other side. An example of a custom peripheral is a neuromorphic vision sensor [170] or event-based audition sensors [171]. Another example where FPGAs are used to control and transfer data are bridges for off-chip accelerators, for example, [172], or [173]. In this context, to illustrate the flexibility of the MCU +eFPGA combination, a controller for the systolic Long short-term memory Recurrent Neural Network (LSTM-RNN) accelerator presented in [172] has been implemented in the eFPGA. The LSTM-RNN accelerator is made of four chips implemented in UMCL 65 nm technology, and it is used to classify

phonemes in real-time. The eFPGA uses 36 GPIOs to interact with the accelerator using a custom interface.

In the first phase, the eFPGA sends the weights of the RNN-model into the four chips. Then, for every sample acquired by the MCU I/O subsystem, the CPU extracts the Mel-Frequency Cepstral Coefficients (MFCCs). In parallel, the eFPGA autonomously fetches the coefficients from the main memory of the MCU and sends them to the off-chip accelerator. Once the inference on the accelerator has been computed, the result is sent back to the eFPGA, which stores it to the main memory of the MCU and finally notifies the core with an interrupt. Figure 6.5 shows the data flow from the microphone to the accelerator and back to the MCU. The utilization of the eFPGA is only 10%. Managing 36 GPIOs through MCU firmware (of which one is the clock of the off-chip accelerator) would require the core to run at a higher frequency than the eFPGA due to the sequential nature of software. In this example, the external accelerator is running at 80 MHz. This means that in the best case, the CPU should be able to perform ~7 operations in 12.5 ns, which requires 560 MHz, and 2.5x higher energy consumption than the eFPGA based solution.

6.4.3 CPU subsystem accelerator

In the context of on-the-edge computation, accelerators are used to increase performance and the energy efficiency of such devices [174]. For pattern recognition tasks in the visual domain, deep quantized neural networks are an attractive model due to its limited memory and computational requirements [175]. In extreme cases, single-bit representation for weights and data is chosen to minimize the memory footprint and the computational resources, as it requires simple operations as logic XOR rather than multiplications to compute convolutions. Such neural networks are called Binary Neural Networks (BNN) [99, 154]. The eFPGA has sufficient resources to allow these accelerators to be implemented, freeing the core for other computing tasks.

The BNN accelerator designed for this scope has four interfaces towards the main memory to maximize the bandwidth, and it is a simplified version of the accelerator presented in [83]. It assumes that input layers and filters are organized as a 3D array (number of

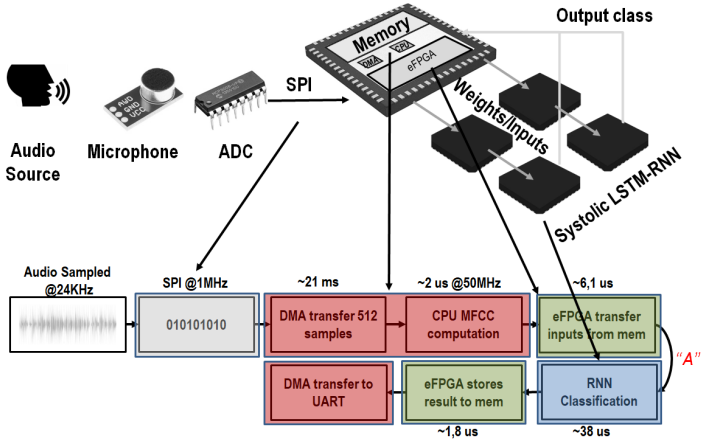


Figure 6.5: Example of an application where the proposed design is driving custom protocol off-chip accelerators. Data coming from microphones are first pre-processed by the MCU, then sent to the off-chip accelerator via eFPGA for classification.

filters x rows x columns) of integers, where each integer represents a 32 one-bit channels. The accelerator is implemented to operate on two 3x3 windows with eight filters f0, ..., f7 in parallel to simplify the controlling part, but this is not a limiting factor for the use-case under study. The accelerator is programmed via the APB interface by the core with the output, input and filter layer pointers, the number of rows and columns of the input layer, and the START command. The eFPGA starts by fetching two 32 bit input elements, then four 32 bit elements are fetched in parallel twice to acquire the eight filter elements.

The eFPGA performs the XOR function between the inputs and the eight filters, accumulates all the single-bit partial results. The sixteen 3x3 convolution results are then compared with a programmed threshold to compute the activation functions. The accelerator autonomously iterates over the input rows and columns; then, it sends an interrupt to the core to signal the end of the computation. During this period, the core can wait for the accelerator to finish in IDLE mode to save power or deal with other tasks in parallel (for example scheduling the next I/O tasks, elaborating previously filtered data, etc.). The design occupies 42% of the SLCs available, and it uses four memory interfaces, the APB port, and it generates one event. The application consumes 12.5 mW (eFPGA+MCU), and it runs in 371 µs at 125 MHz. Although the core implements custom instructions to speed up such kernels (as the pop count instruction), and it can run faster (600 MHz against 125 MHz), to implement the same function, the CPU consumes 15 mW, and it runs in 675 µs, with an energy efficiency 2.2x lower than the eFPGA.

As a second CPU accelerator, a cyclic redundancy check (CRC) accelerator has been implemented in the eFPGA to ensure data integrity and error correction [176]. Such an accelerator uses the I/O DMA interface to leverage the linear address generator already present in the μDMA and thus saving resources in the eFPGA. The CPU programs the μDMA to fetch data from the L2 memory and transmits them to the eFPGA accelerator, which calculates the CRC value. The accelerator has a register to know the number of data to process, whereas the read- and write-pointers are written in the μDMA configuration registers. This low area accelerator consumes only 2% of the SLCs available, and it only uses one interface towards the µDMA with configuration, TX/RX ports. The application consumes only

7.5 mW (eFPGA+MCU), and it runs in 3.7 µs at 193 MHz for 1024 byte data. The CPU consumes 15 mW, and it runs in 78 µs, with an energy efficiency 42.2x less than the eFPGA. To compare the performance of the proposed eFPGA-based system with respect to the Microsemi PolarFire IoT gateway-class FPGA SoC [33], the power estimator from Microsemi has been used. Results show a power consumption of 111 mW, 14.8x higher than our work. The estimation has been performed setting the same frequency, number of LUTs and flip-flops.

Table 6.3 shows the number of GPIOs, the number of flip-flops (FF), and LUTs required by each use case. Power figures (expressed in mW) correspond to the system when the eFPGA runs, and the CPU waits for the result, whereas the final column shows the energy gained by running the accelerator on the eFPGA rather than software. In the Custom I/O example, the SW could not handle the protocol at speed required. For that example, eFPGA was the only viable solution.

Basic interfaces like I2C and UART have been implemented on the eFPGA using the DMA interface with about 5% of eFPGA resources, and a more complex parallel camera interface with full DMA support implementation uses only 12% of available eFPGA resources.

Comparison with SoA

Table 6.2 shows a comparison with various chips reported in the literature. The table includes heterogeneous reconfigurable systems composed of MCU and eFPGA, an embedded domain FPGA SoC, and an advanced low-power MCUs in 28 nm FDSOI. The standalone MCU [71] has a 4x smaller power density (µW/MHz). However, our MCU features 8x larger memory capacity and significantly larger peak performance as well: 7.5x higher maximum frequency, 3.19 vs. 2.33 Coremark/MHz, and almost 6x better performance in near-sensor processing workloads when compared to the ARM Cortex-M0 processor used in [71]. Hence, our energy efficiency on the targeted application domain is 1.5x better.

The advanced MCU +eFPGA system presented in [114] is a high-performance class system implemented in 25 mm^2, where a bigger eFPGA (6x higher leakage power), two application class 64 bit cores, a quad-core cluster accelerator, and 12x bigger memory are used

(including caches). The eFPGA offers 80 MACs blocks, more LUTs, and eFPGA flip-flops, and provides remarkable energy efficiency of 312 GOPS/W. Thanks to the abundance of DSP blocks in the FPGA fabric. However, this system is meant to be used in high-performance applications consuming higher dynamic and leakage power not suitable for IoT applications. On the other hand, although achieving a lower peak efficiency, Arnold is in a power range suitable for IoT applications (below hundreds of mW). Moreover, the reverse body biasing applied to the FPGA fabric can reduce leakage power to a value as low as 20.5 µW, more than two orders of magnitude better than [114]. The Microsemi SmartFusion2 SoC [39] used in [42] is built in 65 nm. The whole system can run up to 160 MHz (> 3.75x slower than the proposed work), and it achieves 21x higher power density. The works of Borgatti [115] and Lodi [116] exploit embedded reconfigurable datapaths to accelerate DSP patterns of signal processing applications, achieving remarkable performance and operating frequency despite the old nodes used for implementation. With respect to these works and the other heterogeneous MCU +eFPGA systems of the same class [112, 115, 116], the proposed SoC has more than 2.9x better efficiency, more than 3.4x better performance, and more than 2.2x larger capacity. Moreover, this is the first design offering flexible connections enabling reconfigurable peripherals, I/O accelerators, shared-memory accelerators, and supporting state-retentive deep sleep based on reverse body bias, paving the way for flexible fully programmable IoT end-nodes.

Chapter 7

Summary and Conclusion

Edge-computing nodes of the Internet-of-things (IoT) are electronic devices that acquire data from sensors, process them, and send the result to the network. As they are typically battery-powered, they are constrained by peak power consumption and battery lifetime. In addition, quality-of-services, such as accuracy of the result, and the application performance requirements, are further constraining these devices to process billions of operations in a short time. To make everything more challenging, programmable devices are desirable for a shorter time-to-market and a longer product lifetime. Micro Controller Units (MCUs) are particularly interesting as the applications are described in software, and the usually large available set of peripherals allows for many sensors to be connected to the MCU. Although highly versatile, software-based solutions are usually less performant and efficient than hardwired fixed function circuits. However, in this Thesis, we exploited different optimizations to close the gap between high-versatile and high-specialized IoT nodes.

We show implementations results and variables tuned to meet Power-Performance-Area (PPA) and energy efficiency requirements for a large set of IoT applications.

As programmable devices are Central Processing Unit (CPU)-centric, application domain-driven optimizations to make the core faster, consume less power, and more energy-efficient, directly transfer their gains at system level. In particular, open-source Instruction Set Architecture (ISA) CPUs, such as RISC-V, are becoming popular thanks to their free costs and the possibility of extending the ISA. In this Thesis, different optimizations have been applied to RISC-V based CPUs to increase the energy efficiency, and improve the PPA metric, leveraging Digital Signal Processing (DSP) and pSIMD ISA extensions and area optimizations. To lower the power consumption and increase energy efficiency, different optimizations should also be applied at MCU and implementation level. In particular, Fully Depleted Silicon-On-Insulator (FDSOI) technology allows to achieve low-power, high energy efficiency results at Near-Threshold Computing (NTC), as well as to operate in a wide voltage range to exploit Dynamic-Voltage-Frequency Scaling (DVFS). Also, body-biasing can be leveraged to trade performance and power consumption. Different power-states can also be exploited to minimize the power consumption during idle modes and to trade memory capacity with energy efficiency in duty-cycled applications. In this Thesis, we implemented MCUs that exploit NTC, DVFS, FBB, and heterogeneous power-independent memory architectures to enable beyond State-of-the-art (SOA) energy efficiency and performant results in limited power budget. To further increase the MCU performance, accelerators can be tightly integrated to specialize the edge-node on a particular application or domain of applications. Hardwired fixed-function accelerators are circuits specialized for a particular application. Such accelerators can be tightly integrated into the MCU by sharing the on-chip memory with the main CPU. Clock-gating and/or power-gating techniques can be exploited to minimize the accelerator overhead when not used. NTC and as well as DVFS can be exploited to achieve higher energy efficiency during active periods. In this Thesis, we extended a Convolutional Neural Network (CNN) accelerator with SIMD operations to reach higher energy-efficiency. We integrated it in a MultiProcessor SoC (MPSoC) with four DSP-enhanced cores. We evaluated its energy efficiency and performance against them, showing that a tightly-coupled accelerator can be implemented in edge-devices in an ultra-low-power budget, still outperforming multiple cores running the same application. On

the other side, software accelerators are implemented as a more performant CPU than the main one, or a cluster of multicores. Such accelerators are optimized for a particular domain of applications. In particular, data processing algorithms are usually highly parallelizable by executing the same code on different CPUs on different data. Thus, shared instruction caches are usually preferred. Data sharing is also important to minimize costs. Thus, data caches are often missing in such low-power clusters. These heterogeneous systems allow splitting the application part on the cores that best meet performance and energy consumption metrics. In this Thesis, we exploit an MCU coupled with an event-driven high-dimensional Analog-to-Digital converter (ADC) to process brain Action Potential (AP) signals. The MCU exploits a single-core when pre-processed data from ADC triggers the computation, whereas it exploits a cluster of eight cores when running in streaming mode. The MCU power-states allow for efficient duty cycling to save as much power as possible. In addition, a cluster of four cores is exploited at NTC to efficiently process the extractions of features in Electroencephalography (EEG) signals, showing that the same architecture can be used efficiently in different applications. We show that such accelerators can be used in applications domain (such as the bio-physical one) where the broad of algorithms and the different nature of data need high versatile systems, still in the limited power domain of edge-devices. We show that our work outperforms MCU-based SOA related solutions in performance, energy efficiency, and maximum input signal bandwidth. Finally, eFPGAs sit between hardwired and software accelerators. They provide high versatility though soft-hardware design, but at the same time, they exploit the parallel nature of hardware execution instead of the sequential software one. eFPGAs enable post-silicon specializations of MCUs and longer product lifetime. They can be used to implement custom peripherals or accelerators. eFPGAs can be tightly coupled in MCUs though different on-chip busses to enable architectural flexibility. In addition, DVFS, clock-gating, and power-gating can be exploited to reduce the eFPGA power overhead. Furthermore, RBB can be exploited to reduce leakage power by keeping the state of the eFPGA. In this Thesis, we implemented an MCU augmented with an eFPGA in Globalfoundries GF22FDX (GF22) technology. The eFPGA is tightly-coupled in the system by sharing the on-chip memory with the

CPU. It is connected to the GPIOs as well as to the I/O subsystem for accelerating streams of data. The MCU exploits NTC and FBB to achieve higher performance on the MCU, and RBB to achieve lower power consumption on the eFPGA. We show that it is possible to embed an eFPGA in a MCU within a low-power budget typical of edge-computing devices. The MCU achieves beyond SOA performance and energy-efficiency of related solutions.

7.1 Overview and main results

Following the most relevant results and contributions of this Thesis are reported.

7.1.1 RISC-V CPUs

In Chapter 2, ISA extensions and selection, and an analysis of three different cores optimized for three different tasks have been carried out to show: ISA extensions for edge-computing devices running data-analytic applications and their implementation on a RISC-V core; area optimizations for cores that run mostly light processing or control-code in always-on domains; how the energy consumption changes among the core micro-architectures, workloads, timing-constraints, operating frequency, and voltage.

The core with DSP instructions (*Riscy*) is the most energy-efficient in data-intensive kernels as it has a highly specialized datapath tailored for computations of edge-computing applications. Such ISA extensions increased the core area by 14% for 3.5x higher performance and 3.2x better energy efficiency on DSP kernels. On DSP applications, *Riscy* is 2.4× more energy-efficient than an area optimized core.

Zero-riscy is the most efficient in arithmetic-control mixed and *Micro-riscy* in pure control code thanks to the minimal resources implemented in for the targeted ISA, leveraging multi-cycles instructions to exploit resource sharing in a time-multiplexed fashion. Even if *Zero-riscy* is 30% slower than *Riscy* on arithmetic-control mixed, the lower power consumption compensate the energy efficiency. Whereas, when no data-processing is needed, the smallest core is the best energy efficient core as power consumption is the lowest.

We also showed that tighter timing constrained netlists can achieve high frequency for a limited energy overhead (∼20%), but they are energy-inefficient at low frequency, due to the huge contribution of leakage to the total consumption (>10x). The same results can be obtained at super-threshold and near-threshold operating points, where the cores consume, on average, 3x less energy. We have shown that for systems that operate in an always-on power domain and execute a task as a consequence of a rare event, the leakage power is the most significant contribution to the energy consumption, hence area optimization is crucial. Thus an analysis a priori of the application profiles running on the edge-computing device should be carried out to select the best CPU micro-architecture and ISA together with the frequency-voltage operating points.

Finally, we proposed an evolutionary-based methodology that was able to generate assembly programs automatically, and we applied it to enhance the verification level of the *Riscy* processor. The proposed methodology combines the use of an evolutionary optimizer, a hardware perturbation module, and a checking mechanism to create verification sets of assembly programs rapidly. The experimental results demonstrated the effectiveness of the method by uncovering ten bugs in the RTL description of the Device Under Verification (DUV).

7.1.2 FDSOI MPU

In Chapter 3, we implemented a RISC-V based MCU that hosts a DSP-enhanced CPU, it exploits a heterogeneous memory subsystem, and it embeds a fixed-function Binary Neural-Network (BNN) accelerator. Such MCU has been implemented in GF22 FDSOI technology. It has three different operational modes to trade memory capacity and energy efficiency in active mode or optimize power consumption in idle mode in a duty-cycled scenario by exploiting a heterogeneous memory subsystem made of 504 kB, and 16 kB of Standard Cell based Memorys (SCMs).

Thanks to the FDSOI and FBB, the MCU can leverage a wide voltage operating range to enable DVFS and NTC from 0.5 V, where it consumes only 0.95 mW running at 156 MHz, to 938 MHz at 0.8 V with FBB applied at 1.4 V.

The MCU is the most performant among competitors as it achieves up to 2400 million equivalent RV32IMC MOPS (3.7x better than SOA) and achieves the best energy efficiency of 483 MOPS/mW for tiny applications that fit in the SCMs (2.3x better than SOA).

When error-tolerant applications such as BNNs are executed on the accelerator, aggressive voltage scaling can be leveraged to trade accuracy and power consumption in tight power-constrained or always-on scenarios. In this condition, the system exploits the BNN accelerator to achieve 13 binary ops per pJ while keeping a peak power envelope of 674 μW.

We showed that open-source microcontroller architectures implemented in advanced technology nodes could achieve top performance and energy efficiency to cope with IoT requirements thanks to the compound of CPU performance, MCU architecture, and technology that can exploit efficiently DVFS and FBB.

7.1.3 Fixed-Function Accelerators

In this Chapter, we presented pSIMD extensions to a fixed-function accelerator that performs 2D-convolutions. The accelerator has been extended to support four parallel 16x4 bit dot-products, two parallel 16x8 bit dot-products, and 16x16 bit multiply-and-accumulate operation. The accelerator's datapath has been designed to optimize area by sharing resources through sub-modules and saving power by extensively applying clock-gating policies. The accelerator is tightly coupled with CPUs for efficient synchronizations and data exchange between processing elements via the shared L1 memory, and no copy at all is required - only a simple pointer exchange. The accelerator has been integrated into *Fulmine*. A 65 nm SoC targeting the emerging class of smart, secure near-sensor data analytics for IoT end-nodes presented in [124]. We showed the accelerator performance and energy efficiency gains from 0.8 to 1.2 V with respect to full-precision support (i.e., 16 bit weights), and with respect to optimized four-core software implementations. Results showed that the accelerator is 82x faster than a single, DSP-enhanced core, and 11x faster than four DSP-enhanced cores, thanks to specialized architecture. When using the reduced 8/4 bit format, the HWCE is 1.8x/2.4x faster. The HWCE can work at near-threshold (0.8 V), achieving 465 GMAC/s/W for

a 5x5 2D-Convolution, 26x more energy-efficient than a cluster of four DSP-enhanced cores.

7.1.4 Software Accelerators

In Chapter 5, we presented *Neuro-PULP*, a power efficient and spike rate proportional, high performance, scalable and versatile event-based system for next generation neural interfaces that deal with: *i)* high-frequency, high-dimension brain APs to perform spike detection and compression; and *ii)* low-frequency, low-dimension EEG signals to extract complex features to perform drowsiness detection.

The system is composed of an MCU implemented in TSMC 40nm based on [64], which leverages a software accelerator made of 8 DSP-enhanced cores, different power states to enable efficient duty cycling, and it connected via SPI to an event-based 64-channel Analog front-end (AFE) [72].

The system offers high flexibility from *streaming* to *event* mode. In *streaming* mode, 64 channels are continuously acquired and compressed by the software accelerator in 8576 μs consuming only 690 μW/channel (or consuming only 478 μW/channel in 16750 ns for the optimized version). Whereas in *event* mode, events are triggered for every detected spike by the ADC and sorted by the single-core in the MCU in an average power budget of 114 μW/channel, outperforming SoA systems by 4x.

We show that thanks to the software accelerator and power states, SOA MCU-based implementations of neuro-applications are possible. We showed that the event-based approach leads to a 4x average power reduction with respect to streaming-based approaches (as the one in [161]) in a typical spike sorting application where 10 spikes/s/channel are detected on average. In addition, the proposed system can execute more complex algorithms when operating in *streaming* mode on a higher number of channels with respect to MCU-based SOA implementations [61] in a limited power budged of 690 μW/channel, consuming 3x less power than SOA DSP-based solutions [162], opening the possibility to implement even more computational demanding algorithms [107] leveraging the software-programmability and the high-performance and energy efficiency of the Parallel Ultra Low Power Platform (PULP) cluster.

Finally, we described how a software accelerator implemented in ST FDSOI 28nm technology working in near-threshold could boost energy efficiency on applications that target lower bandwidth signals like the EEG for drowsiness detection. The system implemented on the PULPv3 MCU, which employs a software accelerator made of 4 DSP-enhanced cores, achieves 63.3 higher energy efficiency.

7.1.5 Soft-Hardware Accelerators

In Chapter 6, we presented a RISC-V based MCU extended with an soft-hardware accelerator (eFPGA) for flexible power-constrained energy-efficient IoT devices. The system has built-in GF22, it occupies 9 mm^2, and it leverages body bias to trade performance and power. The eFPGA is a 32x32 array macro provided by QuickLogic connected to the rest of the system through four parallel memory interfaces (128 bit per transaction); a TX/RX Input/Output (IO) Direct Memory Access (DMA) interface; sixteen events to interact with the CPU; GPIOs; and APB. The system shows how the eFPGA can be used to extend and accelerate the SoC peripheral subsystem, as well as a CPU accelerator. The eFPGA has more than 6K LUTs and 4K flip-flops, enough to implement standard and custom peripherals used in the IoT domain and simple accelerators to enhance the energy efficiency of the SoC. It achieves 46.83 µW/MHz, top in class in the mW domain of IoT devices. The CPU runs up to 600 MHz (620 with FBB), more than 7x faster than the best energy efficient MCU. Leakage power of the whole system can be as low as 552 µW when the MCU runs at 0.5 V, and the eFPGA is kept in state retentive deep-sleep via RBB. We show that integrating an eFPGA in an MCU in GF22 gives to IoT devices the high versatility needed for extended product life and shorter time-to-market, still without waiving performance, power, and energy efficiency. We evaluated the system on a wide range of supply voltage to evaluate DVFS and NTC, showing that the MCU achieves 3.4x better performance and 2.9x better energy efficiency than other fabricated heterogeneous reconfigurable System-On-Chips (SOCs) of the same class.

7.2 Outlook

Following, a view regarding possible future works is provided.

Hardwired fixed-function accelerators are leading in performance and energy efficiency. However, they lack versatility. A possible alternative is exploring highly specialized application-specific instruction-set processors. One possible architecture may consist of a small instruction-fetch unit that feeds big datapath circuits to accelerate some specific kernels as 2D-convolutions or dot-products. Such instructions should be application-specific and should fuse many operations in a single-word to minimize software penalties. In the XNE accelerator presented in Chapter 3, the *Zero-riscy* core has been used to replace the controller of the accelerator to provide post-silicon fixes and programmability. Results have still to be measured.

For what concerns the bio-applications domain, multiple SPI modules can be used to increase the number of channels that the MCU can deal with. Also, computation capabilities can be integrated into the SPI modules on-the-fly to increase performance. A PULPissimo chip with eight SPI modules has been taped-out in 65nm. Results have still to be measured.

Finally, a complete PULP architecture with fixed-function, software, and soft-hardware accelerator can be built to enable maximum efficiency and maximum flexibility. Such SoC should be designed with different power states and RBB to minimize leakage power overheads, as well as exploiting DVFS and NTC when possible.

Appendix A

Notation and Acronyms

Acronyms

ADC	Analog-to-Digital converter
AFE	Analog front-end
ALU	Arithmetic and Logic Unit
AP	Action Potential
APB	AMBA Advanced Peripheral Bus
API	application Programming Interface
ASIC	Application-Specific Integrated Circuit
BNN	Binary Neural-Network
CMOS	Complementary Metal-Oxide Semiconductor
CNN	Convolutional Neural Network
CPU	Central Processing Unit

DMA	Direct Memory Access
DSP	Digital Signal Processing
DUV	Device Under Verification
DVFS	Dynamic-Voltage-Frequency Scaling
DWT	Discrete Wavelet Transform
ECG	Electrocardiography
EEG	Electroencephalography
EMG	Electromyography
EX	Execution
FBB	Forward Body-Biasing
RBB	Reverse Body-Biasing
FCB	Fabric Configuration Block
FDSOI	Fully Depleted Silicon-On-Insulator
FFT	fast Fourier transform
FIR	Finite-Impulse-Response
FLL	Frequency-locked loop
FPGA	field-programmable gate array
FPU	Floating Point Unit
FSM	Finite State Machine
GF22	Globalfoundries GF22FDX
GPIO	General-purpose Input/Output
HW	Hardware
ID	Instruction Decode
IDE	Instruction Decode and Execute
IF	Instruction Fetch
IO	Input/Output
IoT	Internet-of-things
IPC	Instruction Per Cycle

ISA	Instruction Set Architecture
ISS	Instruction-Set Simulator
LFP	Local Field Potential
MAC	Multiply And Accumulate
MCU	Micro Controller Unit
MPSoC	MultiProcessor SoC
NT	Near-Threshold
NTC	Near-Threshold Computing
OTS	Off-the-Shelf
PMP	Physical Memory Protection
PPA	Power-Performance-Area
PULP	Parallel Ultra Low Power Platform
ROM	Read-only-memory
SCM	Standard Cell based Memory
SIMD	Single Instruction Multiple Data
SOA	State-of-the-art
SOC	System-On-Chip
SRAM	Static Random Access Memory
SVM	Support-Vector-Machine
SW	Software
WB	Write Back

Bibliography

[1] V. Shnayder, M. Hempstead, B.-R. Chen, G. W. Allen, and M. Welsh, "Simulating the power consumption of large-scale sensor network applications," in *Proc. 2nd Int. Conf. Embed. networked Sens. Syst. - SenSys '04*, 2004, pp. 188–200. [webpage]: http://portal.acm.org/citation.cfm?doid=1031495.1031518

[2] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog computing and its role in the internet of things," in *Proceedings of the first edition of the MCC workshop on Mobile cloud computing*, ser. MCC '12. New York, NY, USA: ACM, 2012, pp. 13–16. [webpage]: http://doi.acm.org/10.1145/2342509.2342513

[3] E. F. Nakamura, A. a. F. Loureiro, and A. C. Frery, "Information fusion for wireless sensor networks," *ACM Comput. Surv.*, vol. 39, no. 3, p. 9, 2007. [webpage]: http://portal.acm.org/citation.cfm?doid=1267070.1267073

[4] M. Alioto, *Enabling the Internet of Things: From Integrated Circuits to Integrated Systems*. Springer, 2017.

[5] D. Rossi, C. Mucci, M. Pizzotti, L. Perugini, R. Canegallo, and R. Guerrieri, "Multicore signal processing platform with heterogeneous configurable hardware accelerators," *IEEE Transactions*

on Very Large Scale Integration (VLSI) Systems, vol. 22, no. 9, pp. 1990–2003, 2013.

[6] G. Desoli *et al.*, "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.

[7] L. Cavigelli and L. Benini, "Origami: A 803 GOp/s/W Convolutional Network Accelerator," *arXiv:1512.04295 [cs]*, Dec. 2015.

[8] R. Andri, L. Cavigelli, D. Rossi, and L. Benini, "YodaNN: An Architecture for Ultra-Low Power Binary-Weight CNN Acceleration," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. PP, no. 99, pp. 1–1, 2017.

[9] K. Ando *et al.*, "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, Apr. 2018.

[10] 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.

[11] 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*.

[12] W.-S. Khwa *et al.*, "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*.

[13] A. Jafari, A. Ganesan, C. S. K. Thalisetty, V. Sivasubramanian, T. Oates, and T. Mohsenin, "Sensornet: A scalable and low-power deep convolutional neural network for multimodal data classification," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 66, no. 1, pp. 274–287, 2019.

[14] X. Yu *et al.*, "A data-center fpga acceleration platform for convolutional neural networks," in *2019 29th International Conference on Field Programmable Logic and Applications (FPL)*, Sep. 2019, pp. 151–158.

[15] A. Ibrahim and M. Valle, "Real-time embedded machine learning for tensorial tactile data processing," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 65, no. 11, pp. 3897–3906, 2018.

[16] H. Chen, S. Madaminov, M. Ferdman, and P. Milder, "Sorting large data sets with fpga-accelerated samplesort," in *2019 IEEE 27th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2019, pp. 326–326.

[17] A. G. Sawant, V. N. Nitnaware, and A. A. Deshpande, "Spartan-6 fpga implementation of aes algorithm," in *ICCCE 2019*. Springer, 2020, pp. 205–211.

[18] J. Liao *et al.*, "Fpga implementation of a kalman-based motion estimator for levitated nanoparticles," *IEEE Transactions on Instrumentation and Measurement*, vol. 68, no. 7, pp. 2374–2386, Jul. 2019.

[19] H. Homulle, S. Visser, and E. Charbon, "A cryogenic 1 gsa/s, soft-core fpga adc for quantum computing applications," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 63, no. 11, pp. 1854–1865, 2016.

[20] A. Jafari, N. Buswell, M. Ghovanloo, and T. Mohsenin, "A low-power wearable stand-alone tongue drive system for people with severe disabilities," *IEEE Transactions on Biomedical Circuits and Systems*, vol. 12, no. 1, pp. 58–67, Feb. 2018.

[21] P. Anagnostou *et al.*, "Torpor: A power-aware hw scheduler for energy harvesting iot socs," in *2018 28th International Symposium on Power and Timing Modeling, Optimization and Simulation (PATMOS)*, Jul. 2018, pp. 54–61.

[22] V. Rosello, J. Portilla, and T. Riesgo, "Ultra low power fpga-based architecture for wake-up radio in wireless sensor networks," in *IECON 2011 - 37th Annual Conference of the IEEE Industrial Electronics Society*, Nov. 2011, pp. 3826–3831.

[23] I. Williams, S. Luan, A. Jackson, and T. G. Constandinou, "Live demonstration: A scalable 32-channel neural recording and real-time fpga based spike sorting system," in *2015 IEEE Biomedical Circuits and Systems Conference (BioCAS)*, Oct. 2015, pp. 1–5.

[24] Xilinx: Virtex Ultrascale+. [webpage]: Availablewebpageathttps://www.xilinx.com

[25] Intel Cyclone 10 GX Device Overview. [webpage]: Availablewebpageathttps://www.intel.com

[26] H.-C. Ng, C. Liu, and H. K.-H. So, "A soft processor overlay with tightly-coupled fpga accelerator," *arXiv preprint arXiv:1606.06483*, 2016.

[27] C. Heinz, Y. Lavan, J. Hofmann, and A. Koch, "A catalog and in-hardware evaluation of open-source drop-in compatible risc-v softcore processors," in *IEEE Proc. International Conference on ReConFigurable Computing and FPGAs (ReConFig). IEEE*, 2019.

[28] R. Höller, D. Haselberger, D. Ballek, P. Rössler, M. Krapfenbauer, and M. Linauer, "Open-source risc-v processor ip cores for fpgas—overview and evaluation," in *2019 8th Mediterranean Conference on Embedded Computing (MECO)*. IEEE, 2019, pp. 1–6.

[29] Y. Choi, K. You, J. Choi, and W. Sung, "A real-time fpga-based 20000-word speech recognizer with optimized dram access," *IEEE*

Transactions on Circuits and Systems I: Regular Papers, vol. 57, no. 8, pp. 2119–2131, 2010.

[30] A. Lindoso, L. Entrena, M. García-Valderas, and L. Parra, "A hybrid fault-tolerant leon3 soft core processor implemented in low-end sram fpga," *IEEE Transactions on Nuclear Science*, vol. 64, no. 1, pp. 374–381, Jan. 2017.

[31] Xilinx: Zynq-7000 SoC. [webpage]: Xilinxds190-Zynq-7000datasheetavailablewebpage

[32] Intel: Arria V SOC FPGAS. [webpage]: IntelArriaVSOCFPGAdatasheetavailablewebpage

[33] PolarFire SoC Advance Product Overview . [webpage]: https://www.microsemi.com/product-directory/soc-fpgas/5498-polarfire-soc-fpga#resources

[34] B. Pandey *et al.*, "Performance evaluation of fir filter after implementation on different fpga and soc and its utilization in communication and network," *Wireless Personal Communications*, vol. 95, no. 2, pp. 375–389, 2017.

[35] W. Qiao, Z. Fang, M. F. Chang, and J. Cong, "An fpga-based bwt accelerator for bzip2 data compression," in *2019 IEEE 27th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, Apr. 2019, pp. 96–99.

[36] Microsemi IGLOO nano Low Power Flash FPGAs datasheet. [webpage]: Availablewebpageathttps://www.microsemi.com

[37] Lattice Semiconductor iCE40 UltraLite Family Data Sheet. [webpage]: Availablewebpageathttp://www.latticesemi.com

[38] 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)*, Jun. 2017, pp. 1–6.

[39] SmartFusion: SmartFusion2 SoC. [webpage]: https://www.microsemi.com/product-directory/soc-fpgas/1692-smartfusion2

[40] Microsemi: Open. Lowest Power. Programmable RISC-V Solutions. [webpage]: https://www.microsemi.com/product-directory/mi-v-embedded-ecosystem/4406-risc-v-cpus

[41] T. Gomes, S. Pinto, T. Gomes, A. Tavares, and J. Cabral, "Towards an fpga-based edge device for the internet of things," in *2015 IEEE 20th Conference on Emerging Technologies Factory Automation (ETFA)*, Sep. 2015, pp. 1–4.

[42] A. P. Fournaris, C. Alexakos, C. Anagnostopoulos, C. Koulamas, and A. Kalogeras, "Introducing hardware-based intelligence and reconfigurability on industrial iot edge nodes," *IEEE Design Test*, vol. 36, no. 4, pp. 15–23, Aug. 2019.

[43] O. Azizi, A. Mahesri, B. C. Lee, S. J. Patel, and M. Horowitz, "Energy-Performance Tradeoffs in Processor Architecture and Circuit Design : A Marginal Cost Analysis Categories and Subject Descriptors," *Comput. Eng.*, vol. 38, no. 3, pp. 26–36, 2010. [webpage]: http://www.ncbi.nlm.nih.gov/pubmed/17546741

[44] ARM, "ARM Cortex M-0 Technical Reference Manual," Tech. Rep.

[45] ARM, "ARM Cortex M-4 Technical Reference Manual," Tech. Rep., 2015.

[46] NXP: Power consumption and measurement of i.MXRT1050. [webpage]: https://www.nxp.com/docs/en/application-note/AN12094.pdf

[47] STMicroelectronics: STM32L476xx datasheet. [webpage]: https://www.st.com/resource/en/datasheet/stm32l476je.pdf

[48] Silicon Labs: EFM32 Giant Gecko 11 32bit Microcontrollers. [webpage]: Datasheetavailablewebpage

[49] Synopsys, *DesignWare ARC Processors*. [webpage]: https://www.synopsys.com/designware-ip/processor-solutions/arc-processors.html

[50] A. Waterman *et al.*, "The risc-v instruction set manual," 2014.

[51] 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, Nov 2019.

[52] C. Celio, K. Asanovic, and D. Patterson, "The Berkeley Out-of-Order Machine (BOOM!): An Open-source Industry-Competitive, Synthesizable, Parameterized RISC-V Processor," https://riscv.org/wp-content/uploads/2016/01/Wed1345-RISCV-Workshop-3-BOOM.pdf.

[53] Y. Lee, "RISC-V "Rocket Chip" SoC Generator in Chisel," https://riscv.org/wp-content/uploads/2015/02/riscv-rocket-chip-generator-tutorial-hpca2015.pdf.

[54] C. Wolf, "Picorv32 - https://github.com/cliffordwolf/picorv32."

[55] Y. Lee, A. Ou, and A. Magyar, "Z-scale: Tiny 32-bit RISC-V Systems," https://riscv.org/wp-content/uploads/2015/06/riscv-zscale-workshop-june2015.pdf.

[56] P. D. Schiavone *et al.*, "Slow and steady wins the race? a comparison of ultra-low-power risc-v cores for internet-of-things applications," in *2017 27th International Symposium on Power and Timing Modeling, Optimization and Simulation (PATMOS)*, Sep. 2017, pp. 1–8.

[57] M. Gautschi *et al.*, "Near-Threshold RISC-V Core With DSP Extensions for Scalable IoT Endpoint Devices," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. PP, no. 99, pp. 1–14, 2017.

[58] S. Benatti *et al.*, "A versatile embedded platform for EMG acquisition and gesture recognition," *IEEE Transactions on Biomedical Circuits and Systems*, vol. 9, no. 5, pp. 620–630, Oct. 2015.

[59] S. A. Imtiaz, L. Logesparan, and E. Rodriguez-Villegas, "Performance-power consumption tradeoff in wearable epilepsy monitoring systems," *IEEE Journal of Biomedical and Health Informatics*, vol. 19, no. 3, pp. 1019–1028, May 2015.

[60] Texas Instrument, "http://www.ti.com/ww/en/msp430.html," Tech. Rep.

[61] D. Y. Barsakcioglu and T. G. Constandinou, "A 32-channel MCU-based feature extraction and classification for scalable on-node spike sorting," in *IEEE ISCAS*. IEEE, 2016, pp. 1310–1313.

[62] M. Konijnenburg *et al.*, "28.4 a battery-powered efficient multi-sensor acquisition system with simultaneous ECG, BIO-z, GSR, and PPG," in *2016 IEEE International Solid-State Circuits Conference (ISSCC)*. IEEE, Jan. 2016, pp. 480–482.

[63] N. Ickes, D. Finchelstein, and A. P. Chandrakasan, "A 10-pJ/instruction, 4-MIPS micropower DSP for sensor applications," in *2008 IEEE Asian Solid-State Circuits Conference*. IEEE, Nov. 2008, pp. 289–292.

[64] A. Pullini, D. Rossi, I. Loi, G. Tagliavini, and L. Benini, "Mr.wolf: An energy-precision scalable parallel ultra low power soc for iot edge processing," *IEEE Journal of Solid-State Circuits*, vol. 54, no. 7, pp. 1970–1981, Jul. 2019.

[65] M. Perotti *et al.*, "Hw/sw approaches for risc-v code size reduction."

[66] "Sia," https://www.semiconductors.org/main/resources/.

[67] M. Cockrell. (2018) Use of RISC-V on Pixel Visual Core. [webpage]: https://tmt.knect365.com/risc-v-workshop-barcelona/speakers/matt-cockrell#workshop_use-of-risc-v-on-pixel-visual-core/

[68] "Running sting on pulpino platform," http://valtrix.in/programming/running-sting-on-pulpino.

[69] E. Sanchez, M. Schillaci, and G. Squillero, *Evolutionary Optimization: the uGP toolkit*. Springer US, 2011, p. 178.

[70] L. Benini, A. Bogliolo, and G. De Micheli, "A survey of design techniques for system-level dynamic power management," *IEEE transactions on very large scale integration (VLSI) systems*, vol. 8, no. 3, pp. 299–316, 2000.

[71] D. Bol *et al.*, "19.6 a 40-to-80mhz sub-4μw/mhz ulv cortex-m0 mcu soc in 28nm fdsoi with dual-loop adaptive back-bias generator for 20μs wake-up from deep fully retentive sleep mode," in *2019 IEEE International Solid-State Circuits Conference-(ISSCC)*. IEEE, 2019, pp. 322–324.

[72] Y. Liu *et al.*, "A 64-channel versatile neural recording SoC with activity-dependent data throughput," *IEEE Trans. Biomed. Circ. Syst.*, vol. 11, no. 6, pp. 1344–1355, 2017.

[73] 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.

[74] M. Hiienkari *et al.*, "A 0.4-0.9v, 2.87pj/cycle near-threshold arm cortex-m3 cpu with in-situ monitoring and adaptive-logic scan," in *2020 IEEE Symposium in Low-Power and High-Speed Chips (COOL CHIPS)*, 2020, pp. 1–3.

[75] D. S. Truesdell, J. Breiholz, S. Kamineni, N. Liu, A. Magyar, and B. H. Calhoun, "A 6–140-nw 11 hz–8.2-khz dvfs risc-v microprocessor using scalable dynamic leakage-suppression logic," *IEEE Solid-State Circuits Letters*, vol. 2, no. 8, pp. 57–60, 2019.

[76] D. Rossi *et al.*, "PULP: A Parallel Ultra Low Power Platform for next Generation IoT Applications," in *Hot Chips 27 Symposium (HCS), 2015 IEEE*. IEEE, 2015, pp. 1–39.

[77] D. Rossi *et al.*, "A self-aware architecture for pvt compensation and power nap in near threshold processors," *IEEE Design & Test*, vol. 34, no. 6, pp. 46–53, 2017.

[78] J. Lee *et al.*, "19.2 a 6.4 pj/cycle self-tuning cortex-m0 iot processor based on leakage-ratio measurement for energy-optimal operation across wide-range pvt variation," in *2019 IEEE*

International Solid-State Circuits Conference-(ISSCC). IEEE, 2019, pp. 314–315.

[79] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems 25*, 2012, pp. 1106–1114.

[80] L. Cavigelli, M. Magno, and L. Benini, "Accelerating Real-time Embedded Scene Labeling with Convolutional Networks," in *Proceedings of the 52Nd Annual Design Automation Conference*, ser. DAC '15. New York, NY, USA: ACM, 2015, pp. 108:1–108:6.

[81] R. Girshick, J. Donahue, T. Darrell, J. Malik, and U. C. Berkeley, "Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation," in *Proceedings of 2014 IEEE Conference on Computer Vision and Pattern Recognition*, 2014, pp. 580–587.

[82] G. E. Dahl, D. Yu, L. Deng, and A. Acero, "Context-Dependent Pre-Trained Deep Neural Networks for Large-Vocabulary Speech Recognition," *IEEE Transactions on Audio, Speech, and Language Processing*, vol. 20, no. 1, pp. 30–42, 2012.

[83] 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.

[84] E. Flamand *et al.* (2018, Jul.) GreenWaves Technology: GAP8 PRODUCT BRIEF. [webpage]: GAP8productbriefavailablewebpage

[85] T. Karnik *et al.*, "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.

[86] R. Bansal and A. Karmakar, "Closely-coupled lifting hardware for efficient dwt computation in an soc," *Journal of Signal Processing Systems*, pp. 1–13, 2019.

[87] M. Cavalcante, F. Schuiki, F. Zaruba, M. Schaffner, and L. Benini, "Ara: A 1-ghz+ scalable and energy-efficient risc-v vector processor with multiprecision floating-point support in 22-nm fd-soi," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 28, no. 2, pp. 530–543, Feb. 2020.

[88] T. Fritzmann, U. Sharif, D. Müller-Gritschneder, C. Reinbrecht, U. Schlichtmann, and J. Sepulveda, "Towards reliable and secure post-quantum co-processors based on risc-v," in *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2019, pp. 1148–1153.

[89] S. R. Sridhara *et al.*, "Microwatt embedded processor platform for medical system-on-chip applications," *IEEE J. Solid-State Circuits*, vol. 46, no. 4, pp. 721–730, 2011.

[90] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," *arXiv:1512.03385 [cs]*, Dec. 2015.

[91] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the Inception Architecture for Computer Vision," *arXiv:1512.00567 [cs]*, Dec. 2015.

[92] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. Alemi, "Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning," *arXiv:1602.07261 [cs]*, Feb. 2016.

[93] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer, "SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <0.5MB model size," *arXiv:1602.07360 [cs]*, Feb. 2016.

[94] 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.

[95] S. Zhou, Y. Wu, Z. Ni, X. Zhou, H. Wen, and Y. Zou, "DoReFa-Net: Training Low Bitwidth Convolutional Neural Networks with Low Bitwidth Gradients," *arXiv:1606.06160 [cs]*, Jun. 2016.

[96] B. Moons, K. Goetschalckx, N. Van Berckelaer, and M. Verhelst, "Minimum Energy Quantized Neural Networks," *arXiv:1711.00215 [cs]*, Nov. 2017.

[97] A. Pullini, F. Conti, D. Rossi, I. Loi, M. Gautschi, and L. Benini, "A heterogeneous multi-core system-on-chip for energy efficient brain inspired computing," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. PP, no. 8, pp. 1–1, 2017.

[98] M. Courbariaux, I. Hubara, D. Soudry, R. El-Yaniv, and Y. Bengio, "Binarized Neural Networks: Training Deep Neural Networks with Weights and Activations Constrained to +1 or -1," *arXiv:1602.02830 [cs]*, Feb. 2016.

[99] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, "Xnor-net: Imagenet classification using binary convolutional neural networks," in *European conference on computer vision*. Springer, 2016, pp. 525–542.

[100] ARM, "big.little processing technologies - https://www.arm.com/products/processors/technologies/biglittleprocessing.p Tech. Rep.

[101] Texas Instruments, "CC2650 SimpleLink™ Multistandard Wireless MCU," Tech. Rep. July, 2016.

[102] NXP, "LPC5410x Datasheet," Tech. Rep. July, 2016.

[103] N. Neves, N. Sebastiao, D. Matos, P. Tomas, P. Flores, and N. Roma, "Multicore SIMD ASIP for Next-Generation Sequencing and Alignment Biochip Platforms," *IEEE Trans. Very Large Scale Integr. Syst.*, vol. 23, no. 7, pp. 1287–1300, 2015.

[104] S. Benatti *et al.*, "Scalable EEG seizure detection on an ultra low power multi-core architecture," in *IEEE BioCAS*. IEEE, 2016, pp. 86–89.

[105] V. J. Kartsch, S. Benatti, P. D. Schiavone, D. Rossi, and L. Benini, "A sensor fusion approach for drowsiness detection in wearable ultra-low-power systems," *Information Fusion*, vol. 43, pp. 66–76, 2018.

[106] V. Kartsch, G. Tagliavini, M. Guermandi, S. Benatti, D. Rossi, and L. Benini, "Biowolf: A sub-10-mw 8-channel advanced brain–computer interface platform with a nine-core processor and ble connectivity," *IEEE Transactions on Biomedical Circuits and Systems*, vol. 13, no. 5, pp. 893–906, Oct. 2019.

[107] S. Gibson, J. W. Judy, and D. Marković, "Spike sorting: The first step in decoding the brain: The first step in decoding the brain," *IEEE Signal Processing Magazine*, vol. 29, no. 1, pp. 124–143, 2012.

[108] T. M. Seese *et al.*, "Characterization of tissue morphology, angiogenesis, and temperature in the adaptive response of muscle tissue to chronic heating." *Laboratory investigation; a journal of technical methods and pathology*, vol. 78, no. 12, pp. 1553–1562, 1998.

[109] G. Buzsáki *et al.*, "Tools for probing local circuits: high-density silicon probes combined with optogenetics," *Neuron*, vol. 86, no. 1, pp. 92–105, 2015.

[110] H. Morioka *et al.*, "Learning a common dictionary for subject-transfer decoding with resting calibration," *NeuroImage*, vol. 111, pp. 167–178, 2015.

[111] V. Jayaram, M. Alamgir, Y. Altun, B. Scholkopf, and M. Grosse-Wentrup, "Transfer learning in brain-computer interfaces," *IEEE Computational Intelligence Magazine*, vol. 11, no. 1, pp. 20–31, 2016.

[112] F. Renzini, C. Mucci, D. Rossi, E. F. Scarselli, and R. Canegallo, "A fully programmable efpga-augmented soc for smart power applications," *IEEE Transactions on Circuits and Systems I: Regular Papers*, pp. 1–13, 2019.

[113] Menta: eFPGA IP Cores. [webpage]: Availablewebpageathttps://www.menta-efpga.com/efpga-ips

[114] P. N. Whatmough *et al.*, "A 16nm 25mm2 soc with a 54.5x flexibility-efficiency range from dual-core arm cortex-a53 to efpga

and cache-coherent accelerators," in *2019 Symposium on VLSI Circuits*, Jun. 2019, pp. C34–C35.

[115] M. Borgatti, F. Lertora, B. Foret, and L. Cali, "A reconfigurable system featuring dynamically extensible embedded microprocessor, fpga, and customizable i/o," *IEEE Journal of Solid-State Circuits*, vol. 38, no. 3, pp. 521–529, Mar. 2003.

[116] A. Lodi *et al.*, "Xisystem: a xirisc-based soc with reconfigurable io module," *IEEE Journal of Solid-State Circuits*, vol. 41, no. 1, pp. 85–96, Jan. 2006.

[117] Achronix Speedcore eFPGA. [webpage]: SpeedcoreFPGAdatasheetavailablewebpage

[118] Flex-Logix eFPGA. [webpage]: https://flex-logix.com/efpga/

[119] Quicklogic: ArcticPro 2 eFPGA. [webpage]: Availablewebpageathttps://www.quicklogic.com/products/efpga/arcticpro-2/

[120] P. D. Schiavone *et al.*, "An open-source verification framework for open-source cores: A risc-v case study," in *2018 IFIP/IEEE International Conference on Very Large Scale Integration (VLSI-SoC)*, 2018, pp. 43–48.

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

[122] A. Di Mauro, F. Conti, P. Schiavone, D. Rossi, and L. Benini, "Pushing on-chip memories beyond reliability boundaries in micropower machine learning applications," in *65th International Electron Devices Meeting (IEDM 2019)*, 2019.

[123] 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 22nm iot end-node," 2020.

[124] F. Conti *et al.*, "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.

[125] P. D. Schiavone *et al.*, "Neuro-pulp: A paradigm shift towards fully programmable platforms for neural interfaces," in *2020 2nd IEEE International Conference on Artificial Intelligence Circuits and Systems (AICAS)*, 2020, pp. 50–54.

[126] P. D. Schiavone *et al.*, "Arnold: an efpga-augmented risc-v soc for flexible and low-power iot end-nodes," *arXiv preprint arXiv:2006.14256*, 2020.

[127] A. P. Chandrakasan, S. Sheng, and R. W. Brodersen, "Low-power cmos digital design," *IEEE Journal of Solid-State Circuits*, vol. 27, no. 4, pp. 473–484, 1992.

[128] D. Rossi *et al.*, "A -1.8v to 0.9v body bias, 60 GOPS/w 4-core cluster in low-power 28nm UTBB FD-SOI technology," in *2015 IEEE SOI-3D-Subthreshold Microelectronics Technology Unified Conference (S3S)*. IEEE, Oct. 2015.

[129] D. Rossi *et al.*, "A 60 GOPS/W, -1.8 V to 0.9 V body bias ULP cluster in 28 nm UTBB FD-SOI technology," *Solid-State Electronics*, vol. 117, pp. 170–184, Mar. 2016.

[130] D. Rossi *et al.*, "193 MOPS/mW 162 MOPS, 0.32V to 1.15V Voltage Range Multi-Core Accelerator for Energy-Efficient Parallel and Sequential Digital Processing," in *Cool Chips XIX*, 2016, pp. 1–3.

[131] R. Banakar, S. Steinke, B.-S. L. B.-S. Lee, M. Balakrishnan, and P. Marwedel, "Scratchpad memory: a design alternative for cache on-chip memory in embedded systems," *Proc. Tenth Int. Symp. Hardware/Software Codesign. CODES 2002*, pp. 73–78, 2002.

[132] A. Teman, D. Rossi, P. Meinerzhagen, L. Benini, and A. Burg, "Power, Area, and Performance Optimization of Standard Cell Memory Arrays Through Controlled Placement," *ACM Trans.*

Des. Autom. Electron. Syst., vol. 21, no. 4, pp. 59:1–59:25, May 2016.

[133] 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)*, Sep. 2017, pp. 1–8.

[134] ARM Limited, "https://www.arm.com/products/processors/cortex-m."

[135] R. B. Lee, "Subword parallelism with MAX-2," *IEEE Micro*, vol. 16, no. 4, pp. 51–59, 1996.

[136] A. Shahbahrami, B. Juurlink, and S. Vassiliadis, "A Comparison Between Processor Architectures for Multimedia Application," in *Proc. 15th Annu. Work. Circuits, Syst. Signal Process. ProRisc*, 2004, pp. 138–152.

[137] H. Chang, J. Cho, and W. Sung, "Performance evaluation of an SIMD architecture with a multi-bank vector memory unit," in *IEEE Work. Signal Process. Syst. Des. Implementation, SIPS*, 2006, pp. 71–76.

[138] C.-H. Chang, S.-H. Chen, B.-W. Chen, W. Ji, K. Bharanitharan, and J.-F. Wang, "Transactions Briefs Fixed-Point Computing Element Design for Transcendental Functions and Primary Operations in Speech Processing," *IEEE Trans. Very Large Scale Integr. Syst.*, vol. 24, no. 5, pp. 1993–1997, 2016.

[139] B. Parhami, "Variations on multioperand addition for faster logarithmic-time tree multipliers," in *Conf. Rec. Thirtieth Asilomar Conf. Signals, Syst. Comput.*, 1996, pp. 899–903. [webpage]: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=599074

[140] S. Gal-On and M. Levy, "Exploring coremark™ -a benchmark maximizing simplicity and efficacy," Tech. Rep.

[141] "Pulpino," https://github.com/pulp-platform/pulpino.

[142] S. Tasiran and K. Keutzer, "Coverage metrics for functional validation of hardware designs," *IEEE Design Test of Computers*, vol. 18, no. 4, pp. 36–45, Jul. 2001.

[143] E. Hung, B. Quinton, and S. J. E. Wilton, "Linking the verification and validation of complex integrated circuits through shared coverage metrics," *IEEE Design Test*, vol. 30, no. 4, pp. 8–15, Aug. 2013.

[144] F.-C. Yang, W.-K. Huang, J.-K. Zhong, and J. Huang, "Automatic verification of external interrupt behaviors for microprocessor design," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 27, no. 9, pp. 1670–1683, 2008.

[145] F. Corno, E. Sanchez, M. S. Reorda, and G. Squillero, "Automatic test program generation: a case study," *IEEE Design Test of Computers*, vol. 21, no. 2, pp. 102–109, Mar. 2004.

[146] A. Rahimi, I. Loi, M. R. Kakoee, and L. Benini, "A fully-synthesizable single-cycle interconnection network for shared-l1 processor clusters," in *2011 Design, Automation Test in Europe*, Mar. 2011, pp. 1–6.

[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] L. Yang, D. Bankman, B. Moons, M. Verhelst, and B. Murmann, "Bit Error Tolerance of a CIFAR-10 Binarized Convolutional Neural Network Processor," in *2018 IEEE International Symposium on Circuits and Systems (ISCAS)*, May 2018, pp. 1–5.

[149] D. Bol *et al.*, "Sleepwalker: A 25-mhz 0.4-v sub- mm^2 7-μW/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.

[150] N. Ickes, Y. Sinangil, F. Pappalardo, E. Guidetti, and A. P. Chandrakasan, "A 10 pj/cycle ultra-low-voltage 32-bit microprocessor system-on-chip," in *ESSCIRC (ESSCIRC), 2011 Proceedings of the*. IEEE, 2011, pp. 159–162.

[151] F. Conti and L. Benini, "A Ultra-Low-Energy Convolution Engine for Fast Brain-Inspired Vision in Multicore Clusters," in *Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition*, ser. DATE '15. San Jose, CA, USA: EDA Consortium, 2015, pp. 683–688.

[152] M. Gautschi *et al.*, "Tailoring instruction-set extensions for an ultra-low power tightly-coupled cluster of openrisc cores," in *VLSI-SoC 2015*, Oct. 2015, pp. 25–30.

[153] M. Courbariaux, Y. Bengio, and J.-P. David, "Training Deep Neural Networks with Low Precision Multiplications," *arXiv:1412.7024 [cs]*, Dec. 2014.

[154] M. Courbariaux, Y. Bengio, and J.-P. David, "BinaryConnect: Training Deep Neural Networks with binary weights during propagations," in *Advances in Neural Information Processing Systems 28*, C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, Eds. Curran Associates, Inc., 2015, pp. 3123–3131.

[155] *OpenRISC 1000 Architecture Manual*, 2012.

[156] S. Benatti, F. Montagna, V. Kartsch, A. Rahimi, D. Rossi, and L. Benini, "webpage learning and classification of emg-based gestures on a parallel ultra-low power platform using hyperdimensional computing," *IEEE Transactions on Biomedical Circuits and Systems*, vol. 13, no. 3, pp. 516–528, Jun. 2019.

[157] V. Kartsch, S. Benatti, M. Guermandi, F. Montagna, and L. Benini, "Ultra low-power drowsiness detection system with biowolf," in *2019 9th International IEEE/EMBS Conference on Neural Engineering (NER)*, Mar. 2019, pp. 1187–1190.

[158] V. Kartsch, M. Guermandi, S. Benatti, F. Montagna, and L. Benini, "An energy-efficient iot node for hmi applications

based on an ultra-low power multicore processor," in *2019 IEEE Sensors Applications Symposium (SAS)*, Mar. 2019, pp. 1–6.

[159] R. Aghazadeh, F. Montagna, S. Benatti, D. Rossi, and J. Frounchi, "Compressed sensing based seizure detection for an ultra low power multi-core architecture," in *2018 International Conference on High Performance Computing Simulation (HPCS)*, Jul. 2018, pp. 492–495.

[160] A. Muzet, T. Pébayle, J. Langrognet, and S. Otmani, "Awake pilot study no. 2: Testing steering grip sensor measures," *Technical Report IST-2000-28062, CEPA*, 2003.

[161] S. Luan *et al.*, "Compact standalone platform for neural recording with real-time spike sorting and data logging," *J. Neural Eng.*, 2018.

[162] D. A. Schwarz *et al.*, "Chronic, wireless recordings of large-scale brain activity in freely moving rhesus monkeys," *Nature Methods*, vol. 11, no. 6, p. 670, 2014.

[163] X. Liu *et al.*, "The PennBMBI: design of a general purpose wireless brain-machine-brain interface system," *IEEE Trans. Biomed. Circ. Syst.*, vol. 9, no. 2, pp. 248–258, 2015.

[164] G. Gagnon-Turcotte, Y. LeChasseur, C. Bories, Y. De Koninck, and B. Gosselin, "A wireless optogenetic headstage with multichannel neural signal compression," in *IEEE BioCAS*. IEEE, 2015, pp. 1–4.

[165] C. Pedreira *et al.*, "How many neurons can we see with current spike sorting algorithms?" *J. Neurosci. Meth.*, vol. 211, no. 1, pp. 58–65, 2012.

[166] RHD2000-Series Digital EPhys. Interface Chips. [webpage]: http://intantech.com/files/Intan_RHD2000_series_datasheet.pdf

[167] EFLX: EFLX 4K Product Brief for TSMC 12FFC+/12FFC/16FFC+/FFC/FF+. [webpage]: Availablewebpageathttps://flex-logix.com/efpga/

[168] F. H. Elfouly, M. I. Mahmoud, M. I. Dessouky, and S. Deyab, "Comparison between haar and daubechies wavelet transformations on fpga technology," *International Journal of Computer, Information, and Systems Science, and Engineering*, vol. 2, no. 1, 2008.

[169] A. Burrello, L. Cavigelli, K. Schindler, L. Benini, and A. Rahimi, "Laelaps: An energy-efficient seizure detection algorithm from long-term human ieeg recordings without false alarms," in *2019 Design, Automation Test in Europe Conference Exhibition (DATE)*, Mar. 2019, pp. 752–757.

[170] Inivation. (2020) Inivation dynamic vision platform. [webpage]: https://inivation.com/dvp/

[171] S. Liu, A. van Schaik, B. A. Minch, and T. Delbruck, "Asynchronous binaural spatial audition sensor with $2 \times 64 \times 4$ channel output," *IEEE Transactions on Biomedical Circuits and Systems*, vol. 8, no. 4, pp. 453–464, Aug. 2014.

[172] F. Conti, L. Cavigelli, G. Paulin, I. Susmelj, and L. Benini, "Chipmunk: A systolically scalable 0.9 mm2, 3.08gop/s/mw @ 1.2 mw accelerator for near-sensor recurrent neural network inference," in *2018 IEEE Custom Integrated Circuits Conference (CICC)*, Apr. 2018, pp. 1–4.

[173] 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.

[174] D. Chen, J. Cong, S. Gurumani, W. Hwu, K. Rupnow, and Z. Zhang, "Platform choices and design demands for iot platforms: cost, power, and performance tradeoffs," *IET Cyber-Physical Systems: Theory Applications*, vol. 1, no. 1, pp. 70–77, 2016.

[175] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, "Quantized neural networks: Training neural networks with low precision weights and activations," *The Journal of Machine Learning Research*, vol. 18, no. 1, pp. 6869–6898, 2017.

[176] E. Tsimbalo, X. Fafoutis, and R. J. Piechocki, "Crc error correction in iot applications," *IEEE Transactions on Industrial Informatics*, vol. 13, no. 1, pp. 361–369, Feb. 2017.

Curriculum Vitae

Pasquale Davide Schiavone was born in Turin in 1989, received both BSc. and MSc. degree in computer engineering from the Polytechnic of Turin, Italy, in 2013 and 2016 respectively. His master thesis has been developed at the Integrated System Laboratory, Swiss Federal Institute of Technology, Zurich with the tile "Evaluation of DSP extensions for an open-source RISC processor", under the supervision of Professor Luca Benini. In 2016 he started his PhD in the same laboratory under the supervision of Professor Luca Benini. In 2018, he won the Mobility Grant In Project to visit the Imperial College London for 6 months, at the Centre for Bio-Inspired Technology, under the supervsion of professor Timothy Constantinou. His research interests include datapath blocks design, low-power microprocessors in multi-core systems and deep-learning architectures for energy efficient systems.