Towards Heterogeneous Multi-core Systems-on-Chip for Edge Machine Learning

Vikram Jain • Marian Verhelst

Towards Heterogeneous Multi-core Systems-on-Chip for Edge Machine Learning

Journey from Single-core Acceleration to Multi-core Heterogeneous Systems

Springer

Vikram Jain
ESAT-MICAS
KU Leuven
Leuven, Belgium

Marian Verhelst
ESAT-MICAS
KU Leuven
Leuven, Belgium

ISBN 978-3-031-38229-1 ISBN 978-3-031-38230-7 (eBook)
https://doi.org/10.1007/978-3-031-38230-7

© The Editor(s) (if applicable) and The Author(s), under exclusive license to Springer Nature Switzerland AG 2024
This work is subject to copyright. All rights are solely and exclusively licensed by the Publisher, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, reuse of illustrations, recitation, broadcasting, reproduction on microfilms or in any other physical way, and transmission or information storage and retrieval, electronic adaptation, computer software, or by similar or dissimilar methodology now known or hereafter developed.
The use of general descriptive names, registered names, trademarks, service marks, etc. in this publication does not imply, even in the absence of a specific statement, that such names are exempt from the relevant protective laws and regulations and therefore free for general use.
The publisher, the authors, and the editors are safe to assume that the advice and information in this book are believed to be true and accurate at the date of publication. Neither the publisher nor the authors or the editors give a warranty, expressed or implied, with respect to the material contained herein or for any errors or omissions that may have been made. The publisher remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

This Springer imprint is published by the registered company Springer Nature Switzerland AG
The registered company address is: Gewerbestrasse 11, 6330 Cham, Switzerland

Paper in this product is recyclable.

Preface

Machine Learning (ML), specifically deep learning (DL), has become the workhorse for emerging applications in vision, audio, sensing, and data analytics. State-of-the-art DL models are incredibly costly regarding model size, computational resources required, and the energy consumption of running the models. Owing to their size and complexity, they can only be deployed on large devices like GPUs typically used in cloud servers or data centers. The cloud-computing paradigm, however, comes with several drawbacks. As the large amount of data collected at the end-user devices needs transmission to the resourceful cloud servers, the energy consumption of data communication increases. This is further exacerbated by the fact that there is an exponential growth in the number of end-user devices resulting in copious amounts of data that needs to be transmitted. Moreover, the security of the collected data is not guaranteed, and recent privacy concerns have also come to the forefront. Finally, issues like latency and reliability have additionally become concerns that degrade a seamless real-time experience. Given the drawbacks of cloud computing, the logical consequence is to process data closer to the end-user devices and only sparsely transmit the data necessary for further processing or making decisions.

The paradigm of near-end-user or near-sensor processing has been coined the term (extreme-)edge-computing. (Extreme-)edge-computing can alleviate the drawbacks (energy, privacy, latency, and reliability) of cloud computing by using (extreme) edge devices for data processing as close to the sensor as possible. However, it comes with new challenges, as these devices are battery-operated and severely resource-constrained. Furthermore, as the Internet of Things becomes more pervasive, the number of sensors connected to each edge device proliferates, thereby increasing the data to be processed. Computing at the (extreme) edge requires highly energy-efficient and flexible hardware to map diverse ML and DL workloads, enabling various applications on a single platform. Moreover, it needs algorithms and models specifically designed for resource-constrained devices, thereby requiring a careful co-optimization of hardware and software. This book focuses on the first aspect of the abovementioned challenges of (extreme-)edge-computing, i.e., the design of energy-efficient and flexible hardware architectures and hardware-software co-optimization strategies to enable early design space exploration of hardware architectures.

The book first focuses on the design of the highly specialized single hardware accelerator optimized for the application of object detection in drones. As the application and model to be accelerated are fixed, the hardware is optimized for mapping only convolutional and dense layers of a DL model in an object detection pipeline. Emerging DL applications deployed on the (extreme) edge devices, however, require multi-modal support, which demands, on the one hand, the need for much more flexible hardware accelerators and, on the other hand, complete standalone systems with the always-on and duty-cycled operation. Heterogeneity in hardware acceleration can enhance a system's flexibility and energy efficiency by utilizing various energy-efficient hardware accelerators supporting multiple DL workloads on a single platform. With this motivation, the book presents a versatile all-digital heterogeneous multi-core system-on-chip with a highly flexible ML accelerator, a RISC-V core, non-volatile memory, and a power management unit. A highly energy-efficient heterogeneous multi-core system-on-chip is presented next by combining a digital and analog in-memory computing core controlled by a single RISC-V core.

Increasing the core count further can benefit the performance of a system. However, data communication in multi-core platforms can quickly become a bottleneck if the design is not optimized. Multi-core CPUs have extensively used classical network-on-chips (NoCs) to address the data communication bottleneck. However, these NoCs use serial packet-based protocols suffering from significant protocol translation overheads toward the endpoints. In the book's final part, an open-source, fully AXI-compliant NoC fabric is proposed to better address the specific needs of multi-core DL computing platforms requiring significant burst-based communication. The NoC enables scaling DNN platforms to multi-accelerator systems, thus allowing the journey toward high-performance heterogeneous multi-core systems.

Accumulating the learning from the results obtained throughout the book, the aim is to enable flexible, high-performance, and energy-efficient (extreme) edge hardware platforms through cross-domain optimization to cater to the needs of the evolving field of machine learning. This would help win the *hardware lottery*—the problem of hardware implementation lagging behind the pace of algorithmic evolution. It will also democratize the impressive power of artificial intelligence to build a more efficient world.

Leuven, Belgium

Vikram Jain
Marian Verhelst

Acknowledgments

We would like to acknowledge several individuals and institutions for their support and contributions to this work.

We express our deepest gratitude to Peter Karsmakers, Luc Van Eycken, Hans Hallez, Luca Benini, and Ying Cao for the insightful discussions and detailed feedback that helped improve the previous versions of this manuscript. We would like to acknowledge several collaborators for their contributions to the works presented throughout this book: Linyan Mei, Ninad Jadhav, Sebastian Giraldo, Jaro De Roose, Kodai Ueyoshi, Giuseppe Sarda, Pouya Houshmand, Man Shi, and Matheus Cavalcante.

Finally, we would like to thank Magics Technologies and FOD Economie Belgium for their financial support under the ISAAC project (Energietransitiefonds (oproep II)). We would also like to thank Flanders AI (FAIR) and European Research Council (ERC) for supporting several projects presented in this book, and acknowledge the generous support from GlobalFoundries for the chip tapeouts.

Contents

1 Introduction 1
- 1.1 Machine Learning at the (Extreme) Edge 2
  - 1.1.1 Applications 3
  - 1.1.2 Algorithms 6
  - 1.1.3 Hardware 10
- 1.2 Open Challenges for ML Acceleration at the (Extreme) Edge 12
- 1.3 Book Contributions 15

2 Algorithmic Background for Machine Learning 21
- 2.1 Support Vector Machines 21
- 2.2 Deep Learning Models 26
  - 2.2.1 Neural Networks 28
  - 2.2.2 Training 28
  - 2.2.3 Inference: Neural Network Topologies 31
  - 2.2.4 Model Compression 39
- 2.3 Feature Extraction 40
- 2.4 Conclusion 42

3 Scoping the Landscape of (Extreme) Edge Machine Learning Processors 43
- 3.1 Hardware Acceleration of ML Workloads: A Primer 43
  - 3.1.1 Core Mathematical Operation 44
  - 3.1.2 General Accelerator Template 44
- 3.2 Evaluation Metrics 46
- 3.3 Survey of (Extreme) Edge ML Hardware Platforms 49
- 3.4 Evaluating the Surveyed Hardware Platforms 51
- 3.5 Insights and Trends 56
- 3.6 Conclusion 57

4 Hardware–Software Co-optimization Through Design Space Exploration 59
- 4.1 Motivation 59

4.2 Exploration Methodology 60
4.2.1 ZigZag 60
4.2.2 Post-Processing of ZigZag's Results 61
4.3 DNN Workload Comparison 62
4.3.1 Exploration Setup 63
4.3.2 Visualization of the Complete Trade-Off Space 64
4.3.3 Impact of HW Architecture on Optimal Workload 66
4.3.4 Impact of Workload on Optimal HW Architecture 67
4.4 Conclusion 67

5 Energy-Efficient Single-Core Hardware Acceleration 71
5.1 Motivation 71
5.2 Metrics for Hardware Optimization 74
5.3 State-of-the-Art in Object Detection on FPGA 75
5.4 Cost-Aware Algorithmic Optimization 77
5.4.1 Object Detection Algorithms 77
5.4.2 Quantization of Tiny-YOLOv2 78
5.5 Cost-Aware Architecture Optimization 80
5.5.1 Hardware Mapping of Convolutional Layers 80
5.5.2 Hardware Architecture of the Accelerator 84
5.6 Cost-Aware System Optimization 86
5.6.1 Data Communication Architecture 86
5.6.2 Tiling Strategy 87
5.7 Implementation Results 88
5.8 Conclusion 91

6 TinyVers: A Tiny Versatile All-Digital Heterogeneous Multi-core System-on-Chip 93
6.1 Motivation 93
6.2 Algorithmic Background 94
6.2.1 Convolution and Dense Operation 95
6.2.2 Deconvolution 95
6.2.3 Support Vector Machines (SVMs) 96
6.3 TinyVers Hardware Architecture 97
6.3.1 Smart Sensing Modes for TinyML 98
6.3.2 Power Management 99
6.4 FlexML Accelerator 100
6.4.1 FlexML Architecture Overview 101
6.4.2 Dataflow Reconfiguration 103
6.4.3 Efficient Zero-Skipping for Deconvolution and Blockwise Structured Sparsity 104
6.4.4 Support Vector Machine 106
6.5 Deployment of Neural Networks on TinyVers 106
6.6 Design for Test and Fault-Tolerance 107
6.7 Chip Implementation and Measurement 108
6.7.1 Peak Performance Analysis 108

6.7.2 Workload Benchmarks 110
6.7.3 Power Management 111
6.7.4 Instantaneous Power Trace 113
6.8 Comparison with SotA 115
6.9 Conclusion 117

7 DIANA: DIgital and ANAlog Heterogeneous Multi-core System-on-Chip 119
7.1 Motivation 119
7.2 Design Choices 121
7.2.1 Dataflow Concepts 121
7.2.2 Design Space Exploration 123
7.2.3 A Reconfigurable Heterogeneous Architecture 125
7.2.4 Optimization Strategies for Multi-core 125
7.3 System Architecture 126
7.3.1 The RISC-V CPU and Network Control 126
7.3.2 Memory System 127
7.4 AIMC Computing Core 129
7.4.1 AIMC Core Microarchitecture 129
7.4.2 Memory Control Unit (MCU) 129
7.4.3 AIMC Macro 131
7.4.4 Output Buffer and SIMD Unit 131
7.5 Digital DNN Accelerator 132
7.6 Measurements 133
7.6.1 Efficiency vs. Accuracy Trade-Off in the Analog Macro 134
7.6.2 Peak Performance and Efficiency Characterization 135
7.6.3 Workload Performance Characterization 136
7.6.4 SotA Comparison 139
7.7 Conclusion 141

8 Networks-on-Chip to Enable Large-Scale Multi-core ML Acceleration 143
8.1 Motivation 143
8.2 Background 145
8.2.1 Network-on-Chips 145
8.2.2 AXI Protocol 147
8.3 Interconnect Architecture of PATRONoC 149
8.4 Implementation Results 151
8.5 Performance Evaluation 153
8.5.1 Uniform Random Traffic 153
8.5.2 Synthetic Traffic 154
8.5.3 DNN Workload Traffic 156
8.6 Related Work 159
8.7 Conclusion 160

9 Conclusion 163
9.1 Overview and Contributions 164
9.2 Suggestions for Future Work 166
9.2.1 The Low Hanging Fruits 166
9.2.2 Medium Term 168
9.2.3 Moonshot 169
9.3 Closing Remarks 169

References 171

Index 185

List of Abbreviations

AE	Autoencoders. 95, 103, 115, 165
AI	artificial intelligence. 6, 26, 50
AIMC	analog in-memory compute. 18, 119–127, 129, 131–135, 138, 139, 141, 165
ANN	artificial neural network. 28
ASIC	application-specific integrated circuits. 72, 91
AUC	area under curve. 9, 46
CNN	Convolutional Neural Network. 31–35, 43, 47, 48, 54, 71–73, 75, 80, 81, 87, 94, 95, 103, 108–111, 115, 121, 132, 156, 158, 165
CPU	central processing unit. 2, 3, 10, 13–15, 18, 43, 50, 76, 85, 91, 93, 97, 98, 126, 128, 141, 143–146, 153, 159, 161, 165, 166, 169
DL	deep learning. 21, 27, 42, 48, 69
DMA	direct memory access. 86, 97, 107, 126–128, 148, 153, 154
DNN	deep neural networks. 9, 10, 15, 43, 46, 47, 59–61, 94, 95, 101, 108, 115, 119–121, 126, 132, 136, 139, 141, 143–147, 150, 153, 155, 156, 158–160, 167
DSE	design space exploration. xiv, xviii, 59, 60, 63, 67
FC	fully connected. 31, 34, 54, 94, 103, 110, 115, 136, 165
FeRAM	ferroelectric random access memory. 100
FIFO	First-in first-out. 87, 102, 104
FPGA	field programmable gate array. 11, 17, 69, 71–78, 80, 82, 84, 86–89, 91, 100, 133, 164
FPR	false positive rate. 9
GOPS	giga operations per second. 51, 109, 117
GPU	graphics processing unit. 2, 10, 11, 27, 30, 71, 72
HW	hardware. 60, 61, 63, 66–68
IMC	in-memory computing. 44, 50, 54, 57, 141
IoT	internet of things. 1–3, 5, 10, 13, 14, 42, 49, 50, 54, 55, 57, 93–95, 98, 115, 116, 143, 163, 165
KWS	keyword spotting. 4, 47, 110, 113, 115, 168
LSTM	long short-term memory. 36, 37, 95

LVSR	large vocabulary speech recognition. 4
MAC	multiply-and-accumulate. 44, 45, 50, 60, 74, 84, 101, 104, 119, 121, 132, 134
MCU	microcontroller unit. 2–4, 14, 15, 43, 93, 94, 98
MFCC	mel-frequency cepstrum coefficient. 9, 27, 41
MFEC	mel-frequency energy coefficient. 41, 113, 114
ML	Machine learning. 1–4, 6, 7, 10–15, 17–19, 21, 28, 30, 34, 40, 42–45, 47–52, 54–57, 93–95, 97, 100, 101, 103, 106, 110, 113, 115–117, 141, 160, 163–169
MLP	multi-layer perceptron. 31
MOPS	mega operations per second. 109
MRAM	magneto-resistive random access memory. 13, 14, 94, 97, 100, 113, 117
NN	neural network. 2, 28, 29, 31, 59, 60, 62–64, 66–68, 93, 94, 100, 102, 106, 107, 128
NoC	networks-on-chip. 19, 126, 143–156, 158–160, 166, 169
NPU	neural processing unit. 11, 14, 145, 146
NVM	non-volatile memory. 14, 100
OC-SVM	One-class support vector machines. 21, 96, 106, 110, 111
PCA	principal component analysis. 21
PCM	phase change memory. 100
PE	processing element. 3, 44, 45, 59, 61, 63, 68, 69, 75, 81–85, 87, 89, 91, 95, 97, 101–104, 106, 108, 119, 126, 132, 145, 165
RBF	radial basis function. 25, 96
ReRAM	resistive random access memory. 13, 14, 100
RNN	Recurrent Neural Networks. 36, 38, 43, 47, 48, 54, 55, 95, 103, 110, 115, 165
ROC	receiver operator characteristic. 9, 46
RTL	register-transfer level. 153, 157, 159
SIMD	single-instruction multiple-data. 45, 84–86, 101–103, 129, 131, 132
SoC	system-on-chip. 50–55, 57, 94, 97, 99, 100, 107–115, 117, 120, 121, 125, 126, 147, 157, 165, 168, 169
SotA	state-of-the-art. 1, 43, 55, 56, 71, 73, 76, 80, 89, 94, 109, 119, 164–169
SRAM	static random access memory. 13, 14, 52–55, 97, 100, 102, 127, 128, 131, 133
SVM	support vector machines. 21, 23–25, 43, 46, 48, 55, 95, 96, 100, 103, 110, 115, 165
TCN	Temporal Convolutional Networks. 37–39, 43, 48, 94, 95, 103, 104, 110, 111, 113, 115, 165
TOPS/W	tera operations per second per watt. 51, 109–111, 115, 117, 165
TPR	true positive rate. 8, 9
ULP	ultra-low power. 94

List of Figures

Fig. 1.1 IoT hierarchy showing the cloud–edge continuum 2
Fig. 1.2 Building blocks of ML at the edge 4
Fig. 1.3 (**a**) A speech recognition pipeline showing KWS and LVSR systems running on edge device and cloud, respectively, (**b**) an image recognition pipeline detecting object, and (**c**) a machine monitoring pipeline for industrial predictive maintenance application 5
Fig. 1.4 Machine learning pipeline overview. Based on source [60] 7
Fig. 1.5 Examples of optimum-fit, under-fit, and over-fit models trained on a binary classification task 8
Fig. 1.6 Confusion matrix for a binary classification task (left) and an ROC curve example (right) 9
Fig. 1.7 Overview of the book contributions 16
Fig. 2.1 (Left) Possible hyperplanes in a binary SVM classifier and (right) optimal hyperplane with maximum margin 22
Fig. 2.2 (Left) Hyperplane in a 2D feature space is a line and (right) hyperplane in a 3D feature space is a plane................. 22
Fig. 2.3 (Left) Hyperplane with a small margin and (right) hyperplane with a large margin 23
Fig. 2.4 Kernel trick transforms the feature space from lower dimension to higher dimension to make the classes easily separable 25
Fig. 2.5 (Top) Classical Machine Learning with handcrafted features and (bottom) deep learning with integrated feature extraction and classification operating on raw input 26
Fig. 2.6 An example of a single hidden layer MLP with commonly used non-linear activation functions 32
Fig. 2.7 An example of a multi-layer stacked convolutional neural network model 33
Fig. 2.8 An example of a multi-layer stacked residual neural network showing the internals of the residual block 34

Fig. 2.9 MobileNet's depthwise separable layer divides standard convolutional layer into depthwise and pointwise layers 35
Fig. 2.10 Example of a convolutional autoencoder 35
Fig. 2.11 (**a**) A recurrent neural network and its temporally unrolled form (based on source [165]), (**b**) internal gates of an LSTM network, and (**c**) notations used in (**b**) (Source [165]) 37
Fig. 2.12 Dilated convolution of input x_t to produce output y_t in a temporal convolutional network 38
Fig. 2.13 Pruning of weights and neurons to achieve model compression 40
Fig. 2.14 MFCC and MFEC feature extraction pipeline and computational steps 41
Fig. 3.1 General accelerator template for ML platforms 45
Fig. 3.2 An example dataflow 46
Fig. 3.3 Throughput vs. peak power for the surveyed hardware platforms. Each MAC is equivalent to two operations 51
Fig. 3.4 Data type and precisions vs. performance for the surveyed hardware platforms 52
Fig. 3.5 On-chip SRAM memory size vs. energy efficiency for the surveyed hardware platforms 53
Fig. 3.6 Area vs. area efficiency for the surveyed hardware platforms 53
Fig. 3.7 Area vs. energy efficiency for the surveyed hardware platforms 54
Fig. 4.1 ZigZag framework diagram [147] 61
Fig. 4.2 Exploration results for MobileNetV2 for different layers shown in bottom subplot as L1, L2, L3, etc. The top 3 subplots describe different memory hierarchies, with the colored points representing memory sharing of one or more operands (W/I/O) at memory level 0/1/2 and black points indicating private memories at all memory levels. The bottom plot shows the energy (stack bars) and area (black curve) of individual layers deployed on each hierarchy. All results are derived with CACTI7 [8] 65 nm technology node 62
Fig. 4.3 Memory hierarchy template for the DSE, visualizing memory levels/sharing/bypassing of Table 4.1 63
Fig. 4.4 Energy–Latency–Area comparison for mapping 12 NNs on 720 accelerator architectures each. Each NN architecture pair corresponds to two points in the figure, one with min-energy mapping and one with min-latency mapping. The Pareto-optimal accelerator architectures for each neural network are highlighted and connected [147] 65

Fig. 4.5 Box and whisker plots for energy (top) and latency (bottom) for individual NNs across 200 hardware architectures 66
Fig. 4.6 Box plots for energy and latency for 100 hardware architectures with 12 NNs deployed on them. The energy, respectively, latency, values of each NN-HW combination are normalized to min-energy, respectively, latency, of that NN. The top 3 subplots show the memory hierarchy, with the colored points representing memory sharing of one or more operands at L1/L2 and black points indicating private memories at all levels 68
Fig. 5.1 An example object detection application for autonomous drones 72
Fig. 5.2 Estimated $DSP_{efficiency}$ of PEs for C|K spatial unrolling as a function of PE array size when running the layers of tiny-YOLOv2. L1, L2, ... shows the nine layers of the model 82
Fig. 5.3 Estimated $DSP_{efficiency}$ of PEs for OX|K spatial unrolling as a function of PE array size when running the layers of tiny-YOLOv2. L1, L2, ... shows the nine layers of the model 83
Fig. 5.4 Block diagram of the hardware implementation of the accelerator showing the SIMD array, activation unit, pooling unit, control unit, and input, weight, and output memories 84
Fig. 5.5 Timing diagram showing the simultaneous operation of read, write, and execute, without waiting for complete input data to be available. L_1, L_2, .. represents the layers of the neural network 85
Fig. 5.6 System-level architecture showing the communication infrastructure between the accelerator and external memory (DRAM) of the CPU (ARM A9) 86
Fig. 5.7 Tiling scheme applied to the hardware implementation shown as a pictorial representation of the convolution operation between input feature maps with filters to generate output feature maps 88
Fig. 5.8 Demonstrating tiny-YOLOv2 on Xilinx xc7z035 with bounding box and class probability of a person (left) and airplane (right). The demo achieves an FPS of 11.54 enabling real-time operation 89
Fig. 6.1 Different ML models and their mathematical representation in terms of MMM and MVM. The nested *for*-loop representation can be mapped onto specialized accelerators through spatial and temporal unrolling 96

Fig. 6.2 Overview of the complete TinyVers SoC showing the different power domains (PDs) with their constituting modules and the power modes supported 98
Fig. 6.3 Power simulation of post-synthesis netlist undertaken in Cadence Genus tool for the three power modes. In all three modes, I2S data is collected at a sampling frequency of 44.1 kHz for a window of 2 seconds. Full active power reported includes configuration of uDMA by RISC-V core and interrupt handling procedure, in addition to data collection 99
Fig. 6.4 Flow diagram showing the hierarchical FSM used in the WuC 100
Fig. 6.5 FlexML accelerator architecture overview with ucode instruction 101
Fig. 6.6 Block diagram of the processing elements used in the flexMl accelerator, showing the precision-scalable MAC unit and the additional hardware to support SVM 102
Fig. 6.7 Diagram showing the dataflow reconfiguration used to switch from OX|K dataflow (left) for MMM to C|K dataflow for MVM. The nested *for*-loops below show the addition of parfor loops for the spatial unrolling used 103
Fig. 6.8 Representation of deconvolution layer in software (top left), control unit running the zero-skip operation (bottom left), the architectural change required on the L0 FIFO to support deconvolution (top right), and cycle-by-cycle operation of the FIFO and PEs (bottom right) 105
Fig. 6.9 Blockwise structured sparsity applied to CNN and dense layers (top) and control unit operation in tandem with sparsity index memory to support zero-skipping (bottom) 105
Fig. 6.10 Compiler for ML deployment 107
Fig. 6.11 (Left) The step and hold mechanism and (right) the WuC bypass shift register 108
Fig. 6.12 Measurement setup and chip microphotograph 109
Fig. 6.13 Peak performance analysis of CNN 3×3 layer 109
Fig. 6.14 Power breakdown of the peak perf. analysis with CNN3×3. MRAM power consumption is negligible as it is OFF in active mode. MRAM(A) and MRAM(P) represent MRAM array and MRAM periphery, respectively 109

Fig. 6.15 Energy breakdown showing the distribution of measured energy of the chip modules for running a single inference of the four real-time workloads on FlexML and RISC-V with input data already available in L2 memory. The power and latency measurements start from the setting up of accelerator parameters by RISC-V, data movement from L2 to L1, inference computations, and end with post-processing by the RISC-V core. MRAM power consumption is negligible as it is OFF in active mode 112
Fig. 6.16 Deep sleep power–latency–frequency trade-off 113
Fig. 6.17 Instantaneous power trace showing the KWS application scenario with one full period of smart sensing and TCN processing followed by idling 114
Fig. 6.18 Instantaneous power trace showing the machine monitoring application scenario with one period of smart sensing, FE, and CAE processing followed by idling.............. 115
Fig. 7.1 Computer vision networks exhibit different workloads: the number of parameters per layer grows quadratically, while the feature maps size decreases with the same speed. The result is shallow layers with huge feature maps and small-scale weights and deep layers with a large number of weights but small features. A heterogeneous, reconfigurable architecture is thus required to support different workload characteristics 120
Fig. 7.2 (**a**) 7-nested loop representation of a 2D convolutional layer and of a matrix–matrix multiplication and how different dataflows map the workload on the hardware, (**b**) weight stationary FX, FY, C|K dataflow for AIMC, and (**c**) output stationary OX|K dataflow and C|K dataflows for Digital 122
Fig. 7.3 A design space exploration of the AIMC core for different workloads is carried out (**a**) based on an AIMC core hardware template or (**b**) exploring for different design points by sweeping over the listed AIMC array and activation buffer sizes. Pareto frontier points in (**a**) report the optimal configurations for each network. The stars indicate the configuration selected for the DIANA design 123
Fig. 7.4 By adopting the output pixel unrolling mapping of the array, weights are duplicated OX_u times in the array, while a corresponding number of pixels are computed in parallel (**b**) instead of a single one (**a**). As example (**c**) describes the significant savings in latency that can be achieved for the layers in the first ResBlock of ResNet18 with a 1152×512 array, linearly proportional with the OX_u, with a negligible increase of extra weight loading (**c**) 124

Fig. 7.5 Architectural block diagram of the heterogeneous system 126
Fig. 7.6 Timing diagram of the system under different workloads; each workload reports an indicative number of the time duration of each task, in clock cycles at 270 MHz 127
Fig. 7.7 L1 memory design with detail of the analog core interfaces 128
Fig. 7.8 (**a**) AIMC core block diagram and (**b**) description of a single bitcell in the analog domain (from [167]), and (**c**) reports a timing diagram of the processing stages and their synchronization signals 130
Fig. 7.9 Microarchitecture of the MCU (left) and example of operation for data reuse of the activation buffer 130
Fig. 7.10 Microarchitecture of the output buffer and the SIMD unit 132
Fig. 7.11 Digital core architecture diagram 133
Fig. 7.12 Die photo 134
Fig. 7.13 (Left) Error and power measurements as a function of the AIMC macro operating point for a uniform distribution of accumulation values between [0:4096]; each grid element corresponds to one operating point and the label, its mean percent error on the ADC output. (Right) the most accurate operating point of the macro for different accumulation value ranges 135
Fig. 7.14 Peak performance and efficiencies at system level of the (**a**) complete analog core and (**b**) complete digital core, both including peripheries 136
Fig. 7.15 ResNet20 performance measurement (**a**) and energy efficiency. (**b**) The network achieves low spatial utilization of the AIMC macro, thus not achieving peak efficiencies 137
Fig. 7.16 ResNet18 performance measurement (**a**) and efficiency. (**b**) TOP/s and TOP/s/W are highly dependent on the temporal and spatial utilization of the AIMC macro 137
Fig. 7.17 ResNet18 mapping of the ResBlock layers on the AIMC core, with their relative spatial utilization and required number of weight-write operations. OX unrolling of 4 is used for the ResBlock64 138
Fig. 7.18 By immediate reuse of intermediate data, memory requirements for the activations can be reduced by 7.2×, allowing to keep only a subsection of the feature map in memory as depicted in (**a**); furthermore, by scheduling different layers on different cores, computations can be overlapped in time (**b**) with subsequent latency savings (**c**) 139
Fig. 8.1 Roofline model showing requirements of DNN workloads on NoC bandwidth 144
Fig. 8.2 Overview of traditional NoCs vs proposed AXI NoC 146

Fig. 8.3 Multi-channel architecture of AXI read-and-write interfaces ... 147
Fig. 8.4 Transaction ordering requirement for AXI protocol ... 148
Fig. 8.5 PATRONoC instances as a 2×2 mesh (left) and a 4×4 mesh (right). The AXI masters and slaves are not shown in the 4×4 mesh for ease of readability. Elementary blocks used for the NoC are also shown: XP (bottom-left) and XBAR (bottom-right). Red XP is 3-master and 3-slave, light blue XP is 4-master and 4-slave, and dark blue XP is 5-master and 5-slave ... 149
Fig. 8.6 Implementation results showing area versus bisection bandwidth of PATRONoC and ESP-NoC [74] in 2×2 mesh configurations. PATRONoC's configurations are represented as AXI_AW_DW_IW ... 152
Fig. 8.7 Implementation results showing area vs. bisection bandwidth of PATRONoC in 4×4 mesh configurations (left). Configurations are represented as AXI_AW_DW_IW. Area vs. MOT trade-off for DW = 64 bits (right) ... 152
Fig. 8.8 Uniform Random Traffic with Poisson distribution using Noxim simulator for a 4×4 2D mesh and uniform random traffic on the slim PATRONoC with increasing DMA burst length ... 154
Fig. 8.9 Synthetic traffic patterns for the performance evaluation. (**a**) All global access. (**b**) Max two-hop access. (**c**) Max single-hop access ... 154
Fig. 8.10 Throughput versus injected load for the synthetic random traffic running on the slim and wide PATRONoC using all global access, max 2 hop, and max 1 hop traffic patterns with different DMA burst sizes ... 156
Fig. 8.11 NoC utilization at maximum injected load for the synthetic random traffic running on the slim and wide PATRONoC using all global access, max 2 hop, and max 1 hop traffic patterns with different DMA burst sizes ... 157
Fig. 8.12 Overview of the DNN workloads used for PATRONoC evaluation. FWD and BWD in (**a**) represent the forward and backward propagation workloads used in DNN training. (**a**) Distributed training. (**b**) Convolution parallelized. (**c**) Convolution pipelined ... 158
Fig. 8.13 Throughput analysis for DNN workload traffic on the PATRONoC ... 159

List of Tables

Table 1.1 Models commonly used in machine learning 10
Table 3.1 Surveyed hardware platforms 49
Table 3.2 Flexibility of the surveyed hardware platforms 55
Table 3.3 Power management and autonomy of the surveyed hardware platforms 56
Table 4.1 Memory pool for design space exploration 64
Table 5.1 YOLO networks comparison based on the required accuracy, the number of operations, and memory 78
Table 5.2 YOLO networks comparison based on the required throughput and DSPs 78
Table 5.3 Accuracy of tiny-YOLOv2 with quantization applied using two schemes: PTQ and QAT 79
Table 5.4 Comparison with SoTA in object detection implementation on FPGA 90
Table 6.1 Workload benchmarks 110
Table 6.2 Measurement results of different low power modes 112
Table 6.3 Performance comparison with state-of-the-art 116
Table 7.1 Network end-to-end accuracy and performance summary 139
Table 7.2 State of the art DNN accelerators comparison 140
Table 8.1 Main parameters of the PATRONoC 2D mesh 150
Table 8.2 Comparison of PATRONoC with state-of-the-art NoCs in SoCs .. 160