Efficient Architectures for High-Performance Embedded Computing

Mostafa Rizk

HABILITATION A DIRIGER DES RECHERCHES

HDR

L'UNIVERSITE BRETAGNE SUD

ECOLE DOCTORALE N° 601
Mathématiques et Sciences et Technologies de l'Information et de la Communication
Spécialité : *STIC*

Par
Mostafa RIZK

Efficient Architectures for High-Performance Embedded Computing

Thèse présentée et soutenue à Brest, le 18/10/2022
Unité de recherche : Lab-STICC, UMR CNRS 6285 - 2AI team

Rapporteurs avant soutenance :

Ian O'CONNOR	Professeur, Ecole Centrale de Lyon
Maurizio VALLE	Professeur, Université de Gênes
Guy GOGNIAT	Professeur, Université Bretagne Sud

Composition du Jury :

Rapporteurs :	Ian O'CONNOR	Professeur, Ecole Centrale de Lyon
	Maurizio VALLE	Professeur, Université de Gênes
	Guy GOGNIAT	Professeur, Université Bretagne Sud
Examinateurs :	Amer BAGHDADI	Professeur, IMT Atlantique
	Adnan HARB	Professeur, Lebanese International University
	Jean-Philippe DIGUET	Directeur de recherche CNRS, Head of CROSSING - IRL 2010

Contents

List of Figures

List of Tables

Dedication

To my family

Glossary

2D	Two Dimensions
GPP	3^{rd} Generation Partnership Project
5G	5^{th} Generation Mobile Networks
ACT	Approximate Computing Technique
ADL	Architectural Description Language
AI	Artificial Intelligence
AIS	Automatic Identification System
ALU,	Arithmetic Logic Unit
ALTD,	Average-link Token Delay
APTD,	Average-path Token Delay
ASIC	Application-Specific Integrated Circuit
ASIP	Application-Specific Instruction-set Processor
ASP	Application Specific Processors
AWGN	Additive White Gaussian Noise
BF	Butterfly
BNN	Binary Neural Network
CB	Connection Block
CTP	Code transferring packet
CNN	Convolutional Neural Networks
CMEM	Computational Memory
CMOS	Complementary Metal–Oxide–Semiconductor
COCO	Common Object in Context
CPU	Central Processing Unit
CV	Computer Vision
CW	Control Word
DF	Dataflow
DRP	Data Reading Request Packets
DNN	Deep Neural Network
DVB	Digital Video Broadcasting
ECDIS	Electronic Chart Display and Information System
e-skin	Electronic skin
EU	European Union
FIP	FIFO Index Packet
FIR	Finite Impulse Response
FFT	Fast Fourier Transform
FPG	Full Power Gating

FPGA	Field Programmable Gate Array
FLOPS	Floating Point Operations Per Second
FPS	Frames Per Second
GNSS	Global Navigation Satellite System
GPGPU	General-Purpose Computing on Graphics Processing Unit
GPS	Global Positioning System
GPU	Graphics Processing Unit
H-BWN	Hybrid-precision Binary Weight Network
HCNN	Hybrid Fixed-point Binary Convolution Neural Network
HDL	Hardware Description Language
HLS	High-Level Synthesis
HMA	Horizontal Microcoded Architectures
HP	Hewlett-Packard
HRS	High Resistance State
IMC	In-Memory Computing
INS	Inertial Sensors
ISA	Instruction Set Architecture
KPN	Kahn Process Network
kNN	k-Nearest Neighbor
LDPC	Low Density Parity Check Codes
LiDAR	Light Detection and Ranging
LLR	Log Likelihood Ratio
LRS	Low Resistance State
LTE	Long Term Evolution
LTE-A	Long Term Evolution Advanced
MAGIC	Memristor Aided Logic
mAP	Mean Average Precision
M$_p$IP	Mapping Information Packet
MB	Move Based
MBC	Memory-based Computing
MAJ	Memristor-based Majority
MCP	Mapping Confirmation Packet
MIMO	Multiple-Input Multiple-Output
M$_n$IP	Monitoring Information Packet
ML	Machine Learning
MMSE-IC	Minimum Mean Square Error-Interference Canceler
MoC	Model of Computation
MOL	Memristor Overwrite Logic
MPEG	Moving Picture Experts Group
MPSoC	Multi-processor System-on-Chip
MRAM	Magnetoresistive Random-Access Memory
mrFPGA	Memristor-based Reconfiguration mrFPGA
MRL	Memristor Ratioed Logic
MTJ	Magnetic Tunnel Junction
NB-LDPC	Binary-Low Density Parity Check
NI	Network Interface

NISC	No-Instruction-Set-Computer
NoC	Network-on-Chip
NMBC	NoC-Memory Based Computing
NP	Notification Packet
NM	Notifying Memories
NMC	Near-Memory Computing
NVM	Nonvolatile Memory
OCPG	Only Cell Power Gating
OFDM	Orthogonal Frequency Division Multiplexing
OOP	Object Oriented Programming
Orcc	Open RVC-CAL Compiler
PCB	Printed Circuit Board
PE	Processing Element
PG	Power Gating
PIM	Processing-In-Memory
PLDs	Programmable Logic Devices
QoS	Quality of Service
RBF	reconfigurable Butterfly
R-CNN	Region-Convolutional Neural Network
RTOS	Real Time Operating System
RR	Run-time Remapping
RTL	Register Transfer Level
RVC	Reconfigurable Video Coding
SB	Switch Block
SIFT	Scale Invariant Feature Transform
SISO	Single-Input-Single-Output
SLAM	Simultaneous Localization and Mapping
SNN	Sparse Neural Networks
SoC	System-on-Chip
SoP	Sum of Products
SRAM	Static Random-Access Memory
SSD	Single Shot Detector
STT-MRAM	Spin-Transfer-Torque Magnetoresistive Random-Access Memory
STM	Scenario-based Run-time Task Mapping
SVD	Singular Value Decomposition
SVM	Support Vector Machine
TSVM	Tensorial SVM
UAV	Unmanned Aerial Vehicle
VIA	Vertical Interconnect Access
WiMax	Worldwide Interoperability for Microwave Access
WiFi	Wireless Fidelity
X-MRL	Crossbar Memristor Ratioed Logic
YOLO	You Only Look Once

Long Abstract

This Habilitation to supervise research presents the numerous research activities performed since 20114 targeting the development of flexible and efficient architectures for high performance embedded computing.

The presented research activities aim at the realization of flexible and efficient architectures in multitude application domains such as digital communication, data-flow, neural networks, embedded machine learning and embedded vision. These research works have addressed the design and implementation of novel hardware architectures aiming to attain the emergent flexibility requirement, and the ever-increasing requirements of enhanced performance and reduced power consumption and implementation resources. The performed work has targeted the elaboration of new algorithms and hardware architectures using different design paradigms. Several approaches have been adopted including (1) the demonstration of design approaches that lead to better exploiting of resources; (2) making use of optimization techniques such as approximate computing, quantization, and building dedicated hardware architectures; (3) refinement of available algorithms or proposing new algorithms that enhance the performance; (4) exploiting the technological advancements especially related to memristive devices; (5) tackling emergent computing styles such as smart memory architectures, memory-based computing and in-memory computing and (6) benefiting from MPSoC design approaches.

In this context, several research works have been initiated through completed or ongoing research projects, two defended PhD theses and several Master theses. The most significant achievements are presented by grouping them in four sub-themes: (1) Flexible yet efficient architectures for applications in the digital communication domain; (2) Efficient algorithms and architectures for dataflow applications; (3) Efficient and flexible design paradigms based

on emergent memristive devices and (4) Efficient implementations of machine learning algorithms.

In the first theme, the No-Instruction-Set-Computer (NISC) design methodology is explored to design dedicated modules of turbo receiver. Two NISC-based architectures are demonstrated. The obtained results in terms of performance and implementation area confirm the feasibility of adopting NISC concept in designing flexible yet efficient application-specific processors in the domain of digital communications.

In the second theme, the novel Notifying Memories (NM) concept is introduced to eliminate useless accesses to memories. Memories are transferred into masters that notify processors when the data is ready to be processes. The use of the NM concept leads to significant reductions in latency, injection rate, transported flits and switch conflicts. Also, the throughput is highly enhanced while saving power. In addition, a novel remapping algorithm and network-on-chip (NoC)-based architecture design are proposed to cope with the dynamicity of dataflow actors.

In the third theme, memristive devices are adopted to enhance the computing performance. First, memory-based computing on NoC architecture is introduced by making use of emergent non-volatile memories with power-gating capabilities to enhance the performance and attain low power consumption. In addition, the use of memristors to design architectures which combine flexibility and efficiency, has been explored and introduced through the proposal of original architectures that break the limits of the existing ones. The exploration and study have been conducted in three main levels: (1) interconnect level, (2) processing level and (3) memory level.

The fourth theme focuses on the design and implementation of efficient machine learning algorithms targeting the tactile sensing application. In particular, approximate computing techniques and designing of custom hardware accelerators are adopted to reduce the computational complexity and achieve higher performance with low power consumption.

Current research activities focus on embedded computer vision (CV) and artificial intelligence (AI) with the goal of achieving efficient implementations on edge devices with low computational resources and low power budget. The application domains are real-time drone navigation, marine object detection using deep learning and multi-view object detection. Re-

sults achieved so far are novel and very encouraging.

Concerning the research perspectives, in the short and medium term, several perspectives are proposed: interactive machine learning on edge devices for object detection application, distributed learning on connected devices and tiny machine learning. In the longer term, it is planned to focus on building memory centric low-power processor.

Résumé long

Ce mémoire d'habilitation à diriger des recherches présente mes nombreuses activités de recherche menées depuis 2014 visant le développement d'architectures flexibles et efficaces pour le calcul embarqué haute-performance.

Les activités de recherche présentées visent la proposition et la conception d'architectures flexibles et efficaces dans plusieurs domaines applicatifs tels que la communication numérique, les flots de données, les réseaux de neurones, l'apprentissage automatique et la vision embarquée. Ces travaux de recherche ont porté sur la conception et la mise en œuvre de nouvelles architectures matérielles visant à répondre aux exigences émergentes en matière de flexibilité, ainsi qu'aux exigences toujours plus grandes en termes de performances et de réduction de la consommation d'énergie et des ressources matérielles. Plusieurs approches ont été explorées, notamment : (1) la démonstration d'approches de conception permettant une meilleure exploitation des ressources, (2) l'utilisation de techniques d'optimisation telles que le calcul approximatif, la quantification et la construction d'architectures matérielles dédiées, (3) l'optimisation d'algorithmes existants ou la proposition de nouveaux algorithmes améliorant les performances, (4) l'exploitation des avancées technologiques, notamment en ce qui concerne les dispositifs memristifs, (5) la prise en compte des paradigmes de calcul émergents tels que le calcul en mémoire et les architectures de mémoire intelligente et (6) la mise à profit des approches de conception de MPSoC.

Dans ce contexte, plusieurs travaux de recherche ont été initiés à travers des projets de recherche terminés ou en cours, deux thèses de doctorat soutenues et plusieurs thèses de Master. Les réalisations les plus significatives sont présentées en les regroupant en quatre sous-thèmes : (1) Architectures flexibles et efficaces pour des applications dans le domaine des communications numériques, (2) Algorithmes et architectures efficaces pour des applica-

tions de flot de données, (3) Paradigmes de conception efficaces et flexibles basés sur des dispositifs memristifs émergents et (4) Implémentations efficaces d'algorithmes d'apprentissage automatique.

Dans le premier thème, la méthodologie de conception No-Instruction-Set-Computer (NISC) est explorée pour concevoir des accélérateurs dédiés aux récepteurs itératifs. Deux architectures basées sur NISC sont proposées et conçues. Les résultats obtenus en termes de performance et de surface confirment la pertinence de l'adoption du concept NISC dans la conception de processeurs flexibles et efficaces dans le domaine des communications numériques.

Dans le deuxième thème, le nouveau concept de Notifying Memories (NM) est introduit pour éliminer les accès inutiles aux mémoires. Les mémoires sont transposées dans des maîtres qui informent les processeurs lorsque les données sont prêtes à être traitées. L'utilisation du concept NM conduit à des réductions significatives de la latence, du taux d'injection, les flits transportés, des conflits de commutation. De plus, le débit est fortement amélioré tout en réduisant la consommation énergétique. En outre, un nouvel algorithme de remapping et une architecture basée sur un réseau-sur-puce (NoC) sont proposés pour faire face à la dynamicité des acteurs du flot de données.

Dans le troisième thème, les dispositifs memristifs sont explorés. Tout d'abord, une nouvelle architecture de NoC avec du calcul basé sur mémoire est introduite en utilisant des mémoires non-volatiles pour améliorer les performances et réduire la consommation énergétique. En outre, l'utilisation des memristors pour concevoir des architectures combinant flexibilité et efficacité a été explorée et introduite par la proposition d'architectures originales qui dépassent les limites des architectures existantes. L'exploration et l'étude ont été menées à trois niveaux : (1) au niveau des interconnexions, (2) au niveau du traitement et (3) au niveau de la mémoire.

Le quatrième thème porte sur la conception et la mise en œuvre d'algorithmes efficaces d'apprentissage automatique ciblant l'application de détection tactile. En particulier, des techniques de calcul approximatif et la conception d'accélérateurs matériels dédiés sont investiguées pour réduire la complexité du calcul et atteindre des performances plus élevées avec une faible consommation d'énergie.

Mes activités de recherche actuelles se concentrent sur la vision par ordinateur et l'intelligence artificielle embarquées dans le but de réaliser des implémentations efficaces sur des dispositifs embarqués avec de faibles ressources de calcul et un faible budget énergétique. Les domaines d'application concernent la navigation de drones en temps réel, la détection d'objets marins par apprentissage profond et la détection d'objets avec multi-vues. Les résultats obtenus jusqu'à présent sont originaux et très encourageants.

En ce qui concerne les perspectives de recherche, à court et à moyen terme, plusieurs pistes sont proposées : l'apprentissage automatique interactif sur les dispositifs embarqués pour les applications de détection d'objets, l'apprentissage distribué sur dispositifs connectés et l'apprentissage machine embarqué (TinyML). À plus long terme, il est prévu d'explorer la construction de processeurs basse consommation centrés sur la mémoire.

Preface

This document represents my Habilitation thesis manuscript. It summarizes my academic and professional career including recent and future research activities. It is organized as follows:

The first part includes my detailed Curriculum Vitae. It includes 3 chapters. Chapter 1 presents my personal resume. Chapter 2 and Chapter 3 provide my research and teaching experiences respectively.

The second part gives a high level view on my previous research activities. It includes 4 chapters:

- Chapter 4 presents the first research topic related to the proposal of flexible and efficient architectures for applications in digital communication.
- Chapter 5 presents the second research topic related to efficient algorithms and architectures for dataflow applications.
- Chapter 6 presents the third research topic that tackles the use of memristive devices in realizing efficient hardware architectures.
- Chapter 7 presents the fourth research topic about designing efficient hardware architectures for machine learning algorithms.

The third part provides a brief review of current research work (Chapter 8) and long-term future work (Chapter 9).

Part I

Detailed Curriculum Vitae

Chapter 1

Resume

Current professional address

Lab-STICC UMR CNRS 6285
IMT Atlantique – MEE Department
Technopôle Brest-Iroise — CS 83818
29238 Brest Cedex 3
France
Phone : +33 2 29 00 15 95
Fax : +33 2 29 00 11 84
Email : mostafa.rizk@imt-atlantique.fr
Web : https://sites.google.com/liu.edu.lb/mostafa-rizk/

1.1 Summary

I received my Maitrise degree in Electronics, M.Sc in Biomedical Physics, and M.Sc in Signal, Telecom, Image, and Speech from the Lebanese University in 2007, 2008 and 2010 respectively. Then, I pursued my Ph.D. degree in Sciences and Technologies of Information and Communication from IMT Atlantique (former Telecom Bretagne) in France in 2014.

Then, I was a post-doctoral researcher at University of Bretagne Sud (UBS) and the Laboratory of Science and Technology of Information, communication, and Knowledge (Lab-STICC), CNRS, UMR 6285, in Lorient, France. Later in 2016, I joined the Lebanese International University in Lebanon as an assistant and then associate professor at the Computer and Communication Engineering Department. Currently, I am working at Lab-STICC UMR 6825 CNRS/IMT Atlantique in Brest, France where I am conducting my research in the domain of embedded deep learning and embedded computer vision.

My research activities have been conducted at Lab-STICC. In particular, I performed my research work with in three teams: the team specialized in the Interaction between Algorithm and Silicon (IAS), the team specialized in Methods, tOols, Circuits and Systems (MOCS),

and the team specialized in Algorithm Architecture Interactions (2AI) in collaboration with research laboratories in Lebanon.

The performed research work has focused on the hardware/software co-design for embedded architectures, circuits and systems. It stands at the crossing point between development and refinement of algorithms and designing digital embedded architectures. My conducted research has been supported in several projects and has been valorized through a number of publications in well reputable journals and through international and national conferences.

Since I defended my PhD in the end of 2014, which addressed the topic of designing novel flexible and efficient implementations for applications in the domain of digital communication, I have been doing research consistently in the domain of embedded systems targeting the realization of flexible and efficient architectures with reduced power implementations in multitude application domains. My research work has tackled emergent topics such as smart memory architectures, memory-based computing and in-memory computing. An important work has been done to investigate adopting novel non-volatile memory technologies and memristive devices to achieve flexible architectures with reduced power consumption. My research work has addressed the design and implementation of novel hardware architectures for applications in the fields of data-flow, embedded machine learning, embedded vision, and embedded intelligence aiming to attain the emergent flexibility requirement, and the ever increasing requirements of enhanced performance and reduced power consumption. The performed work has targeted the elaboration of new algorithms and hardware architectures in these fields.

During the last seven years , I have been involved in a number of research projects. I participated in the supervision of two successfully defended Ph.D theses. I was the direct tutor for several Master theses and undergraduate projects. Also, I served as member in several evaluation committees and participated in a number of theses juries.

My teaching activities lay in the domains of electronics, computer and communication engineering and informatics for undergraduate and graduate levels.

1.2 Professional Experiences

2021–present	Contractual researcher at Lab-STICC, CNRS, UMR 6285, France
2017–present	Instructor at the Lebanese University, Faculty of Sciences, Physics and Electronics Department, Hadat, Lebanon
2016–present	Associate professor at the Lebanese International University (LIU), School of Engineering, Department of Communication and Computer Engineering, Beirut, Lebanon
2017–2020	Associate researcher at Institute Mines-Télécom, IMT Atlantique, Brest, France
2015–2016	Post-doctoral fellow at UBS, Lorient, France and Methods, tOols, Circuits and Systems (MOCS) research team at Lab-STICC
2014–2015	Research Engineer at UBS, Lorient, France and MOCS research team at Lab-STICC
2011–2012	Instructor at the American University of Culture and Education (AUCE), Lebanon, Faculty of Arts and Sciences, Department of Computer Sciences
2009–2010	Six-month research internship at advanced data management company (ADM), Lebanon
2007–2008	Six-month professional internship at quality department and biomedical engineering department at RAH, Lebanon
2008–2011	Freelancing engineering projects (electronics, embedded systems and assembly)

1.3 Educational Achievements

2011–2014 Doctorate's degree

Organizations: Institute Mines Télécom, IMT Atlantique (former Télécom Bretagne), France and Lebanese University

Laborartory: Laboratory of Science and Technology of Information, Communication and Knowledge (Lab-STICC) CNRS, UMR 6285

Major: Information and Communication Sciences and Technologies and Electronics and Telecommunication

Distinction: Very honorable

2009–2010 Research Master's degree

Organization: Lebanese University, Doctoral School of Sciences and Technology, Lebanon

Major: Signal, Telecoms, Image and Speech (STIP)

2008–2009 Professional Master's degree

Organization: Lebanese University, Faculty of Sciences, Lebanon

Major: Biomedical physics; Option: Quality control

2006–2007 Teaching diploma in sciences

Organization: Lebanese University, Faculty of Sciences, Lebanon

Major: Physics; Option: Electronics

2003–2006 License degree of science

Organization: Lebanese University, Faculty of Sciences, Lebanon

Major: Physics; Option: Electronics

2002–2003 Baccalaureate in sciences

1.4 Training Courses

- **Scientific Computing Accelerated on FPGA**: Three-day training on Vitis HLS (AMD-XilinX) and OneAPI (Intel) to accelerate scientific computing on FPGAs at Maison de la Simulation, Saclay, France.
- **Developing Accelerators using Vitis and PYNQ**: Full-day tutorial about Vitis development flow for compute acceleration on Xilinx FPGA; Xilinx University Program
- **Channel Coding**: Five-day training course about channel coding at Télécom Bretagne, Brest, France.
- **Networks and Telecommunications**: Twenty-hour training course about advanced architectures of stationary and mobile telecom at Doctoral School of Sciences and Technology, Lebanese University, Lebanon.

- **Information Systems**: Twenty-hour training course about information systems at Doctoral School of Sciences and Technology, Lebanese University
- **Scientific English**: Five-day training course about scientific English language at Télécom Bretagne, Brest, France.
- **Design Methodologies for Adaptive Circuits and Systems**: Full-day tutorial about designing adaptive circuits at IEEE Automation and Test in Europe Conference (DATE'13).

1.5 Synergistic Activities

2016–2017	Member of the evaluation committee of SoC exhibition at LIU, Lebanon
2012–2014	Member of planning committee of IEEE student branch at Télécom Bretagne
2008–2009	Participation in organizing the PhD student day of ED-SICMA, Brest, France

Chapter 2

Research Experiences

2.1 Research Topics

My research topics are mainly focused on the hardware/software co-design. My main interests stand at the crossing point between algorithm development and digital architecture design for embedded systems including but not limited to:

- Digital design of flexible processing systems
- Intellectual Properties for wireless communication
- Hardware/software implementations
- Application Specific Processors (ASP)
- Network-on-Chip (NoC) design
- Embedded systems design in the fields of audio, video, and data-flow domains
- Embedded Multi-processor System-on-Chip (MPSoC) architectures
- Embedded systems based on emerging non-volatile memory technologies
- Memory-based computing, near-memory computing and in-memory computing
- Algorithm development and refinement for digital base-band components
- Algorithm development for task scheduling on MPSoC
- Embedded computer vision
- Embedded machine learning

2.2 Visiting Scholar

December 2019	visiting professor, UBS and Lab STICC, Lorient, France
September 2018	visiting professor, IMT-Atlantique, Brest, France
September 2017	visiting professor, Lab-STICC CNRS 6285, Brest, France

2.3 Laboratory Membership

2010–2014	member of TTS research group at Hasan Kamel Sabbah Lab at the Lebanese University - Faculty of Sciences, Hadat, Lebanon
2011–2014	member of the research team specialized in the Interaction between Algorithm and Silicon (IAS) in Lab-STICC, CNRS, UMR 6285, Brest, France
2014–2016	member of the research team specialized in Methods, tOols, Circuits and Systems (MOCS) in Lab-STICC, CNRS, UMR 6285, Brest, France
2016–2017	member of the research team specialized in Electronics in EEE department at the Lebanese International University, Beirut, Lebanon
2017–present	member of the research team specialized in Algorithm Architecture Interactions (2AI) in Lab-STICC, CNRS, UMR 6285, Brest, France

2.4 Research Collaboration

The following list enumerates the institutes where I share research activities:

- CROSSING - IRL 2010, Australia
- University of Genoa, Italy
- Université Bretagne Sud, France
- Tohoku University, Japan
- American University of Culture and Education, Lebanon
- American University of Beirut, Lebanon
- Lebanese University, Lebanon
- Lebanese International University, Lebanon

2.5 Involvement in Research Projects

I participated in the following research projects:

1. **Mobile and wireless communications Enablers for Twenty-twenty Information Society (METIS)**:
   - **Goal**: aims to lay the foundation of 5G, the next generation mobile and wireless communications system
   - **Funding**: co-funded by the European Commission as an Integrated Project under the Seventh Framework Program for research and development (FP7)
   - **My Role**: PhD student
2. **Design Oriented Model of Computation for Embedded and Adaptive Multiprocessor (COMPA)**:
   - **Goal**: aims to propose generic models for adaptive multi-processors embedded systems
   - **Partners**: gathers three laboratories in Bretagne region (Lab-STICC, IETR, and IRISA)
   - **Funding**: funded by the French National Research Agency (ANR, project ANR-11-INSE-0012)
   - **My Role**: Research Engineer
3. **Hierarchical Hardware Architectures Based on Associative-Memories and Network on Chips for Cyber-Attack Detection in Telecom Networks (CyAM)**:
   - **Goal**: aims to exploit associative-memories and network on chips in Cyber-Attack detection
   - **Partners**: conducted as collaboration between Lab-STICC and University of Tohoku in Japan
   - **Funding**: funded by Bretagne region, France
   - **My Role**: Post-doc Researcher
4. **Optimization of Observation, Detection, Classification and Tracking of Seaships for Maritime Security by Mean of Embedded Deep Learning (ODESSA)**
   - **Goal**: aims to offer a new optoelectronic system that can detect a maritime security problem as early as possible while costing less than a radar tool
   - **Partners**: conducted as collaboration between Lab STICC (IMT Atlantique and UBS) and two French Companies: Exavision and Inpixal
   - **Funding**: funded by Bretagne region, France
   - **My Role**: Post-doc Researcher

5. **Deployment of electro-optical sensors and drones for living human detection and localization in search and rescue missions**

    - **Goal**: aims to demonstrate a novel low-cost solution that makes use of the availability and the affordability of drones to enhance detecting and locating wounded and lost individuals during and after disasters.
    - **Funding**: funded by the Lebanese University
    - **My Role**: Project technical leader

2.6 Publications

The results of my research activities led to the following publications:

- 1 Ph.D. thesis
- 2 Master theses
- 2 Book chapters
- 18 Journal papers
- 34 international conference papers
- 5 national conference papers

Further information about the publications is detailed in Appendix A.

2.7 Supervision and Tutorship

I have been involved in supervision and tutorship of 2 Ph.D. theses, 24 Master theses and 14 graduation projects.

2.7.1 Supervision of Ph.D. Theses

1. **Supervision of the Ph.D. thesis of Dr. Hamoud Younes**
   Post doctoral fellow at IMT Atlantique, Brest, France

    - **Title**: Embedded machine learning emphasis on hardware accelerators and approximate computing for tactile data processing
    - **Academic Years**: 2018-2021
    - **University**: University of Genoa, Italy

2. **Supervision of the Ph.D. thesis of Dr. Khaled Alhaj Ali**
   Post doctoral fellow at IMTAtlantique, France

- **Title**: New design approaches for flexible architectures and in-memory computing based on memristor technologies
- **Academic Years**: 2016-2020
- **University**: IMT Atlantique, Brest, France

2.7.2 Supervision of Master Theses

Since 2015, I have supervised the several Master theses in different universities. The following list presents these theses classified according to universities and the Master's programs.

1. **Supervision of Master theses at Lebanese University, Lebanon**:
   - 4 Master theses in the program of the professional Master's degree of Sciences in Field Electro-Mechanical Engineering
   - 1 Master thesis in the program of the professional Master's degree of Sciences in Medical Physics and Imaging Technologies
   - 3 Master theses in the program of the research Master's degree of Sciences in Medical Physics and Life Imaging
   - 1 Master thesis in the program of the research Master's degree of Sciences in Field Electro-Mechanical Engineering
   - 1 Master thesis in the program of the research Master's degree of Sciences in Signal, Telecoms, Image, and Speech (STIP)
   - 1 Master thesis in the program of the research Master's degree of Sciences in Information System and Big Data Intelligence
2. **Supervision of Master theses at Lebanese International University, Lebanon**:
   - 11 Master theses in the program of the Master's degree of Sciences in Computer and Communication Engineering
   - 2 Master theses in the program of the Master's degree of Sciences in Electrical and Electronics Engineering
3. **Supervision of Master theses at IMT Atlantique, Mathematical and Electrical Engineering Department, France**:
   - 2 internships in the program of Engineering in Embedded Systems
   - 1 internship in the field of embedded artificial intelligence collaboration with the Lebanese University
4. **Supervision of Master theses at Lab STICC, Lorient, France**:
   - 1 internship in the field of Embedded Systems

2.7.3 Supervision of Graduation Projects

Since 2016, I have supervised several graduation projects in different universities. The following list presents these projects classified according to the universities and majors.

1. **IMT Atlantique, Mathematical and Electrical Engineering Department, France**
    - 1 graduation project in the program of the Bachelor's degree of Engineering in Embedded systems
2. **Lebanese International University (LIU), School of Engineering, Lebanon**
    - 1 graduation project in the program of the Bachelor's degree of Sciences in Electronics Engineering
    - 10 graduation projects in the program of the Bachelor's degree of Sciences in Computer Engineering
3. **International University of Beirut (BIU), School of Engineering, Lebanon**
    - 2 graduation projects in the program of the Bachelor's degree of Engineering

Further information about my supervision experience is detailed in Appendix B.

2.8 Other Responsibilities and Activities

2.8.1 Reviewing Activities

1. **Scientific journals**:
    - IEEE Transactions on Signal Processing
    - IEEE Transactions on Circuits and Systems (TCAS-I)
    - IEEE Transactions on Circuits and Systems-Part II (TCAS-II)
    - IEEE Access
    - IEEE Embedded Systems Letters
    - Elsevier Embedded Hardware Design journal (Microprocessors and Microsystems)
    - International Journal of Embedded Systems (IJES)
    - Journal of Systems Architecture
    - Journal of Aerospace Technology and Management
2. **International conferences**:
    - IEEE Design, Automation and Test in Europe (DATE)
    - IEEE International New Circuits and Systems Conference (NewCas)

- IEEE International Conference on Communications and Information Technology (ICCIT)
- IEEE International Conference on Microelectronics (ICM)
- IEEE New Generation of Circuits and Systems Conference (NGCAS)
- IEEE International Conference on Computer and Applications (ICCA).
- IEEE International Conference on Electronics Circuits and Systems (ICECS).
- IEEE International Symposium on Rapid System Prototyping (RSP)
- IEEE Microwave Theory and Techniques in Wireless Communications (MTTW)
- International Conference on System-Integrated Intelligence (SysInt)
- IEEE International Conference on Innovation and Intelligence for Informatics, Computing and Technologies (3ICT)
- The International Conference on Smart Systems and Power Management (IC2SPM)

2.8.2 Technical Program Committee Member

I have served as technical program committee (TPC) member in the following international conferences and workshops:

- IEEE International Conference on Microelectronics (ICM 2013)
- IEEE International Conference on Electronics Circuits and Systems (ICECS 2019)
- IEEE International Conference on Innovation and Intelligence for Informatics, Computing and Technologies (3ICT 2020)
- IEEE International Symposium on Rapid System Prototyping (RSP 2021)
- IEEE Microwave Theory and Techniques in Wireless Communications (MTTW 2021)
- IEEE International Symposium on Rapid System Prototyping (RSP 2022)
- IEEE Microwave Theory and Techniques in Wireless Communications (MTTW 2022)
- Workshop on Maritime Computer Vision (WACV 2023)

2.8.3 Committee Membership

I have served as a committee member for evaluating Master theses and graduation projects in different majors at different universities. The following list shows my participation in evaluation committees classified according to university, level and major:

1. **Lebanese University (LU), Faculty of Sciences (FS), Lebanon**:

- 4 Master theses in the program of the professional Master's degree of Sciences in Field Electro-Mechanical Engineering (FEME)
- 2 Master theses in the program of the professional Master's degree of Sciences in Medical Physics and Imaging Technologies (MPIT)
- 5 Master theses in the program of the research Master's degree of Sciences in Medical Physics and Life Imaging (MPLI)

2. **Lebanese International University (LIU), School of Engineering (SoE), Lebanon**:

   - 9 Master theses in the program of the Master's degree of Sciences in Computer and Communication Engineering (MCCE)
   - 10 graduation projects in the program of the Bachelor's degree of Sciences in Computer Engineering (CENG)

Further information about my teaching experience is detailed in Appendix C.

Chapter 3

Teaching Experiences

3.1 Teaching and Administrative Responsibilities

I have worked as course coordinator for several courses at different universities. Course coordination involves following up the course progress and examination process in different branches and sections. The following list presents the courses that I have coordinated:

- Introduction to Mechatronics course in the program of the Master's degree of Science in Electrical and Electronics Engineering at School of Engineering at Lebanese International University
- Introduction to Computer course in the program of the Bachelor's degree in Computer at School of Arts and Sciences the American University of Culture and Education
- Microprocessors and Interfaces course in the program of the Bachelor's degree in Computer at School of Arts and Sciences the American University of Culture and Education

3.2 Construction and Development of University Courses

I have worked on remodeling and development of several courses at different universities. The reconstruction of a course includes the following duties: developing the syllabus, enhancing the course material, choosing the textbooks and updating the course assessment (exams and assignments).

- Microprocessors and Interfaces course in the program of the Bachelor's degree in Computer at School of Arts and Sciences the American University of Culture and Education
- Introduction to Computer course in the program of the Bachelor's degree in Computer at School of Arts and Sciences the American University of Culture and Education
- Introduction to Mechatronics course in the program of the Master's degree of Science in Electrical and Electronics Engineering at School of Engineering at Lebanese International University

- Contract Administration and Project Management course in the program of the Professional Master's degree of Science in Field Electromechanical Engineering at the Faculty of Sciences at Lebanese University

- Neural Network and Deep Learning course in the Image-Signal Module in the program of the Research Master's degree of Science in Medical Physics and Living Imaging at the Faculty of Sciences at Lebanese University

- Digital Electronics course in the program of the Master I degree of Science in Fundamental Physics (Biomedical Physics Course) at the Faculty of Sciences at Lebanese University

3.3 Teaching University Courses

3.3.1 Teaching E Overview

Since 2011, I have taught several courses in the programs of various degrees in the following universities:

1. Lebanese University, Faculty of Sciences, Lebanon:
    - Research Master's degree of Sciences in Signal, Telcom, Image and Speech
    - Professional Master's degree of Sciences in Field Electro-Mechanical Engineering
2. Lebanese University, Institute of Social Sciences, Lebanon:
    - Bachelor of Social Sciences
3. Lebanese International University (LIU), School of Engineering, Lebanon
    - Bachelor's degree of Sciences in Computer Engineering
    - Bachelor's degree of Sciences in Electrical Engineering
    - Bachelor's degree of Sciences in Electronics Engineering
    - Master's degree of Sciences in Computer and Communication Engineering
4. American University of Culture and Education (AUCE), Lebanon
    - Bachelor's degree of Sciences in Computer Sciences

Further information about my teaching experience is detailed in Appendix D .

3.3.2 Teaching Summary

3.3.2.1 Common core courses

I have participated in teaching several courses and practical work related to electric circuits, electronics and fundamental logic circuits. During my academic career and for several academic years, I have taught "Electric Circuits", "Fundamentals of Digital Logic Design" and "Signals and Systems" courses, where I have applied the concepts in a fun and operational way using software simulators and basic hardware platforms.

Note that in my previous teaching work, I have participated in developing the plan of study to include the "Introduction to Engineering" which aims to teach engineering students how to solve a problem and implement the solution through building simple algorithms and implementing them on available hardware platforms, in particular Arduino toolset and its compatible sensors. Several teaching methods have been adopted especially project-based teaching in order to give the students the opportunity to develop knowledge and skills through engaging projects set around real-life challenges and problems.

3.3.2.2 Courses in the field of embedded systems

I have a long experience in teaching courses about processors and microcontrollers, advanced digital logic systems, hardware description languages, computer architecture and embedded systems for under graduate and graduate levels.

I have taught "Advanced Logic Design" course that introduces designing of logic elements used in computations targeting Programmable Logic Devices (PLDs) and Field Programmable Gate Array (FPGA) devices. It aims to teach how to design, in VHDL, several modules such ad decoders, memories, ALU, multiplexers, registers, etc. Students are directed to use ModelSim in order to verify the designed modules. Also, I contributed in developing a practical course for graduate level about designing MIPS processors from scratch. The course aims to teach how to design, in VHDL, the sub-modules of the processor as well as the instruction set. Students are directed to use ModelSim in order to verify the designed processors.

I participated in teaching the technical work of the "Signals and Image Processors" course in the research Master program in Signals, Telecom, Image and Speech (STIP). This course focuses on implementing image processing techniques on FPGAs from algorithm till the hardware realization. In particular, the course introduces the design and implementation of Sobel filter on Spartan-6 FPGA.

I have taught the "Computer Architecture" course for graduate level. The course aims to introduce essential computer architecture design and analysis techniques. The course treats fundamental methods used to improve performance of microprocessors like pipelining, caches hierarchies, superscalar processors and out-of-order execution. This graduate course also considers parallel processing topics at different levels (instruction-level parallelism, data-level parallelism and thread-level parallelism). Scheduling and fault-tolerance strategies are also covered in this course.

For several years, I have taught microprocessors and microcontrollers courses, which target the microchip PIC, ARM-based and Atmel microcontrollers. In general, these courses introduce the architectures of the targeted processors and their corresponding instruction sets. Moreover, these courses introduce programming of the microcontrollers including looping, branching, arithmetic and logical operations, timers, interrupts, communication protocols (UART, I2C, SPI, etc.) and controlling input-output pins. Students are directed to use the the associated software development tools (MPLab, CCS, STM32CubeMX, Arduino IDE, etc.) and software simulation tools - such as Proteus - to develop projects

Furthermore, I have taught the "Embedded Systems" course for graduate students which aims to provide the students with the basic information about embedded systems. The first part of this course introduces in details the ARM-based/AVR microcontrollers. The second part is devoted to the programming of embedded systems, where the students learn to design and develop a programmable embedded platform from scratch and interface a variety of sensors. In addition, the course cover multitasking by introducing real time operating system (RTOS) and using FreeRTOS to apply Queues, Semaphore and Mutex targeting Atmel-based processors (integrated in Arduino platforms) and ARM-based processors (integrated in STM32 platforms). Moreover, I have taught "Operating Systems" course and the "Linux" technical lab targeting Raspberry Pi platforms.

3.3.2.3 Courses in the field of digital communication

Since 2016, I have taught the course of "Digital Communication" which is an introduction to modern digital communications at a graduate/senior undergraduate level. The coverage emphasizes a conceptual understanding of principles, techniques, and fundamental limits in digital communication systems. This course covers modulation for digital communications over additive white Gaussian noise (AWGN) channels; bandpass and low-pass signal representation; signal space representation of waveforms; modulation; demodulation; optimum receivers for AWGN channels; probability of error analysis; channel coding; synchronization; an introduction to digital communication through band-limited channels.

The course has been further developed in 2018 to include, in addition to theoretical concepts, direct applications of the theoretical concepts using MATLAB. It gives hands-on experience in translating digital communication concepts into software-defined radio technology using MATLAB coding of the different basic blocks of a digital communication system. Students can experiment a complete functional digital communication system by putting all the designed blocks together. Note that the Lab course of this work includes implementing the algorithms using USRPs from national instruments using LabView.

3.3.2.4 Courses in the field of artificial intelligence and computer vision

I have taught the "Multimedia Networks" course for graduate level. The course aims to introduce different media types such as audio, image and video showing their characteristics and the techniques used in coding along with the corresponding available standards. Also, I have taught for graduate level the course of "Image Processing and Computer Vision" that includes the topics of filtering, transformation, edge detection, feature extraction and the use of neural networks and deep learning in computer vision.

I have initiated the teaching course “Neural Network and Deep Learning” in the Image-Signal Module in the program of the Research Master’s degree of Science in “Medical Physics and Living Imaging” at the Faculty of Sciences at Lebanese University.

3.3.2.5 Courses in the field of software engineering

I have taught practical course about software applications using object oriented programming (OOP). I teach an introductory course about relational database that includes a practical work that enables students to design an application embedding a database from scratch (SQL, phpmyadmin, Java, etc.).

3.3.2.6 Courses in the field of mechatronics

I have developed a course to introduce mechatronics to electronics engineers. The course scope is to practically lead students to collect data from different sensors, analyze, and take decisions to control electromechanical actuators. The course introduces mechanical mechanisms (gears, pulleys, belts, pinions, screws, etc.); motion sensors (GPS, encoders, accelerometers, speed sensors, proximity sensors, etc.) and actuators (motors, servo motors, solenoids, etc.). I have organized practical tutorials for students that demonstrate the interaction between embedded systems and electromechanical system.

3.4 Technical Training Courses

I provide technical training courses in the domain of basic and advanced programming, telecommunication and electronics (logic, digital systems, OOP programming, circuit design, and PCB assembly, FPGAs) to engineers, university and technical institutes students. These lectures are presented during private training sessions and workshops.

Appendix A

List of publications

A.1 Book chapters

[BC2] **M. Rizk**, F. Slim, A. Baghdadi, and J-Ph. Diguet "Towards Real-time Human Detection in Maritime Environment Using Deep Learning", in Advances in System-Integrated Intelligence. SYSINT 2022. Lecture Notes in Networks and Systems, vol 546. Springer, Cham, pp. 583-593. DOI: 10.1007/978-3-031-16281-7_55

[BC1] H. Younes, M. Alameh, A. Ibrahim, **M. Rizk** and M. Valle, "Efficient Algorithms for Embedded Tactile Data Processing", in Electronic Skin: Sensors and Systems, River Publishers, 2020, pp.113-138.

A.2 Published journal papers

[J18] **M. Rizk**, K. Martin and J-Ph. Diguet, "Run-time remapping algorithm of dataflow actors on NoC-based heterogeneous MPSoCs", in IEEE Transactions on Parallel Distributed Systems (TPDS), 2022. DOI: 10.1109/TPDS.2022.3177957

[J17] K. Haj-Ali, A. Baghdadi, E. Dupraz, M. Léonardon, **M. Rizk** and J-Ph. Diguet, "MOL-based In-Memory Computing of Binary Neural Networks", in IEEE Transactions on Very Large-Scale Integration (VLSI) Systems, 2022. DOI: 10.1109/TVLSI.2022.3163233

[J16] H. Younes, A. Ibrahim, **M. Rizk** and M. Valle, "An Efficient Selection-Based kNN Architecture for Smart Embedded Hardware Accelerators," in IEEE Open Journal of Circuits and Systems (OJCAS), vol. 2, pp. 534-545, 2021. DOI: 10.1109/OJCAS.2021.3108835

[J15] **M. Rizk**, A. Baghdadi and M. Jézéquel, "A literature survey on algorithms and hardware architectures of Max-Log-MAP demapping", Journal of Circuits, Systems, and Computers (JCSC), vol. 31, no. 3, p. 2230001, 2021. DOI: 10.1142/S021812662230001X

[J14] H. Younes, A. Ibrahim, **M. Rizk** and M. Valle, "A Shallow Neural Network for Real-Time Embedded Machine Learning for Tensorial Tactile Data Processing," in IEEE Transactions on Circuits and Systems I: Regular Papers, vol. 68, no. 10, pp. 4232–4244, 2021. (DOI: 10.1109/TCSI.2021.3102303)

[J13] K. Haj-Ali, **M. Rizk**, A. Baghdadi, J-Ph. Diguet and J. Joumaa, "Hybrid memristor-CMOS implementation of combinational logic based on X-MRL" in Electronics, vol. 10, no. 9, April 2021. DOI: 10.3390/electronics10091018

[J12] **M. Rizk**, A. Baghdadi, M. Jézéquel, Y. Mohanna, and Y. Atat, "No-Instruction-Set-Computer Design Experience of Flexible and Efficient Architectures for Digital Communication Applications: two case studies on MIMO Turbo Detection and Universal Turbo Demapping", in Journal of Design Automation for Embedded Systems (DEAM), January, 2021. DOI: 10.1007/s10617-021-09245-x

[J11] H. Younes, A. Ibrahim, **M. Rizk** and M. Valle, "Algorithmic-Level Approximate Tensorial SVM Using High-Level Synthesis on FPGA," in Electronics, vol. 10, no. 2, January, 2021. DOI: 10.3390/electronics10020205

[J10] K. Haj-Ali, **M. Rizk**, A. Baghdadi, J-Ph. Diguet and J. Joumaa, "Memristive Computational Memory Using Memristor Overwrite Logic (MOL)", in IEEE Transactions on Very Large-Scale Integration (VLSI) Systems, August, 2020. DOI: 10.1109/TVLSI.2020.3011522

[J9] **M. Rizk**, A. Baghdadi, M. Jézéquel, Y. Mohanna, and Y. Atat, "NISC-Based MMSE-IC MIMO Turbo Detector", in Journal of Circuits, Systems, and Computers (JCSC), vol. 30, no.04, p. 2150069, September, 2020. DOI: 10.1142/S0218126621500699

[J8] **M. Rizk**, A. Baghdadi, and M. Jézéquel, "Computational Complexity Reduction of MMSE-IC MIMO Turbo Detection", in Journal of Circuits, Systems, and Computers (JCSC), vol. 28, no. 13, March, 2019. DOI: 10.1142/S0218126619502281

[J7] J-Ph. Diguet, N. Onizawa, **M. Rizk**, M. J. Sepulveda, A. Baghdadi, and T. Hanyu, "Networked Power-gated MRAMs for Memory-Based Computing", in IEEE Transactions on Very Large-Scale Integration (VLSI) Systems, vol. 26, no. 12, pp. 2696-2708, December, 2018. DOI: 10.1109/TVLSI.2018.2856458

[J6] **M. Rizk**, A. Baghdadi, M. Jézéquel, Y. Mohanna, and Y. Atat, "Efficient Quantization and Fixed-Point Representation for Turbo-Equalization and Turbo-Demapping", in EURASIP Journal on Embedded Systems, vol. 2017:33, December, 2017. DOI: 10.1186/s13639-017-0081-y

[J5] Z. Bazzal, A. M. Ahmad, I. El-Bitar, **M. Rizk** and M. Raad, "Proposition of an Adaptive Retransmission Timeout for TCP in 802.11 Wireless Environments", in International Journal of Engineering Research and Application, vol. 7, Issue 2, (Part -4), February, 2017. DOI:10.9790/9622-0702046471

[J4] **M. Rizk**, A. Baghdadi, M. Jézéquel, Y. Mohanna, and Y. Atat, "Design and Prototyping Flow of Flexible and Efficient NISC-Based Architectures for MIMO Turbo Equalization and Demapping", in Electronics, Special Issue "Rapid System Design with

Dedicated Architectures and Specific Software Tools", vol. 5, Issue 3, August, 2016. DOI:10.3390/electronics5030050

[J3] **M. Rizk**, A. Baghdadi, M. Jézéquel, Y. Mohanna, and Y. Atat, "NISC-Based Soft-In Soft-Out Demapper", in IEEE Transactions on Circuits and Systems II (TCAS-II), vol. 62, Issue 11, November, 2015. DOI: 10.1109/TCSII.2015.2455991

[J2] Y. Atat and **M. Rizk**, "Top Down Approach: SIMULINK Mixed Hardware / Software Design", International Journal of Computer Science Issues, vol. 9, Issue 3, No 3, May, 2012.

[J1] **M. Rizk**, Y. Mohanna, H. Mougnieh, M. Hamad, F. Khalil, and A. Ghaddar, "Arabic Text to Speech Synthesizer : Arabic Letter to Sound Rules, in International Review on Computers and Software (IRECOS), vol. 6 n. 1, January, 2011.

A.3 Published international conferences

[C34] M. Yaacoub, H. Younes and **M. Rizk**, "Acoustic Drone Detection Based on Transfer Learning and Frequency Domain Features", in proc. of the IEEE International Conference on Smart Systems and Power Management (IC2SPM), November, 2022.

[C33] J. Sharafaldeen, **M. Rizk**, D. Heller, A. Baghdadi and J-Ph. Diguet, "Marine Object Detection Based on Top-View Scenes Using Deep Learning on Edge Devices", in proc. of the the IEEE International Conference on Smart Systems and Power Management (IC2SPM), November, 2022.

[C32] D. Heller, M. Rizk, R. Douguet, A. Baghdadi and J-Ph. Diguet, "Marine Objects Detection Using Deep Learning on Embedded Edge Devices", in proc. of the IEEE International Workshop on Rapid System Prototyping (RSP), part of Embedded Systems Week (ESWEEK), Shanghai, October 2022.

[C31] Z. Ning, **M. Rizk**, A. Baghdadi and J-Ph. Diguet, "Enhancing Embedded AI-Based Object Detection Using Multi-View Approach", in proc. of the IEEE International Workshop on Rapid System Prototyping (RSP), part of Embedded Systems Week (ESWEEK), Shanghai, October 2022.

[C30] **M. Rizk**, D. Heller, R. Douguet, A. Baghdadi and J-Ph. Diguet, "Optimization of Deep-Learning Detection of Humans in Marine Environment on Edge Device", in proc. of the IEEE International Conference on Electronics Circuits and Systems (ICECS), Glasgow UK, October 2022.

[C30] **M. Rizk**, A. Baghdadi, and J-Ph. Diguet "Towards Real-time Human Detection in Maritime Environment Using Deep Learning" accepted to International Conference on System-Integrated Intelligence (SysInt 2022), Genova, Italy, September, 2022.

[C29] **M. Rizk**, F. Slim and J. Charara, "Toward AI-Assisted UAV for human detection in search and rescue missions", in proc. of the IEEE International Conference on

Decision Aid Sciences and Applications (DASA), Bahrain, December, 2021. DOI: 10.1109/DASA53625.2021.9682412

[C28] **M. Rizk**, A. Baghdadi and J-Ph. Diguet, "Rapid design and verification experience using flexible cycle-accurate NoC simulator", in proc. of the IEEE International Multidisciplinary Conference on Engineering Technology (IMCET), Beirut, Lebanon December 2021. DOI: 10.1109/IMCET53404.2021.9665518

[C27] S. Hraybi and **M. Rizk**, "Examining YOLO for Real-Time Face-Mask Detection", in proc. of the IET Smart Cities Symposium (SCS), Bahrain, November, 2021. DOI: 10.1049/icp.2022.0402

[C26] M. Badran, S. Kareh and **M. Rizk**, "PharmServ: A digital platform to facilitate searching for pharmaceuticals in Lebanon", in proc. of the IET Smart Cities Symposium (SCS), Bahrain, November, 2021. DOI: 10.1049/icp.2022.0432

[C25] H. Younes, A. Ibrahim, **M. Rizk** and M. Valle, "Hybrid Fixed-point/Binary Convolutional Neural Network Accelerator for Real-Time Tactile Processing," in proc. of the IEEE International Conference on Electronics, Circuits and Systems (ICECS), Dubai, UAE November, 2021. DOI: 10.1109/ICECS53924.2021.9665586

[C24] H. Younes, A. Ibrahim, **M. Rizk** and M. Valle, "A Mixed-Precision Binary Neural Network Architecture for Touch Modality Classification," in proc. of the International Conference on Synthesis, Modeling, Analysis and Simulation Methods, and Applications to Circuit Design (SMACD) and International Conference on PhD Research in Microelectronics and Electronics (PRIME), online conference, July, 2021.

[C23] H. Younes, A. Ibrahim, **M. Rizk** and M. Valle, "Efficient FPGA Implementation of Approximate Singular Value Decomposition based on Shallow Neural Networks," in proc. of the IEEE International Conference on Artificial Intelligence Circuits and Systems (AICAS), Washington DC, DC, USA, June, 2021. DOI: 10.1109/AICAS51828.2021.9458453

[C22] **M. Rizk**, H. Hmaydan and M. Hajj, "Proposition of Low-Cost Wireless Sensor Network for Real-Time Monitoring and Early Wildfire Detection in Lebanon's Forests" in proc. of the IEEE International Conference on Innovation and Intelligence for Informatics, Computing and Technologies (3ICT), Bahrain, December, 2020. DOI: 10.1109/3ICT51146.2020.9311994

[C21] K. Haj-Ali, **M. Rizk**, A. Baghdadi, J-Ph. Diguet, and J. Joumaa, "Memristor Overwrite Logic (MOL) for Energy-Efficient In-Memory DNN", in proc. of the IEEE International Symposium on Circuits and Systems (ISCAS), Seville, Spain, October, 2020. (DOI: 10.1109/ISCAS45731.2020.9180549)

[C20] **M. Rizk**, "Autonomous Thermometer System to Assist COVID-19 Pandemic Monitoring", in proc. of the IET Smart Cities Symposium (SCS), Bahrain, September, 2020. DOI: 10.1049/icp.2021.0944

[C19] **M. Rizk** and J. Charara, "Real-time SLAM based on Image Stitching for UAV Autonomous Navigation in GNSS-denied Regions," in proc. of the IEEE International Conference on Artificial Intelligence Circuits and Systems (AICAS), Genova, Italy, March 2020. DOI: 10.1109/AICAS48895.2020.9073793

[C18] H. Younes, A. Ibrahim, **M. Rizk** and M. Valle, "Algorithmic Level Approximate Computing for Machine Learning Classifiers," in proc. of the IEEE International Conference on Electronics Circuits and Systems (ICECS), Genova, Italy, November 2019. DOI: 10.1109/ICECS46596.2019.8964974

[C17] **M. Rizk**, A. Mroue, K. Haj-Ali and J. Charara, "Development and Implementation of an Embedded Low-Cost GNSS Receiver Emulator for UAV Testing," in proc. of the IEEE International Conference on Electronics Circuits and Systems (ICECS), Genova, Italy, November 2019. DOI: 10.1109/ICECS46596.2019.8964928

[C16] K. Haj-Ali, **M. Rizk**, A. Baghdadi, J-Ph. Diguet and J. Joumaa, "Crossbar Memory Architecture Performing Memristor Overwrite Logic", in proc. of the IEEE International Conference on Electronics Circuits and Systems (ICECS), Genova, Italy, November 2019. DOI: 10.1109/ICECS46596.2019.8964910

[C15] K. Haj-Ali, **M. Rizk**, A. Baghdadi, J-Ph. Diguet and J. Joumaa, "MRL Crossbar-Based Full Adder Design", International Conference on Electronics Circuits and Systems (ICECS), Genova, Italy, November 2019. DOI: 10.1109/ICECS46596.2019.8964702

[C14] H. Younes, A. Ibrahim, **M. Rizk** and M. Valle, "Data Oriented Approximate K-Nearest Neighbor Classifier for Touch Modality Recognition," in proc. of the IEEE International Conference on PhD Research in Microelectronics and Electronics (Prime), Lausanne, Switzerland, July 2019. DOI: 10.1109/PRIME.2019.8787753

[C13] **M. Rizk**, T. Naser Al-Deen, H. Diab, A. M. Ahmad and Z. Bazzal, "Proposition of Online UAV-Based Pollutant Detector and Tracker for Narrow-Basin Rivers a case study on Litani River," in proc. of the IEEE International Conference on Computer and Applications (ICCA), Beirut, Lebanon, July 2018. DOI: 10.1109/COMAPP.2018.8460374

[C12] **M. Rizk**, H. Semaan, H. El-Cheikh-Ibrahim, Ziad Noun and Z. Bazzal, "Online Detection and Location of Soft Faults in Wired Power Networks," in proc. of the IEEE International Conference on Computer and Applications (ICCA), Beirut, Lebanon, July 2018. DOI: 10.1109/COMAPP.2018.8460256

[C11] M. Sabbah, **M. Rizk**, A. Al-Ghouwayel, S. Omar and Z. El-Bazzal "Enhanced Throughput and Energy-Efficient MRAM-based NB-LDPC Decoder" in proc. of the IEEE International Conference on Computer and Applications (ICCA), Beirut, Lebanon, July 2018. DOI: 10.1109/COMAPP.2018.8460254

[C10] **M. Rizk**, A. Baghdadi, M. Jézéquel and A. Al-Ghouwayel, "NISC Design Experience of Flexible Architectures for Digital Communication Applications" in proc. of the IEEE International Conference on Computer and Applications (ICCA), Beirut, Lebanon, July 2018. DOI: 10.1109/COMAPP.2018.8460355

[C9] H. Younes, A. M. Ahmad, Z. Noun, **M. Rizk** and Z. El-Bazzal, "Gender and Identity Recognition using LabVIEW" in proc. of the IEEE International Conference on Computer and Applications (ICCA), Beirut, Lebanon, July 2018. DOI: 10.1109/COMAPP.2018.8460405

[C8] M. Sabbah, A. Al-Ghouwayel, **M. Rizk** and Z. El-Bazzal "M-RAM Based Memorization System for a NB-LDPC Decoder" in proc. of the IEEE International Conference on Microelectronics (ICM), Beirut, Lebanon, December 2017. DOI: 10.1109/ICM.2017.8268886

[C7] K. Haj-Ali, **M. Rizk**, A. Baghdadi, J-Ph. Diguet and J. Joumaa, "Towards Memristor-Based Reconfigurable FFT Architecture" in proc. Of the IEEE International Conference on Microelectronics (ICM), Beirut, Lebanon, December 2017. DOI: 10.1109/ICM.2017.8268885

[C6] **M. Rizk**, J-Ph. Diguet, N. Onizawa, M. J. Sepulveda, Y. Akgul, V. Gripon, A. Baghdadi and T. Hanyu, "NoC-MRAM Architecture for Memory-Based Computing: Database-Search Case Study" in proc. of the IEEE International New Circuits and Systems Conference (NEWCAS), France, Strasbourg, June 2017. DOI: 10.1109/NEWCAS.2017.8010167

[C5] K. Martin, **M. Rizk**, J-Ph. Diguet and M.J. Sepúlveda, "Notifying Memories: A Case-Study on Data-Flow Applications with NoC Interfaces Implementation" in proc. of the ACM /IEEE International Design Automation Conference (DAC), USA, Austin, TX, June 2016. DOI: 10.1145/2897937.2898051

[C4] **M. Rizk**, A. Baghdadi, M. Jézéquel, Y. Mohanna and Y. Atat, "Design and Prototyping Flow of NISC-Based Flexible MIMO Turbo-Equalizer" in proc. of the IEEE International Symposium on Rapid System Prototyping (RSP), New Delhi, India, October 2014. DOI: 10.1109/RSP.2014.6966687

[C3] **M. Rizk**, A. Baghdadi, M. Jézéquel, Y. Mohanna and Y. Atat, "Quantization and Fixed-Point Arithmetic for MIMO MMSE-IC Linear Turbo-Equalization" in proc. of the IEEE International Conference on Microelectronics (ICM), Beirut, Lebanon, December 2013. DOI: 10.1109/ICM.2013.6735008

[C2] **M. Rizk**, A. Baghdadi, M. Jézéquel, Y. Mohanna and Y. Atat, "Flexible and Efficient Architecture Design for MIMO MMSE-IC Linear Turbo-Equalization", in proc. of the IEEE International Conference on Communications and Information Technology (ICCIT), Beirut, Lebanon, June 2013. DOI: 10.1109/ICCITechnology.2013.6579576

[C1] **M. Rizk**, A. Baghdadi, M. Jézéquel, Y. Mohanna and Y. Atat, "Statically-Scheduled Application-Specific Processor Design: A Case-Study on MMSE MIMO Equalization", in proc. of the IEEE Design, Automation and Test in Europe (DATE), Grenoble, France, March 2013. DOI: 10.7873/DATE.2013.146

A.4 Published national conference papers

[NC5] **M. Rizk**, A. Baghdadi, and J-Ph. Diguet, "Real-Time Human Detection in Marine Environment Using Deep Learning on Edge Devices", in proc. of the GDR SoC2: Groupe de recherche System on Chip – Systèmes embarqués et Objets Connectés, Colloque National, Strasbourg, France, June 2022.

[NC4] **M. Rizk**, A. Baghdadi, M. Jézéquel, Y. Mohanna and Y. Atat, "MIMO MMSE Turbo-Equalizer Compatible with LTE-Advanced", in proc. of the 22th LAAS International Science Conference (LAAS16), Beirut, Lebanon, April 2016.

[NC3] **M. Rizk**, A. Baghdadi, M. Jézéquel, Y. Mohanna and Y. Atat, "Implementation of NISC-Based Flexible Architecture for MIMO MMSE-IC Turbo-Equalization", in proc. of the Journées Nationales du Réseau Doctoral en Micro-nanoélectronique (JNRDM), Lille, France, May 2014.

[NC2] **M. Rizk**, A. Baghdadi, M. Jézéquel, Y. Mohanna and Y. Atat, "Application-Specific Processor Design: A Case-Study on MMSE MIMO Equalization", in proc. of the GDR SOC SiP: Groupe de recherche System on Chip - System in Package, Colloque National, Lyon, France, June 2013.

[NC1] **M. Rizk**, A. Baghdadi, M. Jézéquel, Y. Mohanna and Y. Atat, "Designing a NISC-Based Flexible Architecture for MIMO MMSE-IC Turbo-Equalization", in proc. of the Journées Nationales du Réseau Doctoral en Micro-nanoélectronique (JNRDM), Grenoble, France, June 2013.

Appendix B

Supervision Activities

Table. B.1, Table. B.2 and Table. B.3 show the Master theses I have supervised classified according to the university and major. Table. B.4 shows the senior projects that I have supervised classified according to the university and major.

Table B.1 — Supervised Master theses at Lab STICC and IMT Atlantique

Academic Year	**Thesis Title**	
2014-2015	Design of smart memories on multiprocessor architecture for data-flow applications	SEH
2020-2021	Optimizing embedded systems for multi-view AI-based object detection	SEH
2020-2021	Smart decision-making approaches for multi-view AI-based object detection on edge embedded devices	SEH
2020-2021	Top-view real-time embedded AI detector and tracker to assist multi-view marine object surveillance	SEH

Table B.2 — Supervised Master theses at Lebanese University

Academic Year	Thesis Title	Major
2021-2022	Acoustic Classification Using Machine Learning on Embedded Edge Devices to Assist Real-Time Drone Detection	STIP
2018-2019	Evaluation of the conformity of the maintenance department at the company Flashmed to the quality control technical and management requirements	MPIT
2019-2020	Surgical tool detection in cataract surgery	MPLI
2019-2020	Examining image processing techniques for living human detection and localization for search and rescue missions	MPLI
2020-2021	Crowd detection in closed areas to assure physical distancing during COVID-19 pandemic	MPLI
2020-2021	Remote real-time monitoring of the environment of pharmaceuticals and food	FEME
2017-2018	Hadat faculty of sciences buildings electrical auditing and power economy	FEME
2016-2017	Hadat desalination using reverse osmosis	FEME

Table B.3 — Supervised Master theses at Lebanese International University

Academic Year	Thesis Title	Major
2020-2021	Computer vision and artificial intelligence to detect commercial drone in no-fly zones	MCCE
2020-2021	Crowd detection in closed areas to assure physical distancing during COVID-19 Pandemic	MCCE
2020-2021	Securing the MavLink protocol for unmanned aerial systems	MCCE
2019-2020	Design and implementation of a smart and flexible wireless sensor network for early detection of environmental disasters	MCCE
2019-2020	Path guidance robot for blind person assistance	MCCE
2018-2019	Building a MongoDB-based data analysis system for industrial applications	MCCE
2018-2019	Change detection using drone images and GIS	MCCE
2017-2018	Exploring image processing and localization techniques for UAV-based pollutant detector and tracker for narrow-basin rivers	MCCE
2016-2017	Detecting and locating soft faults in wiring networks using time reversal techniques	MCCE
2016-2017	Design and implementation of automatic electrical load manager for smart facilities	MCCE
2016-2017	Study and design of a MRAM-based memorization system for a NB-LDPC decoder	MCCE
2016-2017	Detecting and locating soft faults in wiring networks using time reversal techniques	MEEE
2016-2017	Design and implementation of automatic electrical load manager for smart facilities	MEEE

Table B.4 — Supervised senior projects

Inistitute	Academic Year	Thesis Title	Major
IMT-Atlantique	2021-2022	Smart-cameras network for multi-view AI-based object detection	SEH
LIU	2020-2021	Municipality (Baladiyati) digital platform	CENG
LIU	2020-2021	Barber shop application	CENG
LIU	2020-2021	Online shop application	CENG
LIU	2020-2021	Lebanese COVID-19 tracker application	CENG
LIU	2020-2021	Pharmacy services application	CENG
LIU	2016-2017	Web-based water intake monitoring system for remote cardiac patients	CENG
LIU	2016-2017	Web-based rental management system	CENG
LIU	2016-2017	Smart irrigation system for playgrounds	CENG
LIU	2016-2017	Smart home: security and automation	CENG
LIU	2016-2017	Remote vehicle indicators monitoring	CENG
LIU	2016-2017	Home Automation System	LENG
BIU	2020-2021	Ambulance emergency response time scale-up	BEC
BIU	2020-2021	Building a machine learning tool	BEC

Appendix C

Contributions In Evaluation Committees

Table. C.1 shows the senior projects I have participated in their evaluation committees classified according to the university and major.

Table C.1 — Senior projects I have participated in their evaluation committees

Inistitute	Academic Year	Project Title	Major
LIU	2020-2021	Hospital management system	CENG
LIU	2020-2021	Travel agency system	CENG
LIU	2020-2021	Blood bank system	CENG
LIU	2020-2021	CV building application	CENG
LIU	2020-2021	Bank queuing system	CENG
LIU	2017-2018	A school-to-parent communication platform	CENG
LIU	2017-2018	A school bookstore website	CENG
LIU	2016-2017	Concealing an Image inside a Video File	CENG
LIU	2016-2017	Data Transmission of Blood Pressure	CENG
LIU	2016-2017	Wireless Baby Temperature Monitor	CENG

Table. C.2 shows the Master theses I have participated in their evaluation committees classified according to the university and major.

Table C.2 — Master theses I have participated in their evaluation committees

Inistitute	Academic Year	Thesis Title	Major
LIU	2020-2021	University system for advising (USA)	MCCE
LIU	2020-2021	Human-computer interface using eye gaze	MCCE
LIU	2020-2021	Deep learning with JavaScript	MCCE
LIU	2020-2021	Android malware detection using machine learning	MCCE
LIU	2019-2020	Optimal RSU placement for VANET	MCCE
LIU	2019-2020	Development of NB-LDPC codes based on different GF for iterative receivers	MCCE
LIU	2019-2020	Parallel implementation of feed-forward neural networks	MCCE
LIU	2017-2018	A Lebanon wide IoT network	MCCE
LIU	2017-2018	Augmented reality using Kinect	MCCE
UL	2018-2019	Facilities management and security: preparing for accreditation at DAR-AL-AMAL hospital	MPIT
UL	2018-2019	Microwave microfluidic sensor for detecting biological cells	MPIT
UL	2018-2019	Analysis of the effect of auditory and visual simulations on EEG signals using Wavelet Transform	MPLI
UL	2019-2020	Texture analysis of uterine fibroma on MRI	MPLI
UL	2019-2020	X-ray image segmentation and fracture detection using image processing techniques	MPLI
UL	2020-2021	Covid-19 lung infected image analysis and interpretation	MPLI
UL	2020-2021	Image Analysis and Interpretation of Knee Osteoarthritis	MPLI
UL	2020-2021	Heating, ventilation and air-conditioning systems in the context of COVID-19	FEME
UL	2019-2020	An optimum selection of secondary batteries for UPS systems: a comprehensive study	FEME
UL	2019-2020	Distance Patient Monitoring	FEME
UL	2018-2019	Pharmaline BMS project	FEME

Appendix D

Teaching Activities

Table. D.1, Table. D.2 and Table. D.3 list the taught courses classified according to the university, level, and major.

Table D.1 — Taught courses at American University of Culture and Education, Lebanon

Academic Years	Title	Level	Major	Length
2011-2012	Microprocessors and Interfaces	BS-L3	CS	$40h$
2011-2012	Introduction to Computers	BS-L1	CS	$40h$
2011-2012	Calculus I	BS-L1	CS	$40h$

Table D.2 — Taught courses at Lebanese University

Academic Years	Title	Level	Major	Length
2011-2013	Signals and Image Processors	MS-M2	STIP	$24h$
2017-2018	Contract Administration	MS-M2	FEME	$24h$
2018-2021	Project Management	MS-M2	FEME	$24h$
2019-2021	Informatics	BS-L1	SS	$50h$

Table D.3 — Taught courses at Lebanese International University

Academic Years	Title	Level	Major	Length
2015-2018	Electric Circuits I	BS-L1	LENG	$45h$
2017-2018	Fundamentals of Digital Logic Design	BS-L2	CENG	$45h$
2017-2020	Advanced Digital Logic Design	BS-L2	CENG	$45h$
2015-2017	Fundamentals of Optoelectronics	BS-L3	LENG	$45h$
2015-2016	Electrical Systems Simulation	BS-L3	EENG	$45h$
2016-2017	Operating Systems	BS-L3	CENG	$45h$
2018-2021	Introduction to Database Systems	BS-L2	CENG	$45h$
2016-2017	Microprocessors and Microcontrollers	BS-L2	CENG	$45h$
2016-2017	Signals and Systems	BS-L2	CENG	$45h$
2016-2017	Senior Project - LENG	BS-L3	LENG	$45h$
2016-2021	Senior Project - CCE	BS-L3	CENG	$45h$
2018-2020	Communication Networks Lab	BS-L3	CENG	$45h$
2015-2016	Fiber optics	MS-M1	MCCE	$45h$
2016-2020	Digital Communication	MS-M1	MCCE	$45h$
2016-2019	Computer Architecture	MS-M1	MCCE	$45h$
2019-2021	Multimedia Networks	MS-M2	MCCE	$45h$
2020-2021	Image Processing and Computer Vision	MS-M2	MCCE	$45h$
2017-2020	Introduction to Mechatronics	MS-M1	MLENG	$45h$
2017-2021	Embedded Systems	MS-M1	MCCE	$45h$
2017-2018	Linux Lab	BS-L3	CENG	$23h$
2016-2018	Software Applications and Design Lab	BS-L2	CENG	$23h$
2017-2018	Embedded Systems Lab	MS-M1	MCCE	$23h$
2016-2019	Computer Architecture Lab	MS-M1	MCCE	$23h$

Part II

Overview of Past Research Activities

Introduction to My Research Activities

My previous research activities since the beginning of my PhD thesis have addressed four topics:

1. **Flexible yet efficient architectures for applications in the digital communication domain**:
   This topic was the subject of my PhD thesis. However, I continued working on this topic afterward. It spans the refinement of algorithms to meet with hardware constraints and exploring new design methodologies to achieve flexiblity, efficiency and designing productivity. In particular, the No-Instruction-Set-Computer design methodology has been explored to design hardware architectures of the base-band modules.

2. **Efficient algorithms and architectures for dataflow applications**:
   In this topic, new algorithms and architectures have been devised to enhance the performance of dataflow applications and to reduce the communication overheads and the power consumption. In particular, the notifying memory concept has been introduced to reduce the number of useless attempts, which processors perform to read from and/or write into memories. Also, a novel approach has been introduced to remap the actors (tasks) during run-time on NoC-based heterogeneous MPSoCs in order to fit with the dynamic behavior of dataflow applications.

3. **Efficient and flexible design paradigms based on emergent memristive devices**:
   An important part of my research activities targets exploiting emergent memristive devices to achieve flexible high performance architectures. The novel NoC-Memory based computing concept has been realized and validated by making use of emergent non-volatile magnetoresistive random-access memory with power-gating capabilities targeting database search application implemented with neuromorphic architecture based on Sparse-Neural-Network. In addition, the use of memristors to design architectures which combine flexibility and efficiency, has been explored and introduced through the proposal of original architectures that break the limits of the existing ones. The exploration and study have been conducted in three main levels: (1) interconnect level, (2) processing level and (3) memory level.

4. **Efficient implementations of machine learning algorithms**:
   The main objective of this topic is to design efficient implementations of machine learning algorithms with as much reduced hardware area and energy consumption. The application targets tactile data processing. This topic addresses basically designing specific hardware accelerators and exploiting approximate computing techniques to accelerate

demanding portions of machine learning algorithms and to adequately reduce the algorithms computational complexity respectively.

In the following, for each topic, the context of the conducted work is given. After introducing the topic, motivations are provided along with the background showing the related performed work and what is missing from the state-of-the-art. Then, the list of contributions and the performed technical work is provided. Most important results are shown next followed by the most impacting publication on the subject.

Chapter 4

Flexible and Efficient Architectures for Applications in Digital Communication

4.1 Preface

After finishing my PhD, I had the chance to personally work on the refinement of the work done during my PhD and to valorize the obtained results in well-recognized scientific journals and international and national conferences. This chapter demonstrates briefly the conducted work during my PhD and presents the publications that have been published since the end on my PhD.

4.2 Introduction

Applications in the field of wireless digital communications are becoming increasingly complex and diverse. Recent emergent wireless communication standards support various modes and configurations related to channel coding type, modulation type, and antenna dimension for multiple-input multiple-output (MIMO) transmission techniques. On the other hand, iterative concept is also utilized at the receiver side to alleviate the destructive effects of the channel [3], but data the cost of additional computational complexity.

In digital communication application domain, flexibility is an emerging design feature that permits the hardware architectures to cope with the various requirements of multiple communication standards for different system configurations. Circuits and systems adopted in this application domain must not only consider performance and implementation constraints, but also the requirement of flexibility. Because of the rapid evolution of related standards, modern wireless digital communication systems are highly concerned about the flexibility feature. This flexibility requirement is foreseen to become even much higher with the multiple communication scenarios and modes [4].

Nowadays, highly flexible architectures are supposed to be exploited in multiple wireless communication standards (WiFi, WiMax, LTE, LTE-A, and DVB). The implementations of these architectures should function properly within different system configurations and

should not be restricted to certain types of modulation, mapping styles, and/or antenna dimensions. Application-specific processor design approaches allow to tune the performance-flexibility tradeoff as required by the target application. The concept has emerged several years ago. Note that the flexibility degree can be fully tuned by the designer. The concept has no limitation in this regard. In this context, flexible application-specific hardware architectures implementing the functionalities of digital baseband components of the receiver are under design scope. Application-specific processors constitute a key trend in implementing definite blocks of wireless system since they provide a good solution in designing flexible architectures that can fulfill nowadays requirements in terms of low error-rate performance and high throughput, and satisfy the tight constraints on implementation area and power consumption.

The combination of flexibility and the ever increasing performance requirements demands design approach that provides better ways of controlling and managing the hardware. In general, available design approaches that adopt dynamic scheduling of instructions add an overhead due to the instruction decoding. To minimize this overhead, several approaches have been introduced, which opt static scheduling. In this context, No-Instruction-Set-Computer (NISC) concept has been introduced to design application-specific processors without an instruction set. NISC concept proposes that there is no need to first design and then use an instruction set when the hardware is programmed by its designers rather than its users. NISC designing approach offers a good compromise between flexibility, productivity, and quality for the design of a digital system.

In this research work, NISC approach is explored through the design of flexible and efficient architectures dedicated for digital communication applications which fulfill the requirements imposed by multiple emergent communication standards.

4.3 Context and Motivations

4.3.1 Design Approaches Overview

The baseband modules could be implemented using dedicated or programmable hardware implementations, depending on the target performance requirements. The dedicated hardware implementations adopting low-level design approaches at Register Transfer-Level (RTL) ensure best exploitation of the allocated resources to execute the desired functions, however the main issue of such approach concerns designer productivity and the typical low degree of flexibility. On the other hand, programmable hardware, i.e. instruction-set based processors, are highly flexible through software programmability, yet present several limitations in terms of achievable performance and energy/area efficiency. High-Level Synthesis (HLS) tries to improve the productivity by converting directly high-level language, C for example, description into RTL. When using HLS, the designer cannot manage precisely the correlation between the modifications applied to the application and the obtained implementation performance/cost metrics such as area, power, clock frequency, routable layout, etc. To improve the quality, the designer can depend only on guess-try-check strategy. The quality of the result is noticeably low compared to manual RTL, particularly for complex applications. Furthermore, in HLS the area and delay of the design increase rapidly as the size of

the software application description increases. Moreover, the design generated by HLS is not reprogrammable contradicting the ever increasing requirement of flexibility in emerging applications. Figure 4.1 illustrates the traditional available design approaches.

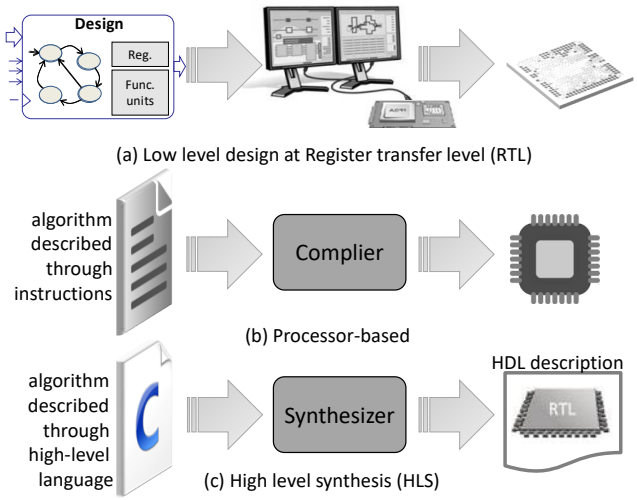


Figure 4.1 — Traditional available design approaches

The emerging flexibility need in designing application-specific processors dedicated for modules of digital receiver sets up a new design metric added to the requirements of efficiency and productivity. Application-Specific Instruction-set Processor (ASIP) design concept has been introduced as an increasingly popular option for implementing flexible architectures. ASIP concept offers a tradeoff in terms of designer productivity and the final quality of the hardware implementation. In ASIP designs, the instruction set is designed to fit in with a specific application. This specification of ASIP designs provides a compromise between the flexibility of a general-purpose processor and the performance of application-specific integrated circuit (ASIC) [5]. In ASIP, the functionality and datapath structure can be adjusted to fit in with specific application through using custom instructions. At run-time, custom instructions are decoded and executed by the dedicated hardware. Figure 4.2 illustrates the ASIP design approach. The productivity of the designer is reduced since the task of

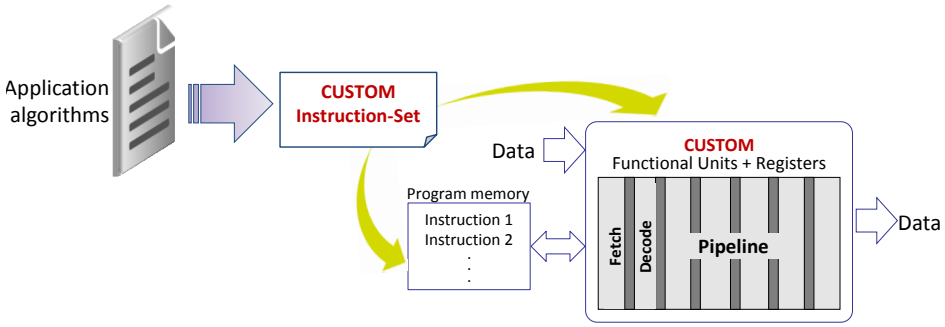


Figure 4.2 — ASIP design approach

describing the controller and instruction decoder is tedious and very time-consuming. Additionally, all available ASIP design approaches demand the design of instruction decoder. In

cases where the hardware is dedicated for a specific application, designing and implementing the instruction decoder create an overhead in terms of complexity, power consumption, and implementation area.

NISC design approach has been introduced as a solution to fill the gap between designer productivity and design metrics, which are two important factors for emerging design methodologies. The approach simplifies ASIP design approach by eliminating the tedious task of defining and describing custom instruction set at the designing phase and the complex task of decoding instructions at run-time. The removing of the instruction set enhances the productivity of the designer and reduces the time-to-market. In contrast to ASIP, NISC approach reduces the complexity of the architecture since the hardware resources are allocated to fit in with the exact needs of the required application. The hardware is simplified due to omitting of instruction decoder. A simple controller is added instead of fetching and decoding stages. The program memory containing instructions is replaced by control memory that contains the control words that must be applied to the datapath components at run-time. Figure 4.3 illustrates the transition from ASIP design approach to NISC design approach.

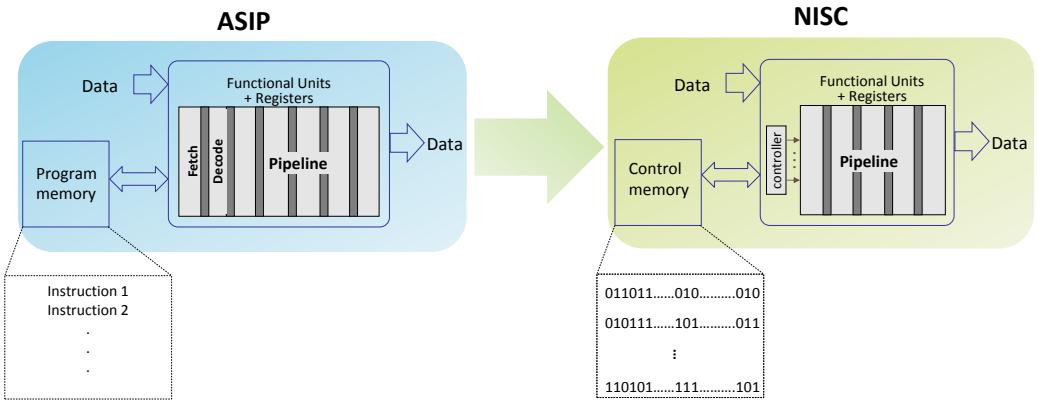


Figure 4.3 — Transition from ASIP to NISC design approach

In NISC design approach, the fundamental functionalities of the controller in a processor such as fetching and decoding the instruction, analyzing the dependency, and scheduling the instructions are performed statically by the compiler. The compiler is neither restricted by a specific hardware resources due to the size of implementation area nor limited by timing constraints due to the desired execution performance. At compilation time the compiler generates the set of control words (*CWs*), which should be delivered to the datapath at run-time, and stores it in the control memory. At run-time, the controller simply loads *CWs* at every clock cycle and derives them to the control ports of the components constituting the datapath. In other words, the decoding and scheduling of instructions are done at compiler's execution time. Hence, their overhead does not impact the application execution time. Figure 4.4 demonstrates the stages of NISC designing approach.

4.3.2 Overview on NISC design approach

NISC is a custom-hardware design technology which has been developed initially at the Center for Embedded Computing Systems (CECS), UC Irvine [6]. NISC architectures do

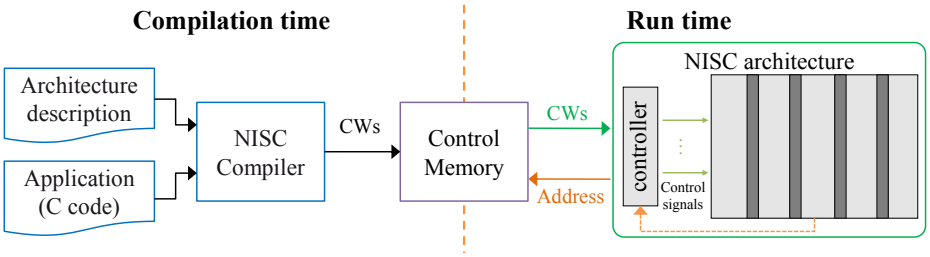


Figure 4.4 — NISC designing approach stages

not include an instruction set abstraction. The designer selects a custom datapath for the desired processor. A compiler first captures the code of the application, which is developed using a high level language. Then, at each clock cycle, it generates the proper control signals for the specified datapath.

Figure 4.5 shows the general overview of a processor that can be described as a controller and a datapath. In each clock cycle, the controller transfers the control bits to the datapath in order to perform a specific task and gets a feedback from the datapath in order to determine the state of the system in the next clock cycle. Every processor has a controller that transfers the sequence of control signals in memory (address space dimension) to a sequence of control words for each clock cycle (time dimension). Each instruction implicitly determines the behavior of all datapath components at specified execution cycle. NISC architectures follow the same architectural template as general processors except that instruction decoder is removed and it is shifted from hardware at run-time to the compiler at compilation time.

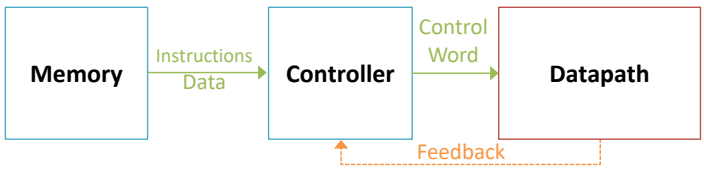


Figure 4.5 — General processor block diagram

Recently, the compiler has been given more control over the processor. This design concept has constituted a key trend in implementing hardware designs. The idea of expanding the functionality of the compiler and consequently increasing its control over the processor refers to many facts. The compiler has a bigger observation window than the hardware decoder because of its ability to parse the entire application program. In addition, the compiler can implement easier complex algorithms since it is not restricted by limited hardware resources. Moreover, transferring tasks to be performed in software rather than hardware results in simplifying the latter complexity and reduces the run-time overhead.

In NISC technology, the compiler schedules operations and decodes them into control words that control the hardware resources at run-time at each clock cycle. NISC designs have no instructions; hence no decoding stage is required. Furthermore, since all operations are scheduled statically, NISC designs do not include run-time hardware scheduler. In other words, the roles of the decoder and scheduler are transferred from hardware to software compiler. The control words (also called nanocodes), which are stored in program memory, are directly loaded to control ports of datapath components. Architectures adopting NISC

concept face the risk of having extra memory needs to store the control words. This issue is called *code bloating* and is considered as a common problem in statically scheduled Horizontal Microcoded Architectures (HMA) [7]. Thus, the size of the control memory creates a new design challenge for the designers. In fact, the size of the control memory depends on the target application and the opted architecture choices. However, several techniques have been proposed and applied to reduce the memory size such as code packing, dictionary based and arithmetic-based compression [8].

4.3.3 NISC Dynamic flexibility

In NISC-based architectures, dynamic flexibility can be attained by storing CWs related to different configurations in the control memory where each configuration has a different starting offset address. Hence in this case, offset address information is sufficient to change the configuration. Figure 4.6 illustrates how to attain dynamic flexibility in NISC.

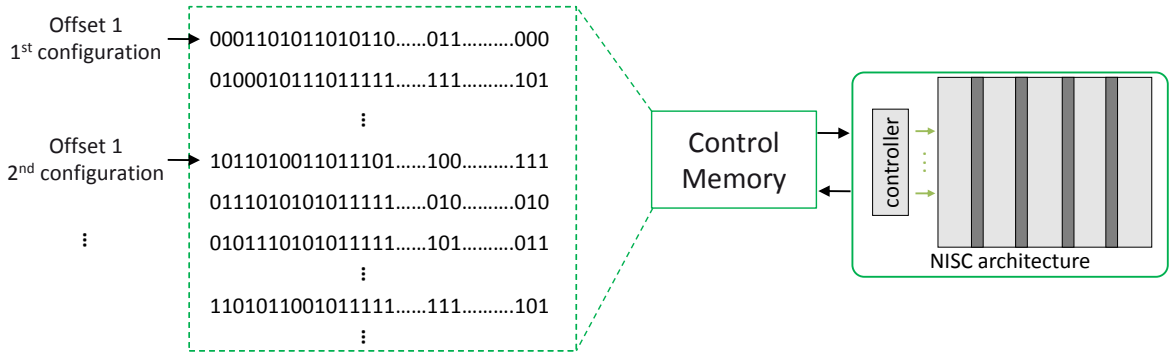


Figure 4.6 — Dynamic flexibility in NISC

On the other hand, the large size of *CWs* constitutes typically an issue for statically scheduled architectures. However, this issue depends on the considered application and target flexibility requirement, besides the devised architectural choices for the datapath components. Furthermore, several optimization techniques such as code packing, dictionary-based or arithmetic-based compression techniques can be applied to overcome this issue [8].

4.4 Contributions and Performed Work

In this work, NISC concept is adopted to design flexible architectures dedicated for turbo equalization [9] and turbo demapping [10] functions that can be utilized in digital communication systems which fulfill the requirements imposed by multiple communication standards. The objective of this research work is to apply jointly flexibility and optimization on algorithms and hardware, with the aim of finding the best compromise between performance of the algorithms and the cost of their flexible implementation. The originality and novelty of the work mostly lie in designing simplified versions of the equalization and demapping functions of single-input-single-output (SISO) wireless receivers, with a manageable degradation of detection performance, then applying the designs on a NISC architecture, and then

implementing the architecture on an FPGA. It has been shown that the devised approach is different from the ones described in the literature, and illustrated how the produced performance results have consistently superior to those of the other approaches.
The contributions of this research work is summarized in the following subsections:

4.4.1 Study the requirements and features of modern communication systems

The objective of this work is to explore a new design approach for flexible architectures dedicated for wireless digital communication applications. Each wireless communication standard specifies a set of parameters concerning different components of the wireless communication system. So, the first step is studying the requirements and features of modern communication systems, including WiFi [11], WiMax [12], LTE [13], LTE-Advanced [14] and DVB (RCS [15, 16],T2 [17, 18] and S2 [19]). The specifications supported by these different wireless communication standards are investigated thoroughly.

Accordingly, the common building blocks of the transmission chain are analyzed, especially those that are computationally intensive and could represent bottlenecks when processing information symbols, both on the transmitter and receiver sides. In this work, the two blocks that were identified are the receiver's equalizer and the demapper which complexities have increased considerably, especially with the advent of Multiple Input Multiple Output (MIMO) transceiver designs.

4.4.2 Exploring NISC design methodology and toolset

NISC design approach is investigated to find its relevance in designing flexible yet efficient architectures in counterpart other available design approaches. The advantages of NISC architecture are analyzed to determine its potentials in speeding up computations, and hence making the case for its suitability and ability for accelerating the receiver's functions, for reducing power consumption, and for reducing implementation complexity.

On the other hand, NISC design methodology is explored in deep along with the available toolset. In fact, based on this exploration, an appropriate design and prototyping flow, which combines the conventional NISC toolset flow and direct controlling of hardware resources to ensure both productivity and implementation efficiency is devised.

4.4.3 Contributions in algorithmic domain

In order to design efficient low complexity architectures, the algorithms of soft-in soft-out turbo equalization and turbo demapping are refined toward hardware implementation:

- Both targeted algorithms are thoroughly investigated. Algorithmic computations are surveyed to extract the characteristics of all involved parameters. Sufficient mathematical analysis is devised to simplify the computational overhead of operations in different algorithmic steps.

- For the equalization algorithm, several novel decompositions is applied to complex number operations and complex matrix operations leading to enhanced performance and to a significant reduction of utilized computations. This work has been published in the *Journal of Circuits, Systems and Computers (JCSC)*, 2019 [20].

- For the demapping algorithm, the developed simplifications have been analyzed to achieve best tradeoff in terms of performance and complexity for different system configurations. In addition, reference software model is developed to validate all refinement techniques performed on these algorithms. Note that a detailed survey about the simplifications on the demmaping algorithm and the corresponding hardware implementations has been published in 2022 in the *Journal of Circuits, Systems and Computer s(JCSC)* [21].

- For both algorithms, the complexity of the adopted refined forms is evaluated. The required numbers and types of real operations are determined. Studying and analyzing in details the functions of the receiver's equalization and demapping blocks, and proposing simplified alternatives to their typical implementation architectures, while at the same time considering the resulting degradation in symbol error rate performance. Not only these simplifications make the algorithms more suitable for implementation on NISC architectures, but they can also improve performance in terms of speed and power consumption, with anticipated tolerable degradation in detecting symbols correctly.

4.4.4 Contributions on Hardware implementations

In the following, the main contributions to realize the hardware designs of two NISC processors for turbo equalization and turbo demapping functions:

- The dependency among algorithm steps is studied to provide the scheduling of computational operations achieving high parallelism degree.

- The requirement of flexibility in both designs is analyzed based on the specifications of emergent wireless communication standards. For each architecture design, several flexibility parameters is defined to cope with multiple standard requirements for different system configurations.

- A careful numerical study is conducted to determine the accurate quantization and fixed-point representation of all parameters and computational values involved in the algorithms at different algorithmic steps to ensure numerical stability of implemented algorithm. A reference software model is developed to evaluate the impact of devised quantization and fixed-point arithmetic on the error-rate performance. The detailed numerical study and its impact on the error rate performance have been published in *EURASIP Journal on Embedded Systems*, 2017 [22].

- A hybrid design flow is proposed as a combination between available NISC toolset flow and direct controlling of hardware resources. The adopted design flow to realize the hardware implementations has been described in *Electronics*, 2016 [1]. Figure 4.7 shows an overall presentation of the adopted flow.

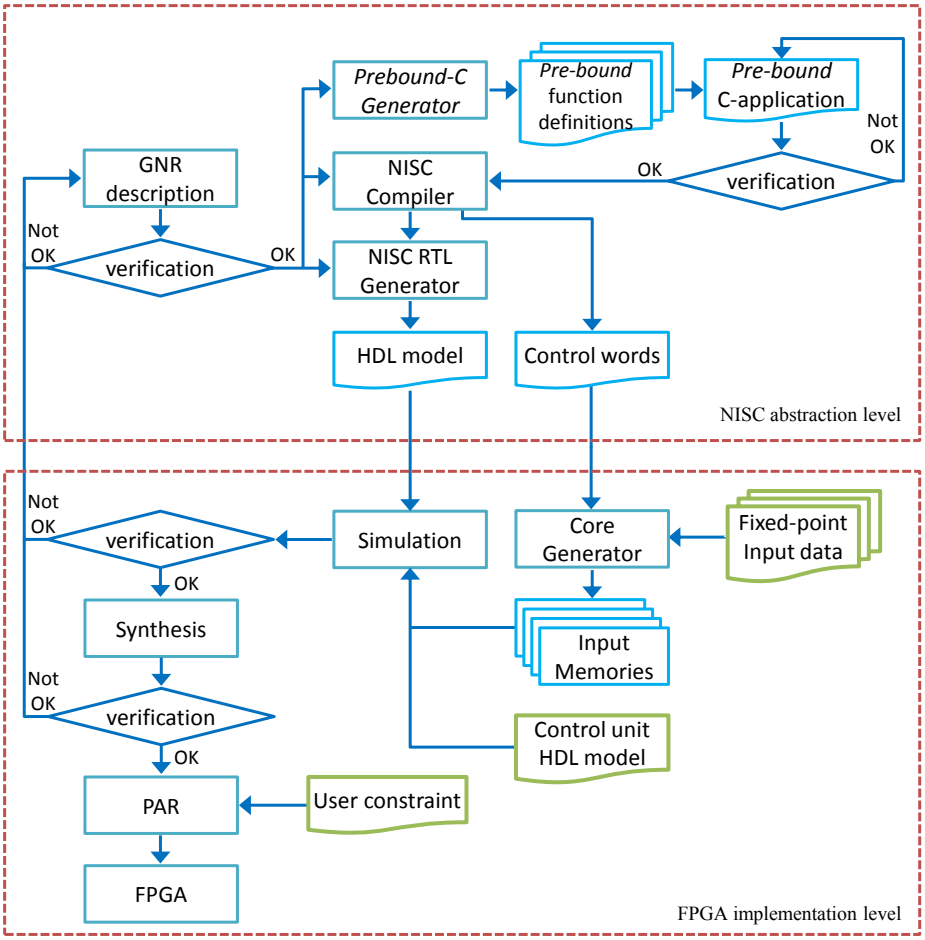


Figure 4.7 — Adopted design flow [1].

- Taking into account the refined algorithm with reduced complexity and based on the standards requirements, flexibility parameters are specified. All computational operations are scheduled achieving maximum parallelism. Depending on required flexibility parameters and operation scheduling, the architecture choices are selected to achieve an efficient architecture for different system configurations. The hardware design of flexible NISC-based architectures dedicated for turbo equalization and turbo demapping functions is tailored such that the hardware resources are allocated according to selected architecture choices.

- The proposed NISC architectures are implemented on top of FPGA fabric, specifically using the well known Xilinx Virtex-7 XC7VX485T FPGA. Also, the devised NISC-based architectures are synthesized towards ASIC target. The detailed architectures of the devised demapper and detector have been published respectively in *IEEE Transactions on Circuits and Systems II*, 2015 [23] and in the *Journal of Circuits, Systems and Computers (JCSC)*, 2021 [24].

- Using proven analysis and testing tools, the design and the implementation are verified and tested. On-chip validation of the two proposed NISC-based architectures is performed to verify the exact performance of the implemented architectures. Xilinx

ChipScope Pro Analyzer is used to record the output results of the design as illustrated in Figure 4.8.

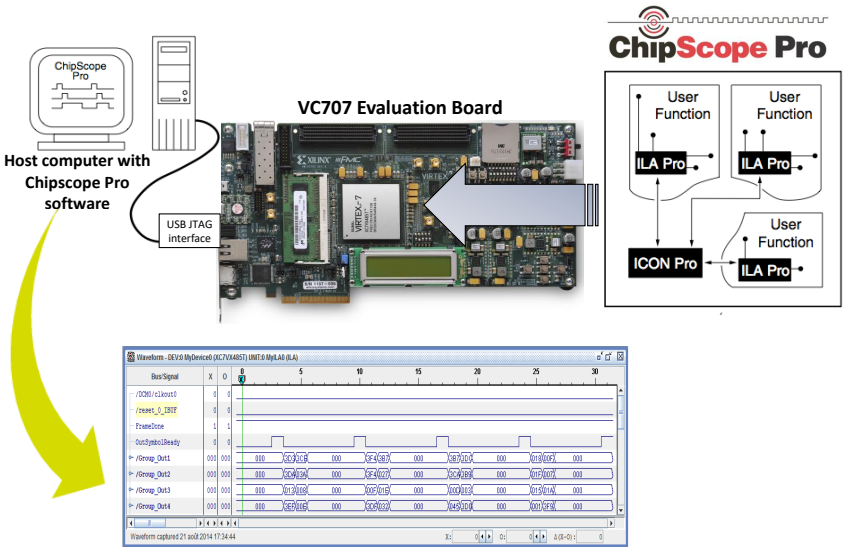


Figure 4.8 — On-board validation using Xilinx ChipScope Pro Analyzer

- Testing and validation allow for producing descriptive results for related performance measures and complexity costs of the implementations and for comparing the obtained results with those of other designs and implementations found in the literature [25, 26, 27, 28, 29, 30, 31, 32, 33]. The adopted approach, results and comparisons have been published in 2018 in *IEEE International Conference on Computer and Applications (ICCA)*, 2018 [34] and then detailed in *Design Automation for Embedded Systems (DAEM)*,2021 [35].

4.5 Findings

In both designed NISC-based architectures, the obtained results in terms of performance and implementation area confirm the feasibility of adopting NISC concept in designing flexible yet efficient application-specific processors in the domain of digital communications.

Both designed NISC-based architectures have been compared to state-of-the-art ASIP-based architectures using similar computational resources and supporting same flexibility parameters. The obtained results show that the proposed NISC-based architectures provide a significant improvement in execution performance while having reduced implementation costs.

Also, the obtained results show that the size of the required control memory depends on the considered algorithm and the devised architectural choices. In the detector module, the adopted re-usability of allocated resources imposes separate controlling of each component; hence, additional control signals are implied. Whereas for the demapper module, implemented hardware components are considered to perform specific operations and to deal with the same type of data; hence, the number of control signals can be reduced significantly.

4.6 Selected papers

1098 IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS—II: EXPRESS BRIEFS, VOL. 62, NO. 11, NOVEMBER 2015

NISC-Based Soft-Input–Soft-Output Demapper

Mostafa Rizk, *Student Member, IEEE*, Amer Baghdadi, *Member, IEEE*, Michel Jézéquel, *Senior Member, IEEE*, Yasser Mohanna, and Youssef Atat

***Abstract*—Applications in wireless digital communication field are becoming increasingly complex and diverse. Circuits and systems adopted in this application domain must not only consider performance and implementation constraints but also the requirement of flexibility. The combination of flexibility and the ever increasing performance requirements demands design approach that provides better ways of controlling and managing hardware resources. An application-specific instruction-set processor (ASIP) design approach is a key trend in designing flexible architectures. The ASIP concept implies dynamic scheduling of a set of instructions that generally leads to an overhead related to instruction decoding. The no-instruction-set-computer (NISC) concept has been introduced to reduce this overhead through the adoption of static scheduling. In this brief, the NISC approach is explored through a case-study design of universal demapper for multiple wireless standards. The proposed design has common main architectural choices as a state-of-the-art ASIP for comparison purpose. The obtained results illustrate a significant improvement in execution time and implementation area while using identical computational resources and supporting same flexibility parameters.**

***Index Terms*—Demapper, flexibility, iterative processing, multi-standard wireless system, no-instruction-set-computer (NISC).**

I. Introduction

CURRENTLY, flexibility is a major design requirement of embedded systems and circuits. Hardware architectures are supposed to accommodate multitude system configurations as well as their corresponding algorithmic variants. Because of the rapid evolution of related standards, modern wireless digital communication systems are highly concerned about the flexibility feature. However, the emergent flexibility need should not come at the cost of performance and implementation requirements. Application-specific processors are increasingly adopted to implement definite blocks of wireless system since they provide a good solution in designing flexible architectures that can fulfill nowadays requirements in terms of low error-rate performance and high throughput, and satisfy the tight constraints on implementation area and power consumption.

The application-specific instruction-set processor (ASIP) concept offers a tradeoff in terms of the flexibility of general-purpose processors and the efficiency of application-specific integrated circuit (ASIC) by customizing the functionality and the data path structure through a custom instruction set [1]. This tradeoff can be tuned in a language-based ASIP design approach, when the degree of flexibility is limited, to reach an implementation efficiency comparable to parameterized architectures using classical register-transfer level (RTL) design approach [2]. However, when hardware is dedicated for a specific application, processes of specifying and describing instructions at the designing phase and decoding them at runtime form an overhead in terms of productivity, execution performance, and implementation costs. The no-instruction-set-computer (NISC) concept [3] adopts static scheduling of operations instead of dynamic scheduling (i.e., decoding instructions at runtime to determine which operations to execute) to simplify the ASIP approach. Design productivity is increased by obviating the task of finding and designing a custom instruction set. The design quality is better ensured by reducing design complexity to match exactly the requirements of the desired application.

In a previous work, presented in [4], the NISC concept has been explored to realize flexible turbo equalizer. The comparison with a similar ASIP implementation, which uses identical computational resources and supports the same flexibility parameters, illustrates significant improvement in throughput with reduced implementation costs. However, additional memory locations are required to implement the control memory with respect to the ASIP program memory. In fact, this is directly related to the considered application and devised architecture choices. the NISC concept is evaluated in this brief through a different application design and architecture choices. This brief presents a novel NISC-based universal demapper. The proposed architecture is thoroughly described. In addition, the design is compared with a state-of-the-art ASIP-based equivalent implementation in terms of performance, throughput, and area of implementation. Recent emergent wireless communication standards, support various modes and configurations related to the characteristics of the target constellation such as modulation type and mapping style. Different order constellations have been employed starting from binary phase-shift keying (BPSK) up to 256-ary quadrature amplitude modulation (QAM).

Constellations with Gray mapping are adopted to achieve the lowest possible bit-error probability [5]. Furthermore, the independence between the in-phase (I) and the quadrature (Q) components of a symbol in Gray-mapped constellations can be exploited to reduce the computational complexity without suffering any performance loss. Accordingly, numerous simplifications have been proposed for specific constellations [6]. These simplifications can not be applied when incorporating signal space diversity (SSD) with rotated constellation introduced in DVB-T2 standard since the independence between I and Q components is broken.

On the other hand, an iterative receiver can significantly improve the performance compared with the noniterative receiver [7]. In iterative schemes, *a priori* information, which is generated by the decoder, is involved in demapping and imposes additional complexity.

Manuscript received March 1, 2015; revised April 23, 2015; accepted June 12, 2015. Date of publication July 13, 2015; date of current version October 30, 2015. This brief was recommended by Associate Editor G. Masera.

M. Rizk, A. Baghdadi, and M. Jézéquel are with the Department of Electronics, Telecom Bretagne, Le Centre national de la recherche scientifique (CNRS) unites mixtes de recherche (UMR) 6285 Laboratoire des Sciences et Techniques de l'Information, de la Communication et de la Connaissance (Lab-STICC), 29238 Brest, France (e-mail: mostafa.rizk@telecom-bretagne.eu; amer.baghdadi@telecom-bretagne.eu; michel.jezequel@telecom-bretagne.eu).

Y. Mohanna and Y. Atat are with the Faculty of Sciences, Lebanese University, Beirut, Lebanon (e-mail: yamoha@ul.edu.lb; youssef.atat@ul.edu.lb).

Color versions of one or more of the figures in this brief are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TCSII.2015.2455991

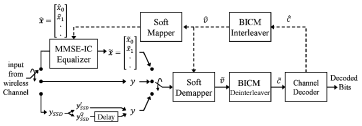


Fig. 1. Iterative receiver block diagram.

Most demapper implementations reported in the literature are of limited flexibility. In [8], the proposed soft-decision demapper architecture supports only four modulation schemes as specified in the DVB-S2 standard. Other demapper design has been presented in [9] for rotated QAM constellations targeting the DVB-T2 standard. To the best of our knowledge, only one universal demapper has been introduced in [2]. The demapper architecture is ASIP based and covers all flexibility requirements for recent wireless standards. Such wide flexibility to support different mapping styles, modulation schemes, SSD with rotated constellation, iterative and noniterative processing schemes becomes crucial in the current trend toward the convergence of wireless communication services [10] and the requirement of multistandard terminals. Furthermore, demonstrating the ability of designing highly flexible, yet efficient, demapping architectures can trigger the proposal of new modulation schemes and parameters that better suit the application and environment conditions. Such new schemes, associated with efficient flexible implementations, can then constitute potential candidates for adoption in next-generation communication systems.

II. System Model and Algorithm

Fig. 1 shows the block diagram of iterative receiver. Depending on the transmitter configuration and propagation conditions, the input from the wireless channel can be either directly delivered to the demapper or passed through a channel equalizer. To reduce the computational complexity, the demapper works in logarithmic domain and generates probabilities $\tilde{v}$ on received sequence in the form of log-likelihood ratios (LLRs), where v represents the binary mapping of the transmitted sequence. These LLRs construct, after deinterleaving, the input $\tilde{c}$ to channel the decoder. Through the feedback loop, the *a posteriori* information output from the decoder is interleaved and then fed back to the demapper and the equalizer. In this brief, the channel fading has a Rayleigh distribution with additive white Gaussian noise (AWGN). To compute the LLRs, the following expression is used [11]:

$$L\left(\tilde{v}_t^i\right) = \ln\left[\frac{\sum\limits_{x\in\mathcal{X}_1^i}\left(e^{-\frac{1}{2\sigma^2}|y_t-\rho_t\cdot x|^2}\cdot\prod\limits_{\substack{l=0\\ l\neq i}}^{m-1}P\left(\hat{v}_t^l\right)\right)}{\sum\limits_{x\in\mathcal{X}_0^i}\left(e^{-\frac{1}{2\sigma^2}|y_t-\rho_t\cdot x|^2}\cdot\prod\limits_{\substack{l=0\\ l\neq i}}^{m-1}P\left(\hat{v}_t^l\right)\right)}\right] \quad (1)$$

where m is the number of bits per symbol, $i = 0, 1, \ldots, m-1$, $L(\tilde{v}_t^i)$ is the LLR of the ith bit of transmitted symbol at time t, $\mathcal{X}_0^i$ and $\mathcal{X}_1^i$ are the symbol sets of constellation for which symbols have their ith bit equals $b \in \{0, 1\}$, ρ_t is the channel fading coefficient and σ^2 is the AWGN variance, and $P(\hat{v}_t^l)$ is the probability of lth bit of symbol x computed through *a priori* information. To reduce the complexity, max-log approximation [12] is applied by using the following formulas:

$$\ln\frac{a}{b} = \ln(a) - \ln(b)$$

$$\ln(e^{\delta_1+\cdots+e^{\delta_n}}) \approx \max_{i\in\{1,\ldots,n\}}\delta_i \max(a) - \max(b)$$

$$= \min(-b) - \min(-a).$$

The expression in (1) becomes

$$L\left(\tilde{v}_t^i\right) \approx \min_{x\in\mathcal{X}_0^i}(D - Ap_i) - \min_{x\in\mathcal{X}_1^i}(D - Ap_i) \quad (2)$$

where

$$D = \frac{\left|y_t^I - \rho_t^I\cdot x^I\right|^2 + \left|y_t^Q - \rho_t^Q\cdot x^Q\right|^2}{2\sigma^2} \quad (3)$$

$$Ap_i = \sum_{\substack{l=0\\ l\neq i \text{ and } v^l=1}}^{m-1} L\left(\hat{v}_t^l\right) \quad (4)$$

where v^l is the lth bit of each received modulated symbol.

The simplified expression in (2) exhibits four main computation steps: 2-D Euclidean distance computation, *a priori* LLR summation, minimum operations referred by the min functions, and subtraction operation of minimum values. In the case of noniterative demodulation, no *a priori* information is provided to the demapper. The expression of LLRs in (2) becomes

$$L\left(\tilde{v}_t^i\right) \approx \min_{x\in\mathcal{X}_0^i}(D) - \min_{x\in\mathcal{X}_1^i}(D). \quad (5)$$

Moreover, for Gray-mapped constellations, I and Q components are independent from each other; hence, the Euclidean distance is calculated in one dimension. In case where m is even, further simplification can be applied. LLR computation expression in (5) can be transformed in this case into the following [6]:

$$L\left(\tilde{v}_t^i\right) \approx \min_{x\in\mathcal{X}(I)_0^i}(D^I) - \min_{x\in\mathcal{X}(I)_1^i}(D^I) \text{ for } i = 0, 1, \ldots, \frac{m}{2} - 1 \quad (6)$$

$$L\left(\tilde{v}_t^i\right) \approx \min_{x\in\mathcal{X}(Q)_0^j}(D^Q) - \min_{x\in\mathcal{X}(Q)_1^j}(D^Q) \text{ for } j = \frac{m}{2}, \ldots, m-1 \quad (7)$$

where

$$D^I = \frac{\left|y_t^I - \rho_t^I\cdot x^I\right|^2}{2\sigma^2} \qquad D^Q = \frac{\left|y_t^Q - \rho_t^Q\cdot x^Q\right|^2}{2\sigma^2}$$

and $\mathcal{X}(I)_b^i$ and $\mathcal{X}(Q)_b^j$ are the constellation point sets on I-axis and Q-axis with ith and jth bits of symbol x that have a value equals to b. Applying this simplification, $2^{m/2}$ 1-D Euclidean distances are computed instead of 2^m 2-D Euclidean distances for each LLR. For rotated constellations, a simplification has been proposed in [9] to reduce the number of Euclidean distance computations by dividing the constellation into four subregions. This subpartitioning technique reduces the number of candidate constellation points involved in computing 2-D Euclidean distances of QAM schemes from 2^m to $(2^{(m-2)/2}+1)^2$.

III. Architecture Design

The variety of specifications in multiple wireless standards imposes designing hardware architecture of high flexibility to enable the computation of LLR values for different system

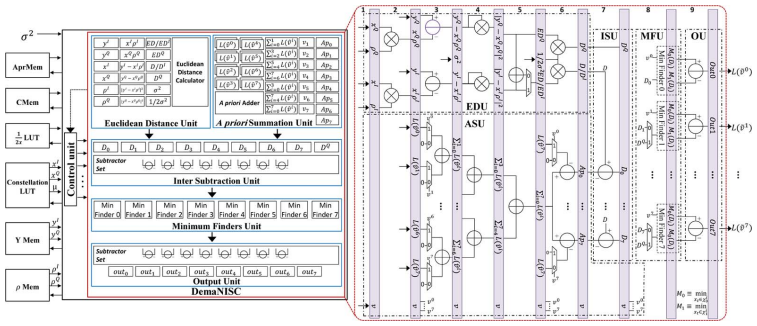


Fig. 2. Block diagram of the proposed NISC-based architecture detailing the pipeline structure of *DemaNISC* module.

configurations. The resources of the demapper flexibility are related to the characteristics of target constellation and the iterative demodulation concept. In this section, the architecture design proposed for the universal demapping is described. It is capable to generate soft-output probabilities in the logarithmic domain for various modulation schemes starting from BPSK up to 256-QAM with and without iterative demodulation using either the generic form or simplifications relative to the constellation rotation and the Gray mapping.

A. Architecture Choices

Toward achieving flexible demapper design, the following architecture choices are considered:

1) Modulation Order: Constellation information is stored in a lookup table (LUT) whose depth varies according to m. I and Q components of symbols and their corresponding binary mapping μ are rewritten for each target constellation. In addition, sufficient hardware operators, which are required to find minimum values and perform their corresponding subtractions required in (2), (5), (6), and (7), are allocated considering the largest target constellation (256-QAM). For lower order modulation schemes, unused resources are not activated. Sharing hardware resources among these operations decreases the throughput particularly for high-order modulation schemes. Furthermore, this architecture choice allows a fair comparison with the ASIP-based design presented in [2].

2) Mapping Style: The same hardware operators are utilized in computing Euclidean distances for Gray or non-Gray mapping styles. In fact, the computation of one 2-D distance is equivalent to that of two separate 1-D distances.

3) Iterative Demapping: Operators involved in *a priori* LLR summation are instantiated to accommodate all target constellations.

4) Quantization and Fixed-Point Arithmetic: To reduce the implementation complexity, fixed-point arithmetic is used, and computational values are quantized. Targeting a fair comparison with an ASIP-based design [2], identical quantization has been adopted for all computational parameters. With the aid of long simulations and analysis, bit widths are carefully selected to ensure least performance degradation. Using a reference software model, a degradation below 0.05 dB is measured at 10^{-3} frame-error rate over a fast fading Rayleigh channel.

5) Pipelining: Temporal parallelism using pipelining is applied to minimize the length of critical path and to enhance the computation performance and the efficiency of utilized hardware resources.

B. NISC Architecture

Fig. 2 presents the structure of the proposed architecture and shows the input/output connections. The inputs to the demapper architecture are the LLRs stored in *AprMem*, variance σ^2, *CWs* saved in *CMem*, constellation information arranged in *Constellation LUT*, and received symbols and fading factors reserved in *YMem* and *ρMem*, respectively. In addition, the LUT $(1/2x)$*LUT* provides the $1/2x$ inverse values required in the inversion operations.

The designed architecture is basically composed of a simple control unit and the module that performs the demapping functionality. Here, this module is referred to as *DemaNISC*.

1) Control Unit: It is mainly responsible for loading *CWs* from *CMem* into the components of *DemaNISC* module. To accomplish this functionality, the unit handles the address of *CMem* memory and constructs links to distribute the control-signal bits of *CWs* to appropriate components. In addition, the control unit manages the flow of input data coming from *YMem*, *ρMem*, and *Constellation LUT*. These basic tasks reveal the simple hardware structure required to implement the control unit.

2) DemaNISC Module: It is considered the main core of the demapper architecture. From a hierarchical scope, it can be viewed as a concatenation of five units.

a) EDU: This unit includes all hardware resources that incorporate in computing the Euclidean distance expressed in (3). It is supplied by I and Q components of received symbol y^I and y^Q, constellation symbol x^I and x^Q, and fading factor ρ^I and ρ^Q in addition to the noise variance σ^2. At each computation, Euclidean distance unit (EDU) can deliver two 1-D distances or one 2-D distance. EDU contains 18 registers, 6 real multipliers, 1 real adder, 2 real subtractors, and 1 2-to-1 multiplexer. The operators of the Euclidean distance calculator spread over five pipeline stages (stages 2–6, Fig. 2). In the second, third, and fourth pipeline stages, I and Q components of y, x, and ρ are exploited to compute two 1-D Euclidean distances. At the 5th pipeline stage, the two calculated distances may be added into one 2-D distance ED satisfying the implementation requirements of (2) and (5). In the case of Gray mapping style with no *a priori* information, the two 1-D distances are transferred to the next stage (ED^I and ED^Q). At this stage (fifth), the inverse value of σ^2, being provided at the $1/2x$ LUT index in the third pipeline stage, is retrieved. I and Q components of 1-D Euclidean distance (D^I and D^Q) or 2-D Euclidean distance D are ready at the end of the sixth pipeline stage (see Fig. 2).

b) ASU: In the case of turbo demodulation, the hardware resources embedded in this unit are responsible for generating

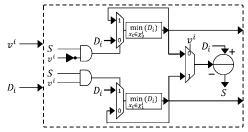


Fig. 3. Architecture of the minimum finder block.

the summation of input *a priori* LLRs as described in (4). The inputs to a priori LLR summation unit (ASU) are LLR values saved in *AprMem* memory and vector v representing the binary mapping μ of symbols from the *Constellation LUT*. ASU contains 38 registers, 7 real adders, 8 real subtractors, and 16 2-to-1 multiplexers. These components spread over five pipeline stages (stages 3–7, Fig. 2). The summation process is managed by the bit values of v, which represents the binary mapping of constellation symbol x under consideration and propagates along pipeline stages. In the case of turbo demodulation, input LLRs $L(\hat{v}_t^i)$, which are loaded in the second pipeline stage, are summed cumulatively in the third, fourth, and fifth stages. At this computational level, a copy of input *a priori* information is needed to subtract the LLR corresponding to bit v^i as expressed in (4). The final *a priori* information summations Ap_i corresponding to all bits are delivered at the end of the 6th pipeline stage.

c) ISU: This unit collects *a priori* LLR summation values produced by ASU and subtracts them in parallel from the value of 2-D Euclidean distance calculated by EDU. To perform this functionality, intersubtraction unit (ISU) includes a subtractor set, which is made of eight real subtractors considering the highest modulation order with $m = 8$. ISU also contains nine registers to store the subtraction results ($D_i = D - Ap_i$) in addition to the value of the Q-component of the 1-D Euclidean distance D^Q. Subtractors of ISU are placed in the seventh pipeline stage and are capable of producing up to eight D_i values (D_0 to D_7). If turbo demodulation is omitted, the computed values of Euclidean distance are transferred to the next pipeline stage (eighth) with no modifications. At the end of the seventh stage, three types of data are possible to be achieved:

- one 2-D distance minus the *a priori* information as expressed in (2);
- one 2-D distance only (noniterative demodulation case) as expressed in (5);
- two 1-D distances (Gray mapping with noniterative demodulation case) as expressed in (6) and (7).

d) MFU: This unit integrates eight minimum finder blocks whose architecture is presented in Fig. 3 and are established to realize minimum functions listed in (2), (5), (6), and (7) considering 256-QAM constellation. Each block is concerned to find minimum values associated to a bit location v^i along all constellation symbols. As shown in Fig. 3, the minimum finder structure contains two registers. The first register stores the updated minimum value that corresponds to symbol set $\mathcal{X}_0^i$; whereas the second register stores the minimum value corresponding to $\mathcal{X}_1^i$. For each new received symbol y, the two registers are initialized by loading the maximum numerical value. Each minimum finder benefits from new-updated D_i values in addition to v^i bits. Depending on v^i value ("0" or "1"), one of the two registers is chosen to be updated. The current value is replaced by input value D_i if the latter is smaller than the former. Otherwise, the register maintains its current value. A comparison is established by evaluating the sign S of the difference resulting from the subtraction operation of current value from input value D_i. At the input of each minimum finder block, a multiplexer is allocated to control input data flow to minimum finder block according to the required dimension of the Euclidean distance. Overall, minimum finder unit (MFU) is composed of 16 registers, 8 real subtractors, 31 2-to-1 multiplexers, 16 AND-gates, and 16 negators. All these components are placed at the eighth pipeline stage.

e) OU: This unit is in charge of delivering finally the LLR values corresponding to each bit $L(\tilde{v}_t^i)$ constellation symbol x. Inputs of the output unit (OU) are the minimum values available in the registers of MFU. Once minimum values of all constellation points are determined, the OU produces the difference between minimum pairs ($\min_{x_t \in \mathcal{X}_0^i}$ and $\min_{x_t \in \mathcal{X}_1^i}$) corresponding to each bit location v^i. After processing all constellation symbols, final resultant differences are loaded to their corresponding output registers (out_0 to out_7).

TABLE I
SYNTHESIS RESULTS

	Proposed design	*DemASIP* [2]
FPGA Synthesis Results (Xilinx Virtex5 xc5vlx330)		
Slice Registers	1,328 out of 207,360	1,918 out of 207,360
Slice LUTs	1,524 out of 207,360	3,201 out of 207,360
DSP48Es	6 out of 192	6 out of 192
Frequency	240 MHz	186 MHz
ASIC Synthesis Results (Synopsis Design Compiler)		
Technology	**65** nm **ST CMOS**	
Conditions	**nominal case** ($1V$; $25°C$)	
Area	0.048 mm^2	0.053 mm^2
Frequency	520 MHz	518 MHz

TABLE II
EXECUTION PERFORMANCE RESULTS

Modulation Type	**Clock Cycles** for 1 symbol		**Throughput (Mega LLR/second)** FPGA		ASIC	
	Proposed design	*DemASIP* [2]	Proposed design	*DemASIP* [2]	Proposed design	*DemASIP* [2]
Adopting Gray mapped constellations						
QPSK	3	4	160	93	347	259
16-QAM	5	6	192	124	416	345
64-QAM	9	10	160	111.6	347	310.8
256-QAM	17	18	112.94	82.67	244.7	230.2
Adopting DVB-T2 rotated constellations						
64-QAM	26	27	55.38	41.33	120	115.11
256-QAM	82	83	23.41	17.93	50.73	49.93

IV. RESULTS AND COMPARISON

In this brief, the NISC design toolset has been used. The adopted design flow is thoroughly described in [13]. Table I summarizes the synthesis results, whereas Table II shows the number of clock cycles required to produce LLRs for different system configurations along with the achieved throughput. The results are in addition compared with those of *DemASIP* [2], which is an ASIP dedicated for demapping with customized data path and instruction set. Both processors support same flexibility parameters, use identical computational resources, and adopt identical quantization for all computational parameters. The comparison between the two designs shows a significant improvement in terms of execution time and implementation area. For fair comparison, logic synthesis of the proposed architecture HDL description has been conducted targeting the same device (Xilinx Virtex-5 LX330 FPGA) and using the same synthesis options and tools. Logic utilization is reduced by 30.8% for slice registers and 52.4% for slice LUTs compared with *DemASIP*. In fact, the implementation of resources responsible for instruction fetching and decoding increases hardware resource utilization. Whereas, in the NISC-based proposed demapper, the architecture is designed

to match exactly the requirements of the application. Moreover, *DemASIP* can operate at a maximum frequency of 186 MHz, whereas the proposed architecture can achieve a maximum operating frequency of 240 MHz. Hence, it is 1.29 times faster than *DemASIP*. This is due to the *DemASIP* critical path that includes several levels of combinational logic and which is related to fetch program address generator, whereas only one level of logic exists in the critical path of the proposed design. In addition, Table I shows a comparison between the ASIC implementation of *DemASIP* and that of the NISC-based architecture on dedicated chips. In fact, the HDL code generated by Synopsys (ex CoWare) for the ASIP-based design (which is available at our research group) and the HDL code generated by NISC toolset for NISC-based design are delivered as sources to the Design Compiler tool from Synopsys to achieve the ASIC implementation targeting the ST CMOS 65-nm technology. The comparison shows that the NISC-based design offers about 10% area reduction compared with the ASIP-based design. Note that the detailed results illustrate that the control unit occupies only 1.6% of total implementation area of proposed architecture, whereas the components dedicated for fetching and decoding instructions occupy 7% of *DemASIP* implementation area.

On the other hand, operations in ASIP-based design are limited to the available instruction set. Overlapping of operations is not allowed. The instructions are fetched and decoded at runtime, and their functionalities are limited to their structure. Whereas, the NISC-based proposed design enables a direct and mastered access to control signals of hardware resources. Different operations are combined and then scheduled statically. Merging of operations ensures less execution time compared with *DemASIP*. At runtime, operations are directly performed, and LLRs are generated with no additional overhead. Table II shows that the proposed architecture outperforms *DemASIP* in all system configurations. For all combinations of mapping styles, modulation types, and SSD, better throughput is always provided.

Concerning *CMem* requirements, its width is significantly optimized (from 91 bits to 18 bits) by specifying same control bits to all components that have the same executions in all steps, whereas its depth varies according to the number of the needed execution steps to compute all LLRs corresponding to one input symbol. The required memory size varies from 23 B for quadrature PSK (QPSK) with Gray-mapped constellation to 594 B for 256-QAM with non-Gray mapped constellation. Compared with the ASIP program memory, *CMem* requires less memory space to be implemented. In fact, the assembly code used for *DemASIP* includes in addition to PROCESS instruction, which is the core instruction of LLR generation, instructions dedicated to loading input data, exporting output LLRs, looping, and no-operation instructions [2]. The available assembly code for QPSK non-Gray constellation shows that these additional instructions forms 54% of the total number of listed instructions [2]. For this system configuration, the memory space required to implement the program memory of *DemASIP* is 60.3% more than that required to implement the control memory (*CMem*) of the NISC-based proposed demapper. Note that, for both processors, CWs and assembly code are produced and optimized by hand as in this case the hardware is highly dedicated to the application, and it is programmed by its designers and not by its users.

Furthermore, the proposed demapper is compared with the dedicated architectures reported in [8] and [9], which use a classical RTL design approach. These architectures support soft-decision demapping; however, their flexibility is limited to the requirements of DVB-S2 and DVB-T2 standards, respectively. Targeting a fair comparison, our proposed design has been implemented on the same target devices used in [8] (Virtex II XC2-V6000) and [9] (Virtex II Pro XC2VP30). Compared with the architecture in [9], the proposed demapper requires almost 3.33 times less dedicated multipliers, 3.1 times less LUTs, but 2.2 times more registers. In terms of throughput, it outperforms the design in [9] for QPSK by 6.2 times, for 16-QAM by 3.1 times, and for 64-QAM by 1.2 times. Whereas for 256-QAM, the design in [9] shows better throughput (2.6 times) since it can concurrently compute nine Euclidean distances. In [8], the timing information about the hardware implementation is not available. The presented logic utilization summary reveals the need of 1.8 times more logic devices and 2.67 times more multipliers compared with the proposed NISC-based demapper.

V. Conclusion

In this brief, an NISC-based architecture of universal demapper for multiple wireless communication standards has been proposed. The flexibility of the demapper is not restricted to certain modulation types and/or mapping styles. Hardware design and implementation have been conducted using the NISC design approach. While using identical computational resources and supporting same flexibility parameters, the proposed NISC-based demapper architecture outperforms state-of-the-art ASIP-based architecture with reduced implementation costs. In addition, less memory space is required to implement control memory compared with the ASIP program memory.

References

[1] F. Vahid and T. Givargis, *Embedded System Design: A Unified Hardware/Software Introduction*. New Delhi, India: Wiley, 2006.

[2] A. R. Jafri, "Architectures multi-ASIP pour turbo recepteur flexible," Ph.D. dissertation, Elect. Dept., Telecom Bretagne, Brest, France, 2011.

[3] D. Gajski, "NISC: The ultimate reconfigurable component," Center Embedded Comput. Syst. (CECS), Univ. California, Irvine, CA, USA, Tech. Rep., Oct. 2003.

[4] M. Rizk *et al.*, "Flexible and efficient architecture design for MIMO MMSE-IC linear turbo-equalization," in *Proc. IEEE ICCIT*, Jun. 2013, pp. 340–344.

[5] E. Agrell *et al.*, "On the optimality of the binary reflected Gray code," *IEEE Trans. Inf. Theory*, vol. 50, no. 12, pp. 3170–3182, Dec. 2004.

[6] E. Akay and E. Ayanoglu, "Low complexity decoding of bit-interleaved coded modulation for M-ary QAM," in *Proc. IEEE ICC*, Jun. 2004, vol. 2, pp. 901–905.

[7] X. Chen *et al.*, "VLSI implementation of a high-throughput iterative fixed-complexity sphere decoder," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 60, no. 5, pp. 272–276, May 2013.

[8] J. W. Park *et al.*, "Low complexity soft-decision demapper for high order modulation of DVB-S2 system," in *Proc. IEEE ISOCC*, Nov. 2008, vol. 2, pp. II 37–II 40.

[9] L. Meng, C. Abdel Nour, C. Jego, and C. Douillard, "Design of rotated QAM mapper/demapper for the DVB-T2 standard," in *Proc. IEEE Workshop SiPS*, Oct. 2009, pp. 18–23.

[10] A. Osseiran *et al.*, "Scenarios for 5G mobile and wireless communications: The vision of the METIS project," *IEEE Commun. Mag.*, vol. 52, no. 5, pp. 26–35, May 2014.

[11] C. Abdel Nour, "Spectrally Efficient Coded Transmission for Wireless and Satellite Applications," Ph.D. dissertation, Dept. Elect., Telecom Bretagne, Brest, France, 2008.

[12] P. Robertson *et al.*, "Optimal and sub-optimal maximum a posteriori algorithms suitable for turbo decoding," *Eur. Trans. Telecom.*, vol. 8, no. 2, pp. 119–125, Mar./Apr. 1997.

[13] M. Rizk *et al.*, "Design and prototyping flow of NISC-based flexible MIMO turbo-equalizer," in *Proc. IEEE Int. Symp. RSP*, Oct. 2014, pp. 16–21.

Chapter 5

Efficient Algorithms and Architectures for Dataflow Applications

5.1 Notifying Memories Concept

5.1.1 Preface

This research topic has been addressed during my work at South Brittany University (UBS) in MOCS research team at Lab-STICC in Lorient, France in the context of the ANR COMPA research project. The notifying memory concept has been published in the proceeding of the *Design Automation Conference (DAC)* in 2016 [36]. An extension of the results along with further details about the devised simulator have been also published in [37].

5.1.2 Introduction

Considering the evolution towards manycore and multiprocessor architectures in embedded systems, the interconnect network has a strong impact on performance and energy efficiency. On the software side, we observe a renewed interest for dataflow (DF) programming models that offer clear design guidelines to deal with application complexity and scalability. It allows for explicitly specifying both spatial and temporal parallelism of an application. DF programming is also a very convenient approach to manage the evolution of standards based on a large set of reused functions and to generate correct-by-construction code. One of the side effects of dataflow applications is the overhead of memory accesses. DF actors (dataflow nodes) must check firing rules including availability of input data and output buffer space. This penalty is even more important for dynamic actors, which are unavoidable in real life applications.

NoCs implement concurrent communications, increasing the bandwidth and taking advantage of parallelism at the actor level. However, NoCs also increase communication latency, penalizing DF applications which rely on a large number of requests to memories for firing rule validation. Some recent work has addressed the problem of NoC latency. It relies on path or time slot allocation and processor or buffer mapping to minimize read and write

delays, but it is still based on the fact that memories are slaves [38]. The idea of smart or active memory aims also to reduce the number of transactions and improve memory access efficiency by moving part of computations to the memory side [39, 40].

In this research work, we address the question with a totally new approach that transforms memories into masters, able to initiate transfers by means of notifying actions when data is ready, so to get rid of useless accesses. We call this concept *Notifying Memories* (NM). The immediate consequence of a new balance between memory and processors is the reduction of the number of transactions and injection rates. It provides memories with notification and processors with listening mechanisms, which are conceptually close to the observer design pattern, used in software engineering. They are implemented in the network interface (NI). Our solution is thus independent from memory, processor architectures and NoC parameters. Moreover, the notifying memories perfectly match with the DF models. In real-life DF applications, both static and dynamic actors are required. The uncertainty of computing due to data-dependency prevents from any static scheduling.

5.1.3 Context and Prior Related Work

Many approaches rely on the same idea of the one we propose: reducing the number of transactions to save bandwidth for useful communications by trimming conflicts. In [40], an active memory processor is introduced. It is a processor that resides inside the memory controller to process data on the memory side. The computation on a set of data is done directly inside the memory and thus it reduces the number of transactions on the bus. In [39], the concept of smart memory is presented. It is a modular reconfigurable architecture that can be adapted to better match the application requirements. The goal is to propose a target that can efficiently execute applications specified with a wide range of programming models, from the stream programming model to speculative and random programming model. The works in that domain mainly focus on improving the efficiency of memory access [40]. By means of intelligent cache, memory and protocol controllers, the goal is to break the passivity of the memories. The contribution of this work follows the same trend in the way that memory is the best element to know what is in the memory. The main idea is to program memories so that they can trigger a notification. It supposes that the event is known in advance. The data-flow paradigm with the different models of computation makes this possible.

Executing data-flow applications onto an NoC-based architecture has been an active topic and gained renewed interest with the advent of parallel architectures [41]. Executing a data-flow application, naturally parallel, onto a parallel architecture seems obvious but the problem is actually more challenging [42]. The approaches in the data-flow domain propose to extend a baseline model of computation to support architectural features, to adopt a specific scheduling policy, or to tune the architecture to fit the programming model. Our approach is complementary to the existing works and is compatible with any DF model of computation (MoC) and any scheduling policy.

5.1.4 Motivations

Figure 5.1 presents the actor network model. An actor can contain several *actions*. An action is executed (*fired*) when a set of conditions is satisfied. This set is called a *firing rule*. This firing rule usually consists of checking that the number of tokens in the input First-In First-Out (FIFO) is greater than the number needed to compute, and that the output FIFO is empty enough to store the produced tokens. In this work, we propose the FIFO to notify directly the processor about its content. This idea is inspired from the observer design pattern that is used for software design.

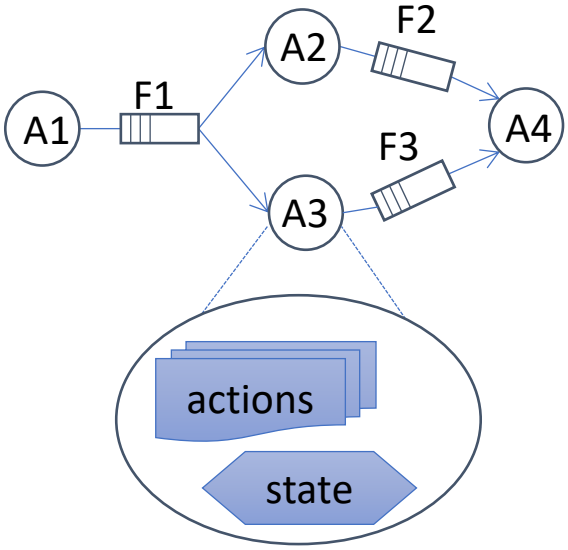


Figure 5.1 — Actor network model

This work stands in this context of data-flow application where different models of computation are used. We use as a case study an MPEG-4 Simple Profile decoder (MPEG4-SP) specified in RVC-CAL [43]. This decoder is specified with heterogeneous MoC and contains up do 40% of dynamic actors [44]. The Open RVC-CAL Compiler (Orcc) tool is used for compiling and software synthesis [45]. This research work relies on the C-backend that generates C code for multi-core platforms. Since the number of actors (about 40) is greater than the number of processing cores, an *actor scheduler* is needed. Different policies have been proposed, one of the most efficient one remains the Round Robin which is the the default in Orcc. They all have the same major drawback related to DF principles, which is the inefficient polling that leads to useless accesses to the memory when a scheduling attempt is not successful. Figure 5.2 presents the structure of the software FIFO generated by Orcc tool. It is composed of five parts:

1. size of the FIFO
2. FIFO content, where memory allocation is done according to the FIFO size and size of data type
3. number of readers, since one actor can write in a FIFO but there might be several readers

4. index of readers, each reader has its own index
5. index of writer

The synchronization is handled through indexes. Only one actor can write in a FIFO but there might be several readers. Each reader has its own index. In the simple case where there is one reader, the difference between the reader index and writer index is computed to know the number of tokens inside the FIFO. When multiple readers occur, the minimum index is used. It might happen that one slow reader blocks the other readers.

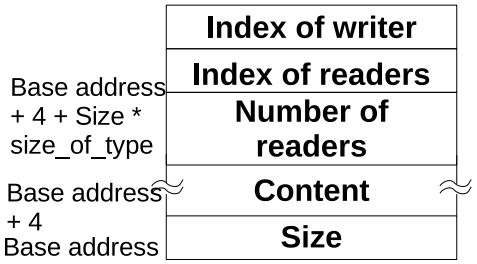


Figure 5.2 — Structure of a software FIFO generated by Orcc tool

Given this FIFO structure, the processor that executes an actor has to read the values of the different indexes in order to know the number of tokens in the FIFO. This leads to two memory requests for each of the concerned FIFO. Let us take the example of the firing rule given in listing 5.1. In order to compute `numTokens_FIFO1`, the number of tokens in the first FIFO, the processor emits two requests, one for the index of the writer, and one for the index of the reader. Note that in this example we consider only one reader for sake of simplicity. For the second FIFO, the processor emits another couple of two requests. Then, in order to check for the output FIFO, two other requests are emitted on the NoC. In C language, if the first condition is not satisfied, the whole test is stopped. The worst case is when the input conditions are satisfied but not the output condition, which would lead to six transactions for no action firing. Of course, next scheduling attempt will test again these conditions although the conditions on the input FIFO are satisfied. It has to be noticed that these conditions cannot become false on the next trial.

Listing 5.1 — Example of a firing rule

```
\centering
if (numTokens_FIFO_IN1 >= 64  && numTokens_FIFO_IN2 >= 1){
   if(SIZE_FIFO_OUT - numTokensFIFO_OUT > 64){
        // fire action
}}
```

Some experiments are conducted using the C backend of Orcc by executing the MPEG4-SP decoder on a desktop computer in order to trace the number of firings of each actor. The number of zero firings (no action could be fired) is determined out of the total number of scheduling attempts. Figure 5.3 shows the percentage of useless scheduling attempts for different video sequences from [46]. There are two reasons why no action can be fired:

1. one of the input FIFO is empty (i.e. does not contain enough tokens)

2. one of the output FIFO is full (i.e. does not contain enough space)

Figure 5.3 also shows the repartition between empty and full FIFO. The obtained results show that at least 20% of scheduling attempts suffer failure. Although the lack of tokens in the input FIFO seams to be the major reason, the disparity among the different video sequences prevents from drawing any clear conclusion.

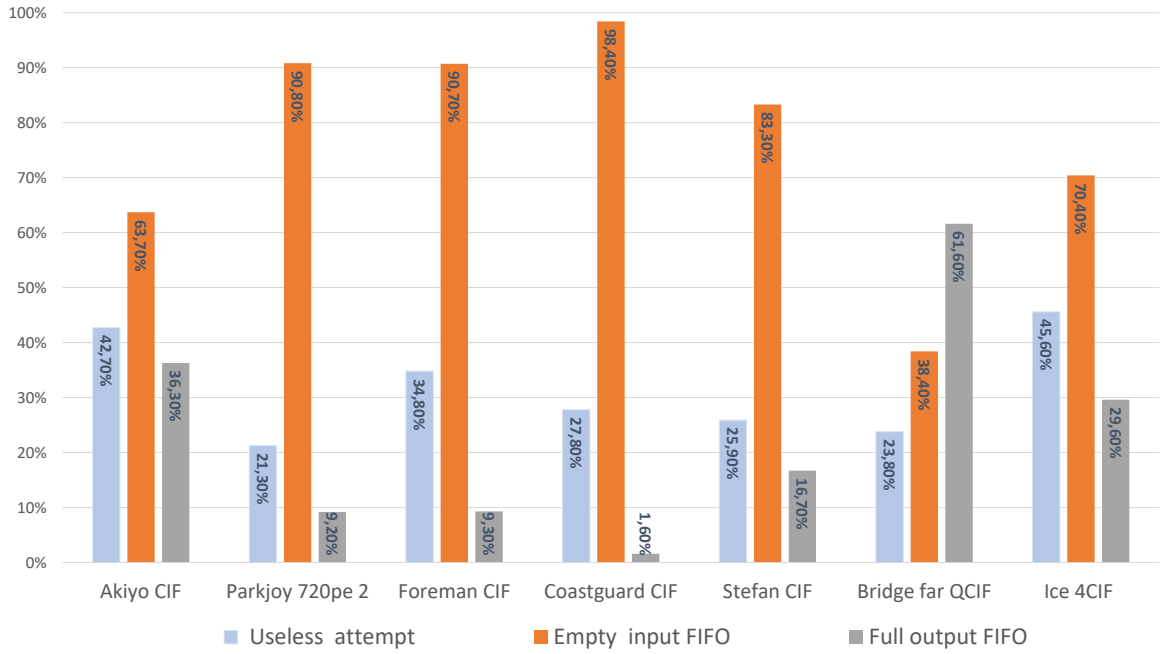


Figure 5.3 — Useless scheduling attempts by the MPEG4-SP decoder for different video

5.1.5 Contributions and Performed Work

This research work has focused on how to delete useless memory requests, independently from the implemented scheduling policy although combining our approach with new scheduling policy is relevant. The main issue is to stop the polling on the NoC because:

- It is useless when no action can be fired.
- It consumes bandwidth that would be useful for effective transactions.

The current situation is that memories are subjected to processor requests. The idea is to give new capabilities to memories so that they can inform the concerned processors that their FIFO content has changed.

Our proposition is inspired from the software observer design pattern. The observer design pattern *defines a one-to-many dependency between objects so that when one object changes state, all its dependents are notified and updated automatically* [47]. That behavior is exactly what we expect when the content of a FIFO changes: notify the concerned processors that execute the actors connected to that FIFO.

The main contributions of this research work are:

- Introducing the Notifying Memories concept, which includes a notifier module in the NI of the memories and a listener module in the NI of the processors. For each devised module, the structure and functionality are identified. Accordingly, their hardware architectures are designed.

- Designing an NoC-based architecture which integrates the notifier and listener moudules. The NIs are modified to accommodate the devised modules without impacting the typical processes. To cope with the NM concept and to remain compliant with any existing processor or memory and to be independent from NoC parameters such as topology, router buffer depth and routing policy, the new elements are integrated into the NI of the NoC. Besides, we need a master component that can send packets through the network and a component that can monitor requests, the NI is the only component to offer such features.

- Demonstrating a model and an implementation of the devised NoC-based architecture in SystemC TLM. A cycle-accurate simulator that models the NoC and connected nodes has been created. The devised model executes an MPEG4-SP decoder with 41 actors and 70 FIFOs specified in RVC-CAL.

- The mean values of execution time of all actions are found from profiling data on a desktop computer. These values are imported to the simulator model and utilized in order to compensate the execution time of actions, which are functionally simulated.

- Introducing a new type of packet, the notification packet. The structure of the packet is described and implemented. The modules of the NIs dedicated to make/de-make packets are modified to generate/parse the new packet type.

- Conducting experimental simulations using the devised simulation model running a real-life MPEG4-SP decoder for different video sequences from [46] in order to demonstrate the interest of the notifying memories concept in terms of performance, data traffic and power consumption. Note that the selected video sequences differ in their characteristics (motion, background, etc.) and formats (QCIF, CIF, 4CIF, etc.).

- In order to check the relevancy of the proposed concept, the obtained results when adopting NM are compared with the ones obtained when adopting ordinary memories while using identical NoC features. The comparison is demonstrated in terms of latency, throughput, injection rate and switch conflicts. As the NM concept induces sending additional data related to the notifier configuration and notifications, a detailed investigation is conducted to determine the impact of new transferred flits on the NoC traffic.

- Preliminary synthesis is achieved to estimate the impact of NM concept on implementation area and power consumption. The implemented design integrates a worst-case design, where a single notifier, which is implemented in all memory NIs, and a single listener, which is implemented in all processor NIs, can manage all firing rules of all actors. The size of the registers in all listener and notifier modules are adjusted to accommodate the signals for all actors. The area and power results are obtained with the Cadence Encounter RTL Compiler RC12.24 tool. The synthesis targets the 65nm

process technology at 500 MHz operating frequency and $25°C$. Total power was obtained by using the leakage and dynamic power of the NoC components relying on the switching activity traced by the SystemC simulation.

5.1.6 Findings

The comparison of the results when adopting notifying memories with those obtained when adopting ordinary memories shows significant reductions in terms of latency, injection rate , switch conflicts and total number of flits along with remarkable throughput improvement. The obtained results confirm the efficiency of NM concept and show that adopting notification memories leads to great reductions reaching 75.14% for latency, 53.96% for injection rate, 58.25% for transported flits, and 75.88% for switch conflicts. Also, the throughput enhancement is improved by up to 9.69%.

The synthesis results show that the NoC adopting notifying memories saves up to 49.1% of power consumption compared to the reference NoC. Besides, the power overhead of the interfaces of the proposed NoC presents a modest value of 16.3%. Regarding the area, the proposed NoC presents an overhead of 12.4%, when compared to reference NoC.

The conducted comparison between the number of extra flits for mapping and notification and the number of flits required to request and retrieve FIFO indexes shows that the added flits for notification and extra mapping in the case of notification memories are negligible compared to the required flits to request and retrieve FIFOs indexes in ordinary memories.

Notifying Memories: a case-study on Data-Flow Applications with NoC Interfaces Implementation

Kevin J. M. Martin[1], Mostafa Rizk[1], Martha Johanna Sepulveda[2], Jean-Philippe Diguet[1]
[1]Univ. Bretagne-Sud, CNRS UMR 6285, Lab-STICC, F-56100 Lorient, France
[2]Institute for Security in Information Technology, Technical University of Munich, Germany
{kevin.martin,mostafa.rizk,jean-philippe.diguet}@univ-ubs.fr,
johanna.sepulveda@tum.de

ABSTRACT

NoC-based architectures overcome the limitations of traditional buses by exploiting parallelism and offer large bandwidths. NoC adoption also increases communication latency, which is especially penalising for data-flow applications (DF). We introduce the notifying memories (NM) concept to reduce this overhead. Our original approach eliminates useless memory requests. This paper demonstrates NM in the context of video coding applications implemented with dynamic DF. We have conducted cycle accurate systemC simulation of the NoC on an MPEG4 decoder to evaluate NM efficiency. The results show significant reductions in terms of latency (78%), injection rate (60%), and power savings (49%) along with throughput improvement (16%).

CCS Concepts

•**Networks** → **Network on chip;** •**Theory of computation** → *Streaming models;* •**Hardware** → *Buses and high-speed links;*

Keywords

NoC based architecture; data-flow; memory

1. INTRODUCTION

Considering the evolution towards manycore and multiprocessor architectures in HPC and embedded systems, the interconnect network has a strong impact on performance and energy efficiency. On the software side, we observe a renewed interest for dataflow (DF) programming models that offer clear design guidelines to deal with application complexity and scalability. It allows for explicitly specifying both spatial and temporal parallelism of an application. DF programming is also a very convenient approach to manage the evolution of standards based on a large set of reused functions and to generate correct-by-construction code. One of the side effects of data-flow applications is the overhead of memory accesses. DF actors (data-flow nodes) must check firing rules including availability of input data and output buffer space. This penalty is even more important for dynamic actors, which are unavoidable in real life applications.

Networks-on-Chips (NoCs) implement concurrent communications, increasing the bandwidth and taking advantage of parallelism at the actor level. However, NoCs also increase communication latency, penalising DF applications which rely on a large number of requests to memories for firing rule validation. Some recent work has addressed the problem of NoC latency. It relies on path or time slot allocation and processor or buffer mapping to minimise read and write delays, but it is still based on the fact that memories are slaves [4]. The idea of smart or active memory aims also to reduce the number of transactions and improve memory access efficiency, by moving part of computations to the memory side [1, 11].

In this work, we propose to address the question with a totally new approach that transforms memories into masters, able to initiate transfers by means of notifications when data is ready, so to get rid of useless accesses. We call this concept *Notifying Memories* (NM). The immediate consequence of a new balance between memory and processors is the reduction of the number of transactions and injection rates. It provides memories with notification and processors with listening mechanisms, which are conceptually close to the observer design pattern, used in software engineering. They are implemented in the network interface (NI). Our solution is thus independent from memory, processor architectures and NoC parameters. Moreover, the notifying memories perfectly match with the DF models. In real-life DF applications, both static and dynamic actors are required. The uncertainty of computing due to data-dependency prevents from any static scheduling. We need to demonstrate the efficiency of the approach with real application execution, so we prove the concept and the implementation with cycle accurate SystemC simulation. We have chosen an MPEG4-SP decoder and related benchmark videos to show how we can significantly improve throughput, latencies, injection rate and so power consumption.

The contributions of this paper are:

- The Notifying Memories concept, which includes a notifier in the NI of the memories and a listener in the NI of the processors.
- A model and an implementation of the notifying memory in SystemC. It includes the introduction of a new kind of packet, the notification packet.
- Experimental results running an MPEG4-SP decoder

DAC '16, June 05-09, 2016, Austin, TX, USA

DOI: http://dx.doi.org/10.1145/2897937.2898051

for different video sequences that demonstrate the interest of the concept in terms of performance, injection rate and power consumption.

Section 2 presents the related work. Section 3 introduces the initial situation and a motivational example. Section 4 reminds briefly the observer design pattern and presents in details the notifying memories. Section 5 shows the results and Section 6 concludes this paper.

2. RELATED WORK

Many approaches rely on the same idea of the one we propose: reducing the number of transactions to save bandwidth for useful communications by trimming conflicts. An active memory processor is introduced in [11]. It is a processor that resides inside the memory controller to process data on the memory side. As the computation on a set of data is done directly inside the memory, it reduces the number of transactions in the NoC. In [1], the concept of smart memory is presented. It is a modular reconfigurable architecture that can be adapted to better match the application requirements. The goal is to propose a target that can efficiently execute applications specified with a wide range of programming models, from the stream programming model to speculative and random programming model. The work in that domain mainly focuses on improving the efficiency of memory access [11]. By means of intelligent cache, memory and protocol controllers, this kind of work breaks the passivity of the memories. Our contribution follows the same objective but is compliant with any memory controller. Our idea is to program memories so that they can trigger a notification according to some events. These events can be application specific or related to the programming model. The DF models of computation (MoC) are based on a paradigm that makes our approach particularly efficient.

Executing DF applications onto an NoC-based architecture is an active topic and gained renewed interest with the advent of parallel architectures [5]. Executing a DF application, naturally parallel, onto a parallel architecture seems obvious but the problem is actually more challenging [8]. One commercial example is the MPPA programmed with sigmaC [7]. The approaches in the DF domain propose to extend a baseline MoC to support architectural features, to adopt a specific scheduling policy, or to tune the architecture to fit with the programming model. Our approach is complementary to existing works and is compliant with any DF MoC and any scheduling policy.

3. MOTIVATIONAL EXAMPLE

Figure 1 presents the actor model of the Caltrop Actor Language (CAL). An actor can contain several *actions*. An action is executed (*fired*) when a set of conditions is satisfied. This so-called *firing rule* usually consists of checking that the number of tokens in the input FIFO is greater than the number needed to compute, and that the output FIFO is empty enough to store the produced tokens. In this paper, we propose the FIFO to notify directly the processor about its content. This idea is inspired from the observer design pattern, used in software engineering.

We use as a case study an MPEG-4 Simple Profile decoder (MPEG4-SP) specified in RVC-CAL [6]. This decoder is specified with heterogeneous MoC and contains up do 40%

Figure 1: CAL actor model [3]

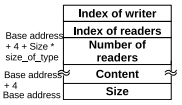


Figure 2: Structure of a SW FIFO generated by ORCC

```
if (numTokens_FIFO_IN1 >= 64
 && numTokens_FIFO_IN2 >= 1){
   if(SIZE_FIFO_OUT -
    numTokensFIFO_OUT > 64){
        // fire action
}}
```

Listing 1: Example of a firing rule

of dynamic actors [9]. The ORCC tool is used for compiling and software synthesis [6]. Our work relies on the C-backend that generates C code for multi-core platforms. Since the number of actors (41) is greater than the number of processing cores, an *actor scheduler* is required. Different policies have been proposed, one of the most efficient one remains the round-robin (the default in ORCC). They all have the same major drawback related to DF principles, which is the inefficient polling that leads to useless accesses to the memory when a scheduling attempt is not successful.

Figure 2 presents the structure of the SW FIFO generated by ORCC [6]. It is composed of five parts: i) size of the FIFO; ii) FIFO content, where memory allocation is done according to the FIFO size and size of data type; iii) number of readers, since one actor can write in a FIFO but there might be several readers; iv) index of readers, each reader has its own index; and v) index of writer. Synchronisation is handled through indexes. The difference between the reader index and writer index determines the number of tokens inside the FIFO. When multiple readers occur, the minimum index is used. It might happen that one slow reader blocks the other readers. Many papers deal with FIFO sizing and FIFO handling but it is out of the scope of this paper. Given this FIFO structure, the processor that executes an actor has to read the values of the different indexes in order to determine the number of tokens in the FIFO. This leads to a set of memory requests for each FIFO. Taking the example of the firing rule given in Listing 1, to compute `numTokens_FIFO_IN1`, the number of tokens in the first FIFO, the processor emits two requests, one for the index of the writer, and one for the index of the reader (one for each reader actually). For the second FIFO, the processor emits another set of requests. Then, in order to check for the output FIFO, other requests are emitted on the NoC. In C language, if the first condition is not satisfied, the whole test is stopped. The worst case occurs when the input conditions are satisfied but not the output condition, which would lead to six transactions for no action firing. Of course, next scheduling attempt will test again these conditions although the conditions on the input FIFO are satisfied. It has to be noticed that true conditions cannot become false on the next trial. Our contribution also relies on this property.

Table 1: Unsuccessful scheduling by the MPEG4-SP decoder for different video sequences and formats

Video		Useless	Empty	Full
Sequence	Format	attempt	input FIFO	output FIFO
Akiyo	CIF	42.7%	63.7%	36.3%
Parkjoy	720p	21.3%	90.8%	9.2%
Foreman	CIF	34.8%	90.7%	9.3%
Coastguard	CIF	27.8%	98.4%	1.6%
Stefan	CIF	25.9%	83.3%	16.7%
Bridge far	QCIF	23.8%	38.4%	61.6%
Ice	4CIF	45.6%	70.4%	29.6%

We have carried out some experiments using the C backend of ORCC and executed the MPEG4-SP decoder on a desktop computer. We have traced the number of firings of each actor during one scheduling attempt. We have counted the number of zero firings, i.e. no action could be fired, out of the total number of scheduling. Table 1 shows the percentage of unsuccessful scheduling for different video sequences from [10]. There are two reasons why no action can be fired: 1) one of the input FIFO is empty (i.e. does not contain enough tokens); or 2) one of the output FIFO is full (i.e. does not contain enough space). Table 1 also shows the distribution between empty and full FIFO. These results show that at least 20% of scheduling attempts are unsuccessful. Although the lack of tokens in the input FIFO seams to be the major reason, the disparity among the different video sequences prevents from drawing any clear conclusion. This observation motivates the integration of mechanisms able to monitor the FIFO status and to emit notifications. This mechanism can be integrated in the memory NI, close to the FIFO implementation.

This paper focuses on how to delete useless memory requests, independently from the processors, memories, NoC parameters and scheduling policy[1]. The main issue is to stop the polling on the NoC that: 1) is useless when no action can be fired; and 2) consumes bandwidth that would be useful for effective transactions. The current situation is that memories are subjected to processor requests. The idea is to give new capabilities to memories, so that, they can inform the concerned processors that their (FIFO) content has changed.

4. THE NOTIFYING MEMORIES

Our idea is inspired from the observer design pattern, widely used in software engineering. Before detailing the implementation in the NoC of the observer pattern, a brief description of the software pattern is given.

4.1 The observer design pattern

The observer design pattern *defines a one-to-many dependency between objects so that when one object changes state, all its dependents are notified and updated automatically* [2]. That behaviour is exactly what we expect when the content of a FIFO changes: notify the concerned processors that execute the actors connected to that FIFO. Figure 3 shows the UML class diagram of the observer design pattern. The subject is the element to watch. The observer is the element that should react whenever a change occurs from the subject. The subject notifies a change to their observers by means of a method.

[1]although combining our approach with a new scheduling policy is relevant

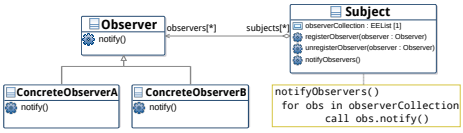


Figure 3: UML Diagram of Observer Design Pattern for software implementation

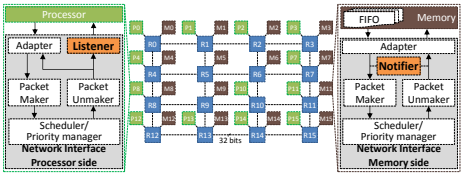


Figure 4: The structure of the used NoC adopting the notifying memories concept

4.2 NoC Implementation

Porting the observer design pattern to our context, the memory is the subject and the processor is the observer. There are two elements to be added in the NoC platform: the notifier and the listener. In order to remain compliant with any existing processor or memory and to be independent from NoC parameters such as topology, router buffer depth and routing policy, we have decided to integrate these elements into the NI of the NoC. Besides, we need a master component that can send packets through the network and a component that can monitor requests, the NI is the only component to offer such features.

Figure 4 illustrates the structure of the NoC used to demonstrate the notifying memory concept. The NoC is a 4×4 mesh-based network which interconnects 28 IP cores (13 processing and 15 memory nodes). It is based on usual wormhole packet switching mode, deterministic XY routing algorithm to avoid deadlocks, and flow control policy without virtual channels. The routers have one arbiter per port and one buffer per input port. Our approach is actually generic and can be applied to NoC with different features. For instance, N-flit buffers can be used to improve performance at the cost of more memory, in that case all transactions including notifications will take advantage of it. The backend part of the NI is typical, it includes a packet maker and packet un-maker to assemble and disassemble the packets, a scheduler/priority manager to synchronize packet transmission and reception. The modifications lie in the front-end of the network interface by either implementing the notifier or the listener. The notifier is implemented in the NI of a memory. The listener is added in the NI of a processor. The structures of the additional components are detailed in the following subsections.

4.2.1 *Notifier*

The notifier is a hardware module that transmits the status of all FIFOs allocated in a node accommodating memory. For each FIFO, the notifier generates a notification sig-

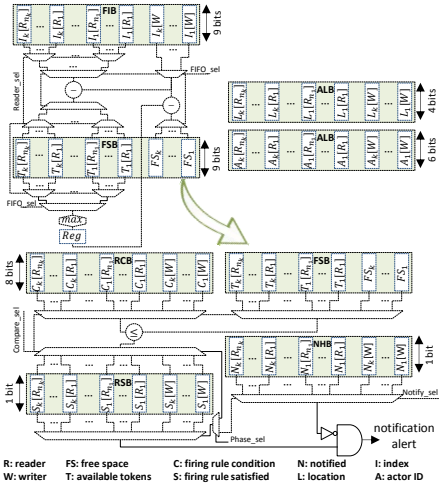


Figure 5: The notifier architecture

Algorithm 1 Checking firing rules satisfaction

Input: k the number of FIFOs, n_f the number of readers of FIFO f

```
for all f ∈ {1, ..., k} do
  if I_f[W] modified then
    for all i ∈ {1, ..., n_f} do
      T_f[R_i] = I_f[W] − I_f[R_i]
      if T_f[R_i] ≥ C_f[R_i] then
        S_f[R_i] = true
      end if
    end for
  end if
  for all i ∈ {1, ..., n_f} do
    if I_f[R_i] modified then
      FS_f = I_f[W] − max_{i∈{1,...,n}} (I_f[R_i])
      if FS_f ≥ C_f[W] then
        S_f[W] = true
      end if
    end if
  end for
end for
```

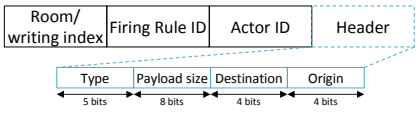


Figure 6: The composition of notification packet

nal that is passed to the FIFO's writer whenever it contains enough space to save new tokens, or to the FIFO's readers whenever enough tokens are available. The notifier architecture is illustrated in Figure 5. It is basically composed of a comparator, 2 subtractors, a maximum finder, an and-gate, an inverter, and set of registers classified in seven banks: FIFOs indexes bank (FIB), FIFOs status bank (FSB), firing rule conditions bank (RCB), firing rule satisfaction bank (RSB), notification history bank (NHB), actors ID bank ($AIDB$) and actors locations bank (ALB). The notifier functionality can be divided into three phases: the configuration phase, the checking phase, and the notification phase.

Configuration phase.

Once the system is launched, the manager processor provides the mapping data to all notifiers. A packet is sent to each memory node specifying the number of involved FIFOs in the notifier scope. Also, it specifies for each involved FIFO the locations (ID of processing nodes) of its writer and readers as well as their firing rule conditions and the IDs of their corresponding actors. The registers of ALB, FCB, and AIDB store this information respectively to be used in the next two phases. The registers of the other banks are reset.

Checking phase.

When a node receives a new packet modifying one of the FIFOs' indexes, the packet un-maker provides the notifier by the FIFO address (f), and its modified writing index ($I_f[W]$) or i^{th} reading index ($I_f[R_i]$). The notifier stores the new received index in its specific register in FIB. If the writing index of FIFO f has changed, the notifier recomputes the number of available tokens for all its readers ($T_f[R_i]$). If one of its reading indexes is modified, the notifier recomputes the available free space (FS_f) in FIFO f.

The notifier assigns the computed values to their corresponding registers of FSB. Later, the notifier compares the updated values in FSB with their relative firing rules conditions saved in RCB. If a value in FSB equals or exceeds its relative value in FCB, a "1" is stored in RSB as an indication of satisfaction of the firing rule condition. The notifier checking algorithm is outlined in Algorithm 1.

Notification phase.

In this phase, the notifier loops around the bits of RSB in a round-robin manner and generates consecutively the notification signals, which should be sent to targets (readers and writers) with satisfied firing rules. The notifier provides the packet maker with the target location, identity number, satisfied firing rule identity number, and the number of available tokens or free space. The packet maker assembles the notification packet as depicted in Figure 6. To reduce the number of sent packets, the notification history is recorded in NHB which reserves one bit for each target. When a notification is sent, the target corresponding bit in NHB is set to "1", and it is reset to "0" when the notification is consumed by the target. By using simple logic (and-gate and inverter) the notifier limits the generation of notifications to targets with "0" in their corresponding bits.

4.2.2 Listener

The listener stores the information included in the notification packets sent by notifiers. The coupled listener to each processor specifies the validation statuses of all firing rules related to all actions of all actors mapped to the processor. The processor accesses the validation status of all firing rules corresponding to an action before it is fired. In addition to the status, the listener stores the value of the available tokens at the input FIFO and the free space at the output FIFO for each input and output firing rules respectively. The information is thus available locally to the processor and no memory requests through the NoC are needed. The

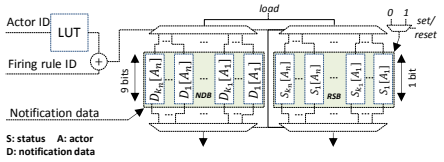


Figure 7: The listener architecture

listener architecture is illustrated in Figure 7. It is composed of a look-up table (LUT), an adder, and set of registers classified in two banks: the firing rule status bank (*RSB*) and the notification data bank (*NDB*). The listener functionality is divided into two phases: the configuration phase and the execution phase.

Configuration phase.
The listener benefits from the mapping information provided to the processor at system boot. It acquires the specification of the mapped actors (number, identification number, number of firing rules) to set its configuration. The look up table (LUT) is configured such that the starting address specified to store notifications of each mapped actor is retrieved at the LUT output when the actor identification number is provided at the LUT input.

Execution phase.
When a node receives a notification packet, the payload is provided to the listener. The actor ID is given to the LUT input; whereas the firing rule ID is used as an offset to determine the storage index j. Accordingly, the listener stores the notification data (free space or available tokens) in the j^{th} register of NDB and set the j^{th} bit of RSB to "1" indicating the validation of the firing rule. To access the notification information of a specific rule, the processor uses the same mechanism of loading data to the listener by providing the actor ID and the firing rule ID. Once the notification information is consumed, its relative bit in RSB is reset to "0".

4.3 DF-level deadlock avoidance

In the DF paradigm, an actor in non preemptive. Once fired, an action consumes and produces the tokens according to the firing rule. The indexes of the FIFOs are updated afterwards (conservative synchronisation scheme). Since the memory requests are triggered by the notifications and the execution of the action is not preemptive, there can never be any race conditions. A valid firing rule at time t is still valid at time $t+1$ if the associated action has not been fired.

5. EXPERIMENTS AND RESULTS

5.1 Experimental Setup

In order to check the relevancy of the proposed approach, the adopted NoC implementing notifying memories has been described in SystemC TLM model as a proof of concept. The devised model executes an MPEG4-SP decoder with 41 actors and 70 FIFOs specified in RVC-CAL. The number of FIFOs are approximately equally distributed on all nodes accommodating memories. The actors are then mapped manually such that the number of hops is minimized between one actor and its FIFOs. The SystemC model is cycle accurate at the NoC level and network interface; whereas actions are functionally simulated. We strive to accurately reproduce the timing features of actions executions in addition to their functionalities. The mean values of execution time of all actions are imported from profiling data on a desktop computer. Multiple simulations have been conducted to decode ten frames for several video sequences from [10]. To evaluate the efficiency of NM concept, the obtained results are compared with the ones of ordinary memories. Both models use identical NoC features (e.g. 500 MHz frequency, switching mode, routing algorithm), processing elements features, and mapping strategy.

Table 2: Results of decoding 10 frames of ice video sequence in 4CIF format

Parameter	Notifying memory	Ordinary memory	gain
Latency (μs)	143.42	665.06	-78.44%
Throughput (frames/s)	27.53	23.29	+15.41%
Injection rate(flits/s)	60 167 732	121 635 294	-50.53%
Switch conflicts	71 182 509	288 574 519	-75.33%
Transported flits	109 264 000	261 123 000	-58.16%
Transported packets	15 376 400	107 050 000	-85.64%

5.2 Results and Comparison

5.2.1 Performance results

Table 2 shows the results obtained after decoding 10 frames of ice video sequence in 4CIF format in terms of throughput, latency[2], injection rate, switch conflicts, total number of transported flits, and power consumption. The comparison between the two cases, ordinary and notifying, shows significant reductions in terms of latency, injection rate, switch conflicts, and number of transported flits and packets. Also the throughput is improved such that it is compatible to the 25 frames per second standard without using additional hardware accelerators or processing speedups.

The analysis of the results reveals an additional traffic overhead in case of ordinary memories which increases significantly the injection rate and switch conflicts. To track its cause, packets are classified according to their functionality into two categories: data and control packets. Packets holding tokens or requests for reading tokens are data packets, while control packets category includes all other packets that are used to transport mapping information, set FIFO indexes, request or retrieve FIFO indexes, and notification signals produced by notifiers. This classification is also applied to flits. Figure 8 presents a comparison between notifying and ordinary memories in terms of control and data categories for transported packets and flits for the ice-4CIF video. The figure shows that ordinary memories induce 19 times more control packets and 10 times more control flits than notifying memories. The number of data packets and flits is approximately the same in the two cases. The simulation is stopped after 10 fully decoded frames while the network contains partially decoded frames. This explains the traffic overhead, due to the additional control packets injected to the NoC. Investigating thoroughly the types of transported control flits shows on one hand that more flits

[2]the time between the first token consumed and the first token produced by the application

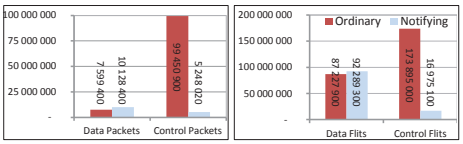


Figure 8: Classification of packets and flits transported after 10 decoded frames of ice-4CIF

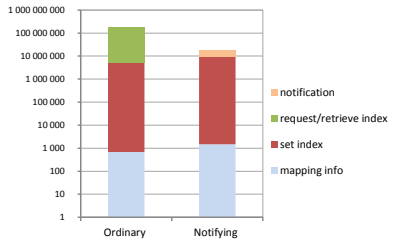


Figure 9: Classification of control flits according to their types after 10 decoded frames of ice-4CIF

are needed for mapping information when notifying memories is adopted since the manager have to send additional information for all notifiers. Also, additional notification flits are transported from notifiers to the listeners. On the other hand, the adoption of notification memories eliminates the use of flits to request and retrieve FIFO indexes. Figure 9 shows in logarithmic scale the number of each type of control flits transported while decoding 10 frames of ice video sequence in 4CIF format for the case of notification memories and ordinary memories. It shows that the added flits for notification and extra mapping in the case of notification memories are negligible (4.58%) compared to the required flits to request and retrieve FIFOs indexes in ordinary memories.

Due to space limitations, the comparisons for other video sequences are summarized in Table 3. These average results confirm the efficiency of NM concept and show that adopting notification memories leads to great reductions reaching 78% for latency, 60% for injection rate, 67% for transported flits, and 85% for switch conflicts. Also the throughput enhancement is improved by up to 16%.

5.2.2 *Preliminary synthesis results*

We have implemented a *worst-case* design, where a single notifier, which is implemented in all memory NIs, and a single listener, which is implemented in all processor NIs, can manage all firing rules of all actors. The number of firing rules of in the MPEG4-SP application is 145. Hence, 145 registers are required in all banks of the 12 listeners (one per processor) and 15 notifiers (one per memory). The area and power results are obtained with the Cadence Encounter RTL Compiler RC12.24 tool. The synthesis targets the 65nm process technology at 500 MHz operating frequency and 25°C. Total power was obtained by using the leakage

Table 3: Notification memory gain for decoding 10 frames of different video sequences

Video Sequence	Format	Throughput	Latency	Injection rate	Switch conflicts	Flits number
Bridgefar	QCIF	+15.53%	-73,96%	-45,80%	-71,38%	-54,22%
bus	CIF	+2.84%	-73,79%	-53,40%	-72,90%	-54,73%
grandma	QCIF	+16.79%	-68,96%	-60,78%	-85,50%	-67,36%
foreman	CIF	+14.26%	-78,41%	-46,81%	-72,86%	-54,39%
ice	4CIF	+15.41%	-78,44%	-50,53%	-75,33%	-58,16%

and dynamic power of the NoC components relying on the switching activity traced by the SystemC simulation of 10 decoded frames of the ice-4CIF video sequence. The results show that the NoC adopting notifying memories saves up to 49.1% of power consumption compared to the reference NoC. Besides, the power overhead of the interfaces of the proposed NoC presents a modest value of 16.3%. Regarding the area, the proposed NoC presents an overhead of 12.4%, when compared to reference NoC.

6. CONCLUSIONS AND FUTURE WORK

This paper presents the notifying memories, a hardware implementation based on the observer design pattern, to enhance communication of data-flow application that are deployed on a multi-processor platform. The notifying memories are baseline memories extended by a notifier implemented into the NI of the NoC. The notifying memories send a new kind of packet, the notification packet that is caught by a listener, a module implemented in the NI of the processor. The notifying memory concept raises new opportunities and new challenges. At the hardware level, an exploration is needed to find out the size of the components: number of possible notifications, size of the listener. Another challenge is to make it configurable or programmable for other programming models. At the software level, the compiler should detect the firing rules to appropriately separate the code onto the notifier and the listener. Finally, new scheduling policies could take advantage of notifications.

7. REFERENCES

[1] A. Firoozshahian et al. A memory system design framework: Creating smart memories. In *ISCA*, 2009.
[2] E. Gamma et al. *Design patterns: elements of reusable object-oriented software*. Addison-Wesley Longman Publishing Co., Inc., 1995.
[3] K. Jerbi et al. Automatic generation of optimized and synthesizable hardware implementation from high-level dataflow programs. *VLSI Des.*, Jan. 2012.
[4] J. Liu et al. Network footprint reduction through data access and computation placement in NoC-based manycores. In *DAC*, 2015.
[5] J.-V. Millo et al. Modeling and analyzing dataflow applications on noc-based many-core architectures. *ACM Trans. Embed. Comput. Syst.*, 14(3), Apr. 2015.
[6] Orcc. The open rvc-cal compiler : A development framework for dataflow programs.
[7] A. Pascal et al. Extended cyclostatic dataflow program compilation and execution for an integrated manycore processor. *Procedia Computer Science*, 2013. International Conference on Computational Science.
[8] A. Singh et al. Mapping on multi/many-core systems: Survey of current and emerging trends. In *DAC*, 2013.
[9] M. Wipliez and M. Raulet. Classification of dataflow actors with satisfiability and abstract interpretation. *IJERTCS*, 2012.
[10] Xiph. Xiph.org video test media.
[11] J. Yoo et al. Active memory processor for network-on-chip-based architecture. *IEEE Transactions on Computers*, 61(5), May 2012.

5.2 Run-Time Remapping of Dataflow Actors on NoC-based Heterogeneous MPSoCs

This research topic has been originally initiated during my work at South Brittany University (UBS) in MOCS research team at Lab-STICC in Lorient, France in the context of the ANR COMPA project. Later, the work on this topic has been continued at the Lebanese International University (LIU) in Lebanon in collaboration with UBS. The proposed algorithm and architecture have been validated along with the obtained results during my visit as visitor professor to UBS in December 2019. This research work has been recently published in the *IEEE Transactions on Parallel and Distributed Systems (TPDS)* [48].

5.2.1 Introduction and Motivations

Multiprocessor system-on-chip (MPSoC) platforms have been emerging as the main solution to cope with processor frequency ceiling and power density issues while still improving performances. Then, networks-on-chip have been adopted to provide the increasing number of processors with the required communication bandwidth as well as with the necessary flexibility. But legacy code for instance, mainly designed for single or few core architectures, does not scale well with manycore architectures and fails to fully benefit from the available parallelism. However, as discussed decades ago [49], dataflow programming can address the limitations of conventional approaches regarding synchronization and shared memory issues. With the rise of massively parallel architectures, we can reconsider the use of dataflow programming as a solution to efficiently exploit the resources of parallel architectures for computing intensive application domains such as video coding, computer vision, machine learning and physics simulation for instance.

A dataflow application can be specified as a graph where nodes, called actors, process data called token(s). The computational models are based on FIFO buffers and respect their formalized read and write rules. Each FIFO holds a set of tokens. A network of actors holds specific features that make it different from a generic task graph. Figure 5.1 presents a simple example on network of actors.

First, an actor is non-preemptive. Once started, an actor ends its execution. Second, the actor can start if and only if there are enough tokens as input, and enough space in the output FIFOs. The FIFOs are considered updated (i.e. tokens consumed and produced) at the end of the execution of the actor, establishing a conservative synchronization scheme, and preventing from any data race.

When the number of actors is larger than the number of processing elements (PEs), then the main design challenge is the mapping of actors on the network of PEs [42]. In the case of static dataflow [50], where the number of tokens produced and consumed by the actors is known, an optimal solution can be computed offline [51]. However, an increasing number of applications cannot be specified with a static graph since the performance improvement of complex applications usually lead to context and data-dependent optimizations. This evolution is, for instance, significant in the domain of video coding.

Dynamic models are then used to express data-dependent behavior of some applications. Dynamic dataflow is a useful MoC for handling streaming data and video processing applications [52]. As the workload of an actor may change according to the input data set, adapting the mapping, while the application runs, is required to optimize the use of the computing and communication resources. Hence, adopting any static or offline dynamic scheduling for mapping tasks will not cope with the computation variations.

This research work introduces an original run-time mapping algorithm based on the Move Based (MB) method targeting a dedicated heterogeneous NoC-based MPSoC architecture to achieve workload balancing and optimized communication traffic. The devised algorithm compromises monitoring the execution of actors while running in real-time on processors. Based on the recorded performance metrics, an estimation of performance gain and cost is determined to move one actor in order to enhance the overall performance.

5.2.2 Context and Related Work

The question of mapping parallel applications on multi or many-core architectures is a very wide problem, with a large number of dimensions, including the programming model, the target architecture (homogeneous or heterogeneous, bus-based or NoC-based, etc.), and the optimization goal (throughput, execution time, energy, etc.) [42].

The mapping problem can be solved based on two main strategies: design-time, and run-time. When solved at design-time, the mapping is called *static* since it's computed offline and does not change while the application runs. This approach allows for exact methods to find an optimal solution [53] [51] [54], but suffers from a lack of flexibility since it cannot capture the dynamic behavior of some applications. Moreover, even in the case of deterministic execution times of actors in a static context, the paper [55] interestingly shows the difference between the optimal mapping obtained from a well-formalized problem and the real execution trace, due to execution variabilities coming from the hardware.

The dynamic workload should be handled using run-time techniques. The run-time mapping strategies can themselves be divided into two categories:

1. on-the-fly mapping
2. hybrid mapping

On-the-fly mapping techniques are application - and platform - agnostic and solve the problem online. Very simple and efficient heuristics should be used to shorten the response time. For NoC-based MPSoCs, various fast heuristics targeting the reduction of communications under constraints have been already proposed [56] [57] [58]. These approaches consider one task per core. Allowing multiple tasks on one core is considered in [59]. Heuristics are fast but can be far from optimal solutions, so hybrid approaches have been introduced. They are based on pre-computed optimal solutions for a set of cases. The job is split into two phases: (1) at design-time, a set of solutions is computed, and (2) one solution is selected at run-time. A wide variety of approaches can then be cited: based on traces in [60], on priority in [61], on scenario in [62], on previously identified design points in [63], or on WCET and scheduling

in [64]. None of these studies demonstrates its efficiency with real video applications running reference sequences. The proposed real-time mapping reconfiguration method in [65] requires to suspend the currently running application and the manager remaps the tasks at run-time according to scenarios previously defined at design-time based on the evaluation of multiple mappings, optimizing for their resource requirements and power consumption. In [66], the authors have proposed a dynamic resource balance algorithm targeting NoC-based many-core homogenous platforms to enhance the system performance by balancing the utilization of on-chip computing resources and communication resources. Most of the available methods focus on determining the suitable mapping of tasks before starting the execution of the application [67, 68, 69, 70, 71].

Finally, a last approach can fall into the family of hybrid mappings, which considers to recompute partially the mapping problem at run-time. This is called *run-time remapping*. Our work follows such an approach for dataflow applications. First, profiling results are used to find at run-time a first mapping. Then, the application is monitored to update the profiling results and a run-time remapping algorithm runs regularly to check if a new mapping would be better than the current one.

Among the innumerable works dealing with task mapping, we consider run-time methods for dataflow tasks, and have identified a limited number of solutions. In [72], the mapping is modeled as a graph partitioning problem, and the problem is solved at run-time by METIS tool, based on profiling information obtained by a first run. Though the migration cost of the actors is not taken into account, the results are promising and could be improved if the mapping does not change completely at each iteration. The approach in [73] allows to successively refine the mapping according to the dynamic behavior of the application, by allowing only one actor to move at a time from one processor to the other. This approach assumes dynamic dataflow application and the target architecture is composed of several heterogeneous cores interconnected by a bus or a NoC. The communication cost is computed based on a rough analytical model of the interconnection network, with the loss of accuracy that comes with it, whereas in our work, we consider profiled values gathered automatically by the system, with a finer grain down to the link.

In [74], the application is specified with KPN (Kahn Process Network) and the target architecture is a shared-memory based MPSoC, with also a model of the communication channel (bus or NoC). The approach proposes to rely on three main steps: the two usual design-time preparation and run-time mapping steps plus a new customization step. The design time step computes a set of candidates and populates a database. The run-time mapping initialization derives from the candidates a new initial mapping for the given workload. Finally, the run-time customization step incorporates a Scenario-based run-time Task Mapping (STM) algorithm that is applied to find new mapping of tasks when the system detects that an objective is unsatisfied.

5.2.3 Contributions and Performed Work

This research work addresses the problem of reconfiguring at run-time and at the application level the mapping of dataflow actors on heterogeneous processors, that share the same instruction set architecture (ISA) while having different coprocessors and different clock

domains. The proposed method, which is called *run-time remapping*, relies on continuous monitoring of exact performance metrics such as the computational time and communication time during real-time execution of the application. Accordingly, a new mapping of the involved actors is determined at run-time targeting the enhancement of the overall performance. This approach is sequentially repeated while the application is running. The application is neither suspended nor modified. The proposed remapping method meets with the dynamic behavior of dataflow applications, where static or offline mapping methods cannot capture the dynamic behavior and thus may not lead to optimal solutions. Also, on-the-fly and hybrid mapping methods suffer from a lack of means to monitor the performance and remap the actors accordingly.
The contributions of this research work are described in the following subsections:

5.2.3.1 Design of NoC architecture for dataflow application

A new heterogeneous MPSoC architecture is proposed. The target architecture contains several different processing elements and shared memories connected with a NoC. The PEs and memory modules are technologically independent of the structure of the NoC. They communicate through the network using a network interface. The devised platform includes heterogeneous PEs. All PEs are implemented be able to execute any of the involved actors. However, some PEs are augmented with hardware accelerators in order to perform special functions more efficiently. In addition, the PEs have been specified randomly to operate on different frequencies. The memory modules include the FIFOs and store the data which is either imported to the system or processed by the PEs.

In order to implement the remapping algorithm, the designed architecture includes a manager and two special types of memories. One memory module is dedicated to store the binary codes of all actor and forwards the binary code of the moved actor to a given PE according to a notification from the manager. Another memory module is in charge of storing the mapping information, which is generated by the manager and indicates which actors are to be executed by each PE, and the monitoring information, which is collected by the PEs during processing a specified number of video frames.

The manager is a PE that executes the following five tasks.

1. map initially the actors on the available PEs
2. parse the feedback collected data from all modules (memories and PEs)
3. apply the run-time remapping algorithm and selects the actor to be moved (if any)
4. notify the corresponding PEs (*looser*, *gainer*, etc.) about the updated mapping
5. manage the transferring of the binary code corresponding to the moved actor from the shared memory into the cache of the gainer processor

5.2.3.2 Implementing the actor scheduler

An actor scheduler is implemented in order to manage the order of execution of actors on PEs, which are considered to run more than one actor as the number of PEs is smaller than the number of actors. In this work, the well-known round-robin scheduling technique has been adopted. The actors are given the attempt to be executed in a circular order without priority. The PE will execute the allocated actor if there are enough input tokens and enough space in the output FIFOs as specified in dataflow applications.

5.2.3.3 Introduce new types of command packets

In this work, new types of command packets are opted in order to manage FIFO accesses, send mapping information, collect monitoring data, and manage the transferring of binary codes. Command packets are initiated by the cores and processed by the NIs of destination nodes. The following list enumerates the devised packet types and describes briefly their corresponding roles.

1. **Notification packet (NP)**: aims to inform the PEs that new information is ready to be requested.
2. **Monitoring/Mapping information reading request packet**: used to request the monitoring or mapping information as a response to the NPs.
3. **Mapping information packet ($\mathbf{M}_p$IP)**: is used by the manager to inform all involved PEs after determining or modifying the actor mapping strategy.
4. **Mapping Confirmation packet (MCP)**: aims to inform the manager that the new mapping information is well received by both the former and the new owner of the actor.
5. **Monitoring information packet ($\mathbf{M}_n$IP)**: holds the feedback information needed by the manager to perform the RR algorithm.
6. **FIFO index packet (FIP)**: holds the writing indexes or reading indexes of FIFOs.
7. **Data reading request packets (DRP)**: holds the reading request of data from PE to memory module
8. **Code transferring packet (CTP)**: holds a command from the manager to transfer the binary code referring to the moved actor from the shared memory to the cache of the new PE.

5.2.3.4 Present new NoC services to implement the observation and adaptation mechanisms

In order to apply the proposed method, the architecture of NoCs is augmented to efficiently provide new services of monitoring performance metrics and remapping the actors, which are not available in conventional networks. In addition, the typical NIs of memory modules are extended to accommodate the services for managing the addressing and arranging the

retrieved output bits into flits. These new functionalities are implemented as additional components in the front-end of the NI corresponding to each memory module type in order to be independent of NoC parameters and to remain compliant with any available memory.

5.2.3.5 Devise new approaches to estimate NoC communication time delay

Communication time delay is a critical factor in HMPSoC platforms using NoCs. The communication time of the moved actor is affected by the location of the new hosting PE in the network. NoC time-delay estimation impacts directly the prediction process of the communication time of the moved actor. Hence, the accuracy level in estimating the delay latency changes the decision on the actor move in the RR algorithm. In this work, two novel methods have been proposed to estimate the communication time delay for transferring one token in the NoC. Both proposed methods make use of the monitoring data, which is collected while processing the video frames in the previous observation window.

The first method is called the *average-path token delay* (APTD) and it is based on finding the average delay for transferring one token depending on the path delays between all nodes of the NoC where a path is considered to be formed from the set of the interconnections between two specific nodes. As a deterministic routing is applied in this work, the packets always use the same path between the source node and the destination node.

The second approach is called the *average-link token delay* (ALTD) and considers the time-delay of the token according to the used physical links connecting the NoC components while transferring the token. A link is defined as the interconnection between two consecutive components of the NoC: router, memory and PE. Each path is segmented into a set of links. Then the average communication time delay per token is determined for each link based on the monitored tokens in the link and the accumulated communication time of these tokens.

5.2.3.6 Optimize the MB algorithm targeting the NoC-based architecture

A novel run-time remapping (RR) algorithm based on the MB method, which allows only one actor to move at a time from one processor to another, is demonstrated targeting NoC-based architecture. The algorithm benefits from the monitoring information captured in PEs and memory modules during run-time to decide the moving of an actor. The devised algorithm is divided into two main stages:

1. In the first stage, all possible candidate actors which their moves would enhance the overall throughput are identified. MB algorithm define the throughput as the inverse of the maximum period over all processors where the period of a PE is the sum of total computation time and total communication time of all actors mapped on this PE. So, the set of candidate actors includes the actors running on the PE with maximum period.

2. In the second stage, a tradeoff between the cost of migration and the predicted improvement of the performance is performed. The actor with maximum total gain is chosen to be moved. The actor selected to be moved should have a maximum total gain. Thus, the manager estimates the total gain achieved for all combinations of mapping the actors which belongs to the candidate list

onto all available PEs. The estimated total gain of a mapping combination is computed by finding the difference between the estimated performance gain and the estimated migration cost of the actor The mapping combination that leads to the maximum estimated total gain is then selected. The engaged processor and actor are specified and so-called the *gainer* processor and *moved-actor* respectively. Note that the estimated NoC communication time delay (found using APTD or ALTD) is utilized to estimate the communication time and the migration time required to compute the total gain.

5.2.3.7 Building an experimental framework

In order to assess the feasibility of our proposed run-time remapping method, a real-time simulator is developed.

- The simulator is described in SystemC TLM model [75]. The devised simulator models the MPSoC platform using NoC concept for interconnecting embedded modules. The platform incorporates heterogeneous processing elements, memory blocks, and the manager. The simulator platform is designed with hierarchical modules that can work concurrently and intercommunicate via ports using simple or complex communication channels. SystemC features are exploited to mimic the accurate functionality of the tailored modules.

- The SystemC model adopted in the simulation platform is cycle accurate at the level of the NoC and the network interfaces. The timing of all corresponding action executions on PE is compensated in the simulation according to the profiling data extracted while running the application on a reference computer. Profiling data provides, for each involved action, the mean value of the number of cycles required to execute it. During SystemC simulations, for each fired action, its corresponding execution time determined in profiling is mapped according to the processor frequency and is used as time delay to compensate the real execution time.

- In this work we target the multimedia application domain. We adopt the well-known MPEG4 part 2 Simple Profile video decoder (MPEG4-SP) which is developed by the Moving Picture Experts Group. This multimedia application is typically used in decompression of encoded video digital data. MPEG4-SP is specified with heterogeneous dataflow MoCs and includes up to 40% of dynamic actors. It is composed of 41 actors and 70 FIFOs [44].

- The functionality of the modeled actors is verified. First, several benchmark video sequences with different formats from [46] have been encoded. The resultant data has been used as input to the simulator. These same encoded videos have been decoded on a desktop computer and the FIFO contents have been traced over the decoding period. The contents stored in the FIFOs in the simulator have been compared to the traced FIFO data. Also, the output data of the simulator have been reconstructed into visual video in order to verify the functionality of the devised simulator.

5.2.3.8 Evaluation of the devised RR algorithm

The encoded video sequences are decoded without applying remapping targeting the same NoC-base architecture and the obtained results are compared to that obtained when the video sequences are decoded adopting the MB remapping algorithm applying ALTD and APTD for estimating the communication time delay while considering an observation window of 10 frames. The comparison is conducted in terms of packet delay, timings and performance.

Applying remapping induces additional control packets and data packets to send the mapping information, monitoring data, notifications and to transfer the binary codes. In order to evaluate the effect of applying the MB remapping algorithm on the traffic in the network, the transported flits and the number of packets that travel through the network during the decoding of the video sequences, are recorded in the case of applying the MB remapping and the case of decoding the video without remapping.

5.2.3.9 Comparison with state-of-the art

In order to determine the relevancy of the devised algorithm, it is compared to the STM method introduced in [74]. In fact, the authors in [74] have presented a hybrid task mapping algorithm that consists of three steps. The two first steps (design-time preparation and run-time mapping initialization) are not related to our work. The third step, so-called run-time customization, incorporates STM algorithm that is applied to find new mapping of tasks when the system detects that an objective is unsatisfied [74]. According to the best of our knowledge, STM is the only similar remapping approach when focusing on the small subset of the existing work around hybrid and run-time (re-)mapping of dataflow applications on NoC-based architectures. To achieve fair comparison, the STM method is modeled and implemented on our devised NoC-based architecture.

In addition, the exact method presented in [53] is implemented for the initial mapping, with two differences: we have used constraint programming instead of ILP, and the objective function is the maximum period as it is our optimization goal. The workload used for the computation time of the actors is based on the profiling of Foreman video.

5.2.4 Findings

Data transport

The recorded number of packets are classified into two main categories. The first category includes the flits which are used basically for dataflow such as data transfer and requesting and setting FIFO indexes. The second category includes the induced flits by applying the remapping algorithm. The comparison shows that the additional flits induced by applying the MB algorithm forms less than 0.02% from total transported flits.

In addition, the impact of transferring the binary code on the traffic is evaluated. The percentage of flits transporting the binary code of migrated actors from the total number of transported flits in the network is computed while decoding several video sequences. The presented percentages illustrate that the impact of actor migration on the traffic is negligible

(varies from 0.0008% to the 0.0348%. This variation depends on number of moves and the sizes of the binary codes of the moved actors.

Timings

It is noticed that applying the MB remapping algorithm affects the time-delay of the packets. The comparison with the case of ordinary decoding illustrates that using MB remapping decreases gradually the total packet time-delay. The traced graphs show that the total packet delay decreases after the conducted moves of actors. This refers to the fact that task remapping contributes in distributing the tasks on PEs that are nearer to the memory modules accommodating the input and output FIFOs.

Performance

The comparison of average FPS achieved when processing multitude video sequencing while adopting different remapping techniques shows that the MB algorithm achieves the maximum average performance enhancements of 26% and 14.11% when adopting ALTD and APTD respectively compared to the achieved throughput of processing the frames without remapping. Whereas, remapping using STM algorithm achieves a maximum average enhancement of 4%. Performance results of all decoded video sequences shows that the MB remapping outperforms STM remapping technique when considering either APTD or ALTD for estimating the NoC communication time delay. Also, the obtained results show that our method has not failed to enhance the performance for all target video sequences. Whereas, in some cases the STM method leads to deterioration in the performance. In fact, the STM method selects critical task to be moved in each observation window without estimating the resulting total performance gain. Moving the task without determining its effects on the whole system performance degrades the overall performance. While in our proposed algorithm, the maximum achieved total gain among all mapping combinations is first determined as explained. Accordingly, a task is specified to be moved if the estimated maximum total gain is positive. It is noticed that in some observation windows no tasks are moved when the MB algorithm is applied. In these cases, the estimation shows that no performance enhancement will be achieved for all mapping combinations.

The results obtained from the "optimal" mapping approach proposed in [53] show that the MB algorithm, starting from a random mapping (without significant initial delay), performs better that the optimal with no remapping for Foreman video sequence in CIF format. Note that the "optimal" mapping corresponds to the best mapping found based on the profiling of Foreman video after a time out of one hour (like the original paper), and the optimality is not proven.

As the optimality is searched for the Foreman profile, we used the optimal mapping as a starting point for the MB algorithm, and the results show that it further improves the throughput. As expected, the optimal mapping for Foreman does not perform good for other video sequences (such as Ice video sequence in 4CIF format and Grandma video sequence in QCIF format).

The so-called optimal method cannot be used for two reasons. First it introduces an unpractical initialization delay without guaranty of optimality. Secondly, a static solution is not appropriate to data-dependent applications since a solution can be good for one data-stream and inefficient for another one and more importantly the efficiency of a mapping varies over time.

5.3 Perspectives

The notifying memory concept raises new opportunities and new challenges. At the hardware level, an exploration is needed to find out the size of the components: number of possible notifications per memory module, size of the listener. Accordingly, the hardware implementations are to be realized in order to determine the exact resources and energy consumption. Another challenge is to make the NI configurable or programmable for other models. At the software level, the compiler should detect the firing rules to appropriately separate the code onto the notifier and the listener. Finally, new scheduling policies could take advantage of notifications.

Concerning the research work about run-time remapping on heterogeneous MPSoC several we are planning to address several topics concerning the hardware architecture and algorithm. At the hardware level, the implementation of the integrated modules in the NIs and estimating their overhead in terms of area and energy will be done. Future work would also address the reduction of NoC power consumption. Another topic to be addressed is the moving of FIFO modules rather than actors. Novel algorithms would be devised targeting the reducing of the communication time and the impact of the migration cost of actors binary codes. Also, a study would be performed to indicate the impact of the size of the observation window used to monitor the metrics on the overall performance. An important work would be done to study the scalability of the devised architecture and its degree of heterogeneity (number and types of memories and processing elements).

In addition, it would be interesting to investigate the impact of using the notifying memories concept on the devised remapping algorithm in terms of performance and power consumption.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2022.3177957, IEEE Transactions on Parallel and Distributed Systems

Run-time remapping algorithm of dataflow actors on NoC-based heterogeneous MPSoCs

Mostafa Rizk, Kevin J. M. Martin, and Jean-Philippe Diguet

***Abstract*—Multiprocessor system-on-chip (MPSoC) platforms have been emerging as the main solution to cope with processor frequency ceiling and power density issues while still improving performances. Then, network-on-chip (NoC) has been adopted to provide the increasing number of processors with the required communication bandwidth as well as with the necessary flexibility. Video processing and streaming applications are adopting dynamic dataflow model of computation as the need for high performance parallel computing is growing. Dataflow applications executed on modern MPSoC-based architectures are becoming increasingly dynamic and more data-dependent. Different tasks execute concurrently with significant modifications in their workloads and resource demanding over time depending on the input data. Hence, adopting any static or offline dynamic scheduling for mapping tasks will not cope with the computation variations. This paper introduces an original run-time mapping algorithm based on the Move Based (MB) method targeting a dedicated heterogeneous NoC-based MPSoC architecture to achieve workload balancing and optimized communication traffic. The performance of the proposed algorithm is verified by conducting cycle-accurate SystemC simulations of the adopted NoC implementing a real MPEG4-SP decoder. The obtained results reveal the effectiveness of our proposed algorithm. For various real-life videos, the proposed algorithm systematically succeeded to enhance significantly the performance.**

***Index Terms*—NoC, Heterogeneous MPSoC, Run-time remapping, Dataflow actor, Move-based algorithm.**

I. Introduction

MULTIPROCESSOR system-on-chip (MPSoC) platforms have been emerging as the main solution to cope with processor frequency ceiling and power density issues while still improving performances. Then, networks-on-chip (NoCs) have been adopted to provide the increasing number of processors with the required communication bandwidth as well as with the necessary flexibility. But legacy code for instance, mainly designed for single or few core architectures, does not scale well with manycore architectures and fails to fully benefit from the available parallelism. However, as discussed decades ago [1], dataflow programming can address the limitations of conventional approaches regarding synchronization and shared memory issues. With the rise of massively parallel architectures, we can reconsider the use of dataflow programming as a solution to efficiently exploit the resources of parallel architectures for computing intensive application domains such as video coding, computer vision, machine learning and physics simulation for instance.

A dataflow application can be specified as a graph where nodes, called actors, process data called token(s). The computational models are based on First-In First-Out (FIFO) buffers and respect their formalized read and write rules. Each FIFO holds a set of tokens. Fig. 1(a) illustrates a network of actors, which exchange tokens through defined FIFO channels [2]. Fig. 1(b) presents an example of a structure of the software FIFO generated with the tool ORCC [3]. A network of actors holds specific features that make it different from a generic task graph. First, an actor is non-preemptive. Once started, an actor ends its execution. Second, the actor can start if and only if there are enough tokens as input, and enough space in the output FIFOs. The FIFOs are considered updated (i.e. tokens consumed and produced) at the end of the execution of the actor, establishing a conservative synchronization scheme, and preventing from any data race.

When the number of actors is larger than the number of processing elements (PEs), then the main design challenge is the mapping of actors on the network of PEs. In the case of static dataflow [4], where the number of tokens produced and consumed by the actors is known, an optimal solution can be computed offline [5]. However, an increasing number of applications cannot be specified with a static graph since the performance improvement of complex applications usually lead to context and data-dependent optimizations. This evolution is, for instance, significant in the domain of video coding. Dynamic models are then used to express data-dependent behavior of some applications [6]. Dynamic dataflow is a useful model of computation (MoC) for handling streaming data and video processing applications.

As the workload of an actor may change according to the input data set, adapting the mapping while the application runs is required to optimize the use of the computing and communication resources. The mapping problem is known as NP-complete. Heuristic methods, for a fast response time, are thus required to address manycore architectures. Run-time adaptation relies on system observation, decisions and configurations. Several previous works have addressed the problem of task mapping at run-time. In [7], the authors have proposed a dynamic resource balance algorithm targeting NoC-based Many-core homogenous platforms to enhance the system performance by balancing the utilization of on-chip computing resources and communication resources. In [8], the authors have introduced a hybrid application mapping

M. Rizk is with CNRS and member of Lab-STICC UMR CNRS 6285, Brest, France also with the School of Engineering, International University of Beirut, Lebanon, and with the Physics and Electronics Department, Lebanese University, Lebanon. e-mail: mostafa.rizk@imt-atlantique.fr

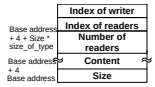
K. J. M. Martin is with Universtité de Bretagne-Sud (UBS) and member of Lab-STICC UMR CNRS 6285, Lorient, France

J-P. Diguet is with CNRS and member of CROSSING, IRL CNRS 2010, Adelaide, Australia

Manuscript received Month DD, YYYY; revised Month DD, YYYY.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2022.3177957, IEEE Transactions on Parallel and Distributed Systems

(a) Network of Actors



(b) Structure of the software FIFO

Fig. 1. Actors and software FIFO Models [2]

that combines design-time analysis with run-time mapping in the context of dynamic thermal and reliability-aware resource management. Most of the available methods focus on determining the suitable mapping of tasks before starting the execution of the application [9], [10], [11], [12], [13], [14]. The mapping of actors is also an active topic for other target platforms like Coarse-Grained Reconfigurable Arrays (CGRA) [15] or Field Programmable Gate Array (FPGA) [16].

This research work addresses the problem of reconfiguring at run-time and at the application level the mapping of dataflow actors on heterogeneous processors. In this work, heterogeneous means that processors share the same instruction set architecture (ISA) while having different coprocessors and different clock domains. The proposed method, which is called *run-time remapping*, relies on continuous monitoring of exact performance metrics such as the computational time and communication time during real-time execution of the application. Accordingly, a new mapping of the involved actors is determined at run-time targeting the enhancement of the overall performance. This approach is sequentially repeated while the application is running. The application is neither suspended nor modified. The proposed remapping method meets with the dynamic behavior of dataflow applications. Static or offline mapping methods cannot capture the dynamic behavior and thus may not lead to optimal solutions. Also, on-the-fly and hybrid mapping methods suffer from a lack of means to monitor the performance and remap the actors accordingly. In order to apply the proposed method, the architecture of NoCs must be augmented to efficiently provide new services of monitoring performance metrics and remapping the actors, which are not available in conventional networks.

Adopting the devised remapping method and NoC-based architecture leads to balancing the workload. The obtained results show that the adoption of the remapping method reduces the standard deviations of the computational times and communication times of involved processors by 38.58% and 69% respectively. Thus, the variation of the use rate of processors is reduced compared to running the application without remapping. In addition, a reduction of 8.6% in the total execution time has been achieved as well as a reduction of 21% in the number of packets' hops is recorded when comparing to the execution without remapping.

In this paper we introduce three contributions:

- First, we optimize for a NoC-based architecture with heterogeneous processors, a new run-time remapping (RR) algorithm based on the Move Based (MB) method [17], which allows only one actor to move at a time from one processor to another. Our solution is then compared with state of the art methods for dataflow architectures.
- Second, we present new NoC services that allow to implement the observation and adaptation mechanisms.
- Finally, we demonstrate our solution with a full implementation of MPEG4-SP, which is available as a reference of a typical dynamic dataflow application. It is also complex enough to exhibit data-dependent execution and communication times. We consider a SystemC packet cycle-accurate NoC simulator to fully decode reference videos and demonstrate the effectiveness of the adaptation mechanism with a real-life dataflow application.

The rest of the paper is organized as follows. Section II presents the related work. Section III illustrates the adopted architecture model. Section IV describes the processing flow. Section V details the conducted experiments and presents the obtained results. Finally, Section VI concludes the paper.

II. Related Work

The question of mapping parallel applications on multi or many-core architectures is a very wide problem, with a large number of dimensions, including the programming model, the target architecture (homogeneous or heterogeneous, bus-based or NoC-based, etc.), and the optimization goal (throughput, execution time, energy, etc.) [18]. The interested reader can refer to the paper gathering different mapping strategies for NoC-based architectures [19]. Following the taxonomy proposed in [18], the mapping problem can be solved based on two main strategies: design-time, and run-time. When solved at design-time, the mapping is called *static* since it's computed offline and does not change while the application runs. This approach allows for exact methods to find an optimal solution [20] [5] [15], but suffers from a lack of flexibility since it cannot capture the dynamic behavior of some applications. Moreover, even in the case of deterministic execution times of actors in a static context, the paper [21] interestingly shows the difference between the optimal mapping obtained from a well-formalized problem and the real execution trace, due to execution variabilities coming from the hardware.

The dynamic workload should be handled using run-time techniques. The run-time mapping strategies can themselves be divided into two categories: on-the-fly mapping, or hybrid mapping. On-the-fly mapping techniques are application- and platform-agnostic and solve the problem online. Very simple and efficient heuristics should be used to shorten the response time. For NoC-based MPSoCs, various fast heuristics targeting the reduction of communications under constraints have been already proposed [22] [23] [24]. These approaches consider one task per core. Allowing multiple tasks on one core is considered in [10]. Heuristics are fast but can be far from optimal solutions, so hybrid approaches have been introduced. They are based on pre-computed optimal solutions for a set of cases. The job is split into two phases: (1) at design-time, a set of solutions is computed, and (2) one solution

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2022.3177957, IEEE Transactions on Parallel and Distributed Systems

is selected at run-time. A wide variety of approaches can then be cited: based on traces in [25], on priority in [26], on scenario in [27], on previously identified design points in [28], or on WCET and scheduling in [29]. None of these studies demonstrates its efficiency with real video applications running reference sequences. The proposed real-time mapping reconfiguration method in [8] requires to suspend the currently running application and the manager remaps the tasks at run-time according to scenarios previously defined at design-time based on the evaluation of multiple mappings, optimizing for their resource requirements and power consumption. Finally, a last approach can fall into the family of hybrid mappings, which considers to recompute partially the mapping problem at run-time. This is called *run-time remapping*. The work presented in this paper follows such an approach for dataflow applications and leverages design-time analysis profiling results to find at run-time a first mapping. The application is then monitored to update the profiling results and a run-time remapping algorithm runs regularly to check if a new mapping would be better than the current one.

Among the innumerable papers dealing with task mapping, we consider run-time methods for dataflow tasks, and have identified a limited number of solutions. In [30], the mapping is modeled as a graph partitioning problem, and the problem is solved at run-time by METIS tool, based on profiling information obtained by a first run. Though the migration cost of the actors is not taken into account, the results are promising and could be improved if the mapping does not change completely at each iteration. The approach in [17] allows to successively refine the mapping according to the dynamic behavior of the application, by allowing only one actor to move at a time from one processor to the other. This approach assumes dynamic dataflow application and the target architecture is composed of several heterogeneous cores interconnected by a bus or a NoC. The communication cost is computed based on a rough analytical model of the interconnection network, with the loss of accuracy that comes with it, whereas in our work, we consider profiled values gathered automatically by the system, with a finer grain down to the link. In [31], the application is specified with KPN (Kahn Process Network) and the target architecture is a shared-memory based MPSoC, with also a model of the communication channel (bus or NoC). The approach proposes to rely on three main steps: the two usual design-time preparation and run-time mapping steps plus a new customization step. The design time step computes a set of candidates and populates a database. The run-time mapping initialization derives from the candidates a new initial mapping for the given workload. Finally, the run-time customization step incorporates a Scenario-based run-time Task Mapping (STM) algorithm that is applied to find new mapping of tasks when the system detects that an objective is unsatisfied. It first detects the so-called *critical task* and then identifies why it misses its objectives: either poor locality or load imbalance. In case of poor locality, an algorithm that considers the communication between tasks is used to find a new mapping. In case of load imbalance, a load balancing strategy based on computational demands of the tasks is used. This step produces a new mapping that may move several tasks, which leads to a (re-)mapping overhead.

When focusing on the small subset of the existing work around hybrid and run-time (re-)mapping of dataflow applications on NoC-based architectures, we consider the work presented in [31] for comparison.

III. Architecture Model

The target architecture is a heterogeneous Multi-Processor System on Chip (HMPSoC) containing several different PEs and shared memories connected with a Network-on-Chip (NoC). Fig. 2 presents the structure of the adopted NoC-based architecture. Our method is scalable and without loss of generality we consider a specific model of architecture which is required for a data-accurate functional simulation with a packet-level time accuracy. The target architecture is a 4×4 mesh-based NoC with 32-bit links that interconnects 28 intellectual property (IP) cores including 15 memory modules, 12 PEs and a processing element that acts as a manager (MGR). The PEs and memory modules are technologically independent of the structure of the NoC. They communicate through the network using a network interface (NI). We consider a simple NoC model that employs the wormhole packet switching mode, the deterministic XY routing algorithm, and a flow control policy without virtual channels. The implemented routers have one buffer of 3 flits per input port and use distributed arbitration logic (one arbiter per port). The back-end part of the NI is typical and includes a packet maker/un-maker, which are used to assemble and disassemble the packets, and a priority manager to synchronize packet transmission and reception.

In this work, it is assumed that $PE1$ imports the incoming streamed data from an Input buffer and $PE12$ outputs the processed data. Fig. 2 illustrates the buffers in order to communicate with external systems. Each PE has its local memory. It is assumed that there are no restrictions to map any MPEG4-SP application actor to any PE. The used PEs can all work in parallel according to dataflow firing rules. However, some PEs are enhanced by hardware accelerators dedicated to certain functionalities in order to perform them more efficiently. The shared memories are distributed in memory blocks which have a unique NI. From an NoC perspective, the novelty is the introduction of new command packets used as instructions to manage FIFO accesses, broadcast mapping information, collect monitoring data, and the transfer of binary codes. In order to cope with the command packet and associated notification packet concepts, the NIs implement some additional logic modules. The command packets were already proposed in [32] but only produced by the manager and for a specific application.

A. Manager

The manager is a PE dedicated to the following five tasks: (1) map initially the actors on the available PEs, (2) parse the feedback collected data from all modules (memories and PEs), (3) apply the run-time remapping algorithm and selects the actor to be moved (if any), (4) notify the corresponding PEs (*looser*, *gainer*, etc.) about the updated mapping and

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2022.3177957, IEEE Transactions on Parallel and Distributed Systems

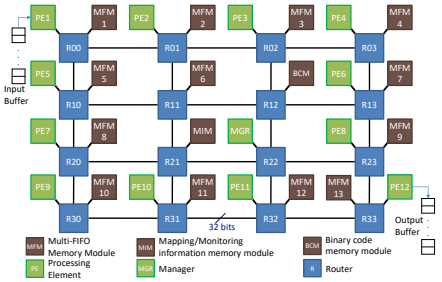


Fig. 2. The structure of the used NoC-based architecture

TABLE I
HARDWARE ACCELERATORS USED IN THE SIMULATION PLATFORM

PE ID	Accelerated Function	Acceleration Ratio
PE3 & PE6	IDCT	1/0.3
PE4	IQ + IAP	1/0.75
PE10	Add	1/0.57
PE11	Interpolation	1/0.4

(5) manage the transferring of the binary code corresponding to the moved actor from the shared memory into the cache of the gainer processor.

B. Processing Elements

The target platform includes twelve PEs. All PEs are supposed to be able to execute any of the forty-one actors involved in the MPEG4-SP application. As the number of PEs is smaller than the number of actors, each PE is considered to run more than one actor. Hence, an actor scheduler is required to manage the order of execution of actors. Mainly, in dataflow applications, all schedulers suffer from inefficient polling which leads to useless memory accesses when a scheduling attempt fails. In this work, the well-known round-robin scheduling technique has been adopted in all PEs. The actors are given the attempt to be executed in a circular order without priority. The PE will execute the allocated actor if there are enough input tokens and enough space in the output FIFOs as specified in dataflow applications.

Furthermore, some PEs are augmented with hardware accelerators in order to perform special functions more efficiently. In this work, we adopt one of the hardware accelerator specification described in previous similar work [17]. Table I shows the list of accelerators adopted in the simulation platform. In addition, the PEs have been specified randomly to operate on different frequencies. Table II shows the randomly chosen operating frequency of all PEs in terms of the NoC operating frequency f.

C. Memory Modules

The tailored platform integrates three types of memory modules. Each module includes a memory block that returns the data allocated at its specified address. Since the PEs and the manager do not recognize the local mapping of stored data

TABLE II
PROCESSING ELEMENT OPERATING FREQUENCY

PE ID	Operating Frequency
PE1, PE12, MGR	f
PE2, PE6, PE10	$2f$
PE3, PE7, PE11	$3f$
PE4, PE8	$4f$
PE5, PE9	$5f$

in each memory module and in order to remain compliant with any available memory, the typical NI is extended to accommodate the services for managing the addressing and arranging the retrieved output bits into flits. These new functionalities are implemented as additional components in the front-end of the NI corresponding to each memory module type in order to be independent of NoC parameters. In the following the functionalities of each memory type is described.

1) Binary code memory module (BCM): It contains the binary codes of all actors. The manager sends a specific packet request to BCM to forward the binary code of the moved actor to a given PE according to the decision taken after executing the RR algorithm. A simple module, so-called memory address mapper (MAM), is integrated into the NI of the BCM in order to find the correct memory address. For a specific actor, MAM determines the starting address of the binary code and its corresponding size based on the actor's ID and by the means of simple look-up-tables that include the starting addresses and the size of the binary codes of all actors. Furthermore, MAM manages the extraction of data from the memory and delivers it to the packet maker unit.

2) Mapping/Monitoring information memory module (MIM): This memory module accommodates twelve memory blocks. Each block is dedicated to a specific PE and is supposed to store two types of data. The first type is the mapping information, which is generated by the manager and indicates which actors are to be executed by each PE in addition to their supplementary information about input and output FIFOs and the reading orders for each input FIFO (III-D1c). The second type is the monitoring information (III-D1e), which is collected by the PEs during processing a specified number of video frames. Storing the monitoring information overwrites the mapping information, which is not needed by the PEs anymore.

When a packet holding either mapping or monitoring information is received, the MIM module first identifies the corresponding PE. Accordingly, it dissembles the packet and stores the data found in the packet payload into the memory block assigned to the identified PE.

Moreover, the MIM informs the manager about the availability of new monitoring information and the corresponding PE about the availability of new mapping information. To do so, the MIM sends notification packets (III-D1a) as per the concept of notifying memory concept demonstrated in [2]. In addition, the MIM responds to reading requests (III-D1b) sent from the manager to acquire the stored monitoring information from the PEs or to get the new mapping information.

3) Multi-FIFO memory module (MFM): This type of memory module is dedicated to store the data which is either

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2022.3177957, IEEE Transactions on Parallel and Distributed Systems

TABLE III
ADDRESSES DETERMINED BY THE MFM-NI CONTROLLER

Packet Type	Starting Address	Offset
Request/Set writing index	$FIFOsize$	0
Request/Set reading index	$FIFOsize+1$	reading order
Reading Request packet	Reading address	incremented till
Data packet	Writing address	reaching data size

imported to the system or processed by the PEs. Each MFM accommodates a specific number of FIFOs. Only one FIFO is used at once. The inputs of all FIFOs are connected to the module's inputs using demultiplexers whereas the FIFOs' output data ports are multiplexed. This signal is buffered from the value of FIFO address which is specified in the payload of the arriving packets (see Fig. 4). The multiplexer and demultiplexers are added to the adapter in the NI.

Moreover, the MFMs receive the following types of packets: (1) FIFO Index packet (III-D1f) that aims either to retrieve or to set the writing and reading indexes, (2) Data Reading Request packet (III-D1g) that demands to read data from a specified FIFO and (3) Data packet (III-D2a) that is used to write data in a specified FIFO.

A simple circuit is integrated into the adapter of the NI in all MFM modules in order to manage the memory addressing for all listed-above packet types. It is composed of a simple controller and two 4-to-1 multiplexers and an adder in order to generate the appropriate address values to be given to the MFM FIFOs. After disassembling the arriving packet, the packet un-maker delivers the packet type and the data size to the controller. Accordingly, the controller generates the control signals to configure the two multiplexers, which are dedicated to select the values of starting address and the offset as listed in Table III. These two values are then added to compute the memory address. In addition, the controller determines the number and type of the required memory accesses. It incorporates a simple comparator and an address counter which is incremented for each required access.

D. Packets' structure

The developed NoC architecture considers two categories of packets: (1) command packets and (2) data packets.

1) Command packets: Command packets are initiated by the cores and processed by the NIs of destination nodes. Several command packets, described hereafter, have been opted in order to manage FIFO accesses, send mapping information, collect monitoring data, and manage the transferring of binary codes.

a) Notification packets (NP): The NPs aim to inform the PEs that new information is ready to be requested. This technique is inherited from the notifying memories (NM) concept presented in [2]. When receiving a NP, the PE will send a reading request to retrieve the available data at the corresponding notifying memory. In this work, notification packets are used either to inform an ordinary PE that new mapping information is available or to notify the manager that updated monitoring information has been generated and stored. The NP has empty payload and aims to trigger the manager and PEs to request data when it is ready rather than frequent inefficient polling.

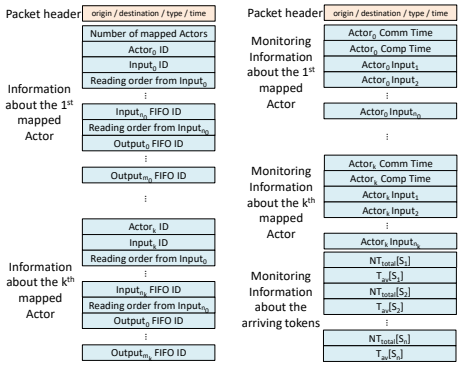


Fig. 3. Packets structure for mapping (left) and monitoring (right) information

b) Monitoring/Mapping information reading request packets (MRP): This type of packet is used to request the information stored in the MIM module as a response to the NP. It is either generated by the manager to acquire the new monitoring information sent from a definite PE or by one of the PEs to get the new mapping information provided by manager. For both information types, monitoring or mapping information, the request packet does not include any payload.

c) Mapping information packets (M_pIP): The manager uses a M_pIP to inform all involved PEs after determining or modifying the actor mapping strategy. Its payload includes the following: (1) the number of actors which are mapped to the PE, (2) the IDs of the mapped actors, (3) the IDs of the input and output FIFOs, and (4) the actor reading order in each input FIFO. Fig. 3 illustrates the structure of the packet holding the mapping information.

d) Mapping Confirmation packets (MCP): A MCP aims to inform the manager that the new mapping information is well received by both the former and the new owner of the actor. The MCP payload is also empty.

e) Monitoring information packets (M_nIP): This type of packet holds the feedback information needed by the manager to perform the RR algorithm. Fig. 3 presents the structure of the monitoring information packet.

f) FIFO index packets (FIP): The FIPs are designed to hold the writing indexes or reading indexes of FIFOs. As mentioned before, DF applications rely on a large number of requests to memories for firing rule checking. So, these indexes are used to determine either the number of available tokens corresponding to each reader actor or the free space in a FIFO. If data is required to be read from input FIFOs, the firing rule is satisfied by checking if the number of available tokens in all input FIFOs is equal or greater than the required number during computation. Whereas, if data has to be written to output FIFOs, the firing rule is satisfied by checking if all output FIFOs have sufficient empty room to accommodate the produced tokens. Hence, before processing an action, a PE

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2022.3177957, IEEE Transactions on Parallel and Distributed Systems

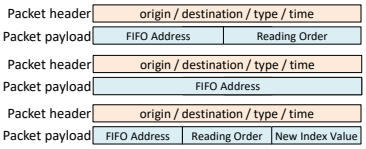


Fig. 4. Packet structures for holding reading (top) / writing (middle) index requests and set index (bottom)

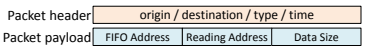


Fig. 5. The structure of the packets holding the reading requests

has to request the reading and/or writing indexes of input and output FIFOs. When the PE receives the value of demanded reading/writing index, it will check the satisfaction of the firing rule. After reading/writing data from/to a FIFO, the reading/writing index has to be incremented by the size of the transferred data. The PE, which consumes/produces data, has to set the new reading/writing index in the targeted FIFO after reading/writing operation is performed. Accordingly, four types of packets are utilized: (1) Request read index, (2) Request write index, (3) Setting read/write index, and (4) Holding read/write index.

As the FIFO may have several reading indexes corresponding to different reader actors, the PE has to determine the reading order of the actor and sends it in the payload of the packet. However, a FIFO has only one writer actor; hence, to attain the value of its writer index the PE has to send the FIFO address in the destination memory module. In both packets, the packet type, given in the packet's header, is used by the NI at the destination memory module to decode the request type. Fig. 4 depicts the structure of the FIP packets holding the requests of a reading index and writing index.

On the other side, whenever a memory module receives a request of reading/writing index it will retrieve its value from the specified FIFO and sends it back to the PE. The NI in the memory module will assemble a 1-flit payload packet as shown in Fig. 4.

In order to set the reading/writing index after finalizing the data transfer operations from/to a FIFO, the PE sends a control packet that notifies the FIFO about its new reading/writing index. It includes one flit that contains the FIFO address in the destination memory module, the reading order of the actor, and the new value of the reading index. Since the FIFOs have only one writer actor, the writing packet payload simply includes the address of the targeted FIFO in the destination memory module and the new value of the writing index.

g) Data reading request packets (DRP): Fig. 5 presents the packet holding the reading request of data from PE to memory module. Its payload consists of one flit that includes the address of the FIFO in the destination memory module, the starting address of reading, and the size of required data.

h) Code transferring packets (CTP): Actor binary codes are stored in a shared memory. When updating the actor mapping, the binary code referring to the moved actor should

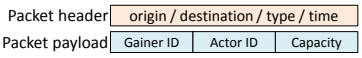


Fig. 6. Manager command requesting the transfer of the moved actor code

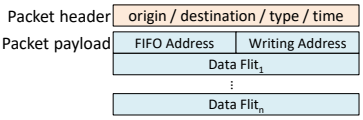


Fig. 7. The structure of packets carrying processed data

be transferred from the shared memory to the cache of the new PE. The manager sends a command of transferring the binary code in the form of a reading request packet. The sent request includes the actor ID, the address of the new PE, and the size of transferred data per packet (see Fig. 6).

2) Data packets: The second category of packets refers to the ordinary flow of data between PEs and memory modules. These packets, described hereafter, carry data that is either processed in a PE and written in a memory module or sent from a memory module as a response to a PE reading request.

a) Dataflow packets (DFP): The DFPs encompass all packets transferred between PEs and the FIFOs distributed in the memory modules. They carry data that is either processed in a PE and will be stored in a FIFO or sent from a FIFO as a response to a PE reading request. Fig. 7 presents the structure of packets carrying processed data in their payloads.

b) Binary code packets (BCP): The BCPs aim to transfer the binary code from the shared memory to the cache memory of the new PE. Note that the binary code is divided into sections of reasonable sizes which are transferred consequently. The size of the transferred data (payload capacity) is specified by the manager according to the monitored traffic in the network and based on the required cache lines to be filled before launching the actor on the new PE. For example, the packet including in its payload 64 flits of 32-bitwidth transfers 256 bytes which form 4 lines of L1 cache.

IV. Processing Flow

A. Initial mapping

Initially, the actors are mapped randomly to the PEs, or can be mapped using the exact method presented in [20]. FIFOs are mapped randomly and are approximately equally distributed on all memory blocks. The manager informs by means of packets all involved PEs. For each PE in charge of executing actors, the manager generates and sends its corresponding mapping information in a separate packet (M_pIP). Packets holding the mapping information are stored in a predefined location in MIM. Then, the involved PEs are notified to retrieve their mapping information from the shared memory using notifying packets. At this stage, the manager waits the PEs, which are incorporated in processing a specific number of video frames N_F to send their monitoring information. Note that N_F is set originally to a default value and may be changed dynamically by the manager.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2022.3177957, IEEE Transactions on Parallel and Distributed Systems

TABLE IV
PARAMETERS AND VARIABLES USED FOR THE MAPPING ALGORITHM

Parameter	Definition
DPN application graph (DPNapp)	
$\|\mathbb{A}\|$	Number (Nb) of actors
$\|\mathbb{F}\|$	Nb of FIFO channels
$\|\mathbb{K}\|$	Nb of data packets
$\|\mathbb{I}_c\|$	Nb of input ports of actor A_c
Architecture graph (arch)	
$\|\mathbb{P}\|$	Nb of processing elements
$\|\mathbb{M}\|$	Nb of memory modules
Profiling data (profile)	
R_i	Mean number of firings of actor i
W^i	Total computation cost of actor i
Cs^i	Instruction code size of actor i

Before receiving the notification packet about initial mapping of actors, all PEs are in idle state. Once it receives the notification packet, the PE sends a request to retrieve the mapping information which includes IDs of actors to be executed, IDs of input and output FIFOs for each actor, and the reading order of each input FIFO. The mapped actors are scheduled according to the order sent from the manager and the PE begins to execute them in round-robin manner.

B. Monitoring actor execution

The execution of actors continues until receiving a new notification packet about changing the mapping information. All involved PEs monitor their running actors during the processing of N_F video frames, which determine the observation window. Precisely, each PE node accumulates for every mapped actor A_c its communication time $T_{cm}[A_c]$, computation time $T_{cp}[A_c]$, and total number of tokens received to each input port $NT_{total}[A_{c_{[I_j]}}]$ where $c \in \{1, ..., |\mathbb{A}|\}$ and $j \in \{1, ..., |\mathbb{I}_c|\}$. In addition, the adapter, which is embedded in the NI of each node n (processor or memory), extracts from each received packet carrying processed data, the following information for each source S_i: (1) the total number of transferred tokens from S_i to n: $NT^n_{total}[S_i, n]$ and (2) the average time delay consumed per token to reach the node n from source S_i: $T_{av}[S_i, n]$.Table IV gathers the variables and parameters used to formalize our mapping approach.

The total number of transferred tokens is simply determined. First, input packets are classified according to their sources S_i. Then, their corresponding sizes $size_{P_k}[S_i]$, which reflect the number of data-flits, are accumulated.

$$NT^n_{total}[S_i, n] = \sum_{k=1}^{\mathbb{K}} size_{P_k}[S_i, n] \quad (1)$$

where $i \in \{1, ..., |\mathbb{P}| + |\mathbb{M}|\}$.

The average time delay per token per each source $T_{av}[S_i, n]$ is calculated by dividing the time delay of each token transferred from S_i by $NT_{total}[S_i, n]$.

$$T_{av}[S_i, n] = \frac{\sum_{k=1}^{\mathbb{K}} size_{P_k}[S_i, n] \times D_{P_k}[S_i, n]}{NT^n_{total}[S_i, n]} \quad (2)$$

where $k \in \{1, ..., |\mathbb{K}|\}$.

$D_{P_k}[S_i]$ is determined by embedding, at the source node, for each packet P_k its sending time-stamp $T_s[P_k]$ in its header then subtracting it from the reception time $T_r[P_k]$ at the destination node. All tokens in a packet are considered to have the same delay.

$$D[P_k] = T_r[P_k] - T_s[P_k] \quad (3)$$

C. Collecting monitoring information

When the number of the processed frames meets the observed window, each PE node generates its own monitoring information packet. The packet is then sent to the MIM module (presented in III-C). Directly, the accumulated values are reset with the beginning of the new observation window. Then, the PE continues executing the previously mapped actors according to the adopted circular order. This guarantees that the remapping does not impose any additional overhead in terms of latency. The MIM module notifies in its turn the manager when new monitoring data is available corresponding to a specific processor throughout a notification packet (III-D1a). Whenever a new notification packet is received by the manager, the latter directly requests to retrieve the new available monitoring data. Also, the manager requests using command packets from all memory modules to send their monitoring information. Note that memory modules respond to the manager and send the requested data directly without any notification process since the adopted MoC allows the direct communication between a memory and a processor. All received monitoring packets are disassembled and their contents are parsed and saved in the manager local registers.

When the feedback data is collected from all modules incorporated in processing the video frames, the manager applies the run-time remapping algorithm. At this stage, the manager owns locally the following data: (1) the communication time of each actor: $T_{cm}[A_c]$, (2) the computation time of each actor: $T_{cp}[A_c]$, (3) the number of input tokens corresponding to every input port of all actors: $NT^a_{total}[A_{c_{[I_j]}}]$, (4) the number of incoming tokens to each processor module from each memory module m: $NT^n_{total}[S_m, p]$, (5) the average communication delay of received tokens to each processor p from each memory module m: $T_{av}[S_m, p]$, (6) the number of incoming tokens to each memory module m from each processor module p: $NT^n_{total}[S_p, m]$, and (7) the average communication delay of received tokens to each memory module m from each processor module p: $T_{av}[S_p, m]$ where $c \in \{1, ..., |\mathbb{A}|\}$, $j \in \{1, ..., |\mathbb{I}_c|\}$, $m \in \{1, ..., |\mathbb{M}|\}$ and $p \in \{1, ..., |\mathbb{P}|\}$.

D. Estimating NoC communication time delay

Communication time delay is a critical factor in HMPSoC platforms using NoCs. The communication time of the moved actor is affected by the location of the new hosting PE in the network. NoC time-delay estimation impacts directly the prediction process of the communication time of the moved actor. Hence, the accuracy level in estimating the delay latency changes the decision on the actor move in the RR algorithm. In this work, two novel methods have been proposed to estimate the communication time delay for transferring one token in the

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2022.3177957, IEEE Transactions on Parallel and Distributed Systems

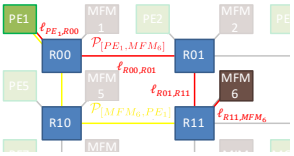


Fig. 8. Example on path declaration in the NoC

NoC. The first method is called the average-path token delay and it is based on finding the average delay for transferring one token depending on the path delays between all nodes of the NoC. The second is called the average-link token delay and considers the time-delay of the token according to the used physical links connecting the NoC components while transferring the token. Both proposed methods make use of the monitoring data, which is collected while processing N_F video frames in the previous observation window. The techniques used in estimating the NoC communication time-delay are described in the following subsections.

1) Average-path token delay (APTD): In this approach, a path is considered to be formed from the set of the interconnections between two specific nodes. As an example, Fig. 8 illustrates in red the path $\mathcal{P}_{[PE_1,MFM_6]}$ between processing element PE_1 to memory module MFM_6. As a deterministic routing is applied in this work, the packets always use the same path between the source node and the destination node. Since the adopted MoC forbids the transfer of packets in between memory modules and in between PEs, the active paths are those connecting either memory modules to PEs or PEs to memory modules. Note that the packets transferred from a processing element p to a memory module m do not follow the same path used in transferring packets from the memory module m to the processing element p. Fig. 8 illustrates in red the followed path to transfer packets from PE_1 to MFM_6 and in yellow the followed path to transfer packets from MFM_6 to PE_1. In APTD, the manager calculates the average path delay per token T_{av} in several steps as shown in Algo. 1. T_{av} refers to the average time delay required to transfer one token from the source node to the destination node, regardless of the path between the source and destination nodes. As an example, the average time delay of all tokens transferred through either the path $\mathcal{P}_{[PE_1,MFM_6]}$ or the path $\mathcal{P}_{[MFM_6,PE_1]}$ (Fig. 8) is considered equal regardless of the number of links constituting each path and the corresponding traffic in each link and the switch conflicts in the connecting routers. T_{av} is computed by dividing the sum of the communication-time delays D_{total} by the total number of transferred tokens in the network NT_{total}:

$$T_{av} = \frac{D_{total}}{NT_{total}} \tag{4}$$

The manager benefits from the collected monitoring data. It makes use of the number of input tokens $NT^n_{total}[S_i, n]$ transferred to each destination node n from each source node S_i to determine the total number of all transferred tokens in the network (NT_{total}) as presented in (5):

$$NT_{total} = \sum_{n=1}^{|\mathbb{P}|+|\mathbb{M}|} \sum_{i=1}^{|\mathbb{P}|+|\mathbb{M}|} NT^n_{total}[S_i, n] \tag{5}$$

Also, the communication-time delays for all tokens transferred in the network are accumulated. The sum of the communication delays D_{total} is determined according to (6):

$$D_{total} = \sum_{n=1}^{|\mathbb{P}|+|\mathbb{M}|} \sum_{i=1}^{|\mathbb{P}|+|\mathbb{M}|} T_{av}[S_i, n] \times NT^n_{total}[S_i, n] \tag{6}$$

where $T_{av}[S_i, n]$ is the collected average time delay required to transfer one token from the source node S_i to the destination node n.

2) Average-link token delay (ALTD): A link is defined as the interconnection between two consecutive components of the NoC: Router, Memory and PE. As an example, Fig. 8 shows the links constituting the path $\mathcal{P}_{[PE_1,MFM_6]}$. In this approach, the average communication time delay per token is determined for each link as shown in Algo. 2. The total communication-time delay in a path $\mathcal{P}_{[S_i,n]}$ connecting the source node S_i and the destination node n is determined from the monitored data as shown in (7):

$$D^{\mathcal{P}}_{total}[S_i, n] = T_{av}[S_i, n] \times NT^n_{total}[S_i, n] \tag{7}$$

Each path is segmented into a set of links $\mathbb{L}_{\mathcal{P}_{[S_i,n]}}$. The average communication time delay per link $D^{\mathcal{L}}_{av}[S_i, n]$ in the path $\mathcal{P}_{[S_i,n]}$ is determined as follows:

$$D^{\mathcal{L}}_{av}[S_i, n] = \frac{D^{\mathcal{P}}_{total}[S_i, n]}{NL[S_i, n]} \tag{8}$$

where $NL[S_i, n]$ is the number of links constructing the path $\mathcal{P}_{[S_i,n]}$. Here, the links constructing a path are assumed to have similar contribution in the total communication time delay monitored in the path. As a link l is shared among different paths, the total link communication-time delay $D_{total}[l]$ is the sum of all average communication-time delay per link computed in all paths in which link l constitutes one of their interconnections:

$$D_{total}[l] = \sum_{n=1}^{|\mathbb{P}|+|\mathbb{M}|} \sum_{i=1}^{|\mathbb{P}|+|\mathbb{M}|} D^{\mathcal{L}}_{av}[S_i, n] \ni l \in \mathbb{L}_{\mathcal{P}_{[S_i,n]}} \tag{9}$$

On the other hand, the tokens passing through a path are definitely passing through all links constructing the path. Hence, the total number of tokens $NT_{total}[l]$ passing through a link l is the sum of all tokens passing through all paths, which link l constitutes one of their interconnections:

$$NT_{total}[l] = \sum_{n=1}^{|\mathbb{P}|+|\mathbb{M}|} \sum_{i=1}^{|\mathbb{P}|+|\mathbb{M}|} NT_{total}[S_i, n] \ni l \in \mathbb{L}_{\mathcal{P}_{[S_i,n]}} \tag{10}$$

The average communication-time delay per token $T_{av}[l]$ for each link l is determined by dividing the accumulated communication-time delay $D_{total}[l]$ by the number of tokens

Algorithm 1 Average-path token delay (APTD)

Step 1: *Find the sum of the communication delays* D_{total}
Step 2: *Find the total number of all tokens* NT_{total}
Step 3: *Calculate the average time delay per token* T_{av}

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2022.3177957, IEEE Transactions on Parallel and Distributed Systems

Algorithm 2 Average-link token delay (ALTD)

Step 1:
for each path $\mathcal{P}_{[S_i,n]}$ **do**
a- *Find the total communication-time delay* $D^{\mathcal{P}}_{total}[S_i,n]$
b- *Calculate average communication time delay per link* $D^{\mathcal{L}}_{av}[S_i,n]$
end for
Step 2:
for each link l **do**
a- *Find the total link communication-time delay* $D_{total}[l]$
b- *Find the total number of tokens* $NT_{total}[l]$
b- *Calculate the average communication time delay per token* $T_{av}[l]$
end for

$NT_{total}[l]$ passing through this link.

$$T_{av}[l] = \frac{D_{total}[l]}{NT_{total}[l]} \tag{11}$$

E. Applying RR algorithm

For each observation window (N_F frames), the manager executes at run-time the RR algorithm, which is divided into two main steps. The first step is dedicated to find all possible candidate actors which their moves would enhance the overall throughput. The second step sets a tradeoff between the cost of migration and the predicted improvement of the performance.

1) Specify the possible candidate actors: In this work, the definitions of the terms period of each processor p ($Period_p$), maximum period ($Period_{max}$) and throughput (Th) have been adopted as introduced in [17]. $Period_p$ is the sum of total computation time $compT_p$ and total communication time $commT_p$ recorded during N_F video frames:

$$Period_p = compT_p + commT_p \qquad \forall p \in \mathbb{P} \tag{12}$$

where $compT_p$ and $commT_p$ of processor p are the sums of the computation times and of the communication times respectively of all actors which are mapped on this processor:

$$compT_p = \sum_{k:\mathbb{P}[k]=p} T_{cp}[A_k] \qquad \forall p \in \mathbb{P} \tag{13}$$

$$commT_p = \sum_{k:\mathbb{P}[k]=p} T_{cm}[A_k] \qquad \forall p \in \mathbb{P} \tag{14}$$

The throughput is defined as the inverse of the maximum period over all processors.

Hence, the first task is to find the PE with the maximum period. The manager computes the periods of all PEs during the current observation window of N_F video frames. Later, a simple comparison between all obtained period values is performed in order to specify the processor with the maximum period. The processor with the maximum period ($Period_{max}$) is nominated as *looser* processor. The algorithm used to determine the *looser* processor is outlined in Algo. 3. The set of candidate actors to be moved $\mathbb{C}$ includes the actors that have been previously executed by the *looser* processor. Fig. 9 demonstrates an example of $Period_p$ and $Period_{max}$. The figure shows three PEs (PE_1, PE_2 and PE_3) that run six actors (A_1, A_2, A_3, A_4, A_5 and A_6). In this example, PE_1 has the largest period, thus it is selected as the *looser* processor.

Algorithm 3 Finding processor with maximum period

$Period_{max} \leftarrow 0$
$looser \leftarrow \phi$
for $p \in \mathbb{P}$ **do**
if $Period_{max} < Period_p$ **then**
$Period_{max} \leftarrow Period_p$
$looser \leftarrow p$
end if
end for

2) Decision of the actor move: The actor selected to be moved should have a maximum total gain. According to the collected monitoring values, the manager estimates the total gain achieved for all combinations of mapping the actors which belongs to the candidate list $\mathbb{C}$ onto all available PEs. The estimated total gain $Gain^e_{total}[C_{A_c,p}]$ of a mapping combination $C_{A_c,p}$, which corresponds to moving A_c to p, is computed by finding the difference between the estimated performance gain $Gain^e_{per}[C_{A_c,p}]$ and the estimated migration cost of the actor $Cost^e_{mig}[C_{A_c,p}]$. The mapping combination that leads to the maximum estimated total gain is then selected. The engaged processor and actor are specified and so-called the *gainer* processor and *moved-actor* respectively.

a) Estimated performance gain: For each actor A_c in the candidate list $\mathbb{C}$, the manager considers it is moved virtually to all PEs except the *looser* processor. For each virtual-move combination, the manager estimates the achieved period of each processing element $Period^e_p[C_{A_c,p}]$. The new period of processor p is estimated by adding to the processor period $Period_p$ the estimated communication time $T^e_{cm}[C_{A_c,p}]$ and the estimated computation time $T^e_{cp}[C_{A_c,p}]$ of the moved actor A_c as shown in the following expression:

$$Period^e_p[C_{A_c,p}] = Period_p + T^e_{cp}[C_{A_c,p}] + T^e_{cm}[C_{A_c,p}] \tag{15}$$

Note that the tokens, which are consumed by a certain reader actor running on a processing element PE_R, are imported from a FIFO f. These tokens are previously generated by another actor running on another processing element PE_W. The generated tokens are first stored in a FIFO f and then transferred once requested to the processing element PE_R where the reader actor is executed. Hence, the tokens pass

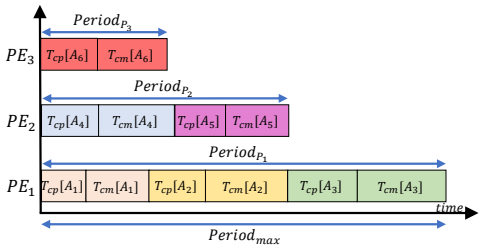


Fig. 9. An example of $Period_p$ and $Period_{max}$

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2022.3177957, IEEE Transactions on Parallel and Distributed Systems

through two paths. The first path $\mathcal{P}_{[PE_W,MFM_f]}$ connects the processing element PE_W, which executes the writer actor, and the memory module that accommodates the FIFO f. On the other hand, the second path $\mathcal{P}_{[MFM_f,PE_R]}$ connects the memory module that accommodates the FIFO f and the processing element PE_R which executes the reader actor. The communication-time delays in both paths are considered when estimating the communication time of the moved actor.

When adopting APTD method for determining the communication delay in the NoC, the estimated communication-time delay per input j for each actor A_c is equal to the total number of input tokens $NT_{total}S[A_{c_{[I_j]}}]$ transferred to the actor at this input multiplied by the double of the calculated average path communication-time delay per token T_{av} (4). The average path delay per token is doubled to compensate the time delay of the two paths $\mathcal{P}_{[PE_W,MFM_f]}$ and $\mathcal{P}_{[MFM_f,PE_R]}$. The total estimated communication time is the sum of all estimated communication-time delays of all inputs:

$$T^e_{cm}[C_{A_c,p}] = \sum_{j=1}^{|\mathbb{I}_c|} 2 \times T_{av} \times NT_{total}[A_{c_{[I_j]}}] \quad (16)$$

Note that the adopted model of computation forbids the transfer of tokens in between actors (running on PEs) directly without passing through a FIFO (allocated in a memory module MFM_f). Hence, tokens produced by the writer actor (running on PE_W) will pass through two paths ($\mathcal{P}_{[PE_W,MFM_f]}$ and $\mathcal{P}_{[MFM_f,PE_R]}$) before arriving to the reader actor (running on PE_R). The exact number of tokens passes through both paths while considering same average path delay per token T_{av}. So, the average path delay per token is doubled in (16).

When adopting ALTD method, the estimated communication-time per input j is equal to the total number of input tokens $NT_{total}[A_{c_{[I_j]}}]$ transferred to the actor A_c through this input multiplied by the sum of all average communication-time delay per token $T_{av}[l]$ for each link l constructing the paths which the input tokens use to reach the processing element running the actor A_c. The total estimated communication time will be the sum of all estimated communication-time delays of all inputs:

$$T^e_{cm}[A_c] = \sum_{j=1}^{|\mathbb{I}_c|} \left(\sum_{i=1} T_{av}[l_i] \right) \times NT_{total}[A_{c_{[I_j]}}] \quad \ni l_i \in \left\{ \mathbb{L}_{\mathcal{P}_{[PE_W,MFM_f]}} \cup \mathbb{L}_{\mathcal{P}_{[MFM_f,PE_R]}} \right\} \quad (17)$$

In addition, the estimated computation time $T^e_{cp}[C_{A_c,p}]$ of the moved actor A_c is determined depending on the recorded computation time of the moved actor A_c during the previous mapping $T_{cp}[A_c]$ and the estimated total speed-up ratio $SU^e_{total}[C_{A_c,p}]$, which is achieved when moving A_c to p:

$$T^e_{cp}[C_{A_c,p}] = T_{cp}[A_c] \times SU^e_{total}[C_{A_c,p}] \quad (18)$$

such that

$$SU^e_{total}[C_{A_c,p}] = \frac{\mathcal{A}_{A_c}[p]}{\mathcal{A}_{A_c}[looser]} \times \frac{f[looser]}{f[p]} \quad (19)$$

where $f[p]$ is the operating frequency of processor p (Table II) and $\mathcal{A}_{A_c}[p]$ is the acceleration enhancement ratio of the moved actor A_c when running on processor p (Table I).

Note that for all mapping combinations, the period of the *looser* processor is modified when an actor A_c is supposed to be mapped to another processor p. Hence, it is updated by subtracting the actual communication time $T_{cm}[A_c]$ and the actual computation time $T_{cp}[A_c]$ of the moved actor A_c:

$$Period^e_{looser}[C_{A_c,p}] = Period_{max} - T_{cm}[A_c] - T_{cp}[A_c] \quad (20)$$

For each mapping combination, the manager determines the maximum estimated period $Period^e_{max}[C_{A_c,p}]$ which denotes the maximum period among all processors when actor A_c is mapped to processor p. Fig. 10 demonstrates an example of finding $Period^e_{max}[C_{A_c,p}]$. The figure considers the example illustrated in Fig. 9. Three actors are mapped to the *looser* processor PE_1. The candidate list $\mathbb{C}$ includes three actors: A_1, A_2 and A_3. Six mapping combinations are illustrated: C_{A_1,PE_2}, C_{A_1,PE_3}, C_{A_2,PE_2}, C_{A_2,PE_3}, C_{A_3,PE_2} and C_{A_3,PE_3}. The figure shows how to find the maximum estimated period $Period^e_{max}[C_{A_c,p}]$ for each mapping combination. It is shown in the figure that both the estimated communication time and estimated computation time of the same actor differ when mapped to different PEs.

These computed new periods are then used to find the performance gain related to each mapping combination:

$$Gain^e_{per}[C_{A_c,p}] = Period^e_p[C_{A_c,p}] - Period_{max} \quad (21)$$

b) Estimated migration cost: The migration cost of an actor is the required time to transfer its binary code into the local memory of the new hosting processing element. It depends on the size of the binary data required to be transferred and the communication-time delay in the network. The sizes of the binary codes of all actors are considered to be known by the manager in terms of number of flits. Accordingly, the migration cost of the moved actor is estimated by the manager using the estimated NoC communication-time delay. When adopting APTD method, the estimated migration cost related to the moving of actor A_c to processor p is calculated as expressed in (22):

$$Cost^e_{mig}[C_{A_c,p}] = size_{bin}[A_c] \times T_{av} \quad (22)$$

where $size_{bin}[A_c]$ is the size of the binary code of actor A_c and T_{av} is the average path communication-time delay per token (4). When ALTD method is adopted, the migration cost of the moved actor A_c is determined by (23):

$$Cost^e_{mig}[C_{A_c,p}] = size_{bin}[A_c] \times \left(\sum_{i=1} T_{av}[l_i] \right) \quad \ni l_i \in \mathbb{L}_{\mathcal{P}_{[BCM,p]}} \quad (23)$$

c) Estimated total gain: The manager computes the total gain estimated to be achieved for all mapping combinations by finding the difference between the estimated performance gain $Gain^e_{per}[C_{A_c,p}]$ and the estimated migration cost of the actor $Cost^e_{mig}[C_{A_c,p}]$.

$$Gain^e_{total}[C_{A_c,p}] = Gain^e_{per}[C_{A_c,p}] - Cost^e_{mig}[C_{A_c,p}] \quad (24)$$

The moving of an actor would lead to permanent performance gain and the migration cost is paid once. However, the estimated performance gain takes the cost of migration into account in order to aggravate the probability of enhancing the overall performance directly after applying the move (in the next observation window). In fact, the variation of the input data and its corresponding effects on executing the

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2022.3177957, IEEE Transactions on Parallel and Distributed Systems

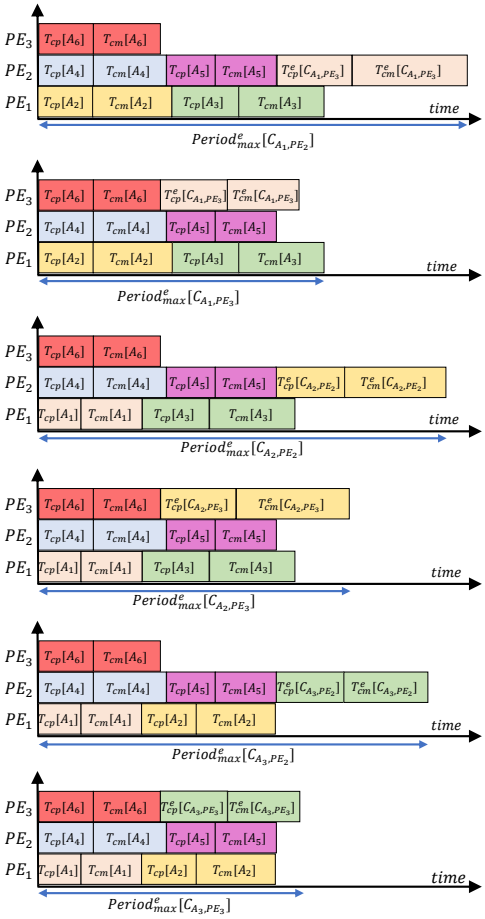


Fig. 10. An example of finding the maximum periods for each mapping combination

involved actors incites to consider worst case (severe) decision where the performance enhancement should be guaranteed once moving the actor.

Then, the manager finds the maximum achieved total gain among all mapping combinations and accordingly specifies the actor to be moved and the *gainer* processing element.

F. Moving the actor to the gainer processor

The PE, after finishing the execution of the current running actor, retrieves the new mapping information and sends directly a confirmation packet so that the manager processor manages the transfer of the object code corresponding to the new mapped actor. Before running the moved actor, the PE checks the availability of the object file corresponding to the actor in its cache memory. Note that for the initial mapping, the manager generates and sends packets to all PEs in charge of executing actors. Whereas, after executing the RR algorithm, the manager informs only the *gainer* and *looser* processors. This procedure reduces the traffic in the network and maintain the processing performance since the PEs that are not affected by remapping process are not disturbed. In fact, the manager informs first the *looser* processor about the new mapping information. Then, it waits until the *looser* processor confirms the well reception. The *looser* processor sends a confirmation packet to the manager whenever it finishes the execution of the moved actor. When the manager receives the confirmation packet, it sends the new mapping information to the *gainer* processor. Later, the *gainer* sends a confirmation packet to the manager that directly manages the transferring of the object code of the mapped actor from the shared memory into the cache memory of the *gainer* processor by making use of BCPs described in subsection III-D2b. This guarantees that the actor is executed by only one PE in the whole platform and ensure better controlling of the traffic while migrating the binary codes. In fact, the manager sends a CTP (subsection III-D1h) which includes the ID of the *gainer* processor, the ID of the moved actor and the size of the BCPs (capacity) as described in subsection III-C1. After receiving the CTP, the MAM module, which is integrated into the NI of the BCM (subsection III-C1), manages retrieving the binary code from the shared memory and dividing it into sections according to the capacity specified by the manager. The generated BCPs will be transferred to gainer processor. In our work, we consider that the gainer processor can start executing the actor once at least 256 bytes, which construct 8 lines of the L1-I cache, are received and stored to the *gainer* processor local memory. The hierarchy of the PEs' local memories includes L1 and L2 caches. L1 cache is broken up into to halves, instruction (L1-I) and data (L1-D) each of 32KB. L2 cache size is of 256KB and is used for instructions and data.

Algorithm 4 Run-time Remapping (RR)

Step 1: *Calculate the period of each PE*
Step 2: *Find PE with Max. period and assign it as looser*
Step 3: *Find the total gain (performance - migration cost)*
for each move **do**
  a- *Find the performance gain*
    . *find the period for each PE*
    . *find the maximum period*
  b- *Find the migration cost*
  c- *Calculate the total gain*
end for
Step4: *Choose the move with Max. positive total gain*

G. RR Algorithm Complexity

The devised algorithm consists of several steps summarized in Algo. 4. The complexity of each step is illustrated to determine the overall complexity. The complexity of Step1,

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2022.3177957, IEEE Transactions on Parallel and Distributed Systems

the step of finding the period of each processor, is $O(|\mathbb{P}|)$. Then, Step2, the step of finding the processor with maximum period has the complexity of $O(|\mathbb{P}|)$. The complexity of Step3, estimating the total gains corresponding the move of the candidate actors to all PEs rather than the looser processor, is $O((|\mathbb{P}|-1).|A_c \in \mathbb{C}|)$. The complexity of Step4, choosing the best move, is $O(|A_c \in \mathbb{C}|.(|\mathbb{P}|-1))$. If we consider a well balanced distribution of actors among the processors at initiation ($|A_c \in \mathbb{C}| \approx \frac{|\mathbb{A}|}{|\mathbb{P}|}$), the overall complexity becomes $O(|\mathbb{P}|) + O(|\mathbb{A}|)$ knowing that $\frac{|\mathbb{P}|-1}{|\mathbb{P}|} \approx 1$. Note that the algorithm is computed when all monitoring data is collected, so the maximum rate is once per execution of the whole data flow, and in practice can be tuned to be slower. With respect to the complexity and the execution rates of actors, this complexity is extremely low.

V. Experiments and Results

A. Application Model

In this work we target the multimedia application domain. We adopt the well-known MPEG4 part 2 Simple Profile video decoder (MPEG4-SP). This multimedia application is typically used in de-compression of encoded video digital data. Fig. 11 presents the structure of decoder as described in Reconfigurable Video Coding framework (RVC) [3] [33].

MPEG4-SP is specified with heterogeneous dataflow MoCs and includes up to 40% of dynamic actors [34]. It is composed of 41 actors and 70 FIFOs specified in RVC-CAL language. The ORCC tool is utilized for compiling and software synthesis [3] and we make use of the generated C-code for multi-core platforms. We also use the structure of the software FIFO presented in Fig. 1-b), which is generated by ORCC.

A FIFO may have several reader actors but only one writer actor. It opts an indexing mechanism such that a specific index is assigned to each reader or writer actor. These indexes are used to determine the number of available tokens corresponding to each reader actor and the free space in a FIFO. The number of available tokens ($T_f[R_i]$) in a FIFO (f) is the difference between the reader index ($I_f[R_i]$) and the writer index ($I_f[W]$). The free space in a FIFO is the number of memory addresses that contain no more needed data from all reader actors. In other words, it is the subtraction of the maximum available tokens from the total FIFO size ($Size_f$).

Each actor has its input and output ports and includes one or several actions. An action describes a specific functionality and is executed (fired) when a set of conditions, so-called firing rules, are satisfied. As an example, a firing rule consists of checking if the number of available tokens in the input FIFO is greater than the required number for computation, and that the output FIFO has sufficient empty room to store the produced tokens. In MPEG4-SP, the number of reader actors ranges from 1 (at least) to 6 (at most).

MPEG RVC defines RVC-CAL applications as dynamic dataflow applications, where the uncertainty of computing due to data-dependency prevents from any static scheduling. They are based on dataflow process network (DPN) model [6]. In such model, the actor executes when at least one of its firing rules is satisfied. For cases where several firing rules

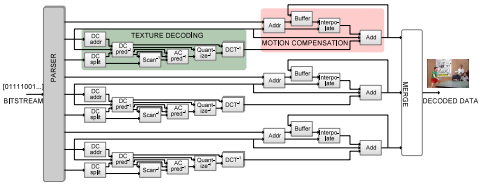


Fig. 11. MPEG4 part 2 SP decoder [33]

are satisfied simultaneously, only one is selected according to its priority. Consequently, its corresponding satisfied action is fired. Each firing consumes input tokens and produces output tokens. The number of the consumed or produced tokens may be fixed or variable.

B. Experimental framework and setup

In order to assess the feasibility of our proposed run-time remapping method, we developed a real-time simulator. The simulator is described in SystemC TLM model [35]. The devised simulator models a MPSoC platform using NoC concept for interconnecting embedded modules. The platform incorporates heterogeneous processing elements (Table I), memory blocks, and the manager. The simulator platform has been designed with hierarchical modules that can work concurrently and intercommunicate via ports using simple or complex communication channels. SystemC features have been exploited to mimic the accurate functionality of the modules described in section III.

The adopted NoC-based architecture, presented in section III, is implemented in the devised simulator platform. In order to accurately model the adopted application, all involved actions are functionally simulated to determine their execution-timing features and generate the real data exchanged by actors during video decoding. The SystemC model adopted in the simulation platform is cycle accurate at the level of the NoC and the network interfaces. The timing of all corresponding action executions on PE is compensated in the simulation according to the profiling data extracted while running the application on a reference computer. Profiling data provides, for each involved action, the mean value of the number of cycles required to execute it. In this work, profiling data has been extracted using on a desktop computer (i7-2620M CPU@2.7 GHz and 8GB memory). We consider that the NoC operating frequency f is 500 MHz. The clock cycle in each PE is determined according to Table II. During SystemC simulations, for each fired action, its corresponding execution time determined in profiling is mapped according to the processor frequency and used as time delay to compensate the real execution time. In addition, several benchmark video sequences with different formats from [36] have been encoded. The selected video sequences have different manner in changes between successive frames. This guarantees to evaluate the performance of the proposed algorithm for different data-dependent behaviors. The resultant data has been used as input to the decoder. These same encoded videos have been

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2022.3177957, IEEE Transactions on Parallel and Distributed Systems

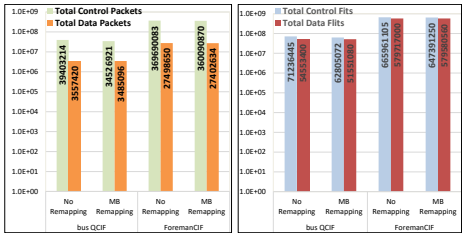


Fig. 12. Classification of transported packets and flits

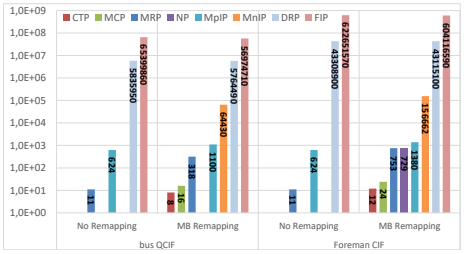


Fig. 13. Classification of control flits according to their types

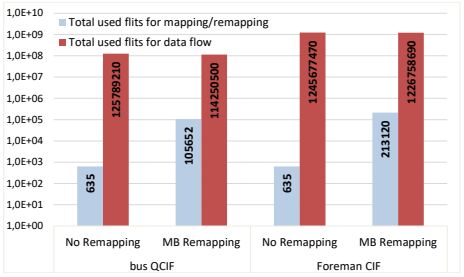


Fig. 14. Number of transported flits

decoded on a desktop computer and the FIFO contents have been traced over the decoding period. To verify the proper functionality of each actor, the contents stored in the FIFOs in the simulator have been compared to the traced FIFO data. Also, the output data of the simulator have been reconstructed into visual video in order to verify the functionality of the devised simulator. The video sequences have been decoded without applying remapping targeting the same NoC-base architecture and the obtained results have been compared to that obtained when the video sequences are decoded adopting the MB remapping algorithm applying ALTD and APTD for estimating the communication time delay while considering an observation window of $N_f = 10$.

C. Experimental Results

1) Transported data: The number of packets that travel through the network during the decoding of the video sequences, and their corresponding flits are recorded in the case of applying the MB remapping and the case of decoding the video without remapping. Fig. 12 presents the number of transported packets and flits in logarithmic scale during decoding the Foreman video with CIF format and Bus video with QCIF format for the case of adopting MB remapping algorithm and the case of ordinary decoding. The packets and flits are classified into control and data categories. The figure shows that the flits of control packets form about 53% of all transported flits in the two cases.

Furthermore, investigating thoroughly the types of transported control flits illustrates that 93% of control flits belong to FIP. This refers to the MoC adopted in dataflow applications which requires checking the firing rules (availability of input data and output buffer space). Fig. 13 shows in logarithmic scale the number of each type of control flits transported while decoding the Foreman video sequence in CIF format and Bus video sequence in QCIF format for the case of MB remapping and the case of ordinary decoding.

Also, Fig. 13 shows that additional flits are transported in the network due to the remapping. In fact, applying remapping induces additional control and data packets. In order to evaluate the effect of applying the MB remapping algorithm on the traffic in the network, the transported flits are classified into two main categories. The first category includes the flits which are used basically for dataflow. This category encompasses the flits which occupy the payload of all FIP, DRP and DFP. The second category includes the induced flits by applying the remapping algorithm. Hence, the second category compromises the flits listed in the payloads of NP, MRP, MCP, M_nIP, M_pIP, CTP, and BCP. Note that both categories include data and control packets. Fig. 14 illustrates the comparison summary in terms of the number of transported flits of both categories. In the figure, the number of transported flits, which is obtained while processing the Foreman video with CIF format and Bus video with QCIF format, is presented in logarithmic scale for both cases (decoding while applying remapping algorithm and ordinary decoding). The comparison shows that the additional flits induced by applying the MB algorithm forms less than 0.02% from total transported flits. In addition, Table V shows the percentage of flits transporting the binary code of migrated actors from the total number of transported flits in the network while decoding several video sequences. The presented percentages illustrate that the impact of actor migration on the traffic is negligible.

TABLE V
PERCENTAGE OF FLITS TRANSPORTED IN BCP FROM TOTAL FLITS

Video		Remapping Algorithm	
Sequence	Format	MB-ALTD	MB-APTD
Foreman	CIF	0.0044%	0.0067%
Bus	CIF	0.0008%	0.0125%
Ice	4CIF	0.0027%	0.0019%
Bus	QCIF	0.0348%	0.0272%

2) Packet time-delay: The packet time-delay is recorded while decoding the video sequences, following the procedure explained in subsection IV-B. Fig. 15 presents the variation

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2022.3177957, IEEE Transactions on Parallel and Distributed Systems

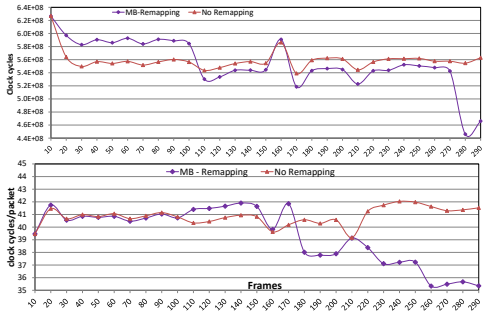


Fig. 15. Sum of packet time-delays (top) and average packet time-delay (bottom) while decoding Foreman video with CIF format [36]

of the sum of packet time-delays throughout the observing windows during the decoding of the Foreman video sequence with CIF format when adopting the MB remapping technique. It is noticed that applying the MB remapping algorithm affects the time-delay of the packets. In addition, the figure shows the comparison with the case of ordinary decoding. The comparison illustrates that using MB remapping decreases gradually the total packet time-delay. Note that the task moves occur after processing 80, 100, 160, and 270 frames. Fig. 15 shows that the total packet delay decreases after the conducted moves. This refers to the fact that task remapping contributes in distributing the tasks on PEs that are nearer to the memory modules accommodating the input and output FIFOs. Also, Fig. 15 presents a comparison in terms of average time-delay of packets transported during the decoding of the Foreman video with CIF format when adopting the MB remapping technique and when using ordinary decoding. The comparison confirms that the use of MB remapping technique contributes significantly in reducing the time-delay.

3) Timings: Fig. 16 presents the recorded total communication time and total computational time throughout the observing windows during the decoding of the Foreman video with CIF format when adopting the MB remapping technique and when using ordinary decoding. It shows that the communication time represents 90% of the total execution time in both cases. Hence, the total execution time is affected more by the variation of the total communication time. Also, Fig. 16(a) shows that the total communication time is almost not changing among observation windows in the case of ordinary decoding. Whereas, when MB technique is applied, the communication time varies significantly and tends to follow a decreasing manner as shown in Fig. 16(b). This illustrates that reducing the time-delay achieved by MB remapping has a direct impact on the communication time.

The communication time of each processing element is investigated through the decoding of all video frames. It is noticed that when applying the MB remapping technique, the variation between communication times of all involved PEs is reduced. The communication time values of all PEs converges gradually to a specific interval as shown in Fig. 17.

4) Performance results: Multiple simulations have been conducted to decode several benchmark video sequences from [36]. Fig. 18(a) presents the achieved throughput in terms of frames per second (FPS) when decoding Foreman video (CIF format) and using ALTD and APTD respectively for estimating the NoC communication time delay. The figure also shows the achieved throughput when decoding the Foreman video (CIF format) without remapping. The letter "M" shown on the curves represents when an actor move occurs. Fig. 18(a) shows that using MB results in significant performance enhancement. In addition, the figure illustrates that adopting ALTD for estimating the NoC communication time delay, while decoding Foreman video sequence with CIF format, increases the achieved enhancement ratio. Other similar simulations have been conducted targeting other video sequences with different formats (CIF, 4CIF and QCIF). The selected videos are of diverse characteristics to ensure that the proposed remapping algorithm is not related to specific formats or video content. The obtained results confirm that adopting MB algorithm ensures enhanced performance when compared to decoding the video without remapping. Also, the results demonstrate that adopting ALTD rather than APTD leads to additional performance enhancement.

D. Discussion and Comparison

In order to determine the relevancy of the devised algorithm, it is compared to the STM method introduced in [31]. To achieve fair comparison, the STM method has been modeled and implemented on our devised NoC-based architecture. We have also implemented the exact method presented in [20] for the initial mapping, with two differences: we have used constraint programming instead of ILP, and the objective function is the maximum period, Eqn. 12, as it is our optimization goal. The workload used for the computation time of the actors is based on the profiling of Foreman video. Simulations have been conducted while running the MPEG4 decoder to process real-life videos.

1) Performance enhancement of MB remapping: The results presented in Fig. 18 show that for Foreman video sequence with CIF format (Fig. 18(a)), the use of MB remapping algorithm when adopting ALTD leads to a maximum performance enhancement of 38.2% (frame 280) and adopting MB-APTD leads to a maximum performance enhancement of 14.8% (frame 210) when compared to the results of processing the video without remapping. For Ice video sequence with 4CIF format (Fig. 18(b)), maximum performance enhancement of 56% (frame 450) and 16.5% (frame 250) are recorded when applying the MB algorithm adopting ALTD and APTD respectively. Furthermore, the use of MB algorithm adopting ALTD and APTD leads to a maximum performance enhancement of 10.92% (frame 120) and 7.6% (frame 50) for Bus video sequence with QCIF format (Fig. 18(c)) and Bus video with CIF format (Fig. 18(d)). For Grandma video with QCIF format (Fig. 18(e)), maximum performance enhancement of 33.2% (frame 170) and 23.9% (frame 190) are recorded when applying the MB algorithm adopting ALTD and APTD respectively. The simulation results show that the link level

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2022.3177957, IEEE Transactions on Parallel and Distributed Systems

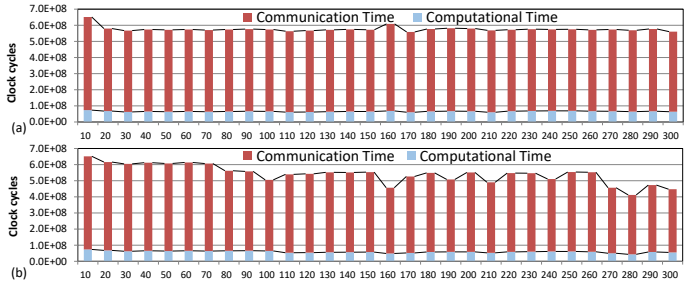


Fig. 16. Total communication and computational times recorded throughout the observing windows during the decoding of the Foreman video with CIF format [36]; when adopting (a) ordinary decoding and (b) MB remapping

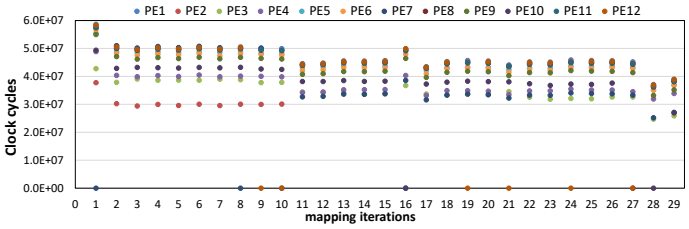


Fig. 17. PE communication time in terms of FPS of decoding Foreman video with CIF format [36] using MB remapping algorithm

estimation of ALTD is more accurate and usually leads to better performance compared to APTD. However, in some cases APTD performs better such as for some observation windows of Grandma video (Fig. 18(e)). This refers to the fact that the heuristic is data-dependent and the link level prediction depends on the monitoring information collected during the previous data which may not match with the that of the current processed data.

2) MB remapping in comparison to STM remapping: Fig. 18 shows a comparison between our proposed remapping and the STM algorithm in terms of throughput (FPS). The results shows that the MB remapping outperforms STM remapping technique when considering either APTD or ALTD for estimating the NoC communication time delay.

Besides, the graphs in Fig. 18 show that in some cases the STM method leads to deterioration in the performance. In fact, the STM method selects critical task to be moved in each observation window without estimating the resulting total performance gain. Moving the task without determining its effects on the whole system performance degrades the overall performance. While in our proposed algorithm, the maximum achieved total gain among all mapping combinations is first determined as explained in subsection IV-E2c. Accordingly, a task is specified to be moved if the estimated maximum total gain is positive. It is noticed that in some observation windows no tasks are moved when the proposed algorithm is applied. A move is indicated by letter "M" in Fig. 18(a). In these cases, the estimation shows that no performance enhancement will be achieved for all mapping combinations.

3) MB remapping in comparison to optimal mapping: Fig. 18 also shows the results obtained from the mapping approach proposed in [20]. Note that the "optimal" mapping corresponds to the best mapping found based on the profiling of Foreman video after a time out of one hour (like the original paper), and the optimality is not proven. The results show that the MB algorithm, starting from a random mapping (without significant initial delay), performs better that the optimal with no remapping for Foreman video sequence in CIF format (Fig. 18(a)). As the optimality is searched for the Foreman profile, we used the optimal mapping as a starting point for the MB algorithm, and the results show that it further improves the throughput. As expected, the optimal mapping for Foreman does not perform good for the Ice video sequence in 4CIF format (Fig. 18(b)) and Grandma video sequence in QCIF format (Fig. 18(e)). But surprisingly, it performs good for the Bus video in QCIF format (Fig. 18(c)) and Bus video in CIF format (Fig. 18(d)). The so-called optimal method cannot be used for two reasons. First it introduces an unpractical initialization delay without guaranty of optimality. Secondly, a static solution is not appropriate to data-dependent applications since a solution can be good for one data-stream and inefficient for another one and more importantly the efficiency of a mapping varies over time.

4) Comparison summary: Table VI summarizes the comparison of average FPS achieved when processing multitude video sequencing while adopting different remapping tech-

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2022.3177957, IEEE Transactions on Parallel and Distributed Systems

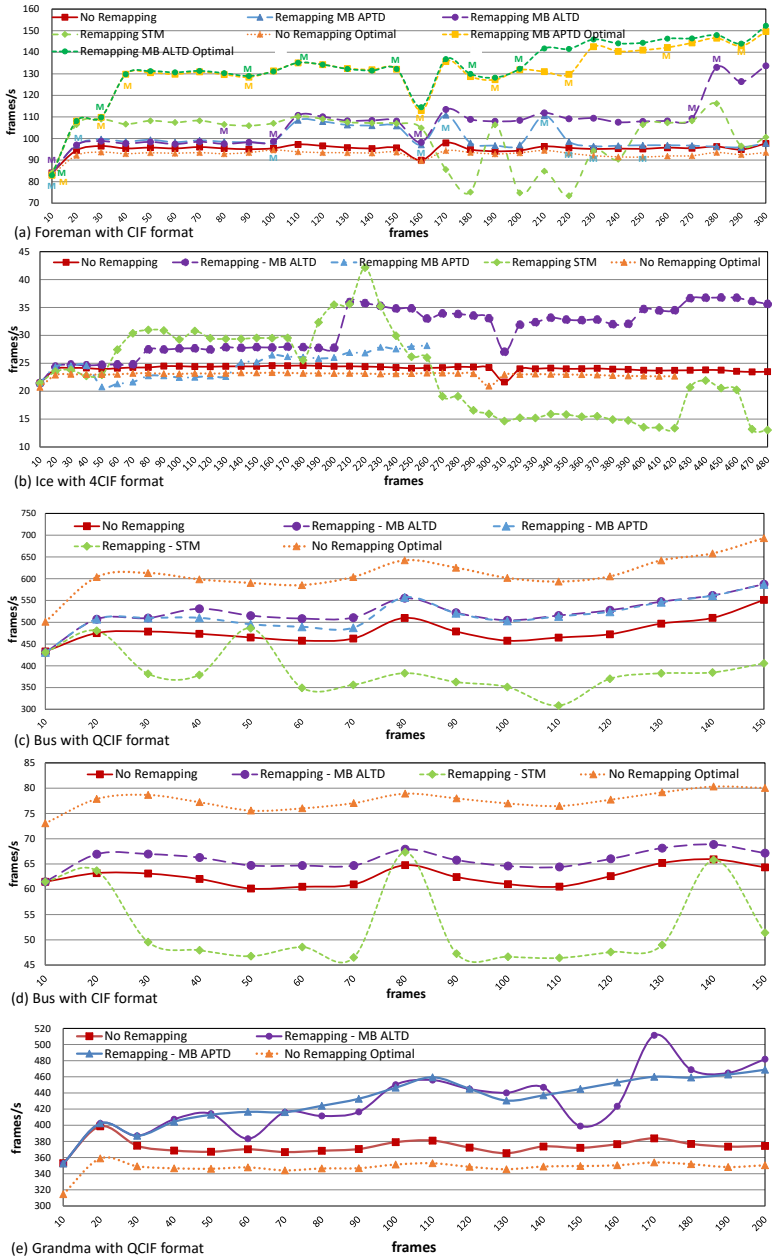


Fig. 18. Throughput in terms of FPS when decoding video sequences [36] using MB and STM remapping algorithms

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2022.3177957, IEEE Transactions on Parallel and Distributed Systems

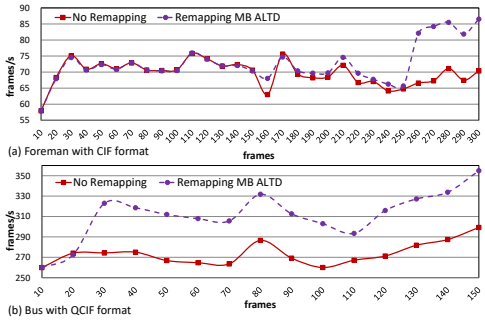


Fig. 19. Throughput in terms of FPS when decoding video sequences [36] using MB remapping algorithms targeting 4×6 NoC

TABLE VI
ACHIEVED RESULTS ADOPTING DIFFERENT REMAPPING TECHNIQUES

Video		Remapping Algorithm		
Sequence	Format	MB-ALTD	MB-APTD	STM
Foreman	CIF	11.4%	5%	4.1%
Bus	CIF	5.4%	5.4%	−17.7%
Ice	4CIF	26.1%	2%	−13.04%
Bus	QCIF	9%	8%	−20%
Grandma	QCIF	14.91%	14.11%	NA

niques. The table shows that the MB algorithm achieves the maximum average performance enhancements of 26% and 14.11% when adopting ALTD and APTD respectively compared to the achieved throughput of processing the frames without remapping. Whereas, remapping using STM algorithm achieves a maximum average enhancement of 4%.

E. Scalability and generality

The scalability of our approach relies first on a negligible extra payload in the context of actor-level dataflow models, which intrinsically require a large amount of small control packets. For example, when decoding the Foreman video sequence the extra flits imposed by remapping (including the flits holding the binary codes of moved actors) constitute less than 0.02% of the flits used for dataflow. The proposed remapping method enhances the performance by exploiting the NoC structure and the characteristics of the available resources. The results show that our method positively impacts the NoC performance. Table VII illustrates the reduction percentages of packet hops when decoding different video sequences adopting the proposed MB remapping compared to ordinary decoding without remapping. The comparison shows that the proposed remapping method reduces the packet hops. The percentage of reduction is more than 20%. Secondly, the method includes the migration cost and so limits the number of moves.

Fig. 19 shows the results obtained for a 4×6 NoC, for Foreman and Bus video sequences, starting from a random mapping. The results show that our approach can also improve the throughout for a larger NoC. On average, the throughout is improved by 13.5% and 4% for Bus QCIF and Foreman CIF videos respectively.

TABLE VII
REDUCTION OF PACKET HOPS WITH MB-ALTD AND MB-APTD

Video		Remapping Algorithm	
Sequence	Format	MB-ALTD	MB-APTD
Foreman	CIF	20.94%	12.64%
Bus	QCIF	3.24%	5.29%
Grandma	QCIF	14.18%	8.33%

VI. CONCLUSION

This paper presents an original *Move*-based algorithm and NoC-based architecture to map the tasks of dataflow application during run-time. The method monitors the performance and intercommunication, takes the proper mapping decision and applies the required mapping configurations. The algorithm and the devised architecture are thoroughly presented. The best way to verify the effectiveness of a run-time mapping, which is by definition data dependent, is to simultaneously execute the target application. However such demonstrations are complex, time consuming and so ignored in the literature. In this paper we address this issue by conducting a SystemC simulation of the MPEG4-SP decoder with several real-life video sequences. The obtained results demonstrate that the proposed algorithm significantly enhances the performance. In addition, the proposed algorithm outperforms the available run-time mapping technique. Future work will consider the implementation of integrated module in the NIs and estimating the overhead in terms of area and energy.

REFERENCES

[1] W. A. Najjar *et al.*, "Advances in the dataflow computational model," *Parallel Computing*, vol. 25, no. 13, pp. 1907 – 1929, 1999.
[2] K. J. M. Martin *et al.*, "Notifying memories: a case-study on data-flow applications with NoC interfaces implementation," in *Proc. of the Design Automation Conf. (DAC)*, June 2016.
[3] H. Yviquel *et al.*, "Orcc: Multimedia development made easy," in *Proc. of the ACM Int. Conf. on Multimedia*, ser. MM '13. New York, NY, USA: ACM, 2013, pp. 863–866.
[4] E. A. Lee and D. G. Messerschmitt, "Synchronous data flow," *Proc. of the IEEE*, vol. 75, no. 9, pp. 1235–1245, Sep. 1987.
[5] Y. Lesparre *et al.*, "Evaluation of synchronous dataflow graph mappings onto distributed memory architectures," in *Proc. of Euromicro Conf. on Digital System Design (DSD)*, Aug. 2016, pp. 146–153.
[6] E. A. Lee and T. Parks, "Dataflow process networks," in *Proc. of the IEEE*, 1995, pp. 773–799.
[7] C. Wang *et al.*, "Dynamic application allocation with resource balancing on NoC based many-core embedded systems," *J. Syst. Archit.*, vol. 79, no. C, pp. 59–72, Sep. 2017. [Online]. Available: https://doi.org/10.1016/j.sysarc.2017.07.004
[8] J. Henkel *et al.*, "Dynamic resource management for heterogeneous many-cores," in *Proc. of the IEEE/ACM Int. Conf. on Computer-Aided Design (ICCAD)*, 2018, pp. 1–6.
[9] S. Kaushik *et al.*, "Computation and communication aware run-time mapping for NoC-based MPSoC platforms," in *Proc. of IEEE Int. SOC Conf.*, Taipei, Taiwan, 2011, pp. 185–190.
[10] T. Maqsood *et al.*, "Dynamic task mapping for network-on-chip based systems," *J. of Systems Architecture*, vol. 61, no. 7, pp. 293 – 306, 2015.
[11] H. R. Mendis *et al.*, "Dynamic and static task allocation for hard real-time video stream decoding on nocs," *Leibniz Transactions on Embedded Systems*, vol. 4, no. 2, pp. 01:1–01:25, Jul. 2017.
[12] M. Rapp *et al.*, "Neural network-based performance prediction for task migration on S-NUCA many-cores," *IEEE Transactions on Computers*, pp. 1–1, 2020.
[13] S. Paul *et al.*, "Adaptive task allocation and scheduling on NoC-based multicore platforms with multitasking processors," *ACM Trans. Embed. Comput. Syst.*, vol. 20, no. 1, Dec. 2020.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2022.3177957, IEEE Transactions on Parallel and Distributed Systems

[14] ——, "A hybrid adaptive strategy for task allocation and scheduling for multi-applications on NoC-based multicore systems with resource sharing," in *Proc. of Design, Automation Test in Europe Conf. Exhibition (DATE)*, 2021, pp. 1663–1666.
[15] Z. Li *et al.*, "Chordmap: Automated mapping of streaming applications onto CGRA," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, pp. 1–1, 2021.
[16] D. Huff *et al.*, "Clockwork: Resource-efficient static scheduling for multi-rate image processing applications on FPGAs," in *Proc. of IEEE Annual Int. Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2021, pp. 186–194.
[17] T. D. Ngo *et al.*, "Move based algorithm for runtime mapping of dataflow actors on heterogeneous MPSoCs," *J. of Signal Processing Systems*, vol. 87, no. 1, pp. 63–80, Apr. 2017.
[18] A. Singh *et al.*, "Mapping on multi/many-core systems: Survey of current and emerging trends," in *Proc. of the Design Automation Conf. (DAC)*, May 2013, pp. 1–10.
[19] P. K. Sahu and S. Chattopadhyay, "A survey on application mapping strategies for Network-on-Chip design," *J. of Systems Architecture*, vol. 59, no. 1, pp. 60–76, 2013.
[20] K. Huang *et al.*, "A scalable and adaptable ILP-Based approach for task mapping on MPSoC considering load balance and communication optimization," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 38, no. 9, pp. 1744–1757, Sep. 2019.
[21] C. Rubattu *et al.*, "Pathtracing: Raising the level of understanding of processing latency in heterogeneous mpsocs," in *Drone Systems Engineering and Rapid Simulation and Performance Evaluation: Methods and Tools*, ser. DroneSE and RAPIDO '21. NY, USA: ACM, 2021, pp. 46–50. [Online]. Available: https://doi.org/10.1145/3444950.3447282
[22] C. Chou and R. Marculescu, "Run-time task allocation considering user behavior in embedded multiprocessor networks-on-chip," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 29, no. 1, pp. 78–91, 2010.
[23] E. L. d. S. Carvalho *et al.*, "Dynamic task mapping for MPSoCs," *IEEE Design Test of Computers*, vol. 27, no. 5, pp. 26–35, 2010.
[24] M. Fattah *et al.*, "Adjustable contiguity of run-time task allocation in networked many-core systems," in *Proc. of Asia and South Pacific Design Automation Conf. (ASP-DAC)*, 2014, pp. 349–354.
[25] A. K. Singh *et al.*, "Resource and throughput aware execution trace analysis for efficient run-time mapping on MPSoCs," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 35, no. 1, pp. 72–85, 2016.
[26] C. Ykman-Couvreur *et al.*, "Linking run-time resource management of embedded multi-core platforms with automated design-time exploration," *IET Computers & Digital Techniques*, vol. 5, pp. 123–135(12), Mar. 2011.
[27] W. Quan and A. D. Pimentel, "A scenario-based run-time task mapping algorithm for MPSoCs," in *Proc. of the Annual Design Automation Conf. (DAC)*, New York, NY, USA, 2013.
[28] A. K. Singh *et al.*, "Accelerating throughput-aware runtime mapping for heterogeneous MPSoCs," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 18, no. 1, Jan. 2013.
[29] S. Skalistis and A. Simalatsar, "Near-optimal deployment of dataflow applications on many-core platforms with real-time guarantees," in *Proc. of the IEEE Design, Automation Test in Europe Conf. Exhibition (DATE)*, 2017, pp. 752–757.
[30] H. Yviquel *et al.*, "Towards run-time actor mapping of dynamic dataflow programs onto multi-core platforms," in *Proc. of the Int. Symposium on Image and Signal Processing and Analysis (ISPA)*, 2013, pp. 732–737.
[31] W. Quan and A. D. Pimentel, "A hybrid task mapping algorithm for heterogeneous MPSoCs," *ACM Trans. on Embedded Computing Systems (TECS)*, vol. 14, no. 1, pp. 14:1–14:25, Jan. 2015.
[32] J.-P. Diguet *et al.*, "Networked power-gated MRAMs for memory-based computing," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 26, no. 12, Dec. 2018.
[33] S. S. Bhattacharyya *et al.*, "Overview of the MPEG reconfigurable video coding framework," *Journal of Signal Processing Systems*, vol. 63, no. 2, Dec. 2011.
[34] M. Wipliez and M. Raulet, "Classification of dataflow actors with satisfiability and abstract interpretation," *Int. J. of Embedded and Real-Time Communication Systems (IJERTCS)*, vol. 3, no. 1, pp. 49–69, 2012.
[35] T. Grotker *et al.*, *System Design with SystemC*. Springer, 2002.
[36] Xiph.org video test media. [Online]. Available: http://media.xiph.org/video/derf/

Mostafa Rizk received his Maitrise degree in Electronics, M.Sc in Biomedical Physics, and M.Sc in Signal, Telecom, Image, and Speech from the Lebanese University in 2007, 2008 and 2010 respectively. He received his Ph.D. degree in Sciences and Technologies of Information and Communication from Telecom Bretagne, France in 2014 and a Ph.D degree in Electronics and Telecommunication from the Lebanese University in 2015. Dr. Rizk has been a post-doctoral researcher at UBS University and Lab-STICC laboratory CNRS, France. Dr. Rizk has been an associate professor at LIU, Lebanon and associate researcher at IMT-Atlantique, France. Currently, Dr. Rizk is a researcher at Lab-STICC laboratory CNRS, Brest, France. His general research interests include both algorithm development and corresponding hardware/software implementations and digital circuit design; NoC design and new MPSoC architectures based on emerging non-volatile memory technologies; ICT of drone systems; embedded machine learning and embedded computer vision.

Kevin J. M. Martin received a M.S. degree in electrical and computer engineering in 2004 and a PhD in computer science in 2010 from the Université de Rennes, France. He is since 2011 an associate professor at Université Bretagne-Sud in Lorient, France, in the Lab-STICC. His research interests stand at the crossing point between architecture, methods and tools, including but not limited to: custom processors, CGRA, multi-processor platforms, high-level synthesis, computer-aided design tools, compilers and software engineering.

Jean-Philippe Diguet is a CNRS director of research. He has been A/Prof. at UBS University and research visitor/invited Prof. in different Institutions (IMEC/Belgium, UQ/Australia, Tohoku/Japan, USP/Brasil). At Lab-STICC he has led the team MOCS from 2008 to 2016 and then the ITC and Drones program. Since 2021 his is the director of CROSSING, an International CNRS Lab in Adelaide, Australia dedicated to Human/Automous-Agents teaming. His research work focused initially on various aspects of embedded system design and now addresses various levels of embedded intelligence.

Chapter 6

Flexible and Efficient Architectures Based on Memristive Technologies

6.1 Networked Power-Gated MRAMs for Memory-Based Computing

6.1.1 Preface

This research work has been conducted during my work at UBS in the context of Cyam research project. The work is a collaboration between Lab-STICC and Research Institute of Electrical Communication at University of Tohoku, Japan in particular with Professor Takahiro Hanyu and Associate Professor Naoya Onizawa. This research work has been published in the proceeding of the *IEEE International New Circuits and Systems Conference (NEWCAS)*, 2017 [76] and in the *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* [77], 2018. Also, additional architectural details and results have been elaborated in the proceeding of the *IEEE International Multidisciplinary Conference on Engineering Technology (IMCET)* [37], 2021.

6.1.2 Introduction

Computing resources are continuously increasing with technology progress, yet the use of the computing capabilities is facing major limitations in terms of power dissipation, off-chip memory access time, and real available parallelism. These limitations lead to a partial use of the offered integration capabilities and increase the gap between accessible technology nodes and available design methodologies. Nevertheless, the evolution of the emerging non-volatile memory (NVM) technologies, open new design perspectives to balance the under-use of available hardware resources.

Besides memory access time reduction due to on-chip integration, the silicon density offered by recent technology nodes can be efficiently used, and significant reductions in power consumption can be achieved as long as the leakage power can be controlled and the

memory bandwidth exploited. Such a configuration is made possible by applying memory-based computing (MBC) [78] while considering a network-on-chip to broadcast requests to NVM devices and full power gating (PG) to switch-OFF unused memories.

This research work proposes a novel approach that distributes and moves computing resources closer to memory. The main idea is to replace computing by memory like. It considers the replacement of results stored in memory to take advantage of dense, low power memories. Distributed memories are adopted to increase the bandwidth by making use of the increase of on-chip memory density and the emerging manycore architectures based on NoC. In practice, all memories are not addressed at the same time leading to inherently partial resource usage in time. Therefore, applying power-gating techniques to unused resources can offer important power savings. For cost and efficiency reasons, emerging NVM memories are reused without modification except the implementation of efficient power-gating techniques. Rather than implementing logic in memories, the NIs of the NoC are enhanced to turn power-gated memories into content-addressable memories. It is also important to note that our NoC-Memory Based Computing (NMBC) concept can be generalized and additional functionality (e.g. encryption, max, sum, sigmoid, etc.) can be implemented in the NIs.

6.1.3 Context and Motivations

A large increase in the applications that can take advantages of the capability to integrate much more on-chip memories. Datacenters related applications are one of the main examples, where the energy efficiency constraints lead to the rise of heterogeneous architectures [79]. Neuromorphic architectures based on convolutional neural networks (CNN) [80], Spiking neural networks [81] or Sparse Neural Networks (SNN) [82] are another major application examples dominated by memory resources. It is worth noting here that this class of algorithms usually implies much more reading than writing accesses in the operational phase after the learning stage.

Another related use of the emerging memory integration capabilities is illustrated in [83], where switch-based logic functions are replaced efficiently by memories especially in the case of molecular crossbars. In this approach, a partitioning of the application graph is applied to map computing elements on memory arrays. Replacing logic by memories is efficient in case of computation redundancy. Such patterns can be extracted from signal processing algorithms that allow partial pre-processing [84]. Another different approach consists in storing the results of frequently used computations in memory. Such methods have shown significant energy improvement in the context of heterogeneous reconfigurable multicore architectures [85] and GPGPU architectures [86].

In memory-based computing, where the application is partitioned and mapped to dense memory arrays and compute elements [85], not all memories are entirely addressed at the same time. Thus, if the memories are implemented as a grid of memory blocks, the resource usage in time is inherently partial. Therefore, applying power-gating techniques allows minimizing the power consumption of partially used resources. In this context, the emergent non-volatile memory technologies, such as magnetoresistive random-access memory (MRAM) with low-leakage characteristics and power-gating capabilities, represent a great opportunity to design energy efficient architectures. In other words, increasing on-chip resources with

low-power behaviors contribute to the global architecture and energy efficiency.

Moreover, such architectures can take real advantage of more than one decade of research and achievements in NoC design, that allow configurable accesses to grids of memories as well as the implementation of pre/post-processing steps in network interfaces [36].

Memristive technologies boost the research towards logic-in-memory approaches with the idea of introducing bit-level logical function in memories. This is presented with a programmable architecture in [87], where power-gating is not considered. The concept of active memory also enhances memory with new functionality, which is the capacity to manage high-level transactions requests with a local processor executing transfer operations [88]. These two approaches lead to specific designs, where logic and memory are tightly coupled.

Magnetic tunnel junction (MTJ) devices [89] represent a main class of emerging non-volatile technologies for low-power memory circuits. Recently, several MRAM designs have been presented [90, 91, 92] that exhibit significantly low leakage currents realized by power-gating, thanks to the non-volatility feature. These MRAM designs are designed using relatively large device technologies, such as 90 nm, yet the MTJ technology allows scaling down to 11 nm as for CMOS devices [93]. The performance of MRAM devices in such recent technology nodes are also estimated in [94, 95].

MRAM devices are designed using different MRAM cells, such as 1T-1MTJ, 2T-2MTJ, and 4T-2MTJ. This corresponds to performance trade-offs between area efficiency and read time. For example, the 1T-1MTJ MRAM exhibits the highest area efficiency with a relatively slow read time, while the 4T-2MTJ MRAM realizes faster read time with lower area efficiency. In terms of power-gating, a simple strategy that shutdowns the power of the whole MRAM device at the idle state is generally used. In the 4T-2MTJ MRAM [91], a cell-level power-gating technique is used that activates only a row at the read/write state, leading to low leakage currents in the cells.

6.1.4 Contributions and Performed Work

This research work introduces a novel computing approach that exploits memory bandwidth and limits the leakage power by applying MBC, while considering a NoC to broadcast requests to memory and NVM to switch-OFF unused memories.
The main contributions of this research work are:

- Propose a novel NMBC architecture that introduces the design of energy-efficient multicore solutions for memory-based computing. The tailored architecture interconnects three types of IP blocks: memory modules, processing elements and managers.

- Introduce a new computing paradigm is presented where the packets of a NoC are used as commands executed by configurable NI to access in parallel multiple memory blocks, which are technologically independent from the NoC. The managers send queries to memories by means of packets where known and unknown fields are specified. Then the processors collect the memories answers and apply a simple winner-take-all algorithm to select records to be send back to managers.

- Upgrade the structure of NIs associated to each memory module in order to decode the requests sent by the manager, handle the communication of results to the processors, and provide managers with monitoring data about bandwidth usage and processor usage. Note that these services are dedicated to NIs in order to remain compliant with any existing memory and to be independent from NoC parameters.

- Introduce novel implementations of Spin-Transfer-Torque Magnetoresistive Random-Access Memory (STT-MRAM) with cell and peripheral power-gating capabilities. Three different types of 256-bit $\times$ 256-word MRAMs with different power gating policies are adopted:

  1. **Type I**: controls one 256×256 MRAM and adopts one PG transistor
  2. **Type II**: controls four 128×128 MRAMs and adopts four PG transistors for thr four 128×128 MRAM subblocks
  3. **Type III**: controls four 128×128 MRAMs and adopts one PG transistor

  Type I controls one 256×256 MRAM; whereas, Type II and Type III control four 128×128 MRAMs. Type I adopts one PG transistor for a 256×256 MRAM. Type II adopts four PG transistors for four 128×128 MRAM subblocks. The maximum write bitwidth for Type III is 32 bits, whereas the maximum read bitwidth is 256 bits as for Types I and II. Adopting PG reduces the leakage current of the cell array compared to that of an Static Random-Access Memory (SRAM) cell array. The peripheral circuits can be power-gated using the pMOS transistor when the MRAM is at the idle state.

  In this work, two different policies of PG have been addressed: only cell PG (OCPG) and full PG (FPG). In OCPG, the the MRAM cells are power-gated at the idle state and the peripheral circuits are always at the active state. In FPG, the whole MRAM is power-gated at the idle state. FPG reduces the leakage current in comparison with the OCPG while a wake-up operation is required for the peripheral circuits, which cost extra energy dissipation and delay time.

- Demonstrate the effectiveness of the proposed approach through a relevant case study of a database search application implemented with neuromorphic architecture based on SNN, which is a neural network model inspired by information theory and error correcting codes. It is especially efficient for the implementation of associative memories, so it can be used to process database queries [82].

- The PEs are implemented as dedicated hardware components. A specific design is tailored in VHDL, which enables to estimate the implementation area of the PEs and include their power consumption in the global power budget.

- The RTL specification of the router and the NI is synthesized in the physical layout estimation mode. The total power was obtained by using the leakage and dynamic power of the NoC and NI components, relying on the switching activity traced by the SystemC simulation.

- Build an experimental setup that model the adopted NoC implementing MRAM memories in SystemC TLM. In order to check the relevance of the proposed approach, the

devised model executes the database search engine application using the Yeast database benchmarks (cellular localization in proteins) from the UCI Machine Learning Repository [96]. The model has been developed to mimic the exact behavior of the target architecture by designing hierarchical modules that work concurrently and intercommunicate via ports. The SytemC model is cycle accurate at the NoC level including NIs. To ensure accuracy, the timing features of memory reading response and wake ups are considered in the model. Furthermore, the execution time of computations in the PEs is based on the HDL simulation and synthesis that provide real-time environment modeling.

- Conduct experimental simulations using the devised simulation model while considering different types of memories, PG policies and number of missing fields in the queries.

- Based on the simulation results, the relevancy of the proposed concept is evaluated. The transported data is tracked. Accordingly the hit rate and injection rate are computed. Also, the power budget is estimated based on the TSMC 65-nm CMOS process, the implementation of PEs and the specifications of the memory models. The percentage of dissipated power in each module is determined and classified into static or dynamic. Also, the obtained results when adopting the MRAMs are compared with those obtained when adopting SRAMs while using identical NoC and PEs features and memory contents.

6.1.5 Findings

- The obtained results demonstrate important power reduction with database hit rates of about 94A slight degradation in the hit rate is recorded when increasing the number of missing fields in the queries. This is due to the fact that the ambiguity ratio increases with the decrease of the number of known fields.

- All FPG solutions provide energy gains over four times with respect to SRAM. This refers to the factor that the maximum recorded percentage of static power in FPG-MRAM is lower than that recorded when SRAM is adopted. This can be explained by the significant reduction of static power in FPG-MRAM when the memory is in idle state and that the memories are at OFF state most of the time. The recorded data shows that the MRAMs are OFF more than 75% of the time. It is worthy to indicate that the static power dissipated in FPG-MRAM in the idle state is highly less than that of SRAM. The wake-up feature in the FPG policy has not added a significant overhead as the percentage of wake-up consumed power is less than 10%. The dissipated power while reading is important since it dominates over the static power and wake-up power. Its percentage approaches to 65% of the total consumed power. However, the SRAM power consumption while reading is higher than the introduced types of MRAM.

- OCPG policy cannot outperform SRAM case with any type of MRAM as the percentage of static power dissipation of OCPG-MRAM is higher than the percentage of static power dissipation of SRAM.

- The comparison in terms of power consumption between the three types shows the following:
  1. Type I outperforms Type II in the case of 256 bits as with 256 bits the gain of read operations does not compensate the increase of the static power of the Type II switch.
  2. Type II is more efficient with respect to Type I when 128 accesses are possible
  3. Type III outperforms Type I and Type II in any case as it takes full benefit of the reduced read energy that does not impact the PG switch overhead. Moreover, Type III can also benefit from the bitwidth granularity and efficiently exploit 32, 64, and 128-bit cases opportunities.
- The comparison of the adopted NI with the baseline NI shows that the area and power of the proposed NI, present overheads of 14.3% and 12.6%, respectively. If we consider the whole NoC, the area and power overheads are 6.1% and 5.2%, respectively.

As a conclusion, important energy savings can be achieved when the application fits with the MBC context, namely when fully power-gated NVMs can take benefit of limited activity rates. The obtained results show that such an approach can significantly modify the power budget breakdown between NoC and memory. This allows to increase the number of memory modules and so to target large-scale applications.

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

IEEE TRANSACTIONS ON VERY LARGE SCALE INTEGRATION (VLSI) SYSTEMS 1

Networked Power-Gated MRAMs for Memory-Based Computing

Jean-Philippe Diguet, *Member, IEEE*, Naoya Onizawa, *Member, IEEE*, Mostafa Rizk, Johanna Sepulveda, *Member, IEEE*, Amer Baghdadi, *Senior Member, IEEE*, and Takahiro Hanyu, *Senior Member, IEEE*

***Abstract*— Emerging nonvolatile memory technologies open new perspectives for original computing architectures. In this paper, we propose a new type of flexible and energy-efficient architecture that relies on power-gated distributed magnetoresistive random access memory (MRAM). The proposed architecture uses a network-on-chip (NoC) to interconnect MRAM-based clusters, processing elements, and managers. The NoC distributes application-specific commands to MRAM devices by means of packets. Configurable network interfaces allow to transform MRAM devices into smart units able to respond to incoming commands. In this context, three types of MRAM designs are proposed with different power-gating policies and granularities. A relevant database search engine case study is considered to illustrate the benefits of this proposed architecture. It is implemented with a sparse-neural-network approach and simulated in SystemC with different scenarios including hundreds of database queries. Hardware designs and accurate power estimations have been conducted. The obtained results demonstrate important power reduction with database hit rates of about 94%. Targeting 65-nm technology, energy savings reach 87% when compared with an static random access memory-based implementation. Moreover, a new asymmetric read/write MRAM type provides from 39% to 50% energy reduction with respect to the other fixed-granularity models. This results in a low-power, highly scalable, and configurable implementation of memory-based computing.**

***Index Terms*— Database, magnetoresistive random access memory (MRAM), network-on-chip (NoC), power gating (PG), sparse neural networks (SNNs).**

I. Introduction

COMPUTING resources are continuously increasing with technology progress, yet the use of the computing capabilities is facing major limitations in terms of power dissipation, off-chip memory access time, and real available parallelism. These limitations lead to a partial use of the offered integration capabilities and increase the gap between accessible technology nodes and available design methodologies. Nevertheless, the evolution in memory design and on-chip memory integration, particularly the emerging nonvolatile memory (NVM) technologies, opens new design perspectives to balance the underuse of available hardware resources.

In fact, besides memory access time reduction due to on-chip integration, the silicon density offered by recent technology nodes can be efficiently used, and significant reductions in power consumption can be achieved as long as the leakage power can be controlled and the memory bandwidth exploited. Such a configuration is made possible by applying memory-based computing (MBC) while considering a network-on-chip (NoC) to broadcast requests to NVM devices and full power gating (PG) to switch-OFF unused memories.

Based on these observations, we first propose a novel NoC-MBC (NMBC) architecture, which relies on an NoC-based multicore architecture that allows to take full benefit from PG capabilities of distributed MRAM. We introduce a new computing paradigm, where the packets of an NoC are used as commands executed by configurable network interfaces (NI) to access multiple distributed memory blocks in parallel. These memory blocks are technologically independent of the NoC and the processing units. Furthermore, we introduce different implementations of spin-transfer-torque magnetoresistive random access memory (STT-MRAM) with cell and peripheral PG capabilities. Finally, the effectiveness of the proposed approach is demonstrated through a relevant case study of a database search application implemented with a neuromorphic architecture based on a sparse neural network (SNN) [1].

This paper presents five new contributions that improve significantly the performances compared with our previous work [2]: 1) an original type of power-gated MRAM memory called *Type III*, which is based on an asymmetric read/write scheme and allows for significant power reductions with respect to previous solutions; 2) rather than a single manager, multiple manager modules can be working concurrently; 3) the processing elements (PEs) are not statically but dynamically allocated; 4) PEs are implemented as dedicated hardware components, and they are included in the global power budget;

Manuscript received September 2, 2017; revised February 9, 2018 and May 4, 2018; accepted June 26, 2018. This work was supported in part by JSPS KAKENHI, Japan, under Grant JP16H06300, in part by the Region Bretagne CyAM Project, France, and in part by the MFC Project of Future and Rupture IMT Program, France. *(Corresponding author: Jean-Philippe Diguet.)*

J.-P. Diguet is with CNRS, Lab-STICC, 56100 Lorient, France (e-mail: jean-philippe.diguet@univ-ubs.fr).

N. Onizawa and T. Hanyu are with Tohoku University, Sendai 980-8577, Japan.

M. Rizk is with Lab-STICC, STICC, 56100 Lorient, France, and also with Lebanese International University, Beirut, Lebanon.

J. Sepulveda is with the Technical University of Munich, 80333 Munich, Germany.

A. Baghdadi is with Lab-STICC, 56100 Lorient, France, and also with IMT Atlantique, 29100 Brest, France.

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TVLSI.2018.2856458

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

and 5) design and synthesis of all modules, including MRAM devices, are done using 65-nm technology node.

The rest of this paper is organized as follows. First, we present the proposed NMBC architecture model and the considered application case study of an SNN-based database search engine in Section III. Then, Section IV introduces the MRAM design models, including the proposed cell and peripheral PG policies and the three types of PG granularities. Section V provides the detailed evaluation setup in terms of selected database for the application case study, NoC implementation, and SystemC simulation of the proposed NMBC architecture. Section VII summarizes and discusses the obtained results before concluding this paper in Section VIII.

II. Related Work

A. Memory-Based Computing

MBC can have several meanings. The first one is a method to solve the memory-wall challenge, for applications dominated by data transfers, by migrating computing within the memory. A first implementation of a full adder was introduced in [3], and a summary of a research decade on this topic is presented in [4]. The solution consists in executing some greedy operations with logic implemented in the memory device in order to reduce the bandwidth demand and latency that limit the use of static random access memory (SRAM) and DRAM. The logic-in-memory concept allows to implement logic function (F) to apply transformations and data transfer in the memory as shown in Fig. 1(b). This solution is especially efficient with NVM technologies, such as spin–orbit torque magnetic random access memory or domain wall motion memory. Fan *et al.* [5] compare these technologies with an Advanced Encryption Standard encryption application, which is a typical in-place computing scheme that can take full benefit from this approach. The idea of moving computing close to the memory reduces latency and increases the bandwidth compare to solution based on SRAM or DRAM. Memristive technologies are also candidate for programmable approaches. This is presented with programmable bit-level logical function in memories in [6], but PG is not considered.

Our approach aims also to distribute and move computing resources closer to memory but with a different approach. Actually, we consider distributed memories to increase the bandwidth. This solution is made possible jointly by the increase of on-chip memory density and by the emerging manycore architectures based on NoC. For cost and efficiency reasons, we also want to reuse existing (or emerging) NVM memories without modification except the implementation of efficient PG techniques. So our solution relies on the enhancement of NI, as shown in Fig. 1(a).

The second meaning of MBC is the idea that computing can be replaced by memory. Replacing logic by memories is efficient in the case of computation redundancy. Such patterns can be extracted from signal processing algorithms that allow partial preprocessing as introduced in [7]. Paul and Bhunia [8] consider dense memory, based on molecular crossbars, to implement logic functions in multi-input-multi-output lookup table (LUT) in a memory array. In this approach,

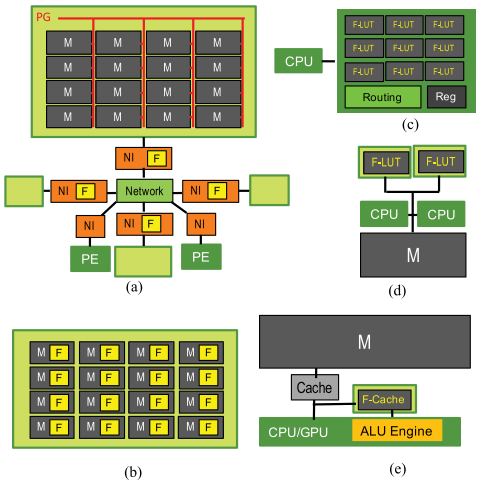


Fig. 1. Types of MBC. (a) NoC-based with separated PG NVM (this paper). (b) Logic in memory. (c) Processing mapped in LUT (fine grain). (d) Processing mapped in LUT (coarse grain). (e) AMM caching frequent computer results.

a partitioning of the application graph is applied to map computing elements on memory arrays. In [9], the idea is extended to coarse-grain implementations. Both cases are shown in Fig. 1(c) and (d), respectively. It is worth noting that the two ideas can be combined if in-memory logic functions are implemented as LUTs.

Our solution also aims to replace computing by memorylike in Fig. 1(c), and however, we do not implement logic in the memories but in the NI to turn power-gated memories into content-addressable memories. It is also important to note that our NMBC concept can be generalized and additional functionality (e.g., encryption, max, sum, sigmoid, and so on) can be implemented in the NIs.

The third meaning is an extension of the second one and is based on the principle of memory cache applied to the results of frequent computations with identical inputs. It is shown in Fig. 1(e). If the same computation with the same operands occurs, then the result is available from the memory without repeating the computation. Such methods have shown significant energy improvement in application domains with important opportunities of result reuse. This technique has been implemented with associative memristive memories (AMMs) in the context of heterogeneous reconfigurable multicore [9] and GPGPU architectures [10]. Our approach considers the replacement of results stored in the memory to take advantage of dense, low-power memories. In practice, all memories are not addressed at the same time leading to inherently partial resource usage in time. Therefore, applying PG techniques to unused resources can offer important power savings. In this context, emergent NVM technologies such as MRAM with low-leakage characteristics and PG capabilities represent a great opportunity. They allow

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

to design energy-efficient architectures where unused memory resources can be switched OFF. In other words, increasing on-chip resources with low-power behaviors contributes to the global architecture and energy efficiency. Moreover, such architectures can now take real advantage of more than one decade of research and achievements in the NoC design. NoC allows configurable accesses to grid of memories as well as the implementation of preprocessing/postprocessing steps in NI [11].

Data-centers-related applications represent a typical case where the energy efficiency constraints lead to the rise of heterogeneous architectures [12]. It covers applications dominated by memory resources, such as database accesses, search engines, data analytics, and so on. In this context, neuromorphic architectures can play a major role. It is worth noting here that this class of applications usually implies much more reading than writing accesses in the operational phase after the learning stage. Different approaches have been explored in the literature. Pham *et al.* [13] present an efficient run-time configurable coarse-grain reconfigurable architecture for convolutional NNs. A smart direct memory access and dedicated switches are used to feed computing nodes with data from an off-chip memory. The architecture is computation intensive and does not fit with MBC except if produced data exhibit reuse. The true north architecture [14] is different and designed for spiking NNs. It is based on an NoC to distribute synapses states to local computing cells. The chip is based on a dedicated architecture that implements 5 Mb of on-chip memory but with no PG technique. Logic-in-memory is also explored for NN applications, and Wang *et al.* [15] implement the different NN function including Sigmoid in an NVM. Finally, an SNNs dedicated architecture is presented in [16]. It is designed using a logic-in-memory approach and MTJ devices with bit-level interconnects. It is highly optimized but at the cost of a very specific design approach. An NoC-based architecture for SNN is presented in [17], yet it is a dedicated architecture and this paper is mainly devoted to the NoC topology and the memory optimization.

Finally, one can think about the concept of active memory, which also enhances memory with new functionality. This is the capacity to manage high-level transactions requests with a local processor executing transfer operations [18]. This type of solution leads to specific designs, where logic and memory are tightly coupled.

We also consider the case of neuromorphic computation as a typical case of data reuse that can take a great benefit of an MBC model. But, as previously mentioned, we consider MRAM with efficient PG and implement additional logic resources in the NI and not in the memories. Our objective is to exploit the bandwidth and the flexibility of packet-switching NoC architecture to efficiently access the target data distributed in arrays of memories that can be switched ON when necessary.

B. Power-Gated MRAM Devices

Magnetic tunnel junction (MTJ) devices [19] represent a main class of emerging nonvolatile technologies for low-power memory circuits. Recently, several MRAM designs have been presented [20]–[22] that exhibit significantly low-leakage currents realized by PG, thanks to the nonvolatility feature. These MRAM designs are designed using relatively large device technologies, such as 90 nm, yet the MTJ technology allow scaling down to 11 nm as for the CMOS devices [23]. The performance of MRAM devices in such recent technology nodes is also estimated in [24] and [25].

MRAM devices are designed using different MRAM cells, such as 1T-1MTJ, 2T-2MTJ, and 4T-2MTJ. This corresponds to performance tradeoffs between area efficiency and read time. For example, the 1T-1MTJ MRAM exhibits the highest area efficiency with a relatively slow read time, while the 4T-2MTJ MRAM realizes faster read time with lower area efficiency. In terms of PG, a simple strategy that shutdowns the power of the whole MRAM device at the idle state is generally used. In the 4T-2MTJ MRAM [21], a cell-level PG technique is used that activates only a row at the read/write state, leading to low-leakage currents in the cells without any additional PG transistor. We use this 4T-2MTJ MRAM model in this paper, and more details are provided in Section IV.

However, advanced PG policies and granularities, including peripheral circuits (e.g., row and column decoders), have not been presented or discussed in the available literature. It is worth noting that conventional PG techniques that can be applied to processing units and NoCs are out of the scope of this paper. These methods are independent and complementary to our proposed approach and can be efficiently combined.

III. NMBC Flexible Architecture

A. Architecture Principle

The proposed architecture model is presented in Fig. 2(a). It is based on an NoC that interconnects three types of IP blocks: memory clusters, PEs, and managers. In the proposed approach, we consider NoC packets as commands initiated by managers, as shown in Fig. 2(b). Memories are distributed in M_C clusters, and each cluster has a unique NI and is composed of M_B memory blocks. The managers are in charge of a set of requests to process and send packets to memory clusters in a unicast, multicast, or broadcast way. They also apply a processor selection for postprocessing and can therefore perform load balancing. All the managers can work in parallel. The decoding of instructions is carried out by smart NIs that implement additional logic elements. NIs are in particular aware of a data mapping and manage the communication of results to processors. NIs also implement some specific bit-level operations such as bit selection that will be detailed in Section V-B3. NIs can also provide managers with monitoring data about bandwidth and processor usage, and this type of information is sent as monitoring packets to managers.

B. Application Case Study of SNN-Based Database

1) SNN Concept: As introduced before, database search engine is one of our target application domains. Hereafter, we explain the use of NMBC for database search engine implementation based on the SNN algorithm [1]. SNN is an NN model inspired by information theory and error correcting codes. It is especially efficient for implementing associative memories, so can be used to process database queries,

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

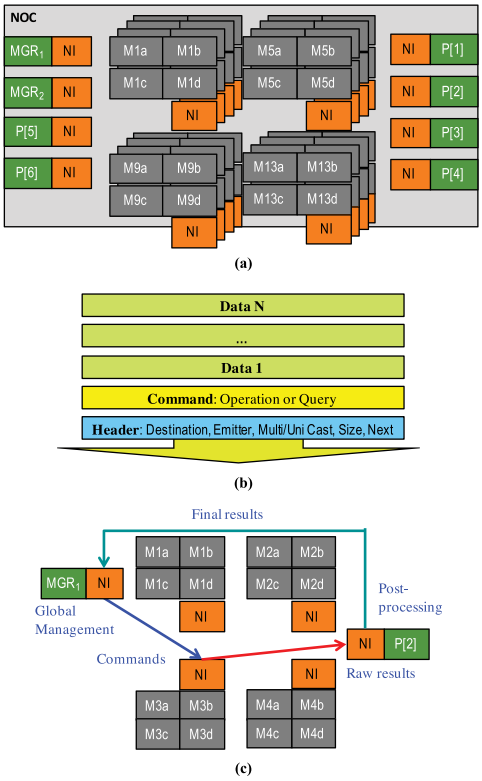


Fig. 2. NMBC architecture example with $M_C = 16$ memory clusters composed of $M_B = 4$ memory modules (Mij), 6 PEs ($P[x]$), and 2 managers (MGR_i). (a) Architecture. (b) Packet-based command. (c) Example of execution steps.

as shown in [16]. SNN relies on two phases: message learning and information retrieval from partial knowledge. It is worth noting that the learning phase consists simply in writing each new record "one time" at the right place in the memories. Therefore, this phase presents a linear complexity with the database size for any devised network architecture. This is fundamentally different from the greedy learning phase of deep learning approaches. So, the write operations cost is negligible compared with read operations cost, which constitutes the major challenge in database search applications. Basically, a record is a clique that fully links neurons that belong to different clusters. The principle can be compared with low-density parity-check codes. If we consider the database application, a record M is composed of F fields (a field $f(i)$ per cluster). The learning phase is the mapping function that associates each field of each record to a single neuron in each cluster. Thus, each field $f(i)$ is coded with $k_i = \lceil \log_2(l_i) \rceil$ bits, where l_i is the number of neurons in cluster $f(i)$. So, a record is a binary sequence of $\sum_{i=1}^{F} k_i$ bits and is

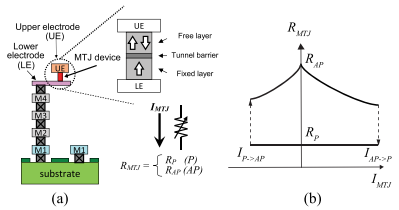


Fig. 3. STT MTJ. (a) Structure. (b) R–I characteristic.

associated with a clique of neurons, and each clique materializes a code word. The query or retrieval of missing data consists in the selection of the best candidate neuron for each cluster that corresponds to an unknown field by means of known neurons is other fields. This choice is made according to a ranking method, which is a winner-take-all algorithm in the original proposal. All clique edges are specified with binary weights such as $C_{XY}(ij) = 1$ which means that at least one clique contains an edge from neuron i in cluster X to neuron j in cluster Y. The score of each neuron, in the target cluster, equals to the number of clique edges that exist between this neuron and known neurons in other clusters. In the original algorithm, additional iterations can be executed if multiple winners have the same score. However, in the case of database search engine, we will consider multiple answers to a given query. Other algorithmic options are possible but are out of the scope of this paper.

2) SNN on NMBC Architecture: The very dominant cost in SNN implementation is memory. Actually, it is necessary to store all the connections between every pair of clusters. We call $m_{i,j}$ the connection memory between clusters i and j. We demonstrated in another application domain that it is possible, by means of a dedicated memory mapping, to use a single memory for implementing $m_{i,j}$ and $m_{j,i}$ [17]. However, in this paper, we consider distinct $m_{i,j}$ and $m_{j,i}$ for two reasons. First, the NMBC architecture must be programmable to match with different configurations so the size of memory cannot fit exactly with the size of the cluster. Second, our objective is to take advantage of future MRAM memory with a low static power consumption so that unused memories do not impact significantly the power consumption. Therefore, memories are used to store the $F \cdot (F-1)/2$ connection memories. Regarding the number of fields and the size of each cluster, the mapping of $m_{i,j}$ on the set of distributed memories results from an optimization process to maximize the use of each memory. The managers send queries to memories by means of packets where known and unknown fields are specified. Then, the processors collect the memories answers and apply a simple winner-take-all algorithm to select records to be send back to managers.

IV. MRAM Model

A. 4T-2MTJ MRAM Cell

MRAM circuits are designed using STT MTJ devices [19] with CMOS transistors. Fig. 3(a) shows an STT MTJ device stacked over a CMOS layer. It mainly consists of three layers: free layer, tunnel barrier, and fixed layer. The resistance of

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

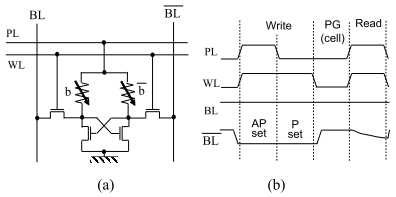


Fig. 4. 4T-2MTJ MRAM cell with PG. (a) Circuit diagram. (b) Timing diagram.

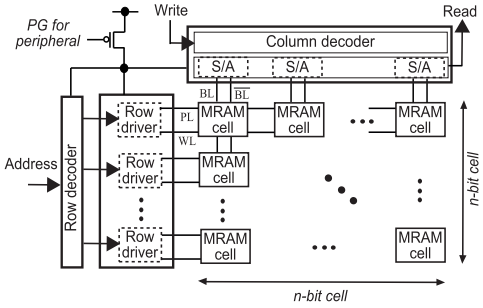


Fig. 5. MRAM with cell PG and selective peripheral PG.

the MTJ device depends on a spin direction of the free layer. If the spin direction of the fixed layer is the same as that of the free layer, the MTJ is at the parallel state R_P (low resistance). Otherwise, it is the antiparallel state R_{AP} (high resistance). The resistance state can be changed using a current signal I_{MTJ}, as shown in Fig. 3(b).

Fig. 4(a) shows a 4T-2MTJ MRAM cell with PG. The MRAM designed based on [21] consists of four transistors and two MTJ devices controlled by the power line (PL), the word line (WL), and the bit lines (BL, $\overline{\text{BL}}$). A data bit is represented by complementary signals, b and $\overline{b}$, stored in the two MTJ devices. Fig. 4 (b) shows a timing diagram of the MRAM cell. In the write operation, a data bit is stored to two MTJ devices for two steps. First, a current signal is generated from PL to one of the bit lines (BL, $\overline{\text{BL}}$) depending on a data bit stored. In this example, $\overline{b}$ is written because $\overline{\text{BL}}$ is low. Then, b is written using a current signal from BL to PL, where the resistance is changed from R_P to R_{AP}. In the read operation, one of BL and $\overline{\text{BL}}$ is low depending on a stored bit. In the idle state (PG), both PL and WL are grounded, which totally eliminates the static power in the 4T-2MTJ cell as no leakage path exists in this case.

B. Power-Gating Policy

Fig. 5 shows an MRAM with cell PG and selective peripheral PG. It consists of the MRAM-cell array with the peripheral circuits: row decoder, row drivers, column decoder, and sense amplifiers (S/A). In the MRAM-cell array, only MRAM cells in a row are active in read/write operation, while the other MRAM cells are power-gated using the row drivers. The cell PG significantly reduces the leakage current of the cell array in comparison with that of an SRAM cell array. The peripheral circuits can be power-gated using the pMOS transistor when the MRAM is at the idle state. For the PG policy, two different policies of PG are exploited as summarized in Table I. The first policy is *only cell PG (OCPG)*, where the MRAM cells are power-gated at the idle state and the peripheral circuits are always at the active state. Therefore, the OCPG-MRAM requires no wake-up operation with negligible leakage current in the MRAM cell array. The second policy is *full PG (FPG)*, where the whole MRAM is power-gated at the idle state. It reduces the leakage current in comparison with the OCPG-MRAM while the wake-up operation is required for the peripheral circuits. The wake-up operation causes extra energy dissipation and delay time. Hence, there is a tradeoff between the OCPG-MRAM and FPG-MRAM. In Section V, the performance of the two different MRAMs with the two different PG policies is exploited for system-level simulations of NMBA.

TABLE I
PG POLICY IN MRAMS

Policy	Only cell power-gating (OCPG)	Full power-gating (FPG)
PG in cell array	Yes	Yes
PG in peripheral	No	Yes
Wake-up operation	No	Yes

C. Power-Gating Granularity

For the PG granularity, three different 256-bit × 256-word MRAMs (Type I, Type II, and Type III) are designed, as shown in Fig. 6. The detailed specifications and performance are summarized in Table II. The MRAM circuits are designed and evaluated using a TSMC 65-nm CMOS process and an MTJ model [26] HSPICE. First, let us explain the differences between Type I and Type II. Type I uses a PG transistor for a 256×256 MRAM. Type II uses four PG transistors for four 128×128 MRAM subblocks. The widths of the PG transistors are determined so as to achieve similar write and read times. Compared with Type I, the advantage of Type II is lower power dissipation of reading/writing if the read/write bitwidths are smaller than or equal to 128 bits. In fact, read and write powers are reduced because parastic capacitances of BLs and WLs are reduced, as these capacitances depend on the sizes of the MRAM subblocks. In contrast, the disadvantage of Type II is larger leakage current as the normalized total PG switch size required is double.

We observe that in MBC applications in general and search applications in particular, data are mainly read and rarely written. So we introduce Type III in order to meet both advantages of Types I and II. Type III is designed with the restriction that the maximum write bitwidths is 32 bits, whereas the maximum read bitwidth is 256 bits as for Types I and II. Considering the asymmetry of read and write occurrences in the target application domain, this restriction does not significantly affect the execution time. As only 32 bits are written in parallel in Type III, the normalized total PG switch size required is 1/4 of the Type II, leading to a lower leakage current. If write capability is limited to 32 bits, Type III offers on

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

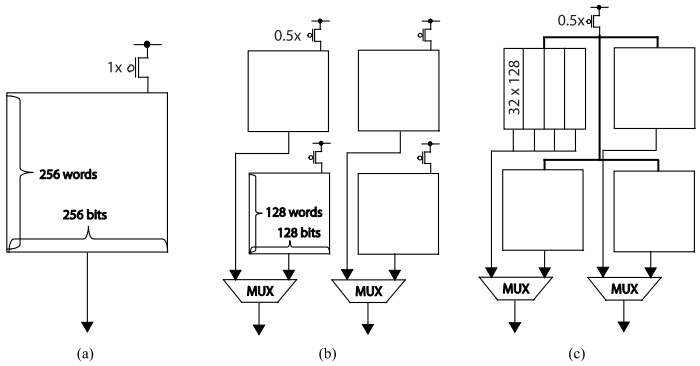


Fig. 6. PG types and granularity in 256-bit × 256-word MRAMs. (a) Type I. (b) Type II. (c) Type III.

TABLE II

PERFORMANCES OF 256 × 256 MRAMS USING THE TSMC 65-nm CMOS PROCESS AND THE MTJ MODEL [26]

	Type I	Type II	Type III
Write bit width	256	128, 256	32
Read bit width	256	128, 256	32, 64, 128, 256
# of PG switches	1	4	1
Normalized PG switch size	1	1	0.5
Normalized total PG switch size	1	2	0.5
Read power/bit@100 MHz [mW]	1.30	1.16	1.03
Write power/bit@ 100 MHz [mW]	2.79	2.48	2.38
Static power w/o PG [mW]	51.3	62.2	43.2
Static power w/ PG [mW]	0.679	0.980	0.300
Wakeup energy [nJ]	0.934	1.013	0.648
Wakeup time [ns]	0.072	0.0045	0.072

TABLE III

NUMBER OF NEURONS PER CLUSTER IN YEAST DATABASE

Cluster	0	1	2	3	4	5	6	7	8	9	10
Neurons	39	39	81	79	53	78	2	3	48	68	10

the other hand opportunities to extend read configurations to 32, 64, 128, or 256 bits and so to adapt the read power according to the application demand, as shown in Table II.

V. CASE STUDY

A. Yeast Database Case

As a representative example, we consider the Yeast database (cellular localization in proteins) from the UCI Machine Learning Repository [27]. The original database is specified with 10 fields that we transform into 11 fields by splitting the first oversized field so that we get a more homogenized network. Table III shows the number of connections per field, i.e., neurons per cluster. The number of available memories is fixed by the architecture, yet the number of connection memories $m_{i,j}$ depends on the application. Therefore, the first design step is the memory mapping that results from an optimization process, which is out of the scope of this paper. In our case study, we map $10 \times 11 = 110$ connection memories in six 256×256-bit memory modules. Fig. 7 details the proposed mapping for one of these six memories (M5, as referenced in Fig. 8).

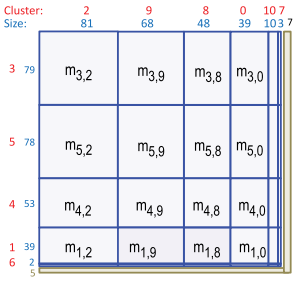


Fig. 7. Yeast database: $m_{i,j}$ mapping in memory M5.

B. NoC Implementation

Fig. 8 presents the structure of the NoC and we consider to demonstrate the NMBC concept. The NoC is a 4×4 mesh-based network, which interconnects 18 IP cores (6 memory clusters composed of single modules, 10 PEs, and 2 managers). It means $M_C = 6$ and $M_B = 1$ if we refer to the general model described in Fig. 2 The NoC employs the wormhole packet-switching mode, the deterministic XY routing algorithm, and a flow control policy without virtual channels. The implemented routers have one buffer of 3 flits per input port and use distributed arbitration logic (one arbiter per port). The back-end part of the NI is typical and includes a packet maker/un-maker and a priority manager to synchronize packet transmission and reception.

In addition to usual interface logic, the frontend of an NI associated with a memory module implements the addressing and bit selector modules. This choice is important since it allows to remain compliant with any existing memory and to be independent of NoC parameters (topology, router buffer depth, and routing policy). In a fully flexible NMBC architecture, an NI includes an area of reconfigurable hardware that can be configured to implement application-specific functions like addressing and bit selector modules in our case study.

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

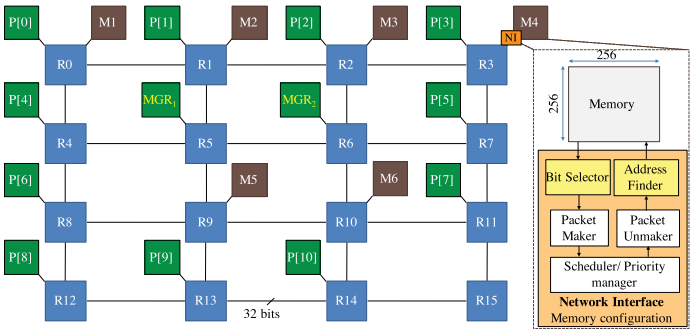


Fig. 8. Structure of the used NoC.

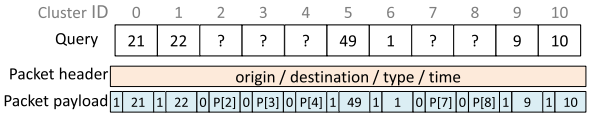


Fig. 9. Structure of the command packet.

The functionality of each IP core and the structure of the additional elements are detailed hereafter.

1) Manager: Upon loading new query (e.g., from a host machine), the manager associates missing fields to clusters with unknown neurons and well-specified fields to identified neurons in other clusters. Consequently, it transmits to all memory modules the command to send to allocated PEs, the available connection information between known neurons in well-defined clusters and all neurons of the undefined clusters. The payload of sent packets is divided into segments, which are relative to definite clusters.

In the adopted application, 11 different clusters exist. Each segment has two parts. The first is one-bit flag, which is used to indicate the status of the corresponding cluster. Respectively, "0" or "1" are placed to specify if the cluster neuron is missing or known. The content of the second part depends on the cluster status. If the cluster has known neuron (i.e., known field of the query), this part contains the initial address of the neuron, which is available in the loaded query, whereas if the cluster's neuron is missing, it will be filled by the address of the PE that is in charge of processing the winner-take-all for this cluster. Fig. 9 presents an example of the sent packet. The loaded query in this example has five clusters with missing neurons (2, 3, 4, 7, and 8). The flags corresponding to these clusters is set to "0." Also, the addresses of the processing modules (P[2], P[3], P[4], P[7], and P[8]) are specified in the second part of each segment. The choice of the PE is made dynamically with a fairness load-balancing policy, whereas for the other clusters (0, 1, 5, 6, 9, and 10), the flags are adjusted to "1" and the second part of each segment is loaded by the values initially available in the query. By arranging this data into 32-bit flits, the sent packet will be composed of four flits only. On the other hand, the manager is first responsible to collect the results corresponding to each cluster from its dedicated processing module. Then, it decides to iterate in order to enhance results or to simply deliver the achieved ones.

2) Processing Elements (32 bits in Parallel): All PEs apply the winner-take-all computational principle in order to find the neurons with the highest score. The score of each neuron is incremented by the number of available connections with all known neurons, which belong to the other clusters. Each PE is assigned to one cluster with a missing field, the assignment is made dynamically by the managers, and the PE is active until the solutions are found. At last, each PE informs by means of packets and the managers by the identity(ies) of the winner neuron(s).

In a fully flexible NMBC architecture, a PE could be a small-size embedded processor or a reconfigurable hardware. In this paper, we have designed a specific PE in VHDL so that we can estimate the power consumption and the area. Each PE can process 32 bits (32 neurons) in parallel. It takes then three cycles to update in the local memory the score of the cluster neurons and the best score neuron. The PE receives the connection information related to their cluster from all memory modules. The total number of cycles depends on the number and size of known/unknown fields that determine the number of packets to be processed.

3) Memory Module: The memory modules store the connection information between the neurons of all clusters at the learning phase. For the adopted case study, six 256×256-bits memory modules are used to accommodate the contents of 110 connection memories. Each module includes a memory that returns the data allocated at its specified address. Each memory module should respond to the reading request and deliver all connection information between all known neurons and missing ones to the specified processing modules.

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

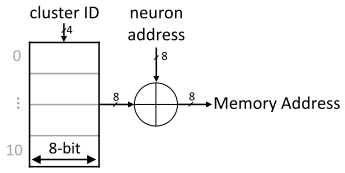


Fig. 10. Architecture of the address finder.

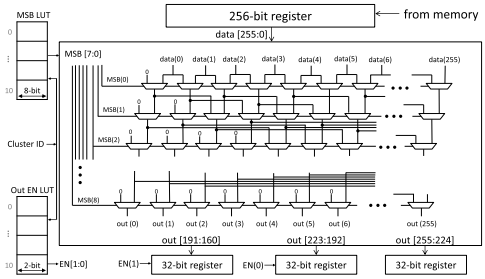


Fig. 11. Architecture of the bit selector.

The manager is not aware of the local mapping of connection memories, and the model is independent of the type of memories. Therefore, the matching between requested connections and physical addresses is performed by NIs. Since the command packet arriving from the manager does not include implicitly the specifications of the data, which is required to be retrieved (address, width, and starting bit), new tasks such as addressing and arranging the output bits into flits are required. Independently of memory and NoC architectures, these new services are implemented in separate elements in the frontend of the NI associated with each memory module, as shown in Fig. 8. Their architectures and functionalities are described in the following.

a) Address finder: It is a simple hardware element, presented in Fig. 10, which determines the addresses that must be used to access the right connection memories. These addresses are based on the ID of unknown filed and the value of known fields indicated in the command packet. It is composed of an adder and a configurable LUT, which includes the starting addresses of all clusters in the associated memory. When a command packet arrives, the packet unmaker disassembles the packet and delivers the payload to the address finder. The latter identifies the clusters with known neurons according to their flag bits. For each of those, its identity is used directly as the LUT index (address) to retrieve the starting address of the cluster. This address is considered as an offset and is used by the adder to determine the address value to be delivered to the memory. For example, in Fig. 9, cluster 5 has known neuron address 49. The starting address of cluster 5 in memory M_5 is 79, as shown in Fig. 7. Therefore, the address finder will deliver the value of $128 = 79 + 49$ to the memory.

b) Bit selector: Fig. 11 shows the architecture of the bit selector. It is composed of one 256-bit barrel shifter, one 256-bit register to store input data from memory, three 32-bit registers to store output data according to a 32-bit flit size, and two LUTs that deliver most significant bit (MSB) and control output data width. The first includes the MSB values corresponding to all clusters. The second controls the output data width. Both LUTs are controlled by the cluster IDs. When the new data are retrieved from the memory, the barrel shifter shifts, for each cluster, the data by $255 - \text{MSB}$ steps and then delivers the result to the three output registers. The registers are then loaded according to the enable signal delivered by the second LUT. The algorithm of the address finding and the bit selection algorithm is outlined in Algorithm 1.

Algorithm 1 Address Finding and Bit Selection

Require: k the number of clusters with known neurons,
m the number of clusters with missing neurons
for all $c \in \{0, ..., k-1\}$ **do**
$Address = offset_c + neuron_c$
$data = Memory[Address]$
for all $c \in \{1, ..., m-1\}$ **do**
$out = data << MSB_c$
$out0 = out[255 : 224]$
$out1 = En_c[0] \wedge out[223 : 192]$
$out2 = En_c[1] \wedge out[191 : 160]$
end for
end for

Finally, for all clusters with missing neurons, the retrieved connection information with known neurons is gathered according to the cluster identities. Then, each group which is relative to a definite cluster is sent consecutively to the packet maker. The latter assembles a packet that is then transmitted to the processing module dedicated to each cluster.

VI. Experimental Setup

In order to check the relevance of the proposed approach, the adopted NoC implementing MRAM memories has been described in the SystemC TLM model as a proof of concept. The devised model executes the database search engine application using the Yeast benchmarks. The model has been developed to mimic the exact behavior of the target architecture by designing hierarchical modules that work concurrently and intercommunicate via ports. In addition, the SytemC model is cycle accurate at the NoC level including NIs. To ensure accuracy, the timing features of memory reading response and wake ups are considered in the model. Furthermore, the execution time per paquet of winner-take-all computations is based on the HDL simulation and synthesis that provide real-time environment modeling.

Multiple simulations have been conducted by using different numbers of queries with various number of missing clusters. In this paper, we consider four cases with 4, 5, 6, and 7 missing fields out of 11. The index of missing fields has been chosen randomly. To evaluate the efficiency of the proposed MRAM designs, the results are compared with those obtained when the SRAM-based design is adopted. The SRAM on the isocapacity condition is used. In both SRAM and MRAM cases, the total memory size is 384 kb, where six 256×256 distributed memories are used. Both the models use identical NoC features, including packet sizes, routing algorithm, PEs features, mapping strategy, and a common clock set to 500 MHz. We consider the two PG policies defined in Section IV-B. FPG is implemented at the query level,

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

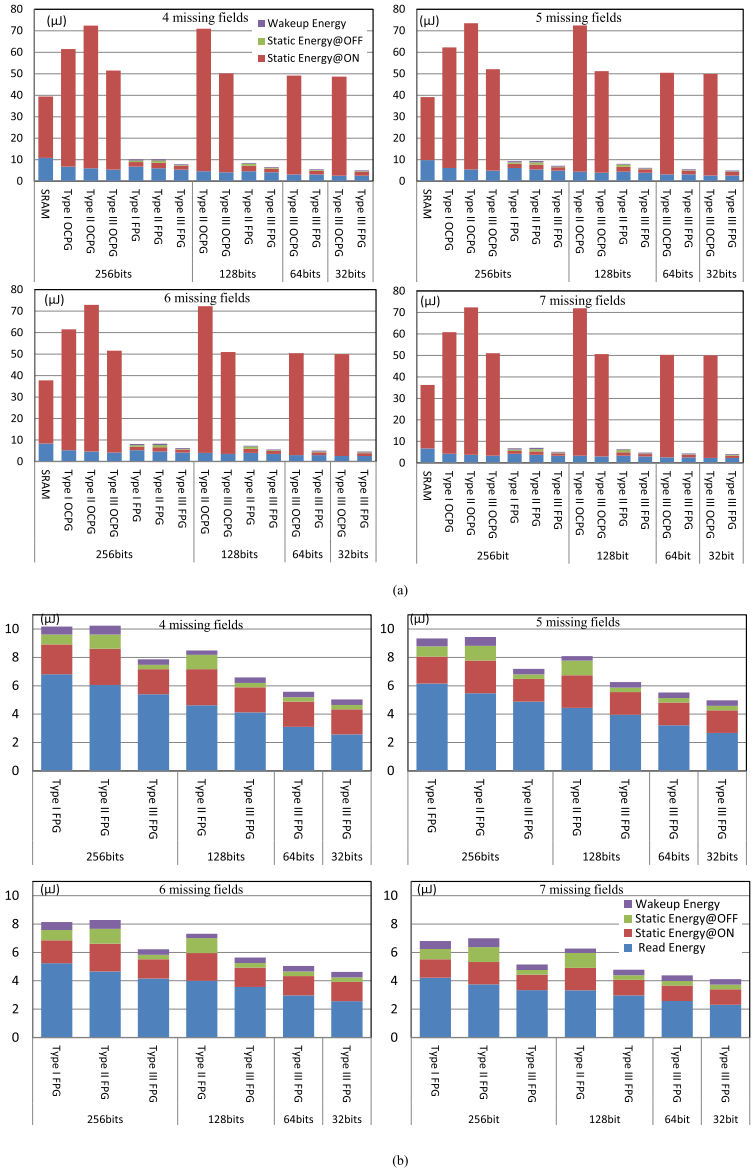


Fig. 12. Energy consumption for the different memory types and PG policies. (a) 2 Managers—600 requests and 4 configurations with 4, 5, 6, and 7 missing fields. (b) 2 Managers—600 requests and 4 configurations [zoomed on cases, where $E < 10$ μJ (all FPG policies)].

which means that the peripherals are switched ON when a packet is received and switched OFF when all the connection data, related to the command, are processed. We compare the three types of MRAM detailed in Section IV-C. In the case of Type II and Type III memories, readings are limited to 128, 64, or 32 bits when it is possible, namely when all the requested data are located in one of the 4 128×128-bit memory blocks.

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

TABLE IV
HIT RATE

Number of missing fields	4 fields	5 fields	6 fields	7 fields
1 or 2 managers	94.83%	94.60%	94.22%	94.19%

TABLE V
INJECTION RATE (FLITS/CLOCK CYCLE)

Number of missing fields	4 fields	5 fields	6 fields	7 fields
1 manager	0.482	0.524	0.552	0.567
2 managers	0.875	0.944	0.995	1.026

TABLE VI
PERCENTAGE OF STATIC POWER FROM TOTAL POWER CONSUMED IN THE CASE OF SEVEN MISSING FIELDS AND TWO MANAGERS

Memory Type	256-bit	128-bit	64-bit	32-bit
SRAM	81.51%	-	-	-
Type I OCPG	93.07%	-	-	-
Type II OCPG	94.82%	95.37%	-	-
Type III OCPG	93.45%	94.13%	94.88%	95.38%
Type I FPG	29.83%	-	-	-
Type II FPG	37.60%	41.92%	-	-
Type III FPG	27.53%	29.65%	32.36%	34.44%

TABLE VII
TIME PERCENTAGE OF THE ON STATE

Number of missing fields	4 fields	5 fields	6 fields	7 fields
1 manager	12.69%	11.27%	12.42%	7.61%
2 managers	23.03%	20.32%	17.21%	13.77%

VII. RESULTS AND ANALYSIS

A. Functional Simulation

The data delivered by the manager is compared with the reference data from the Yeast database for the four cases with 600 random request experiments. As illustrated in Table IV, the obtained results meet our expectations with hit rates over 94% when adopting either one or two managers. The results show a slight degradation in the hit rate when increasing the number of missing fields. This is expected, and it is due to the fact that the ambiguity ratio increases with the decrease of the number of known fields. As indicated in Table V, the NoC injection rate doubles with two managers. This translates in an increase of the NoC activity and the time percentage FPG memories are ON.

B. Power and Energy

1) Memories: Fig. 12 shows the energy dissipated by different types of memory modules in order to process 600 random queries while using the two proposed PG policies with 4, 5, 6, and 7 missing fields. The power estimation is based on the TSMC 65-nm CMOS process and the MTJ model given in Table II. From these results about the required energy budgets, we can draw the following important conclusions. In 65 nm, OCPG policy cannot outperform SRAM case with any type of MRAM. The static power dissipation of OCPG-MRAM reaches 95% of the total power dissipation, as shown in Table VI, whereas in SRAM, the highest recorded percentage of static power dissipation is 81.5%. Hence, the power saving due to the reduction of dissipation while reading is not noteworthy, although it is recorded as 1.6, 1.8, and 2 times less than that of SRAM for Type I, Type II, and Type III, respectively. On the other hand, the static power dissipated in OCPG-MRAM is approximately 1.9, 2.3, and 1.6 times more than that of SRAM for Type I, Type II, and Type III, respectively. This refers to the fact that in OCPG-MRAM, only cells are power-gated at the idle state while the peripheral circuits are always at the ON state. All FPG solutions provide energy gains over four times with respect to SRAM. This refers to different factors. First, the maximum recorded percentage of static power in FPG-MRAM is 42% from the total dissipated power, which is about half of that recorded when SRAM is adopted. This can be explained by the significant reduction of static power in FPG-MRAM when the memory is in idle state and that the memories are at OFF state most of the time.

TABLE VIII
PERCENTAGE OF POWER DISSIPATED WHILE WAKING UP IN FPG-MRAM FROM TOTAL POWER CONSUMED IN THE CASE OF SEVEN MISSING FIELDS AND TWO MANAGERS

Memory Type	256-bit	128-bit	64-bit	32-bit
Type I FPG	8.24%	-	-	-
Type II FPG	8.86%	4.94%	-	-
Type III FPG	7.55%	8.14%	8.88%	9.45%

TABLE IX
PERCENTAGE OF POWER DISSIPATED WHILE READING IN FPG-MRAM FROM TOTAL POWER CONSUMED IN THE CASE OF SEVEN MISSING FIELDS AND TWO MANAGERS

Memory Type	256-bit	128-bit	64-bit	32-bit
Type I FPG	61.93%	-	-	-
Type II FPG	53.54%	53.13%	-	-
Type III FPG	64.92%	62.22%	58.75%	56.11%

TABLE X
AVERAGE NOC POWER CONSUMPTION (MW)

Number of missing fields	4 fields	5 fields	6 fields	7 fields
NoC (routers)	54.46	61.04	65.21	71.04
NoC (network interfaces)	38.17	42.78	45.7	49.79
Total	92.63	103.82	110.91	120.83

Table VII indicates the time percentage of ON state. Table VII shows that in FPG policy, the MRAMs are OFF more than 75% of the time. It is worthy to indicate that the static power dissipated in FPG-MRAM in the idle state is approximately 39.5, 27.3, and 89.3 times less than that of SRAM for Type I, Type II, and Type III, respectively. This fact demonstrates the reduced dissipation of static energy of FPG-MRAM compared with that of SRAM. Second, the wake-up feature in the FPG policy has not added a significant overhead. The percentage of wake-up consumed power is less than 10%, as shown in Table VIII. Third, in FPG-MRAM, the dissipated power while reading is important since it dominates over the static power and wake-up power. Its percentage approaches to 65% of the total consumed power, as shown in Table IX. In addition, the SRAM power consumption while reading 256 bits is recorded about 1.6, 1.8, and 2.0 times more than that of MRAM for Type I, Type II, and Type III, respectively, whereas when 128 bits are retrieved, the SRAM reading power is 3.6 and 4.0 times more than that of MRAM for Type II and Type III, respectively. This ratio increases significantly for the

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

TABLE XI
GLOBAL AVERAGE POWER CONSUMPTION—WORST CASE OF SEVEN MISSING FIELDS

Memory Average Power (mW)						
Memories types		**Static**	**Dynamic**	**Total**	**Ratio vs SRAM case**	**Ratio vs Best case** (32b Type III FPG)
256 bits	SRAM	160.80	36.49	**197.29**	1.00	8.82
	Type I OCPG	307.80	22.93	330.73	1.68	14.78
	Type II OCPG	373.20	20.38	393.58	1.99	17.59
	Type III OCPG	259.20	18.18	277.38	1.41	12.40
	Type I FPG	11.04	25.97	37.02	0.19	1.65
	Type II FPG	14.31	23.75	38.06	0.19	1.70
	Type III FPG	7.71	20.29	28.00	0.14	1.25
128 bits	Type II OCPG	373.20	18.14	391.34	1.98	17.49
	Type III OCPG	259.20	16.17	275.37	1.40	12.31
	Type II FPG	14.31	19.82	34.13	0.17	1.53
	Type III FPG	7.71	18.29	26.00	0.13	1.16
64 bits	Type III OCPG	259.20	13.99	273.19	1.38	12.21
	Type III FPG	7.71	16.11	23.81	0.12	1.06
32 bits	Type III OCPG	259.20	12.55	271.75	1.38	12.14
	Type III FPG	7.71	14.67	**22.38**	0.11	1
Total Average Power (mW)						
NoC		18.2	102.63	120.83	0.35	0.71
PE		4.11	22.60	26.71	0.08	0.16
Total (NoC + PE + SRAM)				344.83	1	2.03
Total (NoC + PE + 32b Type III FPG)				169.91	0.49	1

cases of reading 64 and 32 bits from MRAM Type III to reach 8.0 and 16.1, respectively.

If we look into details [see Fig. 12(b)], we observe the significant benefit of the proposed MRAM Type III. First, we note that Type I outperforms Type II in the case of 256 bits (about 2%). With 256 bits, the gain of read operations does not compensate the increase of the static power of the Type II switch. The balance becomes positive for Type II with respect to Type I when 128 accesses are possible. The application offers opportunities that can be exploited (from 7.7% to 16.5% for 7 and 4 missing fields, respectively). Type III outperforms Type I and Type II in any case. Type III takes full benefit of the reduced read energy that does not impact the PG switch overhead, thanks to the asymmetric scheme. Moreover, Type III can also benefit from the bitwidth granularity and efficiently exploit 32, 64, and 128-bit cases opportunities. With the 32-bit Type III memory, the gains with respect to Type I go from 39.5% to 50.5% for 7 and 4 missing fields, respectively, while the energy budget reduction is about 87% when compared with SRAM. The gains with respect to Type I and Type II decrease with the activity rate, this is logical since the PG impact is reduced as well. However, it remains: 1) strongly significant with respect to Type I even with 7 unknown fields out of 11 and 2) extremely large with respect to SRAM in all cases. Therefore, we conclude that we obtain, as expected, important energy savings when the application fits with the MBC context, namely when fully power-gated NVMs can take benefit of limited activity rates.

2) NoC: The NoC and NI clock frequency is 500 MHz in nominal case operating conditions (25 °C, 1 V). The technology used is 65-nm LP LowK Std Vt High Density Tapless Library from Virage Logic Corporation. The area and power results are obtained with the Cadence Encounter RTL Compiler tool. The register transfer level (RTL) specification of the router and the NI is synthesized in the physical layout estimation mode. The total power was obtained by using the leakage and dynamic power of the NoC and NI components, relying on the switching activity traced by the SystemC simulation. Table X shows the static and dynamic power consumption of the used NoC. When compared with baseline NI, the area and power of the proposed NI, adopting the bit selector and address finder modules, present overheads of 14.3% and 12.6%, respectively. If we consider the whole NoC, the area and power overheads are 6.1% and 5.2%, respectively.

3) PEs: The power consumption of the dedicated 32-bit PE is estimated with the following methodology. The RTL specification of PE is synthesized with the TSMC 65-nm technology. The static and dynamic power consumption of a single PE running at 500 MHz are 0.32 and 3.79 mW, respectively. The activity of each PE is derived from the number of flits, and each PE is processing according to the simulation results. The power consumption of PEs is given in Table XI.

4) Global Power Budget: Finally, we analyze the power consumption of the whole NMBA considering a 65-nm technology node. The whole architecture includes memories, NoC (routers and NI with new modules), and PE (32-bit parallel version). In Table XI, we consider the global power dissipation for seven missing fields, which is the worst case for MRAM versus SRAM since the idle time is reduced. All SRAM- and MRAM-based architectures use the same NoC, the same SNN search algorithm, and the same PE and managers. In addition to the significant power reduction, a main result lies in the major change of the power distribution. With SRAM memories, the NoC represents 35%, while the memories account for more than the half of the total power (57%). If the best MRAM type and PG policy are considered (Type III FPG), the NoC share rises to 71% and the memory one drops down to 13%. So, the use of the right power-gated MRAM leads to a point where the NoC power consumption is higher than the memory one. This allows to increase the number of memory modules and so to target large-scale applications. In our case study, this number is limited to one 256×256-bit memory

module. However, the general model includes multiple memory modules per cluster, as shown in Fig. 2.

VIII. Conclusion

This paper shows the high interest of a new type of architecture designed for MBC that can fully benefit from distributed power-gated MRAMs. Memory blocks are efficiently accessed and controlled by means an NoC with smart NIs that are configured to implement application-specific processing or content-addressable memory schemes. Although memory accesses dominate in such applications, the distributed nature of the proposed memory architecture exhibits idle states so opportunities for PG. Considering the general architecture model, this paper explores different PG schemes as well as memory block and R/W granularities. This paper shows first that limiting the PG to MRAM cells does not reduce energy consumption with respect to the use of SRAM devices. Only a full PG including peripherals is efficient and provides impressive energy reduction, higher than 87% in the considered database search application case study. This significant energy reduction is obtained by considering the specificity of the application domain (database search applications and neuromorphic architectures), where the number of memory write accesses is extremely reduced compared with the number of read accesses. In this context, an original MRAM device model (Type III) is proposed with asymmetric write bitwidth (32 bits) and read bitwidth (32, 64, 128, and 256 bits). It allows to jointly limit the leakage of the power-gate switch and the read/write power consumption. This new approach brings 39%–50% more improvement compared with the symmetric fixed-granularity model. Furthermore, the validity of the proposed architecture is functionally demonstrated with an application that can take advantage of MBC, namely a database search application implemented with an SNN approach. SystemC simulations have been conducted targeting hundreds of database queries with a high hit ratio of about 94%. Finally, the results show that such an approach can significantly modify the power budget breakdown between NoC and memory. This allows to increase the number of memory modules and so to target large-scale applications. Future work should address the reduction of NoC power consumption and the use of programmable logic in NI to allow their dynamic configuration according to the application requirements.

References

[1] V. Gripon and C. Berrou, "Sparse neural networks with large learning diversity," *IEEE Trans. Neural Netw.*, vol. 22, no. 7, pp. 1087–1096, Jul. 2011.

[2] M. Rizk *et al.*, "NoC-MRAM architecture for memory-based computing: Database-search case study," in *Proc. IEEE 15th Int. New Circuits Syst. Conf. (NEWCAS)*, Strasbourg, France, Jun. 2017, pp. 309–312.

[3] S. Matsunaga *et al.*, "Fabrication of a nonvolatile full adder based on logic-in-memory architecture using magnetic tunnel junctions," *Appl. Phys. Exp.*, vol. 1, no. 9, p. 091301, 2008.

[4] T. Hanyu *et al.*, "Standby-power-free integrated circuits using MTJ-based VLSI computing," *Proc. IEEE*, vol. 104, no. 10, pp. 1844–1863, Oct. 2016.

[5] D. Fan, S. Angizi, and Z. He, "In-memory computing with spintronic devices," in *Proc. IEEE Comput. Soc. Annu. Symp. VLSI (ISVLSI)*, Jul. 2017, pp. 683–688.

[6] P.-E. Gaillardon *et al.*, "The programmable logic-in-memory (PLiM) computer," in *Proc. DATE*, Mar. 2016, pp. 427–432.

[7] H.-R. Lee, C.-W. Jen, and C.-M. Liu, "On the design automation of the memory-based VLSI architectures for FIR filters," *IEEE Trans. Consum. Electron.*, vol. 39, no. 3, pp. 619–629, Jun. 1993.

[8] S. Paul and S. Bhunia, "A scalable memory-based reconfigurable computing framework for nanoscale crossbar," *IEEE Trans. Nanotechnol.*, vol. 11, no. 3, pp. 451–462, May 2012.

[9] K. Rahmani, P. Mishra, and S. Bhunia, "Memory-based computing for performance and energy improvement in multicore architectures," in *Proc. Great Lakes Symp. VLSI*, 2012, pp. 287–290.

[10] A. Rahimi, A. Ghofrani, M. A. Lastras-Montano, K.-T. Cheng, L. Benini, and R. K. Gupta, "Energy-efficient GPGPU architectures via collaborative compilation and memristive memory-based computing," in *Proc. DAC*, Jun. 2014, pp. 1–6.

[11] K. J. M. Martin, M. Rizk, M. J. Sepulveda, and J.-P. Diguet, "Notifying memories: A case-study on data-flow applications with NoC interfaces implementation," in *Proc. DAC*, Jun. 2016, pp. 1–6.

[12] J. Cong, M. Huang, D. Wu, and C. H. Yu, "Invited: Heterogeneous datacenters: Options and opportunities," in *Proc. DAC*, Jun. 2016, pp. 1–6.

[13] P.-H. Pham, D. Jelaca, C. Farabet, B. Martini, Y. LeCun, and E. Culurciello, "NeuFlow: Dataflow vision processing system-on-a-chip," in *Proc. 55th Int. Midwest Symp. Circuits Syst. (MWSCAS)*, Aug. 2012, pp. 1044–1047.

[14] P. Merolla *et al.*, "A million spiking-neuron integrated circuit with a scalable communication network & interface," *Science*, vol. 345, no. 6197, pp. 668–673, 2014.

[15] Y. Wang *et al.*, "An energy-efficient nonvolatile in-memory computing architecture for extreme learning machine by domain-wall nanowire devices," *IEEE Trans. Nanotechnol.*, vol. 14, no. 6, pp. 998–1012, Nov. 2015.

[16] H. Jarollahi *et al.*, "A nonvolatile associative memory-based context-driven search engine using 90 nm CMOS/MTJ-hybrid logic-in-memory architecture," *IEEE J. Emerg. Sel. Topics Circuits Syst.*, vol. 4, no. 4, pp. 460–474, Dec. 2014.

[17] J.-P. Diguet, M. Strum, N. Le Griguer, L. Caetano, and M.-J. Sepúlveda, "Scalable NoC-based architecture of neural coding for new efficient associative memories," in *Proc. Int. Conf. Hardw./Softw. Codesign Syst. Synthesis (CODES+ISSS)*, Montreal, QC, Canada, Sep./Oct. 2013, pp. 1–9.

[18] J. Yoo, S. Yoo, and K. Choi, "Active memory processor for network-on-chip-based architecture," *IEEE Trans. Comput.*, vol. 61, no. 5, pp. 622–635, May 2012.

[19] S. Ikeda *et al.*, "A perpendicular-anisotropy CoFeB-MgO magnetic tunnel junction," *Nature Mater.*, vol. 9, no. 9, pp. 721–724, Jul. 2010.

[20] R. Takemura *et al.*, "A 32-Mb SPRAM With 2T1R memory cell, localized bi-directional write driver and '1'/'0' dual-array equalized reference scheme," *IEEE J. Solid-State Circuits*, vol. 45, no. 4, pp. 869–879, Apr. 2010.

[21] T. Ohsawa *et al.*, "A 1 Mb nonvolatile embedded memory using 4T2MTJ cell with 32 b fine-grained power gating scheme," *IEEE J. Solid-State Circuits*, vol. 48, no. 6, pp. 1511–1520, Jun. 2013.

[22] C. Kim, K. Kwon, C. Park, S. Jang, and J. Choi, "A covalent-bonded cross-coupled current-mode sense amplifier for STT-MRAM with 1T1MTJ common source-line structure array," in *Proc. IEEE Int. Solid-State Circuits Conf. (ISSCC)*, Feb. 2015, pp. 1–3.

[23] H. Sato *et al.*, "Properties of magnetic tunnel junctions with a MgO/CoFeB/Ta/CoFeB/MgO recording structure down to junction diameter of 11 nm," *Appl. Phys. Lett.*, vol. 105, no. 6, p. 062403, 2014.

[24] K. C. Chun, H. Zhao, J. D. Harms, T.-H. Kim, J.-P. Wang, and C. H. Kim, "A scaling roadmap and performance evaluation of in-plane and perpendicular MTJ based STT-MRAMs for high-density cache memory," *IEEE J. Solid-State Circuits*, vol. 48, no. 2, pp. 598–610, Feb. 2013.

[25] S. Peng *et al.*, "Interfacial perpendicular magnetic anisotropy in sub-20 nm tunnel junctions for large-capacity spin-transfer torque magnetic random-access memory," *IEEE Magn. Lett.*, vol. 8, 2017, Art. no. 3105805.

[26] N. Sakimura *et al.*, "High-speed simulator including accurate MTJ models for spintronics integrated circuit design," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, May 2012, pp. 1971–1974.

[27] *Ml Repository*. Accessed: Sep. 2017. [Online]. Available: https://archive.ics.uci.edu/ml/

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

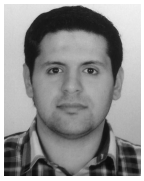
Jean-Philippe Diguet (M'07) received the Ph.D. degree from the University of Rennes 1, Rennes, France, in 1996.

In 1997, he was a Visiting Researcher at imec, Leuven, Belgium. He was an Associate Professor with the University of Southern Brittany, Lorient, France, until 2002. In 2003, he co-funded the dixip company in the domain of wireless embedded systems. Since 2004, he has been a CNRS Researcher with Lab-STICC, Lorient/Brest, France, where he is currently heading the MOCS Team and the CNRS Director of Research. He was a Visiting Researcher at The University of Queensland, Brisbane, QLD, Australia, in 2010, and an Invited Professor at Tohoku University, Sendai, Japan, in 2014, and at the University of São Paulo, São Paulo, Brazil, in 2016. His current research interests include embedded system design: designs and tools for network-on-chip-based MPSoC architectures, self-adaptivity for uncertain environments, and dedicated hardware architectures, including security and embedded vision.

Naoya Onizawa (M'09) received the B.E., M.E., and D.E. degrees in electrical and communication engineering from Tohoku University, Sendai, Japan, in 2004, 2006, and 2009, respectively.

He was a Post-Doctoral Fellow at the University of Waterloo, Waterloo, ON, Canada, in 2011, and McGill University, Montreal, QC, Canada, from 2011 to 2013. In 2015, he was a Visiting Associate Professor with the University of Southern Brittany, Lorient, France. He is currently an Assistant Professor with the Frontier Research Institute for Interdisciplinary Sciences, Tohoku University. His current research interests include the energy-efficient VLSI design based on asynchronous circuits and probabilistic computation and their applications, such as associative memories and brainlike computers.

Dr. Onizawa received the Best Paper Award from the 2010 IEEE ISVLSI, the Best Paper Finalist at the 2014 IEEE ASYNC, the 20th Research Promotion Award from the Aoba Foundation for the Promotion of Engineering in 2014, and the Kenneth C. Smith Early Career Award for Microelectronics Research from the 2016 IEEE ISMVL.



Mostafa Rizk received the Maitrise degree in electronics, the M.Sc. degree in biomedical physics, and the M.Sc. degree in signal, telecom, image, and speech from Lebanese University, Beirut, Lebanon, in 2007, 2008, and 2010, respectively, the Ph.D. degree in sciences and technologies of information from Telecom Bretagne, Brest, France, in 2014, and the Ph.D. degree in electronics and communication from Lebanese University in 2015.

He has been a Post-Doctoral Researcher with the University of Southern Brittany, Lorient, France, and with the Lab-STICC Laboratory CNRS, Lorient. He is currently an Assistant Professor with Lebanese International University, Beirut, and an Associate Researcher with IMT Atlantique, Brest, France. His current research interests include algorithm development for digital baseband components and corresponding hardware/software implementations and digital circuit design, network-on-chip design, and new MPSoC architectures based on emerging nonvolatile memory technologies.

Johanna Sepulveda (M'04) received the M.Sc. and Ph.D. degrees in electrical engineering microelectronics from the University of São Paulo, São Paulo, Brazil, in 2006 and 2011, respectively.

She was a Post-Doctoral Fellow with the Integrated Systems and Embedded Software Group, University of São Paulo, and the Group of Embedded Security, University of Southern Brittany, Lorient, France. She was a Visiting Researcher with the Computer Architecture Group, University of Bremen, Bremen, Germany. In 2014, she was a Senior INRIA Post-Doctoral Researcher at the Heterogeneous Systems Group, University of Lyon, Lyon, France. Since 2015, she has been a Senior Researcher Assistant at the Technical University of Munich, Munich, Germany. She has been involved in the embedded security design for more than 10 years. Her current research interests include high-performance SoC design and new technologies design.

Amer Baghdadi (M'00–SM'15) received the Engineering, M.Sc., and Ph.D. degrees from the Grenoble Institut National Polytechnique, Grenoble, France, in 1998, 1998, and 2002, respectively, and the Accreditation to Supervise Research (HDR) degree in sciences and technologies of information and communication from the University of Southern Brittany, Lorient, France, in 2012.

He is currently a Professor with IMT Atlantique/ Lab-STICC Laboratory, France. He has coauthored over 100 papers on scientific journals and proceedings of international conferences. His current research interests include theoretical and practical aspects, both algorithm development for digital baseband components and corresponding hardware/software implementations and digital circuit design, digital communication applications, in addition to other application domains, and more particularly the design of flexible digital physical layer for future wireless communication standards and terminals.

Dr. Baghdadi serves on the technical program committee for several international conferences.

Takahiro Hanyu (M'89–SM'12) received the B.E., M.E., and D.E. degrees in electronic engineering from Tohoku University, Sendai, Japan, in 1984, 1986, and 1989, respectively.

He is currently a Professor and the Vice Director with the Research Institute of Electrical Communication, Tohoku University. His current research interests include nonvolatile logic circuits and their applications to ultralow-power and/or highly dependable VLSI processors, and postbinary computing and its application to brain-inspired VLSI systems.

Dr. Hanyu received the Sakai Memorial Award from the Information Processing Society of Japan in 2000, the Judge's Special Award at the 9th LSI Design of the Year from the Semiconductor Industry News of Japan in 2002, the Special Feature Award at the University LSI Design Contest from ASP-DAC in 2007, the APEX Paper Award from the Japan Society of Applied Physics in 2009, the Excellent Paper Award at IEICE, Japan, in 2010, the Ichimura Academic Award in 2010, the Best Paper Award at the IEEE ISVLSI 2010, the Paper Award at SSDM 2012, the Best Paper Finalist at the IEEE ASYNC 2014, and the Commendation for Science and Technology by MEXT, Japan, in 2015.

6.2 MRAM-based memorization system for NB-LDPC decoder

This research work investigates the use of emergent non-volatile STT-MRAM technology in designing energy-efficient components in the digital communication domain. The main aim is to reduce the power consumption due to memory accesses during intermediate processing of received data. To address this issue, an adaptive and efficient memorization system based on STT-MRAM is proposed to be implemented within Non Binary-Low Density Parity Check (NB-LDPC) decoder [97]. Based on the available MRAM types and array dimensions, new data mapping and efficient structure of the memorization system are proposed. The power consumption of the NB-LDPC decoder with STT-MRAM is evaluated and compared to SRAM-based one. Simulation results show that when MRAMs are used, the proposed memorization system offers a significant energy reduction along with an increase in terms of throughput.

This work has been performed in the context of the Master's degree project at the Lebanese International University and has been published in the proceedings of *IEEE International Conference on Microelectronics (ICM)* [98] and in the proceedings of *IEEE International Conference on Computer and Applications (ICCA)* [99]. Moreover, the publication in the *IEEE International Conference on Microelectronics (ICM)* [98] was attributed one of three best student paper awards in 2017.

6.3 Memristor Based Reconfigurable FFT Architecture

6.3.1 Preface

This research work has been accomplished as a part of the PhD thesis of Khaled Alhaj Ali at IMT Atlantique, Brest, France. The thesis aimed to explore and introduce new memristor-based designs that combine flexibility and efficiency through the proposal of original architectures that break the limits of the existing ones. This work has focused on using memristive devices at interconnect level to allow high degree of flexibility based on programmable interconnects, where memristors are inserted as reconfigurable switches in order to establish on-chip routing. This research work has been published in the proceeding of the *IEEE International Conference on Microelectronics (ICM)* [100].

6.3.2 Introduction

The recent development of new non-volatile memory technologies based on the memristor [101] concept has triggered many efforts to explore their potential usage in different application domains. This novel type of two terminal nano-scale elements presents very fast switching characteristics, non-volatile dense storage capacity, and low power consumption.

Main part of conducted efforts aims to exploit them for establishing a unified and efficient memory system replacing current flash and CMOS-based memories. Before the discovery of

the HP memristor, memristive devices were only employed as storage elements. Since the HP Labs presented the T_iO_2 memristor in 2008, rapid progress on the fabrication of high-quality memristors has been achieved in the past few years [102][103][104]. These memristors are characterized by huge gap between their low resistance state (LRS) and high resistance state (HRS). This evolution has received significant attention and allowed for employing these devices in new fields such as non-volatile programmable switches. Memristor-based programmable switches are introduced to achieve efficient and low cost flexibility in ASIC designs. Such flexible designs are reconfigured allowing for efficient reuse of resources for different application need. Moreover, memristors have been explored as potential alternative to the traditional programmable interconnects in FPGAs [2][105][106][107].

The possibility to integrate memristors on top of CMOS logic gates allows for new design ideas based on close combination and interaction between memory and computation. This introduces new opportunities of efficient reconfiguration, high performance, and low power design. Designing flexible architectures, which can adapt dynamically to application needs, bring great advantages in terms of energy efficiency and performances. Such flexibility is required at processing, interconnect, and memory levels. Current technologies are inefficient for highly self-adaptive systems, due to the cost of reconfiguration, including delay and power consumption [108][2]. Flexibility is particularly required in digital communication and multimedia applications where new standards and multiple service modes are continuously emerging, with strengthen requirements in terms of performance and energy efficiency.

In this context, the integration of memristors at the interconnect level to enable flexible and efficient configuration of digital architectures is explored. This work has proposed memristor-based reconfigurable fast Fourier transform (FFT) architecture. To the best of our knowledge, this is the first memristor-based FFT design in the literature. The proposed original architecture allows an efficient support of any combination of radix-2 and radix-3 butterflies. Scalability is ensured through a 2D mesh topology. Flexibility is realized at the level of interconnects, allowing for optimized hardware reuse through a memristor-based non-volatile routing.

6.3.3 Context and Motivations

Memristors are employed to implement non-volatile routing switches by logically connecting/disconnecting wires during configuration mode. Non-volatile routing preserves the configuration information stored on-chip while powered OFF, allowing the devices to immediately run when it is powered up. On the other hand, leakage power is minimized during stand-by mode, leading the system to work at extremely low-power. Moreover, the opportunity to fabricate memristors on the top of transistor layer in the same die brings significant reduction of the overall area of the system design.

These features are of high interest when targeting multi-mode ASIC designs as well as high-end FPGAs. It is reported that the routing resources in an FPGA (Figure 6.1) including switch blocks (SBs), connection blocks (CBs) and interconnects can account for up to 70% of the total area, delay, and consumed power [108]. Thus, the improvement of these programmable routing elements in FPGAs is of importance for research and development. In

this context, memristors have been presented in the literature [2][105][106][107] as potential alternative to the conventional SRAM-based programmable interconnects in FPGAs.

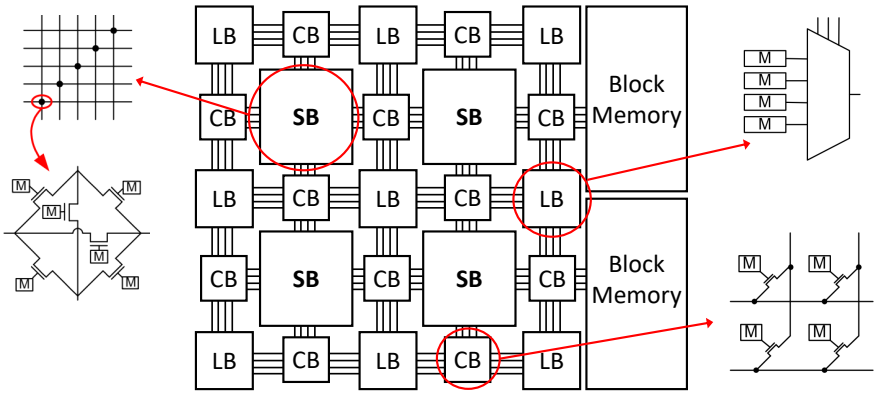


Figure 6.1 — Conventional FPGA architecture

For instance, authors of [2] have proposed an FPGA architecture with memristor-based reconfiguration (mrFPGA). The programmable interconnects of mrFPGA use only memristors and metal wires so that the interconnects can be fabricated over logic blocks, resulting in significant reduction of overall area and interconnect delay. As demonstrated in Figure 6.2, the area of mrFPGA has been reduced to the total area of the logic blocks only, which takes 10% to 20% of the conventional FPGA area.

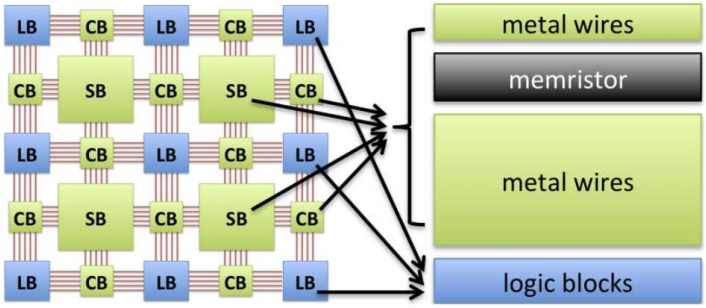


Figure 6.2 — Memristor based-programmable routing structure for FPGA proposed in [2]

Several studies [109][110] have demonstrated the use of memristive devices as programmable switches instead of SRAM-based pass transistors in conventional FPGAs. Figure 6.3 shows the structures of conventional SRAM routing switch and the two memristor switches proposed [110].

FFT [111] has been chosen as application case study since it is used widely in the fields of digital signal processing and telecommunications. Particularly, FFT is considered recently as an effective tool for spectrum enhancement of orthogonal frequency division multiplexing (OFDM)-based waveform, which is a central element in the fifth generation new radio (5G-NR) developments [112]. Furthermore, the size of FFT block can vary according to the system parameters and the communication channel conditions. Thus, there is a real need for highly flexible FFT implementations that support multiple configurations. The attained

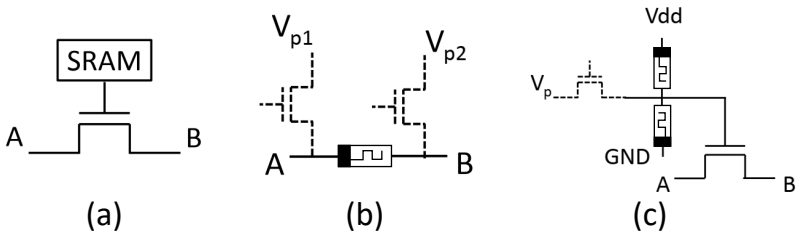


Figure 6.3 — Memristive switches: (a) 7T SRAM routing switch, (b) 2T1R routing switch, (c) 2T2R routing switch

degree of flexibility in the available implementations dedicated for FFT is hampered by the high cost of reconfiguration elements, such as multiplexers, SRAM cells and buffers. These elements constitute the major overhead in terms of area, power consumption and delay. For the sake of realizing efficient flexible FFT design, we have investigated the advantage of exploiting memristive devices as routing switches at the level of interconnects.

It is possible to implement a fully-parallel FFT flow graph directly in hardware. However, for large FFTs it may be infeasible, since the area usage would be too large [113]. In case of a power of 2 FFT size, radix-2 butterfly (BF) is typically used [114]. Different radix butterflies are also proposed either to support different FFT sizes (not power of 2) and/or to reduce the computational complexity. If the application requires to have different radices, typical inefficient implementations duplicate the hardware resources. Reconfigurable implementations of FFT have been presented in the literature. Implementations with simplest form of reconfiguration exploit certain portion of the hardware resources for one radix while other independent parts are reserved for another radices. These implementations allow the concatenation/combination of several processing elements having different radices on the cost of duplication of hardware resources. Thus, they are considered inefficient in terms of area and energy. Other implementations the same processing elements can be reused for executing other radices at different instants of time. These implementations are considered reconfigurable/flexible. Figure 6.4(a), illustrates how certain portion of the hardware resources are exploited for one radix while other independent parts are reserved for another radices. Whereas, Figure 6.4(b), illustrates how the hardware resources, which are utilized for executing certain radix, can be reused for executing other radices at different instants of time.

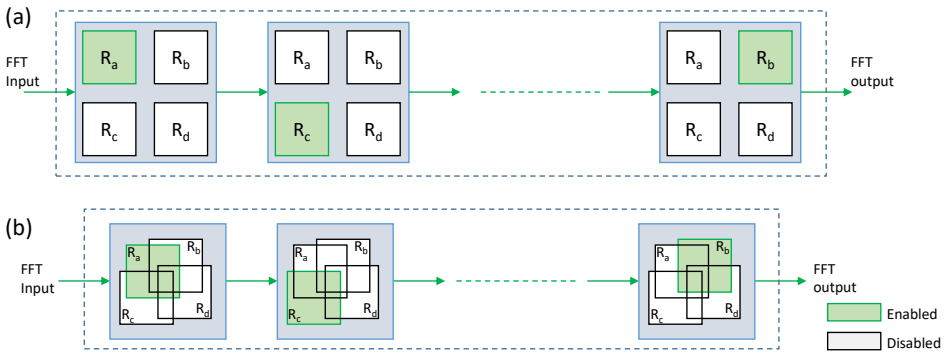


Figure 6.4 — The block diagram of the reconfigurable FFT

The authors in [115] have proposed a reconfigurable FFT design that is able to support

48 different configurations with an FFT size up to 2187. The design is composed of several reconfigurable bricks, which are arbitrarily concatenated to build up a system with reconfigurable FFT size. Flexibility is achieved through a reconfigurable processing element that implements six combinations of radix-2 and radix-3 butterflies.

6.3.4 Conducted Work and Achieved Results

In this research work, a memristor-based reconfigurable FFT architecture (mrFFT) is proposed, which allows the efficient support of many configurations with different FFT sizes. The architecture can be employed in a reuse and systematic way to support radix-2 and radix-3 kernels [115]. Flexibility is achieved by inserting memristors at the interconnect level leading to programmable memristive nodes.
The main contributions of this work are:

- Investigate the structures of the flow graphs of FFT radices. This reveals that it is possible to extract similar/common parts. These similarities are exploited toward realizing efficient re-usability of hardware resources leading to efficient flexible designs. In particular, the analysis of radix-2 and radix-3 flow graphs shows that a radix-3 flow graph can be split into three similar butterflies and a separate real multiplier.

- Introduce the reconfigurable BF, so called RBF, which can be configured to execute radix-2 BF or any part of the radix-3 BF with the minimal added cost.

- Propose the mrFFT architecture based on RBF design with FPGA-like resource arrangement, where a group of RBFs are arranged in a 2D mesh topology with memristor-based interconnections serving as routing elements.

- Design the peripheral circuits, dedicated controller and selection blocks, that are used to reconfigure the states of the implemented memristors.

- Demonstrate the efficiency of the proposed architecture to support of 44 configurations with different FFT sizes including the 32 operating modes that are defined in 3GPP-LTE standard.

- Conduct preliminary comparisons with state-of-the-art work [115] in terms of used resources and hardware re-usability ratio in order to determine the relevancy of our proposed mrFFT design.

6.3.5 Findings

The presented mrFFT design accommodates the 32 operating modes that are defined in 3GPP-LTE standard [116, 115]. However, mrFFT is scalable in size and can be extended. Thus, it is able to support other FFT configurations in case of any future standard changes.

The amount of utilized resources in our proposed design has been estimated and compared to that in [115]. The comparison shows that mrFFT reduces significantly the number

of the utilized adders (25%), multipliers (37%), 2-to-1 multiplexers (59%), 3-to-1 multiplexers (100%) and FIFOs (25%). On the other hand, the design in [115] has a 55.5% less number of 1-bit shifters. However, these shifters have a trivial implementation.

The attained gains in mrFFT come at the cost of the adopted routing scheme, which incorporates a set of 119 memristive node in addition to the corresponding selection blocks. In fact, the implementation cost of the selection blocks can be relatively reduced when shared to other designs targeting the same flexibility requirements.

Moreover, the percentage of resource activation has been evaluated in both designs among all supported configurations. The obtained results show better utilization of adders and multipliers in mrFFT compared to that in [115]. This implies better hardware reusability ratios.

The evaluation of the devised mrFFT architecture has been limited to an analytical estimation of the utilized resources, the percentage of activation and the reusability ratio. Power consumption and delay analysis could not be evaluated for this first proposed design as analog/digital mixed-signal simulation environment is required, which relevant tools have not been easily available.

6.4 Hybrid Memristor-CMOS Design for Logic Computation

6.4.1 Preface

This research work aims to explore the use of memristive devices and their integration with CMOS technologies for combinational logic design. Such hybrid memristor-CMOS designs should exploit the high integration density of memristors in order to improve the performance of digital designs, and particularly arithmetic logic units. It has been conducted as apart of the PhD thesis of Khaled Alhaj ALI at IMT Atlantique, Brest, France. This research work has been initially published in the proceedings of the *IEEE International Conference on Electronics, Circuits and Systems (ICECS)* [117]. Then, the detailed architecture and results have been published in *Electronics* [118].

6.4.2 Introduction

Memristor technology has recently triggered many efforts to extend their usage from memory to computing [119, 120, 121, 122, 123]. Memristor based logic design is an emerging concept targeting efficient computing systems. Several logic families have evolved, each with different attributes. Memristor Ratioed Logic (MRL) has been recently introduced as a hybrid memristor-CMOS logic family [119]. MRL requires efficient design strategy that takes into consideration the implementation phase.

In fact, MRL is a hybrid memristor-CMOS logic family. The goal behind MRL is to implement conventional combinational logic circuits which are the building blocks of digital systems. The main idea is to replace as much as possible transistors with nano-scale size

memristors, while keeping the same role of the intended digital architecture. Several works have utilized MRL design style to implement various digital architectures [119, 124, 125]. The integration of memristors and CMOS devices in MRL still lacks a consistent way for arranging memristors at the top of CMOS layer. The integration should be realized in such a way that exploits efficiently the promising characteristics of memristive devices such as density and scalability.

In this research work a novel MRL-based crossbar design namely X-MRL has been introduced. The proposed structure combines the density and scalability attributes of memristive crossbar arrays and the opportunity of their implementation at the top of CMOS layer. The design methodology of X-MRL efficiently integrates memristors with CMOS devices to improve density and scalability. The evaluation of the proposed approach is performed through the design of X-MRL based full adder. The design is presented with the layout and the corresponding simulation results using Cadence Virtuoso toolset.

6.4.3 Context and Motivations

The fast decline of Moore's law is paving the way to explore new set of emerging technology devices [126], as it is difficult to overcome the various physical limitations of the traditional CMOS technology [127]. In this context, the nano-scale size memristor was introduced as a possible alternative candidate as it offers a lot of advantageous features including the capability of executing Boolean logic [119][128][129] in addition to the storage role. The presence of these two attributes combined has given a great impetus to explore new innovative circuits and systems based on memristors. For instance, one of the main challenges of modern computers nowadays is the memory wall problem which is originated from the mismatch in the performance of processor and memory. There has been a continuous effort to move processing cores closer to where data resides to address the memory wall problem. A memristor-based logic can integrate processing and storage role (in-memory computing), an attribute which can be a promising solution to scale the memory wall.

Several memristor-based logic design styles have emerged in the literature [119] [130] [121] [128] [123] [122] . Each has its own capabilities and thus is adapted for a specific type of applications. A memristive logic design style is able to compute a certain primitive logic such as AND, OR, NOR, etc. Based on these simple operations, complex arithmetic functions can be executed as any Boolean function could be written in the form of the sum of products (SoP).

The advent of memristor-CMOS process that combines CMOS devices with the nano-scale size memristors have provided new opportunity to reduce the utilization of silicon area. However, yet CMOS devices which are considered active, cannot be totally replaced by the passive memristive devices. Thus, the integration of CMOS and memristors is essential to the development of memristor technology. To this end, hybrid configurations have been proposed that make use of the advantages of CMOS while utilizing the high density of memristors. On the other side, out of the available memristor-based logic design styles, the MRL is the only design style that meets the conventional CMOS in terms of the adopted state variable. Both MRL and CMOS use the voltage as the only state variable for representing inputs and outputs throughout all intermediate stages. Thus, MRL is very qualified for the integration in the

current CMOS designs and even later can dominate. Also, the computation is accomplished in one single step. This criterion eliminates the drawbacks of the sequential process of other memristor-based logic devices.

In fact, MRL is a hybrid memristor-CMOS logic family, where the programmable resistance of memristors is exploited in the computation of the Boolean AND and OR functions. The goal behind MRL is to implement conventional combinational logic circuits which are the building blocks of digital systems. The main idea is to replace as much as possible transistors with nano-scale size memristors, while keeping the same role of the intended digital architecture. Several works have utilized MRL design style to implement various digital architectures [119] [124] [125]. The integration of memristors and CMOS devices in MRL still lacks a consistent way for arranging memristors at the top of CMOS layer. The integration should be realized in such a way that exploits efficiently the promising characteristics of memristive devices such as density and scalability.

6.4.4 Conducted Work

This research work introduce the novel X-MRL approach, which is dedicated for the implementation of combinational logic. For evaluation purposes, a hybrid memristor-CMOS full adder based on X-MRL approach has been devised.
The main contributions of this work are:

- Introduce the X-MRL approach for realizing Boolean computation. Using X-MRL, Boolean functions are represented using pairs of memristors mapped efficiently into crossbar structure. A memristive crossbar is a two dimensional grid of memristors distributed along vertical and horizontal nanowires. A memristor is allocated between every vertical nanowire (so-called a column) and horizontal nanowire (so-called a row).

- Devise the architecture design for a full adder based on X-MRL approach. The proposed design includes memristors to execute MRL-AND and MRL-OR operations existing in the SOP of the 1-bit full adder and several CMOS inverters that are responsible for either inverting (NOT operation) and/or performing signal restoration for the logical state of the signal after several cascading stages.

- Determine the layout of the obtained X-MRL full adder architecture using Cadence Virtuoso toolset. Three layers are used. A polysilicon layer is dedicated to the connection of the gates of NMOS and PMOS transistors. Two metal layers are adopted to implement the horizontal and vertical wires of the crossbar structure. As the height of memristors is too short, vertical interconnect accesses (VIAs) are utilized to connect the two metal layers and the memristors are allocated at the top of the VIAs.

- Conduct simulations using Cadence Virtuoso toolset of the tailored architecture while adopting the VTEAM model to realize a realistic modeling of practical memristors. An important work has been done to determine the VTEAM [131] model parameters that fit with the physical parameters of the memristor device that matches the MRL [102]. Transient simulations are performed adopting CMOS $65\,nm$ technology at standard $1.2V$ to produce the performance analysis and energy consumption.

- Compare the proposed hybrid memristor-CMOS based full adder with previous published designs, which are dedicated to the 1-bit full adder.

6.4.5 Findings

Timing:

The conducted simulation shows that the values of the rising time T_r, falling time T_f and time delay T_d are affected by the switching speed of the memristor which in turn can be controlled by the fitting parameters. On the other hand, slowing down the switching speed of the memristors increases the glitches. The high resistance state (R_{OFF}) of the memristors has a direct effect on the value of T_d which increases when increasing the value of R_{OFF}. Moreover, it is noticed that increasing R_{OFF} acts as a filter for the glitches. Therefore, the total delay is directly affected by the memristor physical properties.

Energy consumption:

The value of the average power consumption is relatively high for a full adder circuit. This is due to the low values of low and high resistance states of the adopted memristor device compared to the source-to-drain dynamic resistance in MOSFETs, which minimizes the leakage in current.

Memristive devices are still being actively explored and developed using a variety of materials and deposition techniques. Thus, there is the potential for the device characteristics to be improved. Memristors with high values of (R_{ONN}), which corresponds to LRS, and (R_{OFF}), which corresponds to HRS, have to be developed in order to achieve hybrid architectures with low power consumption.

Area:

The memristors are implemented at the top of CMOS due to their nano-scale and compatibility at the level of fabrication. Thus, the allocated memristors in the proposed X-MRL design do not add any overhead in terms of implementation area. The total required area refers to that occupied by CMOS devices only, which depends on the number of used inverters. The total area of the X-MRL design is $8.16\ \mu m^2$ compared to $14.78\ \mu m^2$ which is utilized in the case of pure CMOS implementation, leading to 44.79% area saving.

Comparison:

All related works published in the literature lack an estimation about the utilized area for their proposed designs. In order to achieve a fair comparison in terms of energy consumption, the energy is evaluated per 1 addition operation. The *Energy.Delay* metric is used for comparison. It reveals an improvement between $\times 5.7$ and $\times 31$ with respect to the available literature [124][132].

electronics

MDPI

Article

Hybrid Memristor–CMOS Implementation of Combinational Logic Based on X-MRL †

Khaled Alhaj Ali [1,*], Mostafa Rizk [1,2,3], Amer Baghdadi [1], Jean-Philippe Diguet [4] and Jalal Jomaah [3]

[1] IMT Atlantique, Lab-STICC CNRS, UMR 29238 Brest, France; Mostafa.rizk@liu.edu.lb (M.R.); amer.baghdadi@imt-atlantique.fr (A.B.)
[2] Lebanese International University, School of Engineering, Block F 146404 Mazraa, Beirut 146404, Lebanon
[3] Faculty of Sciences, Lebanese University, Beirut 6573, Lebanon; jomaah@enserg.fr
[4] IRL CROSSING CNRS, Adelaide 5005, Australia; jean-philippe.diguet@univ-ubs.fr
* Correspondence: khaled.alhaj-ali@imt-atlantique.fr

† This paper is an extended version of our paper published in IEEE International Conference on Electronics, Circuits and Systems (ICECS) , 27–29 November 2019, as Ali, K.A.; Rizk, M.; Baghdadi, A.; Diguet, J.P.; Jomaah, J. "MRL Crossbar-Based Full Adder Design".

Abstract: A great deal of effort has recently been devoted to extending the usage of memristor technology from memory to computing. Memristor-based logic design is an emerging concept that targets efficient computing systems. Several logic families have evolved, each with different attributes. Memristor Ratioed Logic (MRL) has been recently introduced as a hybrid memristor–CMOS logic family. MRL requires an efficient design strategy that takes into consideration the implementation phase. This paper presents a novel MRL-based crossbar design: X-MRL. The proposed structure combines the density and scalability attributes of memristive crossbar arrays and the opportunity of their implementation at the top of CMOS layer. The evaluation of the proposed approach is performed through the design of an X-MRL-based full adder. The design is presented with its layout and corresponding simulation results using the Cadence Virtuoso toolset and CMOS 65 nm process. The comparison with a pure CMOS implementation is promising in terms of the area, as our approach exhibits a 44.79% area reduction. Moreover, the combined Energy.Delay metric demonstrates a significant improvement (between ×5.7 and ×31) with respect to the available literature.

Keywords: CMOS; crossbar; full adder; logic design; memristor

Citation: Alhaj Ali, K.; Rizk, M.; Baghdadi, A.; Diguet, J.-P.; Jomaah, J. Hybrid Memristor–CMOS Implementation of Combinational Logic Based on X-MRL. *Electronics* **2021**, *10*, 1018. https://doi.org/10.3390/electronics10091018

Academic Editor: Kris Campbell

Received: 22 February 2021
Accepted: 18 April 2021
Published: 24 April 2021

1. Introduction

The memristor is the fourth fundamental circuit element, which relates charge and magnetic flux linkage. It was originally predicted by Professor Leon Chua in 1971 [1]. The memristor was realized later by members of HP Labs in 2008 [2]. This successful realization opened a wide area of research on the memristor and its possible applications. The HP memristor is a solid state device formed of a nanometer scale TiO_2 thin film, containing a doped and an undoped region sandwiched between two platinum electrodes. The obtained two-terminal device exhibits a dynamic resistance that is bounded between a minimum value (R_{ON}) and a maximum value (R_{OFF}). Its resistance depends on the magnitude, direction and duration of the applied voltage across its terminals. The last attained resistance value of the memristor before withdrawing the applied voltage is naturally retained. Memristors are promising in the field of non-volatile memories (NVM) because of their capability for data retention [3] with zero standby power and compatibility with a conventional CMOS in terms of fabrication and operating voltages. Due to their versatile nature, the use of memristors has been extended from memory to computing [4]. Several memristive logic design families have emerged in the literature, each with its own characteristics, capabilities and usage. Memristor Aided Logic (MAGIC) [5] and the

Material Implication (IMPLY) [6] are considered as memristive stateful logic families [7]. They were introduced to allow logic computations inside memristive memory systems and are being explored to overcome the memory wall problem. Memristor-Ratioed Logic (MRL) [8] is another memristor-based logic design style. MRL is a hybrid memristor–CMOS logic family. Its goal is to implement conventional combinational logic circuits, which are the building block of digital systems [8–11]. The main idea behind MRL is to replace as many transistors with nano-scale size memristors as possible while keeping the role of the intended digital architecture the same.

Of the above-mentioned logic design styles, MRL is the only approach that matches the conventional CMOS in terms of the adopted state variable. Both MRL and the CMOS use voltage as the only state variable to represent inputs and outputs throughout all intermediate stages. Thus, MRL is the most qualified for integration in current CMOS designs. However, this integration should be performed in a way that efficiently exploits the promising characteristics of memristive devices, such as density and scalability. This can be achieved through the use of the crossbar structure, which is a highly adapted topology for arranging memristors at the top of the CMOS layer.

In this paper, we propose an MRL-based crossbar design: X-MRL. X-MRL is intended for implementing combinational logic. The conventional CMOS logic gates are implemented using MRL, and an original mapping into a crossbar structure is proposed. The proposed methodology efficiently combines the density and scalability attributes of crossbar arrays and the ability to implement memristors at the top of the CMOS layer. The proposed approach is evaluated by designing an X-MRL-based full adder circuit [12]. The designed architecture is implemented and simulated with the Cadence Virtuoso toolset.

The rest of the paper is organized as follows. Section 2 describes the behavior of the memristor and the corresponding available models. Section 3 presents a brief review of the MRL design style. Section 4 presents the proposed X-MRL design for realizing Boolean computation. Section 5 provides and discusses the simulation results and performance analysis. A comparison with previous published designs is presented in Section 6. Finally, Section 7 concludes the paper.

2. Memristor Behavior and Modeling

Chua [1] has defined the memristor as a previously missing relation between the flux ϕ and the charge q, yielding the defining relation

$$M(q) = d\Phi/dq \tag{1}$$

The current–voltage (I-V) characteristic of a memristor has the form of a pinched hysteresis loop, as illustrated in Figure 1a. The hysteresis phenomenon indicates that memristor resistance could be modulated between two resistance states R_{ON} and R_{OFF}. Figure 1b schematizes the 3D structure of the memristor device, and Figure 1c depicts the typically used symbol to represent a single memristor. HP Labs has described the physical model of a memristor as shown in Figure 2; it consists of two layers of TiO_2 sandwiched between platinum contacts [2]. One of the TiO_2 layers is doped with oxygen vacancies, while the other is left undoped. As a result, the doped region behaves as a semiconductor while the undoped region behaves as an insulator.

The width of the doped region $w(t)$ varies between zero and a memristor length of D according to the amount and direction of the electric charges $q(t)$ moving across the memristor. Thus, applying a certain bias to the memristor leads to the flow of current, which in turn changes the value of $w(t)$. Therefore, the virtual boundary separating the doped and undoped regions moves, leading to a variation in the memristor's total resistance R_{MEM} as expressed in Equation (2) [2].

$$R_{MEM}(x) = R_{ON}(x) + R_{OFF}(1 - x) \tag{2}$$

where $x = \frac{w}{D} \in [0,1]$ and R_{ON} and R_{OFF} are the limiting values of memristor resistance when $w = D$ and $w = 0$, respectively. The speed of the boundary movement between the two ends is called the drift velocity and is represented by the state equation [2]

$$\frac{dx}{dt} = ki(t) \quad for \quad k = \mu_v \frac{R_{ON}}{D^2} \tag{3}$$

where μ_v is the dopant mobility. Equation (3) considers that the drift velocity is constant, resulting in a linear drift model of the memristor. However, the experiments presented in [13,14] proved that the behavior of the implemented memristor is non-linear. To manage the issue of nonlinearity, several models have been proposed in the literature. In [14], the authors proposed a non-linear dopant drift model as a relation between the current and voltage (I-V) of the memristor. In [15], the drift velocity was expressed using a window function $f(w)$ in order to model the non-linearity, as expressed in Equation(4).

$$\frac{dw}{dt} = af(w)V(t)^m \tag{4}$$

where a and m are constants, $f(w)$ is the window function and m is an odd integer. The previous presented models are based on the HP physical representation of a memristor. In [13], Pickett et al. proposed a more accurate physical model of a memristor. A resistor is connected in series with an electron Simmons tunnel barrier [16] instead of connecting two resistors in series, as demonstrated in HP's model. This model exhibits non-linear and asymmetric switching characteristics. Its state equation is expressed in Equation (5):

$$\frac{dx}{dt} = \begin{cases} C_{off} \sinh\left(\frac{i}{i_{off}}\right) \exp\left[-\exp\left(\frac{x-a_{off}}{w_c} - \frac{|i|}{b}\right) - \frac{x}{w_c}\right] \\ C_{on} sinh\left(\frac{i}{i_{on}}\right) \exp\left[-\exp\left(\frac{x-a_{on}}{w_c} - \frac{|i|}{b}\right) - \frac{x}{w_c}\right] \end{cases} \tag{5}$$

where the state variable x represents the width of the Simmons tunnel barrier, C_{off}, C_{on}, a_{off}, a_{on}, w_c and b are the fitting parameters, and i_{off} and i_{on} are the current thresholds of the memristor. Obviously, Equation (5) shows that the Simmons tunnel barrier model is more complicated; thus, it is computationally inefficient. In order to attain a simplified and general model, Kvatinsky et al. [17] presented the TEAM model, which represents in simpler expressions the same physical model as the Simmons tunnel barrier model. Equation (6) expresses the state equation representing the TEAM model:

$$\frac{dx}{dt} = \begin{cases} K_{off}\left(\frac{i(t)}{i_{off}} - 1\right)^{\alpha_{off}} f_{off}(x), & 0 < i_{off} < i \\ K_{on}\left(\frac{i(t)}{i_{on}} - 1\right)^{\alpha_{on}} f_{on}(x), & i < i_{on} < 0 \\ 0 \quad \text{otherwise} \end{cases} \tag{6}$$

where i_{on} and i_{off} are the current thresholds of the memristor. K_{on}, K_{off}, α_{on} and α_{off} are fitting parameters, and $f_{on}(x)$ and $f_{off}(x)$ are the corresponding window functions of the memristor. However, experimental data acquired from several memristive devices reveal the existence of a voltage threshold rather than a current threshold [18]. In [18], the TEAM model was extended to the VTEAM model. Equation (7) describes the VTEAM model. It is similar to the expression in Equation (6), except for the voltage dependence $v(t)$ and the respective SET and RESET voltage thresholds v_{on} and v_{off}. Moreover, the VTEAM model is considered as a general model since it can be fitted to any other memristor model [18].

$$\frac{dx}{dt} = \begin{cases} K_{off}\left(\frac{v(t)}{v_{off}} - 1\right)^{\alpha_{off}} f_{off}(x), & 0 < v_{off} < v \\ K_{on}\left(\frac{v(t)}{v_{on}} - 1\right)^{\alpha_{on}} f_{on}(x), & v < v_{on} < 0 \\ 0 \quad \text{otherwise} \end{cases} \tag{7}$$

Normally, the window function $f(x)$ is added for a memristor model in order to decelerate the moving boundary of the memristor before reaching its extremities and to guarantee a zero speed exactly when it reaches either one of them. In this study, we have adopted the VTEAM model to describe the simulated memristor as it provides simple and realistic modeling.

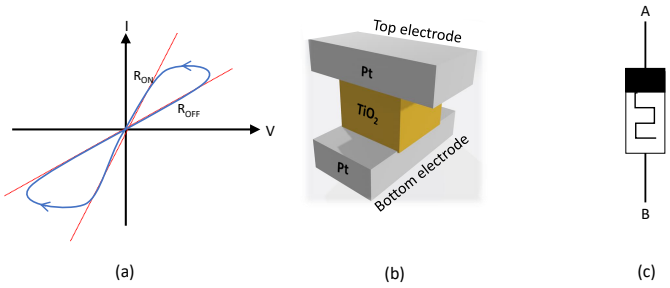


Figure 1. Memristor: (**a**) Pinched hysteresis loop, (**b**) structure: metallic electrodes sandwiching a thin dielectric insulating layer, (**c**) symbol.

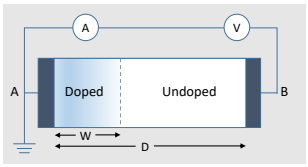


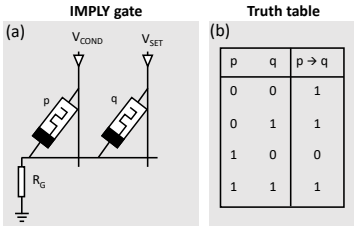
Figure 2. TiO_2 memristor model according to [2].

3. Memristor-Based Logic Design Styles

In the literature, three main design styles for using memristors in logic design can be found. The first two design styles, which are IMPLY [6,19] and MAGIC [5], exploit only memristors for logic implementations. The third design style, known as MRL [8], adopts a combination of CMOS and memristor devices. This section presents a brief overview of these design styles.

3.1. Material Implication IMPLY Gates

In IMPLY, the memristor states (R_{OFF}, R_{ON}) represent the logical state variables (0, 1), respectively. As shown in Figure 3a, the gate consists of the two memristors p and q and the resistor R_G. The initial memristances of p and q represent the input to the gate, while the output is written into memristor q after applying V_{COND} and V_{SET} simultaneously. The truth table of the IMPLY gate is shown in Figure 3b, where $p \longrightarrow q = p^{'} + q$ can be used as a basis for any logic function. As a result, the same memristors are used to store the logical state and/or perform a logical operation. Consequently, the computation requires several sequential operations. Several approaches have been proposed in the literature that adopt IMPLY for the execution of combinational logic [6,20,21]. All available designs require several time steps to accomplish the target computations. This fact leads to an overhead in terms of time delay compared to other logic implementation techniques.



p	q	p → q
0	0	1
0	1	1
1	0	0
1	1	1

Figure 3. IMPLY gate: (**a**) schematic of a memristor-based IMPLY gate, (**b**) truth table of the IMPLY function.

3.2. Memristor Aided Logic (MAGIC)

MAGIC is a memristor-only logic design style that supports Boolean functions [5]. Unlike IMPLY, this logic family makes use of separate memristors to store the input bits, and an additional memristor is used to store the output bit. Figure 4 illustrates the 2NOR1 MAGIC gate where in_1 and in_2 serve as the input memristor and *out* serves as the output memristor. The logic state in MAGIC implementation is represented by the resistance stored in the utilized memristors, where R_{OFF} and R_{ON} represent logic "0" and logic "1", respectively. Thus, when driving the gate with voltage V_0, the result of the NOR operation of in_1 and in_2 is written simultaneously into *out*.

Applications of MAGIC in memristor-based crossbars are straightforward when using MAGIC NOR, while an additional resistor is required in case of other gates. The authors of [22–24] used the MAGIC NOR as the basis to perform logic computation inside the memory, thus adding processing capabilities. In other words, each processing task is divided into a sequence of MAGIC NOR operations, which are executed one after the other using the memory cells as computation elements.

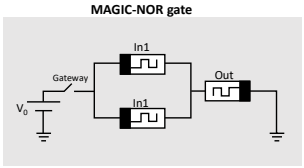


Figure 4. Structure of MAGIC NOR gate.

3.3. Memristor Ratioed Logic (MRL)

The third design style of memristor-based logic is Memristor Ratioed Logic (MRL) [8]. It is a typical hybrid CMOS–memristor logic design where the programmable resistance of memristors is exploited in the computation of the Boolean AND and OR functions. MRL opts for voltage as the state variable, in a similar manner to CMOS-based devices; thus, the computation is accomplished in a single step. This criterion eliminates the drawbacks of the sequential process of IMPLY logic devices. Figure 5 depicts the structures of the MRL AND, NAND, OR and NOR gates. Both OR and AND gates consist of two anti-serial memristors (i.e., connected serially with opposite polarities), whereas for NOR and NAND, a CMOS inverter is added at the output.

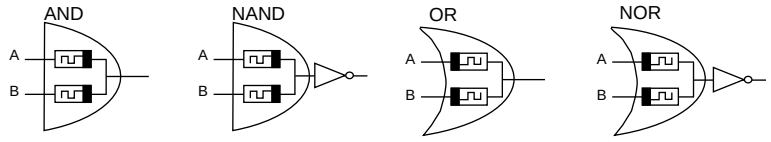


Figure 5. Schematic layout of MRL AND, NAND, OR and NOR gates.

Both MRL AND and OR gates react similarly when identical values are set to their input ports (when both inputs are set either to logic "1" or "0"). In this case, no current flows through the anti-serial memristors, leading to the transfer of the input voltage to the output. In the case where different values are set to the input ports (i.e., the first port is set to "0" and the second port is set to "1", or vice versa), a current flows from the port with higher potential (logic "1") to the port with lower potential (logic "0"). The resulting potential difference changes the internal state of both memristors in an opposite manner. One memristor tends to attain the R_{ON} state while the other tends to attain the R_{OFF} state. In addition, the connected memristors form the well-known voltage divider circuit. Assuming $R_{OFF} >> R_{ON}$, Equations (8) and (9) present the obtained output values V_{out} of MRL OR and AND gates, respectively [8].

$$V_{out,OR} = \left(\frac{R_{OFF}}{R_{OFF} + R_{ON}}\right) \times V_{CC} \quad \approx V_{CC} \tag{8}$$

$$V_{out,AND} = \left(\frac{R_{ON}}{R_{ON} + R_{OFF}}\right) \times V_{CC} \quad \approx 0 \tag{9}$$

Note that the output voltage V_{out} converges to the higher potential (logic "1") in the MRL AND gate and to the lower potential (logic "0") in the MRL OR gate. Figure 6 illustrates the logical operations of the MRL AND gate corresponding to all input combinations.

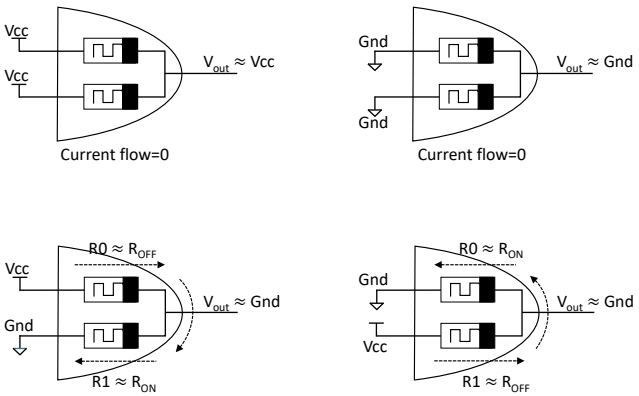


Figure 6. Logical operations performed with MRL AND gate.

However, cascading several MRL gates leads to a floating output (between logic "0" and logic "1") due to voltage degradation [8]. Since memristors are passive devices, they cannot amplify signals. Therefore, CMOS inverters can be used as buffers after several stages to restore the attained logical state [8].

Several recent research works presented in the literature exploit the use of MRL to design basic building blocks. In [8], a design dedicated to a universal full adder circuit was proposed using MRL gates with the aid of CMOS inverters instead of pure CMOS-based gates. In [25], the authors demonstrated a simple circuit based on MRL which is capable of

executing AND, OR and XOR in parallel. The design is considered to be hybrid. However, cascading several stages of this circuit degrades its performance due to the voltage drop at the output. In [11], the authors presented an implementation strategy for a memristor-based Programmable Logic Array (PLA). The memristor-based circuit transformation of PLA is based on MRL gates. However, the arrangement of memristors at the top of the CMOS layer and the corresponding layout were not investigated by the authors.

4. Proposed X-MRL Design for Logic Computation

4.1. X-MRL Structure

A memristive crossbar is a two-dimensional grid of memristors distributed along vertical and horizontal nanowires. A memristor is allocated between every vertical nanowire (called a column) and horizontal nanowire (called a row). A memristive crossbar is characterized by its simple and dense structure [26] and could be fabricated on the top of a CMOS layer [27]. The potential applications of memristive crossbars range from memory to logic and from digital circuits to analog circuits. On the other hand, MRL is the only memristor-based logic design style that adopts voltage as a state variable. Thus, our proposed design considers the implementation of a combinational Boolean function in a crossbar topology.

It is well known that any Boolean function could be written in the form of the sum of products (SoP). Accordingly, it can be implemented using MRL-AND and MRL-OR with the aid of CMOS inverters. In order to clarify the proposed method, Figure 7 illustrates the design and implementation of the simple function $F = AB + AC$. Figure 7a shows that the function F is implemented using two MRL-AND gates and one MRL-OR gate. Figure 7b depicts the schematic layout, which illustrates the equivalent mapping of the function onto a crossbar structure. The vertical pairs of memristors corresponding to MRL-AND generate an output which drives the input of the horizontal pair that represents MRL-OR. Figure 7c presents a 3D view of the resulting crossbar structure. Figure 7d is another simplified representation of the obtained crossbar. The same procedure could be performed to implement other Boolean functions. Although the obtained array is a combination of AND and OR gates, the positive poles of the allocated memristors rely on the same planar side, which is considered to be an advantage at the level of their fabrication.

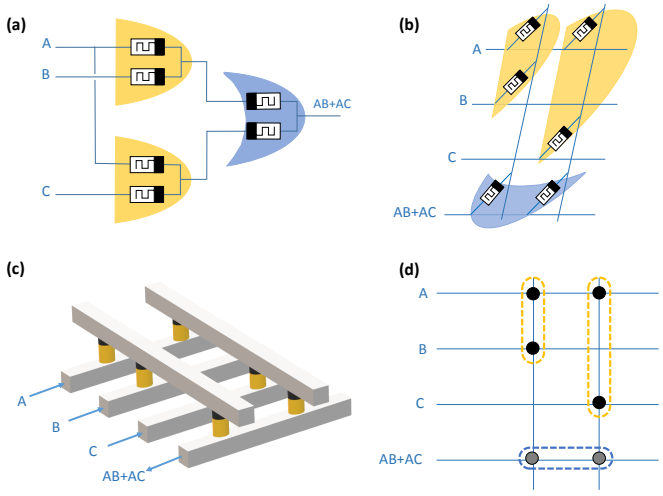


Figure 7. Example of an MRL logic function performed using X-MRL. Reproduced with permission from [12], Copyright 2019, IEEE.

4.2. X-MRL Full Adder

This subsection presents, as an example, the design of the 1 bit full adder using the X-MRL design technique. Equations (10) and (11) present the expressions of the 1-bit full adder in the SoP format.

$$S = A \oplus B \oplus C_{in} = C_{in}(\overline{A\overline{B} + \overline{A}B}) + \overline{C_{in}}(A\overline{B} + \overline{A}B) \tag{10}$$

$$C_{out} = AB + BC_{in} + AC_{in} \tag{11}$$

where A and B are the inputs, C_{in} is the input carry, S is the 1-bit adder output and C_{out} is the output carry. Figure 8a presents the direct form of an MRL based 1-bit full adder. Figure 8b presents the proposed circuit design of the 1-bit full adder using an MRL-based crossbar structure. The design requires 18 memristors, which are distributed among vertical and horizontal wires, in addition to nine CMOS inverters. In the figure, the black vertical pairs of memristors represent the AND gates while the gray horizontal pairs represent the OR gates (as illustrated in Figure 7). The CMOS inverters are responsible for either inverting (NOT operation) and/or performing signal restoration for the logical state of the signal after several cascading stages.

It is worth noting that the designed X-MRL array is different from conventional crossbar arrays as a certain number of crosspoints are vacant. In this array, all memristors are accessed simultaneously, leading to deterministic current paths. Thus, there are no unexpected current paths, and consequently it is sneak-path free.

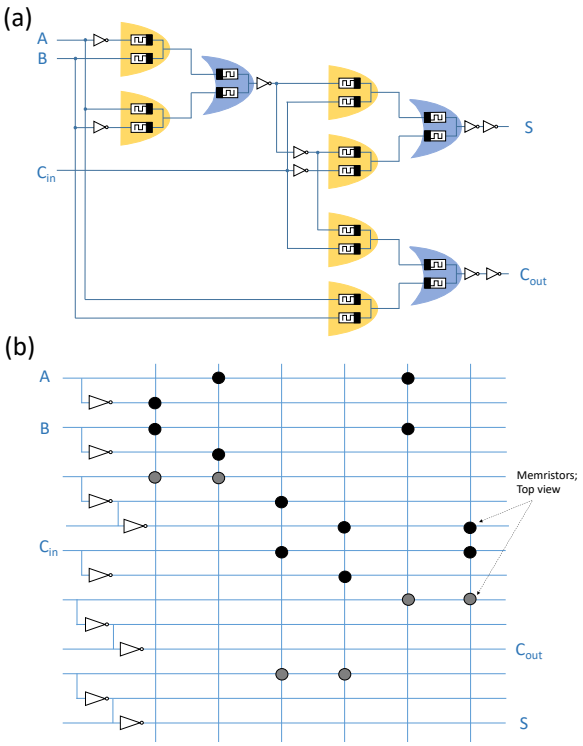


Figure 8. One-Bit Full Adder based on the proposed X-MRL structure. Reproduced with permission from [12], Copyright 2019, IEEE.

4.3. Layout

The circuit of the full adder is composed of a memristor crossbar layer in addition to a few inverters. Figure 9 presents the layout of the circuit using the Cadence Virtuoso tool. In this layout, the positions of the allocated memristors are assigned virtually due to the lack of their definition in the Cadence library. The layout is mainly composed of three layers. The first layer is the polysilicon layer, which is dedicated to the connection of the gates of NMOS and PMOS transistors. This layer is presented in red in the figure. The second and third layers, which are called Metal1 and Metal2 and presented in the figure in violet and blue, respectively, are dedicated to the wiring. In order to achieve the desired crossbar structure, horizontal wires are constructed in the Metal2 layer, while for the vertical wires, the connections that are already utilized for the implementation of the required CMOS inverters are reused to complete the crossbar structure. However, the height of the utilized memristors is too short (around 10 nm [28]) to allow the linkage of horizontal and vertical wires through two different layers. Therefore, these links are achieved through vertical interconnect accesses (VIAs) as demonstrated in [29]. Figure 10 is a schematic view and cross-sectional transmission electron microscopy (TEM) image of a memristor integrated with a CMOS in the same die [29].

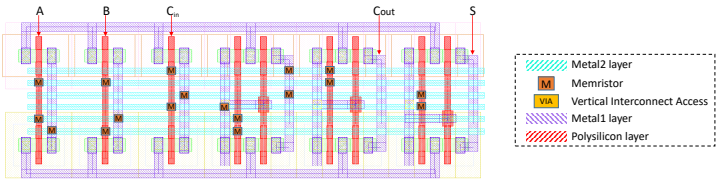


Figure 9. Proposed layout for the hybrid memristor–CMOS 1-bit full adder based on the X-MRL design technique.

The allocated memristors in our proposed layout are implemented at the top of the VIAs immediately under the Metal2 layer. Accordingly, the CMOS inverters occupy most of the utilized area, and the additional Metal2 layer is reserved for memristors. In fact, the designed crossbar causes N-wells and P-wells to be slightly too far from each other. The obtained layout design could be made more compact if the memristors were implemented immediately above the CMOS devices. However, this prevents the realization of an X-MRL approach, as more routing signals would be then added, leading to more wiring in Metal1 and Metal2 layers. This would again cause N-wells and P-wells to be distant from each other, increasing the area overhead.

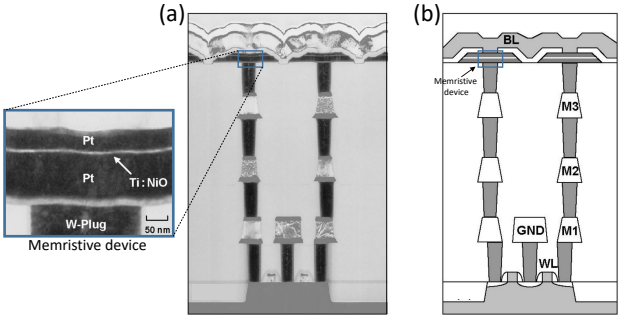


Figure 10. Memristor layer at the top of VIAs [29]: (**a**) a TEM image; (**b**) a schematic view. Reproduced with permission from [29], Copyright 2019, IEEE.

5. Simulation and Performance Analysis

5.1. Memristor Model Fitting

Physical models of memristors, which are based on filament formation and rupture [30,31] rather than a simplified moving boundary (as described by HP), are more realistic. However, compared to physical models, a compact model (e.g., VTEAM) provides the possibility of rapidly reproducing the phenomenological electrical behavior of memristors with a low computational cost. Accordingly, the VTEAM model has been adopted in this paper for simulation. The VTEAM model can mathematically fit the measured electrical behaviors of a memristor and can be easily extended to different types of memristors. In contrast, other models (e.g., the Stanford model [32]) focus on a specific type of memristors or even a single memristor device. Table 1 provides the experimental data of various available memristors with their respective properties. Other memristive devices that belong to the STT-MTJ family [33,34] are excluded from the table. This is due to the fact that MTJ devices usually exhibit a low R_{OFF}/R_{ON} ratio that does not suit the operation of MRL. Of the memristors listed in Table 1, the HfO_x memristor which has been reported in [35] has properties which suit the MRL gates. The device is characterized by a low switching delay 300 ps at a low operating voltage of 1.4 V.

These characteristics mean that this memristor is eligible to be implemented in the same die with the current CMOS devices. Important work regarding the implementation of the VTEAM model parameters that fit with the physical parameters of HfO_x is described in [35]. Table 2 shows the determined VTEAM model parameters. The model parameters are chosen to produce a switching delay of 300 ps for a voltage pulse of 1.4 V, as reported in [35].

Table 1. Practical memristor devices.

Material	R_{ON} (ohm)	R_{OFF} (ohm)	R_{OFF}/R_{ON}	Switching Speed	Voltage Range	Reference
TiO_{2-x}	-	-	>300	1 ns	−1.5 V to +1.5 V	[28]
FTJ	1.6×10^5	4.6×10^7	>200	10 ns	−5.6 V to +4.2 V	[36]
HfO_2	1.2×10^2	10^5	10^3	<1 ns	<1.5 V	[37]
HfO_x	<10 k	>100 k	>100	300 ps	<1.4 V	[35]
TMO	-	100 k	-	10 ns to 100 ns	3 V	[38]
HfO_2	2×10^3	2×10^5	100	-	−1.5 V to +1 V	[39]
$TiN/TiO_x/HfO_x/TiN$	1 k	>1 M	>1000	5 ns	−1.5 V to +1.5 V	[40]

Figure 11 shows the switching behavior of the memristor corresponding to SET and RESET pulses. The device is assumed to be completely switched when the boundary position w reaches either 1% or 99% of the total length D of the memristor, corresponding to SET ($V_{set} = 1.4$ V) and RESET ($V_{reset} = -1.4$ V) operations, respectively. The boundary conditions of the memristor are managed by a Biolek window function. The mathematical function of the Biolek window [41], which is described in Equation (12), provides a continuous and smooth transition of the boundary when reaching one of the extremities of the memristor.

$$f(x) = 1 - (x - stp(-i(t)))^{2p} \tag{12}$$

where $stp(.)$ represents a unit step function and p is a positive integer. Low values of p lead to a smooth transition of the boundary of the memristor when reaching its extremities, whereas high values lead to sharp transitions.

Table 2. VTEAM fitting parameters for the device in [35].

Parameter	Value	Parameter	Value
R_{ON}	1 kΩ	p	2
R_{OFF}	200 kΩ	α_{on}	3
D	3 nm	α_{off}	3
K_{on}	−0.0162 m/s	V_{on}	0.16 V
K_{off}	0.0162 m/s	V_{off}	−0.16 V
x_{on}	0 nm	x_{off}	3 nm

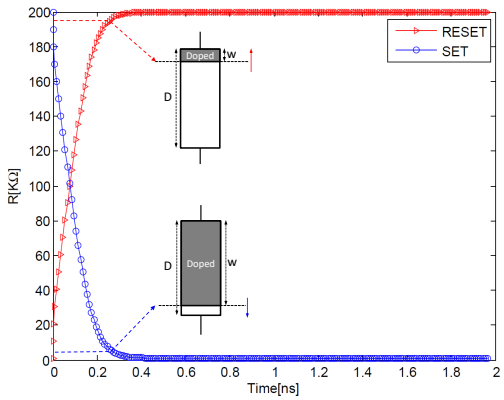


Figure 11. Memristor switching time for $V_{set} = 1.4$ V and $V_{reset} = -1.4$ V according to the device in [35].

5.2. Performance Analysis

A transient simulation was conducted for the proposed design of the X-MRL-based full adder in the Cadence Virtuoso environment. CMOS 65 nm technology at the standard 1.2 V was adopted. Figure 12 shows all the possible combinations at the inputs A, B and C_{in} in addition to the corresponding outputs S and C_{out}. The performance is analyzed below for the proposed design.

5.2.1. Timing Analysis

Figure 13 presents the definition of the rising time (T_r) and the time delay (T_d). Accordingly, the conducted simulation of the proposed design shows that these extracted parameters (T_r and T_d) change among different value combinations of A, B and C_{in}. The maximum recorded values are as follows: $T_r = 82$ ps, $T_d = 1.2$ ns, and $T_f = 586$ ps, where T_f is the falling time. These values are considered for the worst-case performance. The conducted simulation shows that the values T_r, T_f and T_d are affected by the switching speed of the memristor, which in turn can be controlled by K_{on} and K_{off}. On the other hand, slowing down the switching speed of the memristors increases the glitches. Figure 14 shows the appearance of glitches when reducing K_{on} and K_{off} levels to −0.01 m/s and 0.01 m/s, respectively. Particularly, the high-resistance state (R_{OFF}) of the memristors has a direct effect on the value of T_d, which decreases when increasing the value of R_{OFF}. Therefore, the total delay is directly affected by the memristor's physical properties. Moreover, it is observed that increasing R_{OFF} acts as a filter for the glitches. This is due to the fact that a larger R_{OFF} value minimizes the voltage drop at the output ports of MRL gates. Reducing

the voltage drop speeds up the switching of the next cascaded MRLs, resulting in a smaller number of glitches.

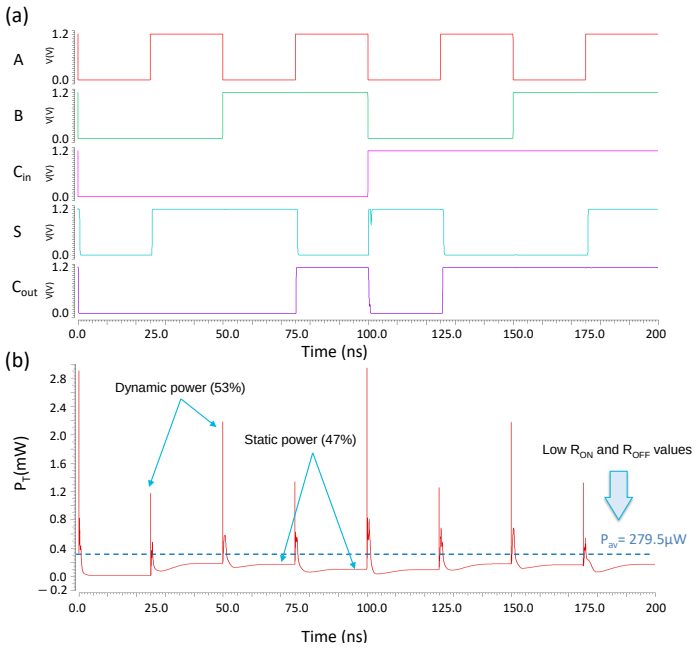


Figure 12. Transient response of the proposed full adder for the input signals A, B and C_{in}.

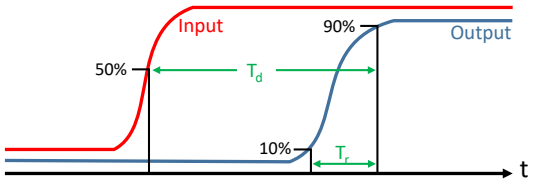


Figure 13. Definition of the rise time T_r and delay T_d.

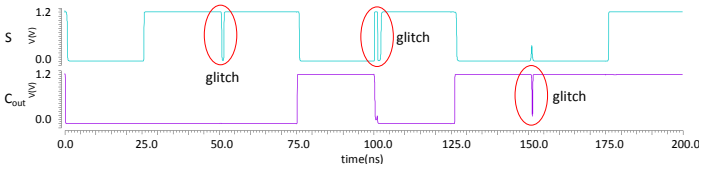


Figure 14. Appearance of glitches when slowing down the switching speed of the memristor. The parameters in Table 2 are adopted except for $K_{on} = -0.01$ m/s and $K_{off} = 0.01$ m/s.

5.2.2. Energy Consumption

Figure 12b shows the total instantaneous power $p_T(t)$ consumed by the proposed design of the full adder. The peak values in $p_T(t)$ refer to the dynamic power consumption. The lower bound in $p_T(t)$, which is formed after the end of each transition, corresponds to

the static power. A slight difference appears between the levels of the static power recorded after each transition. This difference is due to the change in the equivalent resistance state of the cascaded memristors for a new combination of the input signals A, B and C_{in}, which in turn leads to a different level of current leakage. The average power consumed in the proposed design of the full adder is $P_{av} = 279.5\,\mu W$. This value is of P_{av} is evaluated at a frequency f of 200 MHz, which is near to the maximum possible frequency at the inputs of the full adder with a hybrid structure.

The value of the average power consumption is relatively high for a full adder circuit. This is due to the low values of R_{ON} and R_{OFF} of the adopted memristor device compared to the source-to-drain dynamic resistance in MOSFETs, which minimizes the leakage in current. Memristive devices are still being actively explored and developed using a variety of materials and deposition techniques. Thus, there is the potential for the device characteristics to be improved. Memristors with high values of R_{ON} and R_{OFF} have to be developed in order to achieve hybrid architectures with low power consumption.

5.2.3. Utilized Area

A single memristor has an area in the order of $4F^2$ [28], where F is the minimum feature size. Thus, memristors are implemented at the top of the CMOS due to their nano-scale and compatibility at the level of fabrication. Thus, the allocated memristors in the proposed X-MRL design do not add any overhead in terms of the implementation area. The total required area refers to that occupied by CMOS devices only, which depends on the number of inverters, as discussed in Section 4.3. Figure 9 presents the proposed layout. The total area of the X-MRL design is 8.16 μm^2 compared to the area of 14.78 μm^2 utilized in the case of a pure CMOS implementation, leading to a 44.79% area saving.

6. Comparison

The proposed hybrid memristor–CMOS-based full adder was compared with previous published designs dedicated to the 1-bit full adder. Note that related works in the literature lack an estimation of the utilized area of their proposed designs. Moreover, in order to achieve a fair comparison in terms of energy consumption, the energy was evaluated for each additional operation. The time period for an addition operation in our proposed full adder design was set to be the minimum possible time (i.e., the maximum frequency). This subsection presents the comparison summary, which is also shown in Table 3.

Table 3. Comparison with previous approaches.

Reference	Memristors	CMOS Transistors	Energy	Steps	Step Delay	Energy.Delay
(This work)	18	18	0.69 pJ	1	2.5 ns	1.72 pJ.ns
MRL [9]	16	12	-	1	-	-
MRL [42]	18	8	93.7 pJ	1	44.4 ns	4161 pJ.ns
MAGIC [43]	9	Peripheral drivers	0.3 pJ	35	1.89 ns	19.84 pJ.ns
MAGIC (Optimized no. of steps) [23]	10	Peripheral drivers	3.16 pJ	13	1.3 ns	53.40 pJ.ns
MAGIC (Area optimized) [23]	5	Peripheral drivers	3.16 pJ	15	1.3 ns	61.62 pJ.ns
MAGIC [44]	15	Peripheral drivers	0.68 pJ	13	1.12 ns	9.94 pJ.ns
MAGIC (Naive mapping) [24]	15	Peripheral drivers	0.68 pJ	12	1.43 ns	11.66 pJ.ns
MAGIC (Compact mapping)[24]	24	Peripheral drivers	0.89 pJ	16	1.43 ns	20.36 pJ.ns
IMPLY [45]	6	Peripheral drivers	-	23	5 ns	-
IMPLY (Semi-serial) [46]	8	Peripheral drivers	-	12	30 µs	-
IMPLY (Semi-parallel) [47]	5	Peripheral drivers	-	17	50 µs	-

In [9], an optimized implementation of an MRL based 1-bit full adder is proposed. The authors developed an algorithm to search for the best form of the Boolean functions of the sum (S) and carry (C). The desired form should lead to an implementation with the minimum possible number of CMOS inverters. The inverter positions are allocated in such a way that removes signal degradation. The proposed circuit design of the full adder in [9] has a smaller number of memristors as well as CMOS transistors, with reductions of 11.1% and 33.3%, respectively, compared to our proposed design. However, the obtained logic function in [9] is not in the form of SoP. Thus, it is not possible to allocate memristors in a crossbar structure. This leads to more wiring at the fabrication stage, which in turn increases the implementation area dramatically. As regards energy consumption, the values reported in [9] are in the normalized form; thus, they cannot be used for comparison.

In [42], the authors presented a hybrid memristor–CMOS-based full adder circuit based on MRL. The adder is comprised of 18 memristors and 8 MOSFETs, which corresponds to a 66.6% reduction in the number of MOSFETs compared to our proposed design. However, the energy and the step delay in [42] are much higher compared to our presented X-MRL design. The layout is not considered by the authors.

In [43], a design for a 1 bit full adder was proposed based on memristor MAGIC-NOR and NOT gates. A crossbar structure was adopted and several optimization techniques were used to minimize the number of rows and columns of the crossbar as well as the number of computational steps. It has been shown that a compromise exists between the size of the crossbar and the necessary number of steps to perform a full addition. A minimum size of 3×3 crossbars (i.e., nine memristors) with a total latency of 35 computational steps is achieved. In contrast, our proposed design uses 18 memristors distributed in a crossbar structure in addition to nine CMOS inverters. The output is evaluated in one computational step. Concerning energy consumption, the proposed design in [43] consumes 0.3 pJ to achieve a 1 bit full addition process, whereas our proposed design consumes 0.69 pJ.

In [23], an N-bit addition was performed using MAGIC operations (i.e., NOR and NOT gates). Several approaches were presented by the authors for realizing logic within crossbars. The best of these approaches in terms of latency corresponded to $10N + 3$ computational steps, which leads to 13 clock cycle for the case of a 1 bit full adder. However, $13N - 3$ memristors are reserved (i.e., 10 memristors for $N = 1$) to accomplish the 1 bit addition process. For the purpose of minimizing the number of reserved memristors inside the crossbar, an area optimized crossbar structure was also proposed in [23]. Only five memristors were utilized; however, $15N$ (i.e., 15 for $N = 1$) computational steps were required to achieve 1-bit full addition. As a result, our proposed design, which requires one computational step, outperforms the designs presented in [23] in terms of latency. Regarding the energy consumption, all the proposed approaches in [23] have almost the same energy dissipation, which is about 3.16 pJ for the case of $N = 1$. Thus, the proposed design in [23] consumes 4.5 times more energy than our proposed design.

In [44], an N-bit ripple carry adder (RCA) circuit in a memristor crossbar structure was presented. The MAGIC design style was used to implement the logic gates. By considering $N = 1$, which is the case of 1 bit addition, the proposed crossbar MAGIC-based design requires 15 memristors and can perform the addition operation in 13 clock cycles. Compared to our proposed design, the adder design in [44] needs 13 times more clock cycles to perform addition operation, while it requires three fewer memristors to be implemented. On the other hand, our design consumes 1.01 times more energy than the proposed design in [44].

In [24], logic operations were realized by two methods using MAGIC. The first method corresponds to a naive mapping: it maps the NOR/NOT netlist into a single row of the crossbar. For the case of 1 bit full addition, 12 NOT/NOR sequential operations were required for a total number of 15 memristors. The overall energy consumption is estimated as 0.68 pJ. The second method corresponds to the compact mapping: in this method, NOR/NOT MAGIC operations are performed on rows and columns of a crossbar to realize logic functions. A 1 bit full addition process is performed on an 8×3 crossbar structure

(i.e., 24 memristors) and requires 16 computational steps. The overall energy consumption is evaluated as 0.89 pJ. Compared to our design, the naive mapping and the compact mapping consume 1.01 times less and 1.28 times more energy, respectively.

In [45], the authors proposed a 1 bit full adder that was designed using IMPLY logic. The proposed design needs 23 computational steps to perform the addition. The 1 bit full adder proposed in [45] requires six memristors, which is 33.3% of the memristors utilized in our design. However, the IMPLY logic design approach adopts three different voltage levels (V_{COND}, V_{SET} and V_{CLEAR}). Thus, additional circuitry such as analog multiplexers should be added to drive the allocated memristors. This induces an overhead in terms of the total utilized area when compared to our proposed design. Note that the energy consumption was not considered by the authors.

In [46], the authors proposed an IMPLY-based semi-serial adder with a respective addition algorithm. The N-bit full adder is implemented using $2N + 6$ memristors, which correspond to eight memristors for a 1 bit full adder. Compared to our proposed X-MRL design, the authors use 55.5% fewer memristors. The N-bit addition in [46] is completed within $10N + 2$ steps, which correspond to 12 steps for 1 bit addition. Each step requires 30 µs to be completed. Thus, our proposed X-MRL full adder design, which requires one computational step (2.5 ns in total), outperforms that in [46] in terms of latency.

The authors of [47] presented the design of a semi-parallel adder based on IMPLY. As compared to the semi-serial adder mentioned above, the semi-parallel adder reduces the number of memristors to five, but this comes at the cost of an increased number of computational steps and step delay. The full adder design in [47] uses a smaller number of memristors compared to our proposed design. However, it requires a higher number of steps and larger step delay.

Table 3 summarizes the comparison results presented above. The table illustrates the key advantage of the proposed approach regarding the reduced number of computational steps with respect to other existing designs. The energy consumption remains comparable. The Energy.Delay metric is used for a global direct evaluation. This metric combines both delay and energy consumption. As shown in the table, our proposed design outperforms all existing related ones. The improvement in Energy.Delay is between $\times 5.7$ and $\times 31$.

On the other hand, for the works that have adopted MAGIC and IMPLY in [23,24,43–45], the initialization and the evaluation of the rows and columns of the memristive crossbar require a separate CMOS controller. Moreover, a conversion mechanism is required in these designs. This mechanism includes a sensing amplifier to convert the resulting stored bits from the resistance state to the voltage state [8]. These additional peripheral drivers result in additional overheads in area and power consumption.

7. Conclusions

In this paper, an MRL-based crossbar design—namely, X-MRL—is proposed. The X-MRL approach is dedicated to the implementation of combinational logic. The design methodology of X-MRL efficiently integrates memristors with CMOS devices to improve density and scalability. Using X-MRL, a Boolean function is represented using pairs of memristors mapped efficiently into a crossbar structure. The obtained memristive crossbar is stacked at the top of the CMOS layer. For evaluation purposes, we designed a hybrid memristor–CMOS full adder based on the X-MRL approach. Based on a realistic memristor parameter model and CMOS 65 nm process, the design was simulated in the Cadence Virtuoso environment. The obtained layout of the full adder showed a 44.79% area reduction compared to that implemented with pure CMOS technology. Moreover, the Energy.Delay metric was used for comparison. This revealed a significant improvement (between $\times 5.7$ and $\times 31$) with respect to the available literature. As future work, the proposed X-MRL design may be considered for the implementation of flexible logic blocks.

Author Contributions: Conceptualization, K.A.A.; methodology, K.A.A. and A.B.; design, K.A.A.; software, K.A.A.; validation, K.A.A. and A.B.; investigation, A.B., M.R., J.-P.D. and J.J.; writing—original draft preparation, K.A.A.; writing—review and editing, M.R., A.B. and J.-P.D.; supervision, M.R., A.B., J.-P.D. and J.J.; All authors have read and agreed to the published version of the manuscript.

Funding: This research was partially funded by Carnot TSN—IMT Atlantique through the FLEXDEM project.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Chua, L. Memristor-The missing circuit element. *IEEE Trans. Circuit Theory* **1971**, *18*, 507–519. [CrossRef]
2. Strukov, D.B.; Snider, G.S.; Stewart, D.R.; Williams, R.S. The missing memristor found. *Nature* **2008**, *453*, 80–83. [CrossRef] [PubMed]
3. Mladenov, V. Analysis of memory matrices with HfO2 memristors in a PSpice environment. *Electronics* **2019**, *8*, 383. [CrossRef]
4. Hamdioui, S.; Du Nguyen, H.A.; Taouil, M.; Sebastian, A.; Le Gallo, M.; Pande, S.; Schaafsma, S.; Catthoor, F.; Das, S.; Redondo, F.G.; et al. Applications of computation-in-memory architectures based on memristive devices. In Proceedings of the Design, Automation & Test in Europe Conference & Exhibition (DATE), Florence, Italy, 25–29 March 2019; pp. 486–491.
5. Kvatinsky, S.; Belousov, D.; Liman, S.; Satat, G.; Wald, N.; Friedman, E.G.; Kolodny, A.; Weiser, U.C. MAGIC—Memristor-aided logic. *IEEE Trans. Circuits Syst. II Express Briefs* **2014**, *61*, 895-899. [CrossRef]
6. Kvatinsky, S.; Satat, G.; Wald, N.; Friedman, E.G.; Kolodny, A.; Weiser, U.C. Memristor-based material implication (IMPLY) logic: Design principles and methodologies. *IEEE Trans. VLSI Syst.* **2014**, *22*, 2054–2066. [CrossRef]
7. Reuben, J.; Ben-Hur, R.; Wald, N.; Talati, N.; Ali, A.H.; Gaillardon, P.E.; Kvatinsky, S. Memristive logic: A framework for evaluation and comparison. In Proceedings of the 27th International Symposium on Power and Timing Modeling, Optimization and Simulation (PATMOS), Thessaloniki, Greece, 25–27 September 2017.
8. Kvatinsky, S.; Wald, N.; Satat, G.; Kolodny, A.; Weiser, U.C.; Friedman, E.G. MRL—Memristor ratioed logic. In Proceedings of the International Workshop on Cellular Nanoscale Networks and their Applications, Turin, Italy, 29–31 August 2012.
9. Liu, B.; Wang, Y.; You, Z.; Han, Y.; Li, X. A signal degradation reduction method for memristor ratioed logic (MRL) gates. *IEICE Electron. Express* **2015**, *12*, 20150062. [CrossRef]
10. Chowdhury, J.; Das, K.; Rout, K. Implementation Of 24T Memristor Based Adder Architecture With Improved Performance. *Int. J. Electr. Electron. Data Commun.* **2015**, *3*, 91–94.
11. Cho, K.; Lee, S.J.; Eshraghian, K. Memristor-CMOS logic and digital computational components. *Microelectron. J.* **2015**, *46*, 214–220. [CrossRef]
12. Ali, K.A.; Rizk, M.; Baghdadi, A.; Diguet, J.P.; Jomaah, J. MRL Crossbar-Based Full Adder Design. In Proceedings of the 26th IEEE International Conference on Electronics, Circuits and Systems (ICECS), Genoa, Italy, 27–29 November 2019; pp. 674–677.
13. Pickett, M.D.; Strukov, D.B.; Borghetti, J.L.; Yang, J.J.; Snider, G.S.; Stewart, D.R.; Williams, R.S. Switching dynamics in titanium dioxide memristive devices. *J. Appl. Phys.* **2009**, *106*, 074508. [CrossRef]
14. Yang, J.J.; Pickett, M.D.; Li, X.; Ohlberg, D.A.; Stewart, D.R.; Williams, R.S. Memristive switching mechanism for metal/oxide/metal nanodevices. *Nat. Nanotechnol.* **2008**, *3*, 429–433. [CrossRef]
15. Lehtonen, E.; Laiho, M. CNN using memristors for neighborhood connections. In Proceedings of the International Workshop on Cellular Nanoscale Networks and their Applications (CNNA),Berkeley, CA, USA, 3–5 February 2010.
16. Simmons, J.G. Generalized Formula for the Electric Tunnel Effect between Similar Electrodes Separated by a Thin Insulating Film. *J. Appl. Phys.* **1963**, *34*, doi:10.1063/1.1702682. [CrossRef]
17. Kvatinsky, S.; Friedman, E.G.; Kolodny, A.; Weiser, U.C. TEAM: Threshold adaptive memristor model. *IEEE Trans. Circuits Syst. I Regul. Pap.* **2013**, *60*, 211–221. [CrossRef]
18. Kvatinsky, S.; Ramadan, M.; Friedman, E.G.; Kolodny, A. VTEAM: A general model for voltage-controlled memristors. *IEEE Trans. Circuits Syst. II Express Briefs* **2015**, *62*, 786–790. [CrossRef]
19. Kvatinsky, S.; Kolodny, A.; Weiser, U.C.; Friedman, E.G. Memristor-based IMPLY logic design procedure. In Proceedings of the IEEE International Conference on Computer Design (ICCD), Amherst, MA, USA, 9–12 October 2011.
20. Bickerstaff, K.; Swartzlander, E.E. Memristor-based arithmetic. In Proceedings of the Conference Record of the Asilomar Conference on Signals, Systems and Computers, Pacific Grove, CA, USA, 7–10 November 2010.
21. Rahman, K.C.; Hammerstrom, D.; Li, Y.; Castagnaro, H.; Perkowski, M.A. Methodology and design of a massively parallel memristive stateful IMPLY logic-based reconfigurable architecture. *IEEE Trans. Nanotechnol.* **2016**, *15*, 675–686. [CrossRef]
22. Hur, R.B.; Kvatinsky, S. Memristive memory processing unit (MPU) controller for in-memory processing. In Proceedings of the IEEE International Conference on the Science of Electrical Engineering (ICSEE), Eilat, Israel, 16–18 November 2016.
23. Talati, N.; Gupta, S.; Mane, P.; Kvatinsky, S. Logic design within memristive memories using memristor-aided loGIC (MAGIC). *IEEE Trans. Nanotechnol.* **2016**, *15*, 635–650. [CrossRef]
24. Gharpinde, R.; Thangkhiew, P.L.; Datta, K.; Sengupta, I. A Scalable In-Memory Logic Synthesis Approach Using Memristor Crossbar. *IEEE Trans. VLSI Syst.* **2018**, *26*, 355–366. [CrossRef]

25. Teimoori, M.; Ahmadi, A.; Alirezaee, S.; Ahmadi, M. A novel hybrid CMOS-memristor logic circuit using Memristor Ratioed Logic. In Proceedings of the IEEE Canadian Conference on Electrical and Computer Engineering (CCECE), Vancouver, BC, Canada, 15–18 May 2016.
26. Williams, R.S. Finding the missing memristor. *Keynote Talk at UC San Diego CNS Winter*, California, **2010**.
27. Xia, Q.; Robinett, W.; Cumbie, M.W.; Banerjee, N.; Cardinali, T.J.; Yang, J.J.; Wu, W.; Li, X.; Tong, W.M.; Strukov, D.B.; et al. Memristor- CMOS hybrid integrated circuits for reconfigurable logic. *Nano Lett.* **2009**, *9*, 3640–3645. [CrossRef]
28. Yang, J.J.; Strukov, D.B.; Stewart, D.R. Memristive devices for computing. *Nat. Nanotechnol.* **2013**, *8*, 13–24. [CrossRef]
29. Tsunoda, K.; Kinoshita, K.; Noshiro, H.; Yamazaki, Y.; Iizuka, T.; Ito, Y.; Takahashi, A.; Okano, A.; Sato, Y.; Fukano, T.; et al. Low power and high speed switching of Ti-doped NiO ReRAM under the unipolar voltage source of less than 3V. In Proceedings of the IEEE International Electron Devices Meeting (IEDM), Washington, DC, USA, 10–12 December 2007.
30. Kwon, D.H.; Kim, K.M.; Jang, J.H.; Jeon, J.M.; Lee, M.H.; Kim, G.H.; Li, X.S.; Park, G.S.; Lee, B.; Han, S.; et al. Atomic structure of conducting nanofilaments in TiO 2 resistive switching memory. *Nat. Nanotechnol.* **2010**, *5*, 148–153. [CrossRef]
31. Wang, Z.; Joshi, S.; Savel'ev, S.E.; Jiang, H.; Midya, R.; Lin, P.; Hu, M.; Ge, N.; Strachan, J.P.; Li, Z.; et al. Memristors with diffusive dynamics as synaptic emulators for neuromorphic computing. *Nat. Mater.* **2017**, *16*, 101–108. [CrossRef]
32. Guan, X.; Yu, S.; Wong, H.S.P. A SPICE compact model of metal oxide resistive switching memory with variations. *IEEE Electron Device Lett.* **2012**, *33*, 1405–1407. [CrossRef]
33. Wang, Y.; Zhang, Y.; Deng, E.; Klein, J.O.; Naviner, L.A.; Zhao, W. Compact model of magnetic tunnel junction with stochastic spin transfer torque switching for reliability analyses. *Microelectron. Reliab.* **2014**, *54*, 1774–1778. [CrossRef]
34. Wang, Y.; Cai, H.; Naviner, L.A.; Zhang, Y.; Klein, J.O. Compact thermal modeling of spin transfer torque magnetic tunnel junction. *Microelectron. Reliab.* **2015**, *55*, 1649–1653. [CrossRef]
35. Lee, H.; Chen, Y.; Chen, P.; Gu, P.; Hsu, Y.; Wang, S.; Liu, W.; Tsai, C.; Sheu, S.; Chiang, P.; et al. Evidence and solution of over-RESET problem for HfO_x based resistive memory with Sub-ns switching speed and high endurance. In Proceedings of the IEEE International Electron Devices Meeting, San Francisco, CA, USA, 2 March 2010.
36. Chanthbouala, A.; Garcia, V.; Cherifi, R.O.; Bouzehouane, K.; Fusil, S.; Moya, X.; Xavier, S.; Yamada, H.; Deranlot, C.; Mathur, N.D.; et al. A ferroelectric memristor. *Nat. Mater.* **2012**, *11*, 860–864. [CrossRef]
37. Kang, J.; Huang, P.; Gao, B.; Li, H.; Chen, Z.; Zhao, Y.; Liu, C.; Liu, L.; Liu, X. Design and application of oxide-based resistive switching devices for novel computing architectures. *IEEE J. Electron Devices Soc.* **2016**, *4*, 307–313. [CrossRef]
38. Baek, I.; Lee, M.; Seo, S.; Lee, M.; Seo, D.; Suh, D.S.; Park, J.; Park, S.; Kim, H.; Yoo, I.; et al. Highly scalable nonvolatile resistive memory using simple binary oxide driven by asymmetric unipolar voltage pulses. In Proceedings of the IEEE International Electron Devices Meeting (IEDM), San Francisco, CA, USA, 13–15 December 2004.
39. Diokh, T.; Le-Roux, E.; Jeannot, S.; Gros-Jean, M.; Candelier, P.; Nodin, J.; Jousseaume, V.; Perniola, L.; Grampeix, H.; Cabout, T.; et al. Investigation of the impact of the oxide thickness and RESET conditions on disturb in HfO_2-RRAM integrated in a 65 nm CMOS technology. In Proceedings of the IEEE International Reliability Physics Symposium (IRPS), Monterey, CA, USA, 14–18 April 2013.
40. Lee, H.; Chen, P.; Wu, T.; Chen, Y.; Wang, C.; Tzeng, P.; Lin, C.; Chen, F.; Lien, C.; Tsai, M.J. Low power and high speed bipolar switching with a thin reactive Ti buffer layer in robust HfO_2 based RRAM. In Proceedings of the IEEE International Electron Devices Meeting (IEDM), San Francisco, CA, USA, 15–17 December 2008.
41. Biolek, Z.; Biolek, D.; Biolkova, V. SPICE Model of Memristor with Nonlinear Dopant Drift. *Radioengineering* **2009**, *18*, 210–214.
42. Mandal, S.; Sinha, J.; Chakraborty, A. Design of Memristor—CMOS based logic gates and logic circuits. In Proceedings of the 2nd International Conference on Innovations in Electronics, Signal Processing and Communication (IESC), Shillong, India, 1–2 March 2019; pp. 215–220.
43. Thangkhiew, P.L.; Gharpinde, R.; Chowdhary, P.V.; Datta, K.; Sengupta, I. Area efficient implementation of ripple carry adder using memristor crossbar arrays. In Proceedings of the International Design & Test Symposium (IDT), Hammamet, Tunisia, 18–20 December 2016.
44. Thangkhiew, P.; Gharpinde, R.; Yadav, D.N.; Datta, K.; Sengupta, I. Efficient implementation of adder circuits in memristive crossbar array. In Proceedings of the IEEE Region 10 Conference (TENCON), Penang, Malaysia, 5–8 November 2017.
45. Teimoory, M.; Amirsoleimani, A.; Shamsi, J.; Ahmadi, A.; Alirezaee, S.; Ahmadi, M. Optimized implementation of memristor-based full adder by material implication logic. In Proceedings of the IEEE International Conference on Electronics, Circuits and Systems (ICECS), Marseille, France, 7–10 December 2014.
46. TaheriNejad, N.; Delaroche, T.; Radakovits, D.; Mirabbasi, S. A semi-serial topology for compact and fast IMPLY-based memristive full adders. In Proceedings of the 17th IEEE International New Circuits and Systems Conference (NEWCAS), Munich, Germany, 23–26 June 2019; pp. 1–4.
47. Rohani, S.G.; Taherinejad, N.; Radakovits, D. A Semiparallel Full-Adder in IMPLY Logic. *IEEE Trans. Very Large Scale Integr. Syst.* **2019**, *28*, 297–301. [CrossRef]

6.5 Memristor Overwrite Logic (MOL) for In-Memory Computing

6.5.1 Preface

This research work aims to explore new in-memory computing approaches and computational memory architectures that allow efficient combination of storage and processing in order to bypass the memory wall problem and thus to improve the computational efficiency; thereafter, apply the proposed techniques and architectures in real application case studies for the sake of performance evaluation. This work has been initiated during the PhD thesis of Khaled Alhaj ALI and has been continued during his work as a post-doctoral fellow at IMT Atlantique, Brest, France. It has been valorized in two IEEE international conferences [133] [134] and two IEEE transactions [135] [136].

6.5.2 Introduction

Modern systems have to implement additional memory in order to cope with the ever increasing requirement of massive data storage. The aggressive growth in the size of processed data in addition to the increasing numbers of processing cores have placed a high demand on the currently used memory systems as well as to intensive data traffic between memory and processing cores. In-memory computing (IMC) have been introduced to overcome the memory wall problem. Instead of sending large amount of data to the processing cores, part of the tasks are computed inside the memory. This reduces the memory accesses bottleneck and can significantly improve performance.

Emerging memristive technologies such as the ReRAM or MRAM have been considered as promising candidates for the next generation of memories. Memristive memory systems have several desirable attributes. They are classified as non-volatile with near-zero standby power consumption. They have ultra high density. Recent studies have illustrated the ability to perform local computations inside memristive memories (crossbars). Computing within memristive memories is motivated by the unique properties of memristors and their versatile nature. In this context, several recent contributions have been proposed to enable computation within memristive memory arrays. [121, 137, 138, 139, 132, 140, 141, 142, 143, 144]. However, several challenges and limitations exist in terms of manufacturability and computational accuracy regarding device variability, pattern-dependent current leakage and the area overhead of peripheral circuits, and resistance drift effects.

The nonvolatile internal resistance state of memristor could be changed according to the magnitude and duration of the applied bias across its terminals [103]. If sufficient magnitude and duration of the bias across the memristor terminals is guaranteed, the intermediate resistances could be ignored. Accordingly, a memristor could be considered as a two-state element ($R \in \{R_{ON}, R_{OFF}\}$). The mapping of the internal resistance between the terminals of the memristor into Boolean states allows the definition of memristor in digital domain.

Based on this understanding, MOL new logic design style, is proposed for performing logic operations. In this logic design style, the result of OR/AND logic operation is over-

written into the internal state of the memristor. The latter acts either as logic accumulator with its previously stored bit or as logic operator between its two terminals. This novel logic design style is inspired from a digital representation of memristors. Unlike existing approaches, MOL can operate with different memristor technologies, regardless the LRS-HRS margin and with linear as well as threshold-type memristive devices. Furthermore, the proposed original computational memory architecture, with appropriate drivers and control sequences, allows the execution of numerous logic operations, at bit or vector-level, in one or two computational steps at most.

6.5.3 Context and Motivation

Several works have been carried out revealing the ability to execute logic operations inside memristive memory arrays. The Memristor Aided Logic (MAGIC) [128] and the statefull implication (IMPLY) [121] have been introduced as possible solutions for the realization of logic computations inside these memories. Based on MAGIC and IMPLY, storage and processing are both allowed within the same cells of the memristive memory array. Although, these approaches are used as basis logic functions to execute arbitrary Boolean functions inside memristive crossbar arrays, both approaches induce the state drift phenomena. The corresponding states of the memristive cells performing computations are not fully digital in some cases. On the other hand, the performance of these design styles is highly dependent on the technology of the adopted memristive device. Moreover, the corresponding basis functions provided by IMPLY and MAGIC are not diverse enough to allow fast logic mapping with minimum computational cycles. More recently, other in-memory computing techniques have emerged as alternatives. Among these, the memristor-based majority (MAJ) [145] has been introduced to overcome the aforementioned limitations. However, other downsides arise at the architectural level. MAJ design style is relatively complex in terms of peripheral circuits as well as excessive in-out data movement which in turn impacts latency.

In this research work, a novel MOL logic design style is introduced associated with an original MOL-based computational memory. As MOL approach is based on our proposed digital representation of memristive devices [103], the proposed MOL-based computation is fully digital and eligible for highly reliable applications. MOL combines the simplicity of MAGIC/IMPLY [121] techniques and the accuracy of MAJ [122].

Moreover, MOL can operate with different memristive device technologies and allows for significant reduction in the number of required memristors and computational steps. The choice of the memristive device type is not constrained by a specified MOL requirement. It can be observed from the mechanism of MOL technique that it involves direct access to the terminals of memristive devices which highly resembles conventional write operation. During the operation of MOL, the potential difference between the terminals of the memristive device always attains a binary level. Accordingly, MOL can be implemented in a wide range of memristive memories without specifying particular device features. In contrast, the structure of pre-existing logic design styles either establishes a series connection of a resistor (e.g. IMPLY) or series connection of the memristive devices (e.g. MAGIC) for normal operation. This undoubtedly prevents direct access to the memristive device terminals and consequently imposes specific device constraints.

6.5.4 Contributions and Performed Work

This work proposes a MOL-based computational memory architecture. The proposed original architecture, based on coupling two conventional crossbar arrays, is able to perform bitwise AND/OR operations between two stored words. The proposed MOL-memory architecture can be simply configured between storage (memory) and computation (processing) modes.
The main contributions of this work are summarized in the following:

- A new logic design style, namely Memristor Overwrite Logic (MOL), is proposed for performing logic operations. In this logic design style, the result of OR/AND logic operation is overwritten into the internal state of the memristor. The latter acts either as logic accumulator with its previously stored bit or as logic operator between its two terminals.

- Application of the approach to perform MOL inside memristive crossbar arrays. An original feature in this context concerns the ability to perform MOL on a vector of bits and the possibility to select a single or multiple rows or columns of the crossbar.

- Integration of MOL into conventional crossbar-memory architecture, with incoming data bits along bit-lines only. The proposed architecture is able to perform MOL operations in addition to its main storage function. Four supported modes are detailed (write, overwrite, read, idle) together with appropriate architectures for the memory peripheral drivers. This research work has been initially published in the proceedings of the *IEEE International Conference on Electronics, Circuits and Systems (ICECS)* [133].

- Proposal of a MOL-based computational memory architecture which is able to perform MOL operations between two stored words, rather than between one stored word and external arriving input bits. An original architecture with two coupled MOL memory blocks that work in complementary manner is proposed together with appropriate drivers and control sequences. Numerous logic computations can be performed by this architecture, each requiring only one computational step.

- Demonstration of the design methodology through a detailed case study of N-bit full addition implemented using the proposed MOL-based computational memory. The sequence of MOL operations is illustrated in time and space (mapping to the two coupled MOL memory blocks). Different configurations are analyzed and evaluated in terms of required number of computational steps. This research work has been published in the *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* [135].

- Performance analysis of the proposed MOL memory architecture design using Magnetic Tunnel Junction (MTJ) device [146][147]and CMOS 65nm technology node. This includes timing analysis, area utilization, energy consumption and tolerance against device variability, in addition to a detailed comparison with existing approaches. The obtained results show significant reductions in terms of latency and area when compared to existing recent approaches [137, 121, 138, 141, 140, 132, 148].

- Investigate the utility of MOL approach in the field of Neural Networks. An architecture design targeting Deep Neural Network (DNN) computation has been presented to address the inefficiency of moving data between memory and processing cores which is time and energy consuming [149]. The design is composed of interconnected computational memory blocks namely CMEMs. The design is able to execute the weighted accumulation process purely inside the storage cells. This work has been published in the the *IEEE International Symposium on Circuits and Systems (ISCAS)* [134].

- Employment of the computational memories based on MOL design style to perform in-memory computing in the emerging Binary Neural Networks (BNN) [150]. A novel architecture is proposed to execute instructions inside multiple computational memories using the proposed SIMM parallelism model which stands for single-instruction multiple-memory. The architecture employs the advanced quantization algorithm of BNN and the promising MOL-based in-memory computing technique which is well adapted for parallel bit-wise operations. This research work has been published in the *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* [136].

Memristive Computational Memory Using Memristor Overwrite Logic (MOL)

Khaled Alhaj Ali, *Graduate Student Member, IEEE*, Mostafa Rizk, Amer Baghdadi, *Senior Member, IEEE*, Jean-Philippe Diguet, *Senior Member, IEEE*, Jalal Jomaah, Naoya Onizawa, *Member, IEEE*, and Takahiro Hanyu, *Senior Member, IEEE*

***Abstract*— In this article, we present a novel logic design style, namely, memristor overwrite logic (MOL), associated with an original MOL-based computational memory. MOL relies on a fully digital representation of memristor and can operate with different memristive device technologies. Its integration in memristive crossbar arrays and computational memories allows the execution of bit and vector-level primitive logic operations in two computational steps at most. Promising features and performances are demonstrated through the implementation of *N*-bit full addition using the proposed MOL-based computational memory.**

***Index Terms*— Crossbar array, in-memory computation, logic design, memristor, memristor overwrite logic (MOL).**

I. Introduction

MEMRISTOR has been predicted theoretically by Chua [1] in 1971. Chua hypothesized that memristor which is the fourth passive device should exist and hold a relationship between magnetic flux and charge. The first fabrication of a memristor device has been developed by a research team at Hewlett-Packard (HP) Labs [2] in 2008. The device structure is comprised of a stoichiometric (TiO_2) and an oxygen-deficient (TiO_{2-x}) layer sandwiched between two platinum electrodes. The obtained two-terminal nanodevice exhibits a dynamic resistance that can be modulated between two bounds. These bounds correspond to the low and high resistance states, and are referred to as R_{on} and R_{off}, respectively. Memristor possesses the ability to retain the last attained resistance value in a nonvolatile manner.

Given the nanoscale dimensions of memristors and their unique properties, several innovative applications have emerged. One of the most promising applications is the use of memristors to implement arithmetic blocks, such as full adders [3]–[5]. Compared to pure CMOS implementations, these blocks are of relatively high density and could be packed into small chip area. Other applications involve using memristor-based memories to allow processing within the storage cells. This approach, referred to as in-memory computing, is being explored recently to alleviate the time and energy cost of data movement encountered in conventional von Neumann architecture and aggravated by the recent growth in data-centric applications [6]. The concept is different from, yet can be complementary to, that of near-memory computing which dates to the 1990s [7]. The near-memory computing approach aims to place the processing units physically closer to the memory, for example, through advanced die stacking technologies or 3-D integration. Despite the reduction in time and distance to memory access, there still exists a physical separation between the memory and the compute units [8]. For in-memory computing, instead of sending a large amount of data to the processing cores, part of the tasks are computed in place inside the memory itself [9]. Depending on the application, this can reduce the computational complexity of these tasks and/or the amount of data being accessed, leading to significant performance improvement [6].

In this context, several recent contributions have been proposed to enable computation within memristive memory arrays and can be classified into two categories. The first category involves using the memristor as a single-level cell (SLC) [10]–[16]. The second category includes work that uses the memristor as a multilevel cell (MLC) or analog cell [17]–[19]. The MLC-based computing is promising when targeting applications with intensive multiply–accumulate operations, such as convolutional neural networks (CNNs) [19]. However, a number of challenges remain in terms of manufacturability and computational accuracy regarding device variability, pattern-dependent current leakage, and the area overhead of peripheral circuits [20]. Major semiconductor foundries have not included MLC technology in their development roadmaps in the near future [19]. In contrast, SLC cells have a larger readout margin that makes them tolerant against process variation and resistance drift effects. Based on SLCs, different logic design styles have been introduced together with different realizations on memristive crossbar arrays. The Material Implication (IMPLY) [10] and the Memristor Aided loGIC (MAGIC) [21] have been introduced to enable in-memory logic operations. Although promising results are demonstrated, MAGIC and IMPLY techniques still impose specific technology and

Manuscript received February 26, 2020; revised May 11, 2020 and June 17, 2020; accepted July 10, 2020. This work was presented at the 26th IEEE International Conference on Electronics, Circuits and Systems (ICECS), November 2019, Genoa, Italy. *(Corresponding author: Khaled Alhaj Ali.)*

Khaled Alhaj Ali and Amer Baghdadi are with IMT Atlantique, CNRS Lab-STICC Laboratory, 29238 Brest, France (e-mail: khaled.alhaj-ali@imt-atlantique.fr).

Mostafa Rizk is with IMT Atlantique, 29285 Brest, France, also with the School of Engineering, International University of Beirut, Beirut 1105, Lebanon, and also with the Physics Department, Faculty of Sciences, Lebanese University, Beirut 1003, Lebanon.

Jean-Philippe Diguet is with the CNRS Lab-STICC Laboratory, 56100 Lorient, France.

Jalal Jomaah are with the Physics Department, Faculty of Sciences, Lebanese University, Beirut 1003, Lebanon.

Naoya Onizawa and Takahiro Hanyu are with the Research Institute of Electrical Communication, Tohoku University, Sendai 980-8577 Japan.

Color versions of one or more of the figures in this article are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TVLSI.2020.3011522

design constraints. For instance, in order to attain acceptable performance in these techniques, the ratio $R_{\text{off}}/R_{\text{on}}$ of the adopted memristive devices should be relatively high. Moreover, Fang and Tang [22] have reported that IMPLY does not ensure binary resistance switching of memristors in some cases. More recently, other in-memory computing techniques have emerged as alternatives. Among these, the memristor-based majority (MAJ) [23] has been introduced to overcome the aforementioned limitations. However, other downsides arise at the architectural level. MAJ design style is relatively complex in terms of peripheral circuits as well as excessive in-out data movement which in turn impacts latency.

In this work, we introduce a novel logic design style, namely, memristor overwrite logic (MOL), associated with an original MOL-based computational memory. MOL combines the simplicity of MAGIC/IMPLY techniques and the accuracy of MAJ. MOL can operate with different memristive device technologies and allows for a significant reduction in the number of required memristors and computational steps.

The rest of this article is organized as follows. Section II provides a brief survey on existing memristor-based logic design styles. Section III presents our proposed MOL approach. Section IV discusses the integration of MOL into the conventional memory configurations. Section V presents our proposed configurable MOL-based computational memory architecture. The design and its configuration methodology are demonstrated by a case study of a N-bit full addition in Section VI. Simulations and performance analysis are illustrated in Section VII. Comparison with available implementations is presented in Section VIII. Finally, Section IX concludes this article.

Although memristive devices encompass memristors, it is possible to use the term memristor for other memristor devices [24]. In this article, we use the term memristors and memristive devices interchangeably for simplicity.

II. Memristive Devices as Computational Elements

The versatile nature of memristors allows them to be used as computational elements in addition to their storage role. Implementing Boolean logic with memristors has been widely explored. Several memristor-based logic design styles have been introduced in the literature. Each is adapted for a specific type of application and surrounded by specific limitations.

A. Logic Design Styles

The memristor ratioed logic (MRL) has been proposed in [3]. MRL integrates memristors with CMOS transistors to implement combinational functional blocks. These blocks are relatively dense compared to those implemented with pure CMOS transistors. The memristive threshold logic (MTL) has been studied in [25]. The gate uses the configurable conductance of memristors to represent weights during operation. However, these weights are very sensitive to state drift, which can be a critical issue [25]. It is considered simple, but still in preliminary stages of fabrication. IMPLY [10] and MAGIC [21] are intended for in-memory computing. In these design styles, a memristor serves as a memory element as well as a part of a computational gate inside the memory. MAD gate, or memristors-as-drivers gate, has been presented in [26]. MAD has been introduced to overcome the long delays of the IMPLY operations as well as signal degradation and buffering issues in MRL; however, each MAD gate requires a complex driving circuitry and is thus considered unsuitable for integration inside a memristive memory. MAJ has been proposed in [23]. The authors demonstrated that a single memristor is capable of performing a 3-variable majority function. Using additional inversion function (INV), a Boolean expression is represented using a majority-inverter graph (MIG). MIGs are then realized sequentially in conventional memristive crossbar arrays. The complementary resistive switches (CRS) logic has been presented in [27]. CRS logic is capable of realizing two primitive operations denoted as reverse implication (RIMP) and inverse implication (NIMP). This logic design style can be considered as a special case of MAJ (see Section III-B).

In this article, our target application concerns in-memory computing, so some logic design styles such as MRL, MAD, and MTL are excluded.

B. Limitations

MAGIC and IMPLY logic families are widely explored in the literature. Talati *et al.* [12], Rahman *et al.* [28], Hur and Kvatinsky [29], Gharpinde *et al.* [30], and Thangkhiew *et al.* [31] have presented several approaches where logic functions are broken down into several MAGIC or IMPLY operations. These operations are then performed sequentially inside memristive crossbar arrays. However, these approaches have several design constraints.

1) The analysis in [22] shows that IMPLY cannot achieve the full resistance switching of the output memristor in case both input memristors of the IMPLY gate are in the R_{off} resistance state. Hence, the corresponding state of the output memristor is not fully digital.
2) Output memristors in IMPLY and MAGIC may be subjected to state drift [10], [12].
3) The performance of these design styles is highly dependent on the technology of the adopted memristive device (e.g., requirement of memristive devices with high $R_{\text{off}}/R_{\text{on}}$ ratio) [10], [12].
4) The corresponding basis functions provided by IMPLY and MAGIC are not diverse enough to allow fast logic mapping with minimum computational cycles.

MAJ-based logic design has been recently explored by Shirinzadeh *et al.* [15] and Gaillardon *et al.* [23]. MAJ relies on a digital representation of memristors, so the limitations faced in IMPLY and MAGIC can be overcome. However, at the architecture level, other downsides arise.

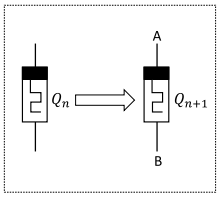
1) In-memory computing architectures based on MAJ, which are available in the literature, require additional load operations, which read data bits outside the memory. This induces the overheads in terms of the total critical path, number of cycles, and the complexity of the dedicated control unit.
2) Architectures based on MAJ involve significant modifications in the peripheral circuitry of the memory. The write operations are performed on bit-lines (BLs) as well as word-lines (WLs) instead of BLs only.

These limitations hold also for CRS logic design approach [27] as it can be considered as a special case of MAJ.

III. Proposed MOL Logic

In this section, we introduce a new memristor-based logic design style, namely, MOL. MOL approach is highly

(a)



(b)

Q_n	A	B	Q_{n+1}
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

Fig. 1. Memristor. (a) Internal state after applying external bias represented by A and B. (b) Truth table.

adapted for computing within memristive crossbar arrays and avoids the limitations encountered by preexisting logic design styles.

A. Digital Representation of Memristive Devices

The nonvolatile internal resistance state of memristor could be changed according to the magnitude and duration of the applied bias across its terminals [32]. However, a nonsufficient magnitude or duration leads to an intermediate resistance state R where $R_{\text{on}} < R < R_{\text{off}}$. In this case, the state of the memristor cannot be considered as binary, which in turn leads to more sophisticated modeling of the internal state of memristive devices in the analog domain. However, in a digital design we can represent the memristor as a two-state element where its resistance $R \in \{R_{\text{on}}, R_{\text{off}}\}$ and ignoring any other intermediate states if we succeed to guarantee a sufficient magnitude and duration of the bias across its terminals. Based on this understanding, the internal state of a memristor is defined in the digital domain. Let Q_n be the current internal state of a memristor while Q_{n+1} is the next state after applying a new external bias represented by A and B as shown in Fig. 1(a). Hence, Q_{n+1} will be a function of the logical states at terminals A and B and the previous internal state Q_n. By considering all the possible combinations of A, B, and Q_n as shown in Fig. 1(b), the state equation of a memristive device is expressed as follows:

$$Q_{n+1} = Q_n A + Q_n \overline{B} + A\overline{B} = M_3(A, \overline{B}, Q_n) \qquad (1)$$

where M_3 represents the 3-variable majority function, which is defined in [33]. This expression demonstrates that a majority function is an intrinsic feature of memristive devices [23]. Based on the Boolean expression presented in (1), the equivalent latch circuit of a memristive device is shown in Fig. 2 where Q is the internal state of the memristor. To translate the Boolean value of Q into a resistance between the terminals of the memristor, an analog multiplexer is added. It selects either one of the two resistors, whose resistances are R_{on} or R_{off}, where $Q = 0$ and $Q = 1$ are mapped to R_{on} and R_{off}, respectively. Note that this schematic is valid and useful from the digital perspective, so it cannot be used for simulation in the analog domain.

B. MOL Logic Procedure

The state representation of memristor expressed in (1) clarifies its computational capability and simplifies its integration in the digital domain. Six possible cases can be derived from (1) and are listed in (2). Fig. 3 is an illustration

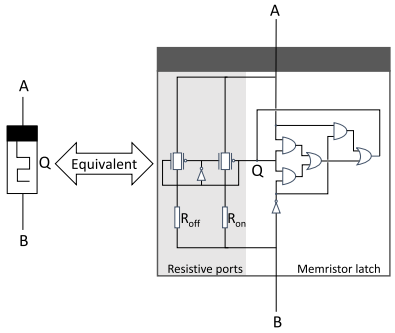


Fig. 2. Equivalent latch circuit of memristor with binary resistive ports.

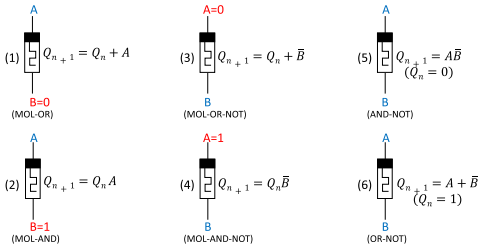


Fig. 3. Six possible logic cases performed by a memristor.

of these cases. They are split into two groups. The first group includes the cases from 1 to 4, which correspond to MOL. In these four cases, a memristor acts as a logic accumulator. The previously stored bit Q_n is subjected to OR/AND with the new input $A/\overline{B}$ while the other terminal of the memristor is set to logic "0" or logic "1" depending on the desired function. The obtained output is simultaneously saved in the form of a new internal state Q_{n+1}. The remaining cases (i.e., 5 and 6) are achieved by initializing the memristor to a known state (logic "0" or logic "1"). The inputs A and B are sent to the memristor ports simultaneously. The output is saved as the new internal state (Q_{n+1}) of the memristor. In fact, these two cases correspond to CRS logic operations that are explored in the literature [16], [23] [27].

Although MOL operations are special cases of the 3-variable majority, working with MOL is much simpler. MOL highly resembles the conventional write operation. One end of each memristor is reserved for the input operands, while the other end is employed for selection. In contrast, MAJ employs both terminals of the memristor for the input operands. This makes MOL more adapted to crossbar memory arrays

$$Q_{n+1} = \begin{cases} Q_n + A\,, & B = 0, \quad \text{case}: 1 \quad \text{(MOL)} \\ Q_n A\,, & B = 1, \quad \text{case}: 2 \quad \text{(MOL)} \\ Q_n + \overline{B}\,, & A = 0, \quad \text{case}: 3 \quad \text{(MOL)} \\ Q_n \overline{B}\,, & A = 1, \quad \text{case}: 4 \quad \text{(MOL)} \\ A\overline{B}\,, & Q_n = 0, \quad \text{case}: 5 \quad \text{(CRS)} \\ A + \overline{B}\,, & Q_n = 1, \quad \text{case}: 6 \quad \text{(CRS)}. \end{cases} \qquad (2)$$

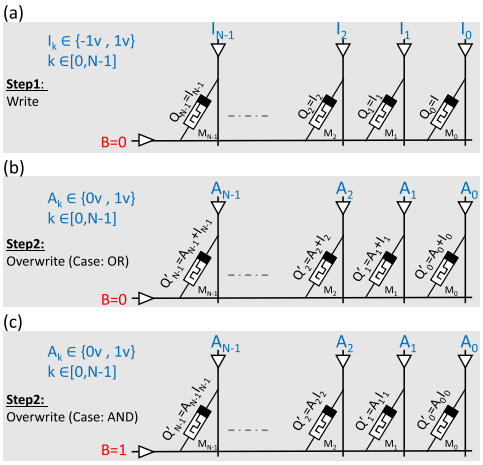


Fig. 4. Performing MOL on a vector of bits. (a) Writing N-bits into memristors. (b) Overwrite step for MOL-OR. (c) Overwrite step for MOL-AND.

The same concept applies to a vector of bits. Fig. 4 illustrates that two consecutive steps are enough for achieving MOL operations on an N-bit vector. In step 1, which is presented in Fig. 4(a), the input vector $I = [I_{N-1}, I_{N-2}, \ldots, I_1, I_0]$ is written into the N memristors by mapping logic "0" and logic "1" to the normalized voltage levels -1 and 1 V, respectively, while the common horizontal line is set to 0 V. At the end of this step, the resulting state of a given memristor M_k is $Q_k = I_k$. In step 2, the same N memristors are overwritten with the input vector $A = [A_{N-1}, A_{N-2}, \ldots, A_1, A_0]$. However, the input voltage level on the common horizontal line is set to 0 or 1 V depending on the desired operation. For the case of MOL-OR [Fig. 4(b)], B is set to 0 V and the result, which is stored in a given memristor M_k, is $Q'_k = A_k + I_k$. For the case of MOL-AND [Fig. 4(c)], B is set to 1 V and the result, which is stored in a given memristor M_k, is $Q'_k = A_k I_k$.

C. Performing MOL Inside Memristive Crossbars

The proposed MOL can be performed in memristive crossbar arrays. The input data bits to the crossbar can be either written or combined logically with the currently stored bits inside the crossbar. This can be simply achieved by choosing the appropriate normalized voltage levels for representing the arriving bits (i.e., -1/1 for write and 0/1 for MOL). Fig. 5(a) illustrates that a single or multiple rows of the crossbar could be selected for either MOL-OR or MOL-AND operations with the incoming data bits $I = [I_{N-1}, I_{N-2}, \ldots, I_1, I_0]$ being applied on the columns. Similarly, Fig. 5(b) shows that a single or multiple columns of the crossbar could be selected for either MOL-OR-NOT or MOL-AND-NOT operations with the incoming data bits of the vector I applied on the rows.

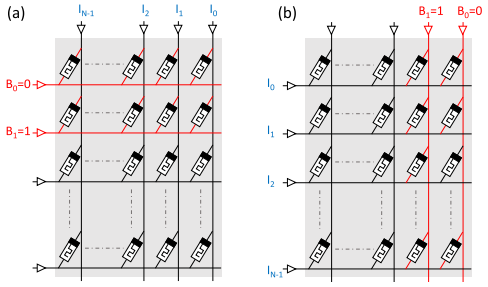


Fig. 5. MOL inside memristive crossbar. (a) MOL-OR or/and MOL-AND. (b) MOL-OR-NOT or/and MOL-AND-NOT.

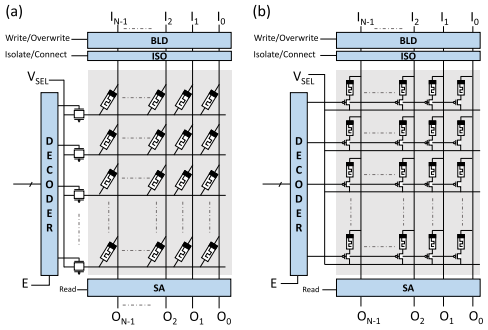


Fig. 6. Configurations of MOL memory architectures. (a) 1M. (b) 1T1M.

IV. Memory Architecture With MOL Capabilities

Crossbars constitute the core element of emerging memristive memories (e.g., RRAMs and MRAMs). Integrating MOL with crossbar-memory architectures can lead to promising enhancements and provides additional computational capabilities to these memories. However, this imposes updating memory peripheral drivers to cope with MOL operations in addition to its main storage function. Fig. 6(a) presents the proposed memory architecture which is capable of performing MOL. As illustrated in Section III, write and overwrite operations could be performed along the rows as well as the columns of the crossbar. However, in a conventional memory architecture, the flow of the incoming data bits is along the BLs only while the WLs are reserved for addressing. Thus, MOL operation, which is similar to a write operation, could be only performed along the BLs. In this case, MOL-OR and MOL-AND are the only supported logic operations in the proposed memory architecture. The architecture shown in Fig. 6(a) can be configured in four different modes:

1) *Write Mode:* The input N-bit vector $I = [I_{N-1}, I_{N-2}, \ldots, I_1, I_0]$ is first mapped via BL driver (BLD) into the normalized voltage levels of -1 and 1 V corresponding for logic "0" and "1," respectively. Fig. 7(a) presents the schematic of BLD at the transistor level. The respective voltage levels (-1 and 1 V) are then provided to the BLs of the memristive

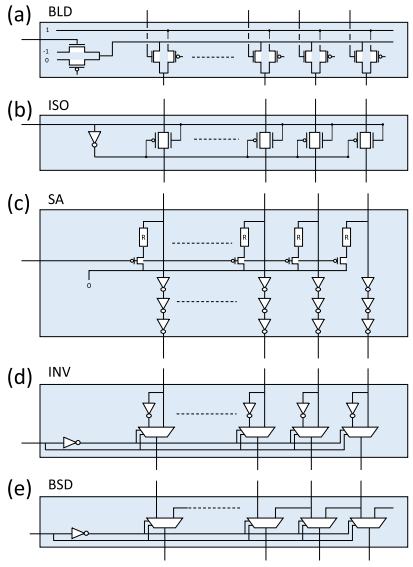


Fig. 7. Drivers architectures for the proposed MOL-memory approach.

crossbar through the Isolation Block (ISO), which acts in this mode as a connecting switch. Fig. 7(b) illustrates the internal structure of the ISO. Simultaneously, the enabled addressing decoder selects a single WL. The selected WL is supplied with a voltage V_{SEL}, which is already shared with the input of each transmission gate corresponding to every WL. The shared voltage V_{SEL} is set to the normalized voltage level of 0 V. The unselected WLs remain floating in the high impedance state (Z).

2) *Overwrite Mode:* In this mode, the function of the memory is switched to perform MOL among its memristive crossbar. As stated earlier, both MOL-OR and MOL-AND have to be supported. For the case of MOL-OR, the input data bits are mapped to the normalized voltage levels of 0 and 1 V corresponding to logic "0" and logic "1," respectively. The addressing decoder performs its normal selection function for a single WL. The ISO is kept at the connecting state. The level of V_{SEL} is also set to 0 V as in the case of write mode. The resulting bits of the MOL-OR operation are simultaneously stored in the selected WL. MOL-AND is performed similarly but V_{SEL} is switched to the high voltage level (i.e., $V_{\text{SEL}} = 1$ V).
3) *Read Mode:* In this mode, a single WL is selected to sense the corresponding states of its allocated memristors individually. BLD is isolated using the ISO block, which acts in this case as an open switch. The selection voltage V_{SEL} is set to 0.5 V (normalized). The sensing current generated through each memristor has to guarantee the stability of its internal state (no state drift). A sensing amplifier (SA) circuitry, whose architecture is illustrated in Fig. 7(c), is used to measure the voltages across the reference resistors of respective resistances R. R is chosen to be the midvalue between R_{on} and R_{off} (i.e., $R = (R_{\text{on}} + R_{\text{off}})/2$). By considering $R_{\text{on}} < R_{\text{off}}$, the voltage across a reference resistor, which is in series to the sensed memristor, would be either in the neighborhood of 0 or 0.5 V. Depending on the state of the sensed memristor, the three cascaded inverters magnify this difference leading to -1 or 1 V at the output.
4) *Idle Mode:* In this mode, the memory is not active. The memristive crossbar is totally isolated to preserve its internal state. The IB block is in the isolation mode. Hence, all BLs are in the high impedance state (Z). Moreover, the address decoder is disabled. Thus, none of the WLs is selected, keeping them in the Z state.

The architecture presented in Fig. 6(a) adopts the 1-memristor (1M) configuration for the structure of the crossbar. In other words, each cell consists of one memristor which connects the vertical and horizontal nanowires of the crossbar. However, the 1M crossbar configuration suffers from the sneak path phenomenon [34]. Sneak paths correspond to current paths through unselected cells in a memristive array. These undesired paths lead, in some cases, to a drift in the state of unselected memristive cells during write or overwrite operations. Moreover, it gives false estimation about the real logical state of a given selected memristor during the reading mode. This phenomenon degrades the overall memory performance. Several efforts have been made in the literature to overcome the sneak path phenomenon [34]–[36]. All proposed methods are limited to a certain crossbar size. Thus, increasing the size of the memristive crossbar beyond a certain limit will eventually lead to the sneak paths. A possible solution to stop these paths is to use a selector in series with each allocated memristive cell. This solution induces overheads in terms of the total utilized area of the memory which in turn loses the ultrahigh density attained in the 1M case. In [37], a transistor is used as a selector. Thus, each cell inside the memory consists of one transistor in series with one memristive device (1T1M). The obtained crossbar architecture for the 1T1M configuration is considered as sneak-path free. Fig. 6(b) presents our proposed 1T1M memory architecture with added MOL capabilities. The WL transmission gates that have been used in the 1M case are no longer used in the case, of 1T1M memory architecture. Normally, each transmission gate is equivalent to two MOSFETs. Thus, for an $N \times M$ memristive crossbar array, additional $NM - 2N$ MOSFETs are used in the 1T1M architecture compared to that in the 1M case. The obtained 1T1M architecture has the same four control modes previously introduced for the 1M case.

V. MOL-Based Computational Memory

In this section, a MOL-based computational memory architecture is introduced. The architecture can perform MOL operations between two stored WLs. The original architecture, which is formed of two interconnected MOL memory blocks, works in a complementary manner.

A. Architecture

The proposed MOL-memory architectures, which are presented in Section IV, act as logic accumulators for the newly arriving bits. In other words, computation in such memory is restricted for logic accumulation. Accordingly, performing general Boolean functions in this memory requires an additional process to load the stored data bits outside the

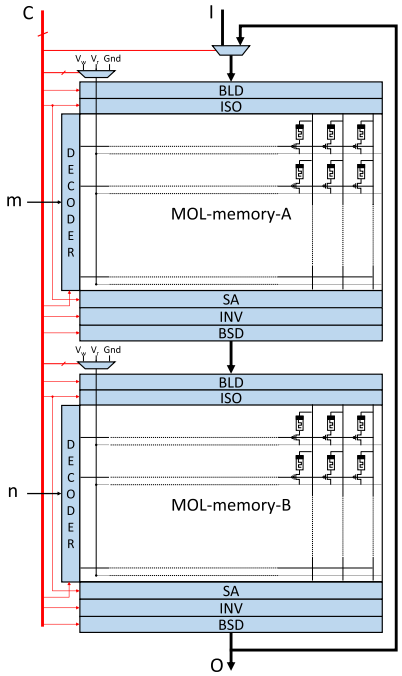


Fig. 8. Computational memory architecture.

memory. These additional load operations are at odds with the concept of computation inside the memory. To overcome these load operations, we propose the use of two coupled MOL memories (MOL-memory-A and MOL-memory-B) which work in a complementary manner. At each time step, one of these memories acts as a source of input data-bits of the second memory. The second memory performs MOL with the previously stored bits in its memristive crossbar. Fig. 8 illustrates our proposed computational-memory architecture. The architectures of MOL-memory-A and MOL-memory-B are identical. A controlled inverting driver (INV) is added after the sensing stages of the two memories. The function of this driver is to achieve a complete logic, as the OR and AND logic operations supported by the memories are not universal. Therefore, additional NOT operation is required to allow the description of any Boolean function. The architecture of INV is illustrated in Fig. 7(d). A 1-bit barrel shift driver (BSD) is added to enable bit-level operations in addition to vector-level operations. The BSD is responsible for ensuring switchable connections between the two memory blocks. It can be reconfigured either to pass the data bits or to shift them on the fly with no need for an additional cycle. The architecture of the BSD is presented in Fig. 7(e). The proposed MOL-memory architecture presented in Fig. 8 is capable of performing numerous operations including logic computation and storage. Table I lists the most important (not all) operations that could be achieved. For each listed operation, a set of appropriate commands are simultaneously sent to the blocks constituting the architecture. A single operation requires one computational step. As an example, case 19 in Table I corresponds to the arithmetic operation expressed in the following equation:

$$M_B(n) = M_B(n) \text{ AND } \overline{M_A(m)} \tag{3}$$

where $M_A(m)$ and $M_B(n)$ are the bit-vectors located at the addresses m and n corresponding for MOL-memory-A and MOL-memory-B, respectively. In this case, MOL-memory-A is set to the read mode. It reads the bit-vector $M_A(m)$, which undergoes a bitwise inversion through INV block. The 1-bit shifter is disabled. Simultaneously, MOL-memory-B is set to the overwrite mode to perform MOL-AND with the vector $M_B(n)$. The result of the bitwise logic operation replaces the previous vector $M_B(n)$. The process is performed during one computational step.

B. Performing General Arithmetic Tasks

Generally, an arithmetic function (e.g., addition, subtraction, compare, and so on) could be expressed in Boolean form. Accordingly, breaking the Boolean form into several MOL operations allows its execution inside the proposed computational memory. The execution of an arbitrary Boolean function requires several computational steps, thus MOL operations are executed iteratively to finalize the desired arithmetic task. For this purpose, an external controller, which arranges these iterative operations, is needed. Fig. 9 shows the block diagram which illustrates the general structure of the memory and the controller. When the controller receives an instruction from the processor, it decides whether the role of the memory is for storage or computation. Specifically, for the case of computation, the controller breaks the received macroinstruction into several iterative microinstructions, which can be performed by the proposed memory. In our case, microinstructions correspond to the set of operations listed in Table I. A processing area should be reserved in each of MOL-memory-A and MOL-memory-B in the proposed computational memory. The area could be dynamically changed according to the need (such as the number of required tasks). Moreover, the location of the processing area could be also changed periodically. The reason for location change is to attain better endurance for the memristive memory cells that are subjected to continuous stress. The design of the controller is beyond the scope of this article.

VI. MOL Based in Memory N-Bit Full Addition

In this section, an N-bit full addition is considered as a case study to evaluate the functionality of our proposed computational memory architecture.

A. Proposed Iterative N-Bit Full Addition Process Dedicated for Computational MOL-Memory

Generally, full adder is the basic digital building block for several computational operations (i.e., addition, subtraction, and multiplication). Thus, implementing a full addition process inside the memory is the first step toward in-memory computing. The following equations present the well-known expressions of the 1-bit full addition:

$$S = A \oplus B \oplus C_{\text{in}} \tag{4}$$

$$C_{\text{out}} = AB + BC_{\text{in}} + AC_{\text{in}} \tag{5}$$

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

TABLE I
ENCODING TABLE

		110 101 011 011 1 0	AND Read Write OR						AND Read Write OR						
			Enable Disable	Write Overwrite	Isolate Connect	Mode	Pass Invert		Enable Disable	Write Overwrite	Isolate Connect	Mode	Pass Invert		Select-out Select-in
Operation	**Binary**	**Micro-Instruction**	E_A	BLD_A	ISO_A	$Mode_A$	INV_A	BSD_A	E_B	BLD_B	ISO_B	$Mode_B$	INV_B	BSD_B	Sel
0	00000	$M_A(m) = I$	1	1	1	0 1 1	1	1	0	x	0	1 1 1	x	1	0
1	00001	$M_B(n) = I$	0	0	1	0 1 1	0	1	1	1	1	0 1 1	1	1	0
2	00010	$O = M_A(m)$	1	x	0	1 0 1	1	1	0	0	1	1 1 1	0	1	x
3	00011	$O = M_B(m)$	0	x	0	1 1 1	x	1	1	x	0	1 0 1	0	1	x
4	00100	$O = \overline{M_A(m)}$	1	x	0	1 0 1	1	1	0	0	1	1 1 1	0	1	x
5	00101	$O = \overline{M_B(m)}$	0	x	0	1 1 1	x	1	1	x	0	1 0 1	1	1	x
6	00110	$M_A(m) = M_B(n)$	1	1	1	0 1 1	x	1	1	x	0	1 0 1	0	1	1
7	00111	$M_B(n) = \overline{M_A(m)}$	1	x	0	1 0 1	1	1	1	1	1	0 1 1	x	1	x
8	01000	$M_A(m) = \overline{M_B(n)}$	1	1	1	0 1 1	x	1	1	x	0	1 0 1	1	1	1
9	01001	$M_B(n) = \overline{M_A(m)}$	1	x	0	1 0 1	0	1	1	1	1	0 1 1	x	1	x
10	01010	$M_A(m) = M_A(m)\ AND\ I$	1	0	1	1 1 0	1	1	0	x	0	1 1 1	x	1	0
11	01011	$M_B(n) = M_B(n)\ AND\ I$	0	0	1	0 1 1	0	1	1	1	1	1 1 0	1	1	0
12	01100	$M_A(m) = M_B(n)\ OR\ I$	1	0	1	0 1 1	1	1	0	x	0	1 1 1	x	1	0
13	01101	$M_B(n) = M_B(n)\ OR\ I$	0	0	1	0 1 1	0	1	1	1	1	0 1 1	1	1	0
14	01110	$M_A(m) = M_A(m)\ AND\ M_B(n)$	1	0	1	1 1 0	x	1	1	x	0	1 0 1	0	1	1
15	01111	$M_B(n) = M_B(n)\ AND\ M_A(n)$	1	x	0	1 0 1	1	1	1	0	1	1 1 0	x	1	x
16	10000	$M_A(m) = M_A(n)\ OR\ M_B(n)$	1	0	1	0 1 1	x	1	1	x	0	1 0 1	0	1	1
17	10001	$M_B(n) = M_B(m)\ OR\ M_A(n)$	1	x	0	1 0 1	1	1	1	0	1	0 1 1	x	1	x
18	10010	$M_A(m) = M_A(m)\ AND\ \overline{M_B(n)}$	1	0	1	1 1 0	x	1	1	x	0	1 0 1	1	1	1
19	10011	$M_B(n) = M_B(n)\ AND\ \overline{M_A(n)}$	1	x	0	1 0 1	0	1	1	0	1	1 1 0	x	1	x
20	10100	$M_A(m) = M_A(m)\ OR\ \overline{M_B(n)}$	1	0	1	0 1 1	x	1	1	x	0	1 0 1	1	1	1
21	10101	$M_B(n) = M_B(m)\ OR\ \overline{M_A(n)}$	1	x	0	1 0 1	0	1	1	0	1	0 1 1	x	1	x
22	10110	$M_A(m) = M_B(n) << 1$	1	1	1	0 1 1	x	1	1	x	0	1 0 1	1	0	1
23	10111	$M_B(n) = M_A(m) << 1$	1	x	0	1 0 1	1	0	1	1	1	0 1 1	x	1	x
24	11000	$M_A(m) = M_A(m)\ AND\ [M_B(n) << 1]$	1	0	1	1 1 0	x	1	1	x	0	1 0 1	0	0	1
25	11001	$M_B(n) = M_B(n)\ AND\ [M_A(n) << 1]$	1	x	0	1 0 1	1	0	1	0	1	1 1 0	x	1	x
26	11010	$M_A(m) = M_A(n)\ OR\ [M_B(n) << 1]$	1	0	1	0 1 1	x	1	1	x	0	1 0 1	0	0	1
27	11011	$M_B(n) = M_B(m)\ OR\ [M_A(n) << 1]$	1	x	0	1 0 1	1	0	1	0	1	0 1 1	x	1	x
28	11100	$M_A(m) = \overline{M_B(n) << 1}$	1	1	1	0 1 1	x	1	1	x	0	1 0 1	1	0	1
29	11101	$M_B(n) = \overline{M_A(m) << 1}$	1	x	0	1 0 1	0	0	1	1	1	0 1 1	x	1	x

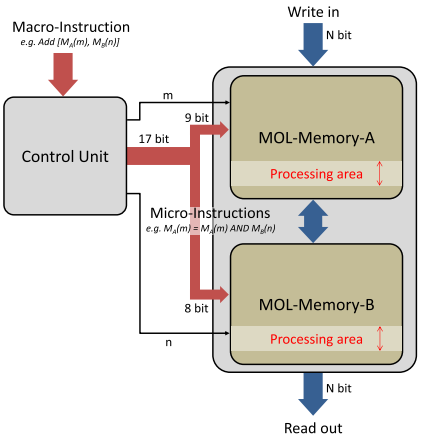


Fig. 9. Architecture diagram of MOL-based computational memory with its dedicated control unit.

where A and B are the inputs, C_{in} is the input carry value, S is the 1-bit adder output, and C_{out} is the output carry. The operator $\oplus$ corresponds to the Boolean XOR. Assume that all the inputs are initially stored in the memory. The Boolean functions of S and C_{out} are written in the form of the sum of products (SoP) so that their expressions could be mapped into the proposed computational memory using sequential MOL operations. The inputs of a given MOL operation should be aligned on the same columns (i.e., same BLs) in the memory; otherwise, a preshifting process is required to align the corresponding inputs. Accordingly, the number of steps required to achieve the computation of S and C_{out} is affected by the relative positions of the input A, B, and C_{in} inside the memory. In order to minimize the number of computational steps as well as reserve the minimum possible processing area, a dedicated N-bit addition process is proposed. The process uses a specific sequence of each operation listed in Table I. Consider the two N-bit vectors A^N and B^N. The addition of A^N and B^N leads to the vector sum S^{N+1}. Normally, the additional 1 bit in S^{N+1} is reserved for the expected overflow in the addition process. We propose to follow the procedure illustrated in Algorithm 1 to achieve a vector level addition of A^N and B^N.

1) *Stage 1:* The vector sum S_0 which is of length $N+1$ is initialized by the bitwise XOR of A^N and B^N. Similarly, the vector carry C_0 of length $N+1$ is initialized by the bitwise AND of A^N and B^N. The expressions of S_0 and C_0 are presented as follows:

$$S_0 = A \oplus B \tag{6}$$
$$C_0 = AB. \tag{7}$$

2) *Stage 2:* Each time, a new vector sum S_{i+1} and a vector carry C_{i+1} are created based on their previous values S_i and C_i, respectively. The following equations demonstrate the respective expressions of S_{i+1} and C_{i+1}. This process is repeated $N-1$ times:

$$S_{i+1} = S_i \oplus (C_i \ll 1) \tag{8}$$
$$C_{i+1} = S_i\,(C_i \ll 1). \tag{9}$$

The operator "$\ll 1$" stands for the 1-bit shift to the left. At the end of this iterative process, the final obtained vector S_{N-1} corresponds to the sum of A^N and B^N while C_{N-1} will be a zero vector.

B. In-Memory N-Bit Full Addition Procedure

The proposed iterative N-bit addition process can be mapped into the computational MOL-memory using the operations listed in Table I. Fig. 10 shows a space–time representation of the N-bit full addition process, which is realized

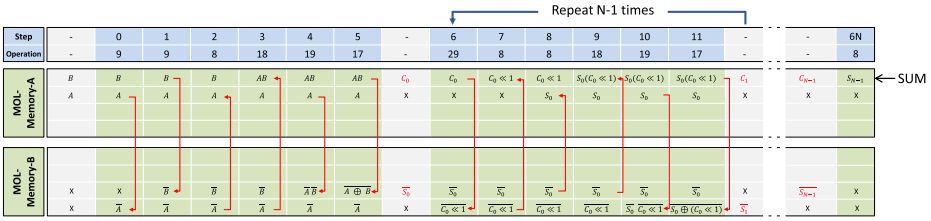


Fig. 10. Operations sequence for an in-memory N-bit addition process using MOL-memory.

Algorithm 1 N-Bit Addition Dedicated for Computation Inside MOL-Memory

procedure ADD(A, B) ▷ A and B are N-bit vectors
  $S_0 \leftarrow A \oplus B$
  $C_0 \leftarrow AB$
  for $i \leftarrow 0$ to $N-2$ **do**
    $S_{i+1} \leftarrow S_i \oplus (C_i << 1)$
    $C_{i+1} \leftarrow S_i(C_i << 1)$
  end for
  return S_{N-1} ▷ The sum of A and B
end procedure

within MOL-memory-A and MOL-memory-B. For each computational step, the new contents of the memories are listed in a new single column in Fig. 10. Assume the case where the two vectors A^N and B^N, which are subjected to addition, are initially stored inside MOL-memory-A at the addresses m_1 and m_2, respectively. Additional two WLs have to be reserved inside MOL-memory-B to attain the addition of A^N and B^N. The two stages that are presented in Section VI-A are realized as follows.

1) *Stage 1:* Corresponds to the steps between 0 and 5 using the six micro-operations that have the sequence order shown in Fig. 10. At the end of this stage, the bitwise AND of A and B (i.e., $C_0 = AB$) is stored in MOL-memory-A while the XNOR of A and B (i.e., $\overline{S_0} = \overline{A \oplus B}$) is stored in MOL-memory-B.
2) *Stage 2:* In this stage, the steps between 6 and 11 are repeated $N-1$ times. Their corresponding microinstructions have the sequence order shown in Fig. 10. Each time, the initial vector C_i is shifted to the left by one bit and the resulting vector undergoes bitwise AND with the initial vector S_i. The obtained result is referred as C_{i+1}, whose expression is presented in (9). Simultaneously, the shifted version of C_i undergoes bitwise XNOR with the initial vector S_i to obtain the new vector sum S_{i+1}. At the end of this process, the vector $\overline{S_{N-1}}$ is stored in MOL-memory-B. Thus, an additional step is required to make a bitwise inversion of the obtained vector. The resulting vector S_{N-1}, which represents the N-bit addition of the vectors A and B, is stored in MOL-memory-A.

C. Space–Time Analysis of the N-Bit Addition Process

The total number of computational steps required to complete the N-bit addition is $6N+1$ steps as shown in Fig. 10. The total number of memristors reserved for the execution of the N-bit addition is $4N$ memristors corresponding to four rows of the MOL-memory architecture. These rows include the initial locations of A and B, although the initial bits of the vectors A and B are lost. However, in some cases, the destruction of the input vectors is undesired, especially when these inputs are required for other computational tasks. In order to avoid this case, precopy operations of the two input vectors A and B could be performed to reserve safe versions of these vectors. Thus, two additional computational steps are required for this case and the new total number of computational steps becomes $6N+3$. The considered operation sequence in Fig. 10 corresponds to the case where A and B are both located in MOL-memory-A. However, other two cases should also be considered: 1) if A and B belong to different MOL-memories, one additional precopy operation could be performed to drag the input vector contained in MOL-memory-B to MOL-memory-A and 2) if A and B are both contained in MOL-memory-B, two additional precopy operations are needed to drag them to MOL-memory-A. These precopy operations are performed to maintain the same operation sequence, which is presented in Fig. 10. Precopy operations can be avoided with different sequences (one for each case, with common parts).

VII. Simulation and Performance Analysis

In this section, we study the performance of the proposed computational memory architecture which is implemented using a realistic model of magnetic tunnel junction (MTJ) device and a CMOS 65-nm technology node. The study includes timing analysis, energy consumption, and robustness against device variability.

A. Adopted Memristive Device

Several memristive devices have been explored in the literature. In fact, the MOL technique could apply to all types of bipolar memristive devices holding two resistance states R_{on} and R_{off}. Among these devices, memristors such as HfO_x [38] and TiO_2 [32] exhibit promising characteristics with their high switching speed (subnanosecond) and their high R_{off}/R_{on} ratio (>100). However, current memristor technologies suffer from endurance limitations. Although several efforts have been carried out to enhance endurance [38], the allowed number of switchings per memristor is still limited in the range of 10^6–10^{12} for the best case. This value is relatively low for targeting intensive computations inside memristive crossbars. The spin-transfer torque magnetic memory (STT-MRAM) [39], which has been redescribed in terms of memristive systems

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

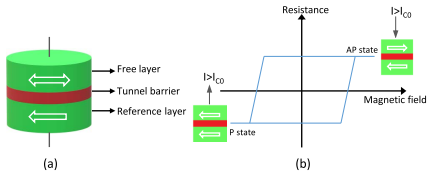


Fig. 11. Typical MTJ. (a) Core structure. (b) Resistance variation.

[40], is considered as one of the most promising nonvolatile memories (NVMs). STT-MRAM is eligible for high-reliability applications [41] due to its high endurance ($>10^{15}$) [32]. As illustrated in Fig. 11, an MTJ cell is mainly composed of two ferromagnetic layers sandwiching an ultrathin tunnel barrier. The resistance of the MTJ cell depends on the relative orientation of magnetization in the free and reference layers. The low-resistance state (logic "0") of the MTJ corresponds to the parallel configuration (P) with resistance R_P, while its high resistance state (logic "1") is reached in the case of antiparallel (AP) configuration with resistance R_{AP}. The magnitude of the applied current I must exceed a critical value noted as I_{C0} to allow switching. In contrast to memristors, MTJs are characterized by a relatively low margin between R_P and R_{AP}. The corresponding margin is commonly evaluated as the tunnel magnetoresistance (TMR) ratio, whose expression is presented in the following equation:

$$\mathrm{TMR} = \frac{\Delta R}{R_P} = \frac{R_{\mathrm{AP}} - R_P}{R_P}. \tag{10}$$

However, such a low margin has no effect on switching an MTJ cell but on the corresponding sensing mechanism of the state of this cell. This requires a more complicated sensing driver to estimate and decide the corresponding state of a given selected row inside the memory. In this work, we have used MTJs with perpendicular magnetic anisotropy (PMA). The adopted PMA MTJ is formed of CoFeB/MgO/CoFeB layers. The physical model describing the static, dynamic, and stochastic behaviors of the STT-PMA-MTJ is presented in [42] and [43]. In order to fit with experimental results in the literature, the technology parameters corresponding to the material composition are kept at their default values. Other parameters, which depend on the designers' choice, are presented in Table II with their corresponding values. It is worth highlighting that the low TMR value of the adopted STT-PMA-MTJ device usually leads to a high complexity of SAs when the memory data need to be readout. However, in our case, the proposed simple design comprised of three cascaded inverters and a reference resistor was verified to be sufficient when combined with the designed INV and BSD blocks (Fig. 7). In fact, these two CMOS-based blocks are leveraged in order to perfectly regenerate the voltage levels corresponding to the amplified output of the cascaded inverters. Fig. 12 shows the switching behavior of an MTJ device when it is fed with a square signal of amplitude 1.2 V. $\tau_{\mathrm{AP}-P}$ and $\tau_{P-\mathrm{AP}}$ correspond to the switching delays from AP to P state and the reverse case, respectively. In fact, switching delay varies according to the applied voltage level. Fig. 13 illustrates the variation of $\tau_{\mathrm{AP}-P}$ and $\tau_{P-\mathrm{AP}}$ with respect to the applied voltage level. The graph indicates that switching delay decreases with the increase of the voltage while switching

TABLE II
ADOPTED VARIABLES AND PARAMETERS FOR PMA MTJ DEVICE

Parameter	Value	Description
t_{ox}	$0.85\,nm$	Thickness of oxide barrier
$TMR(0)$	70%	TMR ratio with 0 stress voltage
$Area$	$\pi \times 20\,nm \times 20\,nm$	MTJ surface
t_{sl}	$1.3\,nm$	Thickness of free layer

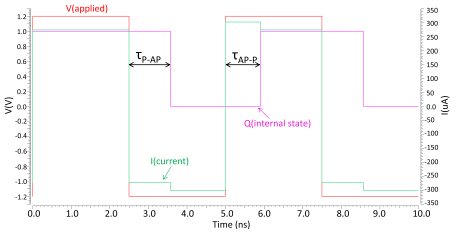


Fig. 12. Switching behavior of MTJ device when fed with square signal.

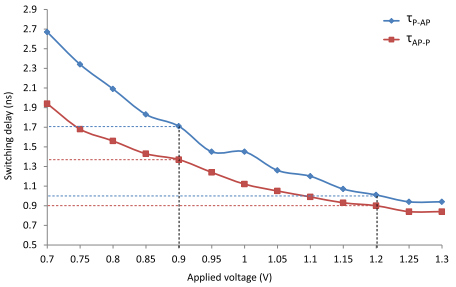


Fig. 13. Switching delay of an MTJ cell as function of applied voltage level.

from P to AP state is faster than the reverse operation (i.e., $\tau_{\mathrm{AP}-P} < \tau_{P-\mathrm{AP}}$).

The choice of the memristive device type is not constrained by a specified MOL requirement. It can be observed from the mechanism of the MOL technique that it involves direct access to the terminals of memristive devices which highly resembles conventional write operation. During the operation of MOL, the potential difference between the terminals of the memristive device always attains a binary level. Accordingly, MOL can be implemented in a wide range of memristive memories without specifying particular device features. In contrast, the structure of preexisting logic design styles either establishes a series connection of a resistor (e.g., IMPLY) or series connection of the memristive devices (e.g., MAGIC) for normal operation. This undoubtedly prevents direct access to the memristive device terminals and consequently imposes specific device constraints, such as the requirement of sufficient HRS/LRS ratio and/or operated with threshold-type devices only.

B. Performance Analysis

A transient simulation has been conducted for the proposed design of the MOL-memory architecture. Based on the adopted STT-PMA-MTJ device and the CMOS 65-nm

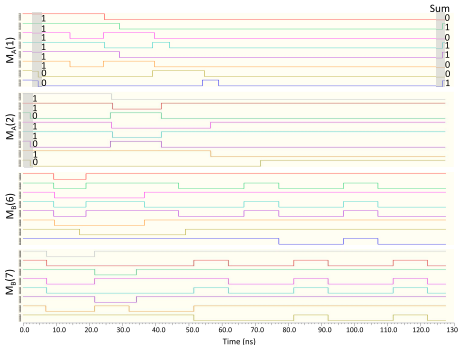


Fig. 14. Transient simulation for the in-memory 8-bit addition process.

process, simulations have been carried out using Cadence Virtuoso toolset. In order to evaluate the performance of the architecture, the N-bit addition process described in Section VI is performed. The size of the crossbar is chosen to be 8×8 for MOL-memory-A as well as MOL-memory-B. The size N is chosen to be 8 bits for both numbers A and B. The corresponding operating voltage is set to 1.2 V for logic "1" and -1.2 V for logic "0." Based on the obtained transient results, total latency is evaluated as well as the total energy consumption. As an example, Fig. 14 presents the corresponding internal states of the four WLs that are reserved for the 8-bit addition process, which is performed on the two arbitrary vectors $A = [01011011]$ and $B = [00111111]$. The control signals of the MOL-memory architecture follow the operation sequence presented in Fig. 10.

1) Timing Analysis: The first two steps correspond to the initialization of vectors A and B inside MOL-memory-A. The corresponding sum $S = [10011010]$ is evaluated after $6N + 1$ computational steps which is equal to 49 for $N = 8$. In fact, the max delay is noticed to be $\tau_{\text{Max}} = 1.7$ ns which is greater than the max switching delay of MTJ devices operating at 1.2 V. This is due to the voltage drop noticed along CMOS drivers. The actual voltage supplied to MTJ devices is 0.9 V (could be interpreted from Fig. 13). This significant voltage drop (25%) is due to the adoption of low values of R_P and R_{AP}. Moreover, the width W of MOSFETs has a direct effect on the voltage drop percentage. This voltage drop could be mitigated by increasing W, but this induces overheads on the total area of CMOS drivers.

Therefore, the duration (T) of each computational step must be greater than τ_{Max}. The variability in τ_{Max} due to the stochastic switching behavior of MTJs should also be considered. Thus, an additional guard interval (τ_g) is introduced to guarantee the switching of the MTJs. The resulting step duration for the proposed MOL-memory architecture is $T = \tau_{\text{Max}} + \tau_g = 1.7 + \tau_g$. We set τ_g at 100 ps which corresponds to 6% of τ_{Max}, so the duration T is equal to 1.8 ns. The minimum time required for finalizing the addition operation (neglecting the two initialization steps) is evaluated as 49×1.8 ns $= 88.2$ ns.

2) Robustness Against Resistance Variability: Due to the limit of the manufacturing technology, the actual thickness of the oxide layer and the free layer of MTJ devices cannot be fixed at a constant value. They typically vary in a small range but can lead to a relatively important variation in the values of LRS and HRS of MTJ. Therefore, we have examined the effect of MTJ resistance variability on the performance of our proposed MOL-based computational memory architecture. Simulations are conducted by performing the 8-bit addition. The adopted MTJ parameters TMR, t_{sl}, and t_{ox} are kept as presented in Table II while subjecting them to a random process. The parameters are chosen to follow either uniform or Gaussian distribution. In Gaussian distribution, no error has been detected even when reaching a variation percentage of 21% for TMR, t_{sl}, and t_{ox}. As for uniform distribution, the tolerated variation reaches 7%. This demonstrates the robustness of the proposed design against the resistance variability of MTJ devices.

3) Energy Estimation: Energy consumption differs according to the operation: read, write, or performing computation. In this section, we will focus on the energy consumed by the memristive crossbar of the MOL-memory architecture neglecting the consumed energy by the peripheral drivers.

a) Write-energy: Consider a single MTJ device located inside MOL-memory architecture. The energy consumed when a single bit is written into this MTJ device mainly depends on its previous resistance state (R_P or R_{AP}) and its final one. Hence, the four cases for write-energy are considered in the following equation:

$$\begin{aligned} E_{w_{0/0}} &= \frac{{V_w}^2}{R'_{\text{AP}}} T \\ E_{w_{0/1}} &= \frac{{V_w}^2}{R'_{\text{AP}}} \tau_{\text{AP}-P} + \frac{{V_w}^2}{R'_P} (T - \tau_{\text{AP}-P}) \\ E_{w_{1/0}} &= \frac{{V_w}^2}{R'_P} \tau_{P-\text{AP}} + \frac{{V_w}^2}{R'_{\text{AP}}} (T - \tau_{P-\text{AP}}) \\ E_{w_{1/1}} &= \frac{{V_w}^2}{R'_P} T \end{aligned} \tag{11}$$

where $E_{w_{i/j}}$ corresponds to the write-energy needed to put the MTJ device in state $i \in \{0, 1\}$, after it was in the previous state $j \in \{0, 1\}$; V_w and T are the write voltage and write duration, respectively; R'_{AP} and R'_P represent the resistance states of 1T1M cell. $R'_{\text{AP}} = R_{\text{AP}} + R_{\text{MOS}}$ and $R'_P = R_P + R_{\text{MOS}}$. Generally, the values of i and j are not deterministic, but the four cases presented in (11) are considered as equiprobable, since there is no preknowledge about the data bits inside the memory as well as the bits that would be written. Thus, the average write-energy is estimated as the average sum of the four write-energy cases as presented in the following equation:

$$E_w = \frac{1}{4} \sum_{i,j} E_{w_{i/j}} = \frac{{V_w}^2}{R'_{\text{AP}}} \left(\frac{T}{2} - \frac{\Delta\tau}{4} \right) + \frac{{V_w}^2}{R'_P} \left(\frac{T}{2} + \frac{\Delta\tau}{4} \right) \tag{12}$$

where $\Delta\tau = \tau_{P-\text{AP}} - \tau_{\text{AP}-P}$. Assuming that the term $(\Delta\tau/4)$ is almost negligible compared to $(T/2)$, the overall expression in (12) is simplified as

$$E_w \approx \frac{{V_w}^2}{2R_w} T \quad \text{with} \quad R_w = \frac{R'_P R'_{\text{AP}}}{R'_P + R'_{\text{AP}}}. \tag{13}$$

R_w represents the equivalent resistance of two MTJs having opposite states and connected in parallel.

TABLE III
ENERGY CONSUMED BY A COMPUTATIONAL OPERATION

In1	In2	MOL-AND	MOL-OR	Copy
0	0	$E_{w_{0/0}} + E_{r_0}$	E_{r_0}	$E_{w_{0/0}} + E_{r_0}$
0	1	E_{r_0}	$E_{w_{1/0}} + E_{r_0}$	$E_{w_{1/0}} + E_{r_0}$
1	0	$E_{w_{0/1}} + E_{r_1}$	E_{r_1}	$E_{w_{0/1}} + E_{r_1}$
1	1	E_{r_1}	$E_{w_{1/1}} + E_{r_1}$	$E_{w_{1/1}} + E_{r_1}$

b) Read-energy: Reading a single MTJ device requires a sensing voltage V_r and a reference resistor R_{Ref} connected in series with a MOSFET. The total resistance of this 1T1R cell is R'_{Ref}. The corresponding state of the sensed MTJ device is assumed to be stable. The two possible cases for read-energy are presented in the following equation:

$$E_{r_0} = \frac{{V_r}^2}{R'_{\text{AP}} + R'_{\text{Ref}}} T$$
$$E_{r_1} = \frac{{V_r}^2}{R'_P + R'_{\text{Ref}}} T \quad (14)$$

where E_{r_0} and E_{r_1} represent the required energy consumption for sensing AP and P states, respectively, during a period T. The corresponding average read-energy is expressed in the following equation:

$$E_r = \frac{{V_r}^2}{2R_r} T \quad \text{with} \quad R_r = \frac{\left(R'_P + R'_{\text{Ref}}\right)\left(R'_{\text{AP}} + R'_{\text{Ref}}\right)}{\left(R'_P + R'_{\text{Ref}}\right) + \left(R'_{\text{AP}} + R'_{\text{Ref}}\right)}. \quad (15)$$

c) Computation-energy: Computational operations that are performed inside MOL-memory architecture are classified into MOL or copy operations. Table III summarizes the energy consumed by each type of operation. Using the specifications of the adopted MTJ which are listed in Table IV, the average energy consumed by an MOL operation could be expressed as $E_{\text{MOL}} = E_w/2 + E_r = 0.196\,pj$ whereas that consumed by a copy operation is calculated as $E_{\text{COPY}} = E_w + E_r = 0.333\,pj$.

Normally, computation inside MOL-memory architecture is performed on N bits simultaneously. For the N-bit addition process which is performed within $6N + 1$ cycles, $3N$ cycles corresponds to MOL operations while $3N + 1$ cycles corresponds to copy operations. Thus, the overall consumed energy (E_T) could be expressed as in the following equation:

$$E_T = (3N)(N\ E_{\text{MOL}}) + (3N + 1)(N\ E_{\text{COPY}}). \quad (16)$$

By substituting the corresponding values of E_{MOL} and E_{COPY} presented in Table III, the expression of the total energy becomes $E_T = 1.587N^2 + 0.333N$. Specifically, for the 8-bit addition process, E_T is equal to 104.2 pJ. The value of the energy consumption extracted by simulation is 124.43 pJ.

VIII. COMPARISON

In this section, the proposed MOL-memory architecture has been compared with recently published relevant designs (listed in Table V) targeting in-memory computing. The comparison has been carried out based on the performance of N-bit addition in terms of latency, energy consumption, and utilized area. Note that the considered area incorporates only the memristors involved in the computation regardless of the size of the crossbar.

TABLE IV
SPECIFICATIONS

Specification	Value
R_{AP}	$6\,K$
R_P	$3.97\,K$
R_{MOS}	$0.5\,K$
R_{Ref}	$4.8\,K$
V_w	$0.588\,V$
V_r	$0.9\,V$
τ_{AP-P}	$1.4\,ns$
τ_{P-AP}	$1.7\,ns$
T	$1.8\,ns$

4) MOL Versus IMPLY and MAGIC:

1) Except for the parallel approach in [10], our proposed design, which uses only $6N + 1$ steps to perform addition, outperforms all IMPLY- and MAGIC-based designs listed in Table V in terms of number of computational steps. In fact, [10] uses the parallel approach which is intended to increase the level of parallelism in computation. However, this approach requires significant modifications in the crossbar structure by adding connections between its rows. This leads to an increased area compared to the conventional crossbar structure.
2) The step delay in our proposed design is 1.8 ns. Although the designs presented in [12], [13], and [30] adopt memristive devices that provide better step delay (1.12 to 1.43 ns), the total latency in our proposed design is still the minimum ($10.8N + 1.8$ ns). The best case achieved with the competitor designs is recorded in [12] with $13N + 3.9$ ns (i.e., $\sim$20% more latency).
3) In the proposed design, $4N$ memristors participate in the execution of the N-bit addition. This number ranges from $11N - 1$ to $24N$ for the majority of the designs based on MAGIC, so our proposed design exhibits $\times$1.75 to $\times$5 area reduction. On the other hand, the IMPLY-based serial approach [10], MAGIC-based area optimized design [12], and the design presented in [11] use a fixed number of memristors to perform addition operation. In other words, the required number of memristors is independent of the size N of the addition operation. This area optimization comes at the cost of a high number of computational steps ($\times$2.5 to $\times$18.8).
4) The average energy consumed in picojoule for the memristive crossbar in our design is $1.5867N^2 + 0.333N$. This quadratic expression indicates a significant energy consumption in the order of $\times N$ as compared to the linear energy expressions of the other designs listed in the table. The reason for this energy gap is that for each step, the same bitwise operation is performed on the whole WL (size N). However, the other approaches from the literature perform 1-bit operation in each step. Although our methodology induces overheads on the total energy consumption, working on the vector level rather than bit level greatly simplifies the corresponding control unit and reduces its complexity.

5) MOL Versus MAJ and CRS:

1) Logic representation using MIGs has experimentally shown promising results in logic optimization [44]. Memristive devices can efficiently execute the intrin-

TABLE V
COMPARISON OF DIFFERENT LOGIC FAMILIES FOR N-BIT ADDITION IN TERMS OF AREA, LATENCY, AND ENERGY CONSUMPTION

Reference	Method	# Steps	Step delay	Latency (ns)	Area (# memristive cells)	Energy (pJ)
(This work)	MOL	$6N+1$	$1.8ns$	$10.8N+1.8$	$4N$	$1.587N^2+0.333N$
[10]	IMPLY Serial	$29N$	-	-	2	$\sim 9.5N$
[10]	IMPLY Parallel	$5N+18$	-	-	$6N-1$	$\sim 9.5N$
[11]	IMPLY	$89N$	-	-	4	-
[12]	MAGIC Area optimized	$15N$	$1.3ns$	$19.5N$	5	$\sim 3.365N$
[12]	MAGIC Latency optimized	$12N+1$	$1.3ns$	$15.6N+1.3$	$11N-1$	$\sim 3.365N$
[12]	MAGIC Transpose I	$15N+1$	$1.3ns$	$19.5N+1.3$	$22N-3$	$\sim 6.53N$
[12]	MAGIC Transpose II	$10N+3$	$1.3ns$	$13N+3.9$	$13N-3$	$\sim 4.72N$
[13]	MAGIC	$12N+1$	$1.12ns$	$13.44N+1.12$	$14N+1$	$0.684N$
[30]	MAGIC (Naive mapping)	$12N$	$1.43ns$	$17.6N$	$15N$	$0.684N$
[30]	MAGIC (Compact mapping)	$16N$	$1.43ns$	$22.8N$	$24N$	$0.894N$
[14]	MAGIC	$20N+15$	$1.89ns$	$37.8N+28.35$	12	$0.3N$
[15]	MAJ (Naive)	$\sim 22N$	-	-	$\sim 4N$	-
[15]	MAJ (MIG rewriting)	$\sim 16N$	-	-	$\sim 3N$	-
[15]	MAJ (Rewriting and compilation)	$\sim 15N$	-	-	$\sim 2N$	-
[16]	CRS (PC-Adder)	$2N+4$	-	-	$2N+1$	-
[16]	CRS (TC-Adder)	$4N+5$	-	-	$N+2$	-

sic resistive MAJ operation. Shirinzadeh *et al.* [15] and Gaillardon *et al.* [23] present a programmable in-memory computing system, namely, programmable logic-in-memory (PLiM). The instruction set for the PLiM architecture is based on the MAJ operation. As investigated in [15], the number of required memristors for the addition is $\sim 2N$, which is equal to 50% of that in our approach. However, the execution of an N-bit addition inside PLiM requires $15N$ cycles for the best case, which is $\times 2.5$ the number of cycles required in our proposed design. This high number of computational steps is related to the repeated read-out operations of intermediate results, which impacts, in addition, the step delay and energy consumption (not evaluated in [15]).

2) The number of computational steps achieved in [16], which uses the CRS approach, is less than that of our proposed computational memory. However, other parameters, such as the step delay which is not investigated by the authors, are expected to be greater. This is due to the fact that the presented architecture, based on two separate memory blocks, uses an intermediate control unit which reads data bits from one memory block and redistributes them along with BLs and WLs of the other memory block. This process increases significantly the overall critical path and consequently the step delay. The number of memristive cells required in [16] is also less than that in our proposed design. However, it is clear that based on this approach, the reserved area corresponds to a fixed location inside the memory, as the input bits cannot be shared with all WLs especially for large memory sizes. This affects the endurance of memristive cells participating in the computation which are subjected to continuous stress.

As explained in Section V-B, the proposed memristive computational memory can perform any general arithmetic function by breaking it into a netlist of iterative MOL operations. As MOL is based on the primitive AND/OR operations, the ABC tool [45], which has been employed for existing logic design styles [30], [31], could be also leveraged in order to realize the synthesis task. This will be considered in our future work.

IX. CONCLUSION

In this article, the MOL design style is introduced together with an original architecture for MOL-based computational memory. This novel logic design style is inspired by a digital representation of memristors. Unlike existing approaches, MOL can operate with different memristor technologies, regardless of LRS-HRS margin and with linear as well as threshold-type memristive devices. Furthermore, the proposed original computational memory architecture, with appropriate drivers and control sequences, allows the execution of numerous logic operations, at bit or vector level, in one or two computational steps at most. In order to illustrate the benefits of the proposed approach and to evaluate its performances, the implementation of an N-bit full addition using the proposed MOL-based computational memory has been detailed. The design is simulated in the Cadence Virtuoso environment using a CMOS 65-nm process and realistic model parameters of the STT-PMA-MTJ device. Results comparison with existing recent approaches demonstrates significant reductions in terms of latency and area.

REFERENCES

[1] L. Chua, "Memristor—The missing circuit element," *IEEE Trans. Circuit Theory*, vol. 18, no. 5, pp. 507–519, 1971.
[2] D. B. Strukov, G. S. Snider, D. R. Stewart, and R. S. Williams, "The missing memristor found," *Nature*, vol. 453, no. 7191, p. 80, 2008.
[3] S. Kvatinsky, N. Wald, G. Satat, A. Kolodny, U. C. Weiser, and E. G. Friedman, "MRL—Memristor ratioed logic," in *Proc. Int. Workshop Cellular Nanosc. Netw. Their Appl.*, 2012, pp. 1–6.
[4] J. Chowdhury, K. Das, and K. Rout, "Implementation of 24T memristor based adder architecture with improved performance," *Int. J. Electr., Electron. Data Commun.*, vol. 3, no. 6, pp. 91–94, 2015.
[5] K. A. Ali, M. Rizk, A. Baghdadi, J.-P. Diguet, and J. Jomaah, "MRL crossbar-based full adder design," in *Proc. 26th IEEE Int. Conf. Electron., Circuits Syst. (ICECS)*, Nov. 2019, pp. 674–677.
[6] A. Sebastian, M. Le Gallo, R. Khaddam-Aljameh, and E. Eleftheriou, "Memory devices and applications for in-memory computing," *Nature Nanotechnol.*, vol. 15, pp. 529–544, Mar. 2020.
[7] D. Patterson *et al.*, "A case for intelligent RAM," *IEEE Micro*, vol. 17, no. 2, pp. 34–44, Apr. 1997.
[8] S. Khoram, Y. Zha, J. Zhang, and J. Li, "Challenges and opportunities: From near-memory computing to in-memory computing," in *Proc. ACM Int. Symp. Phys. Design*, Mar. 2017, pp. 43–46.
[9] S. Hamdioui *et al.*, "Applications of computation-in-memory architectures based on memristive devices," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, Mar. 2019, pp. 486–491.

[10] S. Kvatinsky, G. Satat, N. Wald, E. G. Friedman, A. Kolodny, and U. C. Weiser, "Memristor-based material implication (IMPLY) logic: Design principles and methodologies," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 22, no. 10, pp. 2054–2066, Oct. 2014.

[11] E. Lehtonen and M. Laiho, "Stateful implication logic with memristors," in *Proc. IEEE/ACM Int. Symp. Nanosc. Archit.*, Jul. 2009, pp. 33–36.

[12] N. Talati, S. Gupta, P. Mane, and S. Kvatinsky, "Logic design within memristive memories using memristor-aided loGIC (MAGIC)," *IEEE Trans. Nanotechnol.*, vol. 15, no. 4, pp. 635–650, Jul. 2016.

[13] P. Thangkhiew, R. Gharpinde, D. N. Yadav, K. Datta, and I. Sengupta, "Efficient implementation of adder circuits in memristive crossbar array," in *Proc. IEEE Region 10 Conf. (TENCON)*, Nov. 2017, pp. 207–212.

[14] P. L. Thangkhiew, R. Gharpinde, P. V. Chowdhary, K. Datta, and I. Sengupta, "Area efficient implementation of Ripple Carry adder using memristor crossbar arrays," in *Proc. 11th Int. Design Test Symp. (IDT)*, Dec. 2016, pp. 142–147.

[15] S. Shirinzadeh, M. Soeken, P.-E. Gaillardon, and R. Drechsler, "Logic synthesis for majority based in-memory computing," in *Advances in Memristors, Memristive Devices and Systems*. Cham, Switzerland: Springer, 2017. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-319-51724-7_17#citeas

[16] A. Siemon, S. Menzel, R. Waser, and E. Linn, "A complementary resistive switch-based crossbar array adder," *IEEE J. Emerg. Sel. Topics Circuits Syst.*, vol. 5, no. 1, pp. 64–74, Mar. 2015.

[17] C.-X. Xue *et al.*, "A 1 Mb multibit ReRAM computing-in-memory macro with 14.6 NS parallel MAC computing time for CNN based AI edge processors," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, Feb. 2019, pp. 388–390.

[18] W.-H. Chen *et al.*, "A 16 Mb dual-mode ReRAM macro with sub-14ns computing-in-memory and memory functions enabled by self-write termination scheme," in *IEDM Tech. Dig.*, Dec. 2017, pp. 28.2.1–28.2.4.

[19] C.-X. Xue *et al.*, "A 22 nm 2 Mb ReRAM compute-in-memory macro with 121-28TOPS/W for multibit MAC computing for tiny AI edge devices," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, Feb. 2020, pp. 244–246.

[20] W.-H. Chen *et al.*, "CMOS-integrated memristive non-volatile computing-in-memory for AI edge processors," *Nature Electron.*, vol. 2, no. 9, pp. 420–428, Sep. 2019.

[21] S. Kvatinsky *et al.*, "MAGIC—Memristor-aided logic," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 61, no. 11, pp. 895–899, Nov. 2014.

[22] X. Fang and Y. Tang, "Circuit analysis of the memristive stateful implication gate," *Electron. Lett.*, vol. 49, no. 20, pp. 1282–1283, Sep. 2013.

[23] P.-E. Gaillardon *et al.*, "The programmable logic-in-memory (PLiM) computer," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, 2016, pp. 427–432.

[24] D. Biolek, V. Biolkova, and Z. Biolek, "Pinched hysteretic loops of ideal memristors, memcapacitors and meminductors must be 'self-crossing,'" *Electron. Lett.*, vol. 47, no. 25, pp. 1385–1387, Dec. 2011.

[25] J. Rajendran, H. Manem, R. Karri, and G. S. Rose, "An energy-efficient memristive threshold logic circuit," *IEEE Trans. Comput.*, vol. 61, no. 4, pp. 474–487, Apr. 2012.

[26] L. Guckert and E. E. Swartzlander, "MAD gates—Memristor logic design using driver circuitry," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 64, no. 2, pp. 171–175, Feb. 2016.

[27] E. Linn, R. Rosezin, S. Tappertzhofen, U. Böttger, and R. Waser, "Beyond von Neumann—logic operations in passive crossbar arrays alongside memory operations," *Nanotechnology*, vol. 23, no. 30, 2012, Art. no. 305205.

[28] K. C. Rahman, D. Hammerstrom, Y. Li, H. Castagnaro, and M. A. Perkowski, "Methodology and design of a massively parallel memristive stateful IMPLY logic-based reconfigurable architecture," *IEEE Trans. Nanotechnol.*, vol. 15, no. 4, pp. 675–686, Jul. 2016.

[29] R. Ben Hur and S. Kvatinsky, "Memristive memory processing unit (MPU) controller for in-memory processing," in *Proc. IEEE Int. Conf. Sci. Electr. Eng. (ICSEE)*, Nov. 2016, pp. 1–5.

[30] R. Gharpinde, P. L. Thangkhiew, K. Datta, and I. Sengupta, "A scalable in-memory logic synthesis approach using memristor crossbar," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 26, no. 2, pp. 355–366, Feb. 2018.

[31] P. L. Thangkhiew, R. Gharpinde, and K. Datta, "Efficient mapping of Boolean functions to memristor crossbar using MAGIC NOR gates," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 65, no. 8, pp. 2466–2476, Aug. 2018.

[32] J. J. Yang, D. B. Strukov, and D. R. Stewart, "Memristive devices for computing," *Nature Nanotechnol.*, vol. 8, no. 1, p. 13, 2013.

[33] L. Amaru, P.-E. Gaillardon, and G. De Micheli, "Majority-inverter graph: A new paradigm for logic optimization," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 35, no. 5, pp. 806–819, May 2016.

[34] M. A. Zidan, H. A. H. Fahmy, M. M. Hussain, and K. N. Salama, "Memristor-based memory: The sneak paths problem and solutions," *Microelectron. J.*, vol. 44, no. 2, pp. 176–183, Feb. 2013.

[35] Y. Cassuto, S. Kvatinsky, and E. Yaakobi, "Sneak-path constraints in memristor crossbar arrays," in *Proc. IEEE Int. Symp. Inf. Theory (ISIT)*, Jul. 2013, pp. 156–160.

[36] S. Shin, K. Kim, and S.-M. Kang, "Analysis of passive memristive devices array: Data-dependent statistical model and self-adaptable sense resistance for RRAMs," *Proc. IEEE*, vol. 100, no. 6, pp. 2021–2032, Jun. 2012.

[37] Y. Huai, "Spin-transfer torque MRAM (STT-MRAM): Challenges and prospects," *AAPPS Bull.*, vol. 18, no. 6, pp. 33–40, 2008.

[38] H. Lee *et al.*, "Evidence and solution of over-reset problem for HfO_x based resistive memory with sub-ns switching speed and high endurance," in *IEDM Tech. Dig.*, Dec. 2010, pp. 19.7.1–19.7.4.

[39] S. Ikeda *et al.*, "A perpendicular-anisotropy CoFeB–MgO magnetic tunnel junction," *Nature Mater.*, vol. 9, no. 9, p. 721, 2010.

[40] X. Wang, Y. Chen, H. Xi, H. Li, and D. Dimitrov, "Spintronic memristor through spin-torque-induced magnetization motion," *IEEE Electron Device Lett.*, vol. 30, no. 3, pp. 294–297, Mar. 2009.

[41] T. Hanyu *et al.*, "Standby-power-free integrated circuits using MTJ-based VLSI computing," *Proc. IEEE*, vol. 104, no. 10, pp. 1844–1863, Oct. 2016.

[42] Y. Wang, Y. Zhang, E. Y. Deng, J. O. Klein, L. A. B. Naviner, and W. S. Zhao, "Compact model of magnetic tunnel junction with stochastic spin transfer torque switching for reliability analyses," *Microelectron. Rel.*, vol. 54, nos. 9–10, pp. 1774–1778, Sep. 2014.

[43] Y. Wang, H. Cai, L. A. B. Naviner, Y. Zhang, J. O. Klein, and W. S. Zhao, "Compact thermal modeling of spin transfer torque magnetic tunnel junction," *Microelectron. Rel.*, vol. 55, nos. 9–10, pp. 1649–1653, Aug. 2015.

[44] L. Amarú, P.-E. Gaillardon, and G. De Micheli, "Majority-inverter graph: A novel data-structure and algorithms for efficient logic optimization," in *Proc. IEEE Design Autom. Conf. (DAC)*, 2014, pp. 1–6.

[45] ABC—A System for Sequential Synthesis and Verification. (2005). *Berkeley Logic Synthesis and Verification Group*. [Online]. Available: https://people.eecs.berkeley.edu/~alanmi/abc/

MOL-Based In-Memory Computing of Binary Neural Networks

Khaled Alhaj Ali, Amer Baghdadi, *Senior Member, IEEE*, Elsa Dupraz, *Member, IEEE*, Mathieu Léonardon, *Member, IEEE*, Mostafa Rizk, and Jean-Philippe Diguet, *Senior Member, IEEE*

***Abstract*— Convolutional neural networks (CNNs) have proven very effective in a variety of practical applications involving artificial intelligence (AI). However, the layer depth of CNN deepens as user applications become more sophisticated, resulting in a huge number of operations and increased memory size. The massive amount of the produced intermediate data leads to intensive data movement between memory and computing cores causing a real bottleneck. In-memory computing (IMC) aims to address this bottleneck by directly computing inside memory, eliminating energy-intensive and time-consuming data movement. On the other hand, the emerging binary neural networks (BNNs), which is a special case of CNN, show a number of hardware-friendly properties, including memory saving. In BNN, the costly floating-point multiply-and-accumulate is replaced with lightweight bitwise XNOR and popcount operations. In this article, we propose an IMC programmable architecture targeting efficient implementation of BNN. Computational memories based on the recently introduced memristor overwrite logic (MOL) design style are employed. The architecture, which is presented in semiparallel and parallel models, efficiently executes the advanced quantization algorithm of XNOR-Net BNN. Performance evaluation based on the CIFAR-10 dataset demonstrates between 1.24× and 3× speedup and 49% and 99% energy saving compared to state-of-the-art implementations and up to 273-image/s/W throughput efficiency.**

***Index Terms*— Binary neural networks (BNNs), convolutional neural networks (CNNs), in-memory computing (IMC).**

I. Introduction

DEEP convolutional neural networks (CNNs) are the current state of the art for many computer vision tasks, such as image classification, detection, and localization [1], [2]. In particular, there is an increasing focus on the deployment of CNN in mobile systems, the Internet of Things (IoT) devices, and embedded chips for the mass market [3]. The main challenge that limits the integration of CNN in such systems is the requirement for a substantial amount of computation and memory. For instance, the VGG-19 network exhibits over 140 million floating-point (FP) parameters and requires more than 15 billion FP operations in order to classify one image [4]. Embedding such networks in traditional cores that deploy the von Neumann model (e.g., CPUs and GPUs) poses significant problems in terms of execution speed and power consumption. Massive intermediate data are produced during CNN execution revealing intensive I/O data congestion between memory and processing cores causing a real bottleneck.

Various prior works have been proposed to alleviate the hardware burdens in the von Neumann model in order to get better CNN inference performance. Two of the most promising solutions are network binarization [5]–[7] and in-memory computing (IMC) [8]. Network binarization or binary neural networks (BNNs) quantize all the weights and/or inputs to $+1$ and -1, providing a promising solution to mitigate storage and computation bottlenecks. In the resulting BNN, each convolution is processed by simple bitwise operations (XNORs and popcounts) instead of the multiply-and-accumulate (MAC). While BNNs are compact and efficient for resource-constrained devices, a degradation in accuracy is inevitable compared to their full precision counterparts. However, recent works have been carried out to reduce the decline in accuracy [5]. For instance, Courbariaux *et al.* [9] demonstrated only 3% loss in accuracy when applying BNN to the CIFAR-10 dataset. For the larger ImageNet dataset, Lin *et al.* [10] achieved promising results where the accuracy loss is around 5%.

On the other hand, IMC is one of the emerging techniques that address the memory wall problem encountered in the conventional von Neumann model [11], [12]. By merging processing cores and the memory component into a single unit, IMC allows to perform a part of the computation inside the memory, thus eliminating the need for data exchange. Although IMC is an old concept [8], it has been revisited recently with the advent of emerging nonvolatile memory (NVM) technologies where computing is efficiently enabled on the storage cells, directly on the data location. Several recent IMC architectures [13]–[15] have been developed based on NVM technologies, such as resistive memory (RRAM), magnetic memory (MRAM), and phase-change memory (PCM). Usually, IMC breaks arithmetic tasks into elementary logic operations that are successively executed within the memory cells. Although IMC can execute any arithmetic task, some tasks may be more

Manuscript received August 23, 2021; revised November 30, 2021 and February 13, 2022; accepted March 4, 2022. *(Corresponding author: Khaled Alhaj Ali.)*

Khaled Alhaj Ali, Amer Baghdadi, Elsa Dupraz, and Mathieu Léonardon are with IMT Atlantique, Lab-STICC, UMR CNRS 6285, 29238 Brest, France (e-mail: khaled.alhajali.lb@gmail.com).

Mostafa Rizk is with IMT Atlantique, Lab-STICC, UMR CNRS 6285, 29238 Brest, France, also with the School of Engineering, International University of Beirut, Beirut 1105, Lebanon, and also with the Physics Department, Faculty of Sciences, Lebanese University, Beirut 1105, Lebanon.

Jean-Philippe Diguet is with IRL CROSSING CNRS, Adelaide, SA 5000, Australia.

Color versions of one or more figures in this article are available at https://doi.org/10.1109/TVLSI.2022.3163233.

Digital Object Identifier 10.1109/TVLSI.2022.3163233

efficiently performed in classical CMOS implementations. In this case, dedicated near-memory units are usually added to handle such tasks. More recently, a computational memory (CMEM) architecture [13] based on memristor overwrite logic (MOL) design style has shown promising performance in terms of execution speed and throughput efficiency, especially for bitwise application tasks.

In fact, there is a great synergy between IMC and BNNs when they are combined: the low logic complexity of BNNs makes them well suited for in-memory implementation. In this context, we propose a novel MOL-based in-memory architecture design dedicated for BNNs. The related contributions can be listed as follows.

1) The proposed architecture efficiently implements a parallel rowwise in-memory XNOR-based convolution.
2) A novel mechanism for combining in-memory and near-memory execution is proposed for certain operations such as popcount and max pooling.
3) An original in-memory bubble sorting technique is introduced to execute a majority binarization stage replacing addition and normalization operations.
4) For evaluation, a python-based environment with appropriate library and commands has been developed by emulating the functionality of the adopted MOL-based CMEM and the corresponding control unit.
5) The new architecture is presented in the form of both semiparallel and parallel models and exhibits the lowest energy consumption compared to all existing relevant works, including CPU, GPU, and field-programmable gate array (FPGA).
6) The proposed parallel model reveals the lowest inference latency when compared to recent NVM-based approaches due to the high level of parallelism offered using MOL-based IMC.

The rest of this article is organized as follows. Section II provides an overview on CNN, BNN, and the adopted MOL-based IMC approach. Section III illustrates the devised methodology and algorithms for realizing a BNN inside the CMEM architecture. Section IV describes our proposed semiparallel and parallel architectures that are dedicated for BNNs. Section V presents the environment setup for performance evaluation and discusses the achieved results. Finally, Section VI concludes this article.

II. Preliminaries

This section briefly reviews the basics of CNN and BNN. In addition, it introduces the MOL-based IMC technique, which is adopted in this article.

A. CNN

A CNN is a particular type of neural network. It usually takes an image at the input and computes the probabilities that the image features belong to one of the output classes. Typically, a CNN consists of several convolutional and pooling layers followed by fully connected (FC) layers, as shown in Fig. 1(a). It has been shown that FC layers could be equivalently replaced by convolutions [16].

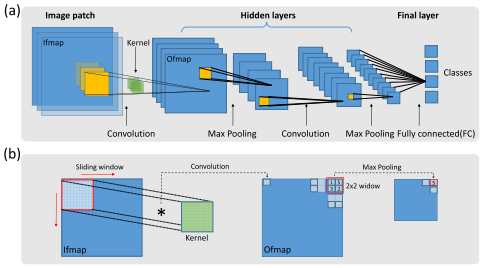


Fig. 1. Structure of the CNN: (a) multiple layers of CNN including convolution layer (CONV), pooling (POOL), and FC layer, and (b) illustration on the convolution and the max-pooling operation.

1) Convolutional Layers: As shown in Fig. 1, a convolutional layer takes an input feature map (Ifmap) represented by a set of channels/matrices and convolves them with a particular set of weights (called kernels) to generate an output feature map (Ofmap). The transfer from Ifmap to Ofmap follows the expression:

$$Y_m = f\left(b + \sum_{n=1}^{N} X_n * W_{n,m}\right). \quad (1)$$

In this expression, X_n represents an Ifmap channel of index n (where $n \in [\![1, N]\!]$) and Y_m represents an Ofmap channel of index m (where $m \in [\![1, M]\!]$). M and N are the number of channels of the Ifmap and Ofmap, respectively. $W_{n,m}$ is a $k \times k$ weight filter window linking X_n with Y_m. The parameter b is the bias. f represents the activation function.

2) Pooling Layers: Pooling is an important feature of CNN as it reduces the dimensionality of a feature map while maintaining the most important information [17]. It allows to reduce the size of the network and the number of parameters used, preventing overfitting. Considering the max pooling, a spatial neighborhood (e.g., a 2×2 window) is defined. The window is slided without overlapping on the Ofmap elaborated by the convolutional layer. The largest element inside that window is taken as an output. Another choice is to take the average (average pooling) or the sum of all elements in that window. In practice, max pooling has been shown to work better [17]. An intuitive example of max pooling is shown in Fig. 1(b).

B. BNN

The multiply–accumulate is the key and the most computationally expensive arithmetic operation in classical CNN. BNNs have been introduced to alleviate the need for these operations. This is achieved by forcing the inputs/weights/gradients to have binary values, especially in the forward propagation.

Various types of BNNs have been explored in the literature [9], [10], [18]. In this article, we adopt the XNOR-Net [18] BNN, which offers significant simplifications and better results than other binarization methods. In XNOR-Net, both the

(a)

XNOR truth table		
Encoding (Value)		XNOR (Multiply)
0 (-1)	0 (-1)	1 (+1)
0 (-1)	1 (+1)	0 (-1)
1 (+1)	0 (-1)	0 (-1)
1 (+1)	1 (+1)	1 (+1)

(b)

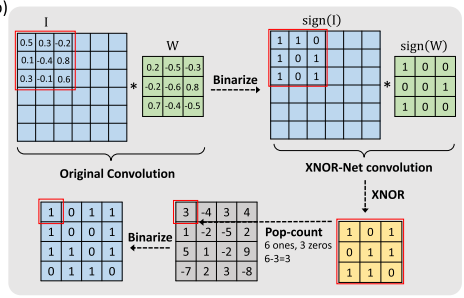


Fig. 2. BNN: (a) XNOR truth table and (b) XNOR-popcount process.

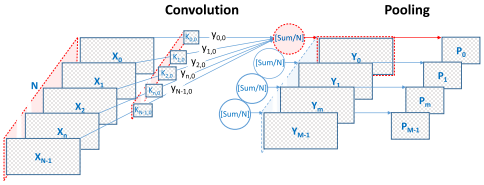


Fig. 3. Convolutional layer in a BNN followed by a pooling layer.

incoming activations and weight parameters of the convolutional layers are constrained to a binary set $\{-1,1\}$ except for the first convolutional layer where the input is the image. For efficient hardware mapping, the values -1 and 1 are encoded to logic "0" and "1," respectively. Then, multiplication of weights and activations is achieved according to the XNOR truth table, as shown in Fig. 2(a).

As shown in Fig. 2(b), the accompanied MAC operations can be then replaced by a series of XNOR operations and a final popcount (difference between the number of zeros and ones). The result is then subjected to normalization and binarization (Norm-bin). The convolutional layer in a XNOR-net BNN is shown in Fig. 3 and can be modeled by the following expressions:

$$y^b_{n,m} = \text{Norm-bin}\left(\text{Popcount}\left(\text{XNOR}\left(W^b_{n,m}, X^b_n\right)\right)\right) \quad (2)$$

$$Y^b_m = \text{Norm-bin}\left(\sum_{n=1}^{N} y^b_{n,m}\right) \quad (3)$$

where $y^b_{n,m}$ represents the output after convolving the nth binary Ifmap X^b_n with its corresponding binary weight kernel $W^b_{n,m}$ and Y^b_m represents the mth Ofmap after adding and binarizing all the N outputs $y^b_{n,m}$.

C. MOL-Based IMC

1) In-Memory Computing: IMC has been widely explored to overcome the memory wall by avoiding the long latency originated from intensive exchange of data between host processor and memory. From the device-level perspective, emerging NVMs are promising for the implementation of IMC. In this context, several recent contributions have been proposed and can be classified into two categories. The first category includes approaches that use the NVM cell as a single-level cell (SLC) [13]–[15]. In the second category, the authors have employed the NVM cell as a multilevel cell (MLC) or analog cell [19]–[21]. MLC crossbars can perform parallelized *in situ* operations by eliminating sequential memory accesses. MLC-based computing is promising when targeting applications with intensive MAC operation (e.g., CNN) [22]. However, a number of challenges remain in terms of manufacturability and computational accuracy regarding device variability, pattern-dependent current leakage, and the area overhead of peripheral circuits [23]. In contrast, the SLC approach involves a larger readout margin that makes NVM cells much tolerant against process variation and resistance drift effects.

Based on the SLC approach, various IMC techniques have been introduced in the literature [14], [15]. All these techniques attempt to realize arithmetic tasks inside NVM arrays by performing successive elementary logic operations on the stored data bits. For instance, the material implication (IMPLY) [15] and the memristor-aided logic (MAGIC) [24] have been introduced to enable in-place logic operations in memristive crossbar arrays. Although promising results have been demonstrated, these techniques present the following limitations.

1) The performance of IMPLY and MAGIC is highly dependent on the technology of the adopted NVM device (e.g., requirement of high ON–OFF margin) [14], [15]. Thus, they are not qualified for the operation with spintronic devices such as spin-transfer-torque (STT)-MRAMs.
2) The analysis in [25] reveals that IMPLY may incur partial switching and significant state drift issues [14], [15] of the NVM devices within the memory array.
3) The corresponding basis functions provided by IMPLY and MAGIC are not diverse enough to allow fast logic mapping with a minimum number of computational cycles.

Other IMC techniques, such as the sensing-based computing, that is introduced in [26], has gained large interest for its ease of implementation and the ability to execute diverse types of bitwise operations. Sensing-based computing redesigns the read circuitry so that it can compute the bitwise logic of two or more memory rows. Although fast, this technique involves a relatively high precision read circuitry that is based on sensing amplifiers (usually Op-Amps) employed as comparators. The read circuitry, which must be activated at each computational step, involves relatively high energy consumption.

In this article, the recently proposed IMC approach [13], namely MOL, is adopted. The main idea behind MOL

was to address the aforementioned limitations. MOL applies for various NVM technologies spanning resistive devices (memristors [27]), spintronics (STT magnetic tunnel junction (MTJ) [28]), and phase-change materials (PCMs [29]). Moreover, it allows for significant reduction in the number of required computational steps as well as the reserved processing area inside the memory.

2) MOL-Based CMEM: The CMEM presented in [13] is composed of two adjacent nonvolatile (NV) subarrays that work in a complementary manner. In this architecture, two wordlines are activated simultaneously in order to perform bitwise logic operations. A vector-level AND/OR operation can be executed within a single computational step. Moreover, shifting and inversion operations are also enabled through a dedicated intermediate driver offering flexibility in the operations, which is crucial for some arithmetic tasks. The intermediate driver involves a simple sensing circuitry followed by a set of 2-to-1 multiplexers. The role of multiplexers is to either pass the value from the sensed wordline or to pass its inverse. It can be configured depending on the desired operation. The multiplexers are followed by a configurable 1-bit shifter that can be enabled/disabled during the run time. The resulting intermediate driver passes the signals on-the-fly without additional computational steps. More details on the working mechanism of the intermediate driver can be found in [13].

An external controller arranges the operations performed inside the memory subarrays in order to finalize a desired arithmetic task. It breaks down the task into series of micro-operations (bitwise MOL) that are executed one after the other. The architecture diagram is presented in Fig. 4.

Generally, the high bandwidth of IMC allows to minimize the step period of each microoperation. However, when the complexity of an arithmetic task scales up, the corresponding number of computational steps becomes large, which again increases latency and energy consumption. In this case, moving data to be processed near memory is more efficient. Hence, the performance of IMC is task-dependent.

In this article, we consider applying the principle of MOL and the corresponding CMEM for the implementation of BNNs. For higher precision neural networks, as previously stated, the multiply–accumulate operation is the crucial operation in these networks. Addition and multiplication tasks can be executed in-memory but usually necessitate a large number of computational steps, which grows when increasing the size of operands (i.e., precision of the network). As a result, such tasks might be more efficient to implement in traditional von Neumann architectures, despite the high communication cost between memory and processing cores. In contrast, the low logic complexity of BNN makes it highly suitable for IMC as the multiply–accumulate operations of a convolutional layer are mainly replaced with basic bitwise XNOR and popcount.

III. MOL-Based in-Memory BNN

A. In-Memory XNOR-Based Convolution

1) Method: The bitwise XNOR represents the most computationally expensive operation in the convolution process.

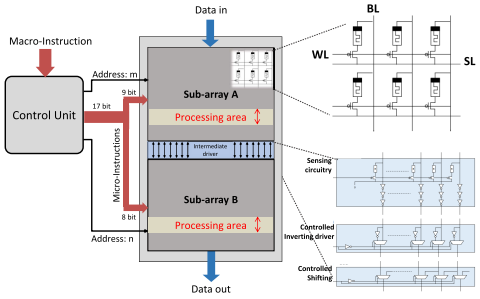


Fig. 4. Architecture diagram of the MOL-based CMEM.

Thus, optimizing its execution inside memory will enhance the overall performance of the BNN. This is why we first propose an efficient implementation of the XNOR convolution inside the CMEM. We consider a binary Ifmap X^b of size $h \times w$ being XNORed with a $k \times k$ weight kernel W^b. A simple example presented in Fig. 5(a) illustrates some of the steps of the proposed procedure. In this example, a 3×3 kernel is adopted. This kernel size is also suitable and used for large networks and datasets [30].

First, an Ifmap, zero padded at its surrounding, is loaded into subarray A of the CMEM, as shown in Fig. 5(a). The corresponding kernel is then loaded into subarray B, though in a tiled pattern fitting the width of the Ifmap. Hence, the size of the tiled kernel is $k \times w$. Here, we assume that the width after zero padding is a multiple of k; otherwise, padding is increased. For better scheduling of the XNOR operations and in order to obtain a higher level of parallelism, the Ifmap is gridded into $k \times k$ slots without overlapping. For simplicity, we use the term sliding grid (instead of sliding window) to point on the selected regions that would be XNORed. The convolution is performed in k successive phases (three phases in this example), while each phase is carried out in k rounds. For a given phase $i \in [\![0, k-1]\!]$ and round $j \in [\![0, k-1]\!]$, the sliding grid is right and down shifted to the position i and j, respectively. Simultaneously, the tiled kernel is right shifted to the position i. In other words, the horizontal movement of the tiled kernel follows that of the sliding grid. This specific shifting operating is enabled by the CMEM. Rowwise XNOR is then performed purely inside the CMEM based on a series of MOL operations that are discussed in Section III-A.2. At each position (i, j) of the sliding grid, a XNOR matrix is obtained in subarray B. The resulting matrix is then moved to a near-memory processing unit to undergo popcount and binarization. In fact, the popcount process is not adapted to IMC. Its implementation is inefficient in terms of the number of required computational steps. Fig. 5(b) shows the occupied regions inside the CMEM after execution, where auxiliary processing regions are required in subarrays A and B in order to finalize the convolution task.

2) Rowwise XNOR: The Ifmap undergoes several rounds of XNOR operations inside the CMEM. Thus, an Ifmap should

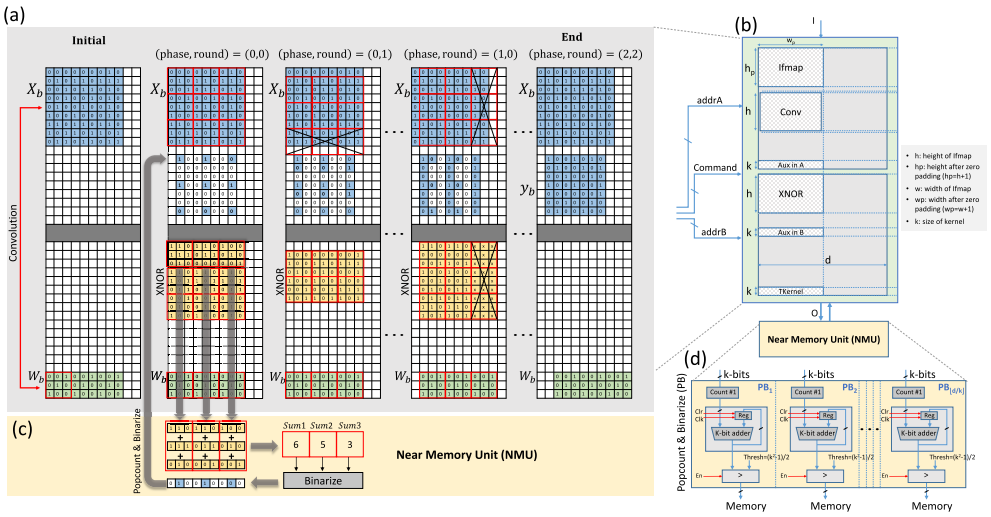


Fig. 5. Proposed in-memory XNOR-based convolution method: (a) samples of the CMEM state taken from different operation phases, (b) CMEM regions partitioning, (c) example on the near-memory popcount and binarization stage, and (d) architecture of the popcount and binarize (PB) block located inside the NMU.

remain safe until the end of the convolution process. Similarly, a safe version of the kernels should be available all the time in order to be reused for later inference. Within these requirements, a nondestructive rowwise XNOR is applied based on a series of MOL operations. As shown in Fig. 6(a), an Ifmap row X_r^b and a kernel row W_r^b are assumed to be initially inside the CMEM subarrays A and B, respectively. In order to avoid destroying W_r^b, a copy is stored in the processing region of subarray B. After that, five successive computational steps are required. In each step, a microoperation $z$at(m, n) is sent to the CMEM, where z corresponds to one of the 30 microinstructions described in [13] and m and n represent the corresponding addresses of subarrays A and B, respectively. The six microoperations are defined in Fig. 6(b). At the end, the value $X_r^b \oplus W_r^b$ is obtained in subarray B while keeping X_r^b and W_r^b safe. As a result, only three auxiliary rows have to be reserved for the XNOR operation to be processed. As presented in Fig. 6(a), one row resides in subarray A (at address m_2), while the other two rows reside in subarray B (at addresses n_1 and n_2). The same auxiliary rows can serve for later XNORs. Yet, it is preferred to change the location of the processing region regularly to avoid the thermal accumulation in certain cells and maintain high endurance. Overall, XNORing an Ifmap of size $h \times w$ located in subarray A, with a tiled kernel of size $k \times w$ located in subarray B, requires a minimum storage space of $S_{\text{XNOR}} = (h + k + 3)w$.

On the other hand, it is possible to retrieve the value of W_r^b logically instead of saving a spare copy. This is because the XNOR operation can be reversed. The value of W_r^b is still contained in the XNOR result. Yet, the required operations for retrieving W_r^b value are definitely more expensive in terms of computational steps, in particular that the same set of kernels is frequently needed for multiple operations and inferences. Therefore, it is more efficient in this case to use the copy-saving strategy.

3) Near-Memory Popcount: The XNOR matrix obtained in subarray B is also grouped into $k \times k$ slots using the sliding grid. Horizontal slots are then moved row by row to the popcount and binarize (PB) block located inside the near-memory unit (NMU), as shown in Fig. 5(c). In fact, the PB process is equivalent to obtaining the majority bit in a given slot. Hence, the PB block counts the number of ones in the horizontal slots simultaneously using parallel adders and accumulators. The resulting number in each slot is then compared with the threshold value $(k^2-1)/2$, where this value corresponds to half of the number of elements in each slot. Exceeding this value indicates that the corresponding majority bit is "1"; otherwise, the bit "0" is returned. The architecture of PB is shown in Fig. 5(d). At the end of k successive steps, a binary row is returned to the memory. The accumulators are cleared before beginning with the next horizontal slots.

B. BNN Layer in-Memory

1) Convolutional Layer: As illustrated in Section II-B, the mth Ofmap Y_m^b is obtained by adding, normalizing, and binarizing the resulting N convolution channels $y_{n,m}^b$ as expressed in (3). In fact, executing these three processes separately inside the CMEM is computationally expensive. In order to handle these processes efficiently by the CMEM, we introduce the majority-bin stage instead.

(a)

MOL-based XNOR (6 steps)

	Command / Address	Initial	6@(m, n)	9@(m, n)	19@(m, n)	7@(m, n)	14@(m, n)	17@(m, n)
Sub-array A	m_1	X^b	X^b	X^b	X^b	X^b	X^b	X^b
	m_2		W^b	W^b	W^b	W^b	W^bX^b	W^bX^b
Sub-array B	n_1			$\overline{W^b}$	$\overline{W^b}\,\overline{X^b}$	$\overline{W^b}\,\overline{X^b}$	$\overline{W^b}\,\overline{X^b}$	$\overline{W^b}\,\overline{X^b} + W^bX^b$ (X^b XNOR W^b)
	n_2					X^b	X^b	X^b
	n_3	W^b	W^b	W^b	W^b	W^b	W^b	W^b

(b)

$M_A(m)$: *memory content in subarray-A at address m*
$M_B(n)$: *memory content in subarray-B at address n*

Command: z@(m,n)	Description	Micro-operation
6@(m,n)	Copy	$M_A(m) = M_B(n)$
7@(m,n)	Copy	$M_B(n) = M_A(m)$
9@(m,n)	Invert	$M_B(n) = \overline{M_A(m)}$
14@(m,n)	AND	$M_A(m) = M_A(m)$ AND $M_B(n)$
17@(m,n)	OR	$M_B(n) = M_B(n)$ OR $M_A(m)$
19@(m,n)	AND-NOT	$M_B(n) = M_B(n)$ AND $\overline{M_A(m)}$

Fig. 6. Rowwise XNOR: (a) operations sequence for an MOL-based in-memory XNOR operation, where each column corresponds to the state of the CMEM at a certain step, and (b) definition of the six microinstructions used.

The majority-bin stage involves computing the majority channel out of the N convolution channels $y^b_{n,m}$. Computing the majority is simply achieved through a proposed in-memory bubble sorting technique. As shown in Fig. 7(a), we consider an arbitrary vector of bits $p = \{p_0, p_1, \ldots, p_i, \ldots, p_{N-1}\}$, where $i \in [\![0, N-1]\!]$. When applying bubble sorting on p, bits holding the value "1" are swept to one side of the vector leading to new vector $q = \{q_0, q_1, \ldots, q_i, \ldots, q_{N-1}\}$. Since the number of "0"s and "1"s after sorting remains the same, the middle bit $q_{N/2}$ (if N multiple of 2) can be considered as the majority bit.

The applied bubble sorting technique is implemented using the logic tree presented in Fig. 7(b), which has a regular form and is based on the proposed bubble-flip module shown in Fig. 7(c). The module is composed of only AND and OR logic gates. It allows to flip the input bits so that "1"s appear at the output port of the OR logic gate as shown in the figure. As an optimization step, since the goal is to compute the majority bit and not to sort the bits inside the vector, bubble-flip modules and logic gates that are not participating in the generation of the majority bit have been discarded from the logic tree. The resulting optimized logic tree has been efficiently realized inside the CMEM, where the equivalent MOL operations are implemented sequentially.

Moreover, operations inside the CMEM are performed on the vector level. Thus, channels sorting is achieved rowwise, speeding up the whole sorting process. Fig. 7(d) shows how rows are being sorted ending up with a majority channel Y^b_m located in the middle. The required number of steps to generate a majority channel in-memory is derived as $S_{\text{maj}} = h((3/2)N^2 - 4N + 3)$. The derived expression shows that the latency overhead of the sorting technique is independent of the width w of the channels. Although the number of steps grows quadratically when increasing the number of channels to be sorted, the high amount of parallelism at the vector level and CMEM banks level is sufficient to compensate for the time overhead of the sequential sorting process.

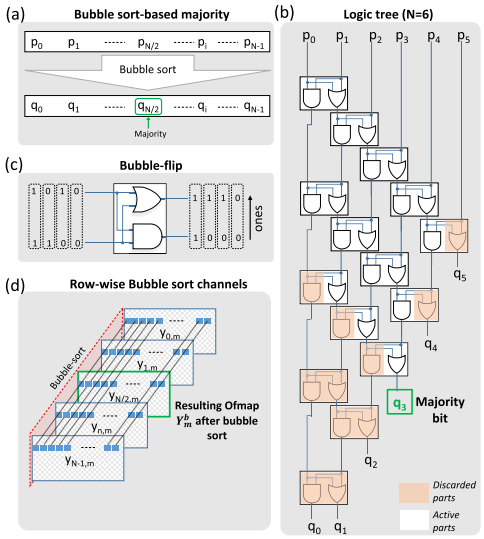


Fig. 7. Proposed bubble sort-based majority technique: (a) middle bit after bubble sort represents the majority bit, (b) used bubble-flip module, (c) logic tree implementation for input vector size N =6, and (d) in-memory rowwise channel sorting in order to obtain Ofmap Y^b_m.

In fact, the concept of sorting has already been adopted in the literature for replacing computationally expensive tasks. Zhang *et al.* [31] adopted the bitonic sorting algorithm in order to replace the FP addition and activation stages in neural networks. Alam *et al.* [32] implemented the bitonic sorter algorithm in memristive memory array achieving significant reduction in the processing time compared to prior sorting designs. Although bubble sort is slower than bitonic, it has a simple and more regular pattern for accessing the data to be sorted. This reduces the complexity of the control part of the architecture. Prasad *et al.* [33] proposed a general hardware/software design to achieve efficient sorting of floating-point numbers for data ranking. However, the mechanism imposes a custom design, which is not adapted for IMC of BNNs.

2) Pooling Layer: The input channel to the pooling layer is gridded into 2 × 2 slots without overlapping. Each slot corresponds to a max-pooling window $p = \{p_0, p_1, p_2, p_3\}$. The max pooling is defined as in [5]: $\max - \text{pool}(p) = 0$ if and only if all elements inside p is equal to "0"; otherwise, $\max - \text{pool}(p) = 1$. In fact, this can be handled by applying an elementary OR operation to the four elements inside p. The resulting bit corresponds to $\max - \text{pool}(p)$. Performing this inside memory is achieved in two stages. The first stage can be efficiently performed using in-memory MOL operations. Elementary OR operations are applied to the rows where all slots are managed simultaneously. Thus, the four bits in each

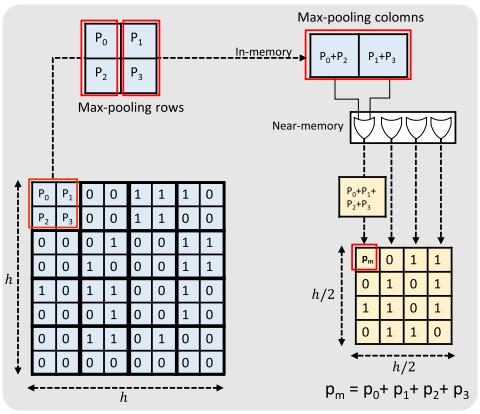


Fig. 8. Proposed in-memory and near-memory combined max-pooling process.

slot are compressed into two bits placed horizontally, as shown in Fig. 8. In the second stage, the resulting two bits are subjected to another OR operation resulting in $\max - \text{pool}(p)$. This stage is realized near memory using simple OR logic gates that are applied to the columns simultaneously.

IV. Semiparallel/Parallel Architecture

This section presents our proposed architecture for the execution of BNN in-memory. Then, the control flow and the data mapping methodology are illustrated.

A. Architecture Design

The execution of a complete BNN layer in a single CMEM block is time consuming, as convolutions would be executed serially. In order to improve parallelism and consequently accelerate processing, we propose to split the CMEM block into smaller interconnected CMEM units. Each CMEM unit is then employed to execute a single Ofmap channel Y_m^b in a given layer. The same units are reprogrammed to perform the other layers. The communication between these CMEM units is managed by a master memory that receives intermediate results and redistributes them in a systematic mapping method that is illustrated in Section IV-B.

Fig. 9 shows our proposed architecture with two different models. The semiparallel model shown in Fig. 9(a) shares one NMU to all CMEM banks, whereas in the parallel model shown in Fig. 9(b), each CMEM bank reserves a separate NMU. It is worth mentioning that all CMEM units are shared with a unique control bus because tasks being executed on different units are identical. Thus, a single control unit is able to cope with all these units. For this purpose, an enabling decoder block is added. It can be configured to either activate a single CMEM unit or all units at a time. Such an architecture model is equivalent to a single-instruction multiple-data (SIMD) model for IMC, so it can be called SIMM that stands for single-instruction multiple-memory.

B. Mapping Methodology

The adopted mapping method is an important factor in our proposed architecture as it highly influences the overall performance of inferencing. We classify the mapped data depending on its static or dynamic nature. Static data, such as weight kernels, is mapped during the configuration phase only and does not require any update during the running phase. On the other hand, the continuously generated data during the running phase is considered as dynamic. This type of data requires redistribution before initializing the next layer. Examples of dynamic data are the generated convolutions and Ofmap channels.

The control flow is thus divided into three phases: configuration phase, running phase, and redistribution phase.

1) *Configuration Phase:* As stated earlier, each CMEM unit is supposed to handle the computation of a single Ofmap channel in each layer. Thus, the mth CMEM unit (CMEM_m) receives from the master memory its own set of weight kernels ($W_{n,m}^b; n \in [\![0, N-1]\!]$) that are required to execute the Ofmap channel Y_m^b. Moreover, the master memory broadcasts the Ifmaps to all CMEM units.
2) *Running Phase:* The control unit receives a flag to begin the inference process. All CMEM units are then activated simultaneously for parallel execution. For the case of the semiparallel architecture model, popcounting and pooling stages are performed sequentially within the NMU. Thus, during these stages, only one CMEM unit is activated at a time. At the end of this phase, an Ofmap is generated in each CMEM unit.
3) *Redistribution Phase:* This phase prepares the execution of the next network layer. Normally, the generated Ofmaps are fed to the next layer as Ifmaps. They are returned one after the other to the master memory, which in turn rebroadcasts them to all CMEM units.

In fact, a single CMEM unit should have a certain minimum space to be capable of holding the inputs, the outputs, and the intermediate results. For a given layer with N Ifmaps and M Ofmaps, generating a single Ofmap requires N tiled kernels. Assuming the size of Ifmaps and tiled kernels as $h \times w$ and $k \times w$, respectively, the minimum storage space required to implement the layer can be expressed as $S_{\text{CMEM}} = w(2hN + kN + 3h)$.

V. Evaluation and Results

This section presents the validation and evaluation results of the proposed in-memory BNN architecture. In addition, it presents the detailed discussions and comparisons with recent relevant works.

A. Simulation Environment and Functional Validation

For functional validation, a python-based environment, with appropriate library and commands, has been developed to

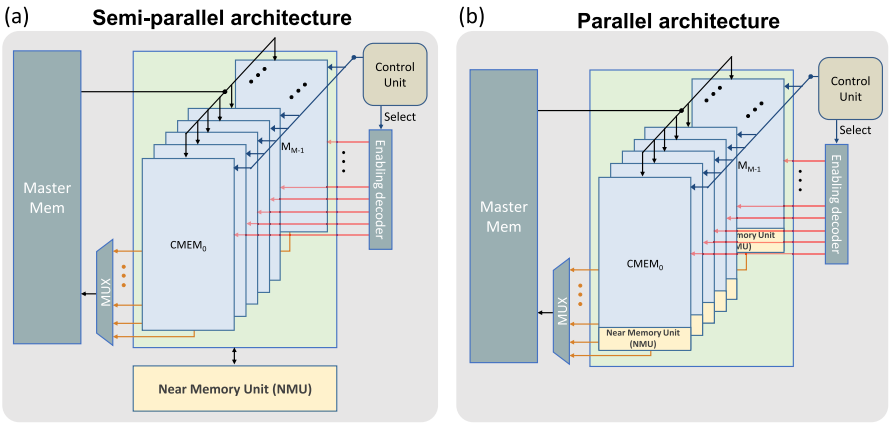


Fig. 9. Proposed in-memory BNN architecture with two different models: (a) semiparallel model shares one NMU to all CMEM banks and (b) parallel model where each CMEM unit reserves a separate NMU.

mimic the MOL-based IMC operations performed in the CMEM. This environment constitutes a proof-of-concept that illustrates the programmability of the proposed IMC architecture. It provides an abstract programming library that hides the complexity of the low-level logic implementation to application designers. In this environment, we deal with the CMEM subarrays as two coupled binary matrices, and each one has a separate input signal, which points to a certain row inside the matrix; "0"s and "1"s of the binary matrices correspond to the ON- and OFF-state of the NVM cells, respectively. The two matrices receive a command that is directly translated into a bitwise elementary operation where 30 types of operations are supported and described in [13]. The developed environment allows for system-level simulations as well as performance evaluation. Within this environment, the proposed in-memory XNOR-based convolution and the majority-bin and pooling operations have been simulated and cross-validated with standard methods. Moreover, the implementation of binary convolutional layers has been functionally validated.

Fig. 10 shows a simple example of a BNN layer implemented with four 28 × 28 Ifmaps taken from the Modified National Institute of Standards and Technology (MNIST) dataset. The black and white pixels represent the logic states "1" and "0" of the memory cells, respectively. This figure, which is extracted from the developed environment, corresponds to the state of the CMEM subarrays at three different instants. The first instant corresponds to the initial state, where four Ifmaps and a set of tiled kernels appear in subarrays A and B, respectively. The second instant is the end of the convolutional layer, where three Ofmaps are generated in subarray B. The last instant corresponds to the end of the combined in-memory and near-memory max-pooling process. The channels after rows and columns max-pooling processes appear inside subarrays A and B, respectively.

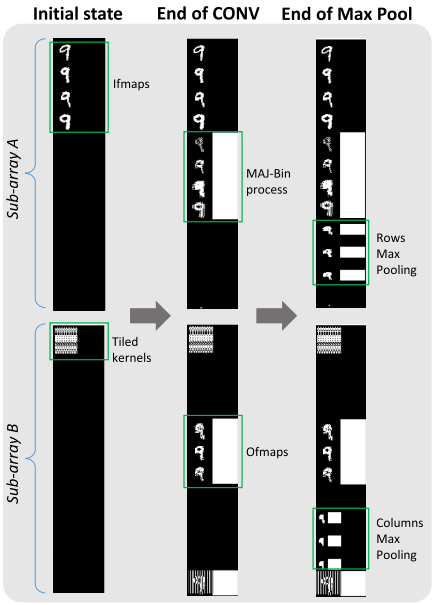


Fig. 10. CMEM state at three different instants: the initial state, the end of the convolutional layer, and the end of the max-pooling layer.

B. Performance Evaluation

In order to evaluate the performance of our proposed architecture, the binarized neural network BinaryNet [9] and the CIFAR-10 dataset have been taken as a case study. The BinaryNet model has been chosen as it achieves near

state-of-the-art results on CIFAR-10 and allows comparing with the available relevant works [3], [34], [35]. For the rest of this article, the adopted model is referred to as the CIFAR-10 BNN model. The BinaryNet BNN consists of six convolutional layers, three fully connected layers, and three pooling layers. All convolutional layers use 3×3 filters and edge padding. Both Ifmaps and weight kernels of the convolutional layers are binarized to -1 and 1 except for the first convolutional layer where the input is the image. All pooling layers employ a 2×2 max-pooling window without overlapping.

We consider the convolutional layers CONV2–5 in our evaluations. For CONV2, CONV3, CONV4, and CONV5 layers, the numbers of input and output channels are 128 and 128, 128 and 256, 256 and 256, and 256 and 512, respectively. The corresponding sizes of the Ifmaps are 32, 16, 16, and 8, respectively. More details on CIFAR-10 BNN model parameters can be found in [34].

For both parallel and semiparallel architectures, we employ 128 CMEM units, each of 34-bit width. The adopted width supports the maximum binary image size (i.e., 32×32) of the CIFAR-10 after edge padding. Layers with a number of output channels exceeding 128 are performed on several stages.

The minimum step period and the average energy consumption of the performed operations are extracted from the Cadence Virtuoso toolset. The 65-nm technology node is used for the peripherals, including the intermediate driver and NMUs, while a realistic STT-MTJ device model [28] is adopted for the CMEM memory cells. The model describes the static, dynamic, and stochastic behaviors of the STT-MTJ device. It has been proven through a resistance variability analysis in [13] that a variation up to 21% in tunnel magnetoresistance (TMR), t_{sl}, and t_{ox} parameters of the MTJ device leads to error-free MOL operations. In this work, we consider that the resistance variability is below this limit. According to [13], a single MOL operation consumes 0.196 pJ of energy per STT-MTJ cell, with a maximum switching delay of 1.8 ns. The reported value of energy includes the average energy consumed by the peripherals as well as the intermediate driver. On the other hand, the switching delay corresponds to a clock frequency of 555 MHz. Since the switching speed of MTJ devices is slower than that of CMOS, this frequency is set to the entire architecture, including the CMOS peripherals.

Moreover, evaluation is carried out using the emerging spin-orbit torque (SOT)-MTJ device, which exhibits fast read/write speed and unlimited endurance. The read/write energy and delay for this device are taken from [36]. The maximum estimated values of energy (0.1 pJ) and delay ($\sim$1 ns), for a single MOL operation, are then deduced according to the analytical expressions derived in [13]. Here, the delay corresponds to a maximum clock frequency of 1 GHz. The average energy consumed by a single MOL-type operation (AND, OR, AND-NOT, and OR-NOT) per MTJ device can be expressed as $E_{\mathrm{MOL}} = E_w/2 + E_r$, where E_r and E_w represent the read and write energy of the adopted MTJ device, respectively, [13]. The energy consumed by a copy operation can be expressed as $E_{\mathrm{COPY}} = E_w + E_r$. Other operations such as invert and shift, have an energy overhead close to that of a copy operation. A slight difference appears due to the change in the configuration of the intermediate driver. To evaluate the energy consumed by 34-bit width operations, the above expressions should be multiplied by a factor of 34. Other sources of energy consumption appear during operands movement from the memory array to the NMU. The NMU is located close to the memory array, and thus, energy consumption is mainly limited to the hybrid CMOS-resistive structure of the read circuitry, as well as the dynamic energy dissipation of the NMU's CMOS structure. Table I presents the energy consumption of the different 34-bit width operations performed in STT and SOT CMEM. This extracted information is provided to the developed python-based environment, which is in turn used as an evaluation tool.

TABLE I
ENERGY CONSUMED PER 34-BIT WIDTH OPERATION IN CMEM (IN pJ)

MTJ device	MOL	Copy	Invert	Shift	XNOR
STT-MTJ	6.66	11.32	11.93	12.3	54.4
SOT-MTJ	3.46	6.15	5.78	5.98	26.5

The total computational energy, power, latency, and throughput efficiency for the CONV2–5 layers are evaluated in both parallel and semiparallel architectures. A comparison is carried out with relevant recent state-of-the-art works implementing BNNs using a similar IMC approach [3], [35]. Moreover, BNN implementations using a conventional computing approach on Intel Xeon E5-2640 processor (CPU), NVIDIA Tesla K40 GPU, and FPGA [34] are considered, although the two approaches are hardly comparable. Indeed, the emerging STT/SOT memory technologies are still in the development phase and there is no mature production technology for them. In contrast, real fabricated devices using mature technologies are available for accurate performance measurement on CPUs, GPUs, and FPGAs. Nevertheless, comparing with these conventional implementation approaches, which we consider as baseline, still provides useful information about the performance level of our proposed architecture.

On the other hand, it is worth noting that the proposed IMC approach does not alter the accuracy of the BNN model with respect to the conventional computing approach. It only provides a different way to implement the same computing operations on the same parameters and same data.

Comparison results are summarized in Table II. As shown in this table, our proposed semiparallel and parallel architectures exhibit the lowest energy consumption regardless of the used NVM technology (STT/SOT). The SOT-based architecture reveals 49%–99% reduction in terms of energy compared to the other implementations. Although the parallel architecture shows better inference speed than the semiparallel one ($6.5\times$), the latter requires less number of near-memory units ($M\times$). The choice between these two architecture models could depend on the application requirements. Furthermore, our proposed architecture with SOT technology reaches 264 to 273-image/s/W throughput efficiency, which outperforms all others, including the CPU, GPU, and FPGA solutions.

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

TABLE II
PERFORMANCE IS COMPARED WITH STATE-OF-THE-ART DESIGNS FOR THE CIFAR-10 BNN MODEL

Item	CPU [34] Intel Xeon E5-2640	GPU [34] NVIDIA Tesla K40	FPGA [34] Zynq-7000 SoC	Read-SOT [35] (SOT)	PXNOR-BNN [3] (SOT)	This work (semi-parallel) (STT/SOT)	This work (parallel) (STT/SOT)
Technology node	32 nm	28 nm	28 nm	45 nm	28 nm	65 nm	65 nm
Clock frequency	2.5 GHz to 3 GHz	745 MHz to 875 MHz	143 MHz	500 MHz	1.4 GHz	555 MHz / 1 GHz	555 MHz / 1 GHz
Execution time (ms/image)	13.2	0.68	2.68	5.35	5.83	50 / 28.1	7.7 / 4.3
Power (W)	95	235	4.7	2.1	1.41	0.15 / 0.13	0.96 / 0.88
Energy (mJ/image)	1254	159.8	12.59	11.23	7.58	7.5 / 3.82	7.5 / 3.82
Throughput efficiency (image/sec/Watt)	0.7	6.2	79.4	89	131.8	133.3 / 273.7	135.2 / 264.2

It is worth noting that the semiparallel and fully parallel architectures have the same energy consumption although they have different implementation complexities. This is because they perform exactly the same operations, yet with different levels of parallelism. A slight difference appears due to the static energy consumed by the additional NMUs in the parallel architecture. However, this static energy is negligible with respect to the predominant dynamic energy consumed by the resistive switching behavior of the MTJ cells.

In terms of inference speed, the proposed parallel architecture with SOT technology achieves 1.24×, 1.35×, and 3× speedup compared to Read-SOT [35], Preset-XNOR (PXNOR)-BNN [3], and Intel Xeon E5-2640 processor (CPU) approaches, respectively, due to the high level of parallelism offered using MOL-based IMC. On the other hand, the GPU and FPGA implementations reveal higher inferencing speed (6.3× and 1.6×, respectively) compared to our SOT-based parallel architecture. However, this higher speed comes at the cost of 41× and 3.3× energy consumption. In this regard, it is worth noting that our architecture design adopts the CMOS 65-nm technology node. If the design is scaled to more recent technologies (e.g., 28 nm), we expect even better results in terms of speed and energy consumption. Finally, although the GPU and FPGA implementations present better results in terms of speed, still, they are less convenient for embedded applications such as the IoT devices where area and energy consumption are the most crucial.

C. Hardware Complexity Evaluation

The proposed architecture, which is presented in two models, is mainly composed of interconnected CMEMs, NMUs, and a control unit. The storage space of a CMEM unit should be sufficient to accommodate the Ifmaps, the resulting Ofmap, and the results of intermediate computations of the network layer. Each CMEM unit in the BinaryNet model should have a minimum of 36 kB of space, for a total of 4.5 MB for all CMEM units. It is worth noting that these units are based on the conventional 1T1M crossbar arrays and thus can benefit from the promising 3-D stacking technology of NVMs [37], [38]. Accordingly, the CMEMs crossbars can be stacked at the top of each other, leading to a compact design. In fact, it has not been possible to evaluate the area of the proposed architecture due to the lack of a layout model for the adopted MTJ devices. However, in order to get an estimation of the complexity, we evaluated the total number of components in each CMEM unit, particularly in the peripheral and intermediate drivers. Table III presents the number of utilized hardware resources for a 34-bit width as well as for a general d-bit width

TABLE III
NUMBER OF PERIPHERAL COMPONENTS PER CMEM UNIT

Type	# Components (d-bit width CMEM)	# Components (34-bit width CMEM)
2:1 Mux	3d	68
Inverters	8d+6	278
Transmission Gate	2d	68
Pass-MOSFET	6d+4	208
Ref-resistor	2d	68

TABLE IV
NUMBER OF COMPONENTS IN EACH NMU

Near memory unit	3-bit adder	3-bit comparator	3-bit register	1-bit full adder
d-bit width CMEM & k=3	$\lfloor d/3 \rfloor$	$\lfloor d/3 \rfloor$	$\lfloor d/3 \rfloor$	$\lfloor d/3 \rfloor$
34-bit width CMEM & k=3	11	11	11	11

$\lfloor . \rfloor$ is the Floor operator

CMEM unit. Here, it is worth noting that the reported number of resources grows when expanding the width d of a CMEM unit. Increasing the depth (i.e., number of wordlines) has no additional cost. Table IV presents the number of components involved in the NMU. The NMU is mainly composed of k-bit adders, constant comparators, and registers. In the presented case study, as we employed 128 CMEM banks, the parallel architecture integrates 128 NMUs, whereas the semiparallel architecture integrates only one NMU.

On the other hand, the functionality of the control unit has been emulated using a python-script code, which automatically generates the commands and the address signals to the interconnected CMEM units. It is considered that this functionality can be handled using a host processor, which is programmed according to the desired application (network model, dataset, and parameters). The use of a host processor ensures programmability requirement.

VI. CONCLUSION

In this work, we presented a novel IMC architecture targeting efficient implementation of BNN. The proposed architecture follows what we called SIMM parallelism model to execute instructions inside multiple computational memories. It employs the advanced quantization algorithm of BNN and the promising MOL-based IMC technique, which is well adapted for parallel bitwise operations. The complex MAC operations are replaced by parallel rowwise XNOR and popcounts. Moreover, the addition and normalization stage is replaced by the majority-bin stage, which is achieved through the proposed in-memory bubble sorting technique of channels. An efficient data mapping methodology has been deployed. For simulation and evaluation purposes,

a python-based environment with appropriate library and commands has been developed by emulating the functionality of the adopted MOL-based CMEM architecture. A study on the CIFAR-10 BNN model has demonstrated that a proposed parallel architecture, implemented in SOT-MTJ technology, has obtained a notable performance improvement compared to recent relevant state-of-the-art works. The results show 1.24× to 3× speedup compared with Read-SOT, PXNOR-BNN, and Intel Xeon E5-2640 processor (CPU) implementations. Moreover, the proposed parallel architecture outperforms all other approaches in terms of power and energy consumption, where 49%–99% reduction is achieved in terms of energy cost. Besides, it reaches 264–273-image/s/W throughput efficiency, which is much higher than all others, including the CPU, GPU, and FPGA solutions. Finally, the proposed architecture is scalable as it is able to handle larger network sizes and can in addition be compatibly applied to other types of emerging memory technologies.

REFERENCES

[1] S. Yu, S. Jia, and C. Xu, "Convolutional neural networks for hyperspectral image classification," *Neurocomputing*, vol. 219, pp. 88–98, Jan. 2017.

[2] C. Chen, A. Seff, A. Kornhauser, and J. Xiao, "Deepdriving: Learning affordance for direct perception in autonomous driving," in *Proc. IEEE Int. Conf. Comput. Vis.*, Dec. 2015, pp. 2722–2730.

[3] L. Chang, X. Ma, Z. Wang, Y. Zhang, Y. Xie, and W. Zhao, "PXNOR-BNN: In/With spin-orbit torque MRAM preset-XNOR operation-based binary neural networks," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 27, no. 11, pp. 2668–2679, Nov. 2019.

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

[5] T. Simons and D.-J. Lee, "A review of binarized neural networks," *Electronics*, vol. 8, no. 6, p. 661, Jun. 2019.

[6] H. Qin, R. Gong, X. Liu, X. Bai, J. Song, and N. Sebe, "Binary neural networks: A survey," *Pattern Recognit.*, vol. 105, Sep. 2020, Art. no. 107281.

[7] M. Courbariaux, Y. Bengio, and J.-P. David, "Binaryconnect: Training deep neural networks with binary weights during propagations," in *Proc. Adv. Neural Inf. Process. Syst.*, 2015, pp. 3123–3131.

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

[9] 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," 2016, *arXiv:1602.02830*.

[10] X. Lin, C. Zhao, and W. Pan, "Towards accurate binary convolutional neural network," 2017, *arXiv:1711.11294*.

[11] S. Hamdioui *et al.*, "Memristor for computing: Myth or reality?" in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, Mar. 2017, pp. 722–731.

[12] C.-J. Jhang, C.-X. Xue, J.-M. Hung, F.-C. Chang, and M.-F. Chang, "Challenges and trends of SRAM-based computing-in-memory for AI edge devices," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 68, no. 5, pp. 1773–1786, May 2021.

[13] K. A. Ali *et al.*, "Memristive computational memory using memristor overwrite logic (MOL)," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 28, no. 11, pp. 2370–2382, Nov. 2020.

[14] N. Talati, S. Gupta, P. Mane, and S. Kvatinsky, "Logic design within memristive memories using memristor-aided loGIC (MAGIC)," *IEEE Trans. Nanotechnol.*, vol. 15, no. 4, pp. 635–650, Jul. 2016.

[15] S. Kvatinsky, G. Satat, N. Wald, E. G. Friedman, A. Kolodny, and U. C. Weiser, "Memristor-based material implication (IMPLY) logic: Design principles and methodologies," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 22, no. 10, pp. 2054–2066, Oct. 2014.

[16] 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," 2016, *arXiv:1606.06160*.

[17] U. Karn, "An intuitive explanation of convolutional neural networks," Data Sci. Blog, Tech. Rep., 2016. [Online]. Available: https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/

[18] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, "XNOR-Net: ImageNet classification using binary convolutional neural networks," in *Proc. Eur. Conf. Comput. Vis.* Springer, 2016, pp. 525–542.

[19] C.-X. Xue *et al.*, "A 1 Mb multibit ReRAM computing-in-memory macro with 14.6 ns parallel MAC computing time for CNN based AI edge processors," in *IEEE ISSCC Dig. Tech. Papers*, Feb. 2019, pp. 388–390.

[20] W.-H. Chen *et al.*, "A 16 Mb dual-mode ReRAM macro with sub-14 ns computing-in-memory and memory functions enabled by self-write termination scheme," in *IEDM Tech. Dig.*, Dec. 2017, pp. 28.2.1–28.2.4.

[21] M. Hu, H. Li, Y. Chen, Q. Wu, G. S. Rose, and R. W. Linderman, "Memristor crossbar-based neuromorphic computing system: A case study," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 25, no. 10, pp. 1864–1878, Oct. 2014.

[22] C.-X. Xue *et al.*, "A 22 nm 2 Mb ReRAM compute-in-memory macro with 121-28TOPS/W for multibit MAC computing for tiny AI edge devices," in *IEEE ISSCC Dig. Tech. Papers*, Feb. 2020, pp. 244–246.

[23] W.-H. Chen *et al.*, "CMOS-integrated memristive non-volatile computing-in-memory for AI edge processors," *Nature Electron.*, vol. 2, no. 9, pp. 420–428, Sep. 2019.

[24] S. Kvatinsky *et al.*, "MAGIC—Memristor-aided logic," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 61, no. 11, pp. 895–899, Nov. 2014.

[25] X. Fang and Y. Tang, "Circuit analysis of the memristive stateful implication gate," *Electron. Lett.*, vol. 49, no. 20, pp. 1282–1283, Sep. 2013.

[26] S. Li, C. Xu, Q. Zou, J. Zhao, Y. Lu, and Y. Xie, "Pinatubo: A processing-in-memory architecture for bulk bitwise operations in emerging non-volatile memories," in *Proc. 53nd Design Automat. Conf. (DAC)*, Feb. 2016, pp. 1–6.

[27] D. B. Strukov, G. S. Snider, D. R. Stewart, and R. S. Williams, "The missing memristor found," *Nature*, vol. 453, no. 7191, pp. 80–83, May 2008.

[28] S. Ikeda *et al.*, "A perpendicular-anisotropy CoFeB–MgO magnetic tunnel junction," *Nature Mater.*, vol. 9, no. 9, pp. 721–724, 2010.

[29] H.-S. P. Wong *et al.*, "Phase change memory," *Proc. IEEE*, vol. 98, no. 12, pp. 2201–2227, Dec. 2010.

[30] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," Tech. Rep., Univ. Toronto, Toronto, ON, Canada, 2009. [Online]. Available: https://www.cs.toronto.edu/ kriz/learning-features-2009-TR.pdf, https://bibbase.org/network/publication/krizhevsky-hinton-learningmultiplelayersoffeaturesfromtinyimages-2009

[31] Y. Zhang *et al.*, "When sorting network meets parallel bitstreams: A fault-tolerant parallel ternary neural network accelerator based on stochastic computing," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, Mar. 2020, pp. 1287–1290.

[32] M. R. Alam, M. H. Najafi, and N. TaheriNejad, "Sorting in memristive memory," *J. Emerg. Technol. Comput. Syst.*, Jan. 2022, doi: 10.1145/3517181.

[33] A. K. Prasad, M. Rezaalipour, M. Dehyadegari, and M. N. Bojnordi, "Memristive data ranking," in *Proc. IEEE Int. Symp. High-Performance Comput. Archit. (HPCA)*, Feb. 2021, pp. 440–452.

[34] R. Zhao *et al.*, "Accelerating binarized convolutional neural networks with software-programmable FPGA," in *Proc. ACM/SIGDA Int. Symp. Field-Program. Gate Arrays*, 2017, pp. 15–24.

[35] S. Angizi, Z. He, and D. Fan, "PIMA-logic: A novel Processing-in-Memory architecture for highly flexible and energy-efficient logic computation," in *Proc. 55th ACM/ESDA/IEEE Design Automat. Conf. (DAC)*, Jun. 2018, pp. 1–6.

[36] Z. Wang *et al.*, "Proposal of toggle spin torques magnetic RAM for ultrafast computing," *IEEE Electron Device Lett.*, vol. 40, no. 5, pp. 726–729, May 2019.

[37] Y. Cao, G. Xing, H. Lin, N. Zhang, H. Zheng, and K. Wang, "Prospect of spin-orbitronic devices and their applications," *IScience*, vol. 23, no. 10, Oct. 2020, Art. no. 101614.

[38] Y. Huai *et al.*, "High density 3D cross-point STT-MRAM," in *Proc. IEEE Int. Memory Workshop (IMW)*, May 2018, pp. 1–4.

Khaled Alhaj Ali received the master's degree in signal, telecommunications, image and speech (STIP) from Lebanese University, Beirut, Lebanon, in 2016, and the Ph.D. degree in electronics from IMT Atlantique, Brest, France, in 2020.

He is currently a Postdoctoral Researcher with the Mathematical and Electrical Engineering Department, IMT Atlantique. His current research interest is focused on the use of emerging nonvolatile memory technologies for in-memory computing and neural networks.

Amer Baghdadi (Senior Member, IEEE) received the Engineering, M.Sc., and Ph.D. degrees from Grenoble INP (Institut National Polytechnique), Grenoble, France, in 1998 and 2002, respectively, and the Accreditation to Supervise Research (HDR) degree in sciences and technologies of information and communication from the Southern Brittany University, Lorient, France, in 2012.

He is currently a Professor with the Lab-STICC, IMT Atlantique, Brest, France. He leads the Algorithm-Architecture Interaction (2AI) Team with the Mathematical and Electrical Engineering Department, IMT Atlantique. He has recently contributed to FlexDEC-5G, H2020 METIS, FANTASTIC-5G, and EPIC Research Projects. His research activities are mainly focused on embedded system design for applications in digital communications and neural networks, with a special interest in flexible implementations, application-specific processor design, hardware accelerators, NoC and MPSoC architectures, energy-efficient designs, processing-in-memory, and nonvolatile memory-based design approaches.

Elsa Dupraz (Member, IEEE) was born in Paris, France. She received the master's degree in advanced systems of radio communications from ENS Cachan, University of Paris-Sud, Bures-sur-Yvette, France, in 2010, and the Ph.D. degree in physics from the University of Paris-Sud, in 2013.

From January 2014 to September 2015, she held a postdoctoral position with ETIS (ENSEA, University Cergy-Pointoise, CNRS), Cergy-Pontoise, France, and the Department of Electrical and Computer Engineering, The University of Arizona, Tucson, AZ, USA. Since October 2015, she has been an Assistant Professor with IMT Atlantique, Brest, France. Her research interests lie in the area of coding and information theory, with a special interest on distributed source coding, LDPC codes, and energy-efficient channel codes.

Mathieu Léonardon (Member, IEEE) received the M.Sc. degree from Bordeaux INP, Bordeaux, France, in 2015, and the joint Ph.D. degree in electronics engineering from Polytechnique Montréal, Montreal, QC, Canada, and the University of Bordeaux, Bordeaux, in 2018.

From 2018 to 2019, he was a Lecturer with Bordeaux INP. He has been an Associate Professor with IMT Atlantique, Brest, France, since 2020. His research interests are on hardware and software implementations. These include two main fields of application: signal processing and in particular error-correcting code decoders on the one hand, and deep neural networks on the other hand.

Mostafa Rizk received the Maitrise degree in electronics, the M.Sc. degree in biomedical physics, and the M.Sc. degree in signal, telecom, image, and speech from Lebanese University, Beirut, Lebanon, in 2007, 2008, and 2010, respectively, the Ph.D. degree in sciences and technologies of information and communication from Telecom Bretagne, Brest, France, in 2014, and the Ph.D. degree in electronics and telecommunication from Lebanese University in 2015.

He was a Postdoctoral Researcher with the Université de Bretagne-Sud (UBS), Lorient, France, and the Lab-STICC, CNRS, Brest. He was an Associate Professor with Lebanese International University (LIU), Beirut, and an Associate Researcher with IMT Atlantique, Brest. He is currently a Researcher with Lab-STICC-UMR 6285 CNRS. His general research interests include both algorithm development and corresponding hardware/software co-design for embedded architectures, circuits, and systems; NoC design and new MPSoC architectures based on emerging computing paradigms using memories and memristive technologies; ICT of drone systems; and embedded machine learning and embedded computer vision.

Jean-Philippe Diguet (Senior Member, IEEE) received the Ph.D. degree from Rennes University, Rennes, France, in 1996.

In 1997, he was a Visitor Researcher with IMEC, Leuven, Belgium. He was an Associate Professor with Université de Bretagne-Sud (UBS), Lorient, France, until 2002. In 2003, he co-funded the Dixip Company, Lorient, France, in the domain of wireless embedded systems. Since 2004, he has been a CNRS Researcher with the Lab-STICC, Brest, France, where he has been heading the MOCS team until 2016. He was a Visiting Researcher with The University of Queensland, Brisbane, QLD, Australia, in 2010, and an invited Professor at Tohoku University, Sendai, Japan, from November 2014 to May 2019, and the University of São Paulo, São Paulo, Brazil, in November 2016. He is currently the CNRS Director of Research with the Lab-STICC. His current work focuses on various aspects of embedded system design: designs and tools for NoC-based MPSoC architectures, including memory-based computing, self-adaptivity for uncertain environments as autonomous vehicles, and design of dedicated hardware accelerators.

Chapter 7

Efficient Implementations of Machine Learning Algorithms

7.1 Preface

This research work has been accomplished during the PhD thesis of Mr. Hamoud Younes which has been conducted in the Department of Electrical, Electronics and Telecommunication Engineering and Naval Architecture (DITEN) at the University of Genoa, Italy under the supervision of Professor Maurizio Valle. It addresses embedded machine learning with emphasis on hardware accelerators and approximate computing targeting the application of tactile data processing. It has been valorized in four well recognized journals, six IEEE international conferences, and one book chapter.

7.2 Introduction

Embedding machine learning algorithms are constrained by the high computational complexity of such algorithms. Moreover, the amount of data processed by machine learning is increasing exponentially [151]. On the contrary, the processing resources are limited especially with the chip shortage in the last few years [152]. This poses challenges relevant to the requirements of real-time execution and low power/energy consumption when targeting portable wearable systems due to their limitation in terms of resources and energy budget. Nonetheless, machine learning is one of many application domains that has intrinsic tolerance to inaccuracy. These applications are mostly not about calculating a precise numerical answer; instead, ”correctness” is defined as providing an outcome that is good enough, or of sufficient quality to achieve an acceptable application performance [153].

Electronic skin (e-skin) is an artificial skin that mimics the sensing capabilities of human skin. It is usually composed of distributed tactile sensors integrated with an embedded electronic system for tactile data decoding. Meaningful information such as texture classification and pattern recognition can be decoded from tactile data by employing Machine Learning (ML) algorithms. Computations using embedded machine learning algorithms may enable

the electronic skin system to be used in various application domains such as wearable, Internet of Things, prosthetic and robotics.

In this perspective, this research work aims at providing efficient implementations of embedded machine learning algorithms for tactile data processing. The core strategy behind delivering efficient implementations is the use of *"Hardware Accelerators"* and *"Approximate Computing"* to accelerate demanding portions of ML algorithms and to adequately reduce the algorithms computational complexity respectively. The implementations should offer real-time processing with a time latency less than 400 ms [154] with as much reduced hardware area and energy consumption as possible compared to existing solutions.

7.3 Context and Motivations

Tactile sensing involves the detection of motion, the measurement of contact parameters, the processing of the signals to extract structured and meaningful information, and the transmission of such information into a higher system levels for interpretation [155]. The data acquired from tactile sensors corresponds to an electrical stimulus. The latter varies according to the type of the sensing material, dimensionality, responsiveness, and structure of the sensor. Processing algorithms should be able to decode and efficiently handle the acquired data. Tactile data processing algorithms presented in the literature could be divided into two categories: pre-processing [156, 157, 158, 159, 160] and classification/regression . Pre-processing algorithms involve feature extraction and dimensionality reduction, while classification and regression algorithms are mainly machine learning algorithms.

Machine learning algorithms are an efficient solution for processing tactile data in various applications [161]. ML algorithms in general, can extract a complex, non linear input–output relationship based on learning by example approach. An ML algorithm is trained using a set of examples, where each example is described by a group of informative features. ML algorithms can support intelligent and predictive systems that can make accurate decisions on unseen data. In this perspective, classification/regression problems supported by ML algorithms could be adopted in applications with tactile information. Several previous works have adopted ML algorithms for tactile sensing [162, 163, 164, 165].

Embedding machine learning in resource-limited and battery-powered applications for tactile data processing must abide with the ever increasing requirements of small hardware area, low time latency, and low energy consumption, which are mostly impacted by the computational complexity of ML algorithms Designing hardware accelerators is a common trend to implement ML algorithms efficiently, where high performance modules are assigned to complex and demanding blocks of an algorithm. Designing a hardware accelerator can be performed using several methodologies. The use of "specific design" methodology suitable for the hardware platform and application. Another designing methodology is the use of high performance cores in multi-core central processing units (CPUs) or graphics processing units (GPUs) for demanding tasks and assign less performing cores for simpler tasks. Recently, a new hardware accelerator design methodology focuses on techniques such as Near-Memory Computing (NMC) and Processing-In-Memory (PIM), have emerged to bring computing as close as possible to the memory arrays to reduce the cost of data movement between computing cores and memories [166].

In this work, "specific design" methodology targeting different machine learning algorithms and hardware devices/platforms is adopted. Mainly, the design and implementation of hardware accelerators for k-Nearest Neighbor (kNN), Support Vector Machine (SVM), and Binary CNNs. Then, a set of approximate computing methods for enhancing the accelerators' performance is discussed. The performance of the exact and approximate accelerators is assessed on a tactile data processing application, mainly touch modality classification.

7.4 Contributions and Performed Work

This work addresses the design and implementation of efficient ML algorithms to handle the challenges retaled to the computational complexity, the performance/availability of hardware platforms, and the application's budget (power constraint, real-time operation, etc.). First, Approximate Computing Techniques (ACTs) are used to reduce the computational complexity of ML algorithms. Then, custom Hardware Accelerators are designed to improve the performance of the implementation within a specified budget. Finally, a tactile data processing application is adopted for the validation of the proposed exact and approximate embedded machine learning accelerators.
The contributions of this work are listed below:

- Propose an exact and approximate kNN classifiers with a classification accuracy comparable with [161] for a touch modality recognition task. The experimental results demonstrate the effectiveness of Data-oriented ACTs on reducing the memory usage and execution time of the KNN classifier with an acceptable accuracy loss.

- Propose an approach for applying algorithmic level ACTs on machine learning algorithms, where each ACT in the proposed approach can serve as a quality configurable knob to trade-off quality for time latency.

- Conduct an overview about energy efficient implementation of machine learning algorithms on hardware platforms highlighting the main challenges when embedding such algorithms. The techniques that could be applied to improve the energy efficiency are reported. Furthermore, the main factors to be taken into consideration when choosing the appropriate platform are highlighted. Lastly, the strategies to overcome the challenges when building an energy efficient embedded machine learning systems are discussed.

- Survey the existing algorithms and tasks applied for tactile data processing. The survey presents the algorithms and tasks, which include machine learning, deep learning, feature extraction, and dimensionality reduction. Moreover, this survey provides guidelines for selecting appropriate hardware platforms for the algorithm's implementation. The algorithms are compared in terms of computational complexity and hardware implementation requirements.

- Perform a comprehensive assessment of applying algorithmic level approximate computing techniques on the FPGA implementation of tensorial SVM.

- Propose an architecture for Singular Value Decomposition (SVD) computation based on approximate computing techniques. The architecture is based on a shallow neural network for finding the SVD of an input matrix with two different dimensions. We provide the structure, tuning, and training of the network. Also, the FPGA implementation of the proposed neural network inference is presented. Implementation results show that the proposed network achieves a significant speedup and reductions in the required hardware resources and power consumption respectively compared to the traditional one-sided Jacobi algorithm.
- The first hardware implementation of a neural network based SVM featuring multi-dimensional tensorial inputs is proposed. The implementation is feasible for real-time touch modality classification with low power consumption. Moreover, the implementation scalability shows that the neural network based SVM is adequate for the acceleration of SVM on resource-limited hardware platforms.
- The design and implementation of a kNN accelerator using a selection based sorter (Selector) is proposed. The proposed accelerator overcomes similar state of the art solutions by reducing the occupied hardware area while providing noticeable speedups.
- A Hybrid fixed-point binary Convolution Neural Network (HCNN) model for touch modality classification is presented. A hardware accelerator architecture and implementation on FPGA for HCNN is proposed. The proposed accelerator can classify an input touch with a higher accuracy compared to SVM and Deep CNN. Moreover, a faster classification time is noticed while providing a low energy per classification value.

The following subsections provide a brief explanation of the proposed architectures and the achieved results along with comparisons with state-of-the-art works.

7.4.1 Algorithmic Level Approximate Computing Techniques for Machine Learning

Applications in domains like computer vision, media processing, machine learning, etc. have intrinsic tolerance to inaccuracy. Studies repeatedly show that such applications consist of both critical and non-critical components [167, 168, 169]. Thus, it is not necessary for every arithmetic operation to be precisely correct and every bit of memory to be preserved at the same level of reliability. From the perspective of approximate computing, not every operation in a program needs the same level of accuracy. Auto-tuning approaches can help empirically identify error-resilient components [170]. However, since machine learning algorithms don't present the same level of accuracy for all applications, the identification process varies from one application to another.

ACTs on the algorithmic level have been firstly investigated in [171]. Authors introduced the concept of incremental refinement. Such concept aims at reducing the number of iterations of iterative processing. A factor of ten reduction in power consumption has been recorded for a speech finite impulse response (FIR) filter implementation. In [172], authors explored the different parameters of the SVM algorithm using algorithm level scaling to reduce the complexity of hardware implementations. A $1.2\times - 2.2\times$ energy savings

have been achieved without any significant accuracy loss. Nogues *et. al* have presented an approach on how to apply algorithmic level approximate computing techniques on HEVC decoding [173]. Authors offered a strategy on how to locate the error-resilient components of the decoder and which approximate technique to be applied. Energy reductions of up to 40% are demonstrated for a limited degradation of the application Quality of Service (QoS).

In this research, we propose an approach for applying algorithmic level approximate computing techniques on machine learning algorithms. The approach is based on the work presented in [173]. kNN and tensorial SVM (TSVM) algorithms are adopted for the evaluation of the proposed approach. The evaluation is performed on both software and hardware levels. For the software evaluation, we monitored the loss in classification accuracy of a touch modality problem presented in [161] with respect to the gain in execution time and memory requirements. The software evaluation has been accomplished on an Intel CPU.

For the hardware evaluation, a FPGA implementation of both algorithms is presented. Then, the gain in hardware area, time latency, and power consumption is recorded when each technique is embedded in the hardware implementation. Results have shown that the kNN execution time and memory usage can be reduced up to 38% and 55% respectively. Similarly, a 29.6% power reduction and speedup up to 3.7$\times$ can be achieved with an approximate kNN FPGA implementation. As for approximate TSVM, the implementation achieves a reduction in power consumption by up to 49% with a speedup of 3.2$\times$. All these reductions are accompanied with a classification accuracy loss than 10%.

This research work has been published in the proceeding of the *IEEE International Conference on Ph.D Research in Microelectronics and Electronics (PRIME)* [174] and the proceeding of the *IEEE International Conference on Electronics, Circuits and Systems (ICECS)* [175] This research work has been also published in the *Electronics* [176].

electronics

MDPI

Article

Algorithmic-Level Approximate Tensorial SVM Using High-Level Synthesis on FPGA

Hamoud Younes [1,2], Ali Ibrahim [1,2,*], Mostafa Rizk [2] and Maurizio Valle [1]

[1] Department of Electrical, Electronic and Telecommunications Engineering and Naval Architecture, University of Genova, 16145 Genova, Italy; Hamoud.Younes@edu.unige.it (H.Y); Maurizio.Valle@unige.it (M.V)
[2] Department of Computer and Communication Engineering, Lebanese International University, Beirut 1105, Lebanon; Mostafa.rizk@liu.edu.lb
* Correspondence: Ali.Ibrahim@edu.unige.it; Tel.:+39-3279-364-917

Abstract: Approximate Computing Techniques (ACT) are promising solutions towards the achievement of reduced energy, time latency and hardware size for embedded implementations of machine learning algorithms. In this paper, we present the first FPGA implementation of an approximate tensorial Support Vector Machine (SVM) classifier with algorithmic level ACTs using High-Level Synthesis (HLS). A touch modality classification framework was adopted to validate the effectiveness of the proposed implementation. When compared to exact implementation presented in the state-of-the-art, the proposed implementation achieves a reduction in power consumption by up to 49% with a speedup of 3.2×. Moreover, the hardware resources are reduced by 40% while consuming 82% less energy in classifying an input touch with an accuracy loss less than 5%.

Keywords: approximate computing; embedded machine learning; tensorial kerne; high-level synthesis; tactile sensing

Citation: Younes, H.; Ibrahim, A.; Rizk, M.; Valle, M. Algorithmic-Level Approximate Tensorial SVM Using High-Level Synthesis on FPGA. *Electronics* **2021**, *10*, 205. http://doi.org/10.3390/electronics10020205

Received: 31 December 2020
Accepted: 15 January 2021
Published: 17 January 2021

1. Introduction

Machine Learning (ML) algorithms are efficient solutions for various tasks including speech recognition, tactile data classification and image processing. Consequently, embedding machine learning is increasingly used in several application domains such as prosthesis, Internet of Things (IoT), robotics, smart appliances and wearable devices. Support Vector Machine (SVM) is one of the most used supervised algorithms as it exploits complex relationships among data samples by using "Kernels" to create an optimal hyperplane that separates different classes [1]. Despite having a high classification accuracy, SVM is characterized by its computational complexity. Thus, hardware implementations dedicated to SVM impose additional overhead in terms of power consumption and execution time [2]. This overhead adds a design challenge when targeting real-time embedded systems. Several hardware implementations have been presented using different computing platforms that fulfill the requirements in terms of limited hardware resources, low power consumption and low latency. These platforms include Advanced RISC Machines (ARM), Application-Specific Integrated Circuits (ASIC) and Field-Programmable Gate Array (FPGA). The authors of [3] showed that the implementation of the tensorial SVM (TSVM) algorithm on an ARM Cortex M4 microcontroller (STM32F405) operating at 165 MHz classifies an input touch in 7 s. The latter is higher than the classification time obtained using the FPGA device presented in [4], which is about 400 ms. As for ASIC, the implementation of machine learning inference is characterized by several challenges such as: reconfigurability and design cycle complexity. On the other hand, FPGAs are suitable for prototyping where their features of programmability make them more cost-effective than ASICs. Thus, FPGAs are suitable platforms for implementing such algorithms. In addition, FPGAs are desirable for ASIC prototyping. In this perspective, FPGA has been

proposed as a hardware platform for implementing SVM due to its powerful and parallel processing as a re-configurable device with an efficient utilization of hardware resources [2].

The authors of [5,6] presented a hardware implementation of the linear SVM model that classifies biomedical data targeting FPGA devices. The implementations proposed in [5,6] have achieved a speedup of 85× and 6×, respectively, when compared to similar implementations targeting General-Purpose Processor (GPP). Mandal et al. [7] used a multiplier less-kernel architecture to implement a polynomial based SVM. The architecture leads to a power reduction of 3.5% compared to the use of vector-based kernel. A 2D pipelined streaming architecture with a Radial Basis Function (RBF) kernel implementation is proposed in [8], achieving a 2× speed improvement. In [4], the authors presented an FPGA implementation of SVM based on the tensorial kernel approach, which is proposed in [9]. The advantage of such implementation is that the tensorial representation of data preserves the implicit structure of the original data. Sidiropoulos et al. [10] conducted a survey that demonstrates the importance of using tensorial data in signal processing and machine learning. The survey covers several algorithms and applications, including optimization and statistical performance algorithms, collaborative filtering, classification and multilinear subspace learning. In addition, tensorial representation is recommended for applications such as image and tactile processing [9,11].

Approximate computing has emerged as a promising solution to obtain considerable resource utilization, energy and time savings at the expense of acceptable accuracy loss [12]. There exist notable hardware implementations of SVM in the literature using ACTs on both the algorithmic and circuit levels. Van Leussen et al. [13] presented a hardware architecture with re-configurable kernels and overflow resilient limiter. The proposed architecture achieved a saving of 15% in the consumed energy and 14% in the implementation area for the epileptic seizure detection application compared to a fully-exact implementation. The traditional Ripple-Carry Adder (RCA) has been replaced by an approximate accumulator ,which achieves up to 70% power reduction for the kernel computations in SVM for hyper-spectral image classification problem [14]. The authors of [15] designed an approximate adder and fixed-width multiplier with a low-cost compensation. Adopting the devised adder and multiplier in SVM classifier leads to reduce the power-delay product (PDP), area and critical path delay by 32.4%, 18.7% and 16%, respectively. All the mentioned architectures imply an accuracy loss less than 8% for the target applications. However, to the best of our knowledge, there is currently no implementations of approximate tensorial SVM classifier presented in the literature.

The work presented in this paper aims to reduce the hardware complexity of the tensorial SVM algorithm using algorithmic level ACTs. A touch modality binary classification problem was adopted to validate the proposed implementation. The exact tensorial SVM classifier in [9] as taken as a reference, and then the impact of ACTs on reducing power consumption, hardware resources and time latency was analyzed. High-Level Synthesis (HLS) with Vivado 2018.3 and Virtex-7 FPGA was used for the implementation. HLS design is adopted since it offers: (1) faster development process compared to RTL design; (2) built-in optimization directives; and (3) easier design manipulation through high-level programming languages (e.g., C/C++). The rest of the paper is organized as follows. Section 2 presents the algorithmic level approximate computing techniques and how they are applied on machine learning methods. Section 3 provides a general overview of the tensorial approach. Section 4 details the proposed approximate tensorial SVM architecture and implementation. Section 5 presents an assessment of the hardware implementation results targeting FPGA using HLS. Finally, Section 6 concludes the paper and illustrates the future work.

2. Algorithmic Level Approximate Computing

Approximate computing techniques can be applied at algorithmic, architecture and circuit levels [16]. Algorithmic level techniques are divided into two categories: data-

oriented and process-oriented. The authors of [17] presented an approach on how to apply these techniques on machine learning algorithms, as shown in Figure 1.

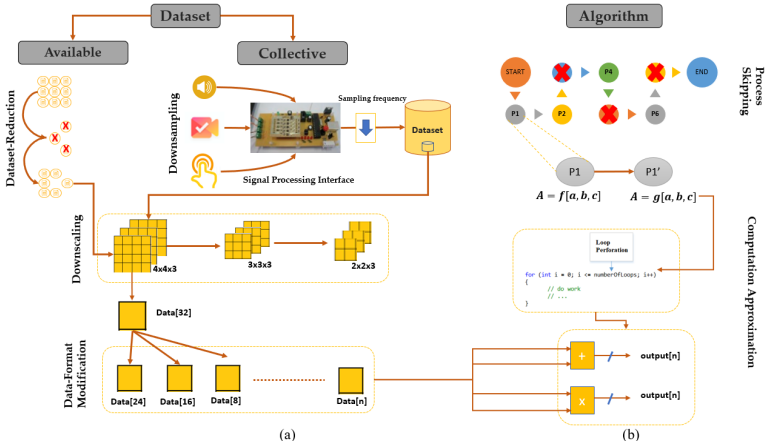


Figure 1. Algorithmic level Approximate Computing Techniques: (**a**) data-oriented; and (**b**) processing-oriented. Adopted from [17].

The data-oriented category involves modifying the data properties (size and bit-width) to minimize the work-load on the circuit level. This category includes:

- Dataset Reduction (DsR) decreases the amount of the processed data by eliminating samples randomly or using a subsampling method as the one proposed in [9]. Furthermore, DsR can be applied through downsampling and downscaling. The former adjusts the sampling frequency of the electronic interface used to collect raw data samples from sensors in the time domain, while the latter reduces the dimension of the collected data themselves (e.g., reducing the tensor size from $4 \times 4 \times 3$ to $3 \times 3 \times 3$), as shown in Figure 1.
- Data Format Modification (DFM) reduces the bit-width of the data and its corresponding arithmetic operations. This can be done by replacing floating-point representation with a fixed-point one. For instance, a 24-bit fixed-point representation data is adopted in [18] instead of floating-point to represent tactile data with a negligible precision loss.

The process oriented category targets the algorithm itself by reducing the number of tasks or replacing some of them with a less-complex counterpart. This category includes [19]:

- Computation Skipping (CS) skips a certain number of tasks in an algorithm. If these tasks are loop iterations, then it is referred to as Loop Perforation (LP). For example, in some machine learning applications, a pre-processing task such as data normalization may be skipped without affecting the quality of service of the target application.
- Computation Approximation (CA) proposes an equivalent version of a computationally complex function. The two versions should be mathematically equivalent with an acceptable output error margin. For example, a division function could be replaced by a reciprocal multiplication [19].

3. Tensorial SVM Algorithm

Gastaldo et al. [9] presented a tensorial kernel approach for touch modalities classification. The approach involves four main steps: (i) Unfolding includes the transformation of a third-order tensor $T(I_1 \times I_2 \times I_3)$ into three matrices $M(I_1 \times I_2 I_3)$, $N(I_2 \times I_1 I_3)$ and

$P(I_3 \times I_1 I_2)$. (ii) Singular Value Decomposition (SVD) is used to find the eigenvectors V for each of the three unfolded matrices as:

$$A = USV^T \tag{1}$$

where U and V^T contain the left and right singular vectors, respectively, and S is the diagonal matrix storing the singular values σ_i of A. SVD computations are achieved by using the one sided Jacobi algorithm, which is considered as one of the fastest methods to converge compared to other algorithms [20]. (iii) Kernel computation extends the Gaussian kernels to tensorial patterns. The kernel function can be expressed as:

$$K(x, y) = \prod_{1}^{z} k^z(x, y) \tag{2}$$

where k^z is the kernel factor defined as:

$$k(x, y) = \exp(\frac{-1}{2\sigma^2}(I_n - trace(Z^T Z))) \tag{3}$$

where $Z = V_x^T V_y$, V_x represents the singular vectors of the unfolded matrix, V_y represents the singular vectors of the unfolded matrix obtained from the training phase and trace represents the sum of diagonal elements. (iv) SVM classification applies the classification function expressed in the following equation:

$$y = f_{SVM}(x) = \sum_{i}^{N_p} \beta_i K(x, y) + b \tag{4}$$

where y represents the predicted label of an input x and β_i represents the coefficients obtained during training with a bias b.

4. Approximate Tensorial SVM

The proposed architecture of Approximate SVM is presented in Figure 2. The architecture is an extension to the exact architecture presented in [21]. For the rest of the paper, the latter is referred to as Exact SVM. It involves two stages: offline learning and online inference.

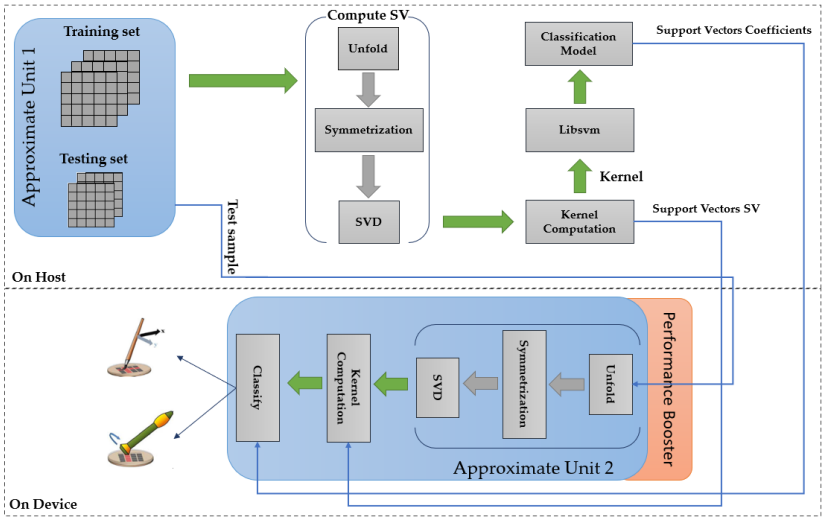


Figure 2. Sketch of the proposed Approximate Tensorial SVM.

4.1. Touch Modality Framework

The touch modality classification problem "Brushing a brush vs. Rolling a washer" referred to as Problem A in [9] was adopted to verify the proposed architecture. This problem is based on a dataset that includes readings from 70 participants. The tactile dataset was acquired by conducting experiments to sense the human touch pressure levels on a 4×4 tactile sensor array for a duration of 10 s with a sampling rate of 3000 samples per second. Thus, each modality can be described by a tensor $\phi(4 \times 4 \times 30{,}000)$. The final dataset was split into training and testing sets with $N_t = 180$ and $N_c = 80$. The proposed algorithmic-level approximate computing method discussed in Section 2 was applied on the exact tensorial SVM for the mentioned classification problem. Table 1 shows the effect of using this method on the classification accuracy.

Table 1. Effect of approximate computing techniques on classification accuracy.

Approximate Computing Technique	Accuracy (%)
None (Exact SVM)	90.47
10% Data Set Reduction	90.47
20% Data Set Reduction	80.95
30% Data Set Reduction	80.95
Loop perforation with sf = 2	90.47
Loop perforation with sf = 3	85.71
Loop perforation with sf = 4	80.95
DFM (24-bit)	85.71
DFM (16-bit)	75

The dataset reduction was applied by randomly removing samples from the original dataset. To ensure credible assessment, if a sample is removed during the 10% reduction, it is automatically removed for the 20% and 30% reductions. Loop perforation was applied on the loops of SVD computation block with a skipping factor sf (i.e., how many loops are perforated). As for data format modification, 24- and 16-bit fixed-point representations were applied for all the SVM operations with $< 8, 16 >$ and $< 6, 10 >$ precision, respectively, using the C libraries used in [18]. Based on the obtained results, the training and inference of the exact tensorial SVM were incorporated with the ACTs that resulted in an acceptable trade-off between accuracy and complexity. Combining several ACTs is also known as "cross-layer approximate computing" [12].

4.2. Offline Learning

The SVM training was conducted offline on a PC with Intel i7 CPU. The training process starts by activating the AU1 (Approximate Unit 1) (see Figure 2). AU1 applies dataset-reduction and downscaling techniques on the dataset by performing the following steps:

- The dataset size is reduced by eliminating data that corresponds to five participants with noisy readings. Figure 3a shows an example of such reading where the voltage is almost constant along the measured time. Therefore, the machine learning model will not learn new information from such sample. Hence, it is removed.
- During data collection of the tactile dataset, no precise instructions were given to the participants regarding the amount of pressure to be applied on the sensor [9]. Thus, some touch samples with silent readings where observed such as the one presented in Figure 3b. Such samples could be pre-processed to extract meaningful information in certain time frame. Each sample is truncated from 10 to 3.3 s by omitting readings outside the interval [3.7, 7] s. This results in a new tensor $\phi(4 \times 4 \times 10{,}500)$.
- To reduce the computational complexity of the tensor-based learning algorithms, the tensor size could be reduced without the loss of information originality using

subsampling. The latter is applied by truncating each sample into a new tensor $\phi(4 \times 4 \times 40)$ with 40 random time readings.

Then, the resulting tensor is unfolded into three matrices $M(4 \times 160)$, $N(4 \times 160)$ and $P(40 \times 16)$ that have to be symmetrized before applying SVD. The resulting support vectors along with the Gaussian parameter $\sigma = 1$ are fed to the kernel computation block (see Figure 2). The block outputs the kernel matrices for $(+1$ vs. $-1)$, $(+1$ vs. $+1)$, $(-1$ vs. $+1)$ and $(-1$ vs. $-1)$ binary classification problems where each row being labeled with the corresponding class label. This step is essential since LIBSVM [22] does not support tensorial kernels by default but can receive precomputed kernels. The LIBSVM library is used to obtain a classification model based on the precomputed kernel. The model contains the coefficients β_i and the bias b.

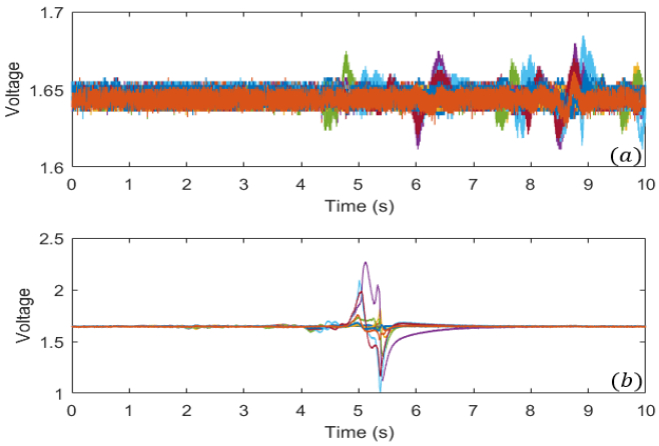


Figure 3. Touch Modalities: (**a**) touch with noisy readings; and (**b**) touch with silent intervals.

4.3. Online Inference

The SVM inference of the proposed architecture was implemented on FPGA through the steps shown in Figure 4. The architecture was coded in C++ using Vivado HLS. Then, the architecture was optimized using HLS directives and synthesized to ensure that it fits in the target FPGA device. Then, a C/RTL simulation was performed to ensure a coherent output from the architecture coded in C++ and the RTL design provided by Vivado HLS. Afterward, the architecture was exported as an RTL IP block targeting a Virtex-7 XC7VX980T FPGA operating at a clock frequency of 120 MHz. The IP block was imported into Vivado; then, a behavioral/combinational simulation was performed to verify the integrity of the exported IP. Then, place and route was performed to implement the architecture on the FPGA device. Finally, a detailed report about the utilized hardware resources, the number of clock cycles and the power consumption was obtained once the implementation was completed.

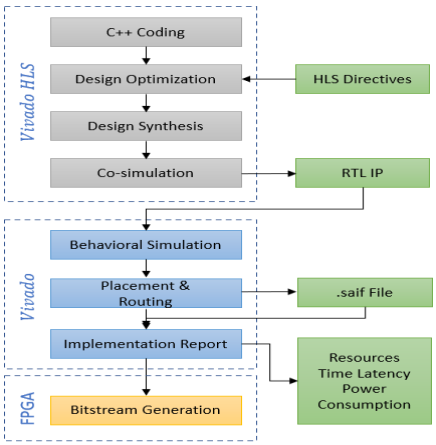


Figure 4. FPGA implementation process.

The inference starts by fetching a sample tensor from the testing set (that was already approximated using AU1). The selected tensor undergoes the unfolding, symmetrization and SVD processes. The obtained support vectors along with the support vectors from the training phase are provided to the kernel computation block. During online inference, Approximate Unit 2 is active. It operates by applying:

- Loop perforation (LP) technique to the SVD block with a skipping factor sf. The support vectors are obtained using the one side Jacobi Algorithm. The latter is an iterative algorithm, thus it is perforated with $sf = 2$. This technique accelerates the SVD computations but a large sf could not be applied to ensure the algorithm's convergence.
- Computation Approximation (CA) to the computation of Z in Equation (3). The obtained singular vector matrices from the SVD block are $V1(160 \times 160)$, $V2(160 \times 160)$ and $V3(16 \times 16)$. These matrices are truncated to $V1'(160 \times 4)$, $V2'(160 \times 4)$ and $V3'(16 \times 2)$. Such truncation reduces the complexity of the matrix multiplication in Equation (3) with an acceptable error margin. This technique was also applied in the offline training phase so that the equation $Z = V_x^T V_y$ has correct dimensions.
- Data Format Modification (DFM) to all the variables and arithmetic operations in different SVM blocks. HLS offers a library called "apfixed", which allows the declaration of variables with fixed-point precision. This declaration is limited by an upper bound [23]. Specifically, the mathematical functions are Square Root ($sqrt$), which is used in SVD calculations, and Exponential (exp), which is used in kernel computations. These functions are supported only for bit-widths $w \leq 32$ and $w \leq 16$, respectively. This limitation was resolved by a variable precision architecture. Hence, all the inference blocks are implemented with 24- bit fixed-point representation with a $< 12, 12 >$ precision except the kernel computation block.

Finally, the output of the kernel computation (i.e., a kernel) is used by the classification block to predict a class for the tested tensor according to Equation (4).

4.4. Performance Booster

The performance of the proposed architecture was enhanced to achieve the lowest possible time latency for applications with timing constraints [24] while increasing the throughput. These requirements are usually accompanied by an increase in hardware

resources, but the use of algorithmic level approximate computing techniques, specifically "dataset reduction" and "data format modification", would compensate such increase.

These requirements are facilitated by the use of Vivado HLS optimization directives [23]. The used directives are:

- Array Partition: tThis directive partitions a large BRAM occupied by a multidimensional array into smaller separate memories. The array partitioning can be complete, cyclic or block. The latter was applied on the tensor $\phi(4 \times 4 \times 40)$ with block size = 16, as shown in Figure 5. This results in an RTL IP block with smaller memories while improving the throughput of the Unfolding process.
- Dataflow: This directive allows functions to overlap in their operations, enhancing the overall throughput and latency of the design. The functions unfold and symmetrization are executed in a task-level pipelining using this directive, as shown in Figure 6.
- Pipelining: This directive allows the parallel execution of loop iterations, hence reducing the time latency. The computation of Z in Equation (3) is executed in parallel, as shown in Figure 7.

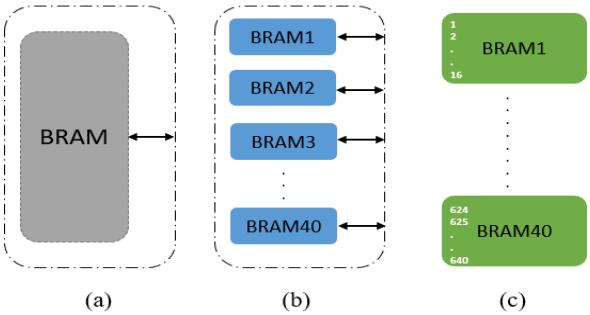


Figure 5. Array partitioning: (**a**) without partitioning; (**b**) block partitioning; and (**c**) block with size=16.

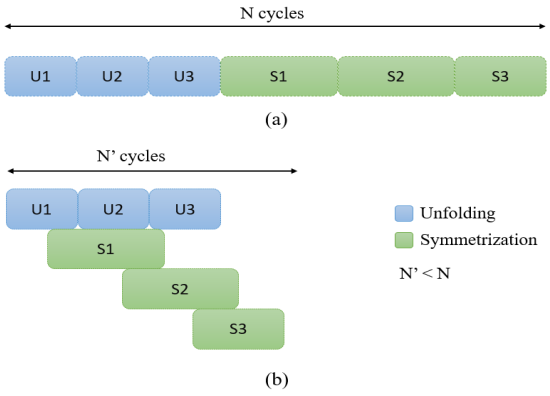


Figure 6. Dataflow pipelining: (**a**) without dataflow; and (**b**) with dataflow.

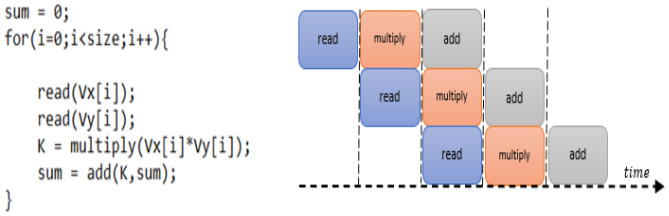


Figure 7. Pipeline directive applied on vector multiplication.

5. FPGA Implementation Results and Assessment

5.1. Implementation Results

Figure 8 shows the normalized speedup and reduction in power consumption while assessing different approximate computing techniques. The latter are applied one-by-one resulting in eight different FPGA implementations. Each implementation is compared to the exact implementation where the time latency (L) is recorded as:

$$L = N \times \frac{1}{f_{max}} \tag{5}$$

where N is the number of clock cycles and f_{max} is the maximum operating frequency. As for the power consumption, a vector-based method was adopted as it provides the power consumption related to the processing under a defined testbench. The method involves generating a "saif" file via post-implementation functional and timing simulations.

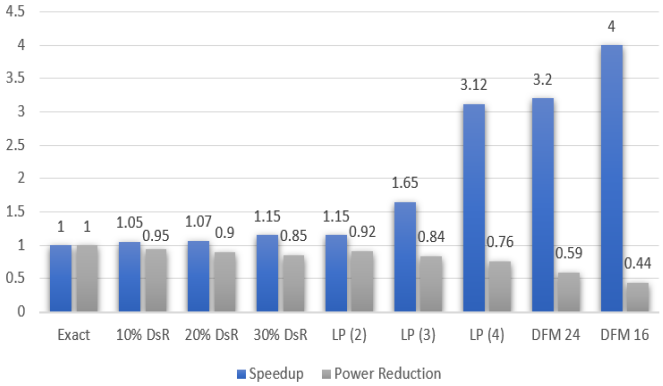


Figure 8. Speedup and power consumption reduction under different ACTs.

Using the obtained results in Figure 8, a cross-layer Approximate SVM implementation was performed where the adopted techniques are: 10% dataset reduction, loop perforation with $sf = 2$ and 24-bit DFM. Moreover, the implementation details was recorded with the performance booster ON and OFF to differentiate between the gain due to approximate computing techniques with and without HLS optimization directives. Table 2 summarizes the performance profile for the FPGA implementations based on the architecture in Figure 2. The Exact SVM is based on the architecture presented in [21]. The boosted Approximate

SVM corresponds to the Approximate SVM where the "Performance Booster" block is activated (See Figure 2), i.e., with HLS optimization directives. The reduction is calculated as:

$$Reduction(\%) = 100 - (\frac{I_{approx}}{I_{exact}} * 100) \quad (6)$$

where I_{approx} and I_{exact} are the implementation element (FF, DSP, LUT, etc.) of the Approximate (or boosted approximate) and Exact SVM, respectively. As for the energy per classification, it is calculated using the equation:

$$E = P \times T \quad (7)$$

where T is the time latency and P is the dynamic power consumption reported in Vivado.

Table 2. FPGA performance profile of Exact and Approximate SVM.

	FF	LUT	DSP	BRAM	SRL	Time Latency (s)	Power Consumption (W)	Energy per Classification (J)	Classification Accuracy (%)
Exact SVM	37057	42261	475	297	1060	2.4	6.3	15.12	90
Approximate SVM	17,187	25,558	283	291	202	0.91	3.12	2.83	86
Boosted Approximate SVM	17,197	25,588	284	292	203	0.75	3.2	2.4	86
Approximate to Exact Reduction	53.62%	39.5%	40.4%	2.02%	80.94%	2.64×	50.4%	81.28%	−4%
Boosted Approximate to Exact Reduction	53.59%	39.45%	40.21%	1.68%	80.84%	3.2×	49.2%	84.12%	−4%

5.2. Implementation Assessment

The obtained results presented in Tables 1 and 2 and Figure 8 demonstrate the effectiveness of using approximate computing techniques to reduce the hardware resources utilization, time latency and power consumption of the FPGA implementation of the tensorial SVM. Such reductions are accompanied by an accuracy loss that varies between 0% and 10%. Another set of remarks can be noticed:

- In general, loop perforation achieves lower latency and power consumption compared to dataset reduction with a comparable accuracy loss. This can be justified since the SVD computation block is among the most complex blocks of the tensorial SVM, as reported in [4].
- The transition to fixed-point representation results in the lowest latency and power consumption compared to other methods. This is expected due to the reduced complexity of the arithmetic operations based on fixed-point representation. This can be seen in the reduced number of required DSPs between the exact and approximate implementations. However, this comes at the expense of high accuracy loss; for example, the use of a 16-bit fixed-point led to a 15% accuracy loss for the target application.
- The number of used BRAMs is high since we are not using any external DRAM for memory read/write operations. The range of the number of LUT and DSPs is expected due to the level of parallelism introduced using HLS directives. For the target FPGA, this is not a problem as long as we obtained a relatively reduced time latency and power consumption in the case of Approximate SVM.
- Using "cross-layer" approximate computing: with an accuracy degradation of 4%, the Approximate SVM requires about 43% less hardware resources and classifies an unseen sample 2.64× faster while consuming 50% less power compared to its exact counterpart.
- The accuracy loss due to the use of "cross-layer" approximate computing is not the sum of the losses obtained for each single approximate technique. This is evident in the final results presented in Table 2.

- The use of ACTs shows a remarkable reduction in the energy per classification up to 82%, since such techniques affect both the time latency and power consumption of the TSVM, as shown in Table 2.
- Applying the adopted HLS optimization directives offered an additional speedup gain to the Approximate SVM in terms of speedup up to 3.2× accompanied with 84% less energy per classification.. This added a negligible overhead less than 1% increase in the hardware resources and power consumption. This is expected due to the fact that pipelining offers a reduction in the number of clock cycles while increasing the resources/power consumption. However, such increase is compensated by the dataflow directive that allows resource sharing, providing an enhanced overall implementation.

6. Conclusions

This paper presents the first FPGA implementation using High-Level Synthesis of an approximate tensorial SVM classifier. An accuracy of 86% was obtained with a speedup of 3.2× and 49% power consumption reduction resulting in up to 82% reduction in the energy per classification. Such results were achieved by utilizing the concept of cross-layer approximate computing. Combining several algorithmic level approximate computing techniques demonstrated their efficiency in optimizing the proposed embedded machine learning implementation. Moreover, specific design choices (e.g., using pipe-lined architecture) could boost the implementation performance with the help of Vivado HLS directives. The obtained reduction factor in hardware resources, time latency, and power consumption through applying ACTs paves the way towards applying such techniques on the RTL HDL design of the TSVM presented in [4] where similar reductions are expected. Such promising results motivate the exploration of different types of ACTs such as circuit-level techniques or the use of neural networks to further reduce the impact of computationally expensive singular value decomposition, as it represents about 70% of the whole implementation.

Author Contributions: Conceptualization, A.I.; methodology, H.Y. and A.I.; software, H.Y.; validation, M.V., A.I. and M.R.; investigation, H.Y. and A.I.; data curation, H.Y.; writing—original draft preparation, H.Y.; writing—review and editing, A.I., M.V. and M.R.; visualization, H.Y.; supervision, A.I., M.V. and M.R.; and funding acquisition, M.V. All authors read and agreed to the published version of the manuscript.

Funding: This research was funded by "TACTIle feedback enriched virtual interaction through virtual realITY and beyond (Tactility)": EU H2020, Topic ICT-25-2018-2020, RIA, Proposal ID 856718.

Conflicts of Interest: The authors declare no conflict of interest.

Abbreviations

The following abbreviations are used in this manuscript:

ACT	Approximate Computing Technique
ARM	Advanced RISC Machines
ASIC	Application-Specific Integrated Circuits
FPGA	Field Programmable Gate Array
GPP	General Purpose Processor
HLS	High Level Synthesis
PDP	Power Delay Product
RCA	Ripple Carry Adder
SVD	Singular Value Decomposition
SVM	Support Vector Machine

References

1. Nayak, J.; Naik, B.; Behera, H.S. A Comprehensive Survey on Support Vector Machine in Data Mining Tasks: Applications & Challenges. *IJDTA* **2015**, *8*, 169–186, doi:10.14257/ijdta.2015.8.1.18.

2. Shereen Moataz Afifi, Hamid Gholam Hosseini and Roopak Sinha. Hardware Implementations of SVM on FPGA: A State-of-the-Art Review of Current Practice. *Int. J. Innov.* **2015**, *2*, 2348–7968.
3. Magno, M.; Ibrahim, A.; Pullini, A.; Valle, M.; Benini, L. Energy Efficient System for Tactile Data Decoding Using an Ultra-Low Power Parallel Platform. In *2017 New Generation of CAS (NGCAS)*; IEEE: Genova, Italy, 2017; pp. 17–20, doi:10.1109/NGCAS.2017.56.
4. Ibrahim, A.; Valle, M. Real-Time Embedded Machine Learning for Tensorial Tactile Data Processing. *IEEE Trans. Circuits Syst. Regul. Pap.* **2018**, *65*, 3897–3906, doi:10.1109/TCSI.2018.2852260.
5. Hussain, H.M.; Benkrid, K.; Seker, H. Reconfiguration-based implementation of SVM classifier on FPGA for Classifying Microarray data. In Proceedings of the 2013 35th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC), Osaka, Japan, 3–7 July 2013; pp. 3058–3061, doi:10.1109/EMBC.2013.6610186.
6. Hussain, H.; Benkrid, K.; Şeker, H. Novel dynamic partial reconfiguration implementations of the support vector machine classifier on FPGA. *Turk. J. Electr. Eng. Comput. Sci.* **2016**, *24*, 3371–3387. doi:10.3906/elk-1402-18.
7. Mandal, B.; Sarma, M.P.; Sarma, K.K. *Implementation of Systolic Array Based SVM Classifier Using Multiplierless Kernel*; 2014; pp. 35–39. Available online: http://www.wseas.us/e-library/conferences/2014/Brasov/ACMOS/ACMOS-46.pdf (accessed on 5 November 2020).
8. Vranjković, V.S.; Struharik, R.J.R.; Novak, L.A. Reconfigurable Hardware for Machine Learning Applications. *J. Circuits Syst. Comput.* **2015**, *24*, 1550064, doi:10.1142/S0218126615500644.
9. Gastaldo, P.; Pinna, L.; Seminara, L.; Valle, M.; Zunino, R. Computational Intelligence Techniques for Tactile Sensing Systems. *Sensors* **2014**, *14*, 10952–10976, doi:10.3390/s140610952.
10. Sidiropoulos, N.D.; De Lathauwer, L.; Fu, X.; Huang, K.; Papalexakis, E.E.; Faloutsos, C. Tensor Decomposition for Signal Processing and Machine Learning. *IEEE Trans. Signal Process.* **2017**, *65*, 3551–3582, doi:10.1109/TSP.2017.2690524.
11. Signoretto, M.; De Lathauwer, L.; Suykens, J.A. A kernel-based framework to tensorial data analysis. *Neural Netw.* **2011**, *24*, 861–874, doi:10.1016/j.neunet.2011.05.011.
12. Mittal, S. A Survey of Techniques for Approximate Computing. *ACM Comput. Surv.* **2016**, *48*, 1–33, doi:10.1145/2893356.
13. Van Leussen, M.; Huisken, J.; Wang, L.; Jiao, H.; Gyvez, J.P.D. Reconfigurable Support Vector Machine Classifier with Approximate Computing. In Proceedings of the 2017 IEEE Computer Society Annual Symposium on VLSI (ISVLSI), Bochum, Germany, 3–5 July 2017; pp. 13–18, doi:10.1109/ISVLSI.2017.13.
14. Wu, Y.; Yang, X.; Plaza, A.; Qiao, F.; Gao, L.; Zhang, B.; Cui, Y. Approximate Computing of Remotely Sensed Data: SVM Hyperspectral Image Classification as a Case Study. *IEEE J. Sel. Top. Appl. Earth Obs. Remote Sens.* **2016**, *9*, 5806–5818, doi:10.1109/JSTARS.2016.2539282.
15. Zhou, Y.; Lin, J.; Wang, Z. Energy efficient SVM classifier using approximate computing. In Proceedings of the 2017 IEEE 12th International Conference on ASIC (ASICON), Guiyang, China, 25–28 October 2017; pp. 1045–1048, doi:10.1109/ASICON.2017.8252658.
16. Ibrahim, A.; Osta, M.; Alameh, M.; Saleh, M.; Chible, H.; Valle, M. Approximate Computing Methods for Embedded Machine Learning. In Proceedings of the 2018 25th IEEE International Conference on Electronics, Circuits and Systems (ICECS), Bordeaux, France, 9–12 December 2018; pp. 845–848, doi:10.1109/ICECS.2018.8617877.
17. Younes, H.; Ibrahim, A.; Rizk, M.; Valle, M. Algorithmic Level Approximate Computing for Machine Learning Classifiers. In Proceedings of the 2019 26th IEEE International Conference on Electronics, Circuits and Systems (ICECS), Genoa, Italy, 27–29 November 2019; pp. 113–114, doi:10.1109/ICECS46596.2019.8964974.
18. Younes, H.; Ibrahim, A.; Rizk, M.; Valle, M. Data Oriented Approximate K-Nearest Neighbor Classifier for Touch Modality Recognition. In Proceedings of the 2019 15th Conference on Ph.D Research in Microelectronics and Electronics (PRIME), Lausanne, Switzerland, 15–18 July 2019; pp. 241–244, doi:10.1109/PRIME.2019.8787753.
19. Nogues, E.; Menard, D.; Pelcat, M. Algorithmic-Level Approximate Computing Applied to Energy Efficient Hevc Decoding. *IEEE Trans. Emerg. Top. Comput.* **2016**, *7*, 5–17, doi:10.1109/TETC.2016.2593644.
20. Zhou, B.; Brent, R.; Kahn, M. Efficient one-sided Jacobi algorithms for singular value decomposition and the symmetric eigenproblem. In Proceedings of the 1st International Conference on Algorithms and Architectures for Parallel Processing, Brisbane, Australia, 19–21 April 1995; Volume 1, pp. 256–262, doi:10.1109/ICAPP.1995.472193.
21. Osta, M.; Ibrahim, A.; Magno, M.; Eggimann, M.; Pullini, A.; Gastaldo, P.; Valle, M. An Energy Efficient System for Touch Modality Classification in Electronic Skin Applications. In Proceedings of the 2019 IEEE International Symposium on Circuits and Systems (ISCAS), Sapporo, Japan, 26–29 May 2019; pp. 1–4, doi:10.1109/ISCAS.2019.8702113.
22. Chang, C.C.; Lin, C.J. LIBSVM: A library for support vector machines. *ACM Trans. Intell. Syst. Technol.* **2011**, *2*, 1–27, doi:10.1145/1961189.1961199.
23. Xilinx, X. *Vivado Design Suite User Guide, High-Level Synthesis*; 2014. Available online: https://www.xilinx.com/support/documentation/sw_manuals/xilinx2018_3/ug902-vivado-high-level-synthesis.pdf (accessed on 5 November 2020).
24. Fares, H.; Seminara, L.; Ibrahim, A.; Franceschi, M.; Pinna, L.; Valle, M.; Dosen, S.; Farina, D. Distributed Sensing and Stimulation Systems for Sense of Touch Restoration in Prosthetics. In Proceedings of the 2017 New Generation of CAS (NGCAS), Genova, Italy, 6–9 September 2017; pp. 177–180, doi:10.1109/NGCAS.2017.54.

7.4.2 Efficient Selection-based k-Nearest Neighbor Architecture on Modern SoCs

Modern System-on-Chips (SoCs) are designed with heterogeneous architectures to support a variety of computationally intensive tasks in many application domains such as IoT systems, industrial automation, robotics, etc. These systems could consist of multi-core processors or multi-processor system on chip, which could be implemented on GPUs, application specific integrated circuits, or field programmable gate arrays. The latter is used for accelerating complex operations and performing tasks concurrently compared to traditional processors.

kNN is a supervised classification algorithm used in a variety of applications such as pattern recognition, computer vision and machine learning [177]. However, kNN imposes significant computational workload since the complexity increases linearly with the size of the dataset and the number of classes [178]. Such workload demands significant memory requirements with high latency and power consumption [179]. Accordingly, the implementation of kNN on embedded systems with limited available energy and resources introduces a design challenge, which makes kNN hardware acceleration a necessity.

kNN algorithm involves independent operations e.g. the distance computation between a point A and point B is independent of that between points A and C. Thus, kNN doesn't require the sorting of the entire distance vector to find the K-Nearest Neighbors. Such characteristics could be exploited to reduce the computational complexity of the algorithm using a pipelined architecture and tweaking the sorting process.

In this research work, we propose the design and implementation of a kNN architecture that is characterized by a novel selection-based sorter (Selector). The proposed selector overcomes similar state of the art solutions by reducing the occupied hardware area by up to 48% while providing a speedup up to $4.5\times$. The proposed kNN architecture is implemented using both exact and approximate computations. The approximate architecture utilizes the use of algorithmic level ACTs. When validated on a touch modality classification problem, both the proposed exact and approximate kNNs offer a real-time classification while consuming 6 μJ and 1.9 μJ respectively when implemented on Xilinx Zyqnberry platform. Compared to similar kNN architectures, the proposed kNN achieves a speedup between $1.4\times$ and $875\times$ with 41% to 94% less energy consumption and 12% to 94% average hardware area reduction. Moreover, applying algorithmic level ACTs on the proposed architecture improves its performance by achieving a 56.4% average area reduction, a speedup by $2.3\times$, and an energy reduction of about 69%. An accuracy degradation of 2.6% has been reported using the proposed approximate architecture. This research work has been published in the *IEEE Open Journal of Circuits and Systems (OJCS)* [180].

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/OJCAS.2021.3108835, IEEE Open Journal of Circuits and Systems

An Efficient Selection-Based kNN Architecture for Smart Embedded Hardware Accelerators

Hamoud Younes, Ali Ibrahim, Mostafa Rizk, Maurzio Valle, *Member, IEEE*

K-Nearest Neighbor (kNN) is an efficient algorithm used in many applications e.g. text categorization, data mining, and predictive analysis. Despite having a high computational complexity, kNN is a candidate for hardware acceleration since it is a parallelizable algorithm. This paper presents an efficient novel architecture and implementation for a kNN hardware accelerator targeting modern System-on-Chips (SoCs). The architecture adopts a selection-based sorter dedicated for kNN that outperforms traditional sorters in terms of hardware resources, time latency, and energy efficiency. The kNN architecture has been designed using High-Level Synthesis (HLS) and implemented on the Xilinx Zynqberry platform. Compared to similar state-of-the-art implementations, the proposed kNN provides speedups between 1.4× and 875× with 41% to 94% reductions in energy consumption. To further enhance the proposed architecture, algorithmic-level Approximate Computing Techniques (ACTs) have been applied. The proposed approximate kNN implementation accelerates the classification process by 2.3× with an average reduced area size of 56% for a real-time tactile data processing case study. The approximate kNN consumes 69% less energy with an accuracy loss of less than 3% when compared to the proposed Exact kNN.

***Index Terms*—Embedded Implementation, Hardware Accelerators, K-Nearest Neighbor, Approximate Computing, Tactile Sensing, Real-time Processing, Energy Efficiency, High Level Synthesis, FPGA**

I. INTRODUCTION

MODERN System-on-Chips (SoCs) are designed with heterogeneous architectures to support a variety of computationally intensive tasks in many application domains such as IoT systems, industrial automation, robotics, etc. These systems could consist of multi-core processors, or specialized hardware such as Graphics Processing Units (GPUs) and Field Programmable Gate Arrays (FPGAs). The latter is used for accelerating complex operations and performing tasks concurrently compared to traditional processors.

K-Nearest Neighbor is a supervised classification algorithm used in a variety of applications such as pattern recognition, computer vision and machine learning [1]. However, kNN imposes significant computational workload since it has a linear scalability with the size of the dataset and the number of classes [2]. For embedded implementations, such workload demands significant memory requirements with high latency and power consumption [3], which makes kNN hardware acceleration a necessity. kNN algorithm involves independent operations e.g. the distance computation between a point A and point B is independent of that between points A and C, and kNN doesn't require the sorting of the entire distance vector to find the K-Nearest Neighbors. Such characteristics could be exploited to reduce the computational complexity of the algorithm using a pipelined architecture and tweaking the sorting process. Another solution for complexity reduction could be the use of Approximate Computing Techniques (ACTs). Approximate computing is the idea of reducing the accuracy to an acceptable limit to save energy, memory, and execution time without affecting the applications' overall quality [4]. Compared to Exact Computation Techniques (ECTs) i.e. performing arithmetic operations following the IEEE-754 and IEEE-854 standards for floating-point operations [5], the challenge of using ACTs is to maintain the approximation error acceptable with respect to the target quality of service (QoS) [6]. ACTs can be divided into two categories as shown in Figure 1: Processing-oriented and Data-oriented [6]. Processing-oriented techniques consist of "Computation Skipping" and "Computation Approximation". The former technique involves removing some processing tasks (e.g. image compression, encoding, etc.). The latter one replaces computationally intensive blocks with approximate ones that implements the same mathematical operations. Data-oriented techniques target dataset modifications as "Dataset Reduction" and "Data Format Modification". Dataset Reduction aims at decreasing the number of processed samples through Downsampling and Downscaling. Downsampling includes adjusting the sampling frequency of a processing block or truncating the size of an acquired sample. Downscaling reduces the dimension of the

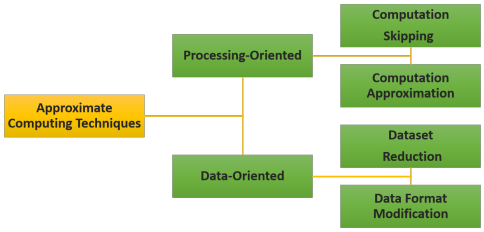


Fig. 1. Approximate Computing Techniques (ACTs)

Hamoud Younes, Ali Ibrahim, and Maurizio Valle are with the Department of Electrical, Electronic and Telecommunications Engineering and Naval Architecture, University of Genova, 16145 Genova, Italy E-mails: hamoud.younes, ali.ibrahim, maurizio.valle@unige.it

Hamoud Younes, Ali Ibrahim, and Mostafa Rizk are with the Department of Computer and Communication Engineering, Lebanese International University, Lebanon E-mails: hamoud.younes, ali.ibrahim, mostafa.rizk@liu.edu.lb

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/OJCAS.2021.3108835, IEEE Open Journal of Circuits and Systems

data by adjusting the matrix or vector size for vector-based applications. As for Data Format Modification, it changes the data from floating-point to fixed-point representation to simplify the involved arithmetic operations."
Approximate computing can lead to significant improvements from algorithmic to circuit level. Authors in [7] have presented an assessment of the different approximate computing techniques applied at the circuit, architecture, and algorithmic levels. Applying ACTs on supervised learning algorithms at the algorithmic level has been the focus of Younes *et. al* in [8] with a case study on the kNN algorithm. Authors reported that approximate kNN offers a computation boost in terms of $2\times$ speedup compared to Exact kNN.

To design a hardware accelerator for the kNN algorithm, an efficient architecture is proposed in this paper. The architecture introduces novel sorting process dedicated for kNN accompanied with several design optimizations. Moreover, ACTs are incorporated to further enhance the performance of the embedded implementation. For the rest of the paper, the term *"performance"* is used to report the characteristics of a kNN hardware implementation in terms of area, time latency, power consumption, and energy per classification. While, the term *"quality"* reflects the highest classification accuracy that a kNN architecture could achieve.

The main contributions of this paper could be summarized as follows:

- It proposes the design and implementation of a selection-based sorter (Selector) for the K-Nearest Neighbor (kNN) algorithm. The proposed selector overcomes similar state of the art solutions by reducing the occupied hardware area by up to 48% while providing a speedup up to $4.5\times$.
- It presents a novel kNN architecture that outperforms similar state of the art solutions in terms of occupied hardware area, time latency, and energy consumption. When validated on a touch modality classification, the proposed kNN achieves a speedup between $1.4\times$ and $875\times$ with 41% to 94% less energy consumption and 12% to 94% average hardware area reduction.
- It applies algorithmic level ACTs on the proposed architecture to improve its *performance*: the results show a 56.4% average area reduction, a speedup by $2.3\times$, and an energy reduction of about 69%. For a touch modality classification problem, an accuracy degradation of 2.6% was reported using the proposed approximate architecture.
- It demonstrates the feasibility of the implemented system for real-time touch modality classification when validated on Xilinx Zyqnberry platform. The proposed exact and approximate kNNs consume 6 μJ and 1.9 μJ respectively.

The rest of the paper is organized as follows: Section II reports on some of the efficient implementations of kNN presented in the literature. Section III explains the design and the high-level synthesis of the proposed selection-based kNN architecture. Also, it describes the selector integration into exact and approximate kNN classifiers. Section IV presents an overview of the tactile data processing case study and the experimental setup used to asses the *quality* of the proposed architecture. This Section also highlights the implementation tools and methodology with emphasis on the adopted design optimization techniques. Section V provides a *performance* analysis for the FPGA implementation of both the exact and approximate kNN classifiers. Also, a comparison with similar works from the literature is conducted. Section VI concludes the paper highlighting some future works.

II. State-of-the-art

Several architectures have been proposed in the literature for implementing the kNN algorithm on hardware platforms. One of the first architectures and implementations is presented in [9]. The architecture is based on a neural network with SIMD-style architecture, which imposed excessive response time while performing complex operations. Authors in [10] designed flexible IP cores based on linear array architecture for the FPGA implementation of the kNN algorithm. The cores achieved a very high throughput when validated on a medium size FPGA device, very large size classification problems, and with thousands of reference data vectors. A novel dynamic partial reconfiguration (DPR) architecture of the kNN algorithm is presented in [11]. The architecture is characterized by an efficient reconfiguration time for different values of K. Speedups between $68\times$ and $76\times$ were recorded when compared to General Purpose Processor (GPP) implementation. Pu et. al designed a kNN-specific bubble sort algorithm to take advantage of the FPGA parallel pipeline structure using OpenCL [12]. The overall implementation showed an enhanced *performance* compared to conventional GPU implementations. Authors in [13] adopted a Bitonic sorting algorithm to implement the kNN algorithm on both FPGA and GPU. Using OpenCL coding style and some HLS directives, the results showed that an FPGA implementation could be comparable to a GPU in terms of execution time. Another HLS based implementation has been reported in [2]. With a reduced number of comparators in the sorting process and by utilizing the memory-mapped AXI4-Master Interface of the Xilinx ZC706 FPGA board, a $35.1\times$ speedup over a GPP based implementation is noticed. In [14], several architectures are presented, where each one adopt a single or multiple HLS optimization directives. Speedups between $3.8\times$ and $58\times$ could be achieved while using the quick sort algorithm and the BCW_9 dataset. Both hardware and software designs using Xilinx MicroBlaze platform were used to verify the bubble sort based architecture of the kNN algorithm in [15]. The hardware design achieved $127\times$ speedup compared to its software counter counter-part. To reduce the impact of the memory-access constraint in kNN classification problems, an HLS based kernel is proposed in [16]. The kernel employs two data access reduction methods: low precision data representation and principal component analysis based filtering (PCAF). The kernel performed to an equivalent 56-thread CPU server while greatly reducing external memory-accesses.

A common characteristic of the most existing efficient kNN architectures is the use of the conventional sorting algorithms without optimizations for the kNN algorithm. The efficiency of these algorithms is affected by the: i) need to sort all the

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/OJCAS.2021.3108835, IEEE Open Journal of Circuits and Systems

vector's elements, ii) large number of required comparators, and iii) increased complexity/latency for large vector size. In this work, we propose a kNN architecture that avoids such sorting algorithms and adopts a selection-based one with a reconfigurable division ratio for complexity and latency trade-offs (see Section III.C). Moreover, existing kNN implementations focus on acceleration gains compared to CPU-based implementations. This work reports a detailed comparison with FPGA-based kNN implementations that sheds the light on different implementation strategies and their effect on the kNN *performance*.

III. K-NN Proposed Hardware Architecture

A. k-Nearest Neighbor Algorithm Overview

The kNN algorithm classifies an input sample according to the class of the majority of K-nearest samples. For an input sample, the kNN classifier 1) calculates the distance between the input sample and all the samples in the training set $T_i \in T$, 2) sorts the distances in ascending order, and 3) selects an output class based on the minimum distance towards K neighbors. The three main considerations for kNN are:

- The number of nearest neighbors K. A 1-NN classifier is naïve, while a large value of K might result in over-fitting. Thus, the value of K is determined using a trade-off between accuracy and complexity.
- The distance metric could be Chebyshev, Manhattan, Cosine, Euclidean, etc [17].
- The sorting algorithm could be bubble, select, quick, etc [18].

B. Selection-based kNN Architecture

The block diagram of the proposed hardware architecture is shown in Figure 2. The "kNN Classifier" block has been designed in HLS (coded in C++), whereas the other blocks are existing IP blocks embedded in Vivado. The SDRAM memory is used to store the training set, which is more suitable than the Block RAM (BRAM) of the FPGA for platforms with limited number of BRAMs or applications with large datasets. The AXI Interconnect IP handles the read and write operations from and to the memory. It adopts an AXI smart connect IP to use one AXI port for 1) writing to the data acquisition block and 2) reading the classification result from the class determination block. The Zynq processing system IP is the main block of the design which uses a processor system reset IP to drive all blocks with a common clock and reset signals. The kNN HLS IP starts operating once the Data acquisition block receives the training samples. To reduce the access overhead imposed by DRAM, we fetch the samples in bursts to reduce the number of memory accesses. Such technique has shown its efficiency in [13] and [19]. First, the distance between a testing sample and all training samples is performed. Then, the K minimum distance values are determined using the proposed selector (see subsection C), and finally a class is assigned for the testing sample using the class majority of the samples corresponding to the K minimum distance values.

C. Nearest Neighbors Selector

The proposed architecture aims to replace the conventional sorter block with a "Selector" which finds the K-minimum distances without sorting the entire vector [20] as depicted in Algorithm 1. While coding the selector in HLS, the minimum K-distance values are saved in the same vector to be sorted, thus decreasing the memory footprint.

The selector operations can be detailed in three steps:

- Step 1: The distance vector V of size S is divided into two vectors $V1$ and $V2$. The suitable division ratio $(a\%{:}b\%)$ is determined via a software simulation. Start by decreasing the size of V until the classification accuracy drops to obtain the value of a. Hence, $b = S - a$.
- Step 2: K-registers are initialized with a maximum value (e.g. 1000). Each distance value $V1[i]$ is compared to the content of register 1. If it is smaller, $V1[i]$ occupies the register, and the old content in register 1 is shifted to register 2. Then, the content in register 2 is shifted to occupy register 3. Consequently, the content in register i is shifted to occupy register $i+1$. Else, $V1[i]$ is compared

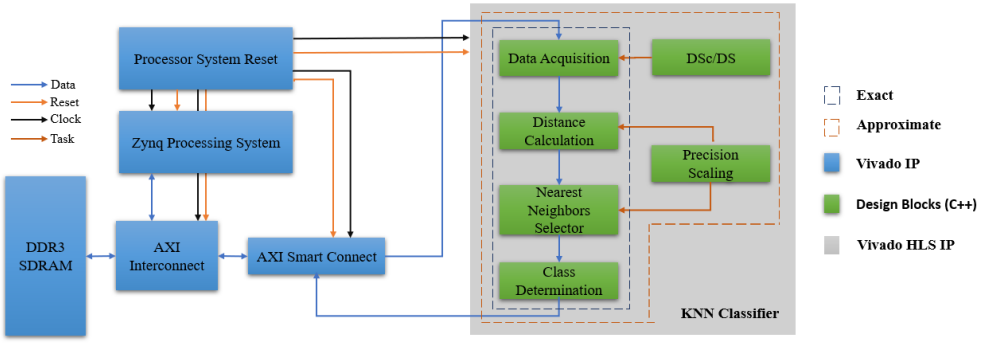


Fig. 2. The Proposed Hardware Architecture

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/OJCAS.2021.3108835, IEEE Open Journal of Circuits and Systems

Algorithm 1: Nearest Neighbors Selector

Input: Vector V with size S, Division ratio a:b, Number of neighbors K
Output: V with the K-minimum elements at the first K indices

$S1 \leftarrow a \times S/100$
for $i \leftarrow$*K to* $S1$ **do**
  if $V[i] <= V[0]$ **then**
    $V[K-1] \leftarrow V[K-2]$
    ⋮
    $V[0] \leftarrow V[i]$
  else if $V[i] <= V[1]$ **then**
    $V[K-1] \leftarrow V[K-2]$
    ⋮
    $V[1] \leftarrow V[i]$
  ⋮
  else if $V[i] <= V[K-1]$ **then**
    $V[K-1] \leftarrow V[i]$
for $j \leftarrow$*S1 to* S **do**
  if $V[j] >= V[K-1]$ **then**
    break
  else
    if $V[j] < V[K-2]$ **then**
      ⋮
      if $V[j] < V[0]$ **then**
        $V[0] \leftarrow V[j]$
      else
        $V[1] \leftarrow V[j]$
    else
      break

to the next register, and so on. At the end of step 2, the K minimum distance values are saved in the K registers in the order $min1 < min2 < ..... < minK$.

- Step 3: Each distance value $V2[i]$ is compared to the highest minimum i.e. $minK$ as shown in Figure. 3 (K=3). If it is larger, the minimums obtained from $V1[i]$ are not updated and a new value of $V2$ is fetched. Else, $V2[i]$ is compared to the other minimums to reach a register to occupy. Once $V2[i]$ occupies a register, the old value of that register is shifted to occupy the register of the next minimum.

The advantage of this architecture compared to the one presented in [14], is that if $V2[i]$ is greater than $minK$, step 3 will not be executed. Thus, the K minimum distances are the output of step 2. This will result in a reduced selection time for hardware implementations. Moreover, the architecture in [14] selects the K minimum distances in a single step, which imposes hardware complexity and increased time latency for large datasets. While in the proposed architecture, the selection is performed in two smaller steps with a high probability that the third step will not be executed ($V2[i] > minK$).
Although the proposed selector finds the K-nearest neighbors without sorting the entire vector, a comparison with two sorters reported in the literature, i.e. QuickSort [12] and Bitonic Sorter

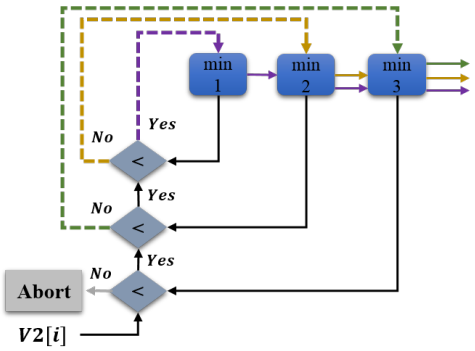


Fig. 3. Sorting Process Step 3 (K=3): Dashed Line (new value), Solid Line (old value), Colored Lines (concurrent operations)

[21], has been carried out. In general:

- Bitonic sorting is a recursive algorithm that sorts a Bitonic sequence in a parallel operating fashion. A Bitonic sequence is a sequence of M elements in which L elements out of M are sorted in ascending form, and the other $M-L$ elements are sorted in descending order [22]. If the sequence is not Bitonic, an additional task is imposed before the ability to sort the vector. The proposed selector can operate on any vector form.
- QuickSort selects one of the elements in the sequence to be the pivot and divides the sequence into two sequences one with all elements less than the pivot while the other contains all the elements greater than the pivot. The process is recursively (add more burden on the hardware) applied to each of the sub sequences. QuickSort has a worst-case complexity of $O(n^2)$ when the given sequence is sorted; this resembles the best-case scenario for the proposed selector as it will select the K minimums faster (the minimums occupy the first K registers). Then all the comparisons fail thus no shift operations are performed. Consequently, step 3 is not executed at all.
- The number of comparators required by the selector depends on the number of neighbors K, while it depends on the size of the vector N in the case of Bitonic and QuickSort. In machine learning applications, usually, it is valid that K (the number of Nearest Neighbors in kNN) $\ll$ N (size of training vector). Given that the selector doesn't sort the complete vector, the number of comparisons is decreased.

Both the sorters presented in [12],[21], and the selector

TABLE I
HLS SYNTHESIS RESULTS OF DIFFERENT SORTERS

Sorter	FF	LUT	Clock Cycles
Proposed Selector	84	176	15 (division ratio 6:4)
[12]	106	226	17
[21]	123	514	96

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/OJCAS.2021.3108835, IEEE Open Journal of Circuits and Systems

have been coded in C++ using Vivado HLS. The distance vector V has been used as a testing vector for the three implementations. In this paper, the best K value and division ratio were determined to be 3 and 6:4 (for the case study presented in section IV.B) i.e. there is a need to sort only 60% of the vector and step 3 can be aborted without affecting the selection process accuracy. The obtained synthesis results for finding the three minimum numbers in V are presented in Table I. Results show that the selector occupies less hardware area than the implementations of both sorters. Specifically, an average reduction of 21.4% and 48.7% is reported compared to the sorters in [12] and [21] respectively. Concerning the time latency of the sorting process, the selector is faster than the sorter in [21] by $4.5\times$. Compared to the sorter in [12] that adopts one of the fastest sorting algorithm (QuickSort), the *performance* depends on the selection of the division ratio in step 1 and the number of neighbors K. In fact, if all the minimum numbers are located in $V2$, or if the required value of K is very large; the selector is now sorting all the elements of V resulting in a slower sorting process. Hence a speedup of $\pm 1.2\times$ (for 6:4 and 5:5 ratios respectively) has been observed for the given task.

D. Approximate kNN Blocks

In [23], we have presented a complete assessment of using algorithmic level ACTs on a kNN classifier. The assessment included the degradation in accuracy to the gain in memory and execution time on Intel i7 CPU. In [8], the studied techniques have been formulated into a general approach that has been tested for the FPGA implementation of two machine learning classifiers. The reported approach has been adopted in this paper where a trade-off between the classifiers *performance* and *quality* has been considered. The trade-off resulted in our selection for the ACTs presented in the proposed approximate kNN architecture. All the adopted techniques belong to the data-oriented approximate computing category [8]. The adopted ACTs are Dataset Reduction (Downsampling (DS) and Downscaling (DSc)), and Data Format Modification (DFM). DS means varying the signal sampling frequency during signal acquisition. Since tactile data used in this work are from an already available dataset, the sampling frequency can't be changed. As a consequence, DS is applied offline on the dataset by reducing the number of samples. DSc is applied by adjusting the sample size as shown in subsection IV.A. DFM reduces the sample resolution. This can be achieved through the use of fixed-point or mixed precision instead of floating-point data representation. In this paper, fixed-point representation is adopted, and the precision is determined as a trade-off between resolution (32, 24, 16, and 8-bit) and classification accuracy." The approximate kNN classifier starts operating once the Data Acquisition block receives the training samples (after DS/DSc has been applied offline) from the memory. Then, the same steps performed by the kNN HLS IP are executed.

IV. Selection-based kNN Implementation and Case Study

A. Case Study: Tactile Data Processing for Electronic Skin Systems

Electronic skin system is an artificial system developed to mimic human skin behaviour or to implement intelligent tasks in applications such as robotics, prosthetic, etc. Intelligence involves the use of learning algorithms for tactile data processing in tasks such as surface texture, object compliance, touch modality, etc. [24], [25]. Touch modality classification allows the integration of gesture-based actions to be performed by robots or humans with prosthetic hands. For a medical purpose of caring for patients with mild mental impairment; a humanoid equipped with artificial skin has been trained to discriminate between nine touch modalities (scratch, tickle, rub, etc.) [26]. Recognition rates up to 96.7% has been achieved using four machine learning algorithms including kNN, SVM, DT, and Logitboost. Authors in [27] and [23] have adopted a touch modality classification problem that involves three patterns: brushing a paint brush, rolling a washer, and sliding a finger on 4×4 tactile sensory array. SVM and Extreme Learning Machine (ELM), kNN, and Deep CNN based on transfer learning have been chosen as learning algorithms. A classification accuracy of 90%, 89.8%, and 76.9% has been recorded respectively.

In this paper, the kNN algorithm is adopted for the design of an embedded tactile data processing architecture due to the: 1) high level of parallelization of the kNN algorithm, which makes it adequate for hardware acceleration, 2) high classification accuracy with a reduced computational complexity compared to state-of-the-art algorithms operating on the same task [28], [29], and 3) ability of complexity reduction without affecting the application *quality* using approximate computing techniques [23].

The dataset collected in [30] has been selected for the validation of the proposed kNN architecture. The experimental setup is shown in Figure 4 and can be described as follows:

- Dataset: The dataset contains records for the two touch modalities performed by 70 participants. Each modality was recorded from a 4×4 tactile sensor for 10 seconds at 3 kHz sampling frequency. Thus, each raw data sample can be modeled in the form of a tensor of size $4\times4\times30{,}000$. The touch modalities were performed on both the horizontal and vertical directions for two trials, resulting in a dataset of 840 samples.
- Simulation Software: An open-source machine learning simulation tool called "Weka" has been used [31]. Weka involves a collection of learning algorithms that can be applied to a pre-defined dataset or invoked from a Java code. The tool has options for classification, clustering, regression, etc.
- Classification Task: the binary problem "Sliding a finger" vs "Washer Rolling". For Weka simulation an Attribute-Relation File Format (ARFF) file is required. Thus, a header describing the features and the possible output class of each sample is added to the original tactile dataset.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/OJCAS.2021.3108835, IEEE Open Journal of Circuits and Systems

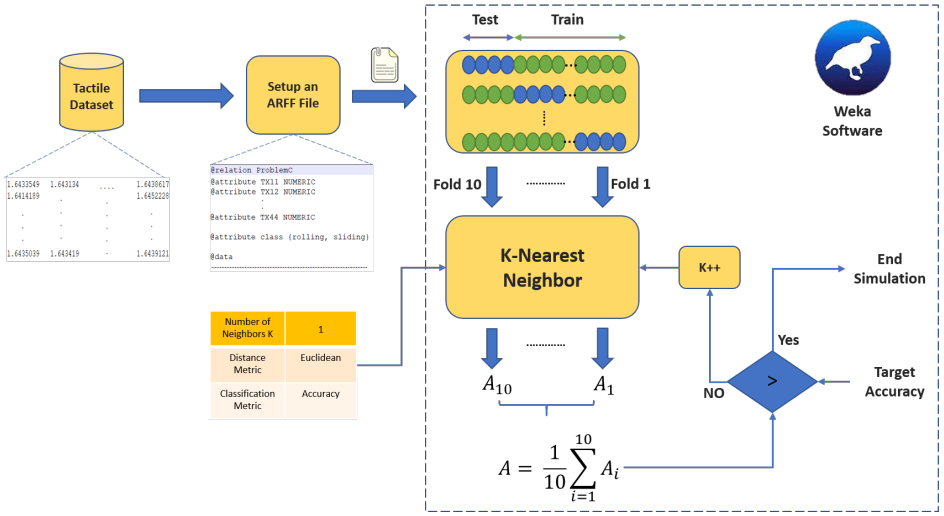


Fig. 4. Experimental Setup

- kNN Characteristics: A 10-fold cross-validation simulation using the Weka tool has been carried to determine the best value of K. The adopted distance between two tactile samples $T1$ and $T2$ is the squared euclidean distance written as:

$$d(T_1, T_2) = \sum_{0}^{16} (TX_{ij} - TX_{mn})^2 \quad (1)$$

where TX_{ij} and TX_{mn} are the taxels inside a 4×4 tactile sample.

- Classification Metric: The kNN algorithm has been assessed by calculating the classification accuracy i.e. the ratio of correctly classified samples to the total number of available samples.

The ARFF file was loaded into Weka and classification using kNN is performed. A kNN classifier with 3-nearest neighbors resulted in the highest classification accuracy of 89.8%. This result was achieved based on a model selection approach. Based on the best kNN model obtained, a novel efficient kNN architecture is proposed in this paper. Furthermore, approximate computing techniques have been applied to enhance the overall *performance* of the proposed architecture without affecting the *quality* of the application.

The best obtained kNN model with K=3 is referred to as Exact kNN. As for Approximate kNN, it employs the following techniques to the Exact architecture: 1) Downsampling which applies an approximate window on the touch modality, where only the data that corresponds to the interval [a,b] seconds is considered. First, the interval [a, b] is selected such that $0.1 < a \ll 1$ and $9.5 < b < 10$. Then, the values of a and b are varied, while calculating the classification accuracy. The interval [3.5, 7] provided the highest accuracy among others. Thus, each modality tensor can be written as $\phi = 4 \times 4 \times 10,500$ (touch readings that belong to the interval [3.5, 7] seconds), and 2) downscaling which reduces the tensor size to $\phi = 4 \times 4 \times 1$, where the last dimension is the mean of the 10,500 readings according to the equation:

$$mean = \frac{\sum TX_{ij}}{10500} \quad (2)$$

where TX_{ij} is the individual taxel inside the 4×4 tactile sample. Figure 5 shows the initial and obtained touch modalities after applying DS and DSc.

It is worth mentioning that the tensor representation of data has been adopted by [30] since it preserves the initial structure of the data, which is still valid after applying approximate computing techniques. This is evident in Figure 5 (b), (c) where the two touch modalities can still be differentiated. For both the Distance Calculation and Nearest Neighbor Selection blocks, the operations are implemented in 24-bit fixed-point representation with a $< 6, 18 >$ precision. The adopted precision is based on a trade-off between complexity and classification accuracy.

B. Hardware and Software Design Tools

The Zynqberry TE0726-03M [32] was adopted for implementation. Zynqberry is a small-sized platform in the form of a Raspberry Pi compatible System-on-Chip (SoC) module integrating a Xilinx Zynq-7010 with a 512 MB SDRAM memory. The Zynq SoC has a hybrid structure of combining a dual-core ARM Cortex-A9 processor as a Processing System (PS) and an FPGA as Programmable Logic (PL) in a single SoC. Moreover, for comparison purposes, the architecture has

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/OJCAS.2021.3108835, IEEE Open Journal of Circuits and Systems

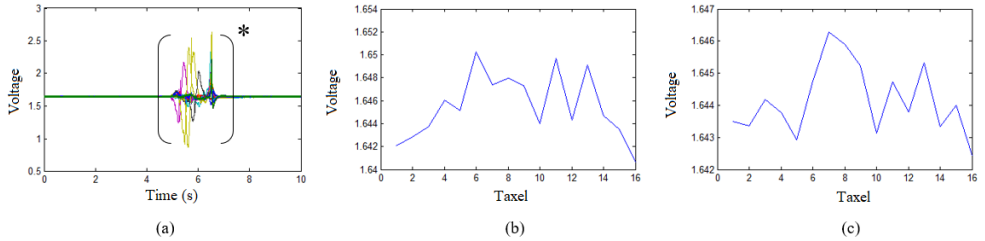


Fig. 5. Touch Modalities: (a) Rolling with DS, (b) Rolling after DSc, (c) Sliding after DSc, * Window

been also implemented on the Virtex-7 FPGA and the NVIDIA GTX 1650 GPU.

As for the software tools, Vivado HLS 2018.3 and Vivado 2018.3 were used. Vivado HLS allows the design of an embedded system on FPGA using a high-level programming language such as C and C++ compared to traditional hardware description languages (HDL). The use of HLS decreases the FPGA development time and effort. Also, it offers a set of optimization directives that can be used to enhance the design *performance*. Once the design is completed, it can be exported as a Register Transfer Level (RTL) Intellectual Property (IP) block. The latter is imported into Vivado and can be connected to the processing system (PS) via built-in IPs; thus, it is possible to obtain the hardware resources, time latency, and power consumption of the whole design.

C. Implementation Methodology

Once the whole design code is completed in HLS and design optimizations are applied, a co-simulation is performed. This simulation runs both the C++ and the RTL simulations together to verify a matching output. Then, the design is exported as an IP block. The latter is imported into Vivado 2018 and connected to the Zynq processor and other IP blocks as seen in Figure 2. First, a behavioral simulation is performed to verify the functionality of the design. Then, synthesis and place and route occur to finalize implementation. At this point, the generated report contains the occupied area percentage (BRAM, DSPs, etc.) and the number of clock cycles passed to generate an output. Concerning power consumption estimation, Vivado offers two methods: Vector-based and Vector less. This estimation can be performed at any stage between post-synthesis to post routing. For a credible estimation, a post-implementation functional and timing simulation is used to generate a Switching Activity Interchange File (SAIF) to be used for a vector-based estimation post-routing.

D. Design Optimization

Targeting the real-time functionality on a small-sized platform such as Zynqberry, the proposed architecture has been optimized to ensure an acceptable balance between time latency and hardware requirements. This has been achieved with several design optimizations as shown in Fig. 6. Such optimizations are facilitated with the use of Vivado HLS directives [33] as depicted in Algorithm 2. These optimizations are summarized as follows:

Algorithm 2: kNN Design Optimization

```
#pragma HLS ARRAYMAP
variable=Distance Instance=AllPatterns horizontal
variable=Modality Instance=AllPatterns horizontal
/* M: nb of features                                  */
function UDC(T_1, T_2):
for i ←0 to M do
    #pragma HLS UNROLL factor=4
    Execute (1)
/* Tactiles: trainingset, q: testing
   point                                              */
/* N: nb of training points                           */
for i ←0 to N do
    #pragma HLS INLINE
    #pragma HLS UNROLL factor=6
    Distance[i]= UDC(p, Tactiles[i])
    Modality[i]=Tactiles[i][16]
/* K: nb of Neighbors                                 */
NearestNeighbors=Selector(K, Modality)
for i ←0 to K do
    #pragma HLS UNROLL
    if NearestNeighbors[i] == 1 then
        modality1count++
    else
        modalitycount2++
```

1) BRAM Resources Reduction: The BRAM size is 18K in the FPGA, if many arrays have a size less than 18K, it is better to combine them into a single array. Since kNN is a supervised algorithm, the class of each query must be known. Thus, we can benefit from this directive to combine the "Distance" and "Modality" arrays into a single array as shown in Figure 6(a). Thus, when the selector block finds the three minimum neighbors, the class of each selected neighbor is available at the same instant. This process is referred to as "Array Map" where the horizontal option means that the two arrays are combined into a single array with more elements (see Algorithm 2).

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/OJCAS.2021.3108835, IEEE Open Journal of Circuits and Systems

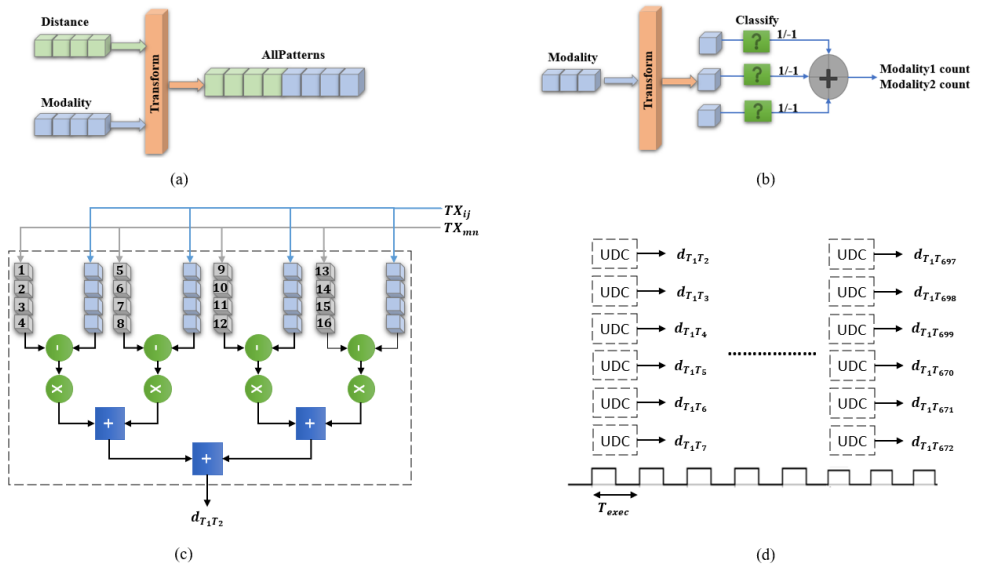


Fig. 6. Design Optimization: (a) BRAM Resources Reductions, (b) Unrolled Class Determination, (c) One Unrolled Distance Calculation (UDC) Block, (d) Complete Unrolled Distance Calculation

2) Parallelization: To exploit the capabilities of the FPGA, the operations of the distance calculation and the class determination blocks are executed in parallel with a small unroll factor. In HLS terms this is known as "Unrolling", where unrolling a loop creates multiple copies of its body in the RTL design, which allows some or all of its iterations to occur in parallel. This optimization has been applied to (1) denoted as Unrolled Distance Calculation (UDC) and the Class Determination blocks leading to accelerating the calculation and output class decision. However, executing all the operations in parallel leads to a high power consumption and increased resource requirements. To avoid the negative impact of unrolling on the hardware cost and power consumption, the loops are partially unrolled (unroll factors of 4 and 6) as it is shown in Algorithm 2, and the design is implemented at an operating frequency of 100 MHz which is lower when compared to similar work [21]. Each touch modality sample has 16 features, thus an UNROLL factor equals to 4 is used. This means that the distance between each 4 features is calculated in a single time interval as shown in Figure 6(c). Similarly, Figure 6(d) shows how the UDC block is used to calculate the distance between the testing sample and all training samples. An UNROLL factor equals to 6 is used, thus 112 timing intervals are required to finish all the distance calculations for a training set size of 80%. The distance from a testing sample to $(80 * 840/100) = 672$ training samples is calculated in batches of 6 calculations per timing interval i.e $672/6 = 112$ intervals. As for Class Determination, since $K = 3$ and we have a binary classification, the process could be fully unrolled as shown in 6(b).
3) Function Inline: The inlined function is treated as a part of the calling function that is calling it rather than a separate entity. This optimization is applied for the distance calculation function. Thus, whenever the classification function is called, the distance calculation is executed within it and it no longer appears as a separate level of hierarchy in the RTL design. Thus improving the overall latency of the classification task.

V. Implementation Results and Assessment

The *performance* and *quality* of the proposed exact and approximate implementations are assessed on the touch modality classification problem mentioned in section IV. The assessment involves three case studies: (1) Proposed Exact kNN versus approximate kNN, (2) Exact kNN on FPGA versus GPU, and (3) Exact kNN versus similar works. For cases (1) and (3), the time latency T is calculated according to the equation:

$$T = N \times 1/f_{max} \quad (3)$$

where N is the number of clock cycles obtained in post-implementation reports and f_{max} is the maximum operating

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/OJCAS.2021.3108835, IEEE Open Journal of Circuits and Systems

frequency the design can achieve. The Joule per classification energy E is calculated as:

$$E = T \times P \tag{4}$$

where T is the time latency and P is the dynamic power consumed by the programmable logic (PL) of the Zynqberry reported by Vivado i.e the power consumed by the simulated kNN architecture to compute a classification of an input sample (excluding the static power of the processing system (PS) as it is device dependent).

For case (2), Exact kNN architecture has been coded using Python language running inside the CUDA computing platform. The GPU power estimation was obtained using NVIDIA System Managment Interface (NVSMI). The latter is a command utility that can be issued on any Python development environment with the CUDA libraries imported [34].

Case 1: Exact versus Approximate kNN

The implementation results on Zynqberry of the proposed Exact and Approximate kNN are shown in Table II. Exact kNN occupies 12% of the hardware resources, consumes 0.236 W, and classifies an input sample within 25.7 μs. The obtained time latency verifies the real-time classification of a touch modality in less than 400 ms [35]. Applying downsampling and downscaling have decreased the input size from $4 \times 4 \times 30,00$ (a 10s sample) to $4 \times 4 \times 1$ (a 3.5s sample) offering a 65% reduction in data size. Such reduction led to a significant decrease in the hardware resources and time latency. Using the 24-bit fixed-point representation instead of 32-bit floating-point one provides a 25% reduction in the word length of data exchanged between the different blocks of the kNN architecture and the complexity of the arithmetic computations. Consequently, the dynamic power consumption has been reduced. Thus, the proposed approximate kNN offers an average hardware resource reduction up to 56.4%, by accelerating the classification of a test sample by 2.3× with an energy reduction of about 69% compared to the proposed Exact kNN. For the whole design, an accuracy degradation of 2.6% is reported. The proposed approximate kNN provides real-time classification of touch modalities with a reduced time latency of 11.2 μs. These results motivate the use of approximate computing techniques.

Case 2: Exact kNN on FPGA versus GPU

TABLE II
IMPLEMENTATION RESULTS OF THE PROPOSED EXACT AND APPROXIMATE CLASSIFIERS ON ZYNQBERRY

Implementation	Exact	Approximate
Classification Accuracy	89.8%	87%
Frequency (MHz)	100	
BRAM	4	4
DSP48E	10	4
FF	3493	1612
LUT	2825	1264
Time Latency (μs)	25.7	11.2
Dynamic Power Consumption (W)	0.236	0.164

TABLE III
EXACT KNN PERFORMANCE ON FPGA AND GPU

Exact kNN	
Time (FPGA/GPU)	25.7μs/80ms
Energy (FPGA/GPU)	6μJ/1.13J
Time Ratio	0.00032
Energy Ratio	5.37E-6

Using the CUDA platform and NVSMI tool, the GPU implementation of Exact kNN provided a classification time of 80 ms while consuming 14.12W for the touch modality classification problem. Table III shows a comparison between the FPGA and GPU implementations in terms of execution time and energy consumption. The results are significantly in favor of the FPGA, where the acceleration of the proposed kNN architecture on FPGA could be achieved with a fraction of the energy consumed using GPUs. This can be justified due to two possible reasons:

- GPUs use DRAM for the communication between the different blocks of the kNN architecture (referred to as kernels) [36], which is slower than using a hybrid structure as proposed in Figure 2 where BRAMs are used to communicate between different blocks and DRAM is used only for dataset storage.
- The proposed kNN architecture exploits the parallelism capabilities of the FPGA. Thus, the ”if-then-else” conditions are executed in parallel. On the other hand, the ”then” and ”else” parts are executed serially on GPUs resulting in a significant time latency increase. Such issue is known as ”thread divergence” [37].

Case 3: Comparison with similar works

Comparing two kNN implementations is not a straight forward task due to the large number of differences such as: the number of nearest neighbors (K), dataset size (N), number of features per sample (f), development environment (HLS or HDL), hardware device used, etc. To achieve a fair comparison with Exact kNN, three similar architectures have been selected. These three architectures have been chosen such that they all have:

- Used HLS for development since the comparison with an HDL implementation is not feasible.
- Achieved a high acceleration gain (i.e speedup) with respect to equivalent CPU-based kNN implementation, so the architecture resembles an efficient accelerator.
- Used similar (and different) values of K, N, and f to generalize the comparison.

TABLE IV
TESTBENCH IMPLEMENTATION SETTINGS FOR THE EXACT KNN AND THREE SIMILAR ARCHITECTURES

Architecture	S_1 [14]	S_2 [15]	S_3 [21]
K	10	3	5
N	699	150	300×10^3
f	9	4	2
Dataset	BCW_9	Iris	Weather
Device	AVNET ZedBoard	Virtex-7	Virtex-7
Frequency (MHz)	100	100	240

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/OJCAS.2021.3108835, IEEE Open Journal of Circuits and Systems

Table IV presents the existing implementation settings for the three architectures. Denote by $S_i = [K_i, N_i, f_i, Dataset_i]$ the settings used in the first [14], second [15] and third implementation [21] respectively. Exact kNN is implemented using the settings S_i i.e the proposed kNN architecture is implemented and validated using the testbench reported in each architecture. The implementation results are shown in Table V with the original implementation results of each architecture.

kNN-S_1 achieves a $12.3\times$ classification speedup with 94% less energy consumption while requiring a 61% less hardware resources compared to the kNN presented in [14]. This is due to two main reasons: 1) the selector used in kNN-S_1 is an enhanced version of the one used in [14] where the division factor plays a key role in decreasing the sorting time as presented in section IV.C, and 2) the aim of the kNN architecture in [14] is to attain the highest speedup possible for real-time embedded applications. This has been accomplished by combining the UNROLL, PIPELINE, and DATAFLOW directives. Such directives are known for speedup gains due to the level of parallelism introduced at the expense of a noticeable increase in the hardware resources. The latter was not an issue when using the relatively large FPGA in the ZedBoard platform. Meanwhile, in kNN-S_1 only the UNROLL directive is used with an unrolling factor that balances the speedup and complexity for the target application, while achieving more speedup with the use of the selector.

When compared to the kNN implementation in [15], kNN-S_2 provides a huge acceleration gain of $875\times$ with 94% less required hardware resources. Such gain is due to the design choices adopted by the authors in [15] such as: 1) using the euclidean distance metric, which compared to (1), has an added complexity due to the square root operation, 2) applying the normalization of the data on-chip, which presents a complexity overhead, and 3) performing the sorting operation using a single comparator and multiple BRAMs to compare each pair of data points, this process is very slow compared to the proposed selector. Although no power/energy details were provided for the kNN in [15], kNN-S_2 is expected to be more efficient due to the 90% reduction in the number of DSPs.

The implementation requirements of kNN-S_3 exceeds the capacity of the FPGA fabric in Zynqberry and thus the design couldn't be routed to achieve the 240MHz operating frequency. Thus, for comparison reasons only, kNN-S_3 is implemented on the target device used in [21] i.e Vertix-7 knowing that implementing kNN on Zynqberry has achieved the real-time and low power consumption demands for the target touch modality application as reported in Table II. kNN-S_3 offers a speedup of $1.4\times$ with 41.5% and 12.3% reduction in hardware resources and energy per classification respectively. Such results are justified with the lower number of FF and LUTs required by kNN-S_3. This is expected since the kNN in [21] uses the Bitonic sorter, which is outperformed by the proposed selector as shown in Table I. Compared to kNN-S_1 and kNN-S_2, the gain achieved by kNN-S_3 is relatively lower since the kNN in [21] exploits the high optimization capabilities of OpenCL for extensive computations and large datasets.

VI. Conclusion

This paper introduces an efficient novel architecture for the hardware acceleration of the k-Nearest Neighbor algorithm using a selection-based sorter. The architecture has been coded in HLS, synthesized, and routed on the Zynqberry platform. Two efficient implementations were provided based on exact and approximate computing. The implementations exploit the parallelism nature of the kNN algorithm along with the use of ACTs to achieve real-time classification with relatively low power consumption. Compared to similar state-of-the-art solutions, the proposed Exact kNN offers acceleration gain between $1.4\times$ and $875\times$ with lower energy per classification between 41% and 94% depending on the used settings. Compared to GPU-based implementations, the proposed kNN-FPGA implementation offers efficient and faster classification for the target application. Such results pave the way towards embedding intelligence using a small-sized platform such as the Zynqberry for applications with low power and real-

TABLE V
Implementation Results Comparison

Architecture	kNN-S_1	[14]	kNN-S_2	[15]	kNN-S_3	[21]
Device	Zynqberry				Virtex-7	
Frequency (MHz)	100				240	
BRAM	4	-	4	293	500	512
DSP	7	9	5	47	12	12
FF	2002	9484	827	-	21677	23892
LUT	1607	8845	1407	-	11416	11838
Time Latency (ms)	22×10^{-3}	0.27	12×10^{-3}	10.5	0.88	1.24
Energy per classification (mJ)	4.84×10^{-3}	70×10^{-3}	-	-	1.86	3.17
Average Resources Reduction (%)*	61%		94%		12.3%	
Speedup	12.3x		875x		1.4x	
Energy Reduction (%)	94%		-		41.5%	
Classification Accuracy (%)	96.2%		93.3%		86.5%	

* Calculated for the available resources only e.g. reduction in BRAM and DSP for kNN-S_2 compared to [15], i.e. Reduction = (BRAM-Reduction + DSP-Reduction)/2, where BRAM-Reduction= 100(1-4/293) and DSP-Reduction= 100(1-5/47).

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/OJCAS.2021.3108835, IEEE Open Journal of Circuits and Systems

time requirements. The implementation on different hardware platforms, under different settings, and operating on different dataset size, verifies the efficiency of the proposed selection-based kNN architecture. Future work will involve the investigation of the use of circuit-level approximate computing techniques that are reported to permit noticeable gains in the *performance* of machine learning hardware implementations.

ACKNOWLEDGEMENT

The authors acknowledge partial financial support from TACTIle feedback enriched virtual interaction through virtual realITY and beyond (TACTILITY) project: EU H2020, Topic ICT-25-2018-2020, RIA, Proposal ID 856718.

REFERENCES

[1] J. Sun, W. Du, and N. Shi, "A Survey of kNN Algorithm," *Inf Eng Appl Comput*, vol. 1, May 2018.

[2] Zhe-Hao Li, Ji-Fang Jin, Xue-Gong Zhou, and Zhi-Hua Feng, "K-nearest neighbor algorithm implementation on FPGA using high level synthesis," in *2016 13th IEEE International Conference on Solid-State and Integrated Circuit Technology (ICSICT)*, (Hangzhou, China), pp. 600–602, IEEE, Oct. 2016.

[3] J. Saikia, S. Yin, Z. Jiang, M. Seok, and J.-s. Seo, "K-Nearest Neighbor Hardware Accelerator Using In-Memory Computing SRAM," in *2019 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*, (Lausanne, Switzerland), pp. 1–6, IEEE, July 2019.

[4] V. Kumar and R. Kant, "Approximate Computing for Machine Learning," in *Proceedings of 2nd International Conference on Communication, Computing and Networking* (C. R. Krishna, M. Dutta, and R. Kumar, eds.), vol. 46, pp. 607–613, Singapore: Springer Singapore, 2019.

[5] F. de Dinechin, M. D. Ercegovac, J.-M. Muller, and N. Revol, "Digital Arithmetic," in *Wiley Encyclopedia of Computer Science and Engineering* (B. W. Wah, ed.), p. ecse578, Hoboken, NJ, USA: John Wiley & Sons, Inc., Mar. 2009.

[6] E. Nogues, D. Menard, and M. Pelcat, "Algorithmic-Level Approximate Computing Applied to Energy Efficient Hevc Decoding," *IEEE Transactions on Emerging Topics in Computing*, pp. 1–1, 2016.

[7] A. Ibrahim, M. Osta, M. Alameh, M. Saleh, H. Chible, and M. Valle, "Approximate Computing Methods for Embedded Machine Learning," in *2018 25th IEEE International Conference on Electronics, Circuits and Systems (ICECS)*, (Bordeaux), pp. 845–848, IEEE, Dec. 2018.

[8] H. Younes, A. Ibrahim, M. Rizk, and M. Valle, "Algorithmic Level Approximate Computing for Machine Learning Classifiers," in *2019 26th IEEE International Conference on Electronics, Circuits and Systems (ICECS)*, (Genoa, Italy), pp. 113–114, IEEE, Nov. 2019.

[9] S. Lucas, "A fast exact parallel implementation of the k-nearest neighbour pattern classifier," in *1998 IEEE International Joint Conference on Neural Networks Proceedings. IEEE World Congress on Computational Intelligence (Cat. No.98CH36227)*, vol. 3, (Anchorage, AK, USA), pp. 1867–1872, IEEE, 1998.

[10] E. S. Manolakos and I. Stamoulias, "Flexible IP cores for the k-NN classification problem and their FPGA implementation," in *2010 IEEE International Symposium on Parallel & Distributed Processing, Workshops and Phd Forum (IPDPSW)*, (Atlanta, GA, USA), pp. 1–4, IEEE, Apr. 2010.

[11] H. Hussain, K. Benkrid, C. Hong, and H. Seker, "An adaptive FPGA implementation of multi-core K-nearest neighbour ensemble classifier using dynamic partial reconfiguration," in *22nd International Conference on Field Programmable Logic and Applications (FPL)*, (Oslo, Norway), pp. 627–630, IEEE, Aug. 2012.

[12] Y. Pu, J. Peng, L. Huang, and J. Chen, "An Efficient KNN Algorithm Implemented on FPGA Based Heterogeneous Computing System Using OpenCL," in *2015 IEEE 23rd Annual International Symposium on Field-Programmable Custom Computing Machines*, (Vancouver, BC, Canada), pp. 167–170, IEEE, May 2015.

[13] F. Muslim, A. Demian, L. Ma, L. Lavagno, and A. Qamar, "Energy-efficient FPGA Implementation of the k-Nearest Neighbors Algorithm Using OpenCL," pp. 141–145, Oct. 2016.

[14] D. Jamma, O. Ahmed, S. Areibi, and G. Grewal, "Hardware accelerators for the K-nearest neighbor algorithm using high level synthesis," in *2017 29th International Conference on Microelectronics (ICM)*, (Beirut, Lebanon), pp. 1–4, IEEE, Dec. 2017.

[15] M. A. Mohsin and D. G. Perera, "An FPGA-Based Hardware Accelerator for K-Nearest Neighbor Classification for Machine Learning on Mobile Devices," in *Proceedings of the 9th International Symposium on Highly-Efficient Accelerators and Reconfigurable Technologies*, (Toronto ON Canada), pp. 1–7, ACM, June 2018.

[16] X. Song, T. Xie, and S. Fischer, "A Memory-Access-Efficient Adaptive Implementation of kNN on FPGA through HLS," in *2019 IEEE 37th International Conference on Computer Design (ICCD)*, (Abu Dhabi, United Arab Emirates), pp. 177–180, IEEE, Nov. 2019.

[17] K. Chomboon, P. Chujai, P. Teerarassammee, K. Kerdprasop, and N. Kerdprasop, "An Empirical Study of Distance Metrics for k-Nearest Neighbor Algorithm," in *The Proceedings of the 2nd International Conference on Industrial Application Engineering 2015*, pp. 280–285, The Institute of Industrial Applications Engineers, 2015.

[18] You Yang, Ping Yu, and Yan Gan, "Experimental study on the five sort algorithms," in *2011 Second International Conference on Mechanic Automation and Control Engineering*, (Inner Mongolia, China), pp. 1314–1317, IEEE, July 2011.

[19] J. Vieira, R. P. Duarte, and H. C. Neto, "kNN-STUFF: kNN STreaming Unit for Fpgas," *IEEE Access*, vol. 7, pp. 170864–170877, 2019.

[20] D. Jamma, O. Ahmed, S. Areibi, G. Grewal, and N. Molloy, "Design exploration of ASIP architectures for the K-Nearest Neighbor machine-learning algorithm," in *2016 28th International Conference on Microelectronics (ICM)*, (Giza, Egypt), pp. 57–60, IEEE, Dec. 2016.

[21] F. B. Muslim, L. Ma, M. Roozmeh, and L. Lavagno, "Efficient FPGA Implementation of OpenCL High-Performance Computing Applications via High-Level Synthesis," *IEEE Access*, vol. 5, pp. 2747–2762, 2017.

[22] Q. Mu, L. Cui, and Y. Song, "The implementation and optimization of Bitonic sort algorithm based on CUDA," *arXiv:1506.01446 [cs]*, June 2015.

[23] H. Younes, A. Ibrahim, M. Rizk, and M. Valle, "Data Oriented Approximate K-Nearest Neighbor Classifier for Touch Modality Recognition," in *2019 15th Conference on Ph.D Research in Microelectronics and Electronics (PRIME)*, (Lausanne, Switzerland), pp. 241–244, IEEE, July 2019.

[24] N. Jamali and C. Sammut, "Majority Voting: Material Classification by Tactile Sensing Using Surface Texture," *IEEE Transactions on Robotics*, vol. 27, pp. 508–521, June 2011.

[25] J. Kwiatkowski, D. Cockburn, and V. Duchaine, "Grasp stability assessment through the fusion of proprioception and tactile signals using convolutional neural networks," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, (Vancouver, BC), pp. 286–292, IEEE, Sept. 2017.

[26] M. Kaboli, A. Long, and G. Cheng, "Humanoids learn touch modalities identification via multi-modal robotic skin and robust tactile descriptors," *Advanced Robotics*, vol. 29, pp. 1411–1425, Nov. 2015.

[27] P. Gastaldo, L. Pinna, L. Seminara, M. Valle, and R. Zunino, "A Tensor-Based Pattern-Recognition Framework for the Interpretation of Touch Modality in Artificial Skin Systems," *IEEE Sensors Journal*, vol. 14, pp. 2216–2225, July 2014.

[28] M. Alameh, A. Ibrahim, M. Valle, and G. Moser, "DCNN for Tactile Sensory Data Classification based on Transfer Learning," in *2019 15th Conference on Ph.D Research in Microelectronics and Electronics (PRIME)*, (Lausanne, Switzerland), pp. 237–240, IEEE, July 2019.

[29] H. Younes, A. Ibrahim, M. Rizk, and M. Valle, "Algorithmic-Level Approximate Tensorial SVM Using High-Level Synthesis on FPGA," *Electronics*, vol. 10, p. 205, Jan. 2021.

[30] P. Gastaldo, L. Pinna, L. Seminara, M. Valle, and R. Zunino, "Computational Intelligence Techniques for Tactile Sensing Systems," *Sensors*, vol. 14, pp. 10952–10976, June 2014.

[31] *Weka 3 - Data Mining with Open Source Machine Learning Software in Java*.

[32] *TE0726 Resources - Public Docs - Trenz Electronic Wiki*.

[33] X. Xilinx, *Vivado Design Suite User guide, High-Level Synthesis.*

[34] NVIDIA, *NVIDIA System Management Interface*. June 2012. Publication Title: NVIDIA Developer.

[35] P. P. Lele, D. C. Sinclair, and G. Weddell, "The reaction time to touch," *The Journal of Physiology*, vol. 123, pp. 187–203, Jan. 1954.

[36] L. Sánchez, J. Ranilla, and A. Cocaña-Fernández, "Eecluster: An energyefficient tool for managing hpc clusters," *Annals of Multicore and GPU Programming*, vol. 2, no. 1, pp. 15–24, 2015.

[37] S. Cook, "Memory Handling with CUDA," in *CUDA Programming*, pp. 107–202, Elsevier, 2013.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/OJCAS.2021.3108835, IEEE Open Journal of Circuits and Systems

Hamoud Younes He received the B.S and M.S. degrees in Computer and Communication Engineering from the Lebanese International University, in 2012 and 2015 respectively. Currently, he is a Ph.D. student at the Department of Electric, Electronic, Telecommunication Engineering and Naval Architecture, University of Genoa. His research interests involve embedded electronic systems, FPGA implementation, embedded machine and deep learning, approximate computing, and energy efficient embedded computing.

Ali Ibrahim received the M.S. degree in industrial control from the Doctoral School of Sciences and Technologies, Lebanese University, in 2009, and the dual Ph.D. degrees in electronic and computer engineering and robotics and telecommunications from the University of Genova and the Lebanese University in 2016. He has been a Post-Doctoral Researcher with the Department of Electric, Electronic, Telecommunication Engineering and Naval Architecture, University of Genova from 2016 to 2018. Currently, he is an assistant professor in the department of Electrical and Electronics Engineering at the Lebanese international University in Lebanon and also an associate researcher Department of Electric, Electronic, Telecommunication Engineering and Naval Architecture, University of Genoa. His research interests involve Embedded machine learning, FPGA implementation, and interface electronics for electronic skin systems, approximate computing, and techniques and methods for energy efficient embedded computing.

Mostafa Rizk received the Maitrise degree in electronics, the M.Sc. degree in biomedical physics, and the M.Sc. degree in signal, telecom, image, and speech from Lebanese University, Beirut, Lebanon, in 2007, 2008, and 2010, respectively, the Ph.D. degree in sciences and technologies of information from Telecom Bretagne, Brest, France, in 2014, and the Ph.D. degree in electronics and communication from Lebanese University in 2015. He has been a Post-Doctoral Researcher with the University of Southern Brittany, Lorient, France, and with the Lab-STICC Laboratory CNRS, Lorient. He is currently an Assistant Professor with Lebanese International University, Beirut, and an Associate Researcher with IMT Atlantique, Brest, France. His current research interests include hardware/software implementations and digital circuit design, network-on-chip design, and new MPSoC architectures based on emerging nonvolatile memory technologies.

Maurizio Valle (MV) received the M.S. degree in Electronic Engineering in 1985 and the Ph.D. degree in Electronics and Computer Science in 1990 from the University of Genova, Italy. From December 2019, MV is full professor of Electronics at the DITEN, University of Genova where he leads the Connected Objects, Smart Materials, Integrated Circuits – COSMIC laboratory. MV has been and is in charge of many research contracts and projects funded at local, national and European levels and by Italian and foreign companies. Professor Valle is co-author of more than 200 papers on international scientific journals and conference proceedings. He is IEEE senior member and member of the IEEE CAS Society. His research interests include bio-medical circuits and systems, electronic/artificial sensitive skin, tactile sensing systems for prosthetics and robotics, neuromorphic touch sensors, electronic and microelectronic systems.

7.4.3 Real-time Accelerated Tensorial Support Vector Machine Architecture

Tensor based learning techniques permit the effective exploitation of the structure of data used in various fields such as vision (e.g. image recognition), neuroscience (e.g. MRI data), chemistry (excitation-emission data), etc. At the beginning of the last decade, ML communities started showing interest in tensors and their use for supervised learning [181]. Authors in [182] proposed a tensorial kernel that could be used for supervised tensor-based learning models while utilizing the structural information embodied in the data and exploiting the algebraic properties of tensors of any order. Such kernel methods lead to flexible nonlinear models that have been proven successful in many different contexts. When used with SVM algorithm, the tensorial kernel achieved better classification accuracy than the Gaussian-Radial Basis Function and linear kernels in an image recognition task.

Gastaldo *et. al* have extended the tensorial kernel approach for tactile data processing in [183]. This approach has been adopted for a touch modality classification problem since it preserves the inherent tensorial structure of the data collected by tactile sensors. As an end result, the tensorial-based SVM has achieved higher accuracy in classifying touch modalities compared to the Regularized Least Square algorithm. In [184], the first FPGA implementation of the SVM algorithm based on tensorial kernel has been presented. Specifically, two implementations were provided: Cascaded and Parallel. The former failed to ensure real-time classification of touch (i.e in less than $400ms$ [154]), and the latter reported a relatively large hardware area and high power consumption of 1.14W. Such results were not acceptable for applications with limited power budget and area constraints [185].

In this research work, we present a new architecture and hardware implementation of the TSVM aiming at reducing the hardware complexity and power consumption while keeping real-time operation. The architecture is characterized by the introduction of a Shallow Neural Network for the SVD computations. The proposed neural network architecture achieves $324\times$ speedup with 58% and 67% reductions in the required hardware resources and power consumption respectively compared to the traditional one-sided Jacobi algorithm. Such reductions demonstrate the feasibility of the implemented TSVM for real-time tactile data classification while consuming 6.28 mJ. The proposed TSVM architecture achieves $131\times$ classification speedup with a 39% and 50% resources and power reductions respectively compared to similar stat-of-the-art solution [184]. Furthermore, a scalability assessment of the proposed TSVM architecture is provided. The assessment shows that replacing the one-sided Jacobi with a neural network demands only 1% increase in the required flip flops compared to 29% when the number of training tensors is doubled.

This research work has been published in the proceeding of the *IEEE International Conference on Artificial Intelligence Circuits and Systems (AICAS)* [186] and then extended in *IEEE Transactions on Circuits and Systems I: Regular Papers* [187].

A Shallow Neural Network for Real-Time Embedded Machine Learning for Tensorial Tactile Data Processing

Hamoud Younes, *Graduate Student Member, IEEE*, Ali Ibrahim, *Member, IEEE*, Mostafa Rizk, *Member, IEEE*, and Maurizio Valle, *Senior Member, IEEE*

***Abstract*— This paper presents a novel hardware architecture of the Tensorial Support Vector Machine (TSVM) based on Shallow Neural Networks (NN) for the Single Value Decomposition (SVD) computation. The proposed NN achieves a comparable Mean Squared Error and Cosine Similarity to the widely used one-sided Jacobi algorithm. When implemented on an FPGA, the NN offers 324× faster computations than the one-sided Jacobi with reductions up to 58% and 67% in terms of hardware resources and power consumption respectively. When validated on a touch modality classification problem, the NN-based TSVM implementation has achieved a real-time operation while consuming about 88% less energy per classification than the Jacobi-based TSVM with an accuracy loss of at most 3%. Such results offer the ability to deploy intelligence on resource-limited platform for energy-constrained applications.**

***Index Terms*— Embedded machine learning, real-time, tensorial kernel, tactile sensors, neural networks, singular value decomposition, FPGA.**

I. Introduction

TENSOR based learning techniques permit the effective exploitation of the structure of data used in various fields such as vision (e.g. image recognition), neuroscience (e.g. MRI data), etc. Authors in [1] proposed a tensorial kernel that could be used for supervised tensor-based learning models while utilizing the structural information embodied in the data. When used with Support Vector Machine (SVM) algorithm, the tensorial kernel leads to better classification accuracy than the Gaussian-Radial Basis Function (RBF) and linear kernels in an image recognition task.

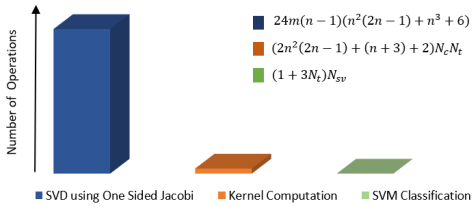


Fig. 1. Computational Complexity of the Tensorial SVM algorithm.

Gastaldo *et. al.* have extended the tensorial kernel approach for tactile data processing in [2]. This approach has been adopted since it preserves the inherent tensorial structure of the data collected by tactile sensors. As an end result, the tensorial-based SVM achieves higher accuracy in classifying touch modalities compared to the Regularized Least Square (RLS) algorithm. In [3], the first FPGA implementation of the Support Vector Machine (SVM) algorithm based on tensorial kernel has been presented. Specifically, two implementations were provided: Cascaded and Parallel. The former failed to ensure real-time classification of touch (i.e in less than 400 *ms* [4]), and the latter reported a relatively large hardware area and high power consumption of 1.14W. Such results were not acceptable for the application with limited power budget and area constraints [5].

In this paper, our main goal is to provide a new architecture and hardware implementation of the tensorial SVM (TSVM) aiming at reducing the hardware complexity and power consumption while keeping real-time operation. For this purpose, we analyzed the complexity of the tensorial SVM architecture to pin-out most computationally complex and demanding blocks. Fig. 1 illustrates the estimated number of operations required in each step of the tensorial SVM algorithm, where m and n are the dimensions of the unfolded matrix, N_c, N_t, and N_{sv} are the number of classes to be discriminated, the number of training tensors, and the number of support vectors, respectively. In [3], the one-sided Jacobi algorithm has been adopted for finding the singular vectors. Such algorithm involves a high number of arithmetic operations and

Manuscript received March 31, 2021; revised June 11, 2021 and July 8, 2021; accepted July 25, 2021. This work was supported in part by the TACTIle feedback enriched virtual interaction through virtual realITY and beyond (TACTILITY) Project under Grant EU H2020 and Grant Topic ICT-25-2018-2020, Research and Innovation Actions (RIA) Proposal ID 856718. This article was recommended by Associate Editor P. K. Meher. *(Corresponding author: Hamoud Younes.)*

Hamoud Younes and Ali Ibrahim are with the Department of Electrical, Electronic and Telecommunications Engineering and Naval Architecture, University of Genova, 16145 Genova, Italy, and also with the Department of Computer and Communication Engineering, Lebanese International University, Beirut 146404, Lebanon (e-mail: hamoud.younes@edu.unige.it; ali.ibrahim@edu.unige.it).

Mostafa Rizk is with the Department of Computer and Communication Engineering, Lebanese International University, Beirut 146404, Lebanon (e-mail: mostafa.rizk@liu.edu.lb).

Maurizio Valle is with the Department of Electrical, Electronic and Telecommunications Engineering and Naval Architecture, University of Genova, 16145 Genova, Italy (e-mail: maurizio.valle@unige.it).

Color versions of one or more figures in this article are available at https://doi.org/10.1109/TCSI.2021.3102303.

Digital Object Identifier 10.1109/TCSI.2021.3102303

requires several iterations to converge [6]. As reported in [3] ($m = 8, n = 20, N_c = 2, N_t = 100$, etc.), the Singular Value Decomposition (SVD) computation corresponds to about 96% of the overall algorithm. Thus, the main focus of the proposed new architecture is to find an alternative algorithm for SVD computation. This alternative should impose complexity reductions without affecting the classification accuracy of the tensorial SVM.

A Neural Network (NN) is one of the candidates for the SVD computation. The idea first surfaced in 1991 when Samardzija *et. al.* proposed an artificial continuous-time neural network to estimate the eigenvectors and eigenvalues [7]. In [8], the convergence and computational complexity through computer simulations of such network are assessed. Another neural network has been presented in [9]. The network is characterized by an order n-Ordinary Differential Equations (ODEs) leading to reduced dimensionality. Such neural network has evolved to further applications such as Principle Component Analysis (PCA) [10].

Triggered by the performance of neural networks in many domains [11] and the continuous quest for efficient designs specifically for resource-limited applications [12], a neural network based tensorial SVM architecture is proposed. The main contributions of this paper are summarized as follows:

- It presents a novel architecture for SVD computation using shallow neural networks. The architecture achieves 324× speedup with 58% and 67% reductions in the required hardware resources and power consumption respectively compared to the traditional one-sided Jacobi algorithm. Such reductions are obtained while providing a comparable performance in terms of Mean Squared Error (MSE) and Cosine Similarity (CS) metrics. Moreover, the proposed architecture is adequate for implementations on resource-limited platforms (e.g. Zynqberry [13]).
- It presents the first hardware architecture and implementation of SVM algorithm featuring multidimensional tensorial inputs, where shallow neural networks are employed to compute the singular value decomposition.
- It demonstrates the feasibility of the implemented system for real-time touch modality classification while consuming 6.28 mJ. The proposed cascade architecture achieves 131× classification speedup with a 39% and 50% resources and power reductions respectively compared to similar stat-of-the-art solution [3].
- It provides scalability assessment of the proposed SVD architecture. Replacing the one-sided Jacobi with a neural network in the tensorial SVM architecture reported only 1% increase in the required FFs compared to 29% when the number of training tensors is doubled.

The rest of the paper is organized as follows: Section II presents an overview of the tensorial SVM for touch modality classification. Section III provides a review on the efficient existing SVD algorithms and their hardware implementations. It also reports the complexity of the proposed architecture compared to existing solutions. Section IV details the process of designing a neural network for SVD and its performance when tested on a tactile dataset. Section V provides the FPGA implementation and verification of the tensorial SVM based on SVD computation via shallow neural networks. Section VI presents a scalability study of the proposed architecture in terms of hardware resources and time latency. Section VII concludes the paper and illustrates some observations.

II. SVM Classification Based on Tensorial Kernel

A. Overview

A theoretical approach that extends kernel methods to tensor data has been presented in [14]. The framework allows the classification of an input tensor using SVM in 4 main steps:

- Tensor Unfolding: A tensor $\phi(I_1 \times I_2 \times I_3)$ is transformed into three matrices $X_1(I_1 \times I_2 I_3)$, $X_2(I_2 \times I_1 I_3)$ and $X_3(I_3 \times I_1 I_2)$.
- SVD Computation: The unfolded matrices are symmetrized into square matrices that can be written in the form:

$$X_1 = USV^T \tag{1}$$

  where U and V^T contain the left and right singular vectors respectively, and S is the diagonal matrix storing the singular values σ_i of X_1.
- Kernel Computation: The tensorial kernel extended from the Gaussian kernel is computed using the function:

$$K(x, y) = \prod_{1}^{z} k^z(x, y) \tag{2}$$

  where k^z is the kernel factor defined as:

$$k(x, y) = \exp(\frac{-1}{2\sigma^2}(I_n - trace(Z^T Z))) \tag{3}$$

  where $Z = V_x^T V_y$, V_x and V_y represent the singular vectors of the unfolded matrix obtained during the inference and training phase respectively, and trace represents the sum of diagonal elements.
- Classification: Applying the SVM classification function expressed as:

$$\hat{y} = f_{SVM}(x) = \sum_{i}^{n} \beta_i K(x_i, x) + b \tag{4}$$

  where $\hat{y}$ is the predicted label of input tensor x, n is the number of training tensors, β_i are the coefficients obtained during training, and b is the bias.

B. Touch Modalities Classification

The tensorial SVM has been initially presented as an effective algorithm for touch modality classification in [14]. In this paper, three binary and one multi-class classification problems are used to test the accuracy of the proposed neural network based tensorial SVM. Specifically, the problems are:

- Problem A: “brushing a paintbrush” versus “rolling a washer”
- Problem B: “brushing a paintbrush” versus “sliding the finger”
- Problem C: “sliding the finger” versus “rolling a washer”
- Problem D: “one versus the others”

These modalities are derived from a tactile dataset that has been collected by 70 participants. Each participant performed the modality on both the horizontal and vertical axes of a 4×4 tactile sensor for a duration of 10 seconds. Thus each touch modality is represented by a tensor $\phi(4 \times 4 \times 30,000)$. However, such tensor size is reduced into $\phi(4 \times 4 \times 20)$ where 20 is the obtained number of samples using the data pre-processing algorithm (Algorithm 1) reported in section IV.

III. SVD Algorithms and Implementations

A. Literature Review

Singular value decomposition can be computed numerically through several methods such as the Jacobi method, the QR method, and the one-sided Hestenes method [15]. For parallel implementations, computing the SVD using the Jacobi method is superior to other methods in terms of complexity and execution time [15]. Brent *et. al.* have shown that two-dimensional systolic array could be used for implementing the Jacobi method [16]. In [17], the authors have presented various realization for the Jacobi SVD computation using Coordinate Rotation Digital Computer (CORDIC) [18]. The latter is adopted in majority of the existing hardware implementations of the Jacobi SVD method. For small matrix dimensions, an efficient implementation of SVD for the use in Multiple Input Multiple Output (MIMO) precoding and real-time signal processing has been presented in [19]. The implementation is based on CORDIC processors. For an arbitrary $m \times n$ matrix, Ibrahim *et. al.* have presented an FPGA implementation with fixed-point arithmetic [20]. The implementation managed to compute the SVD of an 32×127 matrix in 13 ms while occupying 20% and 67% slice registers and LUTs respectively on a Virtex-6 FPGA. Fast and efficient FPGA implementation for computing the singular and eigen value decomposition based on a simplified CORDIC-like algorithm is presented in [21]. The implementation uses fixed-point arithmetic for sequential and parallel operations leading about $3\times$ faster computation in an image denoising application compared to computations via an Intel CPU based PC. The authors in [22] used High-Level Synthesis (HLS) to model the one-sided Jacobi SVD computation on a Zedboard development board. For a 16×16 matrix, SVD computation takes around 1.1 seconds with a power consumption of 1.38W. Using CMOS 28-nm technology, Deng et.al proposed a hardware architecture for tensor SVD [23]. Compared with real-world CPU-based implementations, the architecture provides an average of $14\times$ speed on various workloads.

Targeting the TSVM architecture in [3] where the one-sided Jacobi is identified as a performance bottleneck, the existing alternative implementations for SVD computation share several common challenges: (1) they operate only on square matrices. Thus, if the input matrix is rectangular, an additional complexity is added due to matrix symmetrization [23]. (2) if the implementation uses floating-point representation, the complexity is relatively high even for small matrix dimensions [24], and (3) depending on the required output precision, the algorithm might require additional iterations to converge [6]. Recently, a scalable SVD engine on FPGA has been introduced in [25] targeting these challenges. The proposed engine managed to compute the SVD of rectangular matrices using floating-point arithmetic. However, the implementation results show that a large number of DSPs is required for several matrix dimensions which has a direct impact on the power consumption of hardware implementations. Another noticeable observation is that the authors compared the SVD engine only to CPU-based SVD computations. In this paper, a new architecture for SVD computation based on shallow neural networks is proposed. The architecture offers the ability to operate on rectangular matrices (thus symmetrization is not needed, see Fig. 2) and utilizes floating-point arithmetic. As for convergence, the neural network training is usually performed offline on a high-end computing device. Thus, a network could be trained several times for any given amount of time to achieve top notch performance.

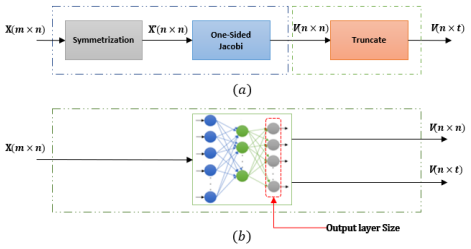


Fig. 2. SVD Computation using: (a) one-sided Jacobi, (b) Neural Network.

B. Computational Complexity

In this section, we compare the complexity of the one-sided Jacobi algorithm with that of a shallow neural network in terms of the total number of operations. Consider a shallow neural network of one hidden layer of size H and an output layer of size O. For an input $A_{m\times n}$, the outputs of the hidden layer Y_h and the output layer Y_O are expressed respectively as:

$$Y_h = f_h(W_h.A + b_h) \tag{5}$$

$$Y_O = f_O(W_O.Y_h + b_O) \tag{6}$$

where W, b, and f represent the weight, bias, and activation function respectively. The output of each layer consists of matrix multiplication, addition, and activation operations. The number of operations for matrix multiplication and addition is expressed as:

$$N_h = H(2m \times n - 1) + H = 2H(m \times n) \tag{7}$$

Assuming that the activation function requires N_{Act} operations, the total number of operations in the hidden layers is expressed as:

$$N_h = 2H(m \times n) + N_{Acth} \tag{8}$$

The same can be applied to the output layer, thus the number of required operations is:

$$N_O = 2H \times O + N_{Act\,O} \tag{9}$$

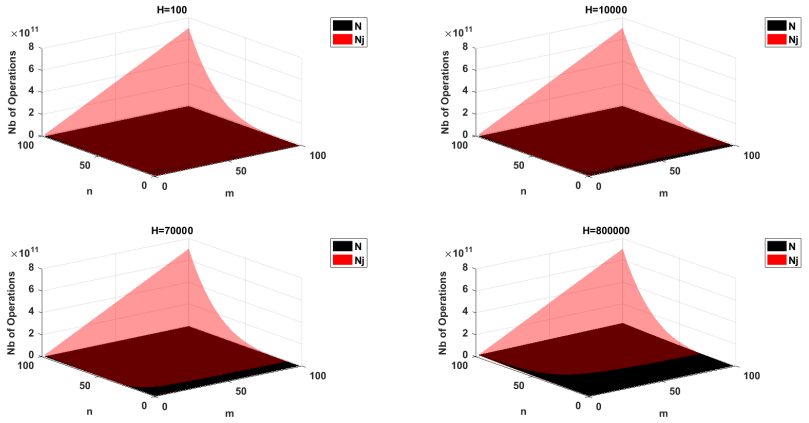


Fig. 3. Number of Operations required in one-sided Jacobi (Nj) and Shallow Neural Network (N), (m,n) are the matirx dimension and H is the hidden layer size.

Finally, the number of operations for the whole network could be expressed as:

$$N = N_h + N_O = 2H(m \times n + O) + N_{Acth} + N_{Act\,O} \quad (10)$$

To estimate N, suppose there exists an upper bound T such that $N \leq T$. T is an upper bound when both N_{Acth} and $N_{Act\,O}$ correspond to the most complex activation function i.e. the tangent hyperbolic function (tanh). The latter is expressed as:

$$f(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}} \quad (11)$$

To find the number of operations required for the term e^z, we referred to the function implementation in the IEEE-754 library in [26]. The implementation uses the Taylor expansion with an order 3 for floating-points, leading to a total of 16 operations. Thus the number of operations $N_{Acth} = 35H$. Similarly, $N_{Act\,O} = 35O$. For the network to output the right singular vectors V of an $m \times n$ matrix, the output layer size O is equal to n^2. This simplifies (10) to:

$$N <= (2H \times n)(m + n) + 35H + 35n^2 \quad (12)$$

Knowing that the number of operations for the one-sided Jacobi algorithm is (see Fig. 1):

$$Nj = 24m(n-1)[n^2(2n-1) + n^3 + 6] \quad (13)$$

through simulations, the values of m, n, and H are varied to compare (12) and (13). Fig. 3 plots the number of operations N_j and N required to compute the SVD of a matrix using one-sided Jacobi and a shallow neural network respectively. Generally, the comparison results are in favor of the neural network approach as shown in Fig. 3. The one-sided Jacobi is superior for very small dimensions such as 2×2 for $H > 21$. As the dimension starts to increase, the neural network requires significantly less number of operations for SVD computations. For instance, for $(m, n) = (20, 16)$ and $(m, n) = (4, 80)$ (these dimensions are often used for tensorial SVD implementations based on the one-sided Jacobi algorithm [3], [27]), computing the right singular vectors V using a shallow neural network requires less number of operations than using the one-sided Jacobi ($N < Nj$) for all values of $H \leq 70,000$ and $H \leq 800,000$ respectively. Such values of H are very large even for the largest existing neural networks. The number of operations (N) required for each activation function is presented in Table I. The hidden layer activation function could be ReLU (Standard, LeakyReLU or Parametric), Sigmoid, or hyperbolic tangent (tanh), it is selected based on the trade-off between complexity and the required performance. In the output layer, only the hyperbolic tangent function can be used, due to the fact that the values of the singular vectors are bounded between -1 and 1.

TABLE I
COMPLEXITY ASSESSMENT UNDER DIFFERENT ACTIVATION FUNCTIONS

Activation Function		Number of Operations (N)
Hidden Layer	Output Layer	
ReLU	Tanh	$(2H \times n)(m+n) + H + 35n^2$
Leaky/PReLU		$(2H \times n)(m+n) + 2H + 35n^2$
Sigmoid		$(2H \times n)(m+n) + 18H + 35n^2$
Tanh		$(2H \times n)(m+n) + 35H + 35n^2$

IV. SVD USING NEURAL NETWORKS

A. Network Structure

A regression model is targeted since the NN is needed to compute the singular vectors. For that, there are two possible categories to work on: (1) Classification NN that should be modified to perform regression and re-trained [28] and (2) Regression NNs [29]. Although the classification accuracy achieved by the one-sided Jacobi TSVM could be obtained by an existing NN model from the two above mentioned categories, the main concern remains in the computational

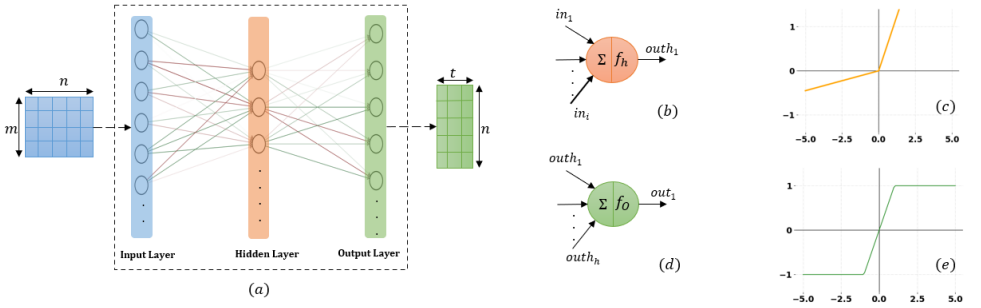


Fig. 4. Proposed Shallow Neural Network: (a) Overall Structure, (b) Hidden Layer Neuron, (c) PReLU Activation Function, (d) Output Layer Neuron, (e) Approximate Hyperbolic Tangent Activation Function.

complexity of such model. Concerning the first category, one could choose Convolutional NN (CNNs), Multi-Layer Perceptron (MLP), Long-short Term Memory (LSTM), etc. Even the smallest models such as MobileNet [30], Shuffle Net [31], and EffNet [32] contains at least four layers. On the other hand, for the regression NNs, with only one hidden layer, a shallow network is considered to be the smallest possible regression NN model.

A tactile tensor $\phi(4\times4\times20)$ is unfolded into three matrices $M(4\times80)$, $N(4\times80)$, and $P(20\times16)$. According to (1) each matrix could be decomposed into:

$$M_{4\times80} = U_{4\times80} \times \Sigma_{80\times80} \times V^T_{80\times80} \tag{14}$$

$$N_{4\times80} = U_{4\times80} \times \Sigma_{80\times80} \times V^T_{80\times80} \tag{15}$$

$$P_{20\times16} = U_{20\times16} \times \Sigma_{16\times16} \times V^T_{16\times16} \tag{16}$$

Authors in [14] and [33] reported that for tensor SVD, only a small number of the columns of V is required to obtain acceptable classification accuracy when embedded in SVM. Using the three touch modality problems reported in section II.B, the V matrices that resulted in the highest classification accuracy are: $V^T_{80\times4}$, $V^T_{80\times4}$, and $V^T_{16\times2}$.

Fig. 4(a) shows the proposed shallow neural network that is capable of computing the right singular vectors V. The network is composed of three fully connected layers: an input layer of size $m \times n$, a hidden layer of size H, and an output layer of size $O = n \times t$, where t is the selected number of columns from V. Thus, two neural networks are designed. one with an 80×4 output and the other with a 16×2 output.

The neural networks share two activation functions (f_h) and (f_O) defined as:

$$f_h(z_i) = \begin{cases} z_i & z_i \geq 0 \\ \beta z_i & otherwise \end{cases} \tag{17}$$

$$f_O(z) = \begin{cases} -1 & z < -1 \\ 1 & z > 1 \\ z & otherwise \end{cases} \tag{18}$$

The function f_h shown in Fig. 4(c) is called Parametric Rectified Linear Unit (PReLU) where z_i is just one feature out of the feature vector z and β is a learnable weight used to keep negative values compared to the standard ReLU function. It is adopted for the hidden layer to preserve the sign of the neurons' output with low computational complexity compared to other activation functions (e.g. Sigmoid function). The function f_O shown in Fig. 4(e) is called hard hyperbolic tangent activation function [34]. It is used at the output layer to output the elements v_i of the V matrix in the range $[-1, 1]$ with a reduced computational complexity compared to the hyperbolic tangent function.

B. Network Training and Tuning

The chosen network model is trained using floating-point representation during both forward and backward propagation. The network is trained to export the right singular vectors V with the least possible error margin compared to exact computations obtained via MATLAB. The proposed network is a regression model that outputs singular vectors, for that the performance is determined based on two metrics: (1) Mean Squared Error (MSE) and (2) Cosine Similarity (CS). These metrics are defined as:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(V_i - \hat{V}_i)^2 \tag{19}$$

$$CS = \frac{1}{n}\sum_{i=1}^{n}\left(\frac{V_i.\hat{V}_i}{||V_i|| \times ||\hat{V}_i||}\right) \tag{20}$$

where V is the matrix generated from the neural network and $\hat{V}$ is the one generated from applying the SVD using MATLAB software. Thus, the training aims at finding a network model that achieves the lowest MSE (i.e. the elements v_i of the V and $\hat{V}$ matrices have similar values) and highest CS (i.e. the vectors V_i of the V and $\hat{V}$ matrices have similar direction i.e CS tends to 1).

The proposed neural network is hand crafted and can be customized. The training process is used to tune the network hyperparameters [35] i.e. parameters that determines the network structure and training behavior (e.g. size of hidden layer H, learning rate) and parameters (e.g. weights). During

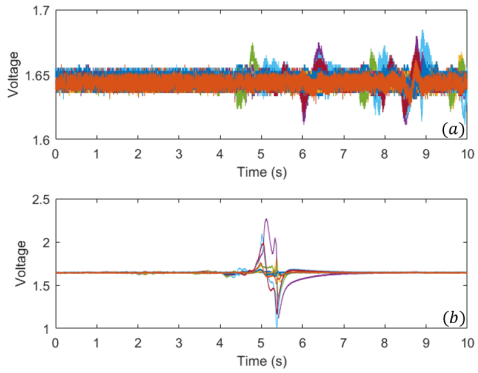


Fig. 5. Touch Modality with: (a) Noisy Readings, (b) Silent Intervals.

training, the weights and biases of the network are randomly initialized, then updated using one of the below optimizers. As for the hyperparameters, the following settings have been tested:

- $H = [10, 20, \ldots\ldots 200]$
- PReLU β: learned parameter through Channel-wise or channel-shared modes [36]
- Learning rate $= [0.1, 0.01, 0.001, \ldots 10^{-5}]$
- Optimizer: [SGD, Adam, Adadelta, RMSprop]
- Batch size $= [50, 100, 150]$

The tactile dataset from [14] is used for training. However, some modifications have been applied based on the following:

- Some participants recordings are noisy (see Fig. 5(a)), thus their corresponding data has been removed from the training dataset.
- Since no particular indications were given to the participants in [14] about the pressure level, silent intervals (i.e. voltage readings from sensor taxels equals to zero, see Fig. 5(b)) are observed in the recordings. These silent intervals will not help the neural network to learn new patterns and thus are removed. Specifically, all reading outside the timing interval [3.5, 7] are omitted.

Algorithm 1 summarizes the pre-processing technique applied to the dataset. The algorithm truncates each modality from $10s$ to $3.5s$ resulting in a tensor $T'(4 \times 4 \times 10,500)$. Afterwards, subsampling is applied to obtain 20 readings ($P = 20$) from the 10,500 resulting in a final tensor $\phi(4 \times 4 \times 20)$. After pre-processing, 4480 matrices of dimensions 4×80 and 20×16 have been derived. Then, their corresponding V matrices are generated using MATLAB. These matrices are divided into 80% for training, 10% for validation, and 10% for testing.

C. Network Performance

The neural network is coded in Python using Tensorflow and Keras libraries. Then, it is trained on an ASUS PC equipped with an NVIDIA GTX 1650 graphics card with 4GB VRAM.

Algorithm 1 Pre-Processing Algorithm

Input: Tensor T of size (:,:,S),
Time Interval [a, b]
Sampling parameter P
Output: Sampled Tensor ϕ of size (:,:,P)
Let $\upsilon 1 \leftarrow a \times S/10$
Let $\upsilon 2 \leftarrow b \times S/10$
Let $S' \leftarrow \upsilon 2 - \upsilon 1$
Let T' be a Tensor of size (:,:,S')
Let $j = 0$
for $i \leftarrow \upsilon 1$ ***to*** $\upsilon 2$ **do**
  $T'(:,:,j) \leftarrow T(:,:,i)$
  $j++$
Let $k = 0$
for $i \leftarrow 0$ ***to*** P **do**
  $\phi(:,:,i) \leftarrow (P/S') * \sum_{i=k}^{S'/P+k} T'(:,:,i)$
  $k+ = S'/P$

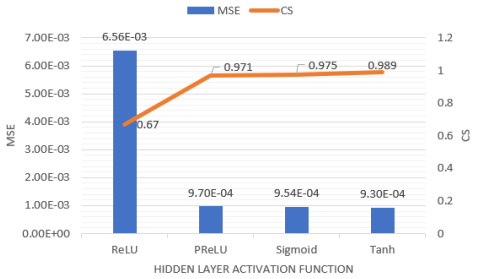


Fig. 6. Network Performance Under Different Activation Functions.

TABLE II
BEST MODEL PERFORMANCE COMPARED TO ONE-SIDED JACOBI

Input Layer Size	20×16	4×80
Hidden Layer Size H	40	140
Output Layer Size	16×2	80×4
PReLU β	Channel-shared	
Learning Rate	0.001	
Batch Size	100	50
Epochs	400	1000

Fig. 6 shows the MSE and CS while testing the proposed network under different activation functions. Although using the hyperbolic tangent function leads to a model with the lowest MSE and highest CS, it imposes the highest computational complexity as reported in Table I. Hence, PReLU activation function has been adopted for the hidden layer as a trade-off between complexity and MSE/CS.

Fig. 7 shows the MSE and CS of the model with best achieved performance. The latter is obtained using the characteristics presented in Table II. One noticeable observation is that the size of the hidden layer differs for the two input dimensions. This is due to the fact that the network has to output 320 elements (80×4) for the input dimension (4×80)

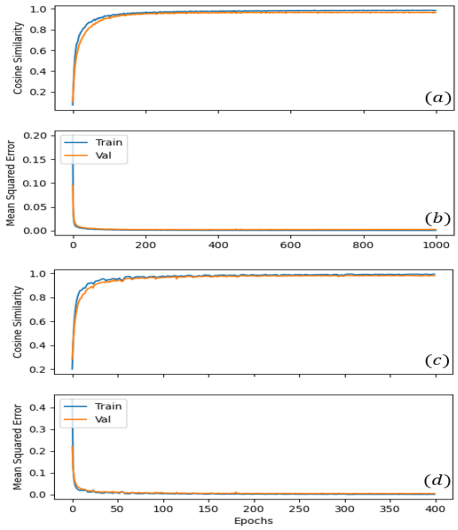


Fig. 7. Best Model Performance: (a) CS for $V(80 \times 4)$, (b) MSE for $V(80 \times 4)$, (a) CS for $V(16 \times 2)$, (b) MSE for $V(16 \times 2)$.

TABLE III
BEST MODEL PERFORMANCE COMPARED TO ONE-SIDED JACOBI

Output Layer Size	16×2	80×4
Training MSE	9.45×10^{-4}	3.7×10^{-4}
Training CS	0.998	0.989
Validation MSE	9.63×10^{-4}	4.02×10^{-4}
Validation CS	0.982	0.969
Testing MSE	9.7×10^{-4}	4.18×10^{-4}
Testing MSE based on [20]	9.21×10^{-4}	3.88×10^{-4}
Testing CS	0.971	0.957
Testing CS based on [20]	≈ 1	≈ 1

compared to 32 elements (16×2) for the input dimension (20×16), which justifies the longer training time required (higher number of epochs). However, the training can be shortened into 250 and 100 epochs for output dimensions (80×4) and (16×2) respectively.

The obtained performance is compared to that of computing the SVD using the one-sided Jacobi algorithm based on the architecture presented in [20]. According to the comparison shown in Table III, the proposed neural network is capable of computing the right singular vectors V while: (1) providing low MSE and high CS during training, validation, and testing, and (2) achieving comparable performance in terms of MSE and CS to the exact computation using the one-sided Jacobi. This is evident for both input dimensions 4×80 and 20×16.

V. HARDWARE IMPLEMENTATION AND VERIFICATION

This section presents the architecture and implementation details of the two shallow neural networks and the overall tensorial SVM. The latter is characterized by adopting these networks for SVD computation. The input, weights, and biases are represented in 32-bit floating point. Each hardware architecture has been coded in C++, synthesized and implemented using Vivado/Vivado HLS 2020.1 targeting Virtex-7 FPGA device operating at 100 MHz. To test and validate the hardware implementation, a C++/RTL co-simulation is performed in Vivado HLS to compare the results between the C++ simulation and the RTL implementation. Afterwards, the RTL implementation has been exported as an Intellectual Property (IP) to Vivado where the hardware resources and number of clock cycles are recorded. The time latency is computed as:

$$T = cc \times 1/f_{max} \tag{21}$$

where cc is the number of clock cycles in post-implementation timing simulation and f_{max} is the maximum operating frequency. As for power consumption, a post implementation functional and timing behavioral simulation is performed to generate a Switching Activity Interchange File (SAIF). This file is used to obtain a vector-based power estimation post-routing.

For the rest of the paper, let NN1 and NN2 denote the neural networks with input dimensions 4×80 and 20×16 respectively.

A. FPGA Implementation of the Shallow Neural Network

Fig. 8 shows the architecture of the proposed shallow neural network. For an input X of size L (one of the unfolded matrices), it outputs the V matrix using sequential operations. The outputs Y_h and Y_O corresponds to the equations (5) and (6), where f_h and f_O are the PReLU and the hard tangent hyperbolic activation functions respectively. The input X and the weights are stored on-chip using BRAMs and the multiplier is fed from the BRAM to perform element-by-element multiplication of the input and weight values. Similarly, the multiplication result is fed to the adder and the bias values are read from on-chip BRAMs. The right singular vectors matrix V is obtained by transforming the output vector Y_O into a 2D array as shown in Fig. 9. The advantage of such architecture is that it allows the use of network pruning without any loss in performance (MSE/CS). The weight and bias matrices obtained from the offline training phase have been analyzed to identify neurons with very low weight/bias values. These neurons could be removed without affecting the network performance during inference. Thus, pruning is applied on matrix multiplication/addition by skipping operations where $W[i], b[i] \leq 10^{-4}$.

Table IV shows the implementation details for the SVD computation on a $4 \times 4 \times 20$ input tensor (i.e. NN1 is utilized twice to compute the SVD of the matrices M, N while NN2 is utilized once to compute the SVD of the matrix P) compared to the one-sided Jacobi based on the architecture presented in [3]. The obtained results show that using neural networks for SVD computations allows for a $324\times$ speedup with an average resources and power reductions of 58% and 67% respectively. Another observation is that the neural network architecture uses slightly more BRAMs. This is due to the fact

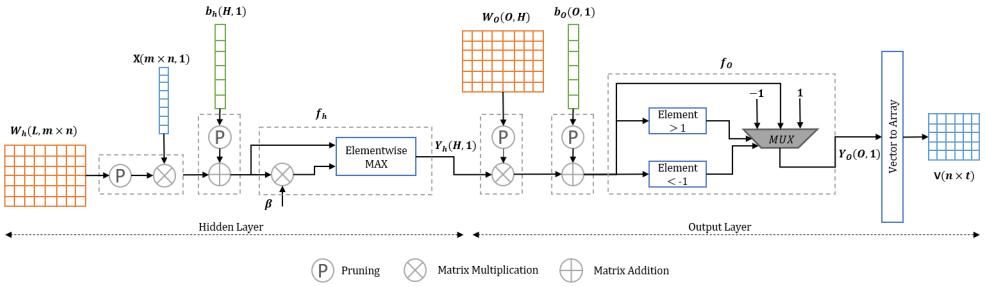


Fig. 8. Shallow Neural Network Architecture.

TABLE IV
IMPLEMENTATION RESULTS FOR TENSOR SVD COMPUTATIONS

Architecture	Neural Network	one-sided Jacobi
BRAM	102	88
DSP	32	105
FF	3714	29277
LUT	4905	43258
Time Latency	14.5 ms	4.7 s
Power Consumption	0.45W	1.35W

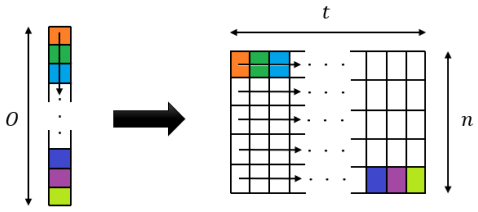


Fig. 9. Vector Y_O to Array V Transformation.

that the weights and biases matrices obtained from network training are mapped into BRAMs and are not saved on an external memory. Knowing that the Virtex-7 FPGA is used for implementation to have a credible comparison with the state-of-the-art, the obtained results show that the proposed neural network for SVD computations is adequate to fit in a resource-limited platform such as the Zynqberry. This is not possible for the implementation of the one-sided Jacobi targeting large matrix dimensions.

B. FPGA Implementation of the Neural Network Based SVM

The neural networks NN1 and NN2 have been embedded into the cascade architecture of the tensorial SVM presented in [3]. The new NN-based TSVM architecture is presented in Fig. 10. The "NN Memory" contains the weights and biases matrices of the designed neural networks. The "SVM Memory" contains the singular vector training matrices. k_1, k_2, and k_3 are the three kernel factors obtained using (3). The architecture performs the SVD computation of the three unfolded matrices using the proposed NN1 and NN2 neural networks. Table V shows the different operating modes in the cascade architecture. For $S_0S_1 = 00$, the first unfolded matrix X_1 is selected and NN1 is activated, then for $S_0S_1 = 01$, the second unfolded matrix X_2 is selected and NN1 is utilized. As for $S_0S_1 = 10$, the third unfolded matrix X_3 is selected and NN2 is activated. When active, each network computes the right singular vector matrix V of each of the unfolded input matrices. The obtained V matrices along with the ones exported from the training phase are used to compute the kernel factors as depicted in (3), which are required to output a classification decision as shown in (4).

TABLE V
NN-BASED TSVM OPERATING MODES

Control		MUX	Active
S_0	S_1	Output	Network
0	0	X_1	NN1
0	1	X_2	NN1
1	0	X_3	NN2
1	1	-	-

TABLE VI
IMPLEMENTATION RESULTS FOR TENSORIAL SVM

Architecture	NN-TSVM	Jacobi-TSVM
BRAM	105 (7.14%)	91 (6.19%)
DSP	133 (3.7%)	206 (5.72%)
FF	11975 (1.38%)	39047 (4.51%)
LUT	20427 4.7%)	60100 (13.87%)
Time Latency (ms)	36	4730
Energy per classification (mJ)	6.28	600

Table VI presents the implementation details of both the NN-based TSVM and Jacobi-based SVM for $N_t = 200$ and $N_c = 2$. The energy per classification is computed as $E = P \times T$ where P is the dynamic power consumption and T is the time latency. The NN-based TSVM and Jacobi-based TSVM recorded 0.9 W and 1.8 W respectively. Results show that replacing the one-sided Jacobi algorithm with a shallow neural network in the architecture of the TSVM leads to faster classification time up to 131×. The NN-based TSVM also requires 39% less average hardware resources with 50% reduced power consumption. This leads

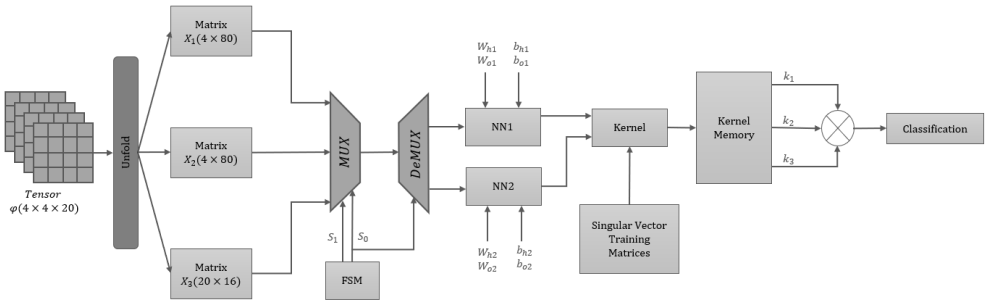


Fig. 10. Neural Network based SVM Cascade Architecture.

TABLE VII
TOUCH MODALITY CLASSIFICATION USING NN-TSVM IN COMPARISON WITH EXISTING METHODS

Problem	Classification Accuracy (%)							
	NN-TSVM	Jacobi TSVM [14]	RLS [14]	k-NN (k=3) [37]	DCNN [38]	LSTM [39]	GRU [39]	TSVM-IRCK [40]
A	90	90	90	68	NA	NA	NA	NA
B	83.3	86.3	89.5	64				
C	80	83.3	75.5	89.6				
D	71	71	73.3	NA	76.9 (Inception Resnet)	74.51 (10 neurons)	73.4 (8 neurons)	77

to 88% reductions in the energy per classification factor. Three main observations could be noted: the proposed NN-based TSVM (1) is capable of real-time classification within 36 ms ($time \leq 400ms$ [4]), (2) achieves real-time classification using cascaded architecture, which was not possible using the Jacobi-based TSVM as reported in [3]. The latter has been the main reason for using the parallel architecture which has led to high power consumption, (3) offers the reductions in resources and energy per classification at the expense of increased memory requirements to store the weights and biases matrices compared to Jacobi-based TSVM.

C. Performance Verification

The NN-based TSVM implementation is verified using the four classification problems mentioned in Section II.B. Table VII presents the classification accuracy achieved by the proposed NN-TSVM in comparison with existing methods targeting the same touch modality classification. The classification accuracy of different methods is tested using a dataset with 30 samples. Using neural networks to compute the right singular vectors V provides approximate values compared to the exact one-sided Jacobi. However, this resulted in acceptable classification accuracy with only 3% loss in the worst case. This is evident in the comparable MSE/CS of both architectures as presented in Table III.

Compared to other methods, for binary classification (A, B, and C), the proposed NN-TSVM shows a worst case of 6% loss compared to RLS for Problem B and a 5% better accuracy for Problem C, and 9.6% loss compared to kNN for Problem C while providing up to 20% accuracy increase in Problems A and B. Problem C has been identified as very challenging for TSVM in [14], it has been solved in [37] using k-Nearest Neighbor (kNN). For multiclass classification (Problem D), NN-TSVM achieves an accuracy comparable to Jacobi-TSVM and RLS, with a 7% worst case loss compared to Deep Convolutional Neural Network (DCNN) and TSVN with Ideal Regularized Composite kernel (TSVM-IRCK).

VI. SCALABILITY OF NEURAL NETWORK BASED TSVM

In order to quantify the scalability of the NN-based TSVM hardware complexity (resources and time latency), two cases are assessed: (1) Scalability of the shallow neural network, and (2) Scalability of the NN-based TSVM. The former is studied by varying the hidden/output layer size and tuning the network to achieve the same MSE/CS reported in Table III. The latter is performed by increasing the number of training tensors while maintaining the overall classification accuracy of the NN-based TSVM as reported in Table VII.

A. Case 1

The scalability of the neural network depends on the size of each layer and the activation function in use. Through Fig. 3, an insight about the number of operations with respect to the dimensions (i.e. m, n, and H) could be learned for a certain application. To assess the scalability of the proposed NN architecture, the hidden and output layer sizes are varied. L is chosen so that the network maintains the same MSE/CS reported in Table III. O is derived from the dimension of the V matrix of the unfolded matrices obtained from the input tensor $4 \times 4 \times 20$. We designed and implemented two more neural networks that compute the right singular vectors V without truncation i.e. output layer size $O = n \times n$ instead of $O = n \times t$. Fig. 11 presents the hardware resources and the time latency with respect to hidden and output layer sizes for

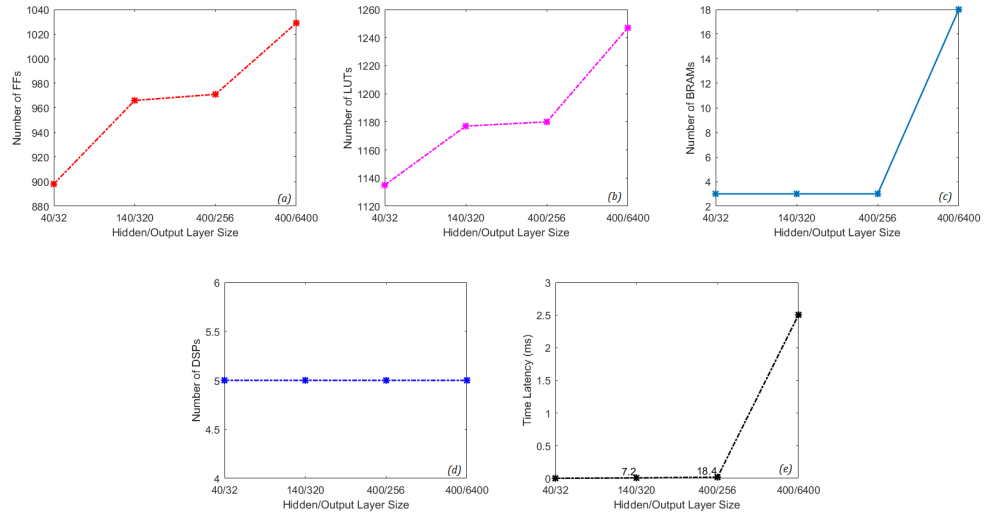


Fig. 11. Scalability of Shallow Neural Network for varying the hidden/output layers size.

a three-layer (input, hidden, output) shallow neural network for the SVD computation of an input matrix. The obtained results are recorded when the network achieved a comparable MSE/CS to those reported in Table III. Analyzing the graphs leads to several observations:

- The number of required FFs and LUTs is not uniform (see Fig. 11(a),(b)). For instance, a similar number of FFs/LUTs is required for networks with 140 and 400 neurons in the hidden layer with the same output layer size. This could be justified with the pruned cascaded architecture where resources are shared for blocks with similar functionality.
- Memory requirements in terms of BRAMs starts to increase once reached an output layer size of 80×80 with 400 neurons in the hidden layer (see Fig. 11(c)). This is justified since the sizes of the weight and the bias matrices increase in such cases, which requires more memory storage.
- As shown in Fig. 11(d), regardless of the input/hidden/output layer size of the network, the number of DSPs is constant for the proposed architecture.
- The SVD computation time is relatively short until reaching a high output layer size as shown in Fig. 11(e). This is due to the longer operations required to perform matrix multiplication/addition. However, according to the comparison in Section III.B, this is faster than using the one-sided Jacobi as long as $H \leq 70,000$ ($H \leq 800,000$) for 20×16 (4×80) matrices.

The presented scalability assessment supports the use of these networks for SVD computations as an efficient solution especially for large matrix dimensions. Hence, the proposed idea could be extended into other applications via a two-stage approach as shown in Fig. 12:

- *Stage 1:* Unfold all the tensors ϕ_i in a dataset into 3 matrices. Then, find the V matrix for each of the unfolded matrices using MATLAB or other software. For the majority of the applications, a tensor has the same first two dimensions (e.g. image, touch modality) hence, two of the generated matrices will have the same dimension hence can be grouped in a subset A. The remaining matrix and its corresponding V matrix will be added to a subset B.
- *Stage 2:* For each of the subsets, a shallow neural network is to be designed. Start with random hyperparameters for the initial model, then tune it using the generated subset to reach the required MSE and CS. Once, the best model is found, the weights and biases matrices could be exported and used by the architecture in Fig. 8. For complexity tuning, one could modify the pruning rule while preserving the required performance metric imposed by the application.

B. Case 2

To study the scalability of the proposed NN-based TSVM, the number of training tensors has been varied between 200 and 900 and the implementation requirements are recorded once the NN-based TSVM recorded a comparable accuracy to the one presented in Table VII. According to the results obtained in Fig. 13:

- The required hardware resources (FFs, LUTs, BRAMs) are slightly increased with the increase of the number of training tensors. In case of BRAMs, a steeper slope is

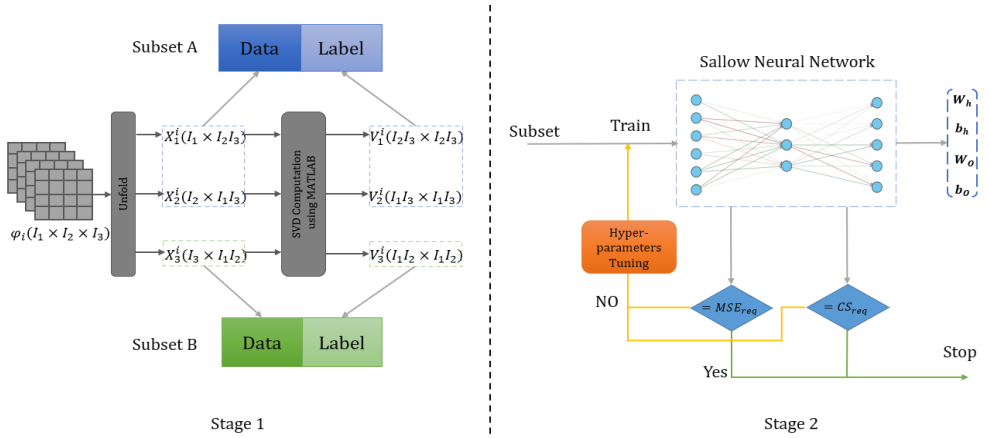


Fig. 12. SVD Computation Approach via Shallow Neural Network.

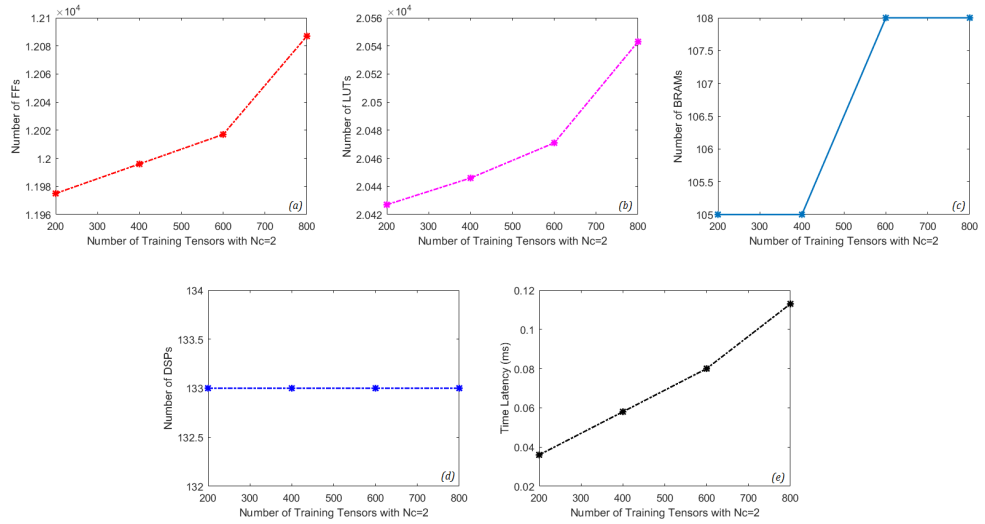


Fig. 13. Scalability of NN-based TSVM for binary classification (Nc = 2) and variable number of training tensors.

observed which is due to the adoption of NN that requires the storage of weights and biases matrices.

- The number of required DSPs is contsant for each size of training tensors.
- The proposed implementation is capable of real-time classification even after 4.5× increase in the number of training tensors.

Compared to the scalability study of the Jacobi-based TSVM presented in [41], Fig. 14 shows that the proposed approach complexity versus the number of training tensors presents a reduced slope. For instance, the Jacobi-based TSVM requires 29% increase in the number of FFs when the number of training tensors is doubled. Using the NN-based TSVM, an increase of less than 1% in FFs is noticed. This is mainly due to two reasons: (1) the neural network requires significantly less resources than that of the one-sided Jacobi. (2) the NN-based TSVM is a cascaded implementation i.e. blocks are being re-used for implementation while increasing the time latency. In [41], the architecture is based on parallel computation due to their time constraint of

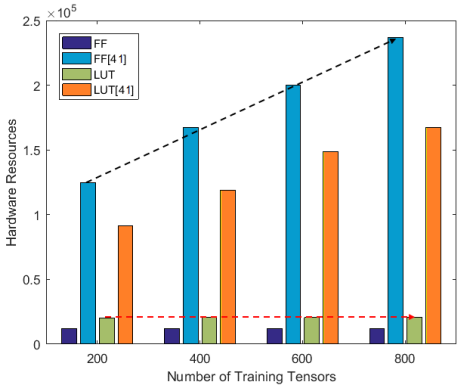


Fig. 14. Scalability Comparison with Existing Methods [41].

real-time classification. The latter is assured using the proposed cascaded architecture for all of training tensors sizes.

The importance of the presented work lies in the ability to scale such architecture for processing larger number of samples while respecting the constraints of the application. When scaled up, the designed NN-TSVM could enable intelligence on smaller platforms (e.g. Zynqberry) if two issues are tackled. The first issue is reducing the number of DSPs: this could be achieved by using some approximate computing techniques [42] or using LUTs-only custom core for matrix operations. The second issue is reducing the number of BRAMs: this could be achieved by further pruning of the weight/bias matrices as long as the application performance is not highly affected. Another method is to offload these matrices completely to external DRAM. This imposes additional timing overhead. However, authors in [43] have presented a strategy to overcome such design challenge.

VII. Conclusion

This paper introduced a shallow neural network architecture for the SVD computation of tensorial inputs. The architecture achieves comparable performance to the state-of-art solutions while imposing significant reductions in the implementation requirements. Once embedded in the SVM architecture, the NN-based TSVM is capable of delivering faster touch modality classification time up to 131× using a cascade architecture. The latter is characterized by a 39% and 88% decrease in the resources and energy per classification respectively compared to the architecture presented in [3] targeting the same application. Moreover, the proposed NN-based SVM obeys the constraints imposed by the tactile data processing application e.g. small size, real-time response, and low power consumption. The encouraging scalability results present the first effective trial for designing an efficient embedded processing unit for an e-skin. A unit that is capable of delivering real-time performance with relatively acceptable power consumption without the need for high performance platform or multi-core devices.

References

[1] M. Signoretto, L. De Lathauwer, and J. A. K. Suykens, "A kernel-based framework to tensorial data analysis," *Neural Netw.*, vol. 24, no. 8, pp. 861–874, Oct. 2011.
[2] P. Gastaldo, L. Pinna, L. Seminara, M. Valle, and R. Zunino, "A tensor-based pattern-recognition framework for the interpretation of touch modality in artificial skin systems," *IEEE Sensors J.*, vol. 14, no. 7, pp. 2216–2225, Jul. 2014.
[3] A. Ibrahim and M. Valle, "Real-time embedded machine learning for tensorial tactile data processing," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 65, no. 11, pp. 3897–3906, Nov. 2018.
[4] P. P. Lele, D. C. Sinclair, and G. Weddell, "The reaction time to touch," *J. Physiol.*, vol. 123, no. 1, pp. 187–203, Jan. 1954.
[5] H. Fares *et al.*, "Distributed sensing and stimulation systems for sense of touch restoration in prosthetics," in *Proc. New Gener. CAS (NGCAS)*, Genova, Italy, Sep. 2017, pp. 177–180.
[6] B. B. Zhou, R. P. Brent, and M. Kahn, "Efficient one-sided Jacobi algorithms for singular value decomposition and the symmetric eigenproblem," in *Proc. 1st Int. Conf. Algorithms Archit. Parallel Process.*, vol. 1, Brisbane, QLD, Australia, Apr. 1995, pp. 256–262.
[7] N. Samardzija and R. L. Waterland, "A neural network for computing eigenvectors and eigenvalues," *Biol. Cybern.*, vol. 65, no. 4, pp. 211–214, Aug. 1991.
[8] Z. Yi, Y. Fu, and H. J. Tang, "Neural networks based approach for computing eigenvectors and eigenvalues of symmetric matrix," *Comput. Math. Appl.*, vol. 47, nos. 8–9, pp. 1155–1164, Apr. 2004.
[9] Y. Tang and J. Li, "Another neural network based approach for computing eigenvalues and eigenvectors of real skew-symmetric matrices," *Comput. Math. Appl.*, vol. 60, no. 5, pp. 1385–1392, Sep. 2010.
[10] J. Qiu, H. Wang, J. Lu, B. Zhang, and K.-L. Du, "Neural network implementations for PCA and its extensions," *Artif. Intell.*, vol. 2012, pp. 1–19, Oct. 2012.
[11] Z. Li, W. Yang, S. Peng, and F. Liu, "A survey of convolutional neural networks: Analysis, applications, and prospects," 2020, *arXiv:2004.02806*. [Online]. Available: http://arxiv.org/abs/2004.02806
[12] M. P. Véstias, "A survey of convolutional neural networks on edge with reconfigurable computing," *Algorithms*, vol. 12, p. 154, Jul. 2019.
[13] *TE0726 Resources—Public Docs—Trenz Electronic Wiki*, Trenz Electron. GmbH, Poing, Germany, 2017.
[14] P. Gastaldo, L. Pinna, L. Seminara, M. Valle, and R. Zunino, "Computational intelligence techniques for tactile sensing systems," *Sensors*, vol. 14, no. 6, pp. 10952–10976, Jun. 2014.
[15] B. Yang and J. F. Böhme, "Reducing the computations of the singular value decomposition array given by Brent and Luk," *SIAM J. Matrix Anal. Appl.*, vol. 12, no. 4, pp. 713–725, 1991.
[16] R. P. Brent and F. T. Luk, "The solution of singular-value and symmetric eigenvalue problems on multiprocessor arrays," *SIAM J. Sci. Stat. Comput.*, vol. 6, no. 1, pp. 69–84, 1985.
[17] J. R. Cavallaro and F. T. Luk, "Architectures for a cordic SVD processor," in *Proc. SPIE*, San Diego, CA, USA, Mar. 1986, p. 45.
[18] J.-M. Delosme, "CORDIC algorithms: Theory and extensions," in *Proc. SPIE*, San Diego, CA, USA, Nov. 1989, p. 131.
[19] D. Milford and M. Sandell, "Singular value decomposition using an array of CORDIC processors," *Signal Process.*, vol. 102, pp. 163–170, Sep. 2014.
[20] A. Ibrahim, M. Valle, L. Noli, and H. Chible, "Singular value decomposition FPGA implementation for tactile data processing," in *Proc. IEEE 13th Int. New Circuits Syst. Conf. (NEWCAS)*, Grenoble, France, Jun. 2015, pp. 1–4.
[21] S. Zhang, X. Tian, C. Xiong, J. Tian, and D. Ming, "Fast implementation for the singular value and eigenvalue decomposition based on FPGA," *Chin. J. Electron.*, vol. 26, no. 1, pp. 132–136, Jan. 2017.
[22] T. Jiang, F. Xie, S. Yuan, S. Yao, K. Wu, and X. Wang, "Implementation of matrix SVD decomposition module for subspace channel estimation," in *Proc. IEEE 3rd Inf. Technol., Netw., Electron. Autom. Control Conf. (ITNEC)*, Chengdu, China, Mar. 2019, pp. 1096–1102.
[23] C. Deng, M. Yin, X.-Y. Liu, X. Wang, and B. Yuan, "High-performance hardware architecture for tensor singular value decomposition: Invited paper," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD)*, Westminster, CO, USA, Nov. 2019, pp. 1–6.
[24] A. Ibrahim, M. Valle, L. Noli, and H. Chible, "FPGA implementation of fixed point CORDIC-SVD for E-skin systems," in *Proc. 11th Conf. Res. Microelectron. Electron. (PRIME)*, Glasgow, U.K., Jun. 2015, pp. 318–321.

[25] Y. Wang, J.-J. Lee, Y. Ding, and P. Li, "A scalable FPGA engine for parallel acceleration of singular value decomposition," in *Proc. 21st Int. Symp. Qual. Electron. Design (ISQED)*, Santa Clara, CA, USA, Mar. 2020, pp. 370–376.

[26] H. Bui and S. Tahar, "Design and synthesis of an IEEE-754 exponential function," in *Proc. Eng. Solutions Next Millennium*, vol. 1, Edmonton, AB, Canada, May 1999, pp. 450–455.

[27] M. Osta, M. Alameh, H. Younes, A. Ibrahim, and M. Valle, "Energy efficient implementation of machine learning algorithms on hardware platforms," in *Proc. 26th IEEE Int. Conf. Electron., Circuits Syst. (ICECS)*, Genova, Italy, Nov. 2019, pp. 21–24.

[28] I. L. Jernelv, D. R. Hjelme, Y. Matsuura, and A. Aksnes, "Convolutional neural networks for classification and regression analysis of one-dimensional spectral data," 2020, *arXiv:2005.07530*. [Online]. Available: http://arxiv.org/abs/2005.07530

[29] M. Fernández-Delgado, M. S. Sirsat, E. Cernadas, S. Alawadi, S. Barro, and M. Febrero-Bande, "An extensive experimental survey of regression methods," *Neural Netw.*, vol. 111, pp. 11–34, Mar. 2019.

[30] A. G. Howard *et al.*, "MobileNets: Efficient convolutional neural networks for mobile vision applications," 2017, *arXiv:1704.04861*. [Online]. Available: http://arxiv.org/abs/1704.04861

[31] X. Zhang, X. Zhou, M. Lin, and J. Sun, "ShuffleNet: An extremely efficient convolutional neural network for mobile devices," 2017, *arXiv:1707.01083*. [Online]. Available: http://arxiv.org/abs/1707.01083

[32] I. Freeman, L. Roese-Koerner, and A. Kummert, "EffNet: An efficient structure for convolutional neural networks," 2018, *arXiv:1801.06434*. [Online]. Available: http://arxiv.org/abs/1801.06434

[33] M. Osta *et al.*, "An energy efficient system for touch modality classification in electronic skin applications," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, Sapporo, Japan, May 2019, pp. 1–4.

[34] C. Gulcehre, M. Moczulski, M. Denil, and Y. Bengio, "Noisy activation functions," 2016, *arXiv:1603.00391*. [Online]. Available: http://arxiv.org/abs/1603.00391

[35] L. Yang and A. Shami, "On hyperparameter optimization of machine learning algorithms: Theory and practice," *Neurocomputing*, vol. 415, pp. 295–316, Nov. 2020.

[36] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification," 2015, *arXiv:1502.01852*. [Online]. Available: http://arxiv.org/abs/1502.01852

[37] H. Younes, A. Ibrahim, M. Rizk, and M. Valle, "Data oriented approximate K-nearest neighbor classifier for touch modality recognition," in *Proc. 15th Conf. Res. Microelectron. Electron. (PRIME)*, Lausanne, Switzerland, Jul. 2019, pp. 241–244.

[38] M. Alameh, A. Ibrahim, M. Valle, and G. Moser, "DCNN for tactile sensory data classification based on transfer learning," in *Proc. 15th Conf. Ph.D Res. Microelectron. Electron. (PRIME)*, Lausanne, Switzerland, Jul. 2019, pp. 237–240.

[39] M. Alameh, Y. Abbass, A. Ibrahim, G. Moser, and M. Valle, "Touch modality classification using recurrent neural networks," *IEEE Sensors J.*, vol. 21, pp. 9983–9993, Apr. 2021.

[40] Z. Yi, T. Xu, W. Shang, and X. Wu, "Touch modality identification with tensorial tactile signals: A kernel-based approach," *IEEE Trans. Autom. Sci. Eng.*, early access, Feb. 9, 2021, doi: 10.1109/TASE.2021.3055251.

[41] A. Ibrahim, P. Gastaldo, H. Chible, and M. Valle, "Real-time digital signal processing based on FPGAs for electronic skin implementation," *Sensors*, vol. 17, no. 3, p. 558, Mar. 2017.

[42] S. Mittal, "A survey of techniques for approximate computing," *ACM Comput. Surv.*, vol. 48, no. 4, pp. 1–33, 2016.

[43] J. Vieira, R. P. Duarte, and H. C. Neto, "KNN-STUFF: KNN STreaming unit for FPGAs," *IEEE Access*, vol. 7, pp. 170864–170877, 2019.

Hamoud Younes (Graduate Student Member, IEEE) received the B.S. and M.S. degrees in computer and communication engineering from Lebanese International University in 2012 and 2015, respectively. He is currently pursuing the Ph.D. degree with the Department of Electric, Electronic, Telecommunication Engineering and Naval Architecture, University of Genoa. His research interests include embedded electronic systems, FPGA implementation, embedded machine and deep learning, approximate computing, and energy efficient embedded computing.

Ali Ibrahim (Member, IEEE) received the M.S. degree in industrial control from the Doctoral School of Sciences and Technologies, Lebanese University, in 2009, and the dual Ph.D. degrees in electronic and computer engineering and robotics and telecommunications from the University of Genova and Lebanese University in 2016. From 2016 to 2018, he was a Post-Doctoral Researcher with the Department of Electric, Electronic, Telecommunication Engineering and Naval Architecture, University of Genova. He is currently an Assistant Professor with the Department of Electrical and Electronics Engineering, Lebanese International University, Lebanon, and also an Associate Researcher with the Department of Electric, Electronic, Telecommunication Engineering and Naval Architecture, University of Genoa. His research interests include embedded machine learning, FPGA implementation, and interface electronics for electronic skin systems, approximate computing, and techniques and methods for energy efficient embedded computing.

Mostafa Rizk (Member, IEEE) received the Maitrise degree in electronics, the M.Sc. degree in biomedical physics, and the M.Sc. degree in signal, telecom, image, and speech from Lebanese University, Beirut, Lebanon, in 2007, 2008, and 2010, respectively, the Ph.D. degree in sciences and technologies of information from Telecom Bretagne, Brest, France, in 2014, and the Ph.D. degree in electronics and communication from Lebanese University in 2015. He has been a Post-Doctoral Researcher with the University of Southern Brittany, Lorient, France, and with the Lab-STICC Laboratory CNRS, Lorient. He is currently an Assistant Professor with Lebanese International University, Beirut, and an Associate Researcher with IMT Atlantique, Brest. His current research interests include hardware/software implementations and digital circuit design, network-on-chip design, and new MPSoC architectures based on emerging nonvolatile memory technologies.

Maurizio Valle (Senior Member, IEEE) received the M.S. degree in electronic engineering and the Ph.D. degree in electronics and computer science from the University of Genova, Italy, in 1985 and 1990, respectively. Since December 2019, he has been a Full Professor of electronics with the DITEN, University of Genova, where he leads the Connected Objects, Smart Materials, Integrated Circuits–COSMIC Laboratory. He has been and is in charge of many research contracts and projects funded at local, national, and European levels and by Italian and foreign companies. He is the coauthor of more than 200 articles on international scientific journals and conference proceedings. His research interests include bio-medical circuits and systems, electronic/artificial sensitive skin, tactile sensing systems for prosthetics and robotics, neuromorphic touch sensors, and electronic and microelectronic systems. He is a member of the IEEE CAS Society.

7.4.4 A hybrid precision architecture for an efficient binary convolutional neural network accelerator

Convolutional neural networks are a promising solution in many application domains such as Internet of Things (IoT), image processing, tactile processing, etc. However, the computational complexity and memory requirements are the main challenge in the deployment of CNNs on resource-limited devices for energy-constrained applications [188]. For instance, the VGG-16 network contains about 140 million 32-bit floating-point parameters and implements 1.6×10^{10} arithmetic operations [189]. There have been numerous efforts on the complexity reduction of CNNs such as network pruning [190], knowledge distillation [191], and weight quantization [192].

Quantization of CNNs may cause an information loss especially if it is applied to the extreme using 1-bit representation i.e. binarization. To address this issue, a variety of methods have been proposed in recent years [189]. These methods aim to: 1) minimize the quantization error, for instance by only quantizing the weights, 2) improve the network loss function to adapt to the binary values propagating through the network, and 3) reduce the gradient error by the adjustment of the Back Propagation (BP) training algorithm to adapt with binarization functions. Among these methods, minimizing the quantization error is the most used technique since it leads to relevant memory saving and complexity reductions [189, 192, 193].

In this research work, a new architecture based on hybrid precision representation is proposed as a trade-off between the reliability of CNNs and the low complexity of Binary Convolution Neural Networks (BCNN). The architecture adopts binarization of hidden layers and 32-bit floating-point for the first and last layer with binary weights. A design methodology is provided on how to select the network topology, placement of binarization layers, and training process. The network is designed and trained using Larq framework [194], which is an extension of Tensor-Flow that offers a library to design, train, and deploy quantized/binarized CNNs. The proposed hybrid-precision binary weight network (H-BWN) achieved more than 35% accuracy increase in classifying touch modalities compared to traditional BCNN topology. The H-BWN requires less than 5 KB of storage requirements achieving an efficient architecture that fits in a wide range of microcontrollers (e.g. STM32F0x2). When implemented on Zynq-7010 platform, the H-BWN accelerator provided a real-time classification within 0.8 ms with a 42.4 μJ energy per classification. Compared to exiting solutions, H-BWN achieved higher classification accuracy with an energy reduction up to 99% accompanied with a speedup up to 6875$\times$.

This research work has been published in the proceeding of the *IEEE International Conference on Ph.D Research in Microelectronics and Electronics (PRIME)* [195] and the proceeding of the *IEEE International Conference on Electronics, Circuits and Systems (ICECS)* [196].

7.4.5 Perspectives

The deployment of machine learning algorithms on embedded devices for applications with a constrained requirements is an active challenge. In this work,two of the main methods that

are widely used to reduce the obstacles faced while overcoming such challenge have been investigated. The first method is the use of approximate computing to reduce the computational complexity of ML algorithms with an acceptable margin of error. The second method is to design custom hardware accelerator architectures optimized for a certain algorithm implemented on a specific hardware platform.

With the continuous adoption of machine learning algorithms in different domains, future work in this research area involves tracking the advancements of such algorithms especially deep learning. For instance, Transformers have been lately investigated for the use of image classification [197, 198]. Which could be possibly adopted for tactile data processing due to the tensorial nature of tactile signals. Consequently, monitoring the new methodologies for the design and implementation of hardware accelerators. In addition, due to the error resilience nature of ML algorithms, new approximate computing techniques are emerging such as adaptive approximate computing [199] and approximate adders with Single LUT delay [200].

Part III

Current Work and Future Directions

Chapter 8

Ongoing Research Works

8.1 Preface

My ongoing research activities focus on applications in the domain of embedded computer vision (CV) and embedded artificial intelligence (AI) with great emphasis on achieving efficient implementations on embedded edge devices with reduced computational resources an power budget. This research track has been initiated during my work at the Lebanese International University and Lebanese University where I have supervised several Master theses that have tackled topics for applications of visual drone navigation, change detection and object detection. Also, I was the manager of a research project that aims to detect humans from areal images during search and rescue mission. Furthermore, I am currently involved in a research project that aims to optimize the detection and classification of marine objects using deep learning techniques on embedded edge devices. The following sections introduce my current research topic, describe the conducted work, present the achieved results and highlight the future work.

8.2 Real-Time Visual SLAM Drone Navigation in GNSS-Denied Regions

The use of unmanned aerial vehicles (UAVs) is expanding in several fields such as commerce, agriculture, scientific research, rescue missions, pollution detection, etc. Initially, UAVs have been developed to be controlled by an on-ground pilot by the means of a remote-control communication system. Currently, UAVs are drawing closer to navigate at high degrees of autonomy, where knowing the location is an essential attribute. Most UAVs exploit global navigation satellite system (GNSS) technology and inertial sensors (INS) to estimate its own geo-spatial positioning. UAVs are augmented with GNSS receivers, which benefit from the received time radio signals transmitted from satellites to calculate the location (longitude, latitude, and altitude). The location estimation based on the INS alone drifts when the GNSS signal drops-off. Hence, in order to navigate autonomously a course accurately, it is essential to ensure the well receiving of radio signals from at least four satellites simultaneously. This

cannot be guaranteed when a UAV mission includes intermediate locations where GNSS signal becomes unreliable, such as urban zones, indoor environments, landscapes covered with forests, valleys surrounded by mountains, etc., or when the radio signals are mitigated by spoofing and gaming operations, which are easily performed especially for GNSS receivers operating on civilian frequencies. Many research works have been conducted to cope with GNSS failure. The technique of localizing a robot in a certain environment and mapping it at the same time is so called Simultaneous Localization and Mapping (SLAM). Several solutions have been demonstrated to apply SLAM adopting INS, ultrasonic and infrared sensors, Laser range finders [201], rotary encoders, inertial sensors, and GNSS. Recently, as computer vision has witnessed rapid development, SLAM methods employing cameras have been actively introduced and referred to as visual SLAM [202]. Several works have discussed the use of visual SLAM for UAV navigation, landing, etc [203, 204, 205].

This research work demonstrates the usage of image stitching method for SLAM to circumvent the limitations of available methods. Image stitching is basically used to form a single panoramic image with high resolution by merging two or more images of the same scene. Originally, image stitching has been wildly used to do the complete mapping of particular region benefiting from digital maps and satellite images. The use of image stitching for SLAM eliminates the need of selecting landmarks and extracting their features. The features of each two consecutive captured images are used to determine the flight distance and bearing angle. Matching the features is performed only between the stitched image and the two consecutive images, which are captured in the same conditions and using the same camera. This reduces the computational complexity, lessens the required memory space and delivers a solution to localize UAVs in unknown paths and cope with landmarks' modifications.

A novel method to localize UAVs in GNSS-denied regions is provided. It adopts Scale Invariant Feature Transform (SIFT) method to detect the features in the captured images. The features extracted from consecutive images are used to stitch the images. Accordingly, the bearing angle and moving distance are estimated. Figure 8.1 shows how SLAM is applied using the proposed method. Figure 8.1(a) and Figure 8.1(b) show two consecutive captured images Im_x and Im_{x+1}. Figure 8.1(c) shows the stitched image $Im_{x,x+1}$. Figure 8.1(d) and Figure 8.1(e) present respectively the matching of the two original images Im_x and Im_{x+1} in the resulting stitched image $Im_{x,x+1}$. In Fig. 8.1(f), the bearing angle is shown between the green line, which represents the original heading, and the red line, which represents the current path. Note that the total required execution time to determine the coordinates is less than the flight duration. In order to check the relevancy of the proposed approach, the location in the meta-data is compared with the achieved results. The comparison shows for the illustrated example an error of 5 meters in the estimated distance and 4.6 degrees at most in the bearing angle as shown in Figure 8.2, where the red line shows the distance and angle based on the proposed method and the yellow line shows the distance and angle based on the meta-data of the frames. Note that the coordinates delivered in the meta-data are provided by a commercial GPS, which has an equivalent user error of 7 meters.

This work has been initiated in the context of a Master's degree project at the Lebanese International University and has been published in *IEEE International Conference on Artificial Intelligence Circuits and Systems (AICAS)* [206]. In order to perform the experiments in the laboratory rather than real flights, a GNSS receiver emulator has been designed and implemented. The emulator has been demonstrated in the *IEEE International Conference on*

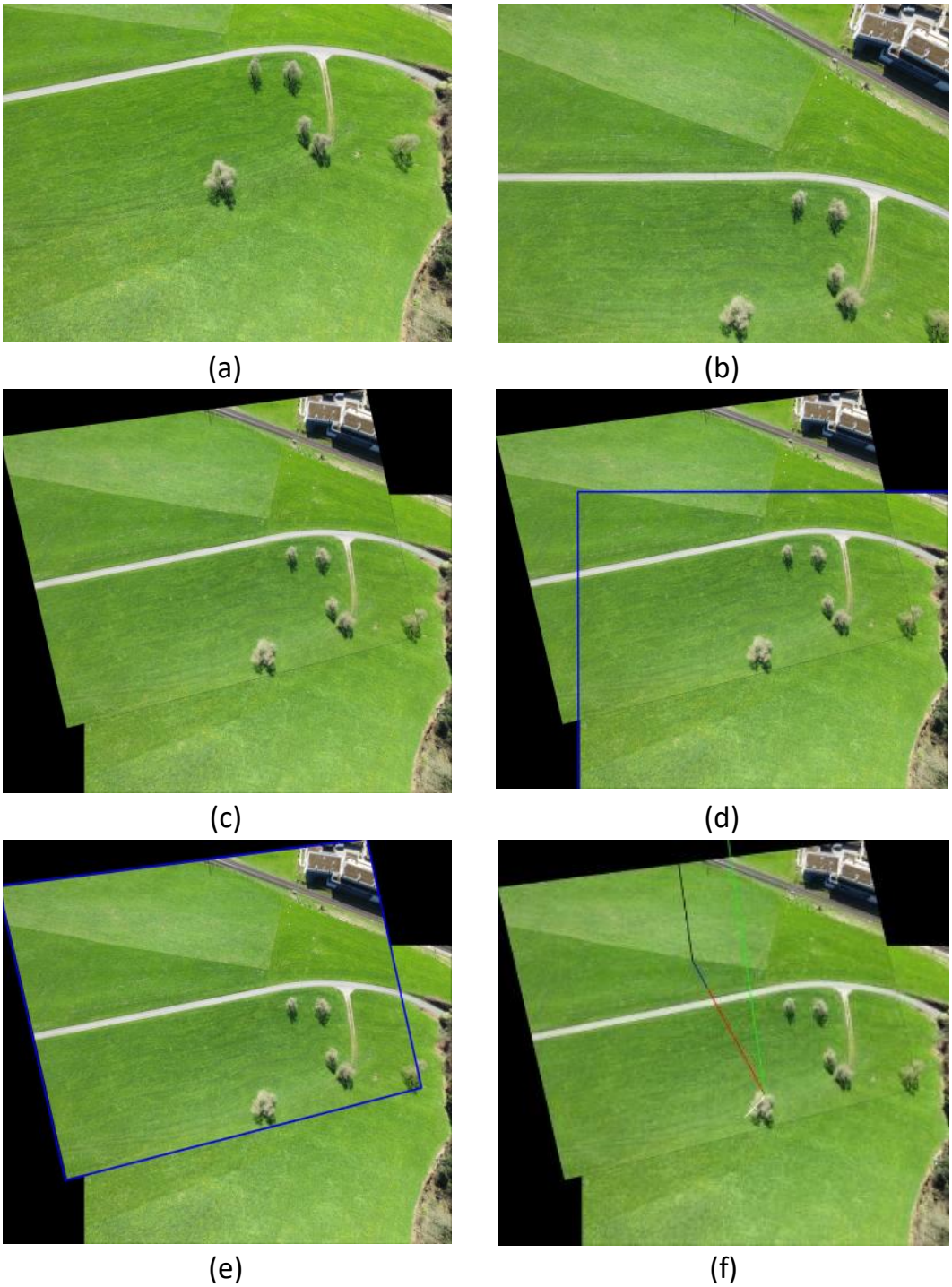


Figure 8.1 — Sample application of the proposed method

Electronics, Circuits and Systems (ICECS) [207].

Future work includes the implementation of the proposed method on edge embedded devices with low weight and reduced size such as FPGA-based platforms or embedded development kits from Nvidia dedicated for AI and computer vision and compatible with the payload constraints of drones such as Jetson Nano, Jetson Xavier NX, and Jetson AGX Xavier. This

Figure 8.2 — Bearing angle comparison results

work includes the phases of implementation and verification in terms of power consumption and performance targeting real-time scenarios. In addition, it is planned to augment the system with a light detection and ranging (LiDAR) to measure the UAV height, which should be involved in the computation during real experiments.

8.3 Marine Objects Detection Using Deep Learning on Embedded Edge Devices

In this work, we address the problem of detecting and classifying maritime objects. Marine object detection and classification are two essential tasks for many applications such as vessel identification and positioning, collision avoidance system, safe autonomous navigation, search and rescue mission, etc. Marine objects can span from stationary floating objects such as buoys to small boats and kayaks and other large vessels such as ferries, passenger ships and cargo ships. Surveillance adopting shared information from Electronic Chart Display and Information System (ECDIS) and GNSS can provide locations of marine vessels [208]. However, this depends on the reliability of data, which may degrade due to spoofing, jamming or even dis-activating automatic identification system (AIS). Radar-based methods can be effective to detect the presence of large vessels. Small boats and floating objects on water surface are difficult to be identified [208].

Visual detection of marine objects using electro-optical sensors provides a solution for detecting and classifying marine objects [209]. Classification and detection of objects using captured images have been widely used in several application domains [210][211]. However, the characteristics of the scenes captured in marine environment arise additional challenges to the task of detecting objects in images or videos compared to other environments. Factors such as dynamic nature of the background, unavailability of static cues, presence of small objects at distant backgrounds and illumination effects impact the performance of commonly used image processing and computer vision approaches [212]. Tides and waves lead to a continuously dynamic background in both spatial and temporal dimensions. Also, floating

objects are subjected to a lot of motion with unpredictable patterns. The illumination of marine scenes varies due to weather conditions (haze, fog, rain, bright sunlight, twilight, etc.). Speckles and glints are mainly induced by the variation of the solar incident angle on water. Furthermore, the disparity of color gamut depends on illumination conditions. Color gamut varies respectively between dark, yellow and red, blue and gray during night, sunset, daylight, and hazy conditions respectively. These factors affect the visibility of objects in marine environments and hence the detection performance. An effective technique for certain cases with specific illumination type, weather condition and water dynamicity may not suit other conditions.

Artificial intelligence techniques based on deep learning provide robust solutions to detect and locate objects. The achieved performance proves the relevance of CNNs in circumventing existing computer vision challenges. Deep learning methods, known as deep neural networks, make use of multiple hidden layers between the input and output layers to learn a hierarchy of features that are invariant to geometric transformations from raw input images. AI methods with CNNs play dominant role in classifying and locating multi objects in images and videos leading to accurate detection. One-stage detection methods have been introduced to provide real-time performance with acceptable precision and accuracy. These methods exclude the stage of pre-selecting the regions of classification and abstract post-processing techniques (refining bounding boxes, eliminating duplicates and adjusting detection scores) used in two-stage methods such as the well known region-convolutional neural network (R-CNN) [213] and its enhanced versions [214, 215] in order to reduce the complexity and ensure real-time detection speed.

You Only Look Once (YOLO) has been recently proposed in [216] as an efficient one-stage CNN-based model that is able to detect multi objects in real-time. Published peer reviewed comparisons [217][208] [218] illustrate that YOLO outperforms two-stage detection methods and other available one-stage methods such as single shot detector (SSD). Since introduced, many versions of YOLO have been introduced such as YOLOv2[219], YOLOv3 [220], YOLOv4 [221] and YOLOv5. In YOLOv2 [219], the fully connected layers at the end have been eliminated and Darknet-19 architecture has been adopted. YOLOv3 [220] uses Darknet-53 architecture and inherits the concept of residual networks. The detections are made at 3 different scales which enables the detection of small objects. YOLOv4 and YOLOv4-tiny proposed initially in 2020 [221] optimize and improve every part of YOLOv3. The main optimization is to use CSPDarknet-53 as its backbone network for extracting features. The difference between YOLOv4-tiny and YOLOv4 is that the tiny version only has two YOLO heads at the end (2 scale factor instead of 3). Figure 8.3 shows the block diagram of YOLOv4 network.

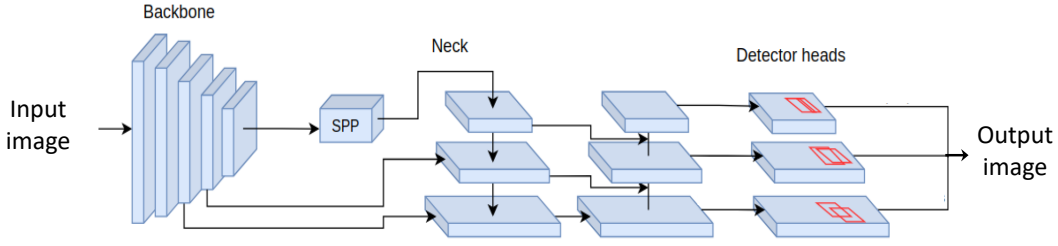


Figure 8.3 — Block diagram of YOLOv4 network

The experiments targeting Microsoft common object in context (COCO) dataset [222] show that YOLOv4 is faster and more accurate than real-time neural networks EfficientDet [223] and RetinaNet [224] provided by Google and Facebook respectively. Comparisons have been made between YOLOv3, YOLOv4 and YOLOv5, in which some authors claim that YOLOv4 is more accurate while others claim that YOLOv5 is more accurate. The reason for different reported results can be attributed to many factors, such as the different datasets used, the modified hyperparameters, etc [225].

The deployment of CNNs on embedded edge devices targeting real-time inference sets a challenge due to the limited computing resources and power budgets. Several optimization techniques such as pruning, quantization and use of light neural networks enable the real-time inference but at the cost of precision degradation. However, using efficient approaches to apply the optimization techniques at training and inference stages enables high inference speed with limited degradation of detection performance.

Accordingly, we investigate different versions of YOLO for real-time object detection and compare their performance for the specific application of detecting maritime objects. The trained YOLO networks are efficiently optimized targeting three recent edge devices: Nvidia Jetson Xavier AGX, AMD-Xilinx Kria KV260 Vision AI Kit, and Movidius Myriad X VPU. Also, we make use of emergent optimization techniques to deploy the trained networks on embedded edge devices. Figure 8.4 shows sample predictions using the trained model.

Figure 8.4 — Sample predictions of marine objects using the trained model

This work comes in the context of ODESSA research project which gathers several researchers and engineers from Lab-STICC and two companies. The performed work and obtained results have been described in a research paper [226] which is submitted to the IEEE International Workshop on Rapid System Prototyping (RSP), which will be held as a part of the Embedded Systems Week (ESWEEK) international conference, Shanghai in October 2022.

The main contributions of this work are:

- Create a diverse dataset of marine objects, which has the widest number of classes and annotations compared to available datasets,
- Train and evaluate several YOLO neural networks with different sizes and architecture specifications,

- Apply structured pruning using sparsifying [227] to reduce the network size while maintaining the detection performance,
- Optimize the trained networks towards implementation on popular edge devices to achieve the best compromise between inference speed and detection performance.

The proposed deployments demonstrate promising results with an inference speed of 90 frames per second (FPS) and a limited degradation of 2.4% in the mean average precision (mAP). The analysis of the obtained results reveals that Jetson Xavier AGX is the most powerful device among the three platforms in terms of detection performance. It achieves the highest inference speed while maintaining acceptable precision values. However, it requires more power budget. It achieves 3.23 FPS/Watt for YOLOv4-tiny and 1.46 FPS/Watt for YOLOv4.

The KRIA KV260 AI VISION KIT power consumption is 4 times less than the GPU on Jetson Xavier AGX. However, the achieved inference speed is 1.6 times and 4 times less than that achieved using the GPU for YOLOv4-tiny and YOLOv4 respectively. It is necessary to realize a graph pruning to achieve 15 FPS with only one DPU B4096@300Mhz. While considering performance per watt criterion only, the KRIA KV260 kit outperforms the other targeted devices when running YOLOv4. It achieves 1.875 FPS/Watt.

The MOVIDIUS MYRIAD X VPU suffers from a lack of computing power, yet it is enough to infer a YOLOv4-tiny at 31 FPS with a power consumption less than 5 Watts with OAK-D camera. The comparison in terms of performance per Watt shows that for YOLOv4-tiny, the VPU outperforms the other two devices as it achieves 7.75 FPS/Watt.

Future work will include the following steps:

- Extend the dataset in order to include new classes of marine objects and to balance the number of images among classes.
- Examine the new version of YOLO so called YOLOv7 and conduct comparisons in terms in detection performance and inference speed.
- Propose and implement an original method to switch dynamically, during run-time, between different scaled neural network models according to environment, power profile or required accuracy and precision.

8.4 Smart-Cameras Network for Multi-View AI-based Object Detection

Implementing CNNs on embedded edge devices with limited computational resources and power budget creates a challenge. Recently, several techniques have been applied such as light networks with fewer layers, pruning to reduce the number of connections, factorization to combine various aspects of the network architecture, and quantization of weight and/or activation values. These techniques ensure better deployment of CNN-based object detection methods on edge devices but at the cost of accuracy and precision drop.

The use of multiple cameras providing different angles of view for the same scene can be used to improve the detection performance leading to the reduction of the number of missing detections or false detections. On the other hand, incorporating multiple camera views for detection alleviates the impacts related to object occlusion, illumination, intra-class variation, similarity between classes, background clutter, ambiguity, etc.

This work makes use of multi-views of the same scene to increase the detection accuracy. It proposes to build and implement a network of smart-cameras, where each consists of a camera connected to an embedded processing unit executing a CNN-based object detection algorithm. Each smart-camera has its own point of view. The distributed smart-cameras collaborate in order to enhance the accuracy and precision of the detection by merging the prediction results of all involved nodes. Figure 8.5 shows an overview of the proposed multi-view smart-camera network.

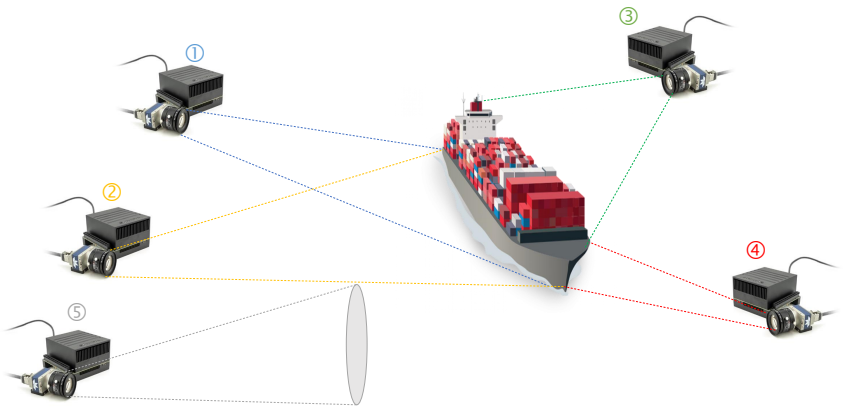


Figure 8.5 — Overview of the proposed multi-view smart-camera network

The main contributions of this work are:

- Create a novel dataset for mutli-view object detection with accurate measurements in terms of orientation and distances by acquiring images with multiple cameras for the same scenes.
- Determine the best hyperparameters of the targeted neural network and conduct training processes accordingly.
- Implement and demonstrate the network of smart cameras.
- Devise and develop an efficient strategy to collect and aggregate prediction data from the nodes of the network in order to reduce the communication overhead and power consumption.
- Propose and implement smart decision-making algorithm to provide the final detection result based on different nodes detections.
- Verify the relevance of the proposed approach by deploying it in the network of smart-cameras with live demonstration and comparing the obtained results with those obtained when multi-view is not adopted.

- Determine the power budget of the proposed system and compare it with the power consumption of single-view system with similar detection performance.

The conducted work has been performed in the 2AI (Algorithm Architecture Interactions) team of the MEE (Mathematical and Electrical Engineering) department at IMT Atlantique, Brest, France in the context of one senior engineering project and two internships. The contributions and achieved results have been valorized in 2 papers [228][229] submitted to the IEEE International Workshop on Rapid System Prototyping (RSP), which will be held as a part of the Embedded Systems Week (ESWEEK) international conference, Shanghai, October 2022.

Future work will tackle the development of the devised approach to consider different aspects such as the location of the smart-cameras, their angles of view and the similarity of objects' occurrences in consecutive video frames in order to enhance the overall detection result. Also, we will investigate the impact of including orientation to the object annotations during training and inference.

In addition, we have initiated a research work that aims to use top-view scenes in order to assist multi-view marine object surveillance using embedded artificial intelligence and computer vision by exploiting the features of top-view scenes. Figure 8.6 illustrates an overview of this proposed topic.

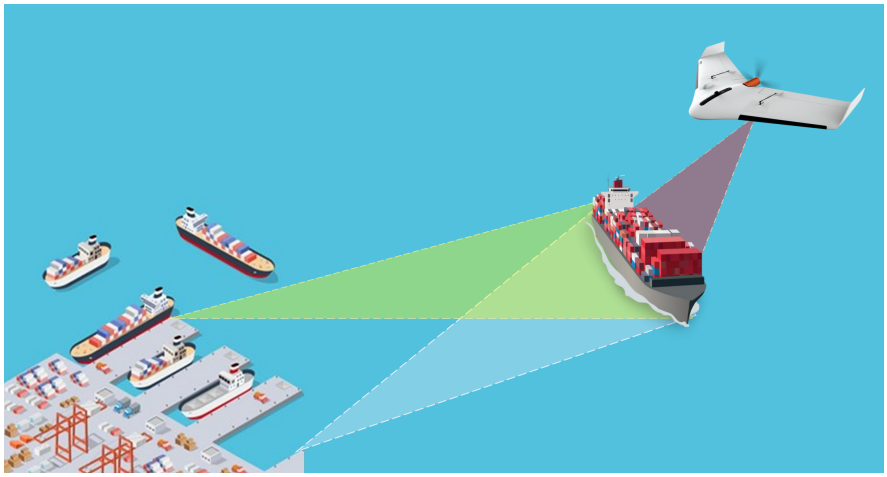


Figure 8.6 — Overview of the proposed top-view detection to assist multi-view marine object surveillance

This work is done in the context of a Masters thesis held at the Lebanese University in collaboration with Lab-STICC and IMT Atlantique. The performed work includes creating a huge dataset that includes images of marine objects captured from top view. We have prepared about 200,000 annotated images taken by the means of drones or satellites. Training is conducted targeting YOLOv4 models. The validation of the trained models show promising results in terms of detection performance. The performed work and obtained results are described in a research paper [230] which is submitted to the IEEE International Conference on Smart Systems and Power Management (IC2SPM), which will be held in Beirut, Lebanon next November 2022.
Future work includes the following tasks:

- Investigate the impact of optimization techniques such as quantization and pruning on detection performance.
- Deploy the optimized networks on emergent edge devices and conduct comparisons in terms of detection performance, inference speed and power consumption.
- Apply the tracking algorithms and examining their corresponding performance on test videos.

8.5 Embedded AI to Assist Search and Rescue Missions

Maritime search and rescue (SAR) missions are crucial for most coastal states. According to the International Organization for Migration 218,062 irregular maritime migration attempts are recorded in the Mediterranean Since 2014 [231]. From which, 23,939 dead and missing persons are recorded during attempted overseas crossings. Furthermore, the European Maritime Safety Agency reports in the Annual Overview of Marine Casualties and Incidents 2021 [232] that during the 2014-2020 period, 367 marine casualties resulted in a total of 550 lives lost and 6921 injuries in the waters of European Union (EU) Member States or involving EU ships. The ability to quickly locate missing people aids in the direction of rescuers and medical personnel, which plays an important role in increasing the chances of saving human lives while also lowering costs.

Years ago, visual surveillance in the maritime domain has been explored. However, most surveillance activities have been assigned to areas near the coasts and ports and mainly depend on human monitoring and analysis for security reasons. Computer vision techniques are also adopted in few works. However, videos and images capturing maritime environment pose challenges that are absent or less severe in other environments such as the dynamic nature of the background, unavailability of static cues, presence of small objects at distant backgrounds and illumination effects [212]. These challenges impact the efficiency of traditional computer vision techniques in detecting individuals in marine environments.

Recently, deep learning approaches have introduced efficient solutions to detect, classify and localize several objects in images and videos. In particular, the evolution of neural networks architectures has elevated the performance to a point that they are considered on par with human performance for some of these problems. The growing use of AI-based detection methods is of great interest in aiding SAR missions [233, 234, 235, 236, 237]. However, only few works have addressed the detection of humans in open water or for man overboard accidents [238, 239, 240]. Other available works adopting deep learning in marine environment have focused mainly on the detection of sea ships [217].

The detection performance of CNN-based methods comes at the cost of increased hardware resources and power consumption especially for real-time scenarios with high requirements of accuracy and precision. YOLO has been recently introduced as an efficient unified model of all phases of a CNN for doing object detection in real-time. The recent version of YOLO, so called YOLOv4, has been justified to detect objects in real-time with high level of precision. Several models of YOLOv4 exist, with different architecture specifications and consequently different detection performance in terms of accuracy and precision, detection

speed and required energy budget. In this work, the relevance of the YOLO in detecting humans in maritime environment is investigated. The available models of YOLOv4 are trained using a custom dataset. The trained models are evaluated using recognized evaluation parameters. In addition, the trained models are optimized targeting embedded low-power hardware platforms dedicated for AI applications.

The main contributions of this work are:

- Create a novel dataset of humans in marine scenes from various positions and from different perspectives and scales with different backgrounds, resolutions and luminosity.
- Train and validate the target networks using different architecture specifications (number of layers, activation functions, shortcuts, etc.), data augmentation methods and input image resolutions.
- Evaluate the trained models and conduct comparisons in terms of well know metrics (mAP, precision, recall, F1-score and average IOU) [241] in order to find the combination that leads to the best performance.
- Examine the detection performance using several recorded videos. Thee videos show humans in open water from different perspectives, points of view and scale.
- Apply several optimization approaches such as pruning to reduce the number of parameters and quantization of weight and/or activation values in order to ensure better deployment of CNN-based object detection methods on edge devices.
- Deploy the optimized models on popular embedded edge devices (Nvidia Jetson Xavier AGX, AMD-Xilinx Kria KV260 Vision AI Kit, and Movidius Myriad X VPU)

The obtained results show that applying data augmentation enhances the detection performance. The use of higher image resolution enhances the mAP performance but at the cost of reduced inference speed and longer training time. Sample detection results from network testing are shown in Figure 8.7. The figure shows that trained models are able to accurately detect and classify the presence of humans in different maritime environments.

The inference speed of the trained networks is evaluated by running several captured videos, showing humans in open water, targeting embedded platforms with different power modes. Figure 8.8 shows samples of the obtained detection results in captured video sequences. The obtained results show that YOLOv4-tiny can achieve real-time detection of humans in maritime environment with acceptable accuracy and precession. As an example, an inference speed of 90 FPS with mAP of 0.8 can be achieved when running YOLOv4-tiny on Jetson Xavier AGX. Whereas, YOLOv4 can achieve 15 FPS with mAP of 0.859.

In order to achieve real-time inference, optimization techniques are applied targeting three hardware platforms: Nvidia Jetson Xavier AGX, AMD-Xilinx Kria KV260 Vision AI Kit, and Movidius Myriad X VPU. The obtained results show that the optimization targeting Jetson Xavier AGX increases the inference speed while achieving better accuracy for all tested networks. This refers to the optimization process that implements several techniques such as kernal fusion, precision calibration, kernel auto-tuning, dynamic tensor memory and multi-stream execution. YOLOv4 can achieve 20 FPS when FP32 is adopted while attaining

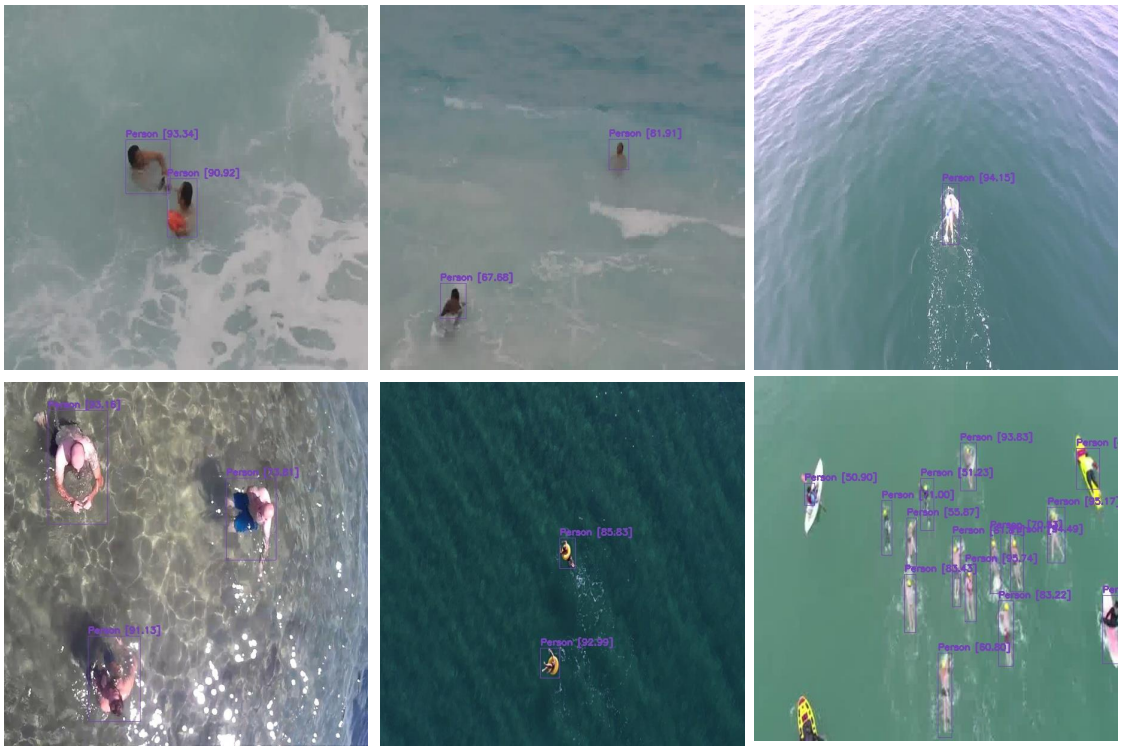


Figure 8.7 — Sample detection results using the testing dataset images

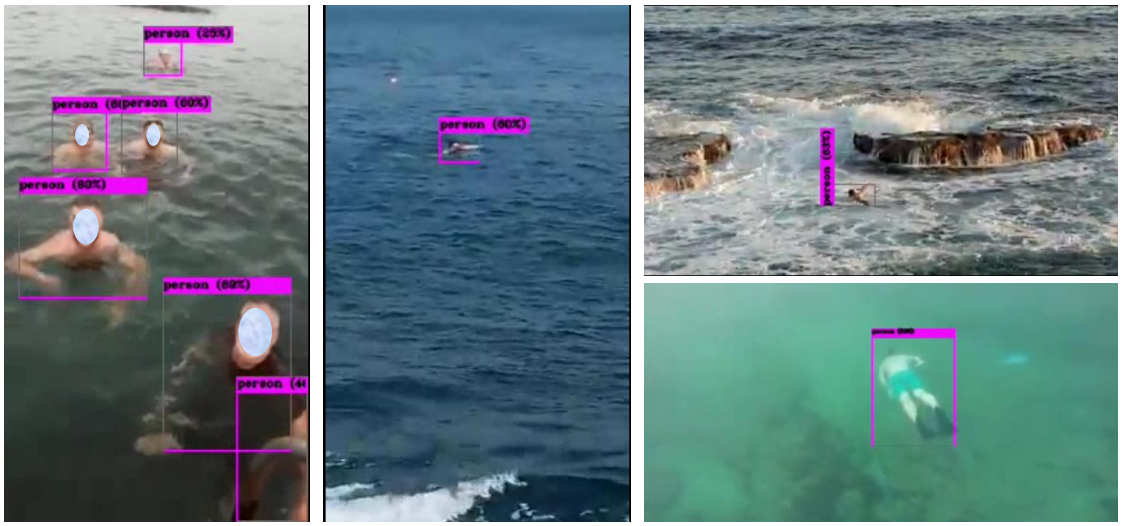


Figure 8.8 — Samples of the obtained detection results using recorded video sequences

a mAP of 0.861 ($+0.2$ points in mAP). The quantization to FP16 leads to better inference rate of 52 FPS ($\times 3.47$) but a cost of degradation in the detection performance (-2.9 points in mAP).

The deployment of trained models on Kria KV260 Vision AI Kit reveals that YOLOv4-tiny can achieve an inference speed of 65 FPS with 0.451 mAP (-2.46 points in mAP) when the network parameters are quantized to INT8, which is the only supported representation. The YOLOv4 model can achieve an inference speed of 11 FPS with 0.818 mAP.

In order to achieve high-speed inference with high-precision detection, YOLOv4 model is pruned at the level of channels and layers. First, training under channel-level sparsity-induced regularization is performed in order to identify insignificant channels [242]. L1 regularization of the loss during training is adopted based on the work presented in [227]. Channel pruning is then applied to eliminate the channels with little contribution by deleting its input-output connections and corresponding weights. After channel pruning, layer pruning is performed in order to address the cross layer connections (residual) in YOLOv4 network where the output of a layer is the input to several subsequent layers. Finally, fine tuning is then applied to assist the pruned model to restore accuracy.

The obtained results after each pruning step of YOLOv4 network with *leakyRelu* activation function are reported. It is shown that the mAP drops by 2.3 points after sparsity training. However, this degradation, which is due to the modification of the loss function, is compensated later by the conducted fine-tuning on the pruned network. Also, the results illustrate that the channel pruning greatly reduces the number of model parameters (-98.4%) and the FLOPS (-91.64%) that involve a speed-up effect on embedded devices without suffering from accuracy loss. Fine-tuning of the YOLOv4 network recovers the accuracy loss due sparsity training. The obtained results show that the fine-tuned network achieves higher accuracy ($+2.1$ points in mAP) than the baseline network while preserving a significant decrease in the required memory (-90.4% reduction of parameters) and computations (-91.66% reduction of FLOPS).

The pruned YOLOv4 network is deployed on Kria KV260 Vision AI Kit. An inference speed of 69 FPS is achieved while the mAP is degraded to 0.348. The pruned YOLOv4 model is also deployed on Movidius Myriad X VPU [243]. The model is optimized and quantized to FP16 representation, which is the only supported representation by the VPU. The resultant model achieves an inference speed of 29.8 FPS (2 threads running on 6 SHAVE cores) and 14.7 FPS (1 thread running on 6 SHAVE cores) with a slight degradation in the mAP (71.01%) when compared to the unquantized pruned network (72.4%). In addition, the YOLOv4-tiny model is deployed on Movidius Myriad X VPU. It achieves 30 FPS inference speed with a mAP of 72.4%.

The performance per Watt is determined when running the models on all targeted embedded edge platforms. Jeston Xavier AGX can achieve 3.83 FPS/W and 1.73 FPS/W for YOLOv4-tiny and YOLOv4 networks respectively. Kria KV260 Vision AI kit can achieve 8.125 FPS/W, 1.375 FPS/W and 8.625 FPS/W for YOLOv4-tiny, YOLOv4 and pruned YOLOv4 networks. Movidius Myriad X VPU can achieve 7.45 FPS/W for pruned YOLOv4 model and 7.5 FPS/W for YOLOv4-tiny model. The analysis of the obtained results shows that Jetson Xavier AGX can achieve best inference speed and mean average precision but at the cost of higher power consumption.

This work has been presented initially in the International Conference on System-Integrated Intelligence (SysInt), 2022 and published as a book chapter in *Advances in System-Integrated Intelligence* [244]. Also, the extended work about the optimization and deployment on hardware devices has been accepted to be published in *IEEE International Conference on Electronics, Circuits and Systems (ICECS)* [245] which will be held in Glasgow, UK in October, 2022.

Future work will focus on the following:

- Enrich the dataset to include more images and add thermal images to enable the detection in all conditions (night, fog, etc.)
- Examine multi-view approach to enhance detection by making use of collaboration between different cameras which capture same scene from different points of view.
- Examine the detection of humans from cameras situated under the water for man overboard application.

Chapter 9

Future Research Directions

9.1 Design of Memory-centric Ultra-Low Power Processor

The possibility to run complex deep learning (DL) algorithms for hours outdoors on unplugged wearable devices paves the way for new and potentially disruptive applications. Recent developments in non-volatile memory (NVM) technologies and processor architectures for information processing make this opportunity possible and allow the design of wearable devices with efficient AI-capabilities.

One of the future research topics is to investigate the design of a novel memory-centric multicore architecture for ultra-low power AI-based object detection. The target architecture should devise the best of memory-oriented techniques in order to drastically minimize the power consumption. Based on our previous results on Memory-based Computing [76, 77] and Processing-In-Memory [133, 134, 135, 136] design approaches, we will target the heterogeneous multicore architecture. MBC replaces logic by Non-Volatile power-gated memories to leverage computation redundancy. PIM is a promising approach to reduce memory transfers and power consumption, particularly when using memristive memory arrays. PIM is very efficient for bulk-bitwise logic operations and for matrix multiplications, where the energy cost of moving data is orders of magnitude higher than the computation itself. Nevertheless, for some complex arithmetic tasks that require many successive basic PIM computation steps, passing the operands to a near-memory computing unit could be more time and energy efficient. Accordingly, the mapping of tasks over PIM, MBC and NMC components will be investigated to devise the best design. Figure 9.1 illustrates the target architecture model combining PIM, MBC and NMC.

9.2 Interactive Machine Learning on Edge Devices for Object Detection Application

This research topic aims to include the Human in the loop, namely designing an interactive (AI $\triangleleft\triangleright$ Human) solution rather than a machine providing visual notifications or applying human commands. The originality of this topic concerns the use of interactive annotations

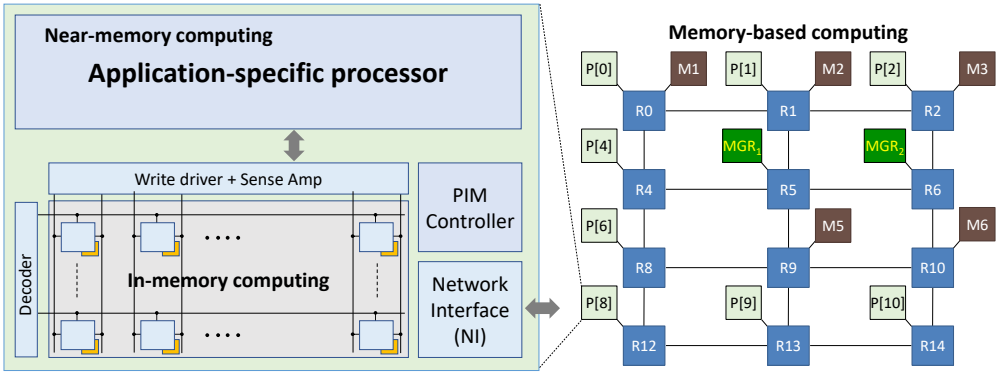


Figure 9.1 — Target architecture model combining PIM, MBC and NMC

to not only track different objects, but also to feed the algorithms to add or reject objects (in incremental learning) considering users' feedback.

While traditional machine learning systems process the data that has been given to them in advance, interactive machine learning considers that the learning process could benefit from interactions with the environment as well as with humans [246], and that inputs and outputs from and for humans carry meaningful information. Indeed, humans may provide input to a learning algorithm [247], including inputs in the form of labels, demonstrations, advice, rewards or rankings. The interaction is all the more useful as the human can guide along the learning process [248] while adapting his guidance to the outputs of the algorithm. Feedback from the users will be used to help algorithms: define zones to focus for detection using hand gestures, refine the confidence of the object class, define new object classes. The AI-based computer vision algorithms will help automatically annotate and overlay visual cues onto real-world objects in the field of vision of the users, as well as to recognize hand gestures allowing them to interact with these objects. For instance, humans could dismiss some tracked objects that are not useful or that can overload the display, or they would also be able to mark in real time objects "missed" by the computer vision algorithm (feeding incremental learning) that they wish to track in the future.

Having a lower power solution opens up opportunities to have more substantial intelligence on the edge devices. Beyond the application itself, it is required to address the challenge of performing online the incremental learning on edge devices.

9.3 Distributed Learning on Connected Devices

Autonomous systems can benefit from growing embedded computing capacities that allow decision making based on multi-sensor fusion and/or complex visual navigation based on semantic recognition [249] and joint mapping and planning [250]. Current challenges are related to learning issues for both object recognition based on offline training of Deep Neural Networks and navigation tasks based on Reinforcement Learning [251]. First offline supervised learning and online inference are efficient but require huge labeled datasets that hardly represent all cases to be experienced by autonomous agents in real-life or at a price of energy and time that can be prohibitive [252]. Therefore, new training phases with updated data-sets

may be required according to edge/cloud computing paradigm [253]. Navigation tasks can be based on pre-trained models but are more efficient if they can learn online from their actions [254] while detecting/identifying obstacles and targets. In both cases, self-adaptivity is required to improve autonomy. The question to be answered is then: how to improve learning on a set of distributed embedded systems (e.g. CPU-GPU) by favoring unsupervised methods such as Reinforcement Learning to improve the autonomy of autonomous systems evolving in groups.

9.4 Tiny Machine Learning (TinyML)

TinyML refers to the integration of machine learning algorithms within ultra-low-power devices, such as Microcontroller Units (MCUs) [255]. Such integration could be the ultimate key towards the reduction of processing power required by applications employed on the edge. Applications such as smart cities and homes, environmental monitoring, industry and business, autonomous driving, and eHealth. This vast spectrum of applications is driving the need for the rapid evolution of TinyML systems, an evolution that faces several challenges along the way. To name a few, the generation of convenient datasets, design of less complex yet effective ML algorithms, memory management, compatibility with existing frameworks for rapid development, etc. [256]. The challenges faced by TinyML cover the whole ML application development pipeline. Thus, solutions like the design of custom hardware accelerators, in-memory computing, approximate computing, network compression, etc. could be a potential interest of the current TinyML research. Moreover, due to the memory constraint of ultra-low-power devices, one could search for new ML algorithms or efficient deep neural network architectures aside from the existing large models.

Bibliography

[1] M. Rizk, A. Baghdadi, M. Jézéquel, Y. Mohanna, and Y. Atat, "Design and prototyping flow of flexible and efficient NISC-based architectures for MIMO turbo equalization and demapping," *Electronics*, vol. 5, no. 3, 2016.

[2] J. Cong and B. Xiao, "mrFPGA: A novel FPGA architecture with memristor-based reconfiguration," in *Proc. of the IEEE/ACM International Symposium on Nanoscale Architectures (NANOARCH)*, 2011, pp. 1–8.

[3] C. Berrou, *Codes and Turbo Codes*. Paris: Springer, 2010.

[4] A. Osseiran *et al.*, "Scenarios for 5G mobile and wireless communications: the vision of the METIS project," *IEEE Communication Magazine*, vol. 52, no. 5, pp. 26–35, May 2014.

[5] F. Vahid and T. Givargis, *Embedded System Design: A Unified Hardware/Software Introduction*. Wiley India Pvt. Limited, 2006.

[6] D. Gajski, "NISC: The ultimate reconfigurable component," Center for Embedded Computer Systems (CECS), University of California, Irvine, Tech. Rep., Oct. 2003.

[7] B. Gorjiara and D. Gajski, "FPGA-friendly code compression for horizontal microcoded custom IPs," in *Proc. of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, ser. FPGA '07. New York, NY, USA: Association for Computing Machinery, 2007, p. 108–115.

[8] B. Gorjiara, M. Reshadi, and D. Gajski, "Merged dictionary code compression for FPGA implementation of custom microcoded PEs," *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, vol. 1, no. 11, pp. 285–297, June 2008.

[9] C. Douillard, M. Jézéquel, C. Berrou, A. Picart, P. Didier, and A. Glavieux, "Iterative correction of inter symbol interference: Turbo-equalization," *European Transactions on Telecommunications (ETT)*, vol. 6, pp. 507–511, Sept.-Oct. 1995.

[10] X. Li and J. Ritcey, "Bit-interleaved coded modulation with iterative decoding," *IEEE Communications Letters*, vol. 1, no. 6, pp. 169–171, 1997.

[11] *IEEE Draft Standard; Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) specifications; Amendment 4: Enhancements for Higher Throughput*, Std., Sept 2007.

[12] *802.16 IEEE Standard for Local and metropolitan area networks, Part 16: Air Interface for Fixed Broadband Wireless Access Systems*, Std., 2004.

[13] *3GPP Technical Specifications 36.212, Multiplexing and Channel Coding (Release 8)*, Std., 2010.

[14] *3GPP Technical Specifications 36.212, Multiplexing and Channel Coding (Release 11)*, Std., 2014.

[15] *Digital Video Broadcasting (DVB); interaction channel for satellite distribution systems*, Std., 2009.

[16] *Digital Video Broadcasting (DVB), Second Generation DVB Interactive Satellite System (DVB-RCS2); Lower Layers for Satellite standard*, Std., 2012.

[17] *Digital Video Broadcasting (DVB); Frame structure channel coding and modulation for a second generation digital terrestrial television broadcasting system (DVB-T2)*, Std., 2012.

[18] *Digital Video Broadcasting (DVB); Implementation guidelines for a second generation digital terrestrial television broadcasting system (DVB-T2)*, Std., 2012.

[19] *Digital Video Broadcasting (DVB); User guidelines for the second generation system for broadcasting, interactive services, news gathering and other broad-band satellite applications (DVB-S2)*, Std., 2005.

[20] M. Rizk, A. Baghdadi, and M. Jézéquel, "Computational complexity reduction of MMSE-IC MIMOturbo detection," *Journal of Circuits, Systems and Computers (JCSC)*, vol. 28, no. 13, p. 1950228, 2019.

[21] M. Rizk, A. Baghdadi, and M. Jézéquel, "A literature survey on algorithms and hardware architectures of Max-Log-MAPdemapping," *Journal of Circuits, Systems and Computers (JCSC)*, vol. 31, no. 3, p. 2230001, 2022.

[22] M. Rizk, A. Baghdadi, M. Jézéquel, Y. Mohanna, and Y. Atat, "Efficient quantization and fixed-point representation for MIMO turbo-detection and turbo-demapping," *EURASIP Journal on Embedded Systems*, vol. 2017, no. 1, p. 33, Oct. 2017.

[23] M. Rizk, A. Baghdadi, M. Jézéquel, Y. Mohanna, and Y. Atat, "NISC-based soft-input soft-output demapper," *IEEE Transactions on Circuits and Systems II*, vol. 62, no. 11, pp. 1098–1102, Nov. 2015.

[24] M. Rizk, A. Baghdadi, M. Jézéquel, Y. Atat, and Y. Mohanna, "NISC-based MIMO MMSE detector," *Journal of Circuits, Systems and Computers (JCSC)*, vol. 30, no. 04, p. 2150069, Sep. 2021.

[25] H. Kim, W. Zhu, J. Bhatia, K. Mohammad, A. Shah, and B. Danesrad, "A Practical Hardware Friendly MMSE Detector for MIMO-ODFM-Based Systems," *EURASIP Journal on Advances in Signal Processing*, 2008.

[26] A. R. Jafri, D. Karakolah, A. Baghdadi, and M. Jézéquel, "ASIP-based flexible MMSE-IC Linear Equalizer for MIMO turbo-equalization applications," in *Proc. of IEEE Design, Automation Test in Europe Conference Exhibition (DATE)*, Apr. 2009, pp. 1620–1625.

[27] A. R. Jafri, "Architectures multi-ASIP pour turbo récepteur flexible," Ph.D. dissertation, Electronics Dept., Telecom Bretagne, Brest, France, 2011.

[28] D. Karakolah, C. Jégo, C. Langlais, and M. Jézéquel, "Architecture dedicated to the MMSE equalizer of iterative receiver for linearly precoded MIMO systems," in *Proc. of IEEE International Conference on Information and Communication Technologies from Theory to Applications(ICTTA)*, Apr. 2008, pp. 1–6.

[29] D. Karakolah, "Conception et prototypage d'un récepteur itératif pour des systèmes de transmision MIMO avec précodage linéaire," Ph.D. dissertation, Electronics Dept., Telecom Bretagne, Brest, France, 2009.

[30] L. Meng, C. Abdel Nour, C. Jégo, and C. Douillard, "Design of rotated QAM mapper/demapper for the DVB-T2 standard," *SiPS 2009 : IEEE workshop on Signal Processing Systems*, Oct. 2009.

[31] M. Li, "Design, implementation, and prototyping of an iterative receiver for bit-interleaved coded modulation system dedicated to DVB-T2," Ph.D. dissertation, Electronics Dept., Telecom Bretagne, Brest, France, 2012.

[32] J. W. Park, M. H. Sunwoo, P. S. Kim, and D.-I. Chang, "Low complexity soft-decision demapper for high order modulation of DVB-S2 system," in *Proc. of the IEEE International SoC Design Conference (ISOCC)*, vol. 02, Nov. 2008, pp. II–37–II–40.

[33] A. R. Jafri, A. Baghdadi, and M. Jézéquel, "ASIP-based universal demapper for multiwireless standards," *IEEE Embedded Systems Letters*, vol. 1, no. 1, pp. 9–13, May 2009.

[34] M. Rizk, A. Baghdadi, M. Jézéquel, and A. C. Al Ghouwayel, "NISC design experience of flexible architectures for digital communication applications," in *Proc. of the IEEE International Conference on Computer and Applications (ICCA)*, Beirut, Lebanon, 2018, pp. 123–129.

[35] M. Rizk, A. Baghdadi, M. Jézéquel, Y. Mohanna, and Y. Atat, "No-instruction-set-computer design experience of flexible and efficient architectures for digital communication applications: two case studies on mimo turbo detection and universal turbo demapping," *Design Automation for Embedded Systems (DAEM)*, vol. 25, no. 1, p. 1–42, Mar. 2021.

[36] K. J. M. Martin, M. Rizk, M. J. Sepulveda, and J.-P. Diguet, "Notifying memories: a case-study on data-flow applications with NoC interfaces implementation," in *Proc. of the Design Automation Conference (DAC)*, Austin, TX, USA, June 2016.

[37] M. Rizk, J.-P. Diguet, and A. Baghdadi, "Rapid design and verification experience using flexible cycle-accurate noc simulator," in *Proc. of the IEEE International Multidisciplinary Conference on Engineering Technology (IMCET)*, Beirut, Lebanon, 2021, pp. 77–83.

[38] J. Liu *et al.*, "Network footprint reduction through data access and computation placement in NoC-based manycores," in *Proc. of the Design Automation Conference (DAC)*, 2015.

[39] A. Firoozshahian, A. Solomatnikov, O. Shacham, Z. Asgar, S. Richardson, C. Kozyrakis, and M. Horowitz, "A memory system design framework: Creating smart memories," in *Proceedings of the 36th Annual International Symposium on Computer Architecture*, ser. ISCA '09. New York, NY, USA: Association for Computing Machinery, 2009, p. 406–417.

[40] J. Yoo *et al.*, "Active memory processor for network-on-chip-based architecture," *IEEE Transactions on Computers*, vol. 61, no. 5, May 2012.

[41] J.-V. Millo *et al.*, "Modeling and analyzing dataflow applications on NoC-based many-core architectures," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 14, no. 3, Apr. 2015.

[42] A. Singh, M. Shafique, A. Kumar, and J. Henkel, "Mapping on multi/many-core systems: Survey of current and emerging trends," in *Proc. of the Design Automation Conference (DAC)*, May 2013, pp. 1–10.

[43] S. S. Bhattacharyya, J. Eker, J. W. Janneck, C. Lucarz, M. Mattavelli, and M. Raulet, "Overview of the MPEG reconfigurable video coding framework," *Journal of Signal Processing Systems*, vol. 63, no. 2, Dec. 2011.

[44] M. Wipliez and M. Raulet, "Classification of dataflow actors with satisfiability and abstract interpretation," *International Journal of Embedded and Real-Time Communication Systems (IJERTCS)*, vol. 3, no. 1, pp. 49–69, 2012.

[45] ORCC. The open RVC-CAL compiler : A development framework for dataflow programs. [Online]. Available: http://orcc.sourceforge.net

[46] Xiph.org video test media. [Online]. Available: http://media.xiph.org/video/derf/

[47] E. Gamma *et al.*, *Design patterns: elements of reusable object-oriented software*. Addison-Wesley Longman Publishing Co., Inc., 1995.

[48] M. Rizk, K. J. Martin, and J.-P. Diguet, "Run-time remapping algorithm of dataflow actors on noc-based heterogeneous mpsocs," *IEEE Transactions on Parallel and Distributed Systems*, pp. 1–1, 2022.

[49] W. A. Najjar, E. A. Lee, and G. R. Gao, "Advances in the dataflow computational model," *Parallel Computing*, vol. 25, no. 13, pp. 1907 – 1929, 1999.

[50] E. A. Lee and D. G. Messerschmitt, "Synchronous data flow," *Proc. of the IEEE*, vol. 75, no. 9, pp. 1235–1245, Sep. 1987.

[51] Y. Lesparre, A. Munier-Kordon, and J. Delosme, "Evaluation of synchronous dataflow graph mappings onto distributed memory architectures," in *Proc. of Euromicro Conference on Digital System Design (DSD)*, Aug. 2016, pp. 146–153.

[52] E. A. Lee and T. Parks, "Dataflow process networks," in *Proc. of the IEEE*, 1995, pp. 773–799.

[53] K. Huang, X. Zhang, D. Zheng, M. Yu, X. Jiang, X. Yan, L. B. de Brisolara, and A. A. Jerraya, "A scalable and adaptable ILP-Based approach for task mapping on MPSoC considering load balance and communication optimization," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 38, no. 9, pp. 1744–1757, Sep. 2019.

[54] Z. Li, D. Wijerathne, X. Chen, A. Pathania, and T. Mitra, "Chordmap: Automated mapping of streaming applications onto CGRA," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, pp. 1–1, 2021.

[55] C. Rubattu, F. Palumbo, S. Bhattacharyya, and M. Pelcat, "Pathtracing: Raising the level of understanding of processing latency in heterogeneous mpsocs," in *Drone Systems Engineering and Rapid Simulation and Performance Evaluation: Methods and Tools*, ser. DroneSE and RAPIDO '21. NY, USA: ACM, 2021, pp. 46–50.

[56] C. Chou and R. Marculescu, "Run-time task allocation considering user behavior in embedded multiprocessor networks-on-chip," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 29, no. 1, pp. 78–91, 2010.

[57] E. L. d. S. Carvalho, N. L. V. Calazans, and F. G. Moraes, "Dynamic task mapping for MPSoCs," *IEEE Design Test of Computers*, vol. 27, no. 5, pp. 26–35, 2010.

[58] M. Fattah, P. Liljeberg, J. Plosila, and H. Tenhunen, "Adjustable contiguity of run-time task allocation in networked many-core systems," in *Proc. of Asia and South Pacific Design Automation Confernce (ASP-DAC)*, 2014, pp. 349–354.

[59] T. Maqsood, S. Ali, S. U. Malik, and S. A. Madani, "Dynamic task mapping for network-on-chip based systems," *Journal of Systems Architecture*, vol. 61, no. 7, pp. 293 – 306, 2015.

[60] A. K. Singh, M. Shafique, A. Kumar, and J. Henkel, "Resource and throughput aware execution trace analysis for efficient run-time mapping on MPSoCs," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 35, no. 1, pp. 72–85, 2016.

[61] C. Ykman-Couvreur, P. Avasare, G. Mariani, G. Palermo, C. Silvano, and V. Zaccaria, "Linking run-time resource management of embedded multi-core platforms with automated design-time exploration," *IET Computers & Digital Techniques*, vol. 5, pp. 123–135(12), Mar. 2011.

[62] W. Quan and A. D. Pimentel, "A scenario-based run-time task mapping algorithm for MPSoCs," in *Proc. of the Design Automation Conference (DAC)*, New York, NY, USA, 2013.

[63] A. K. Singh, A. Kumar, and T. Srikanthan, "Accelerating throughput-aware runtime mapping for heterogeneous MPSoCs," *ACM Transactions on Design Automation of Electronic Systems*, vol. 18, no. 1, Jan. 2013.

[64] S. Skalistis and A. Simalatsar, "Near-optimal deployment of dataflow applications on many-core platforms with real-time guarantees," in *Proc. of the IEEE Design, Automation Test in Europe Conference Exhibition (DATE)*, 2017, pp. 752–757.

[65] J. Henkel, J. Teich, S. Wildermann, and H. Amrouch, "Dynamic resource management for heterogeneous many-cores," in *Proc. of the IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2018, pp. 1–6.

[66] C. Wang, Y. Zhu, J. Jiang, M. Qiu, and X. Wang, "Dynamic application allocation with resource balancing on NoC based many-core embedded systems," *Journal . Syst. Archit.*, vol. 79, no. C, pp. 59–72, Sep. 2017.

[67] S. Kaushik, A. K. Singh, and T. Srikanthan, "Computation and communication aware run-time mapping for NoC-based MPSoC platforms," in *Proc. of IEEE International SOC Conference*, Taipei, Taiwan, 2011, pp. 185–190.

[68] H. R. Mendis, N. C. Audsley, and L. S. Indrusiak, "Dynamic and static task allocation for hard real-time video stream decoding on nocs," *Leibniz Transactions on Embedded Systems*, vol. 4, no. 2, pp. 01:1–01:25, July 2017.

[69] M. Rapp, A. Pathania, T. Mitra, and J. Henkel, "Neural network-based performance prediction for task migration on S-NUCA many-cores," *IEEE Transactions on Computers*, pp. 1–1, 2020.

[70] S. Paul, N. Chatterjee, P. Ghosal, and J.-P. Diguet, "Adaptive task allocation and scheduling on NoC-based multicore platforms with multitasking processors," *ACM Transactions on Embedded Computer Systems*, vol. 20, no. 1, Dec. 2020.

[71] ——, "A hybrid adaptive strategy for task allocation and scheduling for multi-applications on NoC-based multicore systems with resource sharing," in *Proc. of Design, Automation Test in Europe Conference Exhibition (DATE)*, 2021, pp. 1663–1666.

[72] H. Yviquel, E. Casseau, M. Raulet, P. Jääskeläinen, and J. Takala, "Towards run-time actor mapping of dynamic dataflow programs onto multi-core platforms," in *Proc. of the International Symposium on Image and Signal Processing and Analysis (ISPA)*, 2013, pp. 732–737.

[73] T. D. Ngo, K. J. M. Martin, and J.-P. Diguet, "Move based algorithm for runtime mapping of dataflow actors on heterogeneous MPSoCs," *J. of Signal Processing Systems*, vol. 87, no. 1, pp. 63–80, Apr. 2017.

[74] W. Quan and A. D. Pimentel, "A hybrid task mapping algorithm for heterogeneous MPSoCs," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 14, no. 1, pp. 14:1–14:25, Jan. 2015.

[75] T. Grotker, S. Liao, G. Martin, and S. Swan, *System Design with SystemC*. Springer, 2002.

[76] M. Rizk, J.-P. Diguet, N. Onizawa, A. Baghdadi, M.-J. Sepulveda, Y. Akgul, V. Gripon, and T. Hanyu, "NoC-MRAM architecture for memory-based computing: database-search case study," in *Proc. of the IEEE International New Circuits and Systems Conference (NEWCAS)*, Strasbourg, France, June 2017.

[77] J.-P. Diguet, N. Onizawa, M. Rizk, J. Sepulveda, A. Baghdadi, and T. Hanyu, "Networked power-gated MRAMs for memory-based computing," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 26, no. 12, pp. 2696–2708, 2018.

[78] T. Hanyu, T. Endoh, D. Suzuki, H. Koike, Y. Ma, N. Onizawa, M. Natsui, S. Ikeda, and H. Ohno, "Standby-power-free integrated circuits using MTJ-based VLSI computing," *Proc. of the IEEE*, vol. 104, no. 10, pp. 1844–1863, Oct. 2016.

[79] J. Cong, M. Huang, D. Wu, and C. H. Yu, "Heterogeneous datacenters: Options and opportunities," in *DAC*, 2016.

[80] P. H. Pham, D. Jelaca, C. Farabet, B. Martini, Y. LeCun, and E. Culurciello, "Neuflow: Dataflow vision processing system-on-a-chip," in *Proc. of the International Midwest Symp. on Circuits and Systems (MWSCAS)*, Aug. 2012.

[81] P. Merolla et al., "A million spiking-neuron integrated circuit with a scalable communication network & interface," *Science*, vol. 345, no. 6197, 2014.

[82] H. Jarollahi et al., "A nonvolatile associative memory-based context-driven search engine using 90 nm CMOS/MTJ-Hybrid logic-in-memory architecture," *IEEE Jour. on Emerging and Selected Topics in Circuits and Systems*, vol. 4, no. 4, pp. 460–474, Dec. 2014.

[83] S. Paul and S. Bhunia, "A scalable memory-based reconfigurable computing framework for nanoscale crossbar," *IEEE Transactions on Nanotechnology*, vol. 11, no. 3, pp. 451–462, May 2012.

[84] H. R. Lee, C. W. Jen, and C. M. Liu, "On the design automation of the memory-based VLSI architectures for FIR filters," *IEEE Transactions on Consumer Electronics*, vol. 39, no. 3, pp. 619–629, Jun. 1993.

[85] K. Rahmani, P. Mishra, and S. Bhunia, "Memory-based computing for performance and energy improvement in multicore architectures," in *Great Lakes Symp. on VLSI*. ACM, 2012.

[86] A. Rahimi et al., "Energy-efficient GPGPU architectures via collaborative compilation and memristive memory-based computing," in *Proc. of the Design Automation Conference (DAC)*, 2014.

[87] P.-E. Gaillardon, L. Amarú, A. Siemon, E. Linn, R. Waser, A. Chattopadhyay, and G. De Micheli, "The programmable logic-in-memory (plim) computer," in *Proc. of the Design, Automation & Test in Europe Conference & Exhibition (DATE)*, March 2016.

[88] J. Yoo, S. Yoo, and K. Choi, "Active memory processor for network-on-chip-based architecture," *IEEE Transactions on Computers*, vol. 61, no. 5, May 2012.

[89] S. Ikeda et al., "A perpendicular-anisotropy CoFeB-MgO magnetic tunnel junction," *Nature Materials*, vol. 9, pp. 721 – 724, 2010.

[90] R. Takemura, T. Kawahara, K. Miura, H. Yamamoto, J. Hayakawa, N. Matsuzaki, K. Ono, M. Yamanouchi, K. Ito, H. Takahashi, S. Ikeda, H. Hasegawa, H. Matsuoka, and H. Ohno, "A 32-Mb SPRAM with 2T1R memory cell, localized bi-directional write driver and '1'/'0' dual-array equalized reference scheme," *IEEE Journal of Solid-State Circuits*, vol. 45, no. 4, April 2010.

[91] T. Ohsawa et al., "A 1 Mb nonvolatile embedded memory using 4T2MTJ cell with 32 b fine-grained power gating scheme," *IEEE Jour. of Solid-State Circuits*, vol. 48, no. 6, Jun. 2013.

[92] C. Kim, K. Kwon, C. Park, S. Jang, and J. Choi, "A covalent-bonded cross-coupled current-mode sense amplifier for STT-MRAM with 1T1MTJ common source-line structure array," in *Proc. of the IEEE International Solid-State Circuits Conference(ISSCC)*, Feb 2015, pp. 1–3.

[93] H. Sato, E. C. I. Enobio, M. Yamanouchi, S. Ikeda, S. Fukami, S. Kanai, F. Matsukura, and H. Ohno, "Properties of magnetic tunnel junctions with a MgO/CoFeB/-Ta/CoFeB/MgO recording structure down to junction diameter of 11 nm," *Applied Physics Letters*, vol. 105, no. 6, p. 062403, 2014.

[94] K. C. Chun, H. Zhao, J. D. Harms, T. H. Kim, J. P. Wang, and C. H. Kim, "A scaling roadmap and performance evaluation of in-plane and perpendicular MTJ based STT-MRAMs for high-density cache memory," *IEEE Journal of Solid-State Circuits*, vol. 48, no. 2, pp. 598–610, Feb 2013.

[95] S. Peng, W. Kang, M. Wang, K. Cao, X. Zhao, L. Wang, Y. Zhang, Y. Zhang, Y. Zhou, K. L. Wang, and W. Zhao, "Interfacial perpendicular magnetic anisotropy in sub-20 nm tunnel junctions for large-capacity spin-transfer torque magnetic random-access memory," *IEEE Magnetics Letters*, vol. 8, pp. 1–5, 2017.

[96] "ML Repository," https://archive.ics.uci.edu/ml/.

[97] E. Boutillon, L. Conde-Canencia, and A. Al Ghouwayel, "Design of a GF(64)-LDPC decoder based on the EMS algorithm," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 60, no. 10, pp. 2644–2656, 2013.

[98] M. Sabbah, A. Chamas Al Ghouwayel, M. Rizk, and Z. El Bazzal, "MRAM-based memorization system for NB-LDPC decoder," in *Proc. of the IEEE International Conference on Microelectronics (ICM)*, 2017, pp. 1–4.

[99] M. Sabbah, M. Rizk, A. Chamas Al Ghouwayel, S. Omar, and Z. El Bazzal, "Enhanced throughput and energy-efficient MRAM-based NB-LDPC decoder," in *Proc. of the IEEE International Conference on Computer and Applications (ICCA)*, 2018, pp. 172–176.

[100] K. Alhaj Ali, M. Rizk, A. Baghdadi, J.-P. Diguet, and J. Jomaah, "Towards memristor-based reconfigurable FFT architecture," in *Proc. of the IEEE International Conference on Microelectronics (ICM)*, 2017, pp. 1–4.

[101] L. Chua, "Memristor-the missing circuit element," *IEEE Transactions on circuit theory*, vol. 18, no. 5, pp. 507–519, 1971.

[102] H. Lee, Y. Chen, P. Chen, P. Gu, Y. Hsu, S. Wang, W. Liu, C. Tsai, S. Sheu, P. Chiang *et al.*, "Evidence and solution of over-reset problem for HfO_x based resistive memory with sub-ns switching speed and high endurance," in *Proc. of the IEEE International Electron Devices Meeting*, 2010, pp. 19–7.

[103] J. J. Yang, D. B. Strukov, and D. R. Stewart, "Memristive devices for computing," *Nature Nanotechnology*, vol. 8, no. 1, p. 13, 2013.

[104] J. Kang, P. Huang, B. Gao, H. Li, Z. Chen, Y. Zhao, C. Liu, L. Liu, and X. Liu, "Design and application of oxide-based resistive switching devices for novel computing architectures," *IEEE Journal of the Electron Devices Society*, vol. 4, no. 5, pp. 307–313, 2016.

[105] R. Hasan and T. M. Taha, "Memristor crossbar based programmable interconnects," in *Proc. of the IEEE Computer Society Annual Symposium on VLSI*, 2014, pp. 94–99.

[106] M. Lin, A. El Gamal, Y.-C. Lu, and S. Wong, "Performance benefits of monolithically stacked 3-D FPGA," *IEEE Transactions on computer-aided design of integrated circuits and systems*, vol. 26, no. 2, pp. 216–229, 2007.

[107] C. Dong, D. Chen, S. Haruehanroengra, and W. Wang, "3-D nFPGA: A reconfigurable architecture for 3-d cmos/nanomaterial hybrid digital circuits," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 54, no. 11, pp. 2489–2501, 2007.

[108] I. Kuon and J. Rose, "Measuring the gap between FPGAs and ASICs," *IEEE Transactions on computer-aided design of integrated circuits and systems*, vol. 26, no. 2, pp. 203–215, 2007.

[109] P.-E. Gaillardon, M. H. Ben-Jamaa, G. B. Beneventi, F. Clermidy, and L. Perniola, "Emerging memory technologies for reconfigurable routing in FPGA architecture," in *Proc. of the IEEE International Conference on Electronics, Circuits and Systems*, 2010, pp. 62–65.

[110] S. Tanachutiwat, M. Liu, and W. Wang, "FPGA based on integration of CMOS and RRAM," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 19, no. 11, pp. 2023–2032, 2010.

[111] J. W. Cooley and J. W. Tukey, "An algorithm for the machine calculation of complex fourier series," *Mathematics of computation*, vol. 19, no. 90, pp. 297–301, 1965.

[112] J. Yli-Kaakinen, T. Levanen, M. Renfors, M. Valkama, and K. Pajukoski, "FFT-domain signal processing for spectrally-enhanced CP-OFDM waveforms in 5G new radio," in *Proc. of the IEEE Asilomar Conference on Signals, Systems, and Computers*, 2018, pp. 1049–1056.

[113] J. Löfgren and P. Nilsson, "On hardware implementation of radix 3 and radix 5 FFT kernels for LTE systems," in *Proc. of the NORCHIP*, 2011, pp. 1–4.

[114] S. He and M. Torkelson, "A new approach to pipeline FFT processor," in *Proc. of the 10th International Parallel Processing Symposium (IPPS)*, 1996, pp. 766–770.

[115] X.-Y. Shih, Y.-Q. Liu, and H.-R. Chou, "48-Mode Reconfigurable Design of SDF FFT Hardware Architecture Using Radix-3 2 and Radix-2 3 Design Approaches," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 64, no. 6, pp. 1456–1467, 2017.

[116] 3GPP. (2017) Evolved universal terrestrial radio access (E-UTRA); physical channels and modulation. [Online]. Available: https://www.3gpp.org/

[117] K. Alhaj Ali, M. Rizk, A. Baghdadi, J.-P. Diguet, and J. Jomaah, "MRL crossbar-based full adder design," in *Proc. of the IEEE International Conference on Electronics, Circuits and Systems (ICECS)*, 2019, pp. 674–677.

[118] ——, "Hybrid memristor–CMOS implementation of combinational logic based on X-MRL," *Electronics*, vol. 10, no. 9, 2021.

[119] S. Kvatinsky, N. Wald, G. Satat, A. Kolodny, U. C. Weiser, and E. G. Friedman, "MRL–memristor ratioed logic," in *Proc. of the International Workshop on Cellular Nanoscale Networks and their Applications*, 2012, pp. 1–6.

[120] E. Linn, R. Rosezin, S. Tappertzhofen, U. Böttger, and R. Waser, "Beyond von Neumann–logic operations in passive crossbar arrays alongside memory operations," *Nanotechnology*, vol. 23, no. 30, p. 305205, 2012.

[121] S. Kvatinsky, G. Satat, N. Wald, E. G. Friedman, A. Kolodny, and U. C. Weiser, "Memristor-based material implication (IMPLY) logic: Design principles and methodologies," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 22, no. 10, pp. 2054–2066, 2014.

[122] L. Amaru, P.-E. Gaillardon, and G. De Micheli, "Majority-inverter graph: A new paradigm for logic optimization," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 35, no. 5, pp. 806–819, 2015.

[123] L. Guckert and E. E. Swartzlander, "MAD gates–memristor logic design using driver circuitry," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 64, no. 2, pp. 171–175, 2016.

[124] B. Liu, Y. Wang, Z. You, Y. Han, and X. Li, "A signal degradation reduction method for memristor ratioed logic (MRL) gates," *IEICE Electronics Express*, vol. 12, no. 8, pp. 1–6, 2015.

[125] J. Chowdhury, K. Das, and K. Rout, "Implementation of 24T memristor based adder architecture with improved performance," *International Journal of Electrical, Electronics and Data Communication*, vol. 3, no. 6, 2015.

[126] J. M. Shalf and R. Leland, "Computing beyond moore's law," *Computer*, vol. 48, no. 12, pp. 14–23, 2015.

[127] K. J. Kuhn, "Considerations for ultimate cmos scaling," *IEEE transactions on Electron Devices*, vol. 59, no. 7, pp. 1813–1828, 2012.

[128] S. Kvatinsky, D. Belousov, S. Liman, G. Satat, N. Wald, E. G. Friedman, A. Kolodny, and U. C. Weiser, "MAGIC–Memristor-Aided Logic," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 61, no. 11, pp. 895–899, 2014.

[129] S. Kvatinsky, A. Kolodny, U. C. Weiser, and E. G. Friedman, "Memristor-based IMPLY logic design procedure," in *Proc. of the IEEE International Conference on Computer Design (ICCD)*, 2011, pp. 142–147.

[130] L. Gao, F. Alibart, and D. B. Strukov, "Programmable cmos/memristor threshold logic," *IEEE Transactions on Nanotechnology*, vol. 12, no. 2, pp. 115–119, 2013.

[131] S. Kvatinsky, M. Ramadan, E. G. Friedman, and A. Kolodny, "VTEAM: A general model for voltage-controlled memristors," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 62, no. 8, pp. 786–790, 2015.

[132] P. L. Thangkhiew, R. Gharpinde, P. V. Chowdhary, K. Datta, and I. Sengupta, "Area efficient implementation of ripple carry adder using memristor crossbar arrays," in *Proc. of the International Design & Test Symposium (IDT)*, 2016, pp. 142–147.

[133] K. Alhaj Ali, M. Rizk, A. Baghdadi, J.-P. Diguet, and J. Jomaah, "Crossbar memory architecture performing memristor overwrite logic," in *Proc. of the IEEE International Conference on Electronics, Circuits and Systems (ICECS)*, 2019, pp. 723–726.

[134] ——, "Memristor overwrite logic (MOL) for energy-efficient in-memory DNN," in *Proc. of the IEEE International Symposium on Circuits and Systems (ISCAS)*, 2020, pp. 1–5.

[135] K. Alhaj Ali, M. Rizk, A. Baghdadi, J.-P. Diguet, J. Jomaah, N. Onizawa, and T. Hanyu, "Memristive computational memory using memristor overwrite logic (MOL)," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 28, no. 11, pp. 2370–2382, 2020.

[136] K. Alhaj Ali, A. Baghdadi, E. Dupraz, M. Léonardon, M. Rizk, and J.-P. Diguet, "MOL-based in-memory computing of binary neural networks," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 30, no. 7, pp. 869–880, 2022.

[137] E. Lehtonen and M. Laiho, "Stateful implication logic with memristors," in *Proc. of the IEEE International Symposium on Nanoscale Architectures*, 2009, pp. 33–36.

[138] N. Talati, S. Gupta, P. Mane, and S. Kvatinsky, "Logic design within memristive memories using memristor-aided logic (MAGIC)," *IEEE Transactions on Nanotechnology*, vol. 15, no. 4, pp. 635–650, 2016.

[139] P. Thangkhiew, R. Gharpinde, D. N. Yadav, K. Datta, and I. Sengupta, "Efficient implementation of adder circuits in memristive crossbar array," in *Proc. of the IEEE Region 10 Conference (TENCON)*, 2017, pp. 207–212.

[140] S. Shirinzadeh, M. Soeken, P.-E. Gaillardon, and R. Drechsler, *Logic Synthesis for Majority Based In-Memory Computing, Chapter in Advances in memristors, memristive devices and systems*. Springer, 2017.

[141] A. Siemon, S. Menzel, R. Waser, and E. Linn, "A complementary resistive switch-based crossbar array adder," *IEEE journal on emerging and selected topics in circuits and systems*, vol. 5, no. 1, pp. 64–74, 2015.

[142] C.-X. Xue *et al.*, "A 1Mb multibit ReRAM computing-in-memory macro with 14.6 ns parallel MAC computing time for CNN based AI edge processors," in *Proc. of the IEEE international Solid-State Circuits Conference-(ISSCC)*, 2019.

[143] W.-H. Chen, W.-J. Lin *et al.*, "A 16Mb dual-mode ReRAM macro with sub-14ns computing-in-memory and memory functions enabled by self-write termination scheme," in *Proc. of the IEEE international Electron Devices Meeting (IEDM)*, 2017.

[144] C.-X. Xue *et al.*, "A 22nm 2Mb ReRAM compute-in-memory macro with 121-28TOPS/W for multibit MAC computing for tiny AI edge devices," in *Proc. of the IEEE international Solid-State Circuits Conference(ISSCC)*, 2020.

[145] P.-E. Gaillardon, L. Amarú, A. Siemon, E. Linn, R. Waser, A. Chattopadhyay, and G. De Micheli, "The programmable logic-in-memory (PLiM) computer," in *Proc. of the Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2016, pp. 427–432.

[146] Y. Wang, Y. Zhang, E. Deng, J.-O. Klein, L. A. Naviner, and W. Zhao, "Compact model of magnetic tunnel junction with stochastic spin transfer torque switching for reliability analyses," *Microelectronics Reliability*, vol. 54, no. 9-10, pp. 1774–1778, 2014.

[147] Y. Wang, H. Cai, L. A. Naviner, Y. Zhang, J.-O. Klein, and W. Zhao, "Compact thermal modeling of spin transfer torque magnetic tunnel junction," *Microelectronics Reliability*, vol. 55, no. 9-10, pp. 1649–1653, 2015.

[148] R. Gharpinde, P. L. Thangkhiew, K. Datta, and I. Sengupta, "A scalable in-memory logic synthesis approach using memristor crossbar," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 26, no. 2, pp. 355–366, 2018.

[149] R. Balasubramonian, J. Chang, T. Manning, J. H. Moreno, R. Murphy, R. Nair, and S. Swanson, "Near-data processing: Insights from a micro-46 workshop," *IEEE Micro*, vol. 34, no. 4, pp. 36–42, 2014.

[150] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, "XNOR-Net: ImageNet classification using binary convolutional neural networks," in *Proc. of Computer Vision – ECCV 2016*, B. Leibe, J. Matas, N. Sebe, and M. Welling, Eds. Cham: Springer International Publishing, 2016, pp. 525–542.

[151] M. S. Mahmud, J. Z. Huang, S. Salloum, T. Z. Emara, and K. Sadatdiynov, "A survey of data partitioning and sampling methods to support big data analysis," *Big Data Min. Anal.*, vol. 3, no. 2, pp. 85–101, Jun. 2020.

[152] D. Tribe, "Flexibility can help car makers cope with chip supply challenge," *Engineering & Technology*, vol. 16, no. 2, pp. 19–19, Mar. 2021.

[153] S. Venkataramani, S. T. Chakradhar, K. Roy, and A. Raghunathan, "Approximate computing and the quest for computing efficiency," in *Proc. of the Design Automation Conference (DAC)*. San Francisco, California: ACM Press, 2015, pp. 1–6.

[154] P. P. Lele, D. C. Sinclair, and G. Weddell, "The reaction time to touch," *The Journal of Physiology*, vol. 123, no. 1, pp. 187–203, Jan. 1954.

[155] R. Dahiya, G. Metta, M. Valle, and G. Sandini, "Tactile Sensing—From Humans to Humanoids," *IEEE Transactions on Robotics*, vol. 26, no. 1, pp. 1–20, Feb. 2010.

[156] Y.-H. Liu, Y.-T. Hsiao, W.-T. Cheng, Y.-C. Liu, and J.-Y. Su, "Low-Resolution Tactile Image Recognition for Automated Robotic Assembly Using Kernel PCA-Based Feature Fusion and Multiple Kernel Learning-Based Support Vector Machine," *Mathematical Problems in Engineering*, vol. 2014, pp. 1–11, 2014.

[157] K. Lee, T. Ikeda, T. Miyashita, H. Ishiguro, and N. Hagita, "Separation of tactile information from multiple sources based on spatial ICA and time series clustering," in *Proc. of the IEEE/SICE International Symposium on System Integration (SII)*. Kyoto, Japan: IEEE, Dec. 2011, pp. 791–796.

[158] P. Xanthopoulos, P. M. Pardalos, and T. B. Trafalis, "Linear Discriminant Analysis," in *Robust Data Mining*, ser. SpringerBriefs in Optimization, P. Xanthopoulos, P. M. Pardalos, and T. B. Trafalis, Eds. New York, NY: Springer, 2013, pp. 27–33.

[159] W. K, "Tactile sensing for ground classification," *Journal of Automation Mobile Robotics and Intelligent Systems*, vol. 7, no. 2, pp. 18–23, 2013.

[160] H. Nguyen, L. Osborn, M. Iskarous, C. Shallal, C. Hunt, J. Betthauser, and N. Thakor, "Dynamic Texture Decoding Using a Neuromorphic Multilayer Tactile Sensor," in *Proc. of the IEEE Biomedical Circuits and Systems Conference (BioCAS)*. Cleveland, OH: IEEE, Oct. 2018, pp. 1–4.

[161] P. Gastaldo, L. Pinna, L. Seminara, M. Valle, and R. Zunino, "Computational Intelligence Techniques for Tactile Sensing Systems," *Sensors*, vol. 14, no. 6, pp. 10 952–10 976, June 2014.

[162] T. Bhattacharjee, J. M. Rehg, and C. C. Kemp, "Haptic classification and recognition of objects using a tactile sensing forearm," in *Proc. of the IEEE/RSJ International Conference on Intelligent Robots and Systems*. Vilamoura-Algarve, Portugal: IEEE, Oct. 2012, pp. 4090–4097.

[163] I. Bandyopadhyaya, D. Babu, A. Kumar, and J. Roychowdhury, "Tactile sensing based softness classification using machine learning," in *Proc. of the IEEE International Advance Computing Conference (IACC)*. Gurgaon, India: IEEE, Feb. 2014, pp. 1231–1236.

[164] M. Kaboli, P. Mittendorfer, V. Hugel, and G. Cheng, "Humanoids learn object properties from robust tactile feature descriptors via multi-modal artificial skin," in *Proc. of the IEEE-RAS International Conference on Humanoid Robots*. Madrid, Spain: IEEE, Nov. 2014, pp. 187–192.

[165] M. Kaboli, R. Walker, and G. Cheng, "Re-using prior tactile experience by robotic hands to discriminate in-hand objects via texture properties," in *Proc. of the IEEE International Conference on Robotics and Automation (ICRA)*. Stockholm, Sweden: IEEE, May 2016, pp. 2242–2247.

[166] D. Fujiki, X. Wang, A. Subramaniyan, and R. Das, "In-/Near-Memory Computing," *Synthesis Lectures on Computer Architecture*, vol. 16, no. 2, pp. 1–140, Aug. 2021.

[167] V. Wong and M. Horowitz, "Soft Error Resilience of Probabilistic Inference Applications," in *Proc. of the WORKSHOP ON SYSTEM EFFECTS OF LOGIC SOFT (ERRORS)*, 2006.

[168] L. Leem, Hyungmin Cho, J. Bau, Q. A. Jacobson, and S. Mitra, "ERSA: Error Resilient System Architecture for probabilistic applications," in *Proc. of the Design, Automation & Test in Europe Conference & Exhibition (DATE)*. Dresden: IEEE, Mar. 2010, pp. 1560–1565.

[169] S. Misailovic, S. Sidiroglou, H. Hoffmann, and M. Rinard, "Quality of service profiling," in *Proc. of the ACM/IEEE International Conference on Software Engineering (ICSE)*, vol. 1. Cape Town, South Africa: ACM Press, 2010, p. 25.

[170] P. Roy, R. Ray, C. Wang, and W. F. Wong, "ASAC: automatic sensitivity analysis for approximate computing," *ACM SIGPLAN Notices*, vol. 49, no. 5, pp. 95–104, May 2014.

[171] J. Ludwig, S. Nawab, and A. Chandrakasan, "Low-power digital filtering using approximate processing," *IEEE Journal of Solid-State Circuits*, vol. 31, no. 3, pp. 395–400, Mar. 1996.

[172] V. K. Chippa, D. Mohapatra, A. Raghunathan, K. Roy, and S. T. Chakradhar, "Scalable effort hardware design: exploiting algorithmic resilience for energy efficiency," in *Proc. of the Design Automation Conference (DAC)*. Anaheim, California: ACM Press, 2010, p. 555.

[173] E. Nogues, D. Menard, and M. Pelcat, "Algorithmic-Level Approximate Computing Applied to Energy Efficient Hevc Decoding," *IEEE Transactions on Emerging Topics in Computing*, pp. 1–1, 2016.

[174] H. Younes, A. Ibrahim, M. Rizk, and M. Valle, "Data oriented approximate K-nearest neighbor classifier for touch modality recognition," in *Proc. of the Conference on Ph.D Research in Microelectronics and Electronics (PRIME)*, 2019, pp. 241–244.

[175] ——, "Algorithmic level approximate computing for machine learning classifiers," in *Proc. of the IEEE International Conference on Electronics, Circuits and Systems (ICECS)*, 2019, pp. 113–114.

[176] ——, "Algorithmic-level approximate tensorial SVM using high-level synthesis on FPGA," *Electronics*, vol. 10, no. 2, 2021.

[177] J. Sun, W. Du, and N. Shi, "A Survey of kNN Algorithm," *Inf Eng Appl Comput*, vol. 1, no. 1, May 2018.

[178] Zhe-Hao Li, Ji-Fang Jin, Xue-Gong Zhou, and Zhi-Hua Feng, "K-nearest neighbor algorithm implementation on FPGA using high level synthesis," in *Proc. of the IEEE International Conference on Solid-State and Integrated Circuit Technology (ICSICT)*. Hangzhou, China: IEEE, Oct. 2016, pp. 600–602.

[179] J. Saikia, S. Yin, Z. Jiang, M. Seok, and J.-s. Seo, "K-Nearest Neighbor Hardware Accelerator Using In-Memory Computing SRAM," in *Proc. of the IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*. Lausanne, Switzerland: IEEE, July 2019, pp. 1–6.

[180] H. Younes, A. Ibrahim, M. Rizk, and M. Valle, "An efficient selection-based kNN architecture for smart embedded hardware accelerators," *IEEE Open Journal of Circuits and Systems (OJCS)*, vol. 2, pp. 534–545, 2021.

[181] D. Tao, X. Li, X. Wu, W. Hu, and S. J. Maybank, "Supervised tensor learning," *Knowledge and Information Systems*, vol. 13, no. 1, pp. 1–42, Sep. 2007.

[182] M. Signoretto, L. De Lathauwer, and J. A. Suykens, "A kernel-based framework to tensorial data analysis," *Neural Networks*, vol. 24, no. 8, pp. 861–874, Oct. 2011.

[183] P. Gastaldo, L. Pinna, L. Seminara, M. Valle, and R. Zunino, "A Tensor-Based Pattern-Recognition Framework for the Interpretation of Touch Modality in Artificial Skin Systems," *IEEE Sensors Journal*, vol. 14, no. 7, pp. 2216–2225, July 2014.

[184] 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, Nov. 2018.

[185] H. Fares, L. Seminara, A. Ibrahim, M. Franceschi, L. Pinna, M. Valle, S. Dosen, and D. Farina, "Distributed Sensing and Stimulation Systems for Sense of Touch Restoration in Prosthetics," in *Proc. of the IEEE New Generation of CAS (NGCAS)*. Genova, Italy: IEEE, Sep. 2017, pp. 177–180.

[186] H. Younes, A. Ibrahim, M. Rizk, and M. Valle, "Efficient FPGA implementation of approximate singular value decomposition based on shallow neural networks," in *Proc. of the IEEE International Conference on Artificial Intelligence Circuits and Systems (AICAS)*, 2021, pp. 1–4.

[187] ——, "A shallow neural network for real-time embedded machine learning for tensorial tactile data processing," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 68, no. 10, pp. 4232–4244, 2021.

[188] C. Alippi, S. Disabato, and M. Roveri, "Moving Convolutional Neural Networks to Embedded Systems: The AlexNet and VGG-16 Case," in *Proc. of the ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN)*. Porto: IEEE, Apr. 2018, pp. 212–223.

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

[190] Y. He, X. Zhang, and J. Sun, "Channel Pruning for Accelerating Very Deep Neural Networks," in *Proc. of the IEEE International Conference on Computer Vision (ICCV)*. Venice: IEEE, Oct. 2017, pp. 1398–1406.

[191] J. Yim, D. Joo, J. Bae, and J. Kim, "A Gift from Knowledge Distillation: Fast Optimization, Network Minimization and Transfer Learning," in *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Honolulu, HI: IEEE, July 2017, pp. 7130–7138.

[192] Y. Gong, L. Liu, M. Yang, and L. Bourdev, "Compressing Deep Convolutional Networks using Vector Quantization," *arXiv:1412.6115 [cs]*, Dec. 2014.

[193] M. Fischer and J. Wassner, "BinArray: A Scalable Hardware Accelerator for Binary Approximated CNNs," in *Proc. of the IEEE Annual Computing and Communication Workshop and Conference (CCWC)*. NV, USA: IEEE, Jan. 2021, pp. 0197–0205.

[194] L. Geiger and P. Team, "Larq: An Open-Source Library for Training Binarized Neural Networks," *JOSS*, vol. 5, no. 45, p. 1746, Jan. 2020.

[195] H. Younes, A. Ibrahim, M. Rizk, and M. Valle, "A mixed-precision binary neural network architecture for touch modality classification," in *SMACD / PRIME 2021; International Conference on SMACD and 16th Conference on PRIME*, 2021, pp. 1–4.

[196] ——, "Hybrid fixed-point/binary convolutional neural network accelerator for real-time tactile processing," in *Proc. of the IEEE International Conference on Electronics, Circuits, and Systems (ICECS)*, 2021, pp. 1–5.

[197] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention Is All You Need," *arXiv:1706.03762 [cs]*, Dec. 2017, arXiv: 1706.03762.

[198] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby, "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale," *arXiv:2010.11929 [cs]*, June 2021, arXiv: 2010.11929.

[199] P. Huang, C. Wang, W. Liu, F. Qiao, and F. Lombardi, "A Hardware/Software Co-Design Methodology for Adaptive Approximate Computing in clustering and ANN Learning," *IEEE Open J. Comput. Soc.*, vol. 2, pp. 38–52, 2021.

[200] T. Nomani, M. Mohsin, Z. Pervaiz, and M. Shafique, "xUAVs: Towards Efficient Approximate Computing for UAVs—Low Power Approximate Adders With Single LUT Delay for FPGA-Based Aerial Imaging Optimization," *IEEE Access*, vol. 8, pp. 102 982–102 996, 2020.

[201] P. Newman, D. Cole, and K. Ho, "Outdoor SLAM using visual appearance and laser ranging," in *Proc. of the IEEE International Conference on Robotics and Automation, 2006. ICRA 2006.*, 2006, pp. 1180–1187.

[202] T. Taketomi, H. Uchiyama, and S. Ikeda, "Visual slam algorithms: A survey from 2010 to 2016," *IPSJ Transactions on Computer Vision and Applications*, vol. 9, 2017, publisher Copyright: © The Author(s).

[203] J. Engel, J. Sturm, and D. Cremers, "Camera-based navigation of a low-cost quadrocopter," in *Proc. of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2012, pp. 2815–2821.

[204] T. Yang, P. Li, H. Zhang, J. Li, and Z. Li, "Monocular vision SLAM-based UAV autonomous landing in emergencies and unknown environments," *Electronics*, vol. 7, no. 5, 2018.

[205] Y. Lu, Z. Xue, G.-S. Xia, and L. Zhang, "A survey on vision-based UAV navigation," *Geo-spatial Information Science*, vol. 21, no. 1, pp. 21–32, 2018.

[206] M. Rizk, A. Mroue, M. Farran, and J. Charara, "Real-time SLAM Based on image stitching for autonomous navigation of UAVs in GNSS-denied regions," in *Proc. of the IEEE International Conference on Artificial Intelligence Circuits and Systems (AICAS)*, 2020, pp. 301–304.

[207] M. Rizk, A. Mroue, K. A. Ali, and J. Charara, "Development and implementation of an embedded low-cost GNSS receiver emulator for UAV testing," in *Proc. of the IEEE International Conference on Electronics, Circuits and Systems (ICECS)*, 2019, pp. 101–102.

[208] F. E. Schöller, M. K. Plenge-Feidenhans'l, J. D. Stets, and M. Blanke, "Assessing deep-learning methods for object detection at sea from LWIR images," *IFAC-PapersOnLine*, vol. 52, no. 21, pp. 64–71, 2019, iFAC Conference on Control Applications in Marine Systems, Robotics, and Vehicles CAMS.

[209] M. Blanke, S. Hansen, J. Stets, T. Koester, J. Brøsted, A. Maurin, N. Nykvist, and J. Bang, "Outlook for navigation - comparing human performance with a robotic solution," in *Proc. of theInternational Conference on Maritime Autonomous Surface Ships (ICMASS)*. SINTEF, 2019, international Conference on Maritime Autonomous Surface Ships ; Conference date: 08-11-2018 Through 09-11-2018.

[210] D. K. Prasad, D. Rajan, L. Rachmawati, E. Rajabally, and C. Quek, "Video processing from electro-optical sensors for object detection and tracking in a maritime environment: A survey," *IEEE Transactions on Intelligent Transportation Systems*, vol. 18, no. 8, pp. 1993–2016, 2017.

[211] Z. Bi, S. Zhang, Y. Wu, G. Wang, and C. Qi, "A survey of ship target feature extraction based on video surveillance," *Journal of Physics: Conference Series*, vol. 1976, no. 1, p. 012014, July 2021.

[212] D. K. Prasad, C. K. Prasath, D. Rajan, L. Rachmawati, E. Rajabally, and C. Quek, "Challenges in video based object detection in maritime scenario using computer vision," *International Journal of Computer and Information Engineering*, vol. 11, no. 1, pp. 31 – 36, 2017.

[213] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," in *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2014, pp. 580–587.

[214] R. Girshick, "Fast R-CNN," in *Proc. of the IEEE International Conference on Computer Vision (ICCV)*, 2015, pp. 1440–1448.

[215] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards real-time object detection with region proposal networks," in *Advances in Neural Information Processing Systems (NeurIPS)*, C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett, Eds., vol. 28. Curran Associates, Inc., 2015.

[216] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You Only Look Once: Unified, real-time object detection," in *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 779–788.

[217] D. Qiao, G. Liu, T. Lv, W. Li, and J. Zhang, "Marine vision-based situational awareness using discriminative deep learning: A survey," *Journal of Marine Science and Engineering*, vol. 9, no. 4, 2021.

[218] R. Zhang *et al.*, "Survey on deep learning-based marine object detection," *Journal of Advanced Transportation*, vol. 2021, 2021.

[219] J. Redmon and A. Farhadi, "YOLO9000: Better, faster, stronger," in *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.

[220] ——, "YOLOv3: An incremental improvement," 2018.

[221] A. Bochkovskiy, C. Wang, and H. M. Liao, "Yolov4: Optimal speed and accuracy of object detection," *CoRR*, vol. abs/2004.10934, Apr. 2020.

[222] "COCO - common objects in context web site," https://cocodataset.org/, accessed: 2020-06-20.

[223] M. Tan, R. Pang, and Q. V. Le, "Efficientdet: Scalable and efficient object detection," in *Proc. of the IEEE/CVF conference on computer vision and pattern recognition*, 2020, pp. 10 781–10 790.

[224] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár, "Focal loss for dense object detection," in *Proc. of the IEEE international conference on computer vision*, 2017, pp. 2980–2988.

[225] U. Nepal and H. Eslamiat, "Comparing YOLOv3, YOLOv4 and YOLOv5 for autonomous landing spot detection in faulty UAVs," *Sensors*, vol. 22, no. 2, 2022.

[226] D. Heller, M. Rizk, R. Douguet, A. Baghdadi, and J.-P. Diguet, "Marine objects detection using deep learning on embedded edge devices," 2022, manuscript submitted to the IEEE International Workshop on Rapid System Prototyping (RSP).

[227] Z. Liu *et al.*, "Learning efficient convolutional networks through network slimming," in *Proc. of the IEEE International Conference on Computer Vision (ICCV)*, 2017, pp. 2755–2763.

[228] W. Gong, M. Rizk, Z. Ning, A. Baghdadi, and J.-P. Diguet, "Efficient communication and decision strategies for multi-view ai-based object detection on edge devices," 2022, manuscript submitted to the IEEE International Workshop on Rapid System Prototyping (RSP).

[229] Z. Ning, M. Rizk, A. Baghdadi, and J.-P. Diguet, "Enhancing embedded ai-based object detection using multi-view approach," 2022, manuscript submitted to the IEEE International Workshop on Rapid System Prototyping (RSP).

[230] J. Sharafaldeen, M. Rizk, A. Baghdadi, and J.-P. Diguet, "Top-view real-time embedded ai detector to assist multi-view marine object surveillance," 2022, manuscript submitted to the IEEE International Conference on Smart Systems and Power Management (IC2SPM).

[231] "International Organization for Migration Missing Migrants Project website," http://missingmigrants.iom.int, accessed: 2022-05-01.

[232] E. M. S. Agency, "Annual overview of marine casualties and incidents 2021," EMSA, Annual Report, Dec. 2021.

[233] G. Castellano, C. Castiello, C. Mencar, and G. Vessio, "Preliminary evaluation of TinyYOLO on a new dataset for search-and-rescue with drones," in *International Conference on Soft Computing Machine Intelligence (ISCMI)*, 2020, pp. 163–166.

[234] C. Liu and T. Szirányi, "Real-time human detection and gesture recognition for on-board UAV rescue," *Sensors*, vol. 21, no. 6, 2021.

[235] M. Rizk, F. Slim, and J. Charara, "Toward AI-assisted UAV for human detection in search and rescue missions," in *Proc. of the International Conference on Decision Aid Sciences and Application (DASA)*, Sakheer, Bahrain, Dec. 2021, pp. 781–786.

[236] S. Sambolek and M. Ivasic-Kos, "Automatic person detection in search and rescue operations using deep cnn detectors," *IEEE Access*, vol. 9, pp. 37 905–37 922, 2021.

[237] R. L. Rosero, C. Grilo, and C. Silva, "Deep learning with real-time inference for human detection in search and rescue," in *Intelligent Systems Design and Applications*, A. Abraham, V. Piuri, N. Gandhi, P. Siarry, A. Kaklauskas, and A. Madureira, Eds. Cham: Springer International Publishing, 2021, pp. 247–257.

[238] E. Lygouras, N. Santavas, A. Taitzoglou, K. Tarchanidis, A. Mitropoulos, and A. Gasteratos, "Unsupervised human detection with an embedded vision system on a fully autonomous UAV for search and rescue operations," *Sensors*, vol. 19, no. 16, 2019.

[239] V. A. Feraru, R. E. Andersen, and E. Boukas, "Towards an autonomous UAV-based system to assist search and rescue operations in man overboard incidents," in *Proc. of the IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR)*, Abu Dhabi, UAE, Nov. 2020, pp. 57–64.

[240] L. Qingqing, J. Taipalmaa, J. P. Queralta, T. N. Gia, M. Gabbouj, H. Tenhunen, J. Raitoharju, and T. Westerlund, "Towards active vision with UAVs in marine search and rescue: Analyzing human detection at variable altitudes," in *Proc. of the IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR)*, Abu Dhabi, UAE, Nov. 2020, pp. 65–70.

[241] R. Padilla, W. L. Passos, T. L. B. Dias, S. L. Netto, and E. A. B. da Silva, "A comparative analysis of object detection metrics with a companion open-source toolkit," *Electronics*, vol. 10, no. 3, 2021.

[242] M. Tian, X. Li, S. Kong, L. wu, and J. Yu, "Pruning-based YOLOv4 algorithm for underwater gabage detection," in *Proc. of the Chinese Control Conference (CCC)*, 2021, pp. 4008–4013.

[243] M. H. Ionica and D. Gregg, "The movidius myriad architecture's potential for scientific computing," *IEEE Micro*, vol. 35, no. 1, pp. 6–14, 2015.

[244] M. Rizk, F. Slim, A. Baghdadi, and J.-P. Diguet, "Towards real-time human detection in maritime environment using embedded deep learning," in *Advances in System-Integrated Intelligence*, M. Valle, D. Lehmhus, C. Gianoglio, E. Ragusa, L. Seminara, S. Bosse, A. Ibrahim, and K.-D. Thoben, Eds. Cham: Springer International Publishing, 2023, pp. 583–593.

[245] M. Rizk, D. Heller, R. Douguet, A. Baghdadi, and J.-P. Diguet, "Optimization of deep-learning detection of humans in marine environment on edge devices," in *Proc. of the IEEE International Conference on Electronics, Circuits and Systems (ICECS)*, Oct. 2022.

[246] L. Jiang, S. Liu, and C. Chen, "Recent research advances on interactive machine learning," *Journal of Visualization*, vol. 22, pp. 401–417, 2019.

[247] A. Holzinger, M. Plass, M. Kickmeier-Rust, K. Holzinger, G. C. Crişan, C.-M. Pintea, and V. Palade, "Interactive machine learning: Experimental evidence for the human in the algorithmic loop," *Applied Intelligence*, vol. 49, no. 7, p. 2401–2414, July 2019.

[248] S. Amershi, M. Cakmak, W. B. Knox, and T. Kulesza, "Power to the people: The role of humans in interactive machine learning," *AI Magazine*, vol. 35, no. 4, pp. 105–120, Dec. 2014.

[249] A. Mousavian, A. Toshev, M. Fišer, J. Košecká, A. Wahid, and J. Davidson, "Visual representations for semantic target driven navigation," in *Proc. of the International Conference on Robotics and Automation (ICRA)*, 2019, pp. 8846–8852.

[250] S. Gupta, J. Davidson, S. Levine, R. Sukthankar, and J. Malik, "Cognitive mapping and planning for visual navigation," in *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Los Alamitos, CA, USA: IEEE Computer Society, jul 2017, pp. 7272–7281.

[251] J. Kulhánek, E. Derner, T. de Bruin, and R. Babuška, "Vision-based navigation using deep reinforcement learning," in *Proc. of the European Conference on Mobile Robots (ECMR)*, 2019, pp. 1–8.

[252] E. Strubell, A. Ganesh, and A. McCallum, "Energy and policy considerations for modern deep learning research," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, no. 09, pp. 13 693–13 696, Apr. 2020.

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

[254] M. Wortsman, K. Ehsani, M. Rastegari, A. Farhadi, and R. Mottaghi, "Learning to learn how to learn: Self-adaptive visual navigation using meta-learning," in *Proc. of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[255] S. Han, "Putting AI on diet: TinyML and efficient deep learning," in *Proc. of the International Symposium on VLSI Design, Automation and Test (VLSI-DAT)*, 2021, pp. i–i.

[256] H. Han and J. Siebert, "TinyML: A systematic review and synthesis of existing research," in *Proc. of the International Conference on Artificial Intelligence in Information and Communication (ICAIIC)*, 2022, pp. 269–274.

Titre : Architectures efficaces pour calcul embarquée haute-performance

Mots clés : Architectures efficaces, flexibilité, calcul embarquée, haute-performance

Résumé : Ce mémoire d'habilitation à diriger des recherches présente mes nombreuses activités de recherche menées depuis 2011 visant le développement d'architectures flexibles et efficaces pour le calcul embarqué haute-performance. Les activités de recherche présentées visent la proposition et la conception d'architectures flexibles et efficaces dans plusieurs domaines applicatifs tels que la communication numérique, les flots de données, les réseaux de neurones, l'apprentissage automatique et la vision embarquée. Ces travaux de recherche ont porté sur la conception et la mise en œuvre de nouvelles architectures matérielles visant à répondre aux exigences émergentes en matière de flexibilité, ainsi qu'aux exigences toujours plus grandes en termes de performances et de réduction de la consommation d'énergie et des ressources matérielles. Dans ce contexte, plusieurs travaux de recherche ont été initiés à travers des projets de recherche terminés ou en cours, deux thèses de doctorat soutenues et plusieurs thèses de Master. Les réalisations les plus significatives sont présentées en les regroupant en quatre sous-thèmes :
(1) Architectures flexibles et efficaces pour des applications dans le domaine des communications numériques, (2) Algorithmes et architectures efficaces pour des applications de flot de données, (3) Paradigmes de conception efficaces et flexibles basés sur des dispositifs memristifs émergents et (4) Implémentations efficaces d'algorithmes d'apprentissage automatique. Mes activités de recherche actuelles se concentrent sur la vision par ordinateur et l'intelligence artificielle embarquées dans le but de réaliser des implémentations efficaces sur des dispositifs embarqués avec de faibles ressources de calcul et un faible budget énergétique.

Title : Efficient Architectures for High-Performance Embedded Computing

Keywords : Efficient Architectures, Embedded Computing, Flexibility, High-Performance

Abstract: This Habilitation to supervise research presents the numerous research activities performed since 2014 targeting the development of flexible and efficient architectures for high performance embedded computing. The presented research activities aim at the realization of flexible and efficient architectures in multitude application domains such as digital communication, data-flow, neural networks, embedded machine learning and embedded vision. These research works have addressed the design and implementation of novel hardware architectures aiming to attain the emergent flexibility requirement, and the ever-increasing requirements of enhanced performance and reduced power consumption and implementation resources. The performed work has targeted the elaboration of new algorithms and hardware architectures using different design paradigms. In this context, several research works have been initiated through completed or ongoing research projects, two defended PhD theses and several Master theses. The most significant achievements are presented by grouping them in four sub-themes: (1) Flexible yet efficient architectures for applications in the digital communication domain; (2) Efficient algorithms and architectures for dataflow applications; (3) Efficient and flexible design paradigms based on emergent memristive devices and (4) Efficient implementations of machine learning algorithms. Current research activities focus on embedded computer vision and artificial intelligence with the goal of achieving efficient implementations on edge devices with low computational resources and low power budget.