Dynamically Reconfigurable Architectures for Real-time Baseband Processing

Chenxin Zhang

LUND UNIVERSITY

Doctoral Dissertation
Digital Circuit Design
Lund, May 2014

Department of Electrical and Information Technology
Lund University
Box 118, 221 00 Lund
Sweden

Series of licentiate and doctoral thesis
ISSN 1654-790X; No. 60
ISBN 978-91-7473-973-2

Typeset in Computer Modern 10pt,
with the LaTeX Documentation System
using Pontus Åströms thesis template.

Printed in Sweden by Tryckeriet i E-huset, Lund University, Lund.

Abstract

Motivated by challenges from today's fast-evolving wireless communication standards and soaring silicon design cost, it is important to design a flexible hardware platform that can be dynamically reconfigured to adapt to current operating scenarios, provide seamless handover between different communication networks, and extend the longevity of advanced systems. Moreover, increasingly sophisticated baseband processing algorithms pose stringent requirements of real-time processing for hardware implementations, especially for power-budget limited mobile terminals. With existing hardware platforms such as Application-Specific Integrated Circuits (ASICs), Field-Programmable Gate Arrays (FPGAs), and Digital Signal Processors (DSPs), the contradictory design requirements of flexibility, computational performance, and hardware efficiency cannot be attained at the same time.

To achieve a balance between the aforementioned design requirements, a coarse-grained dynamically reconfigurable cell array architecture is proposed. The architecture is constructed from an array of heterogeneous function units interconnected through a hierarchical on-chip network. The adopted in-cell configuration scheme enables fast context switching between standards and between computational tasks during run-time. Although cell array is a generic hardware platform, this thesis focuses on the architectural development of the cell array tailored specifically for digital baseband processing of contemporary wireless communication systems. Various degrees of flexibilities among operating scenarios, algorithms, tasks, and supporting standards are exploited. Besides, high hardware efficiency is attained by conducting algorithm-architecture, hardware-software, and processing-memory co-design.

In this thesis, flexibility, performance and efficiency of the proposed architecture are demonstrated through two case studies. First, the cell array is deployed in a digital front-end receiver, aiming to support concurrent processing of multiple radio standards, 3GPP Long Term Evolution (LTE), IEEE 802.11n, and Digital Video Broadcasting for Handheld (DVB-H). Dynamic configuration of the cell array enables run-time switching between different operation modes, multi-standard single-stream and multi-standard multi-stream, in order to maximize hardware usage for attaining high computational performance while sufficing current processing demands. Implementation results show that the immense flexibility offered by the cell array comes at the cost of only about 16% area overhead in comparison to its ASIC counterpart. In the second study, the cell array architecture is extended with extensive vector computing capabilities, aiming to perform high-throughput MIMO signal processing. As

an illustration, three computationally intensive blocks, namely channel estimation, pre-processing, and symbol detection, of a 4×4 MIMO processing chain in a 20 MHz 64-QAM 3GPP LTE-Advanced downlink are mapped and processed in real-time. With 6 processing and 10 memory cells deployed in the array, the achieved system throughput is 368 Mb/s at 500 MHz and the corresponding energy consumption for processing one information bit is 1.49 nJ/b. Compared to state-of-the-art implementations, the proposed solution outperforms related programmable platforms by up to 6 orders of magnitude in energy efficiency, and is 1.7−13.6 and 1.4−15 times less efficient than ASICs in terms of area and energy, respectively, when performing each individual task.

Learning is not attained by chance,
it must be sought for with ardor and
attended to with diligence.

Abigail Adams (1744 - 1818)

Contents

Preface

This thesis summarizes my academic work carried out during the period between April 2009 and May 2014 in the digital circuit design group, at the Department of Electrical and Information Technology, Lund University, Sweden. The main contributions are derived from the following articles:

- Chenxin Zhang, Liang Liu, Dejan Marković, and Viktor Öwall, "A Heterogeneous Reconfigurable Cell Array for MIMO Signal Processing," submitted to *IEEE Transactions on Circuits and Systems-I: Regular Papers*.

  Contribution The research work has been performed by the first author in the guidance of the remaining authors.

- Isael Diaz, Chenxin Zhang, Lieven Hollevoet, Jim Svensson, Joachim Neves Rodrigues, Leif Wilhelmsson, Thomas Olssson, Liesbet Van der Perre, and Viktor Öwall, "A New Digital Front-End for Flexible Reception in Software Defined Radio," submitted to *Microprocessors and Microsystems: Embedded Hardware Design*.

  Contribution Hardware development of the cell array used in the design and the corresponding task mapping have been performed by the author.

- Stefan Granlund, Liang Liu, Chenxin Zhang, and Viktor Öwall, "A Low-Latency High-Throughput Soft-Output Signal Detector for Spatial Multiplexing MIMO Systems," submitted to *Microprocessors and Microsystems: Embedded Hardware Design*.

  Contribution The author has developed a baseline algorithm and provided a simulation testbed for performance evaluations.

- Chenxin Zhang, Hemanth Prabhu, Liang Liu, Ove Edfors, and Viktor Öwall, "Energy Efficient SQRD Processor for LTE-A Using a Group-sort Update Scheme," in *Proceedings of IEEE International Symposium on Circuits and Systems*, Melbourne, Australia, June 2014.

  Contribution The research work has been performed by the first and the second authors in the guidance of the remaining authors. The first author has developed a hardware platform used for the proposed algorithm.

Isael Diaz, Chenxin Zhang, Lieven Hollevoet, Jim Svensson, Joachim Neves Rodrigues, Leif Wilhelmsson, Thomas Olssson, Liesbet Van der Perre, and Viktor Öwall, "Next Generation Digital Front-End for Multi-Standard Concurrent Reception," in *Proceedings of NORCHIP*, Vilnius, Lithuania, November 2013.

Contribution Hardware development of the cell array used in the design and the corresponding task mapping have been performed by the author.

Stefan Granlund, Liang Liu, Chenxin Zhang, and Viktor Öwall, "Implementation of a Highly-Parallel Soft-Output MIMO Detector with Fast Node Enumeration," in *Proceedings of NORCHIP*, Vilnius, Lithuania, November 2013.

Contribution The author has developed a baseline algorithm and provided a simulation testbed for performance evaluations.

Chenxin Zhang, Liang Liu, Yian Wang, Meifang Zhu, Ove Edfors, and Viktor Öwall, "A Highly Parallelized MIMO Detector for Vector-Based Reconfigurable Architectures," in *Proceedings of IEEE Wireless Communications and Networking Conference*, Shanghai, China, April 2013.

Contribution Based on the original idea of the second author, the first author has developed the algorithm and carried out performance analysis with the help of the third and the fourth author under the supervision of the remaining authors.

Chenxin Zhang, Hemanth Prabhu, Liang Liu, Ove Edfors, and Viktor Öwall, "Energy Efficient MIMO Channel Pre-processor Using a Low Complexity On-Line Update Scheme," in *Proceedings of NORCHIP*, Copenhagen, Denmark, November 2012.

Contribution The research work has been performed by the first and the second authors in the guidance of the remaining authors. The first author has formulated the proposed algorithm and carried out performance analysis.

Chenxin Zhang, Liang Liu, and Viktor Öwall, "Mapping Channel Estimation and MIMO Detection in LTE-Advanced on a Reconfigurable Cell Array," in *Proceedings of IEEE International Symposium on Circuits and Systems*, Seoul, Korea, May 2012.

Contribution The research work has been performed by the first author in the guidance of the remaining authors.

- Chenxin Zhang, Isael Diaz, Per Andersson, Joachim Neves Rodrigues, and Viktor Öwall, "Reconfigurable Cell Array for Concurrent Support of Multiple Radio Standards by Flexible Mapping," in *Proceedings of IEEE International Symposium on Circuits and Systems*, Rio de Janeiro, Brazil, May 2011.

  Contribution The research work has been performed by the first and the second authors in the guidance of the remaining authors. The first author has developed a hardware platform and performed the corresponding task mapping.

- Chenxin Zhang, Thomas Lenart, Henrik Svensson, and Viktor Öwall, "Design of Coarse-Grained Dynamically Reconfigurable Architecture for DSP Applications," in *Proceedings of International Conference on Reconfigurable Computing and FPGAs*, Cancun, Mexico, December 2009.

  Contribution The research work has been performed by the first author in the guidance of the remaining authors.

I have also contributed to the following articles, which are not directly included in the thesis:

- Fengbo Ren, Chenxin Zhang, Liang Liu, Wenyao Xu, Viktor Öwall, and Dejan Marković, "A Modified Square-Root-Free Matrix Decomposition Method for Efficient Least Square Computation on Embedded Systems," under revision at *IEEE Embedded Systems Letters*.

  Contribution The research work has been performed by the first and the second authors in the guidance of the remaining authors. The author has carried out performance analysis of the proposed algorithm.

- Per Andersson, Krzysztof Kuchcinski, Chenxin Zhang, and Jörn W. Janneck, "Beyond von Neumann: Weakly Programmable Processor Arrays and Their Programming," in *Proceedings of First International Software Technology Exchange Workshop*, Stockholm, Sweden, November 2010.

  Contribution The author has designed a hardware platform used for software development carried out by the remaining authors.

The research works included in this thesis have been carried out in the *Reconfigurable Computing* project sponsored by VINNOVA Industrial Excellence Center - System Design on Silicon (IXC SoS), the *High Performance Embedded Computing (HiPEC)* project sponsored by Swedish Foundation for Strategic

Research (SSF), the *Scalable Multi-tasking Baseband for Mobile Communications (Multibase)* project sponsored by Seventh Framework Programme for Research (FP7) European Union, and the *Flexible Embedded Platforms for ELLIIT Applications* project sponsored by Excellence center at Linköping-Lund in Information Technology (ELLIIT) research center.

Acknowledgement

This five-year PhD journey has been adventurous, challenging and exciting. It would not have been so fruitful, joyful and rewarding without many people's help and support.

First and foremost, I would like to show my deep gratitude to my main supervisor Professor Viktor Öwall, who has guided and encouraged me since I was a master student. Without his endless support and caring, I would never have been able to come to this moment. He has not only made invaluable advices on my studies, but also given me great opportunities to gain international research experiences. Over the years, his door was always open whenever I had questions and problems (not only academic ones). I will truly miss your company in my PhD journey (including the joyful drive in Hollywood and Santa Monica). I am indebted to Assistant Professor Liang Liu, who is not only a great supervisor but also a true friend. I will never forget your incredible patience in reading and commenting my works and teaching me about technical writings, as well as many joyful beer & whisky evenings. I am also in gratitude to Dr. Thomas Lenart, who guided me stepping into this fantastic research area six years ago. Ever since then, he has provided me with all possible supports, from commenting on my works to guidance on my career development. I would also like to show my great thanks to Professor Ove Edfors for his tremendous help in the field of communication. Special appreciation goes to his good sense of beer during the trip in Shanghai.

Life as a graduate student comprises hard work and joyful moments with colleagues and friends. I want to thank Assistant Professor Per Andersson, Associate Professor Erik Larsson, Professor Peter Nilsson, and Associate Professor Joachim Rodrigues, for all the constructive guidance and help throughout my PhD studies; Dr. Isael Diaz, Stefan Granlund, Hemanth Prabhu, and Meifang Zhu, for your excellent teamwork; Dr. Deepak Dasalukunte, for taking care of my plants; Dr. Johan Löfgren, for urging me to "Get back to work!"; Reza Meraji and Dr. Yasser Sherazi, for many "scientific discussions"; Katarzyna Burzynska, Mingfa Ding, and Dr. Ping Lu, for supporting and caring as friends. I have also enjoyed many talks with Dr. Mattias Andersson, Oskar Andersson, Rakesh Gangarajaiah, Breeta SenGupta, Erik Hertz, Xiaodong Liu, Yangxurui Liu, Steffen Malkowsky, Babak Mohammadi, Christoph Müller, Dimitar Nikolov, Dejan Radjen, Michal Stala, Xiang Gao, and Farrokh Ghani Zadegan. All my friends and colleagues besides above mentioned deserve thanks.

My work would have been much more complicated if it were not for the administrative and technical staff. I would especially like to thank Anne Andersson, Pia Bruhn, Doris Glöck, Robert Johnsson, Erik Jonsson, Bertil Lindvall, Stefan Molund, Martin Nilsson, and Josef Wajnblom.

I am in great gratitude to Associate Professor Dejan Marković for hosting me at the Department of Electrical Engineering, University of California, Los Angeles (UCLA) from Oct. 2012 to Feb. 2013. I would like to thank him for the fruitful weekly discussions and for giving me opportunities to gain experience from industry research. I would like to extend my gratitude to the colleagues and friends at UCLA, especially Fengbo Ren, Cheng C. Wang, Wenyao Xu, and Fang-Li Yuan. Special thanks to Fengbo for his great help academically and in life in general. Thank you for the unforgettable spring festival dinner.

I would like to express gratitudes from the deepest point of my heart to my parents. Thank you for your unconditional love, endless support and constant encouragement. I am also grateful to my parents-in-law for their great support and care.

Last but not the least, to my wife Lu. I am so glad to have you in my life. Your love has made me what I am today.

Lund, April 22, 2014

Chenxin Zhang

List of Acronyms

ALU	Arithmetic Logic Unit
ASIC	Application-Specific Integrated Circuit
ASIP	Application-Specific Instruction-set Processor
AWGN	Additive White Gaussian Noise
BCJR	Bahl-Cocke-Jelinek-Raviv
BPSK	Binary Phase-Shift Keying
CFO	Carrier Frequency Offset
CGRA	Coarse-Grained Reconfigurable Architecture
CISC	Complex Instruction Set Computing
CMAC	Complex-valued Multiply-ACcumulate
CMOS	Complementary Metal-Oxide-Semiconductor
CORDIC	COordinate Rotation DIgital Computer
CP	Cyclic Prefix
CSI	Channel State Information
CUDA	Compute Unified Device Architecture
CVG	Candidate Vector Generation
DFE	Digital Front-End
DLP	Data-Level Parallelism
DMA	Direct Memory Access
DSP	Digital Signal Processor
DVB	Digital Video Broadcasting
DVB-H	Digital Video Broadcasting for Handheld

ED	Euclidean Distance
EPA	Extended Pedestrian A
EQD	EQually Distributed
ETU	Extended Typical Urban
EVA	Extended Vehicular A
FEC	Forward Error Correction
FER	Frame Error Rate
FFT	Fast Fourier Transform
FIFO	First In First Out
FNE	Fast Node Enumeration
FPGA	Field-Programmable Gate Array
FSD	Fixed-complexity Sphere Decoder
FSM	Finite-State Machine
GALS	Globally Asynchronous Locally Synchronous
GOPS	Giga Operations Per Second
GPGPU	General-Purpose computing on Graphics Processing Unit
GPP	General Purpose Processor
GPR	General Purpose Register
GPS	Global Positioning System
GPU	Graphics Processing Unit
GR	Givens Rotation
GSM	Global System for Mobile communications
HDL	Hardware Description Language
ICI	Inter-Carrier-Interference
i.i.d.	Independent and Identically Distributed
ILP	Instruction-Level Parallelism

IMD	IMbalanced Distributed
IP	Intellectual Property
ISA	Instruction Set Architecture
ISI	Inter-Symbol-Interference
LS	Least Square
LSB	Least Significant Bit
LTE	Long Term Evolution
LTE-A	Long Term Evolution-Advanced
LUT	Look-Up Table
MAC	Multiply-ACcumulate
MGS	Modified Gram-Schmidt
MIMO	Multiple-Input Multiple-Output
ML	Maximum-Likelihood
MMR	Matrix Mask Register
MMSE	Minimum Mean-Square Error
MSB	Most Significant Bit
MSE	Mean Squared Error
NFC	Near Field Communication
NoC	Network-on-Chip
NRE	Non-Recurring Engineering
OFDM	Orthogonal Frequency Division Multiplexing
PC	Program Counter
PDP	Power-Delay Profile
QAM	Quadrature Amplitude Modulation
QoS	Quality of Service
QRD	QR Decomposition

RAM	Random Access Memory
RC	Resource Cell
RISC	Reduced Instruction Set Computing
ROM	Read-Only Memory
RTL	Register Transfer Level
SCENIC	SystemC Environment with Interactive Control
SD	Sphere Decoder
SDBG	Serial DeBuG
SIMD	Single Instruction Multiple Data
SNR	Signal-to-Noise Ratio
SPE	Successive Partial node Expansion
SQRD	Sorted QR Decomposition
SRAM	Static Random-Access Memory
SSFE	Selective Spanning with Fast Enumeration
STS	Short Training Symbol
TDM	Time Division Multiplexing
TLP	Thread-Level Parallelism
UART	Universal Asynchronous Receiver/Transmitter
UMTS	Universal Mobile Telecommunications System
VDP	Vector Dot Product
VHDL	Very High Speed Integrated Circuit (VHSIC) HDL
VLIW	Very Long Instruction Word
VLSI	Very-Large-Scale Integration
VPR	Vector Permutation Register
WCDMA	Wideband Code Division Multiple Access
ZF	Zero-Forcing

List of Definitions and Mathematical Operators

$(\cdot)^*$	Complex conjugate
$(\cdot)^T$	Vector/matrix transpose
$(\cdot)^H$	Hermitian transpose
$(\cdot)^\dagger$	Matrix pseudo-inverse
$(\cdot)_i$	Column vector
$(\cdot)_{i,i}$	$(i,i)^{th}$ matrix element
$\lvert\cdot\rvert$	Euclidean vector length
$\lVert\cdot\rVert_2$	ℓ^2-norm
$\bigodot$	Element-wise vector multiplication
∞	Infinity
$\propto$	Proportional
$\approx$	Approximation
$\mathcal{O}$	Computational complexity
$\lceil x \rceil$	Ceiling function. Rounds x to nearest integer towards ∞
$\lfloor x \rfloor$	Floor function. Rounds x to nearest integer towards $-\infty$
$x \in A$	The element x belongs to the set A
$\mathcal{Q}(x)$	Slicing function in symbol detection, returning a constellation point nearest to x

j	Imaginary unit
$\boldsymbol{I}$	Identity matrix
θ	OFDM symbol start
ε	Fractional carrier frequency offset
N_c	Number of OFDM subcarriers
Δf	OFDM subcarrier spacing
N	Number of antennas
M	Constellation size
$\boldsymbol{n}$	i.i.d. complex Gaussian noise vector
σ_n^2	Variance of $\boldsymbol{n}$
η	Post-detection SNR
s	Technology scaling factor
$\boldsymbol{H}$	Complex-valued MIMO channel matrix
$\boldsymbol{Q}$	Unitary matrix in QR decomposition
$\boldsymbol{R}$	Upper triangular matrix in QR decomposition
$\boldsymbol{P}$	Permutation matrix in sorted-QR decomposition
N_{SW}	Frequency correlation window in R.MMSE-SW
$\boldsymbol{\Omega}$	Node perturbation parameter in MMSE-NP

Chapter 1

Introduction

This thesis discusses an interdisciplinary study in wireless communication and Very-Large-Scale Integration (VLSI) design, more specifically, implementation of digital baseband processing using reconfigurable architectures. Development of such kind of systems, sometimes referred to as baseband processors [1] or Software Defined Radio (SDR) platforms [2], is an important and challenging subject, especially for small-scale base stations (e.g., femtocells) and mobile terminals that must provide reliable services under various operating scenarios with low power consumption.

The importance of the subject is driven by two facts. First, there is a huge demand for wireless communication in the world. The number of devices connected to the Internet in one way or the other is expected to reach 50 billion by 2020 [3,4]. In other words, every person on earth will have around six devices on average. Second, the number of radio standards grows increasingly fast in order to suffice ever-growing user demands such as date rate. For example, compared to the world's first hand-held device demonstrated in 1973, today's fourth-generation (4G) mobile terminals are able to process not only voice and text but also data streaming with the speed of up to gigabit-per-second [5]. Moreover, modern wireless systems need to be backward compatible to support 2G Global System for Mobile communications (GSM) and 3G Universal Mobile Telecommunications System (UMTS), as well as to support a range of different radio standards for improving user experience. Examples of these standards are bluetooth, IEEE 802.11 series, Global Positioning System (GPS), and Near Field Communication (NFC). As envisioned in [6], a single 4G mobile terminal needs to support more than 10 radio standards with tens of operation modes in each standard (e.g., 63 for 3GPP Long Term Evolution). Using

traditional implementation strategies, equipping each of these standards with an Application-Specific Integrated Circuit (ASIC), becomes antiquated and unaffordable with regard to area consumption and development time. Besides, it is unlikely that a user will enable all of these standards at the same time in a single terminal. Thus, there is a need for a flexible hardware platform capable of sharing resources among multiple standards and tasks and allocating resources dynamically to suffice current computational demands.

In addition to the multi-standard multi-task support, flexibility is required to cope with the rapid evolution of baseband processing algorithms and enable run-time algorithm adaptions to provide better Quality of Service (QoS) and maintain robust, reliable, and seamless connectivity. Furthermore, benefiting from the hardware reconfigurability, such architectures have the potential to perform system updates and bug-fixes while the system is in operation. This feature will prolong product life-time and ensure benefits in terms of time-to-market [1,7,8]. Last but not the least, from an algorithm development perspective, reconfigurable computing provides a more software-centric programming approach. This allows hardware platforms to be developed on-demand and potentially in the same language as used for software development. Unified programming environment enhances productivity by simplifying system integration and verification.

Besides its importance, the target subject faces many design challenges in practical implementations, such as requirements of high computational performance and low energy consumption. Primary concerns for contemporary system designs are shifting from computational performance to energy efficiency [9, 10]. This trend becomes more and more prominent in wireless communication designs. For example, the transition from 3G to 4G wireless communication systems demands 3 orders of magnitude increase in computational complexity, whereas the total power budget remains approximately constant in a single mobile terminal [11, 12]. Reconfigurable architectures, since its invention in 1960 [13], promise to offer great hardware flexibility and computational performance. They allow run-time hardware reconfigurations to accelerate arbitrary algorithms, and thus extend the application domain and versatility of the device. However, due to huge routing overhead, they cannot match power and area efficiency of ASICs, in spite of their tremendous developments over the past decades. As an example, fine-grained interconnects in commercial Field-Programmable Gate Arrays (FPGAs) consume over 75% of the chip area [14], and cause $17-54$ times area overhead and $5.7-62$ times more power consumption in comparison to ASICs [15]. Moreover, bit-level function blocks of FPGAs incur additional area and power penalties when implementing word-level computations. The area and power overhead have restricted the usage of reconfigurable architectures in cost-sensitive applications such as wireless com-

munication in mobile terminals. To address these overhead issues, new types of reconfigurable architectures with coarse-grained function blocks have gained increasing attention in recent years in both academia and industry [16–21].

This thesis presents a coarse-grained dynamically reconfigurable cell array architecture, which is designed and tailored with a primary focus on digital baseband processing in wireless communication. By exploiting the computational characteristics of the target application domain, the proposed domain-specific cell array architecture bridges the gap between ASICs and conventional reconfigurable platforms. The flexibility, performance, and hardware efficiency of the cell array are demonstrated through case studies.

1.1 Scope of the Thesis

The goal of the research project is to find efficient reconfigurable architectures that can provide a balance among computational capability, flexibility, and hardware efficiency. The driving application for hardware developments and performance evaluations is digital baseband processing in wireless communication. The target platform is small-scale base stations and mobile terminals, which need to provide real-time performance with restricted budgets of physical size and energy dissipation.

The central part of the thesis is the presentation of a dynamically reconfigurable cell array architecture which is the main result of the work in this project. Performance of the cell array is evaluated through two case studies, which are conducted to address two following questions:

- Can the cell array be used for multi-standard and multi-task processing? Is the control overhead affordable?
- Can the cell array meet real-time requirements when performing sophisticated baseband processing tasks? Under such a use case, what is the area and energy efficiency in comparison to ASICs and conventional reconfigurable architectures?

Throughout the work and by conducting algorithm-architecture co-design, special attention is paid to four distinct areas of the cell array design:

- System architecture design, including various processing elements, memory sub-systems, Network-on-Chip (NoC), and dynamic reconfiguration.
- Design flow of the cell array.
- Design trade-offs, including selection of processing elements and accelerators, task partitioning between hardware and software as well as between processing elements and memory sub-systems.

- Instruction set and function descriptor design for various processing elements and memory sub-systems, respectively.

The main focus of the thesis is the architectural design of the cell array architecture. Thus, system-level explorations, application mapping, and task scheduling are not addressed. The former subject has been studied in previous work [7, 8] carried out in the same group of the author. A SystemC-based exploration environment SCENIC was constructed specifically for this purpose. The latter two subjects are currently carried out in a joint research project with the Department of Computer Science, Lund University.

Digital baseband processing in wireless communication systems includes many tasks such as Orthogonal Frequency Division Multiplexing (OFDM) modulation/demodulation, Multiple-Input Multiple-Output (MIMO) signal processing, Forward Error Correction (FEC), interleaving, scrambling, etc. Among these, this thesis focuses on four crucial blocks in a typical baseband processing chain at the receiver, i.e., Digital Front-End (DFE), channel estimation, channel pre-processing, and symbol detection. However, the same design methodology is applicable for other baseband processing blocks and applications.

1.2 Contributions and Thesis Outline

The thesis is divided into four parts. Chapter 2 and 3, belonging to a background part, serve to give an overview of the research field. Chapter 2 discusses reconfigurable architectures and various processing alternatives. Chapter 3 covers typical digital baseband processing tasks in contemporary wireless communication systems. These two introductory chapters are not intended to give detailed descriptions on each of the subject. They are presented to give reference information on terms and concepts used later in the thesis.

Part I introduces the proposed coarse-grained dynamically reconfigurable cell array architecture, including both system infrastructure and a hardware design flow. Using the cell array as a baseline architecture, Part II and III present two case studies to demonstrate the performance of the proposed domain-specific reconfigurable cell array. The two studies are conducted in accordance to the processing flow of a typical baseband processing chain at the receiver. In addition, the two case studies manifest architectural evolution of the cell array, namely from scalar- to vector-based architecture.

Part I: The Reconfigurable Cell Array

Conventional fine-grained architectures, such as Field-Programmable Gate Arrays (FPGAs), provide great flexibility by allowing bit-level manipulations in system designs. However, the fine-grained configurability results in long config-

uration time and poor area and power efficiency, and thus restricts the usage of such architectures in time-critical and area/power-limited applications. To address these issues, recent work focuses on coarse-grained architectures, aiming to provide a balance between flexibility and hardware efficiency by adopting word-level data processing. In this part, a coarse-grained dynamically reconfigurable cell array architecture is proposed. The architecture is constructed from an array of heterogeneous functional units communicating via hierarchical network interconnects. The strength of the architecture lies in the simplified data sharing achieved by decoupled processing and memory cells, the substantial communication cost reduction obtained by a hierarchical network structure, and the fast context switching enabled by a unique run-time reconfiguration mechanism.

The content of this part is based on the following publication:

- Chenxin Zhang, Thomas Lenart, Henrik Svensson, and Viktor Öwall, "Design of Coarse-Grained Dynamically Reconfigurable Architecture for DSP Applications," in *Proceedings of International Conference on Reconfigurable Computing and FPGAs*, Cancun, Mexico, December 2009.

Part II: Multi-standard Digital Front-End Processing

This part aims at demonstrating the flexibility of the reconfigurable cell array architecture and evaluate the control overhead of hardware reconfigurations, in terms of clock cycles and area consumption. For this purpose, the cell array is configured to concurrently process multiple radio standards. Flexibility of the architecture is demonstrated by performing time synchronization and Carrier Frequency Offset (CFO) estimation in a digital front-end receiver for multiple OFDM-based standards. As a proof-of-concept, this work focuses on three contemporarily widely used radio standards, 3GPP Long Term Evolution (LTE), IEEE 802.11n, and Digital Video Broadcasting for Handheld (DVB-H). The employed reconfigurable cell array, containing 2×2 resource cells, supports all three standards and is capable of processing two concurrent data streams. Dynamic configuration of the cell array enables run-time switching between different standards and allows adoption of different algorithms on the same platform. Thanks to the adopted fast configuration scheme, context switching between different operation scenarios requires at most 11 clock cycles.

The content of this part is based on the following publications:

- Chenxin Zhang, Isael Diaz, Per Andersson, Joachim Neves Rodrigues, and Viktor Öwall, "Reconfigurable Cell Array for Concurrent Support of Multiple Radio Standards by Flexible Mapping," in *Proceedings of IEEE International Symposium on Circuits and Systems*, Rio de Janeiro, Brazil, May 2011.

- Isael Diaz, Chenxin Zhang, Lieven Hollevoet, Jim Svensson, Joachim Neves Rodrigues, Leif Wilhelmsson, Thomas Olssson, Liesbet Van der Perre, and Viktor Öwall, "A New Digital Front-End for Flexible Reception in Software Defined Radio," submitted to *Microprocessors and Microsystems: Embedded Hardware Design*.
- Isael Diaz, Chenxin Zhang, Lieven Hollevoet, Jim Svensson, Joachim Neves Rodrigues, Leif Wilhelmsson, Thomas Olssson, Liesbet Van der Perre, and Viktor Öwall, "Next Generation Digital Front-End for Multi-Standard Concurrent Reception," in *Proceedings of NORCHIP*, Vilnius, Lithuania, November 2013.

Part III: Multi-task MIMO Signal Processing

This part aims at demonstrating the flexibility and real-time processing capability of the cell array as well as evaluating the area and energy efficiency when performing sophisticated baseband processing tasks. The outcome of this work accounts for a large portion of this thesis.

Driven by the requirement of multi-dimensional computing in contemporary wireless communication technologies, reconfigurable platforms have come to the era of vector-based architectures. In this part, the reconfigurable cell array is extended with extensive vector computing capabilities, aiming for high-throughput baseband processing in MIMO-OFDM systems. Besides the heterogeneous and hierarchical resource deployments, a vector-enhanced SIMD structure and various memory access schemes are employed. These architectural enhancements are designed to suffice stringent computational requirements while retaining high flexibility and hardware efficiency. To demonstrate its performance and flexibility, three computationally intensive blocks, namely channel estimation, channel pre-processing, and symbol detection, of a 4×4 MIMO processing chain in a 20 MHz 64-QAM Long Term Evolution-Advanced (LTE-A) downlink are mapped and processed in real-time.

The content of this part is based on the following publications:

- Chenxin Zhang, Liang Liu, Dejan Marković, and Viktor Öwall, "A Heterogeneous Reconfigurable Cell Array for MIMO Signal Processing," submitted to *IEEE Transactions on Circuits and Systems-I: Regular Papers*.
- Chenxin Zhang, Liang Liu, Yian Wang, Meifang Zhu, Ove Edfors, and Viktor Öwall, "A Highly Parallelized MIMO Detector for Vector-Based Reconfigurable Architectures," in *Proceedings of IEEE Wireless Communications and Networking Conference*, Shanghai, China, April 2013.
- Chenxin Zhang, Liang Liu, and Viktor Öwall, "Mapping Channel Estimation and MIMO Detection in LTE-Advanced on a Reconfigurable Cell Array," in *Proceedings of IEEE International Symposium on Circuits and Systems*, Seoul, Korea, May 2012.

- Stefan Granlund, Liang Liu, Chenxin Zhang, and Viktor Öwall, "A Low-Latency High-Throughput Soft-Output Signal Detector for Spatial Multiplexing MIMO Systems," submitted to *Microprocessors and Microsystems: Embedded Hardware Design*.
- Stefan Granlund, Liang Liu, Chenxin Zhang, and Viktor Öwall, "Implementation of a Highly-Parallel Soft-Output MIMO Detector with Fast Node Enumeration," in *Proceedings of NORCHIP*, Vilnius, Lithuania, November 2013.
- Chenxin Zhang, Hemanth Prabhu, Liang Liu, Ove Edfors, and Viktor Öwall, "Energy Efficient SQRD Processor for LTE-A Using a Group-sort Update Scheme," in *Proceedings of IEEE International Symposium on Circuits and Systems*, Melbourne, Australia, June 2014.
- Chenxin Zhang, Hemanth Prabhu, Liang Liu, Ove Edfors, and Viktor Öwall, "Energy Efficient MIMO Channel Pre-processor Using a Low Complexity On-Line Update Scheme," in *Proceedings of NORCHIP*, Copenhagen, Denmark, November 2012.

Chapter 2

Digital Hardware Platforms

Since the invention of the integrated circuit in the 1950's, there has been explosive developments of electronic circuits. Over the last decades, the amount of transistors, which are the fundamental elements of digital and analog circuits, fitting on a single silicon die has increased exponentially, from a few thousands to billions to date. This trend was already observed in 1965 [22] by Intel's co-founder Gordon E. Moore and later came to be known as "Moore's law" coined by Carver Mead. Moore's law has held true since then and is a driving force of the advancements of Very-Large-Scale Integration (VLSI) design [23].

Enabled by the technology advancements, various forms of hardware platforms emerged to cater to a variety of applications. Depending on design trade-offs between flexibility and efficiency, these platforms can be broadly divided into three classes, namely *programmable processors*, *reconfigurable architectures*, and *Application-Specific Integrated Circuits (ASICs)*. Programmable processors include, for example, General-Purpose Processors (GPPs) and Application Specific Instruction Set Processors (ASIPs). Reconfigurable architectures differ from the programmable processors in the way that they expose both data and control path to the user and are "programmable" through hardware configurations. Field-Programmable Gate Array (FPGA) is a well-recognized example of this architecture category. ASICs are customized designs with limited flexibility. Hardware modifications after chip fabrication for new function adoption is barely possible for this type of platforms. They are commonly used in time- and power-critical systems, where flexibility is not a primary concern. Figure 2.1 illustrates a general view of how these three classes of platforms fare in the flexibility-efficiency design space. It should be pointed out that

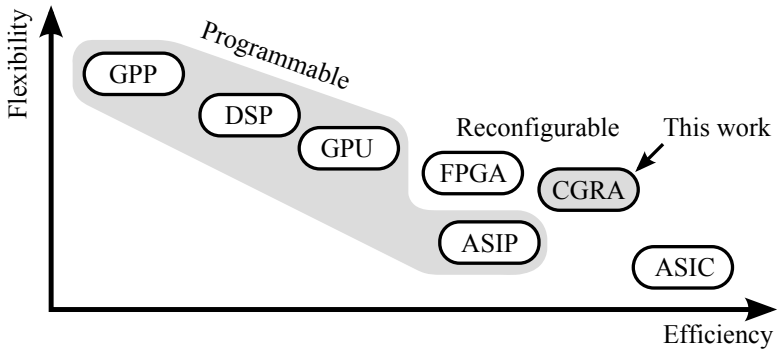


Figure 2.1: Comparison of flexibility and efficiency for various forms of hardware platforms. This work focuses on the design of Coarse-Grained Reconfigurable Architectures (CGRAs).

comparison of particular architecture instances among these classes has become increasingly obscure because of huge architecture varieties and different optimization objectives such as application domains and speed grades. Thus, Figure 2.1 only serves to give an overview of how different platforms trade flexibility for efficiency. Flexibility, including programmability and versatility, is measured as the ability to adopt a platform into different application domains and to perform different tasks. For instance, GPPs are highly flexible platforms since they are designed without having any particular application in mind. Efficiency relates to both computational performance and energy consumption and is a measure of how well a platform performs in an application. In this context, ASICs reveal the highest efficiency because of hardware customizations. This work focuses on Coarse-Grained Reconfigurable Architectures (CGRAs), aiming to bridge the flexibility-efficiency gap between ASICs and the other two classes of platforms, illustrated in Figure 2.1.

2.1 Programmable Processors

Programmable processors are designed based on instruction sets, which are specifications of operation codes (opcodes) used to conduct operations of underlying hardware elements. Depending on design objectives, an instruction set can be optimized with respect to, for example, application domain and friendliness to high-level programming constructs [24]. Some examples of Instruction Set Architecture (ISA) categories are Complex Instruction Set Computing (CISC), Reduced Instruction Set Computing (RISC), and Very Long Instruction Word (VLIW).

Based upon the retargetability of the instruction set, programmable processors can be categorized into fixed and configurable ISAs. Compared to the latter one, fixed ISAs are easy to design and can be optimized for obtaining high performance such as high clock frequency by deep pipelining [24]. Examples of fixed ISAs are GPPs, special-purpose processors, and ASIPs. Configurable ISAs provide the user flexibilities in selecting appropriate instructions for target applications. This way, the ISAs can be customized to attain higher efficiency in comparison to fixed ISAs. However, this instruction set customizability complicates the design of baseline architecture and software tool chain (e.g., compiler and emulator).

General-Purpose Processors

GPPs are highly programmable and support any algorithm that can be compiled to a computer program. Thus, they are dominantly used in personal computers. Although GPPs have always been implemented with the latest semiconductor technology in order to achieve the highest possible processing speed, they suffer from a performance bottleneck: the sequential nature of program execution. To address this issue, many design techniques have been proposed, which range from ISA to microarchitecture design with a goal of increasing the number of executed instructions per second. Examples of these techniques are superscalar and VLIW architectures for exploiting *Instruction-Level Parallelism (ILP)*, Single Instruction Multiple Data (SIMD) architectures (e.g., Intel's Pentium MMX and AMD's 3DNow! ISA) for enabling *Data-Level Parallelism (DLP)*, and multithreading technology (e.g., Intel's hyper-threading [25]) for providing *Thread-Level Parallelism (TLP)*. Furthermore, GPPs have shifted to a multi-core paradigm due to energy and power constrains on growth in computing performance [26]. Figure 2.2 shows the slowdown in processor performance growth, clock speed, and power consumption, as well as the continued exponential growth in the number of transistors per chip [26].

Special-Purpose Processors

Special-purpose processors are designed to be used for a particular application domain. Well-known examples are Digital Signal Processors (DSPs) and Graphic Processing Units (GPUs). DSPs are designed for performing digital signal processing tasks such as filtering and transforms. Commonly used operations in signal processing algorithms are accelerated in DSPs. An example is multiplication followed by accumulation, widely used in digital filters [27]. This operation is performed using dedicated Multiply-ACcumulate (MAC) units in DSPs and usually takes one clock cycle to execute. Other commonly used operations include various addressing modes such as modulo and ring-buffer.

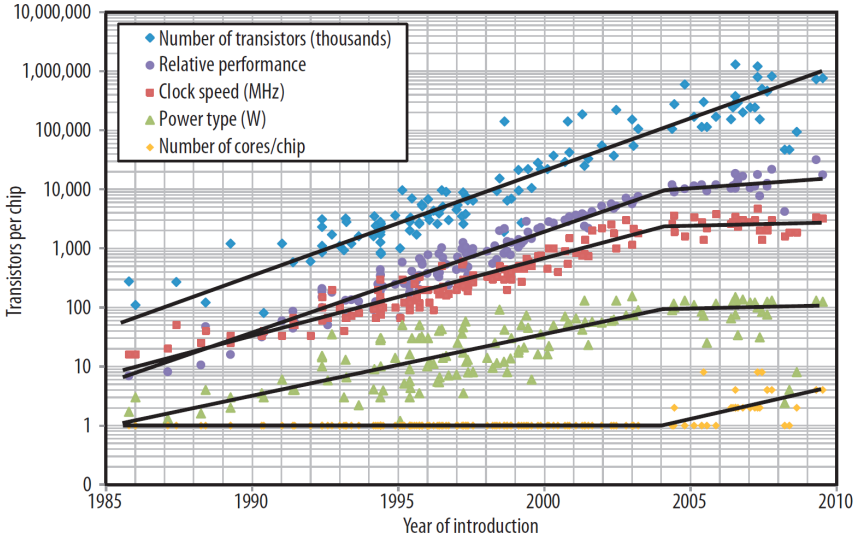


Figure 2.2: Transistors, frequency, power, performance, and processor cores over time [26].

GPUs are specialized computational units dedicated to manipulating computer graphics. Thanks to their highly parallel structure (e.g., containing hundreds of processing cores [28]), they are able to process large blocks of data in parallel. Taking advantage of the high processing capability, General-Purpose computing on GPUs (GPGPU) has recently gained in popularity. An example is the Compute Unified Device Architecture (CUDA) platform [29] from Nvidia, which supports C/C++ and Fortan programming on GPUs and can also be used for Matlab program accelerations [30].

Application-Specific Instruction Set Processors

Compared to DSPs and GPUs, Application-Specific Instruction set Processors (ASIPs) are optimized for a single application or a small groups of applications [31]. A general design flow is that a baseline processor, which could be a RISC processor or DSP, is extended with application-specific instructions. Besides, infrequently used instructions and function units are pruned, aiming to trade flexibility for energy and cost efficiency.

Configurable Instruction Set Processors

Different from the fixed ISAs, configurable instruction set processors provide users a collection of instructions and a baseline architecture containing various hardware features. Depending on target applications, users have the possibility of selecting appropriate instructions to construct a customized instruction set at design-time. Meanwhile, the microarchitecture of the processors can be customized by selecting, for example, different function units and the number of pipeline stages. Once the instruction set and the microarchitecture are fine tuned, hardware implementation of the processor is generated. From the hardware's point of view, the generated processor is a type of ASIP, however, with on-demand function customizations. Xtensa configurable cores [32] from Cadence (previously Tensilica) is an example of the configurable instruction set processor. Thanks to the instruction set and microarchitecture customizations, this type of processors provides high processing performance and hardware efficiency. However, design of the baseline architecture and the corresponding software support are more complicated than fixed ISAs, since they need to cover a huge set of configurations.

2.2 Application-Specific Integrated Circuits

Application-Specific Integrated Circuits (ASICs) are designed to perform specific tasks. Therefore, computational data paths and control circuits can be optimized for particular use cases. This brings ASICs to the far right of the design space in Figure 2.1, indicating that they are the most efficient (in terms of performance and energy consumption) type of platforms among the three classes. Therefore, ASICs are commonly used to achieve real-time performance within the budget for physical size and energy dissipation. However, the specialized hardware architecture limits the capability of adapting system to different applications and operation scenarios. This limitation results in reduced overall area efficiency in terms of hardware reuse and sharing. Additionally, this type of platforms requires a rather long hardware redesign time (for bugfixes or function updates) and exhaustive testing procedures. Furthermore, the exploding silicon design cost limits the adoption of ASICs, especially in deep sub-micro semiconductor technology.

2.3 Reconfigurable Architectures

Reconfigurable architectures are the ones having the capability of making substantial changes to the data path itself in addition to the control flow. This means that not only the software that runs on a platform is modified, but also how the hardware architecture operates [16–21]. With combined control and

data path manipulations, reconfigurable architectures are able to exploit potential parallelism, enable energy efficient computing, allow extensive hardware reuse, and reduce system design cycle and cost [7].

Reconfigurable architectures are either *homogeneous* or *heterogeneous*. In a homogeneous architecture, all elements contain the same hardware resources. This uniform structure simplifies the mapping of user applications, since additional constraints on function partitions and placements are avoided. However, homogeneous structures are inefficient in terms of hardware utilization of logic and routing resources [7]. In contrast, heterogeneous architectures contain array elements with different functionality, such as specialized elements for stream data processing or control-flow handling. Compared to the homogeneous structure, adoption of various types of array elements reduces hardware overhead and improves power efficiency at the cost of more complex mapping algorithms.

The size of the hardware elements inside a reconfigurable architecture is referred to as *granularity*. Fine-grained architectures and Coarse-Grained Reconfigurable Architectures (CGRAs) are two variants of reconfigurable architectures. Fine-grained architectures, such as FPGAs, are usually built up on small Look-Up Tables (LUTs). Such architectures have the ability to map any logic functions at bit-level onto their fine-grained lattice. However, this bit-oriented architecture results in a large amount of control and routing overhead, for example, when performing word-level computations. These overheads also affect power consumption and system configuration time. In contrast, CGRAs are constructed from larger building blocks in a size ranging from arithmetic logic units to full-scale processors. These hardware blocks communicate through a word-level routing network. The increased granularity in CGRAs reduces routing area overhead, improves configuration time, and achieves higher power efficiency despite less mapping flexibility. Besides, CGRAs differ from fine-grained architectures in design methodology. To map functionality into gates, FPGA designs rely on a hardware-centric approach, which usually requires programming in Hardware Description Language (HDL) such as VHDL. In contrast, CGRAs provide a more software-centric programming approach to map functionality to, for example, processing cores using a higher level language like C. Software-centric design approach enhances productivity and simplifies system integration and verification.

This thesis focuses on the development of CGRA, more specifically, domain-specific CGRA for baseband processing in wireless communication systems. Detailed architecture of the proposed CGRA-based dynamically reconfigurable cell array is presented in Part I with case studies in Part II and III.

2.4 A Comment on Power Efficiency

As mentioned in Chapter 1, primary concerns for contemporary system designs are shifting from computational performance to power efficiency [9,10]. Attaining high power efficiency is especially important for the target applications of this work, namely small-scale base stations and mobile terminals, since they are all constrained by stringent power requirements. Thus, it is crucial to have a better understanding of the composition of power consumption.

The total power consumption for a digital circuit built with Complementary Metal Oxide Semiconductor (CMOS) transistors may be expressed as [33]

$$P_{\text{total}} \sim \underbrace{\alpha \cdot (C_{\text{L}} + C_{\text{SC}}) \cdot {V_{\text{DD}}}^2 \cdot f}_{P_{\text{dynamic}}} + \underbrace{(I_{\text{DC}} + I_{\text{Leak}}) \cdot V_{\text{DD}}}_{P_{\text{leakage}}}, \tag{2.1}$$

where P_{total}, P_{dynamic}, and P_{leakage} represent the total, dynamic, and leakage power consumption, respectively. α is the switching activity of the circuit, C_{L} the load capacitance, C_{SC} the short circuit capacitance, V_{DD} the supply voltage, and f the clock frequency. I_{DC} and I_{Leak} represent the static and leakage current, respectively.

In the design of reconfigurable architectures, P_{dynamic} is usually a dominating factor because of high clock frequency and hardware utilization. In comparison, the leakage power is of less concern for such kind of architectures. However, it should be pointed out that leakage power is becoming more and more important with technology scaling and thus needs more attention. One of the well-known approaches for designing low power circuits is to reduce the quadratic term V_{DD}^2 in (2.1) at the cost of performance sacrifice such as clock frequency. To compensate for the performance loss, different techniques can be used such as pipelining and parallel processing [34] but at the expense of area consumption. Thus, it can be seen that designing hardware is a trade-off between various parameters among the design space.

Chapter 3

Digital Baseband Processing

Wireless communication has been experiencing explosive growth since its invention. The wireless landscape has been broadened by incorporating more than basic voice services and low data-rate transmissions. Taking cellular systems as an example, the fourth generation (4G) mobile communication technology promises to provide broadband Internet access in mobile terminals with up to gigabit-per-second downlink data-rate [5]. Compared to the 9.6 Kbit/s data services in its 2G predecessor Global System for Mobile communications (GSM), 4G systems enhances the data-rate by 5 orders of magnitude. This data-rate boost is a result of innovations in wireless technology, such as Orthogonal Frequency Division Multiplexing (OFDM) and Multiple-Input Multiple-Output (MIMO). The high speed data links together with advancements in mobile terminals (e.g., phones, tablet computers, and wearable devices) have opened up a whole new world for wireless communication and changed everyone's life. Besides conventional usage like Internet streaming and multimedia playback, interdisciplinary applications like mobile health (mHealth) [35] are emerging. New applications set new demands on wireless services, pushing forward technology developments.

This chapter aims to give a brief description of some modern wireless communication technologies and standards, introduce basic concepts and terminologies used in the rest of the thesis, and provide an overview of the digital baseband processing tasks in modern systems. Moreover, computational properties of baseband processing tasks are extracted in order to guide hardware developments. Basics of wireless communication, such as symbol modulation and propagation channels, are not addressed but can be found in [36, 37], since

the purpose of the present chapter is to highlight design challenges and point out baseband processing properties that can be exploited to achieve efficient hardware implementations. Worth mentioning is that this thesis work mainly focuses on MIMO-OFDM systems because of their importance and popularity in contemporary wireless communication systems. However, support of other wireless technologies is a natural extension and can be easily mapped onto the proposed reconfigurable cell array thanks to its flexible hardware infrastructure.

3.1 Wireless Communication Technologies

To increase the data-rate of a wireless system, a straightforward method is to allocate larger bandwidth for data communication. A wide frequency band allows for more data to be transferred at any time. This has been used as one of the main techniques in the transition from 2G to 3G systems, achieving ∼40 times data-rate speed-up by increasing bandwidth per carrier from 200 KHz to 5 MHz. This trend continues in 4G systems, which further expand bandwidth to 100 MHz with carrier aggregation. However, larger bandwidth increases implementation complexity. This is because multi-path propagation channels are by nature frequency selective [37], and thus affect signals at different frequency bands differently. OFDM technology [38] has been proposed to circumvent the issue of frequency selectivity. To further increase data-rate without the expansion of bandwidth, since bandwidth is a limited resource, spatial resources are utilized in addition to time and frequency. MIMO [39] is one such technology that provides various ways of utilizing spatial resources. In modern systems, the two aforementioned technologies are often used together, referred to as MIMO-OFDM systems.

Orthogonal Frequency Division Multiplexing

The key idea of OFDM is to divide a wideband channel into a number of narrowband sub-channels, over which the wideband signal is multiplexed. This way, the frequency response over each of these narrowband sub-channels is flattened, thus reducing the complexity of channel equalization. To enable parallel transmission over flat-fading sub-channels without interfering one another, adjacent narrowband sub-carriers need to be separated in frequency (Δf) and arranged such that they are orthogonal to each other. Figure 3.1 illustrates such arrangement. Because of the frequency overlapping, OFDM achieves high spectral efficiency.

In addition to the frequency selectivity, OFDM systems need to cope with channel effects as other wireless systems do. Wireless channels are characterized by multi-path propagation [37]. Signals travelling from one end to the other are reflected, diffracted and scattered by obstacles, forming multi-path

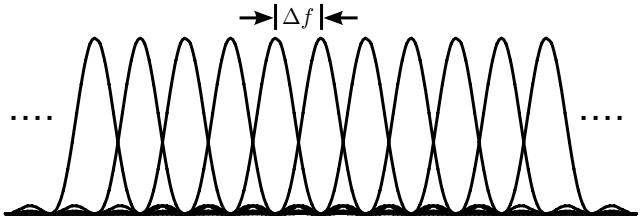


Figure 3.1: Orthogonal subcarriers in OFDM.

components. Depending on the travelled paths, multi-path components may arrive at the receiver at different time instances. The multi-path propagation will incur interferences between adjacent OFDM symbols generally referred to as Inter-Symbol-Interference (ISI). In addition, OFDM systems may suffer from Inter-Carrier-Interference (ICI), since the orthogonality of the subcarriers may be destroyed by multi-path propagation and imperfections in practical implementations such as carrier-oscillator mismatch.

To avoid both the ISI and ICI, each OFDM symbol is extended with a guard time interval designed to allow channel's impulse response to settle. This guard time interval is filled with a Cyclic Prefix (CP), which is a copy of the last part of each OFDM symbol. By discarding CP at the receiver after each symbol reception, given that the CP is long enough to cover the impulse response of the channel, the ISI and ICI can be completely avoided.

Multiple-Input Multiple-Output

MIMO is another important technology in modern wireless communication systems. Compared to single antenna setup, MIMO exploits resources in the spatial domain and provides significant improvements in system capacity and link reliability without increasing bandwidth. In MIMO, three main operation modes exist, namely spatial multiplexing [40], spatial diversity [41], and space division multiple access (also known as multi-user MIMO) [42]. These modes, illustrated in Figure 3.2, are designed to increase average user spectral efficiency, transmission reliability, and cell spectral efficiency, respectively [43]. To suffice ever-increasing user demands in Quality of Service (QoS) while living with the limited bandwidth resources, current trend in wireless systems is to adopt large MIMO dimensions. As an example, the maximum MIMO configuration in the transition from 3GPP Long Term Evolution (LTE) to its successor LTE-Advanced (LTE-A) is increasing from 4×4 to 8×8, while keeping the bandwidth unchanged.

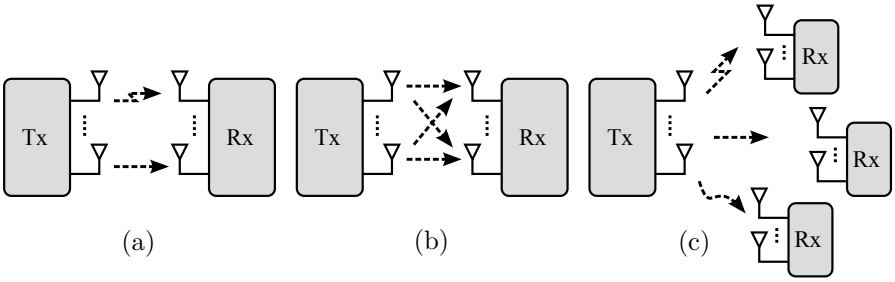


Figure 3.2: Three operation mode in MIMO, (a) spatial multiplexing, (b) spatial diversity, (c) space division multiple access (multi-user MIMO).

The benefits of MIMO entail a significant increase in signal processing complexity and power consumption at the receiver, where sophisticated signal processing is required, especially in a fading and noisy channel. For example, Channel State Information (CSI) between each pair of transmit and receive antennas should be properly estimated and symbol detection is needed to cancel inter-antenna interferences. As a result, efficient hardware implementation of MIMO receivers has become a critical challenge. Moreover, when combining MIMO with OFDM, it is required to perform the corresponding processing at every OFDM subcarrier, posing even more stringent computational and energy requirements.

3.2 Overview of Digital Baseband Processing

This section introduces baseband processing tasks in MIMO-OFDM systems. Figure 3.3 shows a simplified diagram of a typical MIMO-OFDM transceiver. Note that only digital baseband processing blocks are shown in the figure, whereas the RF front-end and Digital-to-Analog/Analog-to-Digital Converters (DACs/ADCs) are left out.

The receiver (Rx) chain is essentially the reverse processing of tasks performed at the transmitter (Tx). However, the receiver is usually more complex than the transmitter, since it has to reconstruct original data, which may be incomplete and distorted during wireless transmission. Some examples of distortions are noise, multi-path channel fading, and imperfections in the RF front-end.

Shaded blocks in Figure 3.3 are selected in this work as use cases for driving the development of the domain-specific reconfigurable cell array. These blocks are unique to the receiver chain and are key in determining the performance of the entire MIMO-OFDM system.

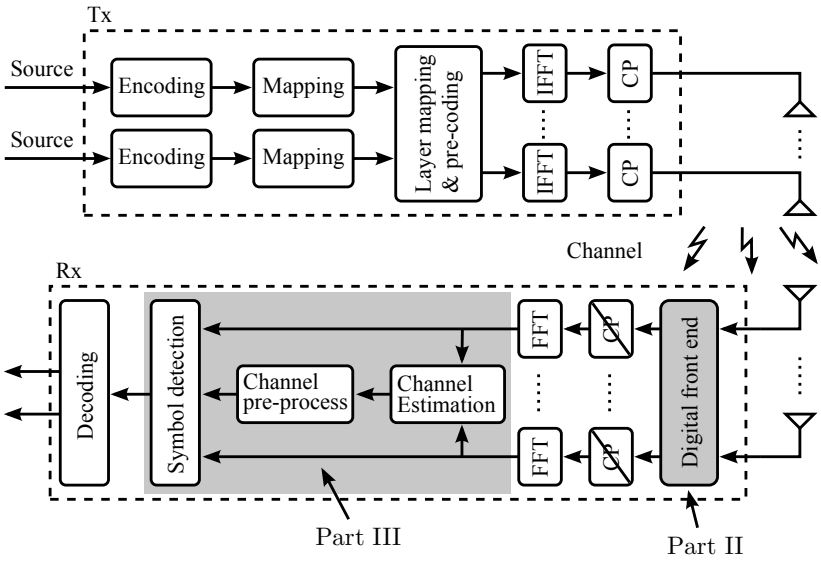


Figure 3.3: Block diagram of the MIMO-OFDM transceiver. This work focuses on mapping shaded blocks onto the dynamically reconfigurable cell array.

Channel Encoding/Decoding

The channel encoding block at the transmitter has two main tasks. First, binary data are encoded with error correcting code, such as convolutional codes, which adds redundant information to help receiver detect and correct a limited number of errors without retransmission. Second, encoded data are interleaved to make sure that adjacent bits are not transmitted consecutively in frequency. Interleaving improves transmission robustness with respect to burst errors. Additionally, scrambling is often used to turn the bit stream into a pseudo-noise sequence without long runs of zeros and ones [44].

Opposite to the encoding block, the channel decoder performs data deinterleaving, error correction, and descrambling. Among these, error correction, such as Low-Density Parity-Check (LDPC) code [45], Viterbi [46], and turbo decoding [47], are compute-intensive.

Symbol Mapping/Demapping

The encoded bit stream is sent for symbol mapping blocks at the transmitter, which are responsible for two tasks. First, in the symbol mapper, the bit stream

is mapped to a stream of symbols based on the adopted modulation scheme such as Quadrature Amplitude Modulation (QAM). Meanwhile, pilots are often added to the symbol stream. Pilots carry known information to both the transmitter and receiver and are used to perform, for example, synchronization and channel estimation at the receiver. Second, the layer mapping block maps the symbol stream onto multiple antennas.

The demapping block (not shown in Figure 3.3) demaps the symbol stream from multiple antennas, removes pilots, and demaps data-carrying symbols back to the binary bit stream.

Domain Transformation

Before sending data to the analog front-end at the transmitter, symbols from all narrowband subcarriers are collected and are simultaneously transformed to a time domain signal using an Inverse Fast Fourier Transform (IFFT). Thereafter, CP is added to each OFDM symbol to protect data transmission from being interfered by ISI and ICI.

At the receiver, CP is removed from each OFDM symbol. Fast Fourier Transform (FFT) is used to separate received time-domain signal back into their respective subcarriers.

Digital Front-End Processing

The Digital Front-End (DFE) is the first digital processing block in the receiver chain and is responsible for two main tasks [48]. The first is to detect an incoming signal by monitoring the amplitude of signal reception. Once a signal is detected, the DFE wakes up the remaining blocks in the baseband processing chain. Likewise, it puts other blocks into sleep mode when no signal is detected after a pre-defined time interval. The second task is to perform symbol synchronization to determine the exact timing of incoming OFDM symbols.

In addition to the aforementioned tasks, DFE is sometimes used to estimate and/or compensate some of radio impairments [48, 49], such as Carrier Frequency Offset (CFO), Signal-to-Noise Ratio (SNR), and IQ imbalance. Part II presents the mapping of DFE onto the reconfigurable cell array. Target processing tasks include OFDM time synchronization and CFO estimation.

Channel Estimation

To be able to recover transmitted data from the distorted signal reception, it is crucial to have the knowledge on how wireless channel "manipulates" (e.g., attenuates and rotates) the signal transmission. In fact, the performance gain of MIMO-OFDM systems heavily depends on the accuracy of CSI. Channel

estimation is used to estimate CSI based on either known information such as pilots and preambles or blind estimation algorithms.

Commonly used channel estimation algorithms are Least Squares (LS), Minimum Mean-Square Error (MMSE) and its derivatives, FFT, and Singular Value Decomposition (SVD) estimation. Among these, MMSE estimator provides the highest performance in terms of estimation accuracy, and LS has the lowest computational complexity. The work presented in Part III adopts an MMSE-based channel estimation algorithm, which provides a balance between performance and computational complexity.

Channel Pre-processing

The estimated channel matrix at each subcarrier needs to be further processed before being sent to the symbol detector. Depending on the adopted symbol detection algorithm, requirements on channel pre-processing may vary. Commonly used pre-processing algorithms include matrix inversion for linear detectors and QR Decomposition (QRD) for tree-search based detectors. Both of these algorithms are used in Part III.

Symbol Detection

In MIMO systems, detection is a joint processing of symbols from all spatial streams, since the symbols all contain a bit of the information after transmitting through the wireless channel. Therefore, the larger the MIMO dimension, the higher the computational complexity is involved in symbol detection. The basic task of a detection is to locate the transmitted data in a constellation diagram. However, since received data are contaminated by channel fading and noise, much effort needs to be spent in the detection process, especially for system operating at high-order modulation and large antenna numbers.

From the performance point of view, Maximum-Likelihood (ML) detection is an optimal detector that solves the closest-point search problem. However, ML detector is infeasible due to the exhaustive symbol search that is known to be NP-complete. Popular practical MIMO signal detection algorithms can generally be categorized into two classes, linear and tree-search based detectors, which all have certain performance sacrifice. Linear detection algorithms are preferred for real-time implementations owing to their low computational complexity. Additionally, they are characterized by high Data-Level Parallelism (DLP), since symbol detection at each spatial stream can be efficiently vectorized and performed in parallel. However, linear detection suffers from huge performance degradation compared to the optimal ML detection, especially for high dimensional MIMO systems. Alternatively, tree-search algorithms are getting much attention because of their near-ML performance. A tree-

search detection formulates a minimum-search procedure as a N-depth M-ary complex-valued tree search problem, where N and M are the antenna number and constellation size, respectively. Practical sub-optimal tree-search detectors solve the NP-complete problem of the optimal ML detection by only traversing through a number of branches. Examples of commonly used algorithms are sphere decoder, K-Best, and their derivatives [50–53]. One fundamental problem with tree-search algorithms is their intrinsic data dependence between adjacent layers, namely that symbol detection at the i^{th} layer is based on the results of $(i+1)^{th}$ layer. Therefore, the native vector structure of MIMO systems is destroyed, resulting in low DLP. In Part III, a vector-level closest point search algorithm in conjunction with linear detectors is proposed, which is highly vector-parallelized, like linear detectors, and at the same time, has the performance close to the level of tree-search detectors.

3.3 Baseband Processing Properties

Based on the analysis of aforementioned digital baseband processing tasks, three computational properties can be observed: *vast complex-valued computing*, *high data-level parallelism*, and *predictable control flow*. These properties should be exploited during the design of domain-specific reconfigurable cell array to ensure its hardware efficiency.

In the digital baseband processing chain shown in Figure 3.3, all blocks, except channel encoding/decoding, operate on I/Q pairs, which are represented in complex-valued data format. Thus, it is essential to design an Instruction-Set Architecture (ISA) that natively supports complex-valued computing, such as data types, data paths, instruction set, and memory access patterns.

A large portion of computations are performed using vectors, thanks to the parallel-structured MIMO streams. Such computations take place in processing blocks, namely FFT/IFFT, channel estimation, pre-processing, and symbol detection. The abundance of vector processing indicates extensive DLP, which can be utilized to improve processing throughput and reduce control overhead. Moreover, in view of the large number of subcarriers in OFDM, multi-subcarrier processing [54] can be carried out. By performing operations simultaneously on multiple subcarriers, multi-subcarrier processing further exploits DLP in addition to the ones obtained on the algorithm-level. This technique is extensively used in work presented in Part III and is proven to be useful and effective.

Observed from baseband processing tasks, there is no or little backward dependency between one another [1]. This makes control flow predictable and can therefore simplify the control path to reduce overhead.

Part I

The Reconfigurable Cell Array

Abstract

Emerging as a prominent technology, reconfigurable architectures have the potential of combining high hardware flexibility with high performance data processing. Conventional fine-grained architectures, such as Field-Programmable Gate Arrays (FPGAs), provide great flexibility by allowing bit-level manipulations in system designs. However, the fine-grained configurability results in long configuration time and poor area and power efficiency, and thus restricts the usage of such architectures in time-critical and area/power-limited applications. To address these issues, recent work focuses on coarse-grained architectures, aiming to provide a balance between flexibility and hardware efficiency by adopting word-level data processing. In this part, a coarse-grained dynamically reconfigurable cell array architecture is proposed. The architecture is constructed from an array of heterogeneous functional units communicating via hierarchical network interconnects. The strength of the architecture lies in simplified data sharing achieved by decoupled processing and memory cells, substantial communication cost reduction obtained by a hierarchical network structure, and fast context switching enabled by a unique run-time reconfiguration mechanism. The proposed reconfigurable cell array serves as a baseline architecture for two case studies presented in Part II and III.

1 Introduction

The evolution of user applications and increasingly sophisticated algorithms call for ever increasing performance of data processing. Meanwhile, to prolong system's operating time of battery operated devices, contemporary designs require low power consumption. A typical example is baseband processing in 4G mobile communication, which demands a computational performance of up to 100 Giga Operations Per Second (GOPS) with a power budget of around 500 mW in a single user terminal [6]. In addition to computational capability and power consumption, flexibility becomes an important design factor, since system platforms need to cope with various standards and support multiple tasks simultaneously. Therefore, it is no longer viable to dedicate a traditional application-specific hardware accelerator to each desired operation, as the accelerators are rather inflexible and costly in system development, validation, and maintenance (e.g., bug-fixes and function updates).

To achieve a balance among the aforementioned design requirements, reconfigurable architectures have gained increasing attention from both industry and academia. These architectures enable hardware reuse among multiple designs and are able to dynamically allocate a set of processing, memory, and routing resources to accomplish current computational demands. Moreover, reconfigurable architectures allow mapping of future functionality without additional hardware or manufacturing costs. Therefore, by using platforms containing reconfigurable architectures it is possible to achieve high hardware flexibility while sufficing the stringent performance and power demands [19].

Fine-grained and coarse-grained arrays are two main variants of reconfigurable architectures. While the former has the ability to map any logic functions at bit-level onto their fine-grained lattice, the latter is constructed from larger building blocks in a size ranging from arithmetic logic units to full-scale processors. Compared to fine-grained architectures, the increased granularity in Coarse-Grained Reconfigurable Architectures (CGRAs) reduces routing area overhead, improves configuration time, and achieves higher power efficiency despite less mapping flexibility.

This part proposes a coarse-grained dynamically reconfigurable cell array architecture, which will be used as a design template in the remaining part of the thesis. The cell array is a heterogeneous CGRA, containing an array of separated processing and memory cells, both of which are global resources distributed throughout the entire network. Array elements communicate with one another via a combination of local interconnects and a hierarchical routing network. All array elements are parametrizable at system design-time, and are dynamically reconfigurable to support run-time application mapping. The following summarizes the distinguished features of the architecture.

- The heterogeneity of the architecture allows integration of various types of resource cells into the array.
- Separation of processing and memory cells simplifies data sharing among resource cells.
- A hierarchical Network-on-Chip (NoC) structure combines high-bandwidth local communication with flexible global data routing.
- In-cell resource reconfigurations conducted by distributed processing cells enable fast run-time context switching.

It should be pointed out that the proposed cell array is a general architecture, which can be, in principle, used to map any algorithms, tasks, and applications. However, this work mainly focuses on signal processing in wireless communication, more specifically, digital baseband processing at the receiver. By exploiting computational properties of the target application domain, various architectural improvements can be carried out on the baseline architecture to further improve hardware performance and efficiency. Improvements will be illustrated through case studies in Part II and III. The present part serves to give an overview of the cell array, which includes the overall architecture, basic functionality and framework of each resource cell, network infrastructure, hardware reconfigurability, and design methodology.

The remainder of this part is organized as follows. Section 2 discusses related work with a focus on architectures designed specifically for digital wireless communication. Section 3 introduces the reconfigurable cell array architecture, presents details of each resource cell, and describes different ways of managing system configurations. Section 4 presents design flow for constructing a reconfigurable cell array. Section 5 summarizes this part.

2 Prior Work and State-of-the-art

A number of reconfigurable architectures have been proposed in open literature for a variety of application domains [16, 18–21, 55]. Presented architectures are characterized with various design parameters, such as granularity, processing and memory organization, coupling with a host processor, communication fabric, reconfigurability, and programming methodology. Describing each of the architectures with respect to those parameters is cumbersome and is in fact unnecessary because of architectural similarities. Instead, previously proposed architectures are classified into three broad categories based upon the coupling between processing and memory units and their interconnects. In addition, the following discussions are restricted to architectures designed specifically for digital baseband processing in wireless communication. These systems are

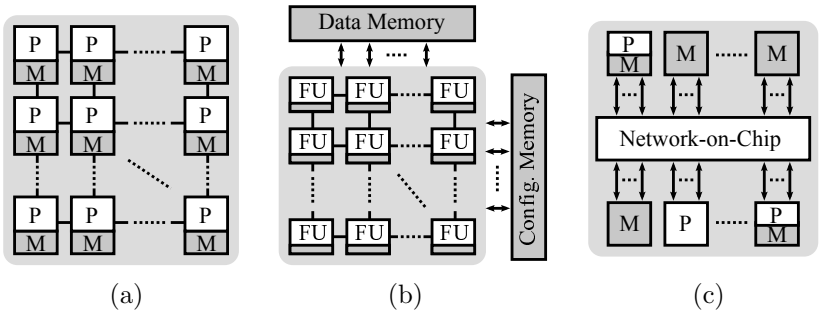


Figure 1: Three classes of reconfigurable architectures, (a) homogeneous processor array, (b) Function Unit (FU) cluster, (c) heterogeneous resource array.

sometimes referred to as baseband processors [1] or Software Defined Radio (SDR) platforms [2]. The three classes of reconfigurable architectures are illustrated in Figure 1.

The first group of architectures (Figure 1(a)), such as the PicoArray [56] from Picochip, the Signal processing On Demand Architecture (SODA) platform [57, 58] from the University of Michigan, Ann Arbor, Cadence's ConnX BaseBand Engine (BBE) [32], and Ninesilica platform [59] are constructed from an array of homogeneous processors. Each processor has exclusive access to its own memory. As an example, Figure 2 shows an instance of SODA architecture. It is made up of four cores, each containing asymmetric dual pipelines, for scalar and Single Instruction Multiple Data (SIMD) execution, and scratchpad memories. According to [20, 60], homogeneous architectures are not cost effective in supporting algorithms in which the workload cannot be balanced among multiple PEs, or algorithms involving hybrid data computations like scalar and various length vector processing. In these cases, all PEs cannot be fully utilized, resulting in reduced hardware efficiency. Additionally, the approach of integrating data memory inside PE results in difficulties when sharing data contents between surrounding elements. This is because PEs at both data source and destination are involved in data transmissions to load and store data contents from and into their internal memory. Consequently, these inter-core data transfers may take a significant amount of processing power, and in some cases may just turn PEs to act as memory access controllers, reducing the hardware usage. Moreover, storage capacities of data memories inside PEs are fixed after chip fabrication, which may reduce the flexibility and applicability of platforms.

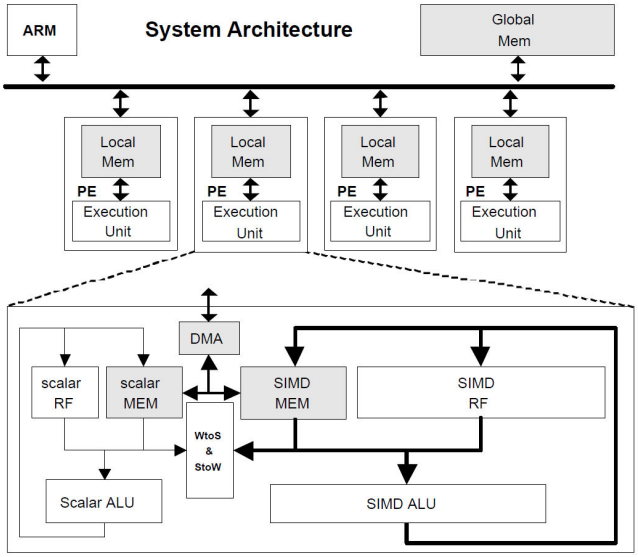


Figure 2: Overview of SODA architecture [57].

Architectures in Figure 1(b) are built from atomic functional units (FU), named as FU cluster. Examples of this group are the Architecture for Dynamically Reconfigurable Embedded Systems (ADRES) [63] from IMEC, NXP's EVP16 processor [64], and the eXtreme Processing Platform (XPP) [65] from PACT Informationstechnologie. Figure 3 illustrates an ADRES instance, which contains a Coarse Grain Array (CGA) of FUs and three Very Long Instruction Word (VLIW) FUs. The VLIW FUs and a limited subset of the CGA FUs are connected to globally shared data Register Files (RFs), which are used to exchange data between the two sections. For this group of architectures, memory accesses may suffer from long-path data transfers, since data memories are accessible only from the border of the cluster. These long-path transfers may result in high data communication overhead especially for large-size clusters. Additionally, centralized memory organization may cause memory contention during concurrent data accesses, which may become a bottleneck for high dimensional computations (e.g., vector processing).

Figure 1(c) shows architectures consisting of heterogeneous units interconnected through an on-chip network. Examples are the Single Instruction stream Multiple Tasks (SIMT) DSP [62] from Coresonic, the FlexCore [66], the Transport Triggered Architecture (TTA) [67], the Dynamically Reconfigurable Resource Array (DRRA) [68], and Adaptive Computing Machine (ACM) [69] from

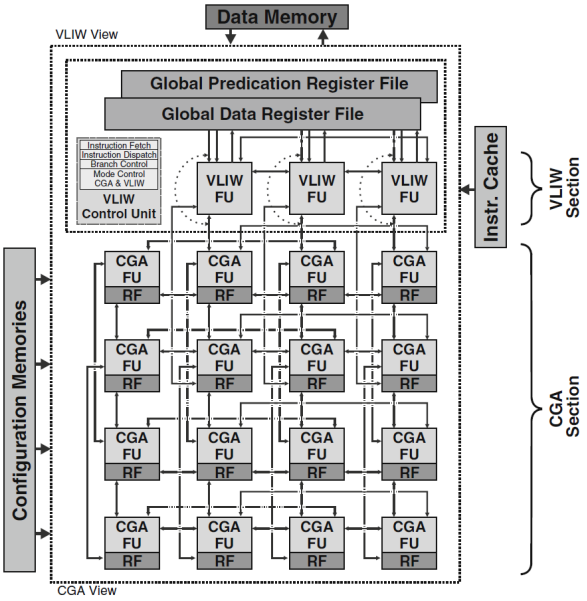


Figure 3: ADRES instance with 16 Coarse Grain Array (CGA) FUs and three Very Long Instruction Word (VLIW) FUs [61].

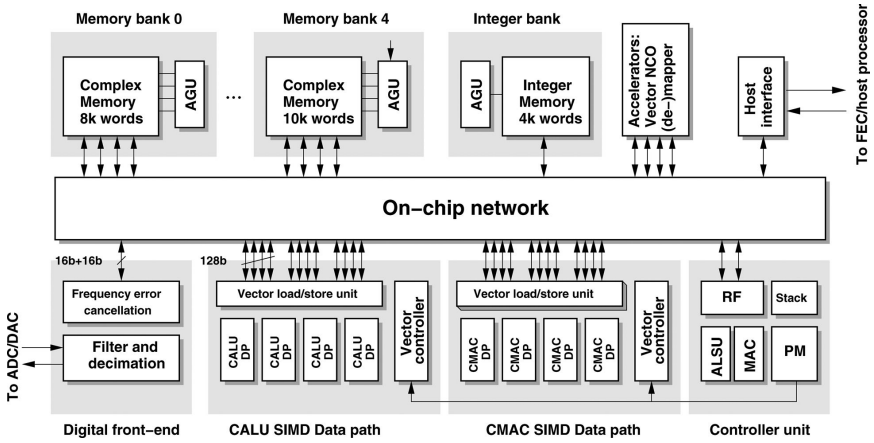


Figure 4: Overview of Coresonic SIMT DSP [62].

Quicksilver. The SIMT DSP from Coresonic is shown as an example in Figure 4, where multiple vector execution units, memory banks, and application specific accelerators are connected to a restricted cross-bar switch. Because of the heterogeneity, this type of architectures can be tailored to specific application domains to achieve efficient computations. However, one potential problem is the overhead of network interconnects, which increases linearly with the number of array nodes. This may restrict the usage of architectures in high dimensional data applications. For instance, the customized network presented in a 16-bit architecture [67] consumes almost the same area as all its arithmetic parts. Thereby, extensions to vector processing using many more array nodes may be unaffordable. Additionally, when considering hybrid computing, various-width data transfers via shared homogeneous network interconnects are not cost effective and may require frequent data alignment operations. Moreover, architectural scaling may require redesign of network interconnects, resulting in poor scalability.

In view of the high hardware efficiency owing to heterogeneity, the proposed cell array is built upon the third architectural category, the heterogeneous resource array. To tackle the NoC overhead and scalability issue, a hierarchical network topology is adopted, which contains high-bandwidth local interconnects and flexible global data routing. Additionally, to ease data sharing between surrounding array elements, processing and memory cells are separated as two distinct function units which are shared as global resources and distributed throughout the entire network. The following sections presents the cell array architecture in detail.

3 Architecture Overview

The reconfigurable cell array is constructed from heterogeneous tiles, containing any size, type, and combination of resource cells. As an example, a 4-tile cell array is shown in Figure 5. Besides the cell array, the entire system platform contains a master processor, a Multi-Port Memory Controller (MPMC), a Stream Data Controller (SDC), a Stream Configuration Controller (SCC), and a number of peripherals. The master processor schedules tasks to both the cell array and peripherals at run-time. The MPMC interfaces with external memories, while the SDC and SCC supply the cell array with data and configurations, respectively. Since the focus of this thesis is on the cell array, other system blocks will not be discussed.

Resource Cell (RC) is a common name for all types of functional units inside the cell array, including *processing*, *memory*, and *network routing cells*. Within the cell array, processing and memory cells are separated as two distinct functional units. This arrangement has following advantages.

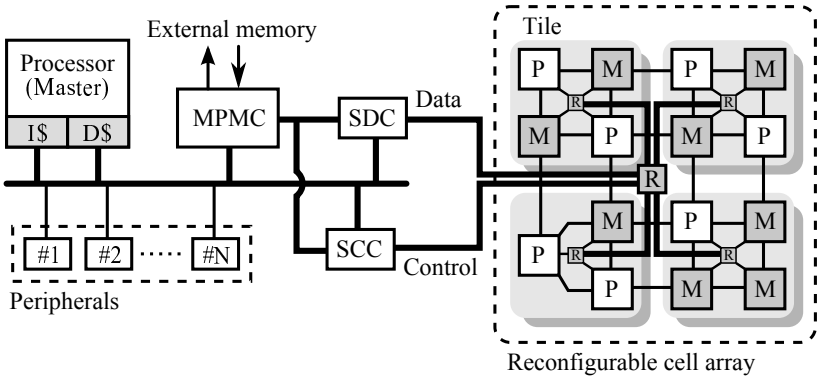


Figure 5: Overview of system platform containing a coarse-grained dynamically reconfigurable cell array. The separated processing (P) and memory cells (M) communicate over a hierarchical network using both local interconnects and global routing with network routing cells (R).

- **Easy data sharing:** the separation of memory from processing cells significantly simplifies data sharing, as memory cells can be shared by multiple processors without physically transferring data. Memory coherence is preserved by allowing direct data transfers between memory cells without involving processors.
- **Flexible memory usage:** memory cells can be individually configured to provide different access patterns, such as First In First Out (FIFO), stack and random access.
- **Advanced data access control:** processing cells can be used as Direct Memory Access (DMA) controller to accomplish irregular or advanced memory access, e.g., bit-reversal in FFT/IFFT.
- **Dynamic memory allocation:** when larger memory capacity is required than a single cell can provide, multiple memory cells can be concatenated at system run-time to provide larger data storage.
- **Simplified programming model:** the separated data processing and memory access management naturally support dataflow programming languages like CAL [70] and computation models like Kahn process networks [71]. Additionally, changes of execution clock cycles in either processing or memory cells have no influence on the control flow of the entire cell array architecture, since inter-cell communication is self-synchronized through NoC data transfers.

- **Natural support for future technology:** the advantage of processing and memory separation may be more pronounced when using 3D stacking technology [72]. With this technology support, processing and memory cells can be placed at different chip layers and interconnected by using Through-Silicon Vias (TSVs). This approach may further increase memory access bandwidth, reduce delays of network interconnects, and ease chip layout and routing process.

To meet the computational and flexibility requirements while keeping a low control overhead, a variety of functional units are integrated. Communication between RCs is managed by combing local interconnects for high data rate and a global routing network for flexibility. Compared to other interconnect topologies, the hierarchical network provides tighter coupling to heterogeneous RCs. For instance, connections within each tile can be localized to suffice both bandwidth and efficiency requirements, while hierarchical links provide flexible routing paths for inter-tile communication. All RCs in the array are configured dynamically on a per-clock-cycle basis, in order to efficiently support run-time application mapping.

3.1 Processing Cell

Processing cells contain computational units to implement algorithms mapped on the cell array. Additionally, they can be used to control operations and manage configurations of other RCs. The heterogeneity of the architecture allows integration of any type of processors in the array to suffice various computational demands. For example, processing cells may be built as general purpose processors or specialized functional units. Each processing cell is composed of two parts, processing *core* and *shell*, illustrated in Figure 6(a). Encapsulated by the processing shell, the core interfaces with other RCs via network adapters in the shell. Thanks to this modular structure, processing cell customization is simplified since only the core needs to be replaced to implement different computational operations. In addition, integration of customized functional units, either user-defined Register Transfer Level (RTL) or licensed Intellectual Property (IP) cores, are supported in the cell array without changing the network interface. The network adapters in the processing shell are mapped as registers that are directly addressable by the core. The number of adapters in a processing cell is parametrizable at system design-time. For illustration, the following presents a generic signal processing cell, named as GPC for short, which has been used in a flexible FFT core presented in [73].

A GPC is a customized Reduced Instruction Set Computing (RISC) processor with enhanced functionality for digital signal processing and support for fast network port access. An overview of the processor pipeline stages

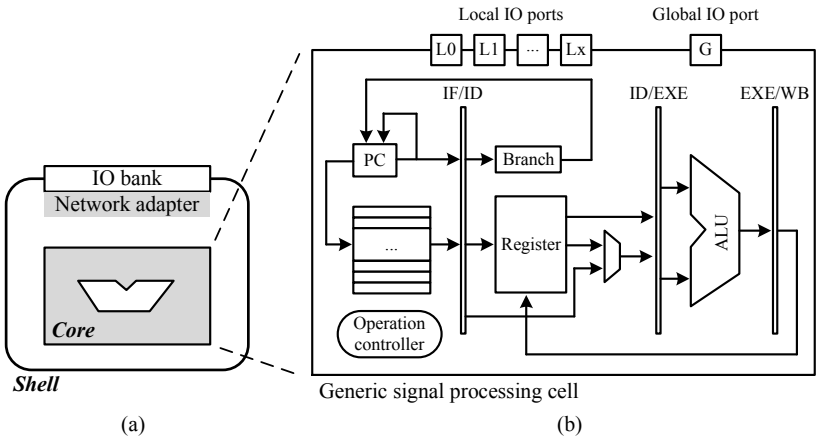


Figure 6: (a) Block diagram of a processing cell, consisting of a *core* and a *shell*. (b) Architecture of a generic signal processing cell (GPC).

and internal building blocks is shown in Figure 6(b). Communication I/O ports are mapped as registers, directly accessible in the same way as General Purpose Registers (GPRs). An instruction that access I/O port registers is automatically stalled until data become available. Hence, additional operations to move data between GPRs and I/O ports are avoided. The GPC performs data memory operations by connecting with one or more memory cells via communication I/O ports. Using direct I/O addressing, load-store and computational operations may be combined into one instruction. Consequently, clock cycles associated with memory operations are eliminated in contrast to conventional load-store architectures. Moreover, the implicit load-store operations lead to a compact code size, and make the memory operations possible in all instructions. Enhanced functionalities for digital signal processing include multiply-accumulate, radix-2 butterfly, and data swap. To reduce control overhead in computationally intensive inner loops, the GPC includes a zero-delay Inner Loop Controller (ILC). The ILC comprises a special set of registers that are used to store program loop count and return address. During program execution, the loop operation is indicated by an end-of-loop flag annotated in the last loop instruction. The operation mode and status of each processing cell can be controlled and traced conditionally during run-time. For example, it is possible to halt instruction execution, step through a program segment, and load a program partially. Due to the simple pipeline structure and enhanced data processing operations, the GPC may be used for regular data processing and control-flow handling, such as linear filter, FFT/IFFT, and DMA control.

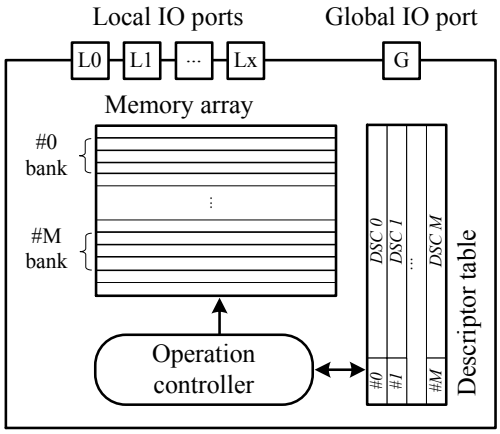


Figure 7: Architectural block diagram of a memory cell.

3.2 **Memory Cell**

The distributed memory cells provide both processing cells and data communication with shared storage to allow buffering, reordering, and synchronization of data. Each memory cell contains a memory array, a DeSCriptor (DSC) table, and an operation controller, as illustrated in Figure 7. The memory array can be dynamically configured to emulate one or more memory banks, while the DSC table is an array of configuration registers containing user-defined memory operations. All stored DSCs are dynamically configurable, and may be traced back for debugging. Each DSC is 64-bit long, which defines the size and operation mode of a memory bank, records memory operation status, and specifies I/O ports for stream transfers. The configuration options of a DSC are listed in Table 1. The 64-bit DSC is composed of two 32-bit parts that are individually configurable. Specifying memory operations using DSCs relieves processing cells from memory access managements, resulting in reduced control overhead and improved processing efficiency. The operation controller manages and schedules DSC execution, monitors data transactions, and controls the corresponding memory operations.

Using memory DSCs, each memory bank can be configured to emulate either FIFO or Random Access Memory (RAM) behaviour. In the FIFO mode, the allocated memory bank operates as a circular buffer. Address pointers are managed by the operation controller and are automatically increased each time the DSC is executed. Multiple memory cells operating in FIFO mode may be cascaded to form one large data array. This feature provides flexible memory usage, and reduces unit capacity requirement as well as hardware footprint in

Table 1: Example of a memory DeSCriptor (DSC).

	Field	Bits	Length	Description
Part I	dtype	31-30	2	Operation mode select
	rd_ok/active	29	1	FIFO reading status/ RAM active transfer flag
	wr_ok/rnw	28	1	FIFO writing status/ RAM read-write select
	src/paddr	27-24	4	FIFO data source port/ RAM address port
	dst/pdata	23-20	4	FIFO data destination port/ RAM data port
	id	19-10	10	Global packet destination ID
	base	9-0	10	Start address
Part II	high	31-22	10	End address
	rptr/ptr	21-12	10	Current FIFO reading pointer/ Current RAM data pointer
	wptr/tsize	11-2	10	Current FIFO writing pointer/ Current RAM data transfer size
	io_bank_rst	1	1	I/O port register reset
	Reserved	0	1	Reserved

a single memory cell. Additionally, since large storage may lead to an irregular physical memory layout, slicing it into smaller modules eases hardware placement and routing. When operating memory cell in the RAM mode, a data service request (read or write) is required to specify the start address and data transfer size. The operation controller is responsible for keeping track of data transfers, managing memory address pointers, and updating DSCs. Conditions to execute a memory DSC are resolved by inspecting both incoming and outgoing packet transfers and current memory status. For example, writing data to a full FIFO will not be executed until at least one data is read.

The length of the DSC table, the size of the memory array, and the number of local I/O ports are configurable at system design-time, while memory descriptors are dynamically reconfigurable.

3.3 Network-on-Chip

To enable communication between any pair of resource cells, most existing NoCs are based on flexible interconnect topologies, such as 2D-mesh, spidergon, and their derivatives [74–77]. Additionally, various routing algorithms (e.g., static and dynamic) and switching techniques (e.g., wormhole and Time Division Multiplexing (TDM)) are employed to reduce traffic congestions, provide service guarantees, and shorten communication latency [78]. Although most NoC implementations can suffice performance requirements with respect to latency and bandwidth, they often appear to be area and power consuming. For instance, NoCs used in [79] and [67] take almost the same area as all their logic parts and the one in [79] consumes about 25% of total power.

This work aims at developing an area and power efficient NoC by fully exploiting the property of *communication locality* in reconfigurable architectures [75]: data traffic is mostly among nearest neighbors (referred to as local communication), while long distance (global) transfers that require routing supervisions are of a small portion. Therefore, the primary concern of local network design should be on high bandwidth and low cost, while simple routing and switching techniques are sufficient for global transfers to provide adequate flexibility support.

In light of the aforementioned property, a hierarchical network architecture is proposed that splits local and global communication into two separate networks, which are handled independently using different network topology and switching techniques. Figure 8 illustrates an overview of the proposed hierarchical NoC deployed in a 4×4 array. Communication between neighboring RCs (local) within each tile is performed using bi-directional dedicated links, whereas inter-tile global transfers are realized through a hierarchy of network routers structured in a tree topology using a static routing strategy, see shaded part in Figure 8. Thanks to the network separation and hierarchical arrangement, the proposed NoC can be easily scaled by extending tree hierarchies of the global network and neighboring local interconnects without affecting others. Additionally, in conjunction with the tile-based architecture, the proposed NoC intrinsically supports Globally Asynchronous Locally Synchronous (GALS) network construction. For example, synchronous transfers are performed within each tile and the global network (together with additional asynchronous FIFOs) is used to bridge between different clock domains.

To connect RCs to the local and global network, adapters are used as a bridge between high level communication interfaces employed by RCs and network specific interfaces implemented in the NoC. In this work, AMBA 4 AXI4-stream protocol [80] is adopted for implementing NoC adapters.

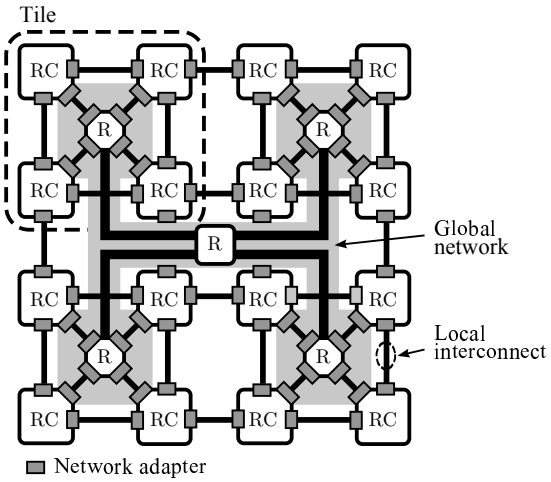


Figure 8: An overview of the proposed hierarchical NoC arranged in a 4×4 array. Neighboring RCs are directly communicated via local interconnects, while global transfers, shaded in grey, are achieved through hierarchical routing with tree-structured network router (R).

Global Network

Global network enables non-neighboring nodes to communicate within the array and provides an interface to external blocks, e.g., memory and master processor. Figure 9(a) depicts a simplified view of the global network in a 4×4 cell array, wherein each RC is labelled with a unique network IDentifier (ID) used as a routing address for global data transfers. Global communication is based on packet switching and carried out using a hierarchy of tree-structured network routers. Routers forward data packets based on a static routing lookup table that is generated at design-time in accordance to physical network connections. For example, the router in the upper-left tile shown in Figure 9(b) forwards packets to RCs with IDs ranging from 0 to 3. In the tree-structured global network, each router is denoted as $R_{i,l}$, where i is the router index number and l is the router hierarchical level, illustrated in Figure 9(c). A link from a router $R_{i,l}$ to $R_{i,l+1}$ is referred to as an uplink. Any packet received by a router is forwarded to the uplink router if the packet destination ID falls outside the range of the routing table. Since routing network is static, there is only one valid path from each source to each destination. This simplifies network traffic scheduling, reduces hardware complexity, and enables each router instance to be optimized individually during hardware synthesis. However, a

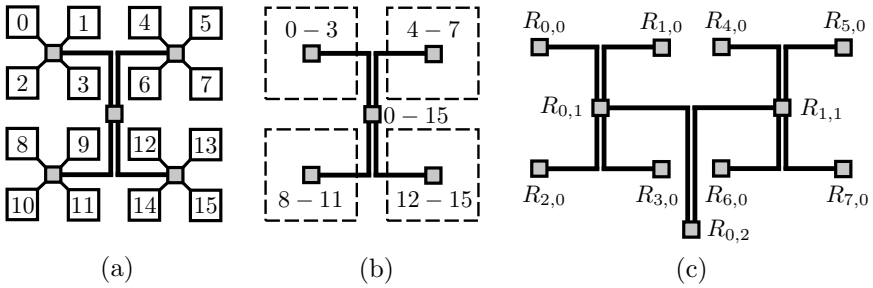


Figure 9: A simplified view of the global network in a 4×4 array. (a) Each RC is labelled with a unique network ID. (b) A range of consecutive IDs (base-high) are assigned to each static routing table. (c) Hierarchical router naming as $R_{index,level}$.

drawback is network congestion compared to adaptive routing algorithms. Systematic analysis on network performance at design-time is therefore crucial to avoid traffic overload. However, considering the high communication locality, this congestion issue is of less concern and does not hinder the performance of the proposed NoC. For network traffic modelling and performance evaluation, a SystemC-based exploration environment SCENIC can be used. Details of SCENIC can be found in [7, 8, 81].

Network Routing Cell

Network router forwards data packets over the global routing network. Each router consists of three main building blocks: a decision unit, a routing structure, and an output packet queue, implemented as pipelined architecture, see Figure 10(a). In each clock cycle, the decision unit monitors incoming and outgoing packets, looks up the routing path, handles data transfers, and configures the routing structure to forward data packets accordingly. The routing structure is made up of a full-connection switch, capable of handling multiple data requests in each clock cycle. The output packet queue, operating in a FIFO basis, buffers data packets travelling through the global network. The depth of the output queue and FIFO type (either synchronous or asynchronous) are design-time configurable, used to suffice different NoC requirements.

Figure 10(b) shows an overview of the decision unit inside each network router. It contains a static routing lookup table, a transaction log table, and a packet arbiter. Every arriving data packet is checked and recorded in a transaction log table, marked with 'X's in Figure 10(b). The logged transactions are prioritized and handled based on different arbitration policies and conditions of output queues. Two simple arbitration policies are currently supported and

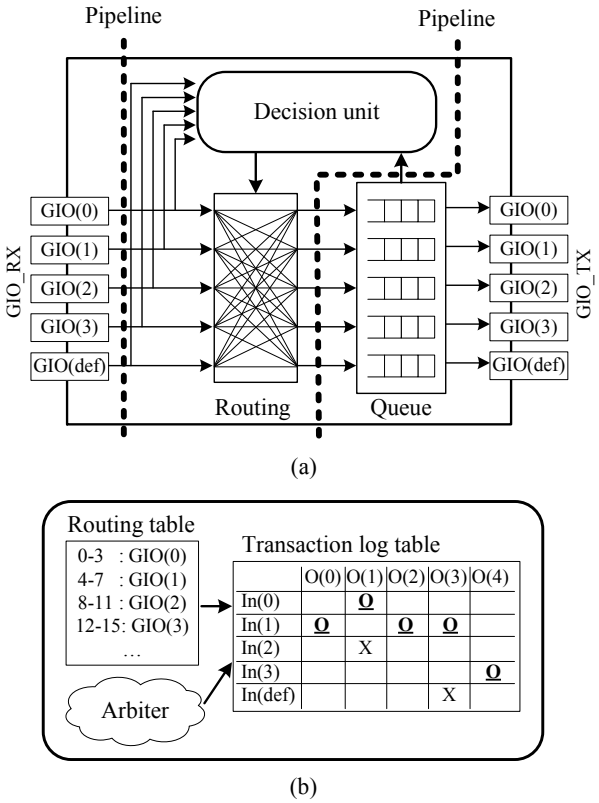


Figure 10: (a) Block diagram of the network router. (b) Internal building blocks of a decision unit.

are design-time configurable: the fixed and round-robin scheme. With fixed arbitration, the arbiter always starts from the first log entry, and traverses column-wise through the entire table until a candidate transaction is found, marked with 'O's in Figure 10(b). A transaction is considered to be a candidate when it is recorded in the log table and the corresponding output packet queue is not full. With this approach, all transactions are assigned with priorities according to their position in the log table. In contrast, the round-robin algorithm provides a starvation-free arbitration, which assigns time slices to each entry in equal portions and handles all transactions in order without priority. After routing arbitration, selected candidate transactions (marked with 'O's in Figure 10(b)) are forwarded to the corresponding output queue in the following clock cycle. Considering delays caused by input I/O register, pipelined routing

operations, and output FIFO, packet forwarding through each network router induces 3 Clock Cycles (CCs) transport latency (without Tx I/O register).

Local Network

Local network consists of dedicated interconnects between neighbouring RCs (Figure 8). Thus, local transfers require no routing supervision and provide guaranteed throughput and transport latency. Compared to nearest-neighbour transfers in conventional mesh-based networks, bandwidth overhead due to redundant traffic headers is completely avoided in the adopted local communication. For example, considering the illustrated 4×4 array and a simple XY routing in mesh networks, it is required to have at least 4 bits in the traffic header to indicate destination coordinates of both X and Y directions. This overhead is more pronounced when sophisticated routing algorithms are used, such as source and adaptive routing. Therefore, avoiding such bandwidth overhead in every neighbouring data transfer contributes to total NoC efficiency.

Communication Flow Control

To assure safe delivery and self-synchronization of each data transfer, flow control is used in both local and global networks, implemented using a FIFO-like handshake protocol, illustrated in Figure 11. In addition to conventional valid-ACKnowledgement (ACK) handshaking, FIFO-like operations are adopted to reduce communication latency. The basic idea is to use I/O registers as eager transport buffers, which are writable as long as the buffers are not full. A communication link is suspended only if all buffers are fully used, transmitter has more data to send, and receiver has not yet responded previous transfers. As indicated in Figure 11, the ACK signal in each I/O register has two acknowledgement mechanisms. In the case of empty buffers, data transfers are automatically acknowledged by the I/O registers. Otherwise, the ACK signal is driven by the succeeding data receiver. This way, the ACK signal acts as an empty flag of the transport buffers and reflects the status of the communication link. Data transmitter can proceed with other operations immediately the ACK from the succeeding stage is received, without waiting for the final destination to respond. Compared to conventional end-to-end handshake protocols, in which the ACK signal is sent all the way from the final destination and requires multiple clock cycles to propagate through all I/O registers, the adopted scheme divides long communication path into smaller segments (hops), each having 1 CC transport latency under the case of eager receiver. As for communication between neighbouring RCs, the FIFO-like handshaking results in at least 2 times latency reduction in comparison to the end-to-end handshake scheme.

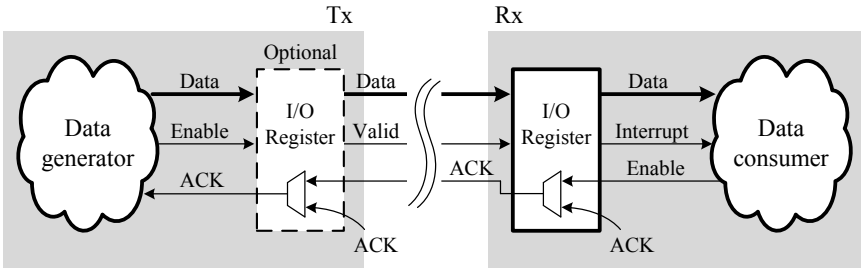


Figure 11: Data communication flow control with FIFO-like handshake protocol. I/O registers at the transmitters are design-time configurable.

Table 2: Summary of the Hierarchical NoC.

	Topology	Switching	Latency	Throughput
Local	Direct link	Circuit (GS)	1 CC	1 CC
Global	Tree	Packet (BE)	4 CCs/hierarchy	1 CC/hop

To sum up, Table 2 lists the characteristics of the proposed NoC. Benefiting from the dedicated interconnects, local network provides Guaranteed Services (GS) and has a transport latency and throughput of 1 CC. The global network offers Best-Effort (BE) packet switching and induces additional 3 CCs transport latency (without Tx I/O register) every time a network router is used. However, the throughput of global transfers via routers is still 1 CC thanks to the pipelined architecture.

3.4 Resource Configuration

Dynamic reconfigurations for all RCs are managed in two ways, either by a master processor via hierarchical network or by any of the processing cells distributed in the cell array. When configuring RCs through the master processor, see Figure 8, a Stream Configuration Controller (SCC) is used to assist network packet transfers. The SCC contains a stream table programmed by the master processor, and provides information about where and how network packets should be transmitted. For each configuration, the SCC loads data from external memory via the MPMC, packs data as network packets, and transfers the packets to target RC via the hierarchical routing network. The advantages of configuring RCs from a centralized master processor are twofold. First, as configurations are loaded from external memories, requirement for the size of configuration files is reduced. Second, the master processor may utilize hard-

ware resources efficiently on a system level, based on the needs of application mapping. After receiving a task, the master processor assesses the computational workload, checks the status of RCs, and assigns the task to achieve maximum efficiency. For instance, the master may partition and assign the task to different RCs or time-multiplex a single RC. A drawback of the centralized configuration is the communication latency through the global network, as mentioned in Section 3.3. As a result, the centralized RC configuration scheme is mainly used for transferring large configuration files to the cell array during, for example, context switching between different application mappings.

RC configurations and supervisions can also be conducted inside the cell array by using distributed processing cells. This is achieved by storing RC configurations as special instructions locally inside processing cells, which transmit configuration packets to the corresponding RCs during program execution. With this approach, run-time configurations are smoothly integrated into the normal processing flow. For example, configurations are issued immediately the current task is completed without interrupting and waiting for responses from an external host. Additionally, configuration packets are transmitted mainly using local interconnects, avoiding long configuration latency due to global communication. Because RC configurations are stored as part of the local programs of processing cells, configuration file size needs to be kept down when using this approach. Therefore, this in-cell configuration scheme is suitable for small function configurations in the cell array, such as adapting algorithms for different standards or operating scenarios.

4 Design Flow

Constructing a reconfigurable cell array generally involves three design phases (Figure 12), *specification*, *design*, and *implementation*, and three design methodologies, *algorithm-architecture*, *hardware-software*, and *processing-memory co-design*. In the specification phase, target wireless communication standards and baseband processing tasks need to be defined first. This is to limit the scope of the development, in order to enable design-space exploration for creating an efficient architecture. To this end, the target standards and tasks should have some common computational characteristics to enable hardware sharing and acceleration, otherwise diverse operation requirements would lead to a generic architecture with low hardware efficiency. After defining the standards and tasks to cover as well as performance specifications, algorithm selection and operation analysis are carried out to provide a good foundation for the following hardware development. Examples of selection criteria are computational parallelism, precision requirement, and regularity of operations. To make use of essential architectural characteristics, algorithms often need to be tailored for

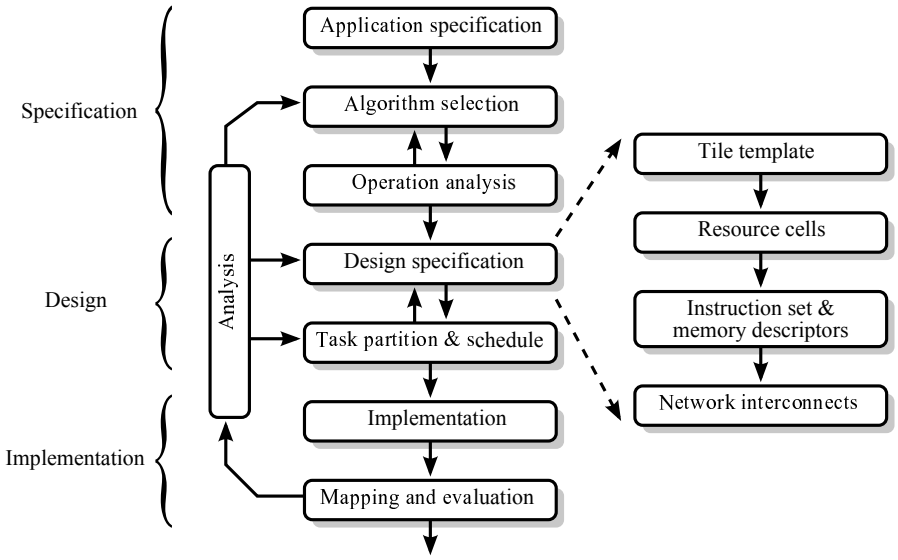


Figure 12: Data flow to construct a reconfigurable cell array.

the target hardware architecture. This is referred to as *algorithm-architecture co-design*, which is usually an iterative process for obtaining an optimum result. It should be pointed out that the algorithm selection is especially important when handling multiple standards and tasks, since it has a great influence on efficient usage of underlying hardware, such as resource sharing over time.

The design phase involves two steps, *design specification* and *task scheduling*. After obtaining a rough estimation on computational requirements and memory usage from the operation analysis, design specifications for constructing the cell array are elaborated. These include design of resource tiles, RC selections, instruction set and memory descriptors, and NoC interconnects (see the sub-figure shown on the right side of Figure 12). First, the number of resource tiles is determined based on the computational requirements and design constraints such as area and timing budget. Besides, a rough task scheduling is performed on the tile-level to partition tasks to resource tiles. Second, RC selection is carried out for each tile based on the rough task partition. The number of RCs assigned to each tile is determined again by the computational requirements and design constraints. Third, instruction set and memory descriptors are specified. The number of instructions and memory descriptors is a trade-off between the ease of software implementation and hardware complexity [1]. This is referred to as *hardware-software co-design*. Last, network

interconnects may be refined to further improve hardware efficiency. Examples include bandwidth enhancements for heavy-traffic links and pruning of unused network connections. With the elaborated design specifications, detailed task partition and scheduling can be carried out, requiring detailed operation analysis to better assign tasks to RCs, namely processing and memory cells. This is referred to as *processing-memory co-design*. For example, memory address manipulations, such as stride access and matrix transpose, are better performed in memory cells by using descriptor specifications, since no physical memory access (read & write) is needed. Moreover, analysis of network traffic is required to avoid problems like network congestion.

Once the design specifications and the task scheduling are completed, the cell array is implemented and the target tasks are mapped. Mapping results are evaluated and fed back to the corresponding design phase for further improvements in case the design requirements are not met.

5 Summary

This part introduces the coarse-grained dynamically reconfigurable cell array architecture, aiming to provide a balance among performance, hardware efficiency, and flexibility. The proposed architecture has three key features. First, processing and memory cells are separated as two distinct function units for achieving easy data sharing and flexible memory usage. Second, a hierarchical NoC structure is adopted for providing high-bandwidth low-latency local communication and flexible global data routing. Third, in-cell configuration scheme employed in the cell array enables fast run-time context switching. To achieve a balanced design, three design methdologies, algorithm-architecture, hardware-software, and processing-memory co-design, are adoped during the construction of a reconfigurable cell array. Using the presented cell array as a baseline architecture, the following parts present further developments and architectural improvements of the cell array, especially the processing and memory cells, through two case studies. Architectural developments are carried out by exploiting computational properties of the target application domain, namely digital baseband processing in wireless communication.

Part II

Multi-standard Digital Front-End Processing

Abstract

To demonstrate flexibility and performance of the reconfigurable cell array architecture introduced in Part I, this part presents a case study of the platform configured for concurrent processing of multiple radio standards. Flexibility of the architecture is demonstrated by performing time synchronization and Carrier Frequency Offset (CFO) estimation for multiple OFDM-based standards. As a proof-of-concept, this work focuses on three contemporarily widely used radio standards, 3GPP Long Term Evolution (LTE), IEEE 802.11n, and Digital Video Broadcasting for Handheld (DVB-H). The employed reconfigurable cell array, containing 2×2 resource cells, supports all three standards and is capable of processing two concurrent data streams. The cell array is implemented in a 65 nm CMOS technology, resulting in an area of 0.48 mm^2 and a maximum clock frequency of 534 MHz. Dynamic configuration of the cell array enables run-time switching between different standards and allows adoption of different algorithms on the same platform. Taking advantage of the in-cell configuration scheme (presented in Part I), context switching between different operation scenarios requires at most 11 clock cycles. The implemented 2×2 cell array is fabricated as a part of a Digital Front-End Receiver (DFE-Rx) and is measured as a standalone module via an on-chip serial debugging interface. Running at 10 MHz clock frequency and at 1.2 V supply voltage, the array reports a maximum power consumption of 2.19 mW during the processing of an IEEE 802.11n data reception and 2 mW during hardware configurations.

1 Introduction

Today, there is an increasing number of radio standards, each having different focuses on mobility and data-rate transmission. For example, 3GPP Long Term Evolution (LTE) [5] aims to offer high mobility with moderate data-rate; IEEE 802.11n [82] provides a high data-rate alternative for network services under stationary conditions; and Digital Video Broadcasting for Handheld (DVB-H) [83] specifically addresses multimedia broadcast services for portable devices. The evolution of radio standards continuously drives the development of underlying computational platforms with increased complexity demands. Meanwhile, requirements on time-to-market and Non-Recurring Engineering (NRE) cost force today's hardware platforms to be able to adopt succeeding amendments of standards. Doing modifications on dedicated hardware accelerators for each standard update is not affordable regarding both time and implementation cost. Furthermore, to obtain a continuous connection or constant data-rate transmission, contemporary user terminals are expected to support more than one standard and to be able to switch between different networks at any time. Therefore, flexibility has become an essential design parameter to help computational platforms cope with various standards and support multiple tasks concurrently.

It is well identified that simultaneous support of multi-standard data receptions using flexible hardware platforms is a great challenge. Although some early attempts have been presented in both academia and industry [84, 85], experiments so far have been limited to the support of a single data stream. Switching between different standards is only possible through off-line configurations and is conducted by an external host controller. Despite not being reported, configuration time during context switching is envisioned to be on a scale of hundreds of clock cycles, since the host controller has to be interrupted to conduct the loading of appropriate programs/configurations before getting ready for new data receptions. Evidently, this off-line switching approach is undesired from users' experience point of view, as terminals are temporarily "disconnected" ever time when they enter a new radio environment. To address this issue, the European Union (EU) project "Scalable Multi-tasking Baseband for Mobile Communications" [86], or Multibase for short, is initiated. It aims to support concurrent processing of multiple data streams in a multi-standard environment and to provide seamless handover between different radio networks. The support of concurrent data streams improves user experiences, e.g., having simultaneous voice communication and video streaming. It also ensures continuous connectivity of user terminals, since an existing network connection can be maintained while a new network service is being established.

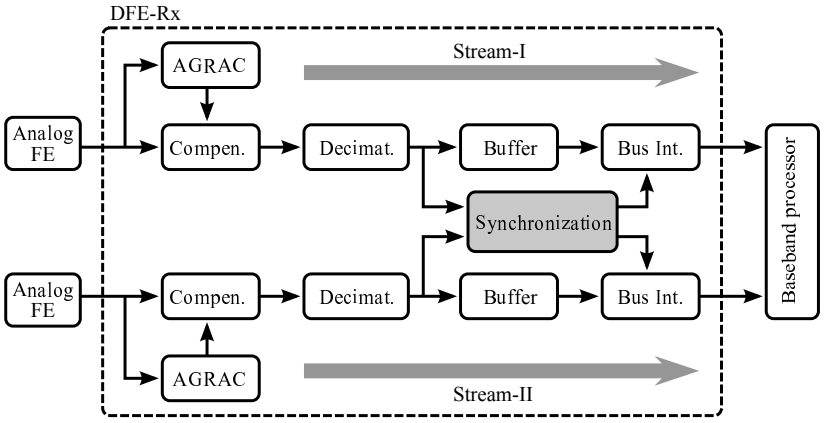


Figure 1: Block diagram of the DFE-Rx constructed in the EU Multibase project. This work focuses on the implementation of the synchronization block, shaded in the figure, by using the reconfigurable cell array.

This work is carried out as a part of the EU Multibase project. The primary focus is on data computations in a Digital Front-End Receiver (DFE-Rx). As a proof-of-concept, three OFDM-based radio standards are selected: 3GPP LTE, IEEE 802.11n, and DVB-H. Besides, it is required to process two concurrent data streams from any of the three standards. Among processing tasks in a DFE-Rx, this work maps time synchronization and Carrier Frequency Offset (CFO) estimation onto the reconfigurable cell array. Given that these tasks are performed during the (re)establishment of a data link between transmitter (Tx) and receiver (Rx), the required computational units are active only for a small fraction of the time when the receiver is on. This motivates the adoption of reconfigurable architectures in order to reuse hardware resources for other processing tasks. Hardware reusing can be exploited in two aspects. Firstly, in a *multi-standard single-stream* scenario, the same hardware can be reconfigured after OFDM synchronization to perform other baseband processing in succeeding stages, such as refined frequency offset estimation and tracking. Secondly, in a *multi-standard multi-stream* scenario, underlying hardware resources can be shared for concurrent processing of multiple data streams. Figure 1 depicts the block diagram of the complete DFE-Rx and highlights the target processing block "synchronization". In the following, each function block of the DFE-Rx is briefly described. Implementation details can be found in [87] and [49].

Taking digitized signals from the analog front-end, the Automatic Gain and Resource Activity Controller (AGRAC) and the compensation block adjust the

Table 1: Sampling frequency of the three target radio standards.

	Bandwidth [MHz]	# Subcarriers	Sampling frequency [MHz]
IEEE 802.11n [82]	40	128	40
3GPP LTE [5]	20	2048	30.72
DVB-H [83]	8	8192, 4096, 2048	9.14

gain of incoming signals and perform DC-offset and IQ imbalance compensations. Besides, the AGRAC is used as a master core in the DFE-Rx to control the operation of other function blocks. The decimation filter chain contains a farrow resampler [88] used to adjust the sample-rate of input data to that of the corresponding radio standards. Data resampling is one of the fundamental tasks when dealing with multiple radio standards, since the elementary sampling frequency varies between standards and is often not an integer multiple of one another. Since no pre-knowledge is given on which standard is going to be processed, the DFE-Rx has to operate at a frequency that is sufficiently high to capture signals of all standards without aliasing. Among the three standards under analysis, IEEE 802.11n has the highest sample-rate, see Table 1, and is thus set as the master sampling frequency of the DFE-Rx. As a result, data streams of LTE and DVB-H after the decimation filter chain have an oversampling factor of $\lfloor 40/30.72 \rfloor = 1$ and $\lfloor 40/9.14 \rfloor = 4$, respectively. The data reception buffer is used to store data inputs temporarily during front-end processing, and the bus interface block adapts the DFE-Rx to the following baseband processor.

The remainder of this part is organized as follows. Section 2 formulates the problem in more detail. Similarities and differences of the OFDM time synchronization in the target wireless radio standards are analyzed. Computational operations required by the synchronization process are elaborated. Section 3 presents hardware developments of the reconfigurable cell array with focuses on processing and memory cells. Section 4 starts by describing the computational and memory resource allocations during concurrent multi-standard processing. A software tool developed for generating hardware configurations of the cell array is presented. Implementation and silicon measurement results are summarized. The flexibility of the proposed solution is demonstrated by mapping a different algorithm onto the cell array after the chip is fabricated. Benefiting from the new algorithm mapping, the number of standards supported by the same cell array is further extended. Finally, Section 5 concludes this part.

2 Algorithm and Implementation Aspects

In OFDM, synchronization is needed due to the lack of common time and frequency references between Tx and Rx. An incorrect symbol timing may result in loss of orthogonality in the narrow-band subcarriers. Moreover, orthogonality may also be destroyed in the presence of a frequency mismatch between oscillators in Tx and Rx. OFDM synchronization makes sure of preserving orthogonality by providing a reliable start[1] of the OFDM symbol and Carrier Frequency Offset (CFO) estimation.

The synchronization process is usually performed in time and frequency domain, commonly referred to as *acquisition* and *tracking* stage, respectively [89]. The acquisition stage aims to find the start of each OFDM symbol and to perform a rough estimation of CFO. The tracking stage aims to refine the parameters obtained from the acquisition stage. This study focuses on the acquisition stage and assumes that the channel impulse response is shorter than the length of CP.

2.1 Time Synchronization and CFO Estimation

Maximum Likelihood (ML) estimation [90] is commonly used to perform time synchronization in OFDM systems. The algorithm can be used on either pilots/preamble or Cyclic Prefix (CP). In the three standards under analysis, CP is present. Besides, IEEE 802.11n contains a preamble, which has specific Short Training Symbols (STSs) designed for data detection and time synchronization [82], see Figure 2. Given that all STSs are identical, the first STS can be considered as the CP of the remaining part in the short training field (t_2–t_{10} in Figure 2). Based on either CP or preamble, the ML estimation algorithm can be expressed as

$$\widehat{\theta} = \begin{cases} \arg\max\limits_{n}\{|\gamma[n]|\} & \text{if } |\gamma[n]| \geq T \\ \text{No symbol start found} & \text{otherwise} \end{cases}, \tag{1}$$

where

$$\gamma[n] = \sum_{k=n-L+1}^{n} r[k]r^*[k-M]. \tag{2}$$

In (1) and (2), $r[n]$ is the received data vector at sample index n, $\gamma[n]$ is the output of moving-sum, $\widehat{\theta}$ indicates the estimated symbol start, and $(\cdot)^*$

[1] Orthogonality of narrow-band subcarriers is preserved as long as the estimated start lies within the Cyclic Prefix (CP) of an OFDM symbol.

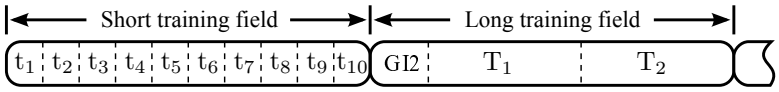


Figure 2: IEEE 802.11n short and long training field [91].

Table 2: Comparison for the length of moving-sum L, autocorrelation distance M, and the number of subcarriers N_c in the ML-based time synchronization.

	L	M	N_c
IEEE 802.11n	16×9	16	64
3GPP LTE	144	2048	2048
DVB-H	64	8192, 4096, 2048	8192, 4096, 2048

denotes the complex conjugate operator. T represents the threshold value, which is used to find the symbol start by detecting the position of the maximum correlation value. T is a function of Signal-to-Noise Ratio (SNR) and is computed off-line and adjusted in accordance to different standards. L is the length of the moving-sum operation, and M is the autocorrelation distance, i.e., the number of samples from the start of CP to its corresponding copy within the OFDM symbol. The values of L and M vary among standards and also between different synchronization methods, CP-based for LTE and DVB-H and preamble-based for IEEE 802.11n. Table 2 summarizes the values of L, M and the number of subcarriers N_c for the three standards. In CP-based synchronization, the autocorrelation distance M equals to N_c, and the size of the moving-sum L equals to the length of the CP. LTE and DVB-H fall into this category. Since better synchronization accuracy is expected when using preambles, preamble-based approach is used for IEEE 802.11n. In this case, M corresponds to the size of a STS and L equals to the size of remaining 9 STSs ($L = 16 \times 9$ in Table 2). This is equivalent to computing correlation between neighboring STSs and accumulating results over the entire short training field.

A common method to estimate CFO is to divide the offset value into two components, expressed as

$$\Delta f_c = \alpha + \varepsilon, \tag{3}$$

where α and ε represent the integer and fractional part of CFO, respectively. Both α and ε are normalized with respect to the sub-carrier spacing. ε is delimited by $|\varepsilon| \leq 0.5$. This study focuses on the computation of the fractional CFO. An approach to estimate ε is based on a phase computation of the

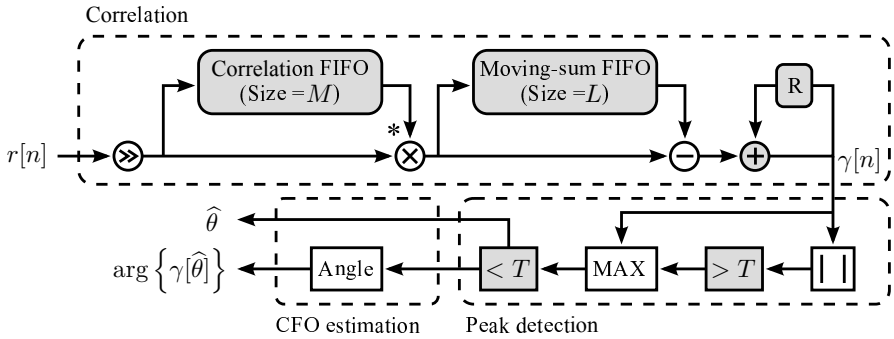


Figure 3: Conceptual implementation diagram of the ML-based time synchronization and CFO estimation. Shaded blocks have various design parameters required by different radio standards.

autocorrelation result at the estimated symbol start $\gamma[\widehat{\theta}]$ [90], i.e.,

$$\varepsilon = \frac{1}{2\pi} \arg\left\{\gamma[\widehat{\theta}]\right\}. \tag{4}$$

This is usually performed by using a COordinate Rotation DIgital Computer (CORDIC) algorithm operating in circular vectoring mode [92].

2.2 Operation Analysis

Based on the aforementioned algorithms, Figure 3 shows a conceptual implementation diagram for performing OFDM acquisition. Operations are partitioned into three main processing blocks: data correlation, peak detection, and CFO estimation. Different design parameters in the three radio standards (Table 2) set different hardware requirements for the shaded blocks in Figure 3. For example, the size of the correlation FIFO (M) changes from 16 to 2048 samples when switching from IEEE 802.11n to DVB-H 2K.

Input samples in this study are 12-bit complex numbers. To reduce memory requirements during the correlation computation, data samples in single-stream mode are truncated down to 4 bits. This relies on an assumption that performance of the synchronization in the acquisition stage only needs to be sufficiently accurate such that refined estimation algorithms in the tracking stage will work properly [93]. During concurrent multi-stream processing, memories are shared between two data streams and the wordlength of data samples is further reduced by half. As an example, performance analysis of CFO estimation with respect to different input data truncation is shown in Figure 4. The Mean Squared Error (MSE) of estimated frequency offset is simulated for

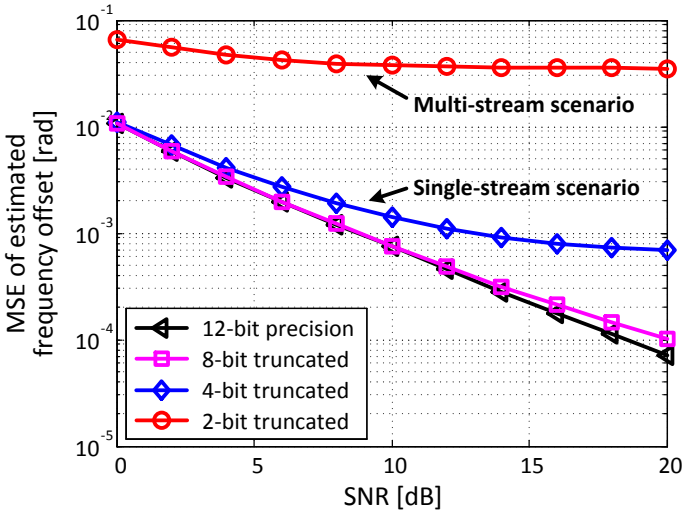


Figure 4: Analysis of input data truncation in CFO estimation for 3GPP LTE with a frequency offset of $\pi/8$.

an Additive White Gaussian Noise (AWGN) channel on an LTE transmission with a frequency offset of $\pi/8$. Although higher data wordlength attains higher processing accuracy, larger input truncation reduces both hardware complexity and memory size. For example, reducing input wordlength from 12 to 4 bits results in a performance degradation of around 0.66×10^{-3} radians at a Signal-to-Noise Ratio (SNR) of 10 dB. This corresponds to a frequency error of $0.66 \times 10^{-3}/(2\pi) \times \Delta f = 1.58$ Hz. Further truncation to 2 bits results in more than 80% memory reduction at the cost of 87.26 Hz frequency error at the same SNR. Since a maximum frequency error of 2 KHz can be tolerated by the receiver in LTE [94], this wordlength reduction is motivated. The same analysis are applied to other radio standards, and the results show that quantization noise due to wordlength reduction is negligible.

3 Hardware Development

Figure 5 shows a block diagram of the reconfigurable cell array deployed in the DFE-Rx for performing OFDM time synchronization and CFO estimation. Based on the operation analysis (Figure 3), the cell array is configured to have two processing and two memory cells. The processing cells are used to perform data operations shown in Figure 3, while the memory cells serve as correlation and moving-sum FIFOs as well as communication buffers between processors.

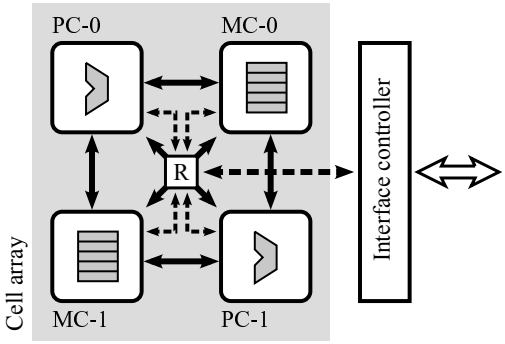


Figure 5: Block diagram of the 2×2 cell array and the interface controller deployed in the synchronization block of DFE-Rx. Solid and dashed lines depict local and hierarchical network interconnects, respectively.

The interface controller, connected to the cell array via hierarchical network interconnects, manages external data communication to other system blocks in the DFE-Rx and is responsible for static configurations of RCs. To cope with multi-standard concurrent data computations, both processing and memory cells in the baseline architecture described in Part I are further developed. The following sections present the detailed architectural improvements.

3.1 Dataflow Processor

In multi-standard multi-stream applications, concurrent processing calls for a processor design that suffices different computational requirements on each individual data stream. Processing cells employed in this work, named as dataflow processors, are Reduced Instruction Set Computing (RISC) cores with improved dataflow control. In addition to the functions equipped in a generic processing cell (Part I), the dataflow processor enhances data processing by supporting Single Instruction Multiple Data (SIMD)-like operations. The processor contains multiple processing lanes capable of performing both complex- and real-valued operations, illustrated in Figure 6. These operations are required by, for example, ML estimation and CORDIC computations, respectively. Taking a 16-bit 4-lane processor as an example, the processing lanes can be grouped into 2 or 4 computation paths, capable of executing 8-bit complex-valued or 16-bit real-valued operations. Figure 7 depicts detailed architecture of the arithmetic part of Arithmetic Logic Unit (ALU) in the 16-bit processor. Basic operations of the ALU are controlled by two mode specifiers, “multiplication” and “vector”. While the former one switches between addi-

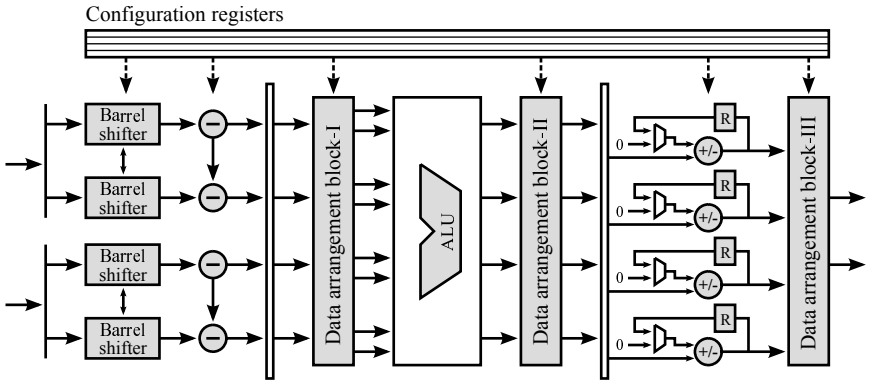


Figure 6: Computation path of the dataflow processor. Configurations of the shaded blocks are stored in registers that are run-time accessible.

tion and multiplication mode, the latter one controls real- or complex-valued operations. Real-valued output is obtained by concatenating results from 'O_3' and 'O_4', while the real and imaginary part of complex-valued output are taken from 'O_1' and 'O_4', respectively.

In addition, computational units are extended to both "instruction decode" and "write back" stage of the processor. As a result, several consecutive data manipulations can be accomplished in a single instruction execution without storing intermediate results. This substantially reduces register accesses. An example of the consecutive operation execution can be found during the computation of (2), where input data conjugate and result accumulation need to be performed before and after complex-valued multiplication, respectively. Without this extended computation capability, (2) needs to be computed iteratively, requiring 3 times more execution clock cycles and register accesses. Moreover, each arithmetic- and logic-type instruction in the dataflow processor is extended to have two operation codes (opcodes), which are capable of performing two different operations on the same input data operands in each clock cycle. The widely used butterfly operation (simultaneous add and subtract [95]) in FFT is a typical example of the dual-opcode instruction. Another example is input data forwarding during computations. This can be used to hide the execution time of data movement operations. For example, input data samples in (2) can be forwarded to other processing or memory cells while being multiplied and accumulated. The complete instruction set of the dataflow processor is included in Appendix A, Figure 1-3 and Table 1-2.

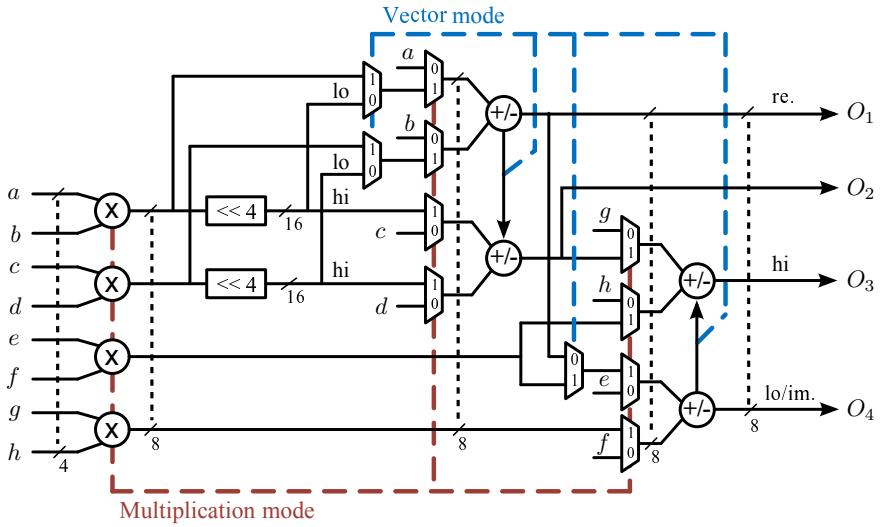


Figure 7: Arithmetic part of the ALU in the dataflow processor, an example of 16-bit case. Real-valued output is taken from 'O_3' and 'O_4', while complex-valued output is drawn from 'O_1' and 'O_4'.

Data Stream Shuffling

For efficient usage of multiple processing lanes, the capability of redirecting input operands into either lane is vital. In the dataflow processor, data operands in each computation stage can be shuffled before and after each operation. Internal data shuffling is carried out by deploying data arrangement blocks at the computation stages, illustrated in Figure 6. In this work, data arrangement blocks are built from multiplexers. Control bits of the multiplexers are stored in configuration registers that are transparently accessible by the user. Different data path configurations can be preloaded into the registers before executing a program or updated dynamically via a special instruction. The stored data path configurations can be applied to any type of instructions, which is accomplished by indexing the configuration registers in each instruction. As an example, trivial multiplications required in an FFT may be implicitly executed using data arrangement blocks, as the operations are equivalent to swapping two input operands without any data manipulation. Thus, specific operations to compute trivial multiplications are avoided, resulting in reduced execution clock cycles and program count. Moreover, with the help of data arrangement blocks, the same instruction can be used to perform different operations. This is accomplished by shuffling input operands to form different data patterns.

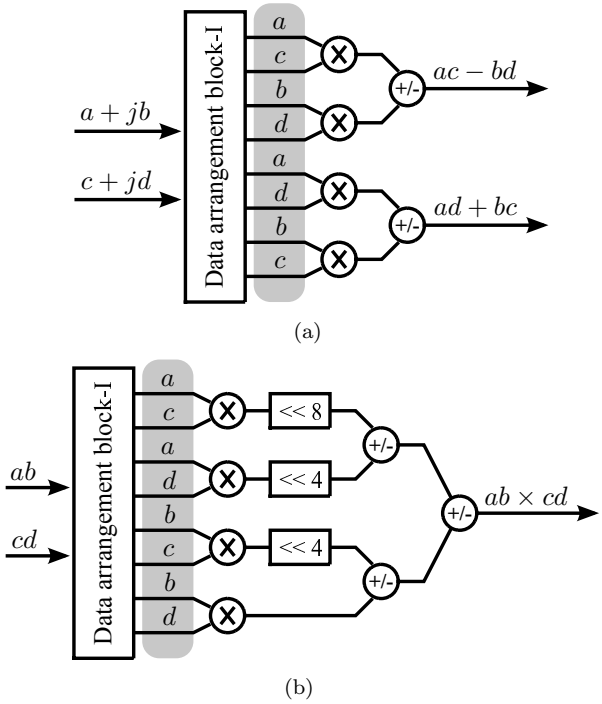


Figure 8: Input data operand arrangements in (a) a 4-bit complex-valued multiplication, (b) an 8-bit real-valued multiplication. These two operations use the same instruction and data inputs but operate on different data sequences, as highlighted in the figure.

As an example, real- and complex-valued multiplications share the same instruction but operate on two different data sequences, illustrated in Figure 8. Detailed architecture of data arrangement blocks and the set of configuration codewords can be found in Appendix A, Figure 4 and Table 3-6.

3.2 Memory Cell

Under multi-standard multi-stream scenarios, memory descriptors are shared by the processing of multiple data streams. To cope with various sample-rate of standards, it is crucial that memory descriptors can be executed in a non-sequential order. Otherwise, data stream with the slowest sample-rate will block entire data processing.

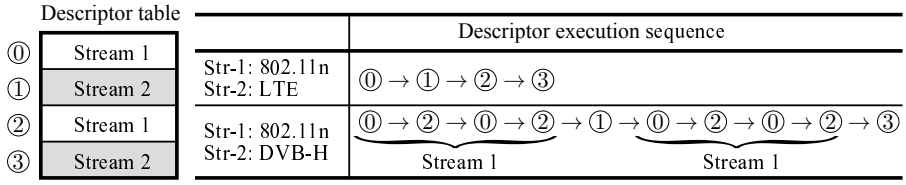


Figure 9: Illustration of descriptor execution program during concurrent multi-stream processing.

Flexible Descriptor Execution

Based on the aforementioned analysis, memory descriptors are extended in the way that they can be configured to execute either in *non-blocking* or *blocking* mode. In non-blocking mode, the operation controller of the memory cell sequentially starts a descriptor execution in each clock cycle, without waiting for response from data receiver regarding last memory access. Therefore, subsequent descriptors can still be issued and executed even if the current one is being blocked. Besides used in multi-stream processing, the non-blocking execution mode is also useful when one memory cell is shared among several hosts (e.g., processing cells) operating on different stream transfer rates. In contrast, blocking execution mode guarantees the completion of each specified memory access before starting a new descriptor execution. This mode can be used to avoid mixing up stream transfers when an I/O port is shared among multiple memory descriptors.

To further improve the flexibility of memory cells, the order of the descriptor execution is run-time programmable. This way, multiple descriptors can be arranged to reorder or repeat data sequences, or to cope with data streams that have different transfer rates. Figure 9 illustrates the use of descriptor execution program during concurrent multi-stream processing. Assuming that the memory cell has four descriptors, which are configured to serve for two different streams in an interleaved manner, namely ⓪ and ② for "stream 1" and ① and ③ for "stream 2". During the processing of IEEE 802.11n and LTE, which both have an oversampling rate of 1, the four descriptors are executed sequentially. However, when dealing with IEEE 802.11n and DVB-H, execution sequence needs to be programmed such that data stream of IEEE 802.11n is processed four times before performing one DVB-H data reception. This is due to the fact that data stream of DVB-H has an oversampling rate of 4, as mentioned in Section 1.

Micro-block Fuction

In addition to the flexible descriptor execution, the data access pattern of a memory cell can be reshaped by using a micro-block function. This enables memory access with finer wordlength than that a physical memory provides. For example, a 32-bit wide memory cell can be configurable to behave as two 16-bit wide or four 8-bit wide memory cells. This feature is useful when supporting multi-standard data processing, as different standards may intrinsically require different processing wordlength. Detailed usage of the micro-block function is further illustrated in Section 4.

A micro-block operation is defined by a *block size*, *stride*, *read and write pointer*, and *data mask*. The block size specifies the wordlength of a micro-block, used to determine the number of data accesses in each memory read and write operation. For a 32-bit memory cell, options of the micro-block size are 1, 2, 4, 8, 16, and 32 bits. Stride is the distance to the next micro-block, measured in bits. Read/write pointers are physical memory addresses and are automatically updated after each operation. Data mask enables bitwise operation on data that is read from or to be written to the memory. Table 3 summarizes configuration fields of the memory descriptor, which is an extended version of the one adopted in the baseline memory cell (Part 3.2).

4 Implementation Results and Discussion

Given that the cell array is aimed to be deployed in mobile terminals, a moderate clock frequency of around 300 MHz is expected. Considering the highest data sample-rate among the three standards (40 MHz in IEEE 802.11n), operations assigned to each processing cell must be completed within 8 clock cycles ($\lceil 300\,\text{MHz}/40\,\text{MHz} \rceil$) in order to suffice real-time constraints. In this work, concurrent processing of two data streams is accomplished by time-multiplexing two streams on the cell array. As a consequence, the required execution time is further reduced by half.

Task-level Pipeline

To meet the stringent timing constraint, data computations are partitioned and mapped onto different processing cells. Specifically, the correlation and the peak detection in Figure 3 are assigned to PC-0 and PC-1 respectively, while the CFO estimation is carried out on both processors after determining the position of the correlation peak. As mentioned in Section 2.1, the phase computation (4) in the CFO estimation can be performed using a CORDIC algorithm [92]. The concept of the CORDIC algorithm is to rotate input vector through a series of micro rotations by applying shift and add operations [73].

Table 3: Configuration fields of memory descriptor.

	Field	Bits	Length	Description
Part I	dtype	31-30	2	Operation mode select
	base	29-20	10	Start address
	high	19-10	10	End address
	rd_ok/active	9	1	FIFO reading status/ RAM active transfer flag
	wr_ok/rnw	8	1	FIFO writing status/ RAM read-write select
	io_bank_rst	7	1	I/O port register reset
	id	6-1	6	Global packet destination ID
	block_opr	0	1	Blocking execution enable
Part II	rptr/ptr	31-22	10	Current FIFO reading pointer/ Current RAM data pointer
	wptr/tsize	21-12	10	Current FIFO writing pointer/ Current RAM data transfer size
	src/paddr	11-8	4	FIFO data source port/ RAM address port
	dst/pdata	7-4	4	FIFO data destination port/ RAM data port
	blk_size	3-1	3	Size of micro-block
	blk_en	0	1	Micro-block enable
Part III	blk_stride	31-27	5	Micro-block step size
	blk_rptr	26-22	5	Current micro-block read pointer
	blk_wptr	21-17	5	Current micro-block write pointer
	blk_mask	16-1	16	Micro-block data mask
	blk_mask_sign	0	1	Data mask sign extension

These operations can be easily mapped onto processing cells by using barrel shifters and ALU as well as data arrangement blocks for intermediate result shuffling. Memory cells interconnected with the two processors are used as correlation and moving-sum FIFOs. Additionally, MC-1 serves as the CORDIC coefficient ROM and communication buffers between the processors.

Based on the wordlength analysis in Section 2.2, the two processing cells are configured as 16-bit cores suitable for handling correlations with 4-bit complex-valued inputs (see Figure 7). The wordlength of memory macros in both mem-

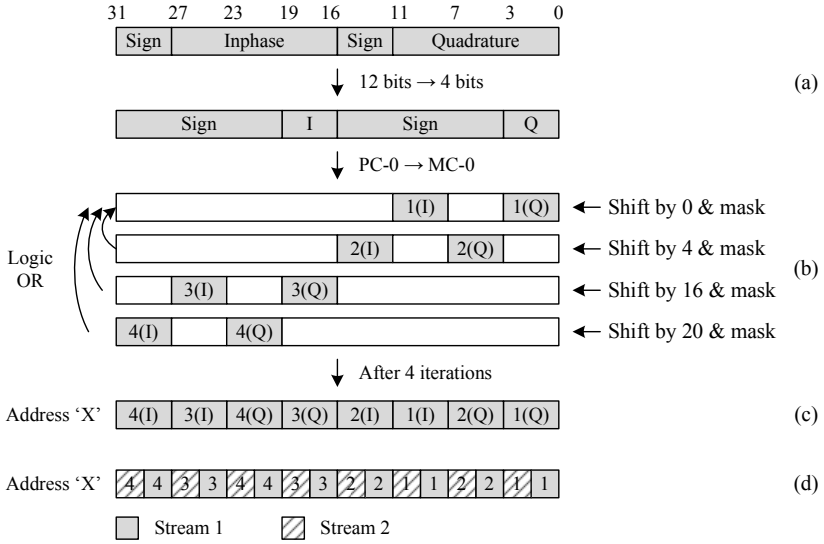


Figure 10: Interleaved data storage in correlation FIFO, MC-0. (a) Received 12 bits data pair in PC-0. Data pairs are truncated down to 4 bits before being transmitted to MC-0. (b) Exploded view of data storage in correlation FIFO for single data stream. (c) Final data storage at address 'x' for single data stream. (d) Final data storage at address 'x' for two concurrent data streams (2 bits data pair of each).

ory cells is 32 bits wide. Therefore, a pair of 16 bits in-phase and quadrature data inputs can be stored in the same memory location. Memory capacity of the correlation and the moving-sum FIFO is configured to suffice the standard with the largest storage requirement, i.e., DVB-H in this case.

Memory Interleaving

In addition to the task-level pipeline, self-governed FIFO operations in memory cells relieve processing cells from address manipulations. Moreover, the micro-block function of memory cells eliminates data alignment operations in processors, as illustrated in Figure 10. Since the wordlength of memory macros is 32 bits wide and the complex-valued data inputs are truncated to 4 bits (in single-stream mode), 4 truncated data pairs need to be stored at one memory location. With the help of the micro-block function, PC-0 is exempt from data shifting and masking operations, as shown in Figure 10(b). The received data pairs are correspondingly left-shifted, masked to set off unused bits, and added

(logic OR) to the write buffer. A final view of the interleaved data storage at a memory location in the correlation FIFO is illustrated in Figure 10(c). Similarly, when data pairs are truncated to 2 bits in the multi-stream mode, leading to 8 micro-block operations, truncated data pairs from both streams are interleaved and stored in the same memory location, see Figure 10(d).

Context Switching

During context switching between different radio standards, both processing and memory cells need to be reconfigured to update parameters such as the threshold value T in the ML estimation (1), the length of the correlation (M), and the moving-sum FIFO (L). When switching from single- to multi-stream processing or vice versa, additional configurations are required. These include memory bank allocations, computational precision adjustments (e.g., from 4 bits down to 2 bits), and corresponding program segment updates in processing cells. Thanks to the adoption of the in-cell configuration scheme presented in Part I, all the aforementioned context switching tasks are conducted inside the cell array during system run-time. In the current implementation, PC-0 is used as a local master core that controls the operation of other RCs and manages all the required resource configurations.

Configuration Generator

To ease resource configurations of the cell array, a graphical user interface is developed in-house, shown in Figure 11. This tool visualizes all possible configurations of the processing and the memory cells deployed in the 2×2 cell array. Additionally, it is able to convert resource configurations into binary codes and generate a final bit stream based on allocated address space of each RC. Moreover, this tool has the ability of streaming data and configurations into the cell array (with the help of peripheral circuits, further discussed in Section 4.3) using a TCP/IP socket. A detailed view of the tool can be found in Appendix A, Figure 5-7. Note that the memory cell MC-1 is shown as two separate units in Figure 11, "MC 1" and "MC 2", which are used as data memories (moving-sum FIFO and CORDIC coefficient ROM) and communication buffers, respectively.

4.1 Hardware Flexibility

With a fixed memory size, concurrent data processing is achieved by sharing memory resources between multiple data streams. Memory sharing is usually accomplished by sacrificing computational precision on all data streams, regardless of running standards and channel condition. Although the rigid uniform wordlength scheduling is easier to implement, higher computational

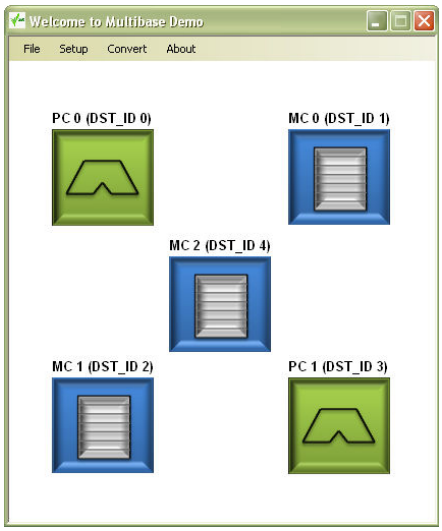


Figure 11: Overview of the configuration generation tool.

precision is desired whenever possible. The cell array is capable of dynamically allocating computational resources to achieve better performance and resource utilization. Hence, computational precision is scheduled adaptively depending on current operating conditions. For the target radio standards, input samples are truncated to either 4 or 2 bits. Table 4 lists all the radio standards supported by the presented 2×2 cell array with the corresponding truncation wordlengths. Memory utilization, shown in the last column of Table 4, is the data memory used by the correlation process as a percentage of the total data storage available in all memory cells. The utilization does not reach 100%, as parts of the data memories are used as the CORDIC coefficient ROM, as well as communication buffers between two processors.

In addition to resource sharing among multiple radio standards, system flexibility is also demonstrated by mapping different algorithms onto the same platform, without additional hardware cost. The adoption of different algorithms may either extend system compatibility by supporting additional standards, or improve system performance by enhancing processing throughput and concurrency. In the conducted experiment, flexibility is illustrated by mapping a novel sign-bit OFDM synchronization algorithm [93] onto the presented cell array. This leads to the support of all three OFDM transmission modes (2K, 4K, and 8K subcarriers) in the DVB-H standard. The initial design choice on memory capacity only supports 4/2-bit synchronization algorithm for DVB-

Table 4: Supported radio standards and memory utilization of a 2×2 cell array.

Concurrency	Standards	Truncation wordlength	Memory utilization
Single-Stream	802.11n	4 bits	8.48%
	LTE	4 bits	65.18%
	DVB-H 2K	4 bits	85.71%
	DVB-H 4K	2 or Sign bit	85.71%
	DVB-H 8K	Sign bit	85.71%
Dual-Stream	802.11n & 802.11n	4 bits	16.96%
	802.11n & LTE	2 bits	45.09%
	802.11n & DVB-H 2K	2 bits	65.63%
	LTE & LTE	2 bits	73.21%
	LTE & DVB-H 2K	2 bits	93.75%

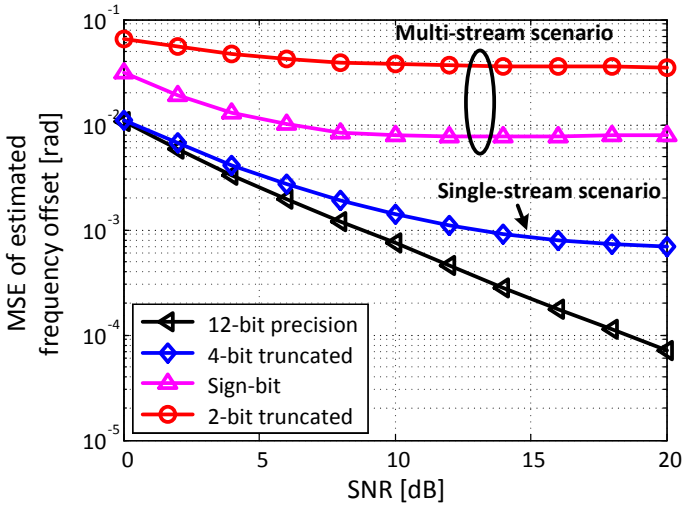


Figure 12: Analysis of sign-bit algorithm [93] in CFO estimation for 3GPP LTE with a frequency offset of $\pi/8$.

H 2K/4K modes. However, with the adoption of the sign-bit algorithm, which has dramatic data storage reduction, support for the DVB-H 8K mode becomes possible. Performance analysis of the sign-bit algorithm, see Figure 12, reveals better computational accuracy than the 2-bit implementation. This is in virtue

Table 5: Area breakdown of the DFE-Rx.

DFE Rx	Area [μm^2]	Percentage
I/O Pads	991,569	38.37%
Synchronization block	479,026	18.54%
Rx data path 1	501,894	19.42%
Rx data path 2	501,894	19.42%
Others	109,851	4.25%
Total	2,584,234	100.00%

of specialized arithmetic used in the sign-bit algorithm, which obtains better immunity to quantization noise [94]. However, sign-bit implementation involves many bit-level data manipulations, which are difficult to map efficiently to a coarse-grained reconfigurable architecture without increasing execution time. Therefore, design complexity in the sign-bit algorithm shifts from memory capacity to data processing.

Figure 13 and 14 illustrate the layout of data storage in the correlation FIFO (MC-0) for various use cases studied in this work. The considered use cases include different standards in signal-stream mode and various standard combinations in multi-stream mode. Streams marked in red in Figure 13 and 14 indicate the first data stream received by the cell array. Note that the concurrent reception of IEEE 802.11n data streams can be processed in either 4 or 2 bits, because of the low memory requirements, see Table 2.

4.2 Implementation Results

Fabricated in a 65 nm CMOS technology, the DFE-Rx has a die size of 5 mm^2 with 144 I/O pads. According to synthesis results, half of the chip area is taken by memories, logic cells and I/O pads, while the remaining half is used for power and signal routing as well as clock tree generation. Figure 15 shows a chip layout. As a prototype, the design is pad-limited due to the large number of I/O ports required for individual tests of function blocks. Table 5 shows an area breakdown of the DFE-Rx. As can be seen, I/O pads occupy about 40% of the area and the remaining part is evenly distributed among the synchronization block and the two receiving data paths. In the following, we focus on the implementation of the synchronization block, namely the 2×2 cell array, and present silicon measurement results obtained from a standalone test.

Shown by the synthesis results in Table 6, the cell array is memory dominant, which consumes about 40% of the area. This is mainly due to the large amount of data required to store. Besides, program memories of processing

Table 6: Area breakdown of the 2×2 cell array in the DFE-Rx.

Resource cell		Memory		Area [μm^2]	Percentage
PC-0	Logic	–	–	37,567	7.84%
	Memory	384×48b	18 Kb	37,009	7.73%
PC-1	Logic	–	–	37,201	7.77%
	Memory	512×48b	24 Kb	41,318	8.63%
MC-0	Logic	–	–	39,167	8.18%
	Memory	512×32b	16 Kb	66,016	13.78%
MC-1	Logic	–	–	64,810	13.53%
	Memory	384×32b	12 Kb	57,657	12.04%
Router cells		–	–	37,976	7.93%
Interface controller		–	–	60,306	12.59%
Total			70 Kb	479,026	100.00%

cells are deployed such that they are large enough to allow further algorithm updates and mapping of other tasks. The entire cell array, including the interface controller, occupies 0.48 mm^2 area and has a maximum clock frequency of 534 MHz. Thanks to the adopted in-cell configuration scheme, switching between different operation modes, such as from OFDM time synchronization to CFO estimation, requires only 11 clock cycles.

The high system flexibility offered by the cell array comes at the cost of area overhead. For comparison, a hardware accelerator only capable of performing the target tasks (OFDM time synchronization and CFO estimation) is implemented. Synthesis result of the accelerator is shown in Table 7. Even though a module-by-module comparison is not possible because of the hardware reusing nature of the reconfigurable architecture, it is evident that both designs are memory dominant. To be able to process two concurrent data streams, two accelerators are taken into the comparison. It shows that the accelerator-based solution uses around 4 times less silicon area and runs at a lower clock frequency (40 MHz) for the same throughput. However, from the entire DFE-Rx point of view, the adoption of the cell array results in only about 16% area overhead. In view of the high flexibility provided by the cell array, as demonstrated in Section 4.1, this overhead is acceptable. Moreover, it should be pointed out that the potential usage of the cell array is not fully explored when only evaluating the mapping of the synchronization algorithms. The architecture has the ability of being dynamically reconfigured to perform different tasks while a hard-coded accelerator implements fixed functionality.

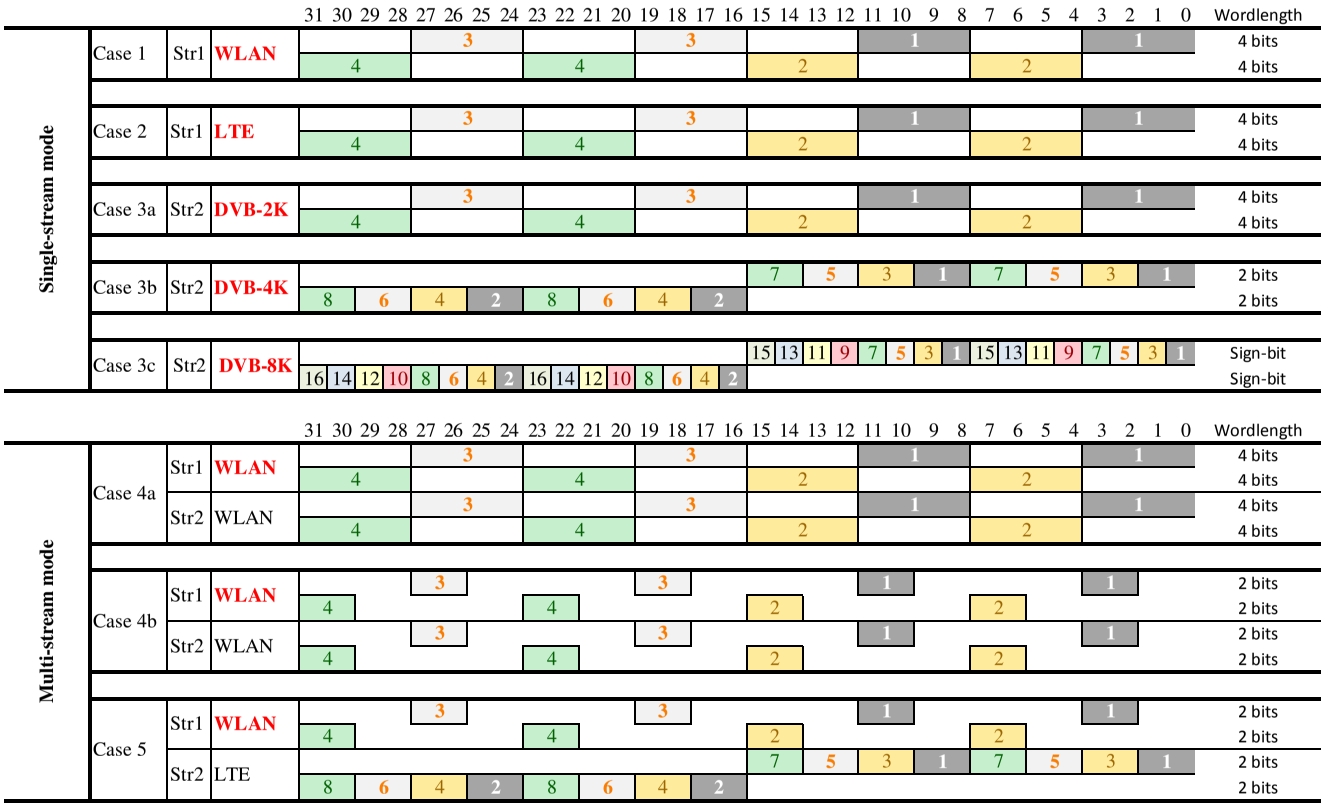


Figure 13: Layout of data storage in the correlation FIFO (MC-0) for different use cases. Streams marked in red indicate the first data stream received by the cell array.

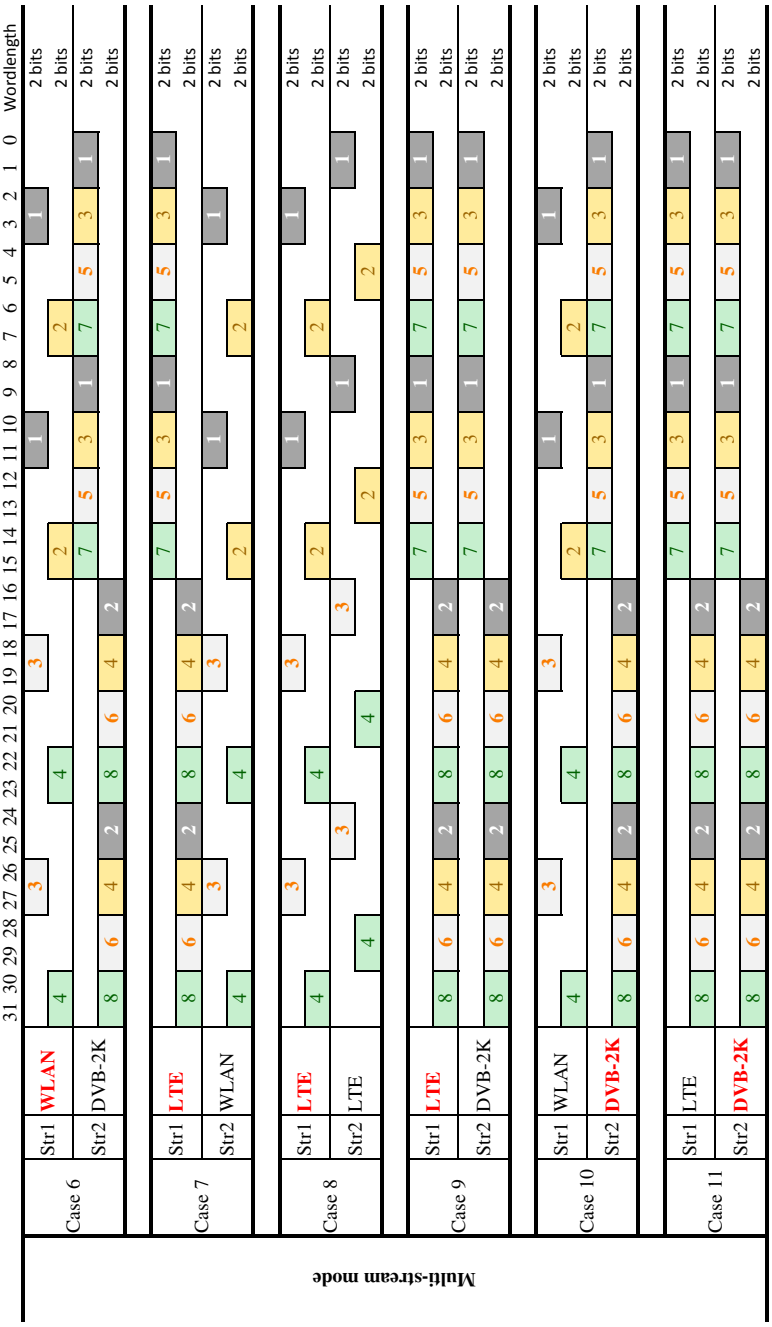


Figure 14: Layout of data storage in the correlation FIFO (MC-0) for different use cases, continued.

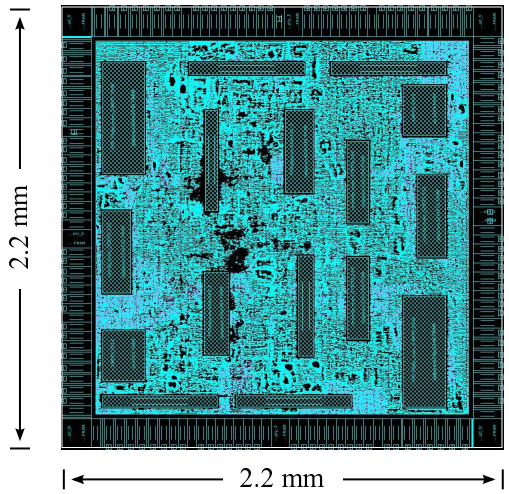


Figure 15: Chip layout of the fabricated DFE-Rx.

Table 7: Synthesis result of a hardware accelerator, only capable of performing time synchronization and CFO estimation for a single data stream.

	Area [μm^2]	Percentage
Correlator	1,296	2.13%
Peak detector	2,305	3.78%
Correlation FIFO	27,232	44.69%
Moving-sum FIFO	25,258	41.45%
CORDIC (time-multiplexed)	3,330	5.46%
Control	1,521	2.5%
Total	60,942	100.00%

4.3 Measurement Results

To verify the functionality of the cell array, a standalone test is carried out in the debugging mode of the DFE-Rx via an on-chip Serial DeBug (SDBG) interface. The SDBG interface, developed based on [96], contains a set of light-weight single-ended serial links capable of operating at 10 Mbps when using ribbon cable connections. Higher speed, up to 40 Mbps, can be achieved with good signal termination and PCB board layout.

Debugging Interface

Typical high speed data-recovery circuits require a Phase-Locked Loop (PLL) module for each serial link to recover data (as well as clock) from an 8b/10b encoded serial link. However, it has been shown in [96] that data can be recovered from a serial link by simply using a 4× clock sampling scheme without the 8b/10b encoding. Additionally, instead of using PLL, the presented data-recovery circuit employs two local clock signals: 'clk' and its 90-degree phase shifted counterpart 'clk90'. These clock signals can be generated from two independent local oscillators or clock generation circuits. However, it is important to maintain the phase-relationship of the two clocks. According to [96], this scheme can recover data from up to 200 Mbps serial links by using the differential signalling for serial transmission and advanced Delay-Locked Loop (DLL) circuit to maintain the phase-relationship of clock signals. The current version of DFE-Rx has no differential I/O pads and DLL circuits equipped. Thereby, these serial links are implemented with single-ended I/O pads. Both clock signals ('clk' and 'clk90') are provided from normal clock pads and are directly used inside the chip without further phase-adjustment.

The DFE-Rx SDBG consists of three serial links: ASIC Control Input Link (AIL-C), ASIC Data Input Link (AIL-D), and ASIC Output Link (AOL). The AIL-C and AIL-D are used to stream configurations and data into the cell array respectively, while AOL is shared for both data and control outputs.

Standalone Cell Array Test

Through the SDBG interface, the cell array is connected to an FPGA platform, Xilinx XUPV5-LX110T, which implements the control and data streaming logics for communicating with the cell array. Figure 16 and 17 illustrate the setup and the measurement testbed for the standalone test, respectively.

For the system running in FPGA, a 32-bit MicroBlaze soft processor core [97] is used as a master controller. The SDBG interface adaptor is embedded as a co-processor connected on a shared Processor Local Bus (PLB). An interrupt controller is used to notify the master controller to receive control/data from the cell array. Communication to an external host (PC) is achieved through an Ethernet connection and a Universal Asynchronous Receiver/Transmitter (UART) interface, which stream data/configuration and issue control commands to the cell array, respectively.

In the standalone test, the embedded system in FPGA operates at 100 MHz, whereas the cell array is clocked at 10 MHz due to the data-rate limitations on ribbon cable connections and single-ended signalling. Data transmission rate over the Ethernet connection is set to 100 Mbps, while the UART line adopts a baud rate of 460.8 Kbps.

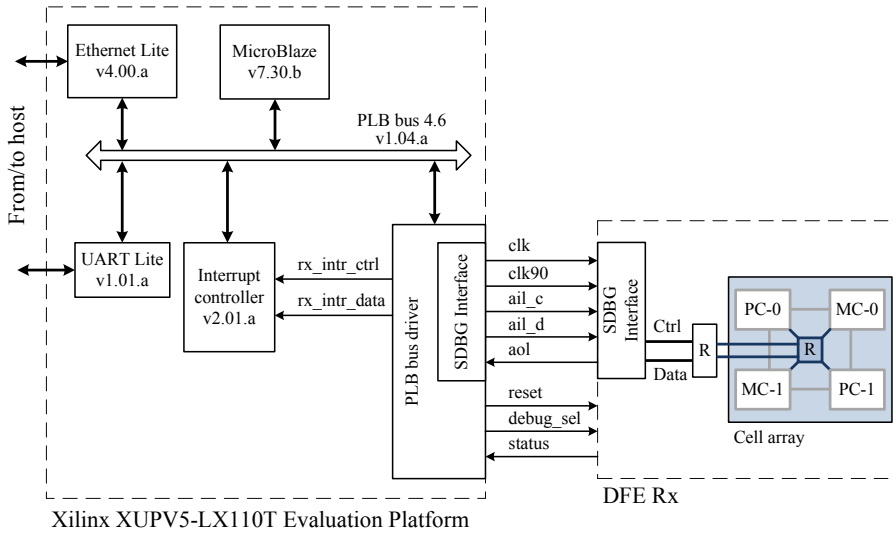


Figure 16: Block diagram of the standalone cell array test.

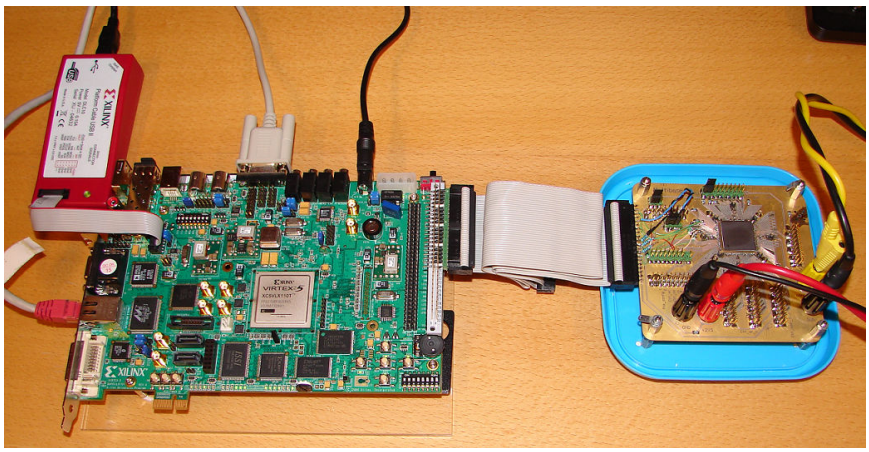


Figure 17: Measurement testbed for the standalone cell array test.

To provide users with an easy way of controlling the embedded system, a user interface in UART line is developed. Resource cell configurations and control and data inputs can be streamed into the cell array by issuing different user commands, see Appendix A, Table 7. In addition, a few pre-loaded configuration scripts are provided for fast system demonstrations. Burst data

Table 8: Example of cell array configuration via a UART interface.

≫ @g#	(Command input) Command 'g', destination RC selection.
≫ 0	(Number input) RC hierarchical IO port ID.
≫ @i#	(Command input) Command 'i', instruction downloading.
≫ 2	(Number input) Number of instructions (plus one) to transfer.
≫ $00010002#	(String input) Header of instruction downloading.
≫ $A8000001#	(String input) Instruction "A8000001".

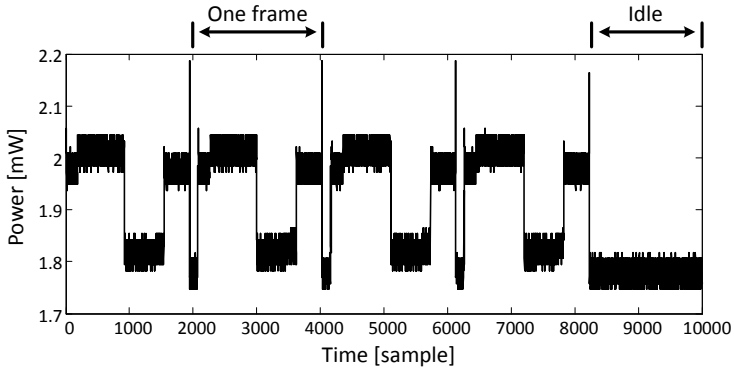


Figure 18: Measured power consumption of the cell array in a standalone test mode when processing an IEEE 802.11n data reception.

transfers are accomplished by sending a user-defined script file. As an example, commands shown in Table 8 download an instruction into PC-0.

Based on the UART control, a high-level user interface is designed in MATLAB. The MATLAB interface provides a more advanced and flexible way to control data streams running into and out of the system. For example, input data sequences can be generated in MATLAB at run-time and data produced by the cell array can be collected and plotted graphically. Detailed user commands in the MATLAB interface are listed in Appendix A, Table 8.

Figure 18 shows the power consumption of the cell array measured under the processing of an IEEE 802.11n data stream at nominal supply voltage of 1.2 V and at 10 MHz clock frequency. During the reception of 802.11n data frames, the measured minimum and maximum power consumption is 1.75 mW and 2.19 mW, respectively. During the loading of hardware configurations (not shown in Figure 18), the cell array consumes 1.95∼2 mW power.

5 Summary

This part presents a case study of the reconfigurable cell array suitable to process multiple radio standards concurrently. The flexibility and performance of the architecture are demonstrated by performing time synchronization and carrier frequency offset estimation in OFDM systems. Using a 2×2 cell array, three widely used standards, IEEE 802.11n, 3GPP LTE, and DVB-H, are supported. Moreover, two independent data streams from the three standards can be processed concurrently by allocating and sharing system resources at run-time. During the reception of a single data stream, the cell array is configured to achieve high computational accuracy by using all available hardware resources. When two concurrent data streams are being received, the cell array adjusts its hardware resources such that data memories are split in two and processing cells are shared over time. Moreover, the potential of the architecture is further illustrated by mapping a different algorithm onto the same platform without any additional hardware cost. Benefiting from the new algorithm mapping, the coverage of the standards supported by the cell array is extended. The proposed 2×2 cell array is fabricated as a part of a Digital Front-End Receiver in the EU Multibase project. Running at 10 MHz clock frequency and at 1.2 V supply voltage, measurement results report a maximum power consumption of 2.19 mW during the processing of an IEEE 802.11n data reception.

Part III

Multi-task MIMO Signal Processing

Abstract

Driven by the requirement of multi-dimensional computing in contemporary wireless communication technologies, reconfigurable platforms have come to the era of vector-based architectures. In this part, the reconfigurable cell array developed in Part I and Part II is extended with extensive vector computing capabilities, aiming for high-throughput baseband processing in MIMO-OFDM systems. Besides the heterogeneous and hierarchical resource deployments, a vector-enhanced SIMD structure and various memory access schemes are employed. These architectural enhancements are designed to suffice stringent computational requirements while retaining high flexibility and hardware efficiency. Implemented in a 65 nm CMOS technology, the cell array occupies 8.88 mm^2 core area. To illustrate its performance and flexibility, three computationally intensive blocks, namely channel estimation, channel pre-processing, and symbol detection, of a 4×4 MIMO processing chain in a 20 MHz 64-QAM Long Term Evolution-Advanced (LTE-A) downlink are mapped and processed in real-time. Operating at 500 MHz and 1.2 V voltage supply, the achieved throughput is 367.88 Mb/s and the average power consumption is 548.78 mW. The corresponding energy consumption for processing one information bit is 1.49 nJ/b. Comparing to state-of-the-art implementations, the proposed solution outperforms related programmable platforms by up to 6 orders of magnitude in energy efficiency, and is 1.7−13.6 and 1.4−15 times less efficient than ASICs in terms of area and energy, respectively, when performing each individual task.

1 Introduction

Given data receptions processed in DFE (Part II) and transformed back to the frequency domain via FFT [73], this part focuses on succeeding blocks in the baseband processing chain and considers systems employing MIMO and OFDM technologies. In addition, 3GPP Long Term Evolution-Advanced (LTE-A) is used as a case study to illustrate architectural development of the reconfigurable cell array. Support for other or multiple radio standards can be developed using the similar approach. However, the focus of this work is on vector enhancements of the cell array and concurrent processing of multiple tasks.

Compared to single antenna systems, MIMO technology exploits design-of-freedom in the spatial domain in addition to time and frequency. Therefore, it provides significant improvements in system capacity and link reliability without increasing bandwidth. However, the price-to-pay is higher complexity and energy consumption due to increased computational dimensions, i.e., in proportion to the antenna size, and required sophisticated signal processing, such as symbol detection for inter-antenna interference cancellations. Moreover, when combining MIMO with OFDM, it is required to perform the corresponding processing at every OFDM subcarrier, posing even more stringent computational and energy requirements. Under such circumstances, building architectures solely on scalar-based function units requires a large number of resource deployment. Although the adoption of massive scalar units provides high flexibility, it reveals poor hardware efficiency in view of the parallel-structured MIMO-OFDM processing, since a large portion of resource controls (e.g., processor instructions and memory configurations) are identical and thus redundant. To achieve efficient computing, it is crucial to extend architectures with vector processing capabilities by fully utilizing the extensive Data-Level Parallelism (DLP) available in MIMO-OFDM systems.

In this part, the reconfigurable cell array presented in Part I and Part II is further developed, aiming at achieving a balance between processing performance, flexibility, and hardware efficiency. Specifically, the architecture is partitioned into distinct vector and scalar processing domains for efficient hybrid-format data computing. In the vector domain, processing cells are deployed with vector-enhanced SIMD cores and VLIW-style multi-stage computation chains to attain low-latency high-throughput vector computing. Memory cells are equipped with flexible vector access schemes for relieving non-computational address manipulations from processing cells. For performance evaluation, three tightly coupled baseband processing blocks, which are unique and crucial in MIMO for exploiting its full superiorities, are mapped onto the cell array in a time-multiplexed manner. The three processing blocks are:

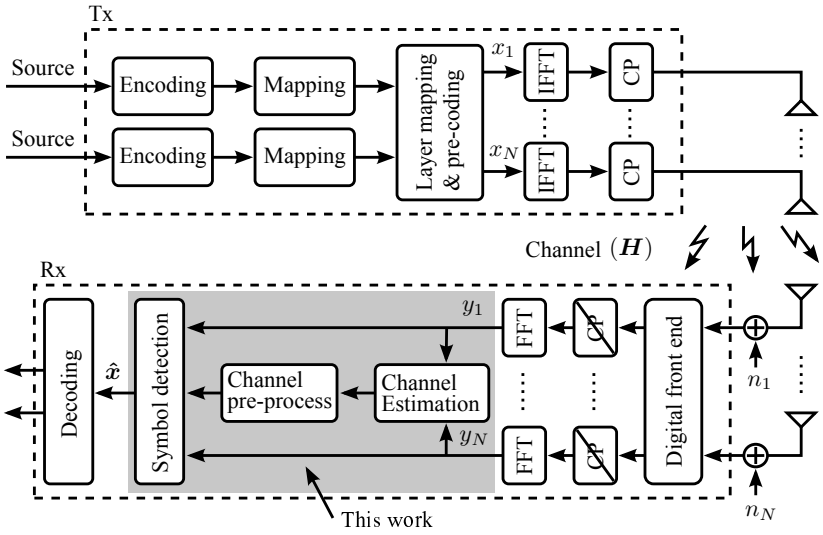


Figure 1: Block diagram of the MIMO-OFDM transceiver. This work maps all three shaded blocks onto the reconfigurable cell array.

- *Estimation* of the channel state information using pilot tones,
- *Channel pre-processing* that is an indispensable step for all kinds of detection algorithms,
- *Symbol detection* that recovers the transmitted vector.

Figure 1 shows a simplified diagram of the MIMO-OFDM transceiver and highlights the target processing blocks. In addition to the vector extension and task-level multiplexing, hardware efficiency of the cell array is further improved by algorithm-level exploitation, in which more than 98% of the total operations involved in all three tasks are vectorized and unified, enabling extensive parallel processing and hardware reuse.

In Section 2, algorithm development for the MIMO processing tasks are presented, including Minimum Mean-Square Error (MMSE) based channel estimation, QR Decomposition (QRD) based channel pre-processing, and node-perturbation-enhanced MMSE symbol detection. In Section 3, performance, computational complexity, and hardware friendliness of the adopted algorithms are evaluated and analyzed in comparison to conventional approaches. Based on the operation profile and the LTE-A specification, data processing flow and timing analysis are conducted to provide guidance for succeeding hardware development. Section 4 describes the detailed array architecture configured for

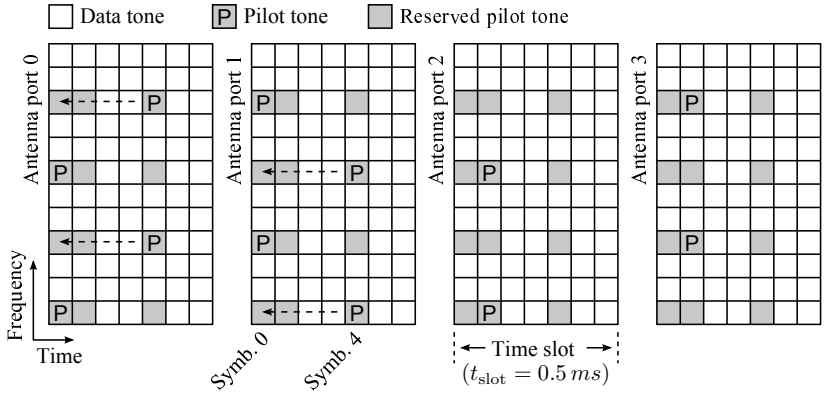


Figure 2: Scattered pilot pattern for four antenna ports in one LTE-A resource block.

MIMO-OFDM signal processing. The focus is on vector extension, including heterogeneous resource arrangement, processing cell enhancements, and various vector memory access schemes. Section 5 summarizes implementation results and compares performance with that of state-of-the-art platforms. Moreover, the flexibility of the proposed solution is further illustrated in Section 6 through a mapping of an adaptive channel pre-processor. The mapping makes use of dynamic resource allocations to adopt appropriate pre-processing algorithms at run-time, which provides a wide range of performance-complexity trade-offs. Section 7 concludes this part.

2 MIMO Signal Processing

This work considers a MIMO-OFDM system with N transmit (Tx) and receive (Rx) antennas. Without loss of generality, the following discussions are based on the consideration of 20MHz LTE-A downlink operating in normal Cyclic Prefix (CP) mode with 4×4 antenna setup and 64-QAM modulation. Similar to the previous part, the maximum excess delay of the propagation channel is assumed to be within the CP of each OFDM symbol. Before presenting the target MIMO processing algorithms, the LTE-A data structure is introduced and the system model described in Chapter 3 is briefly revisited.

Figure 2 shows the structure of a resource block in LTE-A. Each resource block contains 12 consecutive subcarriers and 7 OFDM symbols over a time slot of 0.5 ms. To support operations such as synchronization and channel estimation, pilot tones are distributed over a time-frequency grid. Under the common

assumption of quasi-static[1] channel modelling [98], the accuracy of channel estimates can be improved by inserting pilot tones in the middle of each time slot (symbol 4) into the pilot vector at symbol 0 [99]. This way, channel estimation is performed only once in each time slot, followed by channel pre-processing, while symbol detection is required on every data-carrying subcarrier.

Assuming perfect synchronization and front-end processing, the received vector $\boldsymbol{y}$ after CP removal and FFT can be expressed as

$$\boldsymbol{y} = \boldsymbol{H}\boldsymbol{x} + \boldsymbol{n}, \tag{1}$$

where $\boldsymbol{H}$ denotes the complex-valued channel matrix, $\boldsymbol{x}$ is the transmitted vector obtained by mapping a set of encoded information bits onto a Gray-labelled complex constellation, and $\boldsymbol{n}$ is the i.i.d. complex Gaussian noise vector with zero mean and variance σ_n^2. The average transmit power of each antenna is normalized to one, such that $\mathrm{E}\{\boldsymbol{x}\boldsymbol{x}^H\} = \boldsymbol{I}_N$, where $\boldsymbol{I}_N$ is an identity matrix of size N and $(\cdot)^H$ denotes a Hermitian transpose.

In the following, algorithms adopted for the three processing tasks are presented in detail. It should be pointed out that algorithm selections are not the main scope of this part. They are selected to make use of essential architectural characteristics and to illustrate the performance of the hardware platform.

2.1 Channel Estimation

Based on the scattered pilot arrangement (Figure 2), pilot-aided comb-type channel estimation scheme [100] is employed. It consists of two computation steps. First, channel coefficients at pilot positions are computed by using Least Square (LS) algorithm,

$$\boldsymbol{h}_{p,\mathrm{LS}} = \boldsymbol{y}_p \boldsymbol{x}_p^{-1}. \tag{2}$$

Second, channel coefficients at data-carrying subcarriers are estimated by interpolating and extrapolating $\boldsymbol{h}_{p,\mathrm{LS}}$ in the frequency domain. This process may be seen as a linear filtering of the LS estimation

$$\boldsymbol{h} = \boldsymbol{\mathcal{W}}\boldsymbol{h}_{p,\mathrm{LS}}, \tag{3}$$

where the filter function $\boldsymbol{\mathcal{W}}$ varies with different algorithms (mentioned in Chapter 3.2). Among them, Linear MMSE (LMMSE) estimation algorithm aims to approach the optimal result by using second-order statistics of the channel conditions and noise power. It is defined as

$$\boldsymbol{h}_{\mathrm{MMSE}} = \boldsymbol{\mathcal{W}}\boldsymbol{h}_{p,\mathrm{LS}} = \boldsymbol{R}_{\boldsymbol{h}_d\boldsymbol{h}_p}\left(\boldsymbol{R}_{\boldsymbol{h}_p\boldsymbol{h}_p} + \frac{\beta}{\mathrm{SNR}}\boldsymbol{I}_N\right)^{-1}\boldsymbol{h}_{p,\mathrm{LS}}, \tag{4}$$

[1] Channel coefficients of each subcarrier are stationary over time within one time slot, i.e., 0.5 ms in LTE-A.

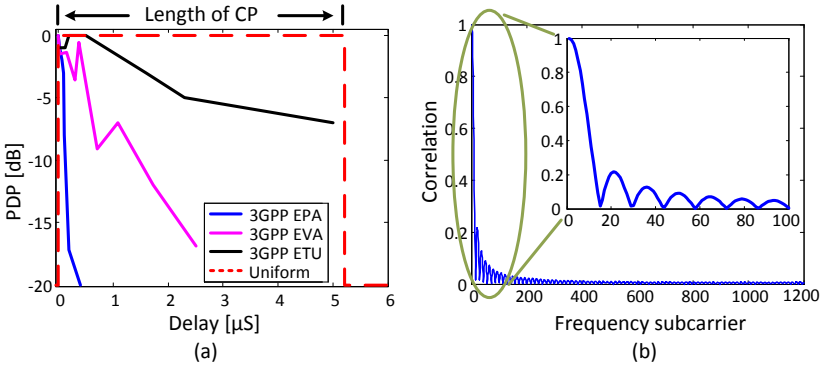


Figure 3: (a) The uniform PDP used in the R.MMSE estimator. PDPs of some typical channel scenarios in LTE-A [101] are included for comparisons. (b) An illustration of the frequency-domain correlations between subcarriers in 20MHz LTE-A, under the case of uniform PDP.

where $\boldsymbol{R}_{\boldsymbol{h}_d\boldsymbol{h}_p}$ is the channel cross-correlation between pilot and data-carrying subcarriers, $\boldsymbol{R}_{\boldsymbol{h}_p\boldsymbol{h}_p}$ represents the channel auto-correlation between pilot subcarriers, SNR denotes the average signal-to-noise ratio of received signals, and β is a constellation dependent constant, e.g., $\frac{180}{67}$ for 64-QAM.

Robust MMSE Estimator

A major drawback of the LMMSE estimator is its high computational complexity. One reason for this is the need for re-computation of $\boldsymbol{\mathcal{W}}$ every time SNR and/or the correlation matrices change. This is infeasible for practical implementations, especially when the number of subcarriers is large. Instead, the Robust MMSE (R.MMSE) algorithm [102] is adopted in this work. The R.MMSE estimator completely removes the need for run-time $\boldsymbol{\mathcal{W}}$ calculations by employing a static function, designed to safely tolerate various channel scenarios and rapid channel variations. The static $\boldsymbol{\mathcal{W}}$ is generated by using underestimated correlation matrices and an overestimated SNR value. Specifically, the correlation matrices are pre-computed based on two assumptions. First, the propagation channel obeys a uniform Power-Delay Profile (PDP). Second, the maximum excess delay of the channel is equal to the length of CP. This is illustrated in Figure 3(a). For comparisons, PDPs defined for some typical channel scenarios in LTE-A are included. As shown, they are all covered by the envelope of the uniform PDP used in R.MMSE, revealing the underestimated design strategy. Regarding SNR, a high value is preferable. This can intuitively be explained by considering a high-pass filtering process, where the

value of SNR acts as the cut-off frequency of the filter. Channel estimation errors are concealed in noise with low SNR (i.e., being attenuated in the stop-band region), but tend to be pronounced with high SNR. Hence, it is better to push SNR towards a high value region to keep the channel estimation error low for a large SNR range. Using these static parameters, $\boldsymbol{\mathcal{W}}$ becomes a constant scaling matrix that may be prepared off-line. To sum up, compared to LMMSE, the R.MMSE approach reduces the complexity at the cost of increased estimator error. Nevertheless, it performs better than the LS estimator, due to of the use of second-order channel statistics and the reduction of noise enhancements.

Modified Robust MMSE Estimator

Although (4) in R.MMSE is reduced to a constant matrix multiplication, it is still computationally intensive when considering the dimension of $\boldsymbol{h}_{p,\mathrm{LS}}$. For example,the vector has a size of 400×1 for 20 MHz LTE-A. To further reduce the complexity, a sliding window approach is applied to the R.MMSE algorithm, named as R.MMSE-SW for short. The key is to apply low-rank approximations [102] on $\boldsymbol{R}_{\boldsymbol{h}_d\boldsymbol{h}_p}$ and $\boldsymbol{R}_{\boldsymbol{h}_p\boldsymbol{h}_p}$ based on the fact that adjacent subcarriers generally have dominant contribution to correlation coefficients. Figure 3(b) illustrates the correlation between the first frequency subcarrier and all remaining ones under the case of uniform PDP. As an example, the first 100 subcarriers contribute to more than 95% of the total correlations. Therefore, a frequency correlation window (N_{SW}) containing only a number of neighboring pilots is applied to each estimation. As a consequence, the size of the matrix multiplication in (4) is dramatically reduced compared to the full-window case, i.e., when the correlation of all subcarriers are considered. In R.MMSE-SW, the size of N_{SW} is a performance-complexity trade-off parameter, which may be adjusted depending on the channel condition and performance demand.

2.2 Channel Pre-processing

Each estimated channel matrix $\hat{\boldsymbol{H}}$ needs to be further processed before being sent to the succeeding symbol detector. There are two commonly used channel pre-processing methods, inversion and QRD of the channel matrix, required in linear and tree-search based detectors, respectively. For matrices of size 4×4 or larger, it has been shown in [103] that matrix inversion can be efficiently computed by means of QRD. Hence, the QRD-based channel pre-processing method is adopted. In addition, sorting is applied to the channel matrix during the QRD process, aiming to improve the detection performance. This is commonly referred to as Sorted QRD (SQRD) [104].

SQRD starts by column-wise permuting $\hat{\boldsymbol{H}}$ based on the post-detection SNR [105] of each spatial stream. Thereafter, $\hat{\boldsymbol{H}}$ is decomposed into an uni-

tary matrix $\boldsymbol{Q}$ and an upper triangular matrix $\boldsymbol{R}$ with real-valued non-negative elements on the main diagonal. With these, the QRD-based channel pre-processing is expressed as

$$\hat{\boldsymbol{H}} = \hat{\boldsymbol{H}}_p \boldsymbol{P}^T = \boldsymbol{Q}\boldsymbol{R}\boldsymbol{P}^T, \tag{5}$$

where the $N \times N$-dimensional permutation matrix $\boldsymbol{P}$ contains the corresponding sorting sequence and the operator $(\cdot)^T$ denotes a matrix transpose.

MMSE-SQRD Algorithm

Considering the use of MMSE criterion during the adopted symbol detection, presented further in Section 2.3, MMSE-SQRD algorithm [106] is employed to pre-process $\hat{\boldsymbol{H}}$. The basic idea is to reduce the probability of ill-conditioned channel matrix by taking the additive noise into account. This is equivalent to performing SQRD of an augmented matrix of size $2N \times N$,

$$\underline{\hat{\boldsymbol{H}}} = \begin{bmatrix} \hat{\boldsymbol{H}} \\ \sigma_n \boldsymbol{I}_N \end{bmatrix} = \underline{\hat{\boldsymbol{H}}}_p \boldsymbol{P}^T = \underline{\boldsymbol{Q}}\,\underline{\boldsymbol{R}}\boldsymbol{P}^T = \begin{bmatrix} \boldsymbol{Q}_a \\ \boldsymbol{Q}_b \end{bmatrix} \underline{\boldsymbol{R}}\boldsymbol{P}^T, \tag{6}$$

where $\boldsymbol{Q}_a$, $\boldsymbol{Q}_b$, and $\underline{\boldsymbol{R}}$ have the same size as $\hat{\boldsymbol{H}}$, i.e., $N \times N$. One interesting property of (6) is that $\underline{\boldsymbol{R}}^{-1}$ is obtained as a by-product of the matrix decomposition process, i.e.,

$$\underline{\boldsymbol{R}}^{-1} = \frac{1}{\sigma_n} \boldsymbol{Q}_b. \tag{7}$$

Therefore, no explicit matrix inversion is needed when computing $\underline{\hat{\boldsymbol{H}}}^{\dagger}$, where $(\cdot)^{\dagger}$ denotes a matrix pseudo-inverse. With the augmented matrix $\underline{\hat{\boldsymbol{H}}}$, the system model in (1) can be rewritten as

$$\tilde{\boldsymbol{y}} = \underline{\boldsymbol{R}}\boldsymbol{x}_p + \tilde{\boldsymbol{n}}_p, \tag{8}$$

where

$$\tilde{\boldsymbol{y}} = \underline{\boldsymbol{Q}}^H \begin{bmatrix} \boldsymbol{y} \\ \boldsymbol{0}_{N\times 1} \end{bmatrix} = \boldsymbol{Q}_a^H \boldsymbol{y}, \tag{9}$$

$\boldsymbol{x}_p = \boldsymbol{P}^T \boldsymbol{x}$ is the row-wise permuted $\boldsymbol{x}$, and $\tilde{\boldsymbol{n}}_p = \boldsymbol{Q}_a^H \boldsymbol{n}$ is the noise vector that has the same statistics as $\boldsymbol{n}$.

Compared to (5), the decomposition of $\underline{\hat{\boldsymbol{H}}}$ results in an increased computational complexity, by roughly 50%, because of the doubled matrix dimension. However, MMSE-SQRD improves the performance of linear detectors and achieves a significant complexity reduction in tree-search based detection algorithms [107].

Iterative Sorting and MGS-QRD Algorithm

The sorting of $\underline{\hat{\boldsymbol{H}}}$ involves a matrix inversion, required for evaluating the post-detection SNR (η_i) of each spatial stream. Under the assumption of $\mathrm{E}\{\boldsymbol{x}\boldsymbol{x}^H\} = \boldsymbol{I}_N$, η_i is defined as in [105]

$$\eta_i = \frac{1}{\sigma_n^2 \left(\underline{\hat{\boldsymbol{H}}}^H \underline{\hat{\boldsymbol{H}}}\right)^{-1}_{i,i}}. \tag{10}$$

For practical implementations, approximation of (10) is commonly used for reducing the computational complexity, such as the one suggested in [104]

$$\eta_i \sim \left\| \underline{\hat{\boldsymbol{h}}}_i \right\|_2^2, \tag{11}$$

with $\|\cdot\|_2$ denoting ℓ^2-norm. Based on (11), various sorting strategies exist. Commonly used ones are one-time and iterative sorting [104]. Different from the former one, the iterative sorting approach keeps track of column changes in $\underline{\hat{\boldsymbol{H}}}$ during the decomposition process, which leads to a better sorting result. The iterative sorting steps are shown in Algorithm 1 on lines 3, 5, and 12.

For QRD computations, several well-known methods exist, such as Gram-Schmidt orthogonalization, Householder transformation, Givens rotation, and their derivatives [108]. The Gram-Schmidt process obtains the orthogonal basis spanning the column space of the matrix by the orthogonality principle [109]. The Householder transformation handles column vectors of the matrix by reflection operations [108]. Givens rotation operates on one element at a time by using a sequence of unitary transformations [107]. Considering the accuracy and numerical stability, computational complexity, and hardware reusability, Modified Gram-Schmidt (MGS) [108] method is used for implementing the QRD. The MGS-QRD algorithm iteratively computes the $\underline{\boldsymbol{Q}}$ and the $\underline{\boldsymbol{R}}$ matrix in N steps. Core operations of MGS-QRD per iteration i are summarized in Algorithm 1 from line 7 to 12.

2.3 Symbol Detection

With the received signal $\boldsymbol{y}$ and pre-processed channel matrix $\underline{\hat{\boldsymbol{H}}}$, transmitted vector $\boldsymbol{x}$ is recovered by using a MIMO symbol detector. As described in Chapter 3.2, there are two commonly used detection schemes. Linear detection algorithms, such as Zero-Forcing (ZF) and MMSE, mainly consist of vector operations and thus are architecture-friendly to vector-based platforms. However, they suffer from significant performance degradation compared to the optimal Maximum-Likelihood (ML) detection, especially in frequency-selective fading channels. On the other hand, near-ML tree-search algorithms, e.g., Sphere

Algorithm 1 MGS-based MMSE-SQRD algorithm

1: $\underline{\hat{\boldsymbol{H}}} = [\hat{\boldsymbol{H}}, \sigma_n \boldsymbol{I}_N]^T$
2: $\underline{\boldsymbol{Q}} = \underline{\hat{\boldsymbol{H}}}$; $\underline{\boldsymbol{R}} = \boldsymbol{0}_{N\times N}$; $\boldsymbol{P} = \boldsymbol{I}_N$
3: $\boldsymbol{\xi} = \left[\|\underline{\boldsymbol{q}}_1\|_2^2, \|\underline{\boldsymbol{q}}_2\|_2^2, \ldots, \|\underline{\boldsymbol{q}}_N\|_2^2\right]^T$
4: **for** $i = 1, 2, \ldots, N$ **do**
5: $j = \arg\min_{l=i,i+1,\ldots,N} \xi_l$ *% Iterative sorting*
6: Exchange columns/elements i and j in $\underline{\boldsymbol{Q}}$, $\underline{\boldsymbol{R}}$, $\boldsymbol{P}$, and $\boldsymbol{\xi}$
7: $\underline{r}_{i,i} = \sqrt{\xi_i}$
8: $\underline{\boldsymbol{q}}_i = \underline{\boldsymbol{q}}_i / \underline{r}_{i,i}$
9: **for** $k = i+1, i+2, \ldots, N$ **do**
10: $\underline{r}_{i,k} = \underline{\boldsymbol{q}}_i^H \underline{\boldsymbol{q}}_k$
11: $\underline{\boldsymbol{q}}_k = \underline{\boldsymbol{q}}_k - \underline{r}_{i,k}\underline{\boldsymbol{q}}_i$
12: $\xi_k = \|\underline{\boldsymbol{q}}_k\|_2^2$ *% Column-norm updating*
13: **end for**
14: **end for**

Decoder (SD), K-Best, and their derivatives [50, 51], do not map efficiently to vector-based architectures. This comes from the fact that the tree-search procedure deals with one spatial layer at a time and involves massive sequential scalar operations, which are frequently switched between node expansion, partial Euclidean distance sorting, and branch pruning. Therefore, the native vector structure of MIMO data streams is destroyed. To tackle this problem, detection algorithms, such as Fixed-complexity Sphere Decoder (FSD) and Selective Spanning with Fast Enumeration (SSFE) [53], are used to bring in vectorized operations that may be performed independently at each layer. However, they do not solve the essential problem of tree-structured detection schemes. Data dependency between spatial layers still restrains the full potential of parallel architectures.

To bridge the algorithm-architecture gap, illustrated in Figure 4, a highly-parallelized symbol detection algorithm is developed. It provides near-ML performance, like tree-search algorithms, while retaining the inherent vectorized operations of linear detection schemes. This is achieved by employing a vector-level closest point search scheme in conjunction with linear detectors. In this work, the proposed algorithm is built upon a linear MMSE detector and is named as Node-Perturbation-enhanced MMSE (MMSE-NP). As a proof-of-concept, this work focuses on the hard-output symbol detection.

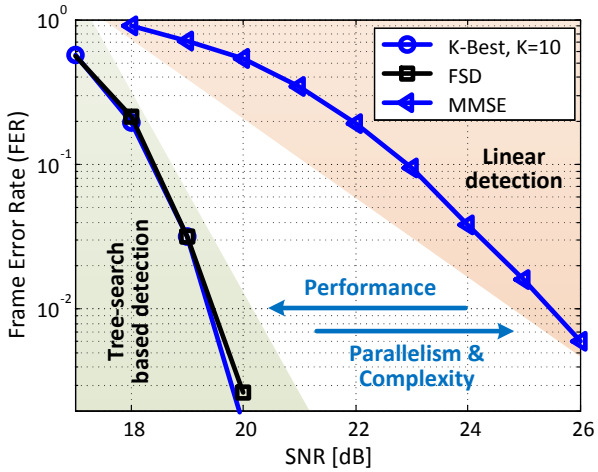


Figure 4: An illustration of the algorithm-architecture gap between linear and tree-search based detection schemes.

Parallel Node Perturbation

MMSE-NP starts by obtaining an initial result using a linear MMSE detection

$$\boldsymbol{x}_p^{\mathrm{MMSE}} = \underline{\boldsymbol{R}}^{-1}\tilde{\boldsymbol{y}} = \frac{1}{\sigma_n}\boldsymbol{Q}_b\boldsymbol{Q}_a^H\boldsymbol{y}. \tag{12}$$

Hard-output detection result $\hat{\boldsymbol{x}}_p^{\mathrm{MMSE}}$ is generated by slicing $\boldsymbol{x}_p^{\mathrm{MMSE}}$ to the nearest constellation point, i.e., $\hat{\boldsymbol{x}}_p^{\mathrm{MMSE}} = \mathcal{Q}\left(\boldsymbol{x}_p^{\mathrm{MMSE}}\right)$. Thereafter, a detection search space is defined by expanding each scalar MMSE symbol with a number of neighbors. Specifically, for the i^{th} symbol of the N-length MMSE vector $(\hat{x}_{p(i)}^{\mathrm{MMSE}})$, a set of $(\Omega_i - 1)$ locally nearest sibling symbols is found:

$$\boldsymbol{x}_{p(i)}^{\mathrm{NB}} = [x_{p(i)}^1, \cdots, x_{p(i)}^{\omega}, \cdots, x_{p(i)}^{(\Omega_i-1)}], \tag{13}$$

with their distances to $\hat{x}_{p(i)}^{\mathrm{MMSE}}$ sorted in ascending order, as

$$|x_{p(i)}^1 - \hat{x}_{p(i)}^{\mathrm{MMSE}}|^2 \leq \cdots \leq |x_{p(i)}^{\omega} - \hat{x}_{p(i)}^{\mathrm{MMSE}}|^2 \leq \cdots \tag{14}$$

Figure 5(a) and (b) illustrate the initial detection and search space delimitation for a case of 2×2 MIMO and 16-QAM modulation.

Once the search space is delimited, detection search paths are defined by generating a list $\mathcal{S}$ of candidate vectors using symbols drawn from the search

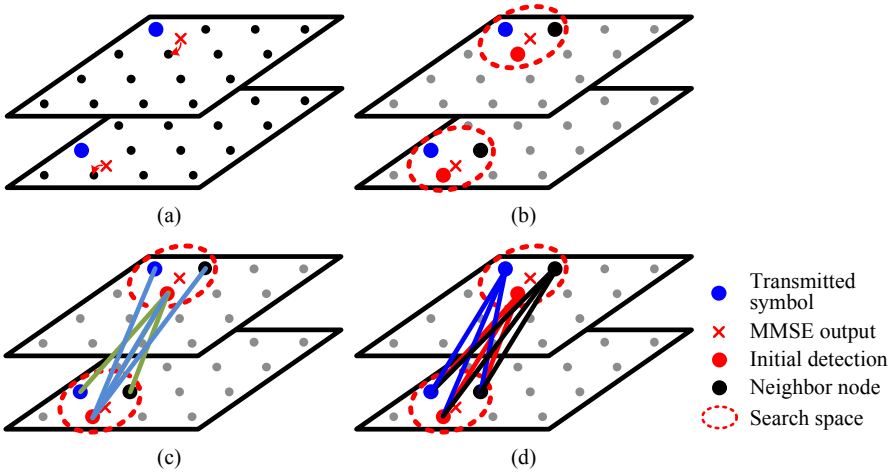


Figure 5: An illustration of the MMSE-NP detection for a 2×2 MIMO setup and 16-QAM modulation. (a) Initial detection by slicing MMSE symbols to nearest constellation points. (b) Parallel node extension to include nearest neighbors into a search space. (c) Single-error candidate vector generation. (d) Full-error candidate vector generation.

space. Two methods exist for Candidate Vector Generations (CVG). First, occurrence of only one error is assumed during the initial detection. Accordingly, candidate vectors are generated by replacing only one symbol in $\hat{\boldsymbol{x}}_p^{\mathrm{MMSE}}$ at a time, while keeping other ones unchanged, i.e., for the expanded $\boldsymbol{x}_{p(i)}^{\mathrm{NB}}$, $(\Omega_i - 1)$ candidate vectors are generated as

$$\begin{aligned} \boldsymbol{s}_i^1 &= [\hat{x}_{p(1)}^{\mathrm{MMSE}}, \cdots, x_{p(i)}^1, \cdots, \hat{x}_{p(N)}^{\mathrm{MMSE}}], \\ \boldsymbol{s}_i^2 &= [\hat{x}_{p(1)}^{\mathrm{MMSE}}, \cdots, x_{p(i)}^2, \cdots, \hat{x}_{p(N)}^{\mathrm{MMSE}}], \\ &\vdots \\ \boldsymbol{s}_i^{(\Omega_i-1)} &= [\hat{x}_{p(1)}^{\mathrm{MMSE}}, \cdots, x_{p(i)}^{(\Omega_i-1)}, \cdots, \hat{x}_{p(N)}^{\mathrm{MMSE}}]. \end{aligned} \tag{15}$$

After (15) being applied to all $\boldsymbol{x}_{p(i)}^{\mathrm{NB}}$ $(i \in [1, N])$, $L = \sum_{i=1}^{N} \Omega_i$ candidate vectors are obtained in the list $\mathcal{S}$ including the initial MMSE result $\hat{\boldsymbol{x}}_p^{\mathrm{MMSE}}$. In low-dimensional MIMO systems, such as 2×2, single-error dominates error events in the MMSE detection. However, for 4×4 or larger MIMO configurations, considering only one error in the initial detection is far from sufficient to cover most of the error events due to the increased degree of spatial selectivity. Hence, the

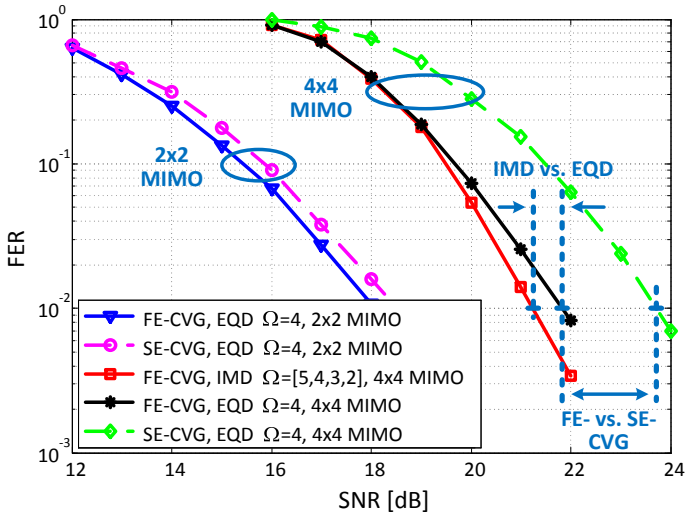


Figure 6: Performance comparisons of different candidate vector generation (CVG) and symbol expansion schemes.

second method considers a full-error scenario, i.e., assuming all symbols in $\boldsymbol{x}_p$ are erroneously detected. In consequence, all combinations of expended symbols $\boldsymbol{x}_{p(i)}^{\mathrm{NB}}$ in $\mathcal{S}$ have to be included, resulting in totally $L = \prod_{i=1}^{N} \Omega_i$ candidate vectors to be searched. Figure 5(c) and (d) illustrate the Single-Error (SE-CVG) and Full-Error (FE-CVG) candidate vector generation schemes. Performance of symbol detections, in terms of Frame-Error-Rate (FER), using these two methods are compared in both 2×2 and 4×4 MIMO systems with 64-QAM modulation, see Figure 6. As expected, performance of the SE-CVG scheme approaches to its full-error counterpart in the 2×2 system, whereas a large performance degradation is observed in the 4×4 case. In comparison, the FE-CVG scheme substantially improves the detection performance, i.e., by ∼2 dB, as shown in Figure 6. Hence, this work adopts the FE-CVG scheme.

The final detection result is generated by searching within $\mathcal{S}$ and finding the vector with the smallest squared Euclidean Distance (ED), i.e.,

$$\hat{\boldsymbol{x}}_p = \arg\min_{\boldsymbol{x}_p \in \mathcal{S}} \|\tilde{\boldsymbol{y}} - \underline{\boldsymbol{R}}\boldsymbol{x}_p\|_2^2 . \tag{16}$$

The recovered transmitted vector $\hat{\boldsymbol{x}}$ with its original symbol sequence is obtained by reordering the rows of $\hat{\boldsymbol{x}}_p$ with the permutation matrix $\boldsymbol{P}$, i.e.,

$$\hat{\boldsymbol{x}} = \boldsymbol{P}\hat{\boldsymbol{x}}_p. \tag{17}$$

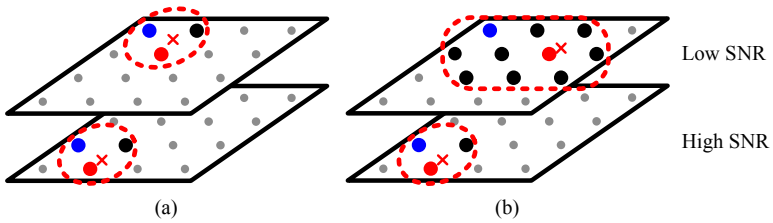


Figure 7: Symbol expansion schemes, (a) equally distributed (EQD) and (b) imbalanced distributed (IMD).

Compared to conventional tree-search based algorithms, MMSE-NP eliminates scalar and data dependent operations, as symbol expansions, candidate generations, and evaluations are carried out in parallel on all spatial layers. As a result, it provides extensive DLP for efficient implementations on vector-based architectures.

Imbalanced Node Perturbation

The perturbation parameter Ω_i in (13) needs to be adjusted to achieve a good performance-complexity trade-off. Basically there are two strategies to determine Ω_i. The first approach, referred to as *EQually Distributed* (EQD) expansion, is to consider the same number of neighbors around each scalar symbol $\hat{x}_{p(i)}^{\text{MMSE}}$, i.e., $\Omega_i = \Omega$. However, EQD expansion may not be cost-effective from a complexity point of view, as channel properties of each antenna port is not utilized when determining the search space. Consequently, search paths in S may be over-selected, which increases computational complexity in (16) without any improvement in performance. Therefore, an *IMbalanced Distributed* (IMD) expansion scheme is proposed to treat symbol expansion in $\hat{x}_{p(i)}^{\text{MMSE}}$ differently and assign Ω_i depending on the channel condition. The idea is to include more neighbors for symbols located in spatial layers with lower post-detection SNRs (η in (10)), i.e., $\Omega_i > \Omega_j$ if $\eta_i < \eta_j$. These two expansion schemes are illustrated in Figure 7 using the previous 2×2 MIMO setup.

Recall that $\underline{\hat{\boldsymbol{H}}}$ is column-wise permutated during MMSE-SQRD with their corresponding η_i sorted in ascending order, i.e., $\boldsymbol{\eta} = [\eta_{min}, \cdots, \eta_{max}]$, the assignments of Ω_i is simplified by arranging the vector $\boldsymbol{\Omega}$ in descending order, namely $\boldsymbol{\Omega} = [\Omega_{max}, \cdots, \Omega_{min}]$.

Thanks to the use of channel properties, the IMD expansion scheme provides better performance than EQD while using fewer number of candidate vectors. As demonstrated in Figure 6, detection using IMD with an expansion vector $\boldsymbol{\Omega} = [5, 4, 3, 2]$ is 0.6 dB better than the case of EQD with $\Omega = 4$ at FER $= 10^{-2}$,

even though the latter one uses 2 times more candidate vectors. Hence, the IMD expansion scheme is employed in this work. The exact $\mathbf{\Omega}$ assignment is a design parameter, which should be fine tuned at run-time.

Successive Partial Node Expansion (SPE)

The average error rate in a MIMO system is generally dominated by the spatial stream that suffers from the worst channel condition. Hence, node expansion for symbol with the smallest η value, i.e., $\hat{x}_{p(1)}^{\mathrm{MMSE}}$, needs to be handled with special care. According to the IMD expansion scheme, $\hat{x}_{p(1)}^{\mathrm{MMSE}}$ needs to be expanded with more neighbors to mitigate the high error probability. This strategy results in larger search space, which considerably increases the total number of candidate vectors, incurring huge computational complexity for the minimum-search process in (16).

To tackle this issue, a Successive Partial node Expansion (SPE) scheme is developed. It reduces the search space for $\hat{x}_{p(1)}^{\mathrm{MMSE}}$ without sacrificing detection performance. The basic idea is to utilize the property of the upper triangular matrix $\underline{\boldsymbol{R}}$ and the fact that the symbol with η_{min} has been moved to the first layer after MMSE-SQRD. With $\underline{r}_{j,1}$ $(j = [2, \cdots, N])$ being zeros, the detection of $x_{p(1)}$ is solely dependent on $\tilde{y}_1$. Thereby, an optimal expansion of $x_{p(1)}$ can be obtained by simply solving a linear equation, given that other symbols have been expanded prior to $x_{p(1)}$. More specifically, SPE starts by expanding "stronger" symbols (i.e., $[\hat{x}_{p(N)}^{\mathrm{MMSE}}, \cdots, \hat{x}_{p(2)}^{\mathrm{MMSE}}]$) and then generates partial candidate vectors $\boldsymbol{x}_p^{[1]}$ of size $(N-1)$. Here, $\boldsymbol{x}_p^{[1]} = [x_{p(2)}, \cdots, x_{p(N)}]^T$ denotes the sub-vector of $\boldsymbol{x}_p$ with the 1^{st} symbol $x_{p(1)}$ being omitted. Thereafter, $x_{p(1)}$ is obtained by substituting $\boldsymbol{x}_p^{[1]}$ into the 1^{st} row of the system model (8):

$$\begin{aligned} \tilde{y}_1 &= \sum_{j=1}^{N} \underline{r}_{1,j} x_{p(j)} = \underline{r}_{1,1} x_{p(1)} + \sum_{j=2}^{N} \underline{r}_{1,j} x_{p(j)} = \underline{r}_{1,1} x_{p(1)} + \underline{\boldsymbol{r}}_1^{[1]} \boldsymbol{x}_p^{[1]} \\ x_{p(1)} &= \mathcal{Q}\left(\left(\tilde{y}_1 - \underline{\boldsymbol{r}}_1^{[1]} \boldsymbol{x}_p^{[1]} \right) \Big/ \underline{r}_{1,1} \right), \end{aligned} \tag{18}$$

where $\underline{\boldsymbol{r}}_1^{[1]} = [\underline{r}_{1,2}, \cdots, \underline{r}_{1,N}]$. For all L possible $\boldsymbol{x}_p^{[1]}$ candidates, L number of $x_{p(1)}$ are found. This way, the search space for $\hat{x}_{p(1)}^{\mathrm{MMSE}}$ is reduced to only include symbols that, in conjunction with $\boldsymbol{x}_p^{[1]}$, generate the most likely search paths, i.e., symbols that result in the smallest ED for the given $\boldsymbol{x}_p^{[1]}$ vectors. Thus, the SPE scheme dramatically reduces the number of candidate vectors and thus the complexity of (16), while providing an equivalent performance as if all possible $x_{p(1)}$ symbols were included in the candidate vectors.

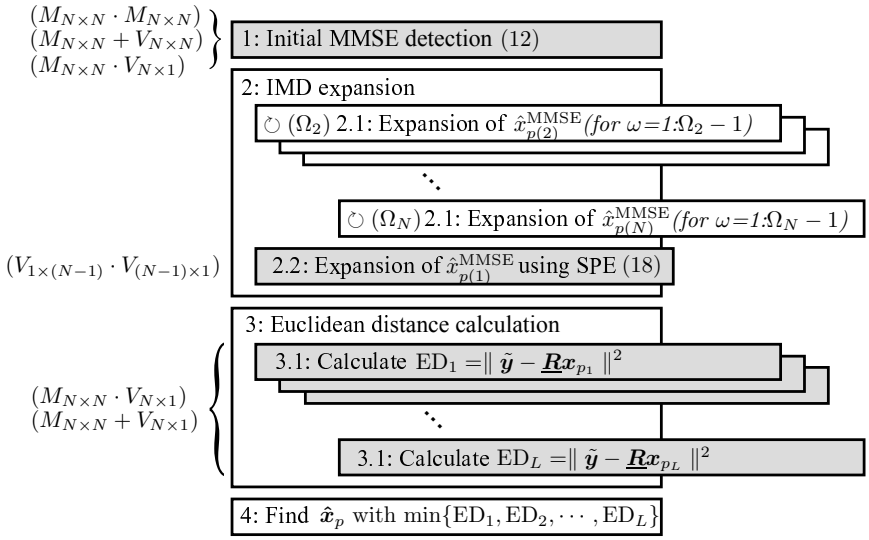


Figure 8: Computation procedure of the MMSE-NP algorithm.

Due to the successive expansion of symbol $\hat{x}^{\mathrm{MMSE}}_{p(1)}$, the SPE scheme partially breaks the structure of the N-length vector $\hat{\boldsymbol{x}}^{\mathrm{MMSE}}_{p}$. However, this adverse effect is substantially outweighed by the reduction of costly ED calculations and the efficiency of the optimal $\hat{x}^{\mathrm{MMSE}}_{p(1)}$ expansion.

Summary and Discussion

Figure 8 summarizes the computation procedure of the proposed MMSE-NP algorithm. It contains four main processing stages: initial linear MMSE detection, IMD symbol expansion, ED calculation, and final detection. In Figure 8, shaded boxes depict vector operations and layered boxes indicate parallel processing. $\circlearrowleft(l)$ represents loops with count l, while V and M denotes vector and matrix operations respectively. As shown, most of the boxes are shaded in the figure, indicating a highly vectorized algorithm. Besides, the overall dataflow is regular, even though one loop structure is found in an inner block, i.e., the symbol expansions of $[\hat{x}^{\mathrm{MMSE}}_{p(2)},\cdots,\hat{x}^{\mathrm{MMSE}}_{p(N)}]$.

3 Algorithm Evaluation and Operation Analysis

This section evaluates the adopted algorithms in terms of processing performance and computational complexity. As a metric for measuring performance, FER is used to show the effectiveness of the algorithms. Regarding computational complexity, the number of arithmetic operations is analyzed for data processing within one LTE-A time slot. Hardware friendliness is evaluated by analyzing the DLP and operation sharing of the three algorithms. Besides the operation analysis, task planning is conducted according to the timing specification of LTE-A.

3.1 Simulation Environment

Based on the structure of the MIMO-OFDM transceiver (Figure 1), a simplified system setup is implemented in MATLAB with a special focus on the LTE-A system. Figure 9 shows the block diagram of the employed simulation environment. In the current setup, data are transmitted through a baseband spatially-uncorrelated MIMO channel, where the maximum excess delay is smaller than the length of CP. Accordingly, domain (time-frequency) transformations and CP insertion/removing blocks are omitted. In addition, no pre-coding is implemented at the transmitter and MIMO systems are assumed to operate in a spatial-multiplexing mode. Moreover, the front-end block is omitted under assumptions of perfect front-end processing at the receiver, e.g., perfect synchronization and IQ-imbalance compensation.

Despite the system-level simplifications, the simulation setup is flexible as each block in Figure 9 can be configured with various parameters. Based on the error correcting code specified in [110], a parallel concatenated turbo code [47] is adopted at the transmitter. Input parameters for this block are the coding rate and the interleaver block size. The generator polynomials are $g_0(D) = 1+D^2+D^3$ and $g_1(D) = 1+D+D^3$. The modulation (mapping) block supports constellation sizes from BPSK to 64-QAM. Before inserting data and pilot tones into the LTE-A time-frequency grid, the layer mapping block maps encoded and modulated source data onto multiple antennas. Currently supported antenna sizes are 2×2 and 4×4. The size of the time-frequency grid is determined by the allocated bandwidth. All bandwidth configurations specified in LTE-A are included, varying from 1.4 to 20 MHz. The propagation channel is modelled as a frequency-selective fading channel, in which its multi-path delay profile complies with the ones defined in the 3GPP specification [101]. Three channel models are supported, Extended Pedestrian A (EPA), Extended Vehicular A (EVA), and Extended Typical Urban (ETU). Besides, the maximum Doppler frequency is used for channel generations. Moreover, both time-invariant (i.e., quasi-static) and variant channel modelling are implemented.

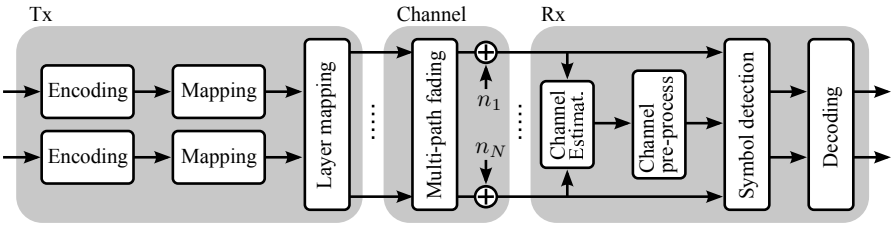


Figure 9: Block diagram of the employed simulation environment.

Table 1: Parameters for performance simulations in a LTE-A downlink.

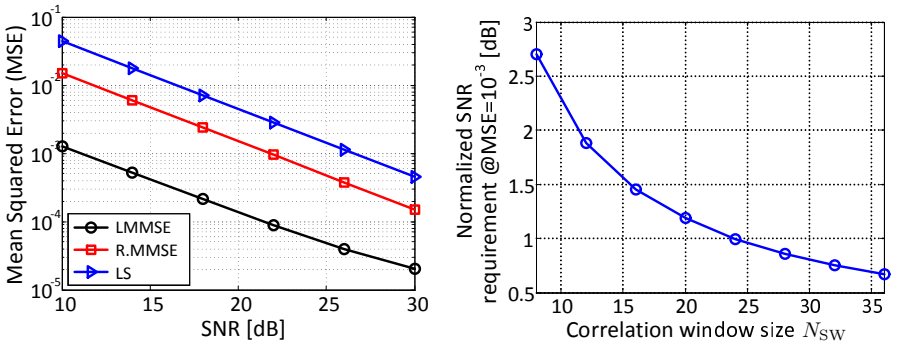
Block	Parameter	Value
Encode	Coding rate	1/2
	Interleaver block size	5376
Mapping	Constellation size	64-QAM
Channel	Antenna size	4×4
	Bandwidth	20 MHz
	Multi-path fading propagation	3GPP EVA
	Maximum Doppler frequency	70 Hz
	Time-variant/invariant	Quasi-static
Decode	Iteration number	6

The former one assumes that channel coefficients remain unchanged within one LTE-A time slot, whereas the latter one emulates the scenario of constantly-changing channels. The decoder at the receiver adopts the Bahl-Cocke-Jelinek-Raviv (BCJR) [111] algorithm with a configurable iteration number.

Detailed parameters used in the following performance simulations are summarized in Table 1. For each simulation, N_s LTE-A subframes (14 OFDM symbols) are transmitted, where N_s is dynamically adjusted to take account of different FERs with respect to SNR values. With a target of FER $= 10^{-2}$ that is a commonly used design criterion, N_s varies from 500 to 6000.

3.2 Performance Evaluation

Using the presented simulation environment, performance of the adopted MIMO processing algorithms are evaluated. It starts with analyzing the frequency correlation window (N_{SW}) in the R.MMSE-SW estimator and the node perturbation parameter ($\mathbf{\Omega}$) in the MMSE-NP detector. Off-line decisions on exact N_{SW} and $\mathbf{\Omega}$ values are not required, as they can be fine-tuned at run-time thanks



(a) (b)

Figure 10: Channel estimation without symbol detection, (a) comparison with LMMSE and LS algorithms, (b) comparison of different correlation window size (N_{SW}) in R.MMSE-SW.

to the hardware flexibilities (Section 4). Therefore, the focus of the following analysis is to compare these algorithms to other alternatives. One example is to see whether the MMSE-NP approach bridges the algorithm-architecture gap between linear and tree-search based detectors. Additionally, the analysis also serves to make better design trade-offs, for example, by studying both positive- and side-effects of parameter variations.

Channel Estimation

To minimize performance impacts from other MIMO processing tasks, the R.MMSE algorithm is firstly compared with LMMSE and LS estimators without involving channel pre-processing and symbol detection. Figure 10(a) shows the performance comparison of the three estimation algorithms in terms of Mean Squared Error (MSE). It demonstrates that R.MMSE achieves a better estimation result than the LS approach, e.g., by 4.7 dB under this simulation setup, thanks to the reduction of noise enhancements and the use of second-order channel statistics. Compared to the LMMSE estimator, some performance degradation is observed, mainly due to the use of the underestimated function $\boldsymbol{\mathcal{W}}$. However, when considering its performance robustness and a huge complexity gain to the LMMSE method (i.e., more than two orders of magnitude as shown further in Section 3.4), the R.MMSE approach is more attractive to implement in practice, especially for resource- and energy-limited devices.

Using the same simulation setup, Figure 10(b) compares R.MMSE-SW and R.MMSE with respect to different N_{SW} values. Numbers at the vertical axis denotes the minimum SNR required to reach the level of MSE $= 10^{-3}$, nor-

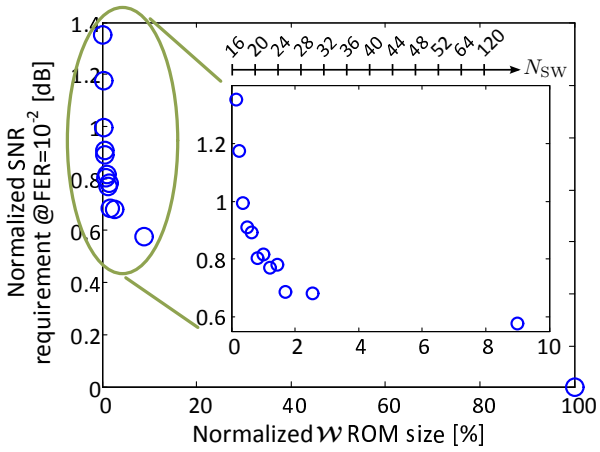


Figure 11: Performance versus coefficient ROM size for different N_{SW} values in the R.MMSE-SW estimator. Metrics are normalized to that of the full-window case - R.MMSE, which has 100% ROM size and zero required SNR at FER $= 10^{-2}$.

malized to the full-window case R.MMSE. Large values of N_{SW}, as expected, lead to small performance degradation in comparison to the R.MMSE case, but result in high computational complexity.

Based on these analysis, the adopted R.MMSE-SW estimator is further evaluated on a system-level, namely by including succeeding channel pre-processing and symbol detection and measuring the output FER. Meanwhile, required coefficient Read-Only Memory (ROM) sizes are calculated to illustrate (to some extend) hardware costs with respect to different N_{SW} values. A more detailed operation analysis is further presented in Section 3.4. To avoid any influence between R.MMSE-SW and the proposed MMSE-NP detector, the conventional near-ML FSD algorithm is used. Figure 11 shows the achieved FERs versus ROM sizes for various N_{SW} values. Both coordinates are normalized to the reference case R.MMSE. It clearly shows that small values of N_{SW} reduce ROM size substantially while retaining a good system performance. As an example, with $N_{\mathrm{SW}} = 24$, the required ROM size is reduced by 99.64% compared to that of the R.MMSE, at the cost of less than 1 dB FER degradation. These differences in FER diminish with increasing N_{SW} values. To conclude, N_{SW} should be fine-tuned at run-time to achieve on-demand performance-complexity trade-offs.

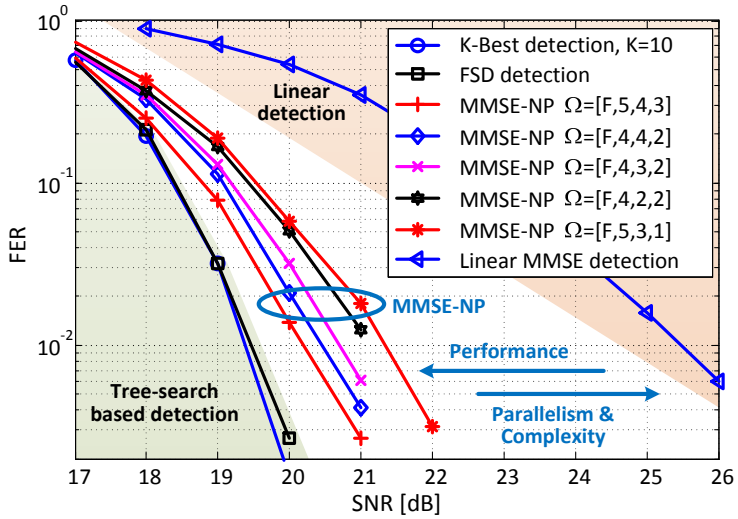


Figure 12: Comparison between linear, tree-search based, and proposed detection algorithms.

Symbol Detection

In this section, the node perturbation parameter ($\mathbf{\Omega}$) in the MMSE-NP detector is evaluated in comparison to conventional linear and tree-search based detection algorithms, e.g., the "linear MMSE" and "K-Best and FSD", respectively. To minimize performance impacts, these detectors are evaluated without performing other processing tasks. In other words, it is assumed that channel knowledge is perfectly estimated at the receiver and that channel matrices are properly processed by performing QRD for the K-Best and FSD cases and inversion (with the MMSE criterion) for the linear MMSE detector.

Figure 12 shows simulated FERs of these detection algorithms. For the MMSE-NP detector, the notation $\mathbf{\Omega} = [\mathrm{F}, \cdots]$ represents the employed SPE scheme (Section 2.3.3). Thanks to the developed techniques in MMSE-NP, i.e., node perturbation, IMQ, and SPE, the performance of the linear MMSE detector is enhanced substantially. More importantly, an FER performance close to that of the K-Best detector and FSD is achieved. For $\mathbf{\Omega} = [\mathrm{F}, 5, 4, 3]$, performance degradation to both K-Best decoder (with $K = 10$) and FSD is less than 1 dB at FER $= 10^{-2}$. Better performance is obtained by including more candidate vectors in the symbol expansion at the expense of implementation complexity. This is similar to the tree-search based detectors with different number of branch traversals, e.g., K-Best algorithm with different K values.

Table 2: Comparison of visited nodes and required SNR at FER = 10^{-2}.

	Parameter	N_{visited}		SNR [dB] @FER = 10^{-2}	
K-Best	$K = 10$	1984	– (ref.)	19.39	0
FSD	$P = 1$	256	7.75×	19.47	+0.08
MMSE-NP	$\mathbf{\Omega} = [\text{F}, 5, 4, 3]$	135	14.70×	20.20	+0.81
	$\mathbf{\Omega} = [\text{F}, 5, 3, 1]$	34	58.35×	21.34	+1.95
MMSE	N/A	N/A	N/A	25.48	+6.09

Figure 12 also compares the FER of different $\mathbf{\Omega}$ assignments. Comparing the cases $\mathbf{\Omega} = [\text{F}, 5, 4, 3]$ and $\mathbf{\Omega} = [\text{F}, 5, 3, 1]$, the former one is 1 dB better than the latter case, but with 4 times more candidate vectors involved in detection, thus demands more computational power.

Using the number of visited nodes as a first-order complexity analysis, Table 2 summaries the performance metrics for the four algorithms. Based on the node perturbation scheme, the node expansion number of the MMSE-NP detection is formulated as

$$N^{\text{MMSE-NP}} = \sum_{i=1}^{N} \Omega_i N_{i+1} = \sum_{i=1}^{N} \Omega_i \left(\prod_{j=i+1}^{N} \Omega_j \right), \tag{19}$$

where N_i is the number of nodes at the i^{th} spatial stream, and $N_1 = \Omega_1 = 1$ when using the SPE scheme. The total number of visited nodes in the K-Best algorithm [112] is calculated as

$$N^{\text{K-Best}} = M \sum_{i=1}^{N} N_F^{i+1}, \tag{20}$$

where $N_F^i = \min(K, MN_F^{i+1})$ denotes the number of parent nodes at the i^{th} layer with M being the constellation size. For the FSD [52], the number of visited nodes is

$$N^{\text{FSD}} = \sum_{i=1}^{N} \prod_{j=i}^{N} p_i. \tag{21}$$

It shows in Table 2 that the number of nodes visited in the MMSE-NP algorithm with $\mathbf{\Omega} = [\text{F}, 5, 4, 3]$ is 15 and 1.9 times fewer than that of the K-Best detector and FSD respectively, which demonstrates the cost effectiveness of the MMSE-NP. In summary, the proposed MMSE-NP algorithm bridges the algorithm-architecture gap between linear and tree-structured detection

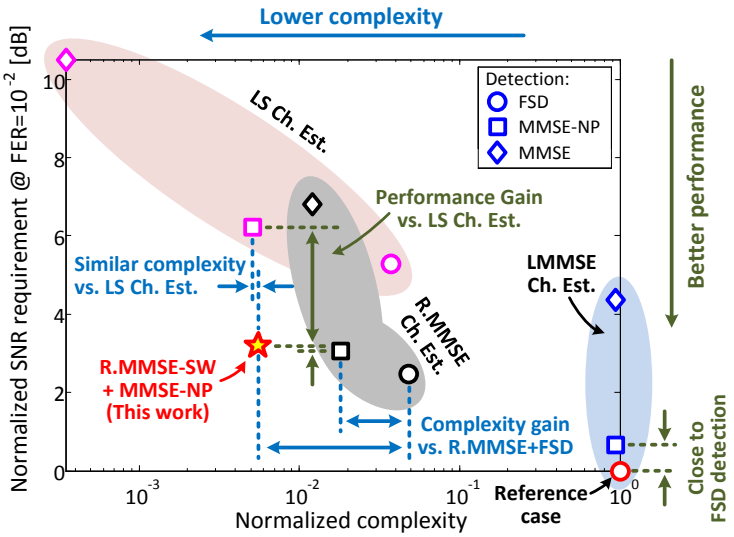


Figure 13: Analysis of processing performance and computational complexity. Metrics are normalized to that of the reference case "LMMSE+FSD" which has unit computational complexity and zero required SNR at FER $= 10^{-2}$.

schemes. In addition, with imbalanced $\mathbf{\Omega}$ assignments, the algorithm is highly scalable, since the symbol detection of each spatial stream can be tuned dynamically to adapt to instantaneous channel condition or currently available computational resources.

MIMO Signal Processing

After the analysis of individual algorithms, the MIMO processing tasks are evaluated together by using different combinations of algorithms. This is aimed to compare the proposed processing scheme, "R.MMSE-SW+MMSE-NP", with other approaches with respect to performance and computational complexity. For the following analysis, parameters of $N_{SW} = 24$ and $\mathbf{\Omega} = [\mathrm{F}, 4, 3, 2]$ are used for R.MMSE-SW and MMSE-NP, respectively. Figure 13 gives a full picture of the performance-complexity trade-offs for different algorithm sets. For better illustration, they are grouped into three clusters based on the involved channel estimation method. In Figure 13, numbers at the vertical axis denotes the minimum SNR required to achieve the target 10^{-2} FER, while the computational complexity measured in the number of operations required in one LTE-A time slot is shown along the horizontal axis. In addition, both coordi-

nates are normalized to a reference case, "LMMSE+FSD", which provides the best performance among these algorithms. According to this setup, algorithms whose coordinates are closed to the bottom-left corner are desired.

It shows in Figure 13 that the adopted scheme "R.MMSE-SW+MMSE-NP" achieves a good design trade-off between performance and complexity. For instance, it provides more than 7 dB performance gain to the "LS+MMSE" method (left-up corner) and achieves more than two orders of magnitude complexity reduction to the "LMMSE+FSD" case (right-bottom corner). It should be re-emphasized that N_{SW} and $\mathbf{\Omega}$ are tunable parameters and should be optimized at run-time. In the following section, DLP, operation sharing, and computational complexity of the three tasks are analyzed in detail.

3.3 Operation and Complexity Analysis

With the presented algorithms, primitive operations required by the R.MMSE-SW estimator, MMSE-SQRD pre-processor, and MMSE-NP detector are characterized. Table 3 summarizes required vector and scalar operations and their proportion in each task. Two meaningful properties can be observed. First, most of the operations are at vector level thanks to the development of algorithm vectorization. Specifically, vector operations occupy more than 98% of the total workload in all three tasks, indicating high DLP. This is an important design criterion for attaining efficient implementations with respect to processing throughput and energy consumption. Second, most of the primitive operations are shared among these algorithms, implying the potential of extensive hardware reuse. Moreover, when considering a vector dot product as an element-wise vector multiplication followed by a vector addition, all vector operations are common to all three tasks.

Based on the operation profiling, computational complexity of the algorithms is analyzed. To simplify the analysis, same precision is assumed for all calculations and a W-bit complex-valued addition is used as a baseline operation. This way, a W-bit complex-valued multiplication has the complexity of W; a W-bit real-valued division and square-root operation has a complexity of KW with K being a scaling factor, e.g., iteration number in Newton-Raphson method [113], and is set to 2 in this study.

Given these assumptions, complexity of "R.MMSE-SW+MMSE-NP" is compared with other algorithms in Figure 13. As for the three cases inside the LMMSE group, the complexity of the LMMSE algorithm is so dominating that it almost conceals any difference between different channel pre-processing and symbol detection algorithms. By comparison, the adopted R.MMSE-SW algorithm shows a similar level of complexity to that of the LS estimator, while providing much better processing performance. In terms of symbol detection,

Table 3: Algorithm profiling for primitive vector ($\boldsymbol{V}$) and scalar (s) operations in the adopted MIMO signal processing.

	Primitive operation	Operation dimension & Proportion in each task						Total proportion
		R.MMSE-SW		MMSE-SQRD		MMSE-NP		
Vector	$\boldsymbol{A} \bigodot \boldsymbol{B}^a$	–	–	$\boldsymbol{V}_{(N\times 1)}$	35%	–	–	4.30%
	$\boldsymbol{A} \cdot \boldsymbol{B}$	$\boldsymbol{V}_{(N_{\mathrm{sw}}\times 1)}$	91%	$\boldsymbol{V}_{(N\times 1)}$	35%	$\boldsymbol{V}_{(N\times 1)}$	84%	80.98%
	$\boldsymbol{A} \pm \boldsymbol{B}$	–	–	$\boldsymbol{V}_{(N\times 1)}$	15%	$\boldsymbol{V}_{(N\times 1)}$	15%	13.36%
Scalar	$x_a \cdot x_b$	$s(x_a \cdot x_b)$	9%	–	–	–	–	0.29%
	Sorting	–	–	$s(x_i)$	5%	$s(x_i)$	∼0%	0.07%
	$1/\sqrt{x}$	–	–	$s(x)$	10%	–	–	0.14%
	Pert.[b]	–	–	–	–	$s(\Omega_i)$	1%	0.86%

[a] Element-wise vector multiplication.
[b] Node perturbation in symbol detection.

the proposed MMSE-NP algorithm demonstrates 2.7 times complexity reduction compared to FSD. The combination of R.MMSE-SW and MMSE-NP is 8.6 times less complex than the "R.MMSE+FSD" case, at the price of less than 1 dB performance degradation. In summary, the employed processing scheme provides a good performance-complexity trade-off and is hardware friendly to vector-based architectures.

3.4 Processing Flow and Timing Analysis

In this part, dataflow and timing analysis is presented according to the LTE-A specification. Table 4 summaries resource allocations in the 20 MHz LTE-A. Within one time slot (0.5 mS), 7 OFDM symbols are transmitted, including 46.88% of data, 7.81% of pilots, 38.65% of guard-band subcarriers, and 6.67% of cyclic prefix. Figure 14(a) shows the structure of a 4×4 MIMO LTE-A data frame and the flow of target baseband processing. As illustrated, the LS computation in (2) can be initiated as soon as the pilot data has been received, followed by the frequency domain interpolation (4). However, channel pre-processing and subsequent symbol detection cannot start until the second OFDM symbol is received due to pilot receptions for antenna port 2 and 3 (Figure 2). As a consequence of this sequential processing flow, one can see from Figure 14(a) that processing gaps widely exist in between neighboring OFDM symbols as well as consecutive time slots. Thus, implementations using task-dedicated hardware will result in poor resource utilization. Moreover, the cyclic prefix and guard band interval between adjacent OFDM symbols further enlarge those processing gaps.

Table 4: Resource allocations in one LTE-A time slot.

Time slot (t_{slot})	0.5 mS		
Bandwidth	20 MHz		
Sampling frequency	30.72 MHz		
Number of subcarriers/symbol	2048		
Total number of subcarriers	14336	466.67 μS	93.33%
Total length of CP	1024	33.33 μS	6.67%
Data-carrying subcarriers	7200	234.38 μS	46.88%
Pilot tones	1200	39.06 μS	7.81%
Guard band subcarriers	5936	193.23 μS	38.65%

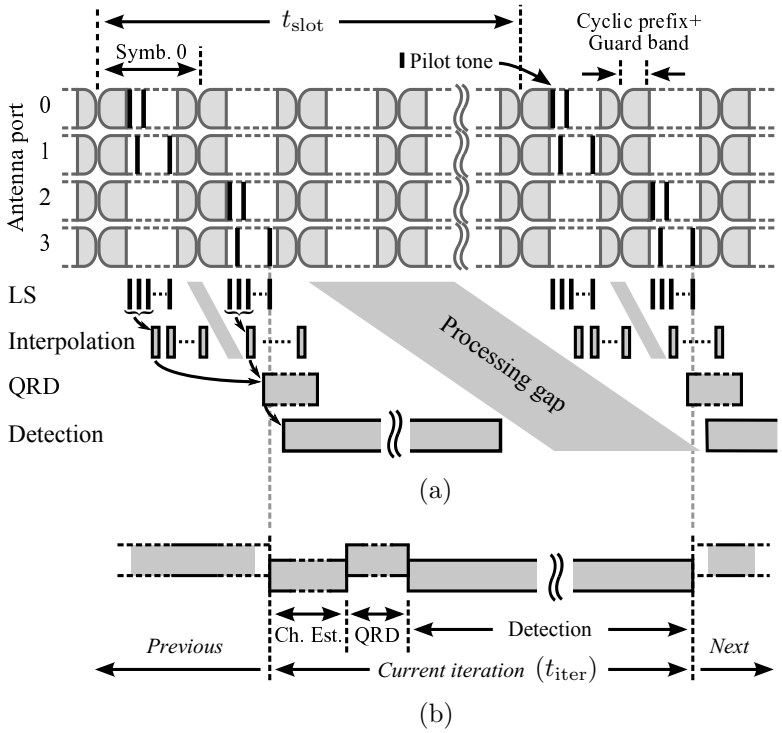


Figure 14: Timing diagram of the MIMO signal processing, (a) an LTE-A frame structure and data dependency between processing tasks, (b) adopted task-oriented processing flow.

To attain efficient hardware usage, this work maps the tasks on one reconfigurable platform by utilizing the sequential nature of processing and non-data-carrying time intervals. Figure 14(b) illustrates a task-oriented processing flow, which performs one task on all subcarriers before switching to the subsequent one. This is different from a subcarrier-oriented scheme (i.e., handling one subcarrier at a time), which requires much more power to carry out frequent context switching. According to [114], dynamic configurations may take up to 40% of the overall power consumption in reconfigurable platforms. Thereby, reducing the number of context switching is an efficient way to achieve power efficient implementation. Note that every processing iteration shown in Figure 14 starts immediately the last pilot tone in OFDM symbol 1 is received. This is arranged to prevent processing gaps due to data awaiting by making sure that all required pilot tones have been buffered.

Baseband processing in LTE-A systems requires data buffering of several OFDM symbols [6] to, for example, cope with the orthogonal pilot patten and processing latency of control channels [115]. Thus, additional buffers are not required in the proposed solution if all the three processing tasks can be handled within the specified time interval. In this work, processing is scheduled on a basis of one LTE-A time slot, see Figure 14. Therefore, the computation time of each iteration (t_{iter}) is constrained by t_{slot}, such that $t_{\text{iter}} \leq t_{\text{slot}} = 0.5\,\text{mS}$. This is used as a design constraint for hardware development.

4 Hardware Development

Using the reconfigurable cell array developed in Part I and Part II as baseline architectures, this section presents a number of hardware enhancements for attaining efficient implementation of MIMO signal processing. The focus here is on vector computing using heterogeneous RCs and various memory access schemes. In addition, a technique for further improving processing throughput and hardware efficiency is elaborated.

Before presenting the architectural development, three main properties of MIMO signal processing are extracted from the aforementioned operation analysis. Correspondingly, hardware requirements are identified with respect to *computation*, *memory access*, and *data transfer*.

- **Massive vector operations**: in view of the massive vector operations, i.e., more than 98% of the total workload in Table 3, efficient vector computing and high bandwidth memory access are essential. Besides, it is beneficial to reduce the number of register/memory accesses and data transfers to keep processing overhead low, since the control (regarded as a part of control overhead) required for performing those operations may consume a large portion of time and power [58].

- **Hybrid data-widths and formats**: the coexistence of scalar and vector operations requires a hybrid computational data-path. Additionally, efficient communication mechanisms are expected to offload processing units from non-computational operations, e.g., data alignments, during data transfers of various data-widths and formats.

- **Multi-subcarrier processing**: as a scheduling technique for further exploiting DLP (Chapter 3.3), multi-subcarrier processing requires various data access patterns to perform operations simultaneously at multiple subcarriers. Therefore, flexible memory access schemes are required, e.g., concurrent accesses of vectors from different channel matrices.

These requirements pose design challenges for hardware development and need to be addressed during the architectural design to ensure implementation efficiency.

4.1 Architecture Overview

Built upon the baseline architecture (Figure 15(a)), the proposed baseband processor is composed of four heterogeneous tiles, which are partitioned into scalar- and vector-processing domains to cope with hybrid data computing, see Figure 15(b). In the vector domain, Tile-0 handles vector processing while Tile-1 provides data storages and various forms of vector and matrix accesses. In the scalar domain, Tile-3 controls other RCs during run-time and handles scalar and irregular operations with memory supports from Tile-2. Data transfers between the two domains are bridged by memory cells using the micro-block function (Part 3.2). This feature efficiently supports hybrid data transfers without additional controls from processing cells. For example, memory cells in Tile-1 provide vector data accesses to RCs in Tile-0 while exchanging data in a scalar form with RCs in Tile-2.

Besides the heterogeneous resource deployments, communication to an external host for both data transfers and off-line configurations are carried out using the hierarchical network. Run-time configurations for all RCs are issued on a per-clock-cycle basis, performed hierarchically within the cell array, and managed jointly by a task manager (i.e., a processing cell in Tile-3) and local controllers distributed in RCs. Specifically, the task manager tracks the overall processing flow, controls context switching (e.g., changing from channel estimation to pre-processing), and handles configuration updating (e.g., parameter updates for N_{sw} and $\mathbf{\Omega}$). Local controllers are responsible for applying configurations onto processing data- and memory access-paths to, for example, switch between operations listed in Table 3.

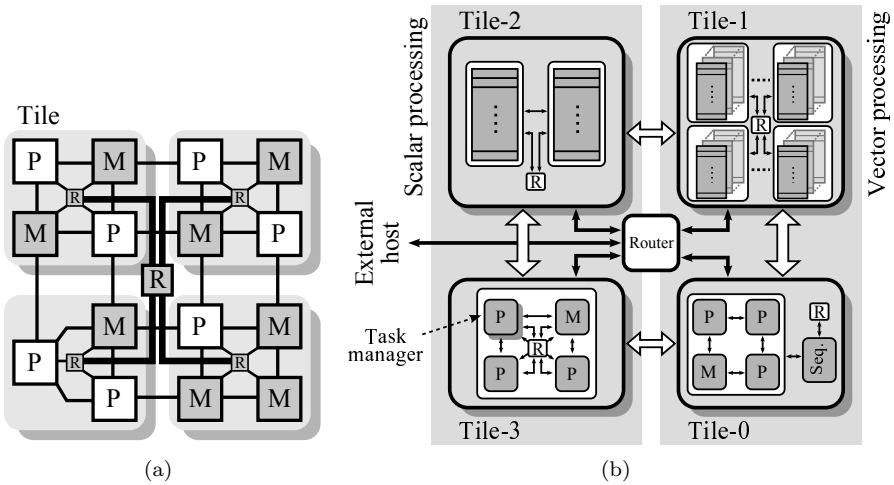


Figure 15: (a) Baseline architecture of the reconfigurable cell array, an example of four tiles. (b) Block diagram of the proposed heterogeneous baseband processor. Distributed controllers within RCs are omitted in the figure for simplicity.

4.2 **Vector Dataflow Processor**

Figure 16 shows the architecture of Tile-0, a vector dataflow processor, consisting of three processing cells (pre-, core-, and post-processing), one memory cell (register bank), and a sequencer. The three processing cells, shown on the upper half of Figure 16, are deployed for vector computations. The register bank provides data accesses from both internal registers and other tiles through register-mapped IO ports. The sequencer controls operations of the other cells via a control bus, drawn in dashed lines in Figure 16.

Atomic operations of Tile-0 are built upon N-length vectors which is the most common data type of the vector processing in Table 3. Vectors exceeding this length are processed by folding, i.e., they are decomposed into data segments suitable for atomic operations. Local data transfers within Tile-0 are carried out on two $N \times N$ matrix and one $N \times 1$ vector bus, arranged both to suffice computational requirements and to improve processing efficiency. The two matrix buses are used to support data intensive operations such as data-tone MMSE interpolation in (4) and Euclidean norm of augmented channel matrix $\underline{\hat{\boldsymbol{H}}}$ in (Algorithm 1 line 3 and 12), both requiring two $N \times N$ matrix inputs. The vector bus, on the other hand, is used to accelerate three-input operations such as column vector update in (Algorithm 1 line 11), which oth-

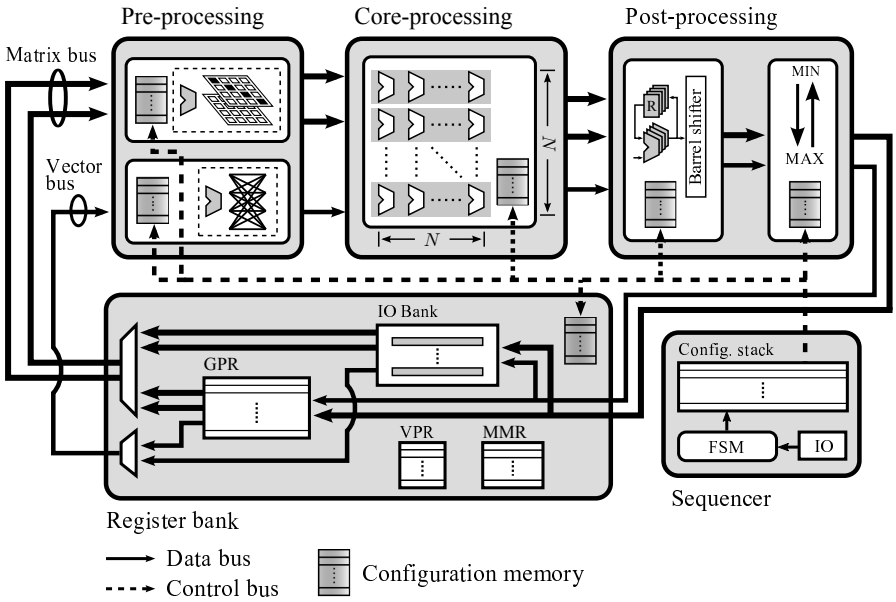


Figure 16: Microarchitecture of the vector dataflow processor (Tile-0). A VLIW-style multi-stage computation chain consists of three processing cells: pre-, SIMD vector core-, and post-processing.

erwise require twice of the clock cycles with additional operations for loading and storing intermediate results.

In the following sections, the adopted configurable Instruction Set Architecture (ISA) and two vector processing enhancements are presented.

Configurable Instruction Set Architecture

Conventionally, processors are implemented based on fixed ISAs, e.g., the generic and dataflow processors presented in Part I and II. Depending on target applications, the ISAs contain different specifications of, for example, instructions and addressing modes, and cannot be changed once they have been implemented. As a consequence, tasks outside the set originally intended may not benefit from the available computing capability, since underlying data- and control-path are hidden inside the fixed ISAs. Therefore, this design strategy often results in either limited flexibility (the case of application-specific) or poor performance (the case of general-purpose). In addition, it may require a deep study of target applications, which may not always be possible, concerning time-to-market, adoption of new algorithms, etc.

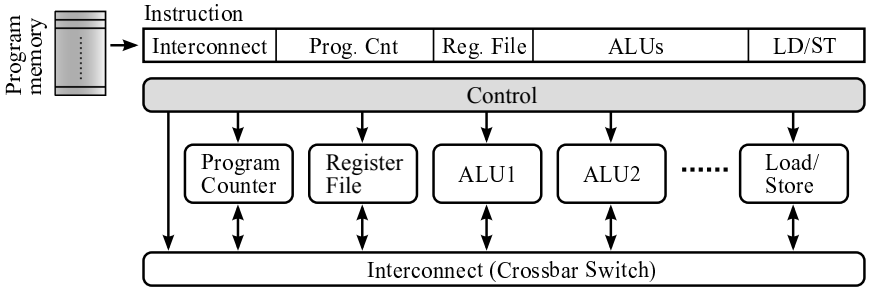


Figure 17: Illustration of a fine-grained centralized control scheme in a configurable ISA, an example of FlexCore [66].

Some processors adopt configurable ISAs, which can be customized for different use. Configurations can be performed either during the chip synthesis [32] or at run-time using a similar approach to FPGAs [66, 67]. In view of the run-time configurability, the latter approach is desired. However, it is commonly implemented using a centralized control scheme, which incurs high control overhead with regard to configuration time, hardware complexity, and storage requirements. As an example, Figure 17 shows the configurable ISA of a FlexCore [66]. The processor exposes its entire data- and control-path to the user via a long instruction word, 91 bits in this example. A controller controls dataflow and operations of each hardware unit based on instructions fetched from a program memory. It is inefficient that configurations of all RCs are centralized in one instruction, since any change among those configurations requires loading of a whole new instruction, resulting in unnecessary program storage and memory access for unchanged parts. Many code size reduction schemes exist which have reported a maximum compression ratio of about 70% on a Very Long Instruction Word (VLIW) processor [116]. However, this reduction comes at an area cost of up to 30% for run-time instruction decompression.

To tackle the aforementioned overhead issue, two control techniques are proposed in the adopted run-time configurable ISA.

Distributed Micro-code Execution Figure 18 illustrates a distributed control scheme employed in this work to reduce the overhead of the long instruction word. The idea is to divide an instruction into a number of smaller ones, termed as micro-codes, each getting dispatched to an RC. For storing the micro-codes, each RC is deployed with a configuration memory, which can be accessed individually without affecting others. The size of these memories can be kept small, since the number of operations required from each RC in an application is

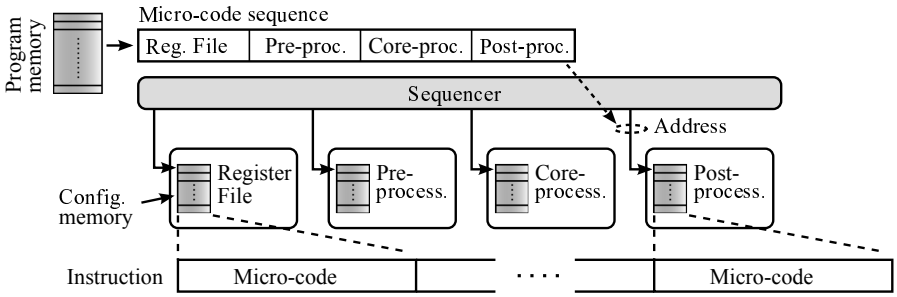


Figure 18: Illustration of the employed distributed micro-code execution scheme in the vector dataflow processor.

often limited. In addition, new micro-codes can be prepared and loaded to the memories while current instructions are being executed. This further reduces the storage requirement of the configuration memories. Using this distributed control scheme, fetching an instruction involves only address managements of the configuration memories. The required list of memory addresses is referred to as a micro-code sequence. Compared to the size of a micro-code, a memory address has much smaller wordlength, thus reducing the control overhead.

To demonstrate the gain of this scheme, a numerical example is given as follows. Considering a case where each configuration memory in Figure 18 is of size $32\,\text{bit}\times 16$. The corresponding wordlength required for fetching an instruction when using the conventional approach is $32\,\text{bits} \times 4 = 128$ bits. With the proposed scheme, a micro-code sequence requires only $4\,\text{bits} \times 4 = 16$ bits, reducing the wordlength of the program memory by 8 times. In the case of storing $D = 256$ instructions, the reduced wordlength leads to a further memory reduction of 5.3 times, since only $(16\,\text{bits} \times D + (32\,\text{bits} \times 16) \times 4)$ bits are required instead of $(128\,\text{bits} \times D)$ bits.

Using the distributed micro-code execution scheme, ISA configuration contains two steps. First, micro-codes of individual RCs need to be defined and loaded to the distributed configuration memories. Second, micro-code sequences need to be specified and stored in the program memory. The complete micro-code set for each processing cell in Tile-0 is presented in Appendix B. Worth mentioning is that no branching instructions (except loops which are treated differently, see the following section) are implemented in the vector dataflow processor, since the processor is intended to be used for data-centric stream processing that often has exposed data dependences and deterministic processing structure. Computation tasks mapped onto the processor are performed by invoking a series of kernel functions, such as matrix multiplication

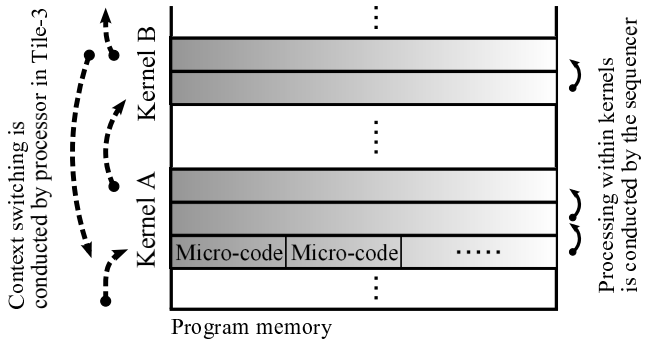


Figure 19: Assisted instruction branching in the vector dataflow processor.

and QR decomposition. Switching between these functions are conducted by the generic processing cell in Tile-3. This way, program branching and the execution of conditional operations are mimicked. The assisted instruction branching is illustrated in Figure 19.

Multi-level Zero-delay Inner Loop Control To further reduce control overhead during loop operations, a multi-level inner loop control scheme is adopted. Part 3.1 presents an inner loop controller designed to conduct loop operations with a zero execution latency. However, it supports only one loop level, far from sufficient for performing baseband processing in MIMO-OFDM systems. Multi-level loops are widely used to process multiple subcarriers per OFDM symbol, multiple spatial layers per subcarrier, and multiple iterations per spatial layer. Hence, the zero-delay one-level loop control scheme is extended to efficiently process loop operations without limit on the loop hierarchy.

To achieve the multi-level loop control, a stack-based architecture is employed, illustrated in Figure 20(a). As shown, the loop controller contains a configuration stack used to store the address of the first instruction in a loop (link address) and the corresponding loop count. Compared to other memory structures, the stack has a simple control mechanism. It natively supports the execution order of multi-level loops. With the help of a Finite-State Machine (FSM), link addresses of loops are pushed into the stack in a "last-in-first-out" manner during the execution of a program. Figure 20(b) shows a snapshot of the stack when the entire loop hierarchy of the enclosed code fragment is pushed into the stack. Each instruction contains a flag used to indicate end-of-loop, similar to the one used in the one-level loop controller (Part 3.1). Upon the completion of a loop iteration, the controller updates the program counter with the link address drawn from the top of the stack in order to jump back to

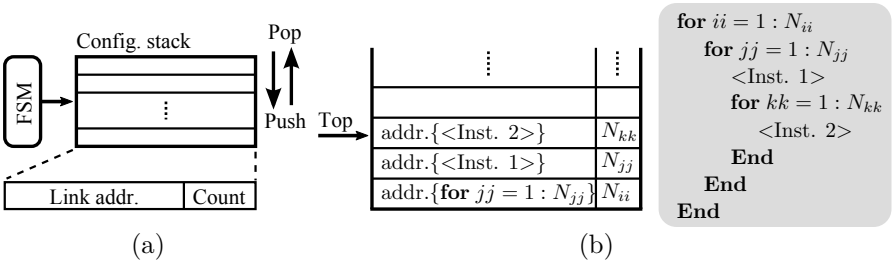


Figure 20: (a) Multi-level inner loop control. (b) A snapshot of the configuration stack and an example of a code fragment.

the start of the loop. Meanwhile, the stored loop count decreases by 1. When the counter value reaches 0, a loop operation is completed and automatically popped out, the stack pointer decreases by 1, and the link address stored at the new stack top is fetched. When the bottom of the stack is reached, loop controller releases the control of the program counter and the subsequent instruction stored in the program memory is fetched for execution. As can be seen, this enhanced loop control scheme requires no loop management operations. Therefore, it eases program writing, speeds up loop processing, and reduces program size and control overhead.

Based on the configurable ISA, the following sections focus on data-path of Tile-0 and present architectural improvements for attaining efficient processing.

Vector-enhanced SIMD Core

In wireless baseband processing, Single Instruction Multiple Data (SIMD) is commonly used as a baseline architecture to exploit inherent DLP. Similarly, an SIMD-based architecture is adopted in the core-processing cell, containing $N \times N$ homogeneous Complex-valued Multiply-ACcumulate (CMAC) units (Figure 16). The two-dimensional CMAC bank is deployed to handle parallel MIMO data streams and perform all vector operations in Table 3. Figure 21 shows a detailed architecture of the CMAC unit, containing four data inputs, three levels of arithmetic elements, and an input operand arrangement unit. With data inputs $\{a, b\}$ and $\{c\}$ coming from the matrix and vector data bus respectively, arithmetic elements in the first two levels are used to conduct complex-valued multiplication and addition. Two adders in level-2 sum up level-1 outputs with different data operands, such as a, c and e, depending on data-path configurations.

Concerning the execution latency of vector operations, conventional SIMD architectures (e.g., [58] and [32]) are inefficient, since they are designed to han-

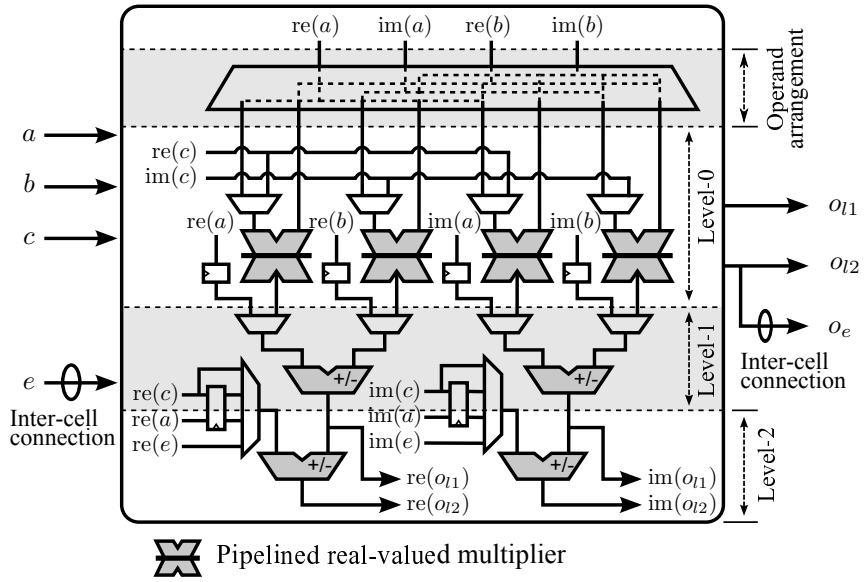


Figure 21: Architectural diagram of a CMAC unit.

dle parallel independent scalar data operands and their function units between processing lanes cannot operate collaboratively during instruction execution. For example, the computation of Vector Dot Product (VDP), which takes more than 80% of entire vector processing in Table 3, requires multiple clock cycles (depending on vector length), since each efficiently mapped VDP operation is performed in a folded fashion using at most one CMAC unit. This not only increases execution latency but also causes a large number of suspended computational resources. Although concurrent operations on multiple data sets may alleviate the latency issue to some extent, they requires additional data buffers for storing intermediate results and a more sophisticated sequence control. Figure 22(a) shows a typical scalar-based SIMD architecture, where data transfers between processing lanes are only possible through internal registers.

In contrast, this work tackles the latency issue by adopting an effective low-complexity vectorization technique in the conventional SIMD architecture. This vector enhancement enables single-clock-cycle execution for all vector operations of length N. Specifically, each processing lane is expanded to have N CMAC units, each of which is equipped with an inter-cell connection (e-path in Figure 21) to link up with neighboring CMACs during instruction execution. For example, the e input in Figure 21 is connected to the level-2 output (O_e)

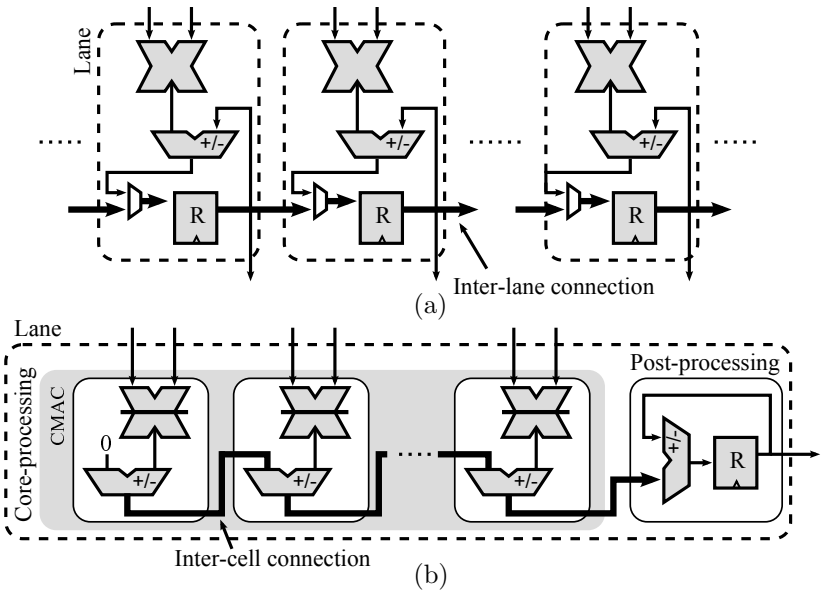


Figure 22: (a) Conventional scalar-based SIMD architecture [117]. (b) Illustration of a processing lane in the vector-enhanced SIMD core.

of the previous CMAC unit. Using this simple connection, level-2 adders of CMACs in every processing lane can be concatenated to form an adder-tree capable of computing one N-length vector in every clock cycle, e.g., a VDP with an atomic operation of '$ab + e$'. Figure 22(b) illustrates the construction of the adder-tree using the vector-enhanced processing lane. For practical implementations, a balanced tree structure (not shown in Figure 22(b)) is used for reducing the critical path of the SIMD core. Vectors exceeding the length N are processed by folding. In other words, they are decomposed into data segments suitable for atomic operations. The net results of this vector enhancement are significantly reduced execution latency and simplified sequence control.

Besides the efficient VDP computing, numerous vector operations are supported by the SIMD core, e.g., vector addition/subtraction '$\boldsymbol{a} \pm \boldsymbol{b}$' (16) and multiply-add '$\boldsymbol{a} \pm \boldsymbol{bc}$' (Algorithm 1 line 11). This is achieved by utilizing the flexible structure of CMAC units, in which each level of the arithmetic elements can be used individually or operated with different combinations of data operands. To further extend the operation set, an operand arrangement unit is deployed at input of each CMAC unit, see Figure 23. It contains three main function blocks, capable of swapping, negating, and shuffling the real and

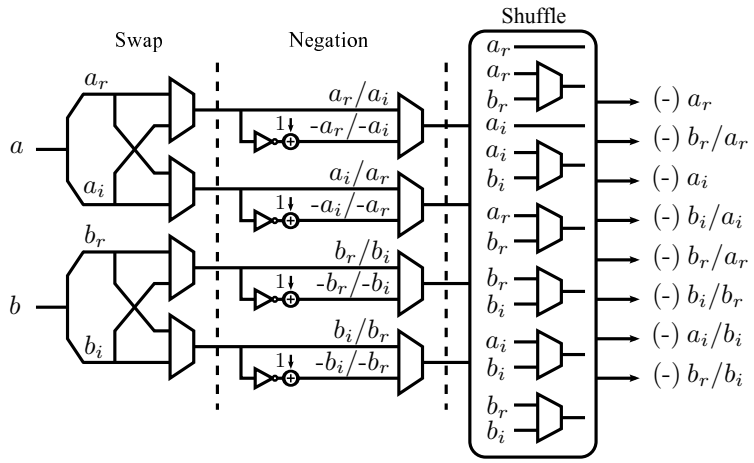


Figure 23: Block diagram of the input operand arrangement unit.

imaginary part of input operands $\boldsymbol{a}$ and $\boldsymbol{b}$, respectively. Giving this flexibility, various data sequences are provided to the following CMAC unit for performing complex- and real-valued operations. Using a_r, a_i, b_r, and b_i to denote the real and imaginary part of operands $\boldsymbol{a}$ and $\boldsymbol{b}$ respectively, Table 5 lists some of the data sequences required by the commonly used operations.

VLIW-style Multi-stage Computing

Another important observation from the algorithm analysis (Section 3) is that most of the vector processing involve several tightly coupled operations, such as complex conjugate (Algorithm 1 line 10) and result sorting (16) performed, respectively, before and after vector computations. Mapping of such "long" processing solely on the SIMD core requires multiple atomic operations, causing not only increased execution time but also redundant register file accesses for intermediate result buffering. Moreover, execution of some operations, such as complex conjugate, only uses a small part of the function units, resulting in poor resource utilization. Hence, the SIMD core is extended by adopting a VLIW-style multi-stage computation chain to accomplish several consecutive data manipulations in one single instruction. Specifically, two distinct processing cells are arranged around the core-processor to pre- and post-process data respectively, see Figure 16. Benefiting from this arrangement, more than 60% of register accesses are avoided, as the combination of pre- and post-processing takes about two-thirds of the total vector computations. As an example, Table 6 summarizes operations required for implementing the MMSE-SQRD al-

Table 5: Some commonly used operations and the corresponding data sequences generated by the input operand arrangement unit.

Operation	Expression	Data sequence
Complex-MUL	$(a_r + \mathrm{j}a_i)(b_r + \mathrm{j}b_i)$	$a_r, b_r, a_i, b_i, a_r, b_i, a_i, b_r$
Complex-MUL with j	$((a_r + \mathrm{j}a_i)\mathrm{j})\,((b_r + \mathrm{j}b_i)\mathrm{j})$	$-a_i, -b_i, a_r, b_r, -a_i, b_r, a_r, -b_i$
Complex-MUL with -j	$((a_r + \mathrm{j}a_i)\text{-j})\,((b_r + \mathrm{j}b_i)\text{-j})$	$a_i, b_i, -a_r, -b_r, a_i, -b_r, -a_r, b_i$
Complex-Squared norm	$(a_r + \mathrm{j}a_i)(a_r + \mathrm{j}a_i)^H$ $(b_r + \mathrm{j}b_i)(b_r + \mathrm{j}b_i)^H$	$a_r, a_r, a_i, a_i, b_r, b_r, b_i, b_i$
Real-MUL	$a_r b_r, a_i b_i$	a_r, b_r, a_i, b_i
Real-Square	$a_r^2, a_i^2, b_r^2, b_i^2$	$a_r, a_r, a_i, a_i, b_r, b_r, b_i, b_i$

gorithm. A similar technique named operation chaining for reducing register accesses can be found in [58].

Implementation of the pre- and post-processing cells depends on the operation profile of target applications. In the case of MIMO signal processing, the pre-processing cell consists of two function units that, respectively, work with matrix and vector data. Data negation and absolute calculation are examples of the pre-processing operations, which can be applied individually to each part (real and imaginary) of complex-valued data operands. For matrix inputs, a matrix data mask function is adopted to ease the run-time generation of regular and frequently used data access patterns, e.g., construction of the identity matrix required by $\underline{\hat{\boldsymbol{H}}}$ (6). Matrix data masks are stored in Matrix Mask Registers (MMR), see Figure 16. Each mask contains a boolean data map, used to indicate the "existence" of the matrix element at the corresponding position. The masking operation is realized by logically ANDing the matrix input with the data mask, real and imaginary part separately, illustrated in Figure 24. Examples of some commonly used masks are identity-, diagonal-, and upper triangular-martix, and real/imaginary part addressing. For vector inputs, data operands can be permuted based on permutation indexes stored in Vector Permutation Registers (VPR). Both the MMR and VPR can be pre-loaded during resource configurations or dynamically loaded with values taken from the vector data bus. In addition to the permutation function, vector operands can be broadcast both horizontally and vertically to the SIMD core to support parallel computing, e.g., broadcasting $\underline{\boldsymbol{q}}_i$ in 'Algorithm 1 line 10' to all processing lanes to compute multiple $\underline{r}_{i,k}$ in parallel.

The post-processing cell works with level-1, level-2, and accumulated (*e*-path) results from the SIMD core. It consists of two function units. With employed barrel shifters, the first unit is mainly used to dynamically adjust

Table 6: An example of the multi-stage computing in MIMO channel pre-processing (MMSE-SQRD, Algorithm 1).

Pre-processing	Pre-1: Complex conjugate
	Pre-2: Vector shuffling & broadcast
	Pre-3: Matrix data masking
Post-processing	Post-1: Barrel shifting
	Post-2: Sorting

Operation	Pre-processing	Core-processing	Post-processing
$\xi_i = \|\underline{\boldsymbol{q}}_i\|_2^2$ & sort	Pre-3	VDP $(ab+e)$	Post-1, 2
$\underline{r}_{i,i} = \sqrt{\xi_i}$	–	VDP $(ab+e)$	Post-1
$\underline{\boldsymbol{q}}_i = \underline{\boldsymbol{q}}_i / \underline{r}_{i,i}$	Pre-2	bc	Post-1
$\underline{r}_{i,k} = \underline{\boldsymbol{q}}_i^H \underline{\boldsymbol{q}}_k$	Pre-1	VDP $(ab+e)$	Post-1
$\underline{\boldsymbol{q}}_k = \underline{\boldsymbol{q}}_k - \underline{r}_{i,k} \underline{\boldsymbol{q}}_i$	Pre-2	$a - bc$	Post-1
$\underline{\boldsymbol{R}}^{-1} = 1/\sigma_n \boldsymbol{Q}_b$	Pre-2	bc	Post-1

data precisions of core-processing outputs. This is a useful function in vector processing especially for iterative and cumulative operations. Additionally, e-path output of each processing lane can be accumulated individually, which is required in supporting over-dimensioned vector operations (e.g., (4) for $N_{\mathrm{SW}} > N$), where a folding technique performs data accumulations on partial data outputs. The second function unit provides capability of permuting vector outputs in ascending, descending, or user-defined order. For example, this feature can be used to perform the sorting operations in MMSE-SQRD (Section 2.2.2).

4.3 Vector Data Memory Tile

Besides vector enhancements and multi-stage computation, the efficiency of the vector processor is contingent on memory access with regard to accessing bandwidth and flexibility. By inspection of algorithms discussed in Section 2, it is required that the SIMD core has access to multiple matrices and/or vectors in each operation, so as to avoid poor resource utilization and low throughput. As an example, efficient mapping of (Algorithm 1 line 3) requires two $N \times N$ matrix inputs, equivalent to having a $2\times(4\times4)\times(16+16)$=1024bits/cycle memory bandwidth for a 16-bit 4×4 MIMO system. In addition to the bandwidth requirement, various forms of data accesses are needed, such as row- and column-wise addressing in matrix transposition. Moreover, to exploit additional DLP from independent data streams, accesses of vectors in different matrices are

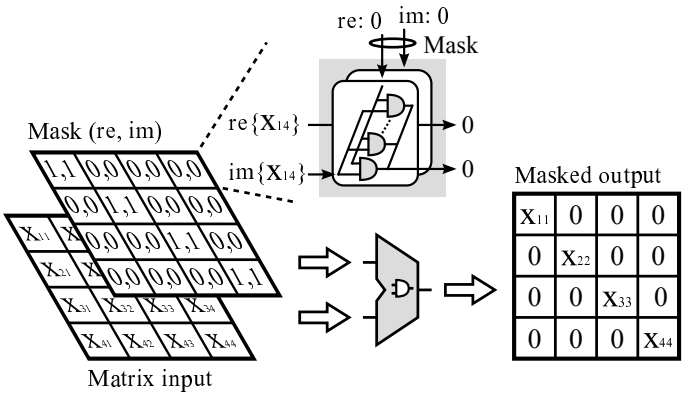


Figure 24: Illustration of a matrix masking operation, an example of the diagonal matrix construction.

required by the multi-subcarrier processing. To meet these requirements, a hybrid memory organization and a flexible matrix access mechanism are adopted in the vector data memory tile (Tile-1).

Hybrid Memory Organization

To suffice the high memory accessing bandwidth, Tile-1 consists of vector and matrix access partitions, allowing simultaneous accesses of both vectors and matrices, see Figure 25(a). The basic element in both partitions is a dual-port memory cell, which provides a vector-level data storage and allows simultaneous read and write operations to ease memory access and improve processing throughput at the price of a larger memory footprint. In addition, the matrix partition provides direct matrix data access, which is realized by concurrently accessing a group of memory cells using only one set of address control. This arrangement is referred to as a memory page, shown in Figure 25(a). The vector accessing wordlength and the number of cells in a memory page are designed to match the processing capacity of the SIMD core in Tile-0, i.e., N scalar elements and N memory cells, respectively. On the other hand, the number of memory cells and pages are application dependent and should be optimized with respect to the bandwidth requirement and hardware cost. To ensure a sufficient memory storage required for the MIMO signal processing, Tile-1 in this work is deployed with 2 memory cells, for buffering data and storing R.MMSE-SW coefficients $\boldsymbol{\mathcal{W}}$ in (4), and 5 pages, for storing $\underline{\hat{\boldsymbol{H}}}$ (Figure 26(a)) and $\underline{\boldsymbol{R}}$. Details of these memory usages are further discussed in Section 5.

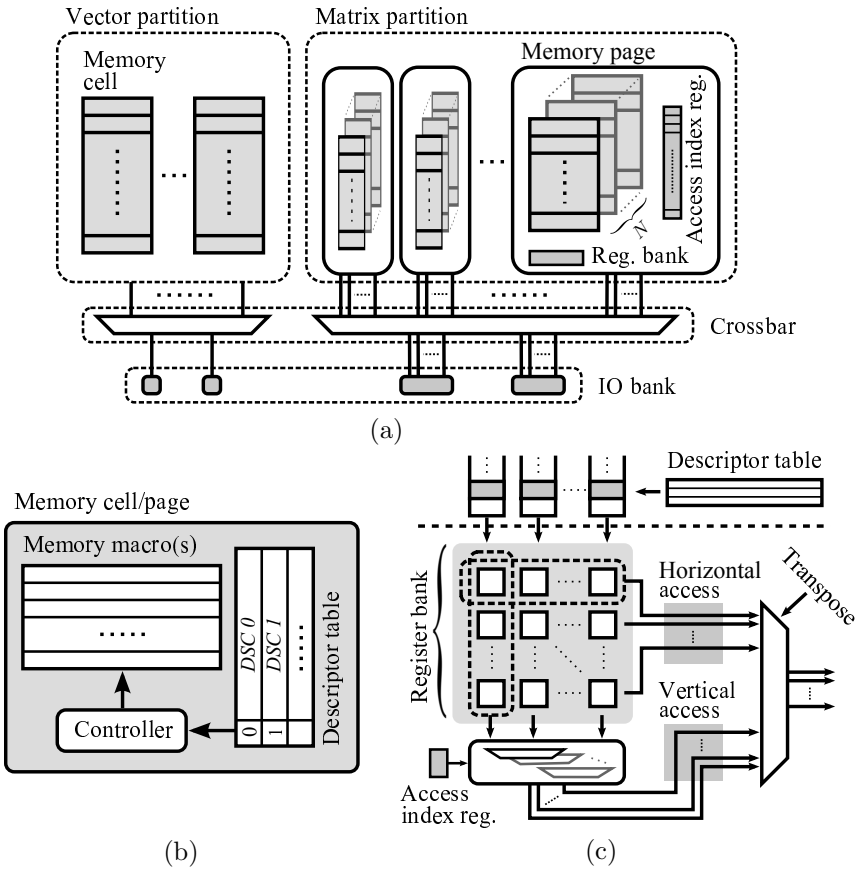


Figure 25: Block diagram of the vector data memory tile (Tile-1), (a) a hybrid memory organization, (b) operation and accessing control, (c) data loading path of a memory page, supporting matrix access indexing and transposition.

Memory operations and accessing modes of each cell and page are managed by a local controller with configurations stored in a descriptor (DSC) table, see Figure 25(b). To communicate with other tiles, memory accesses are multiplexed using a crossbar network and interfaced through IO ports. For the array shown in Figure 15(b), Tile-1 contains four IO ports, allowing simultaneous accesses of two $N \times 1$ vectors and two $N \times N$ matrices for providing accesses to both Tile-0 and Tile-2. Referring to the aforementioned example, this corresponds to a memory bandwidth of 1280 bits/cycle.

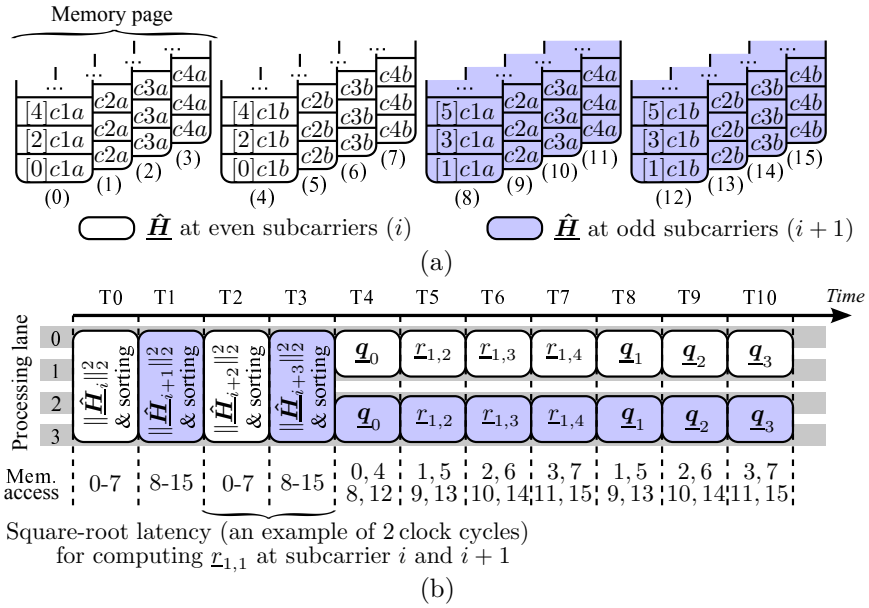


Figure 26: Example of some memory access patterns required for computing MGS-based MMSE-SQRD in a 4×4 MIMO system. (a) Odd and even indexed $\hat{\underline{\boldsymbol{H}}}$ are stored separately in different memory pages. (b) Timing diagram of the multi-subcarrier processing in the SIMD core with $N = 4$ processing lanes.

Flexible Matrix Data Access

The presented multi-page memory arrangement and the crossbar network allow for the flexible data access required by the multi-subcarrier processing. For instance, by storing matrices of successive subcarriers in different memory pages, multiple data sets can be concurrently accessed and multiplexed based on arrangement indexes specified in memory configurations.

To better explain the necessity of the flexible memory access in supporting multi-subcarrier processing, the following shows a case study of various memory access patterns required for computing MMSE-SQRD (Algorithm 1). Figure 26(a) shows a memory layout of $\hat{\underline{\boldsymbol{H}}}$ storage, where $\hat{\underline{\boldsymbol{H}}}$ at even- and odd-indexed subcarriers (labelled as [0], [1], etc.) are stored in different memory pages. Because of the $2N \times N$ dimension, every $\hat{\underline{\boldsymbol{H}}}$ is stored column-wise in two memory pages. In Figure 26(a), "[0]$c1a$" denotes column 1 of the upper half matrix $(N \times N)$ of $\hat{\underline{\boldsymbol{H}}}$ at subcarrier 0, and "[0]$c1b$" represents the lower half.

Figure 26(b) shows a timing diagram of multi-subcarrier processing in Tile-0 for the computation of the first iteration of MMSE-SQRD. Together with the operations performed in each time interval, required memory access patterns are listed. For example, 0-7 indicates concurrent access of memory cells with index from (0) to (7). During the time interval T0−T3, multiple matrix accesses are needed for computing the squared Euclidean norm of $\underline{\hat{\boldsymbol{H}}}$. Multiple vector readings from different memory pages are required during T4−T10. Supported by the flexible memory access schemes, multiple subcarriers can be processed in parallel to efficiently utilize processing gaps caused by data-dependent operations and computation latencies. Without this support, processing lanes during the time intervals of shaded computations in Figure 26(b) would be idle. Therefore, flexible memory access schemes are important for achieving high processing efficiency.

To further improve matrix access flexibility, a data arrangement circuit, illustrated in Figure 25(c), is implemented in each memory page. Specifically, data loaded from each memory page are buffered in a local register bank and are capable of being rearranged vector-wise in a vertical direction, based on an access index associated with each matrix storage. Benefiting from this setup, vector readouts from a matrix can be accessed freely in any order without physically exchanging data. This is useful, for example, in supporting sorted matrix accesses in MMSE-SQRD (Algorithm 1 line 5). The vector access indexes are stored in special registers that are transparent to the users and are configurable during every matrix data transfer. In addition to these index manipulations, the proposed architecture is capable of outputting matrices in a transposed form (used for example in (12)) by selecting either the row or column output. As a result, processing cells are relieved from such data arrangement operations, which otherwise results in enormous underused processing power. Moreover, physical data exchange and redundant memory accesses (due to read and write of the same data contents) are completely eliminated.

4.4 Scalar Resource Cells and Accelerators

In the scalar domain, Tile-2 and 3 perform scalar and conditional operations as well as dynamic configurations of other tiles in the array. Among them, Tile-2 consists of two scalar memories for storing data and configurations respectively. Tile-3 contains one memory for data buffering and three processing cells for computations. Figure 27 shows the three scalar processing cells in Tile-3, which are one generic signal processor and two acceleration units. The generic processor is a customized RISC with optimized conditional instructions and specialized functionality for dynamic RC configurations, similar to the one presented in Part I. The two accelerators behave like co-processors of

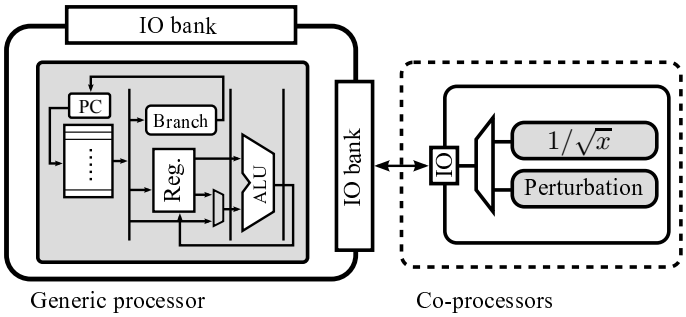


Figure 27: Block diagram of the scalar processing cells in Tile-3, containing one generic RISC-structured processor and two accelerators.

the generic cell for performing irregular operations, i.e., the inverse square-root in MMSE-SQRD and the node perturbation in MMSE-NP, respectively.

Inverse Square Root Unit

To compute the inversion square root of x, where $x \in \mathbb{R}$ and $x > 0$, Newton's method is adopted in this work. Newton's method iteratively computes approximations to the root of a real-valued function $f(y)$ [113]. In the case of inversion square root, $f(y)$ is defined as,

$$f(y) = \frac{1}{y^2} - x, \tag{22}$$

where $y = 1/\sqrt{x}$. A general expression of Newton's method for iteration i is written as

$$y_{i+1} = y_i - \frac{f(y_i)}{f'(y_i)}, \tag{23}$$

where the whole process starts off with some arbitrary initial value y_0. Substituting (22) into (23), the output of each iteration can be expressed as

$$y_{i+1} = 2^{-1} y_i \left(3 - x y_i^2\right). \tag{24}$$

After K iterations, the value of y_{i+1} converges to $1/\sqrt{x}$. The number of iterations required depends on the accuracy requirement of the application and how close the initial value y_0 is to $1/\sqrt{x}$. Fixed-point simulations show that $K = 2$ is sufficient in this work to obtain a near floating-point performance in terms of FER of the presented system setup (Section 3.1). It is worth mentioning that calculation of the square root can be obtained by multiplying the final result y_K by the input x.

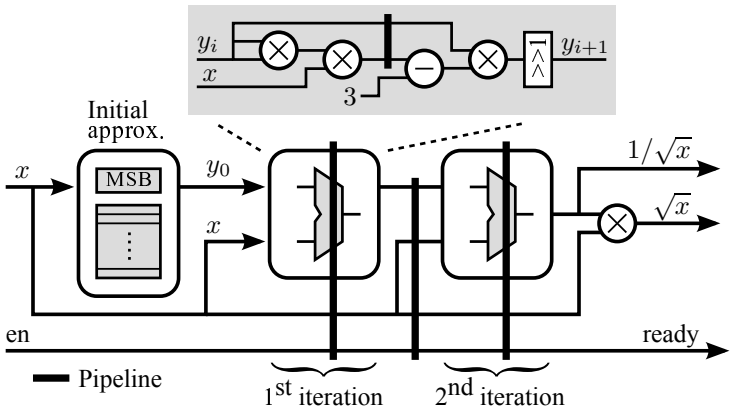


Figure 28: Block diagram of the inverse square root unit using Newton' method with 2 iterations.

Figure 28 shows the block diagram of the inverse square root unit, which is capable of computing both $1/\sqrt{x}$ and $\sqrt{x}$. It consists of three main building blocks, an initial value approximation block and two function units. Given an input data x, the first block generates an initial value y_0 by looking up in a coefficient table. To reduce the table size while providing a good initial value, only three Most Significant Bits (MSBs) of x are used as inputs. The position of the MSB is dynamically detected for each input x. The basic principle of the adopted method is to share the same coefficients stored in the table for different initial value approximations. For example, $y_0 = 1/4$ for $x = 16$, where the coefficient $1/4$ ("01000000" in a unsigned radix-2 '0.8' format) can be used to generate the initial value for $x = 256$ by shifting it 2 bits to the right, that is, $y_0 = 1/16$ with the radix-2 representation of "000**1**0000". Benefiting from the coefficient sharing, only 8 entries are required in the look-up table.

The two function units in Figure 28 are used to compute (24), one per iteration. Each unit contains three real-valued multipliers, a subtracter, and a one-bit logical left shifter for realizing the 'divide by 2' operation. To increase the processing throughput, both units and their connections are pipelined, shown in the bold vertical lines in Figure 28. Besides the square root and its reciprocal, a flow control signal 'ready' is provided on the output for indicating the computation status.

Node Perturbation Unit

In the adopted MIMO symbol detector MMSE-NP, one of the important steps is to find the nearest sibling symbols to the initial MMSE result $\boldsymbol{x}_p^{\mathrm{MMSE}}$ (12) based on the criterion of (14). The entire perturbation process involves enormous fine-grained data manipulations, since $\boldsymbol{x}_p$ consists of normalized constellation points that are drawn from a finite set of integers, e.g., $\sqrt{42}x_{p(i)} \in [\pm 1, \pm 3, \pm 5, \pm 7]$ for 64-QAM. Therefore, the node perturbation process is implemented as an accelerator for attaining high implementation efficiency.

One way to find Ω_i closest symbols for each spatial layer is to compute the distance between all possible M-QAM constellation points to $x_{p(i)}^{\mathrm{MMSE}}$, sort them in ascending order, and pick the first Ω_i points that have the smallest distance values. However, this brute-force method has high complexity, requiring 2 multiplications and 3 additions per constellation point and an M-point exhaustive search at the end. In contrast, this work adopts a Fast Node Enumeration (FNE) scheme, aiming to reduce the computational complexity by exploiting the geometric and symmetric properties of M-QAM. To better explain this, the following discussion focuses on 64-QAM and assumes $\max\{\Omega_i\} = 5$. Other system configurations can be processed by using the similar concept.

Figure 29 illustrates the basic principle of the proposed FNE scheme. The horizontal and vertical axis represent distance (δ) between the constellation points and the initial hard-output MMSE result $\hat{x}_{p(i)}^{\mathrm{MMSE}}$, for real and imaginary part, respectively. With this definition, $\hat{x}_{p(i)}^{\mathrm{MMSE}}$ has the coordinate of $(\delta_{\mathrm{re}}, \delta_{\mathrm{im}}) = (0, 0)$. In this example, N_e is the closest symbol to $x_{p(i)}^{\mathrm{MMSE}}$, assuming $x_{p(i)}^{\mathrm{MMSE}}$ lies within the dashed box in Figure 29. The distance between $x_{p(i)}^{\mathrm{MMSE}}$ and N_e is expressed as

$$\delta = x_{p(i)}^{\mathrm{MMSE}} - \mathrm{N}_e = a + jb. \tag{25}$$

Utilizing the symmetric and equidistant property of M-QAM, a and b in (25) can be shifted around N_e to reduce the search space to the first quadrant. As a result, $\{a, b\} \in [0, 1]$. To find the remaining neighboring symbols, distances between $x_{p(i)}^{\mathrm{MMSE}}$ and other constellation points in Figure 29 are computed and sorted in ascending order. By analyzing the resulting order of those symbols with respect to the position of $x_{p(i)}^{\mathrm{MMSE}}$, the search space can be divided into six unique zones that cover all possible symbol sequences. These zones are labelled with A_1 to A_6, see Figure 29. The corresponding symbol sequences are listed in Table 7.

To determine the zone in which $x_{p(i)}^{\mathrm{MMSE}}$ resides, the real and imaginary value of δ in (25) are compared using the following criteria:

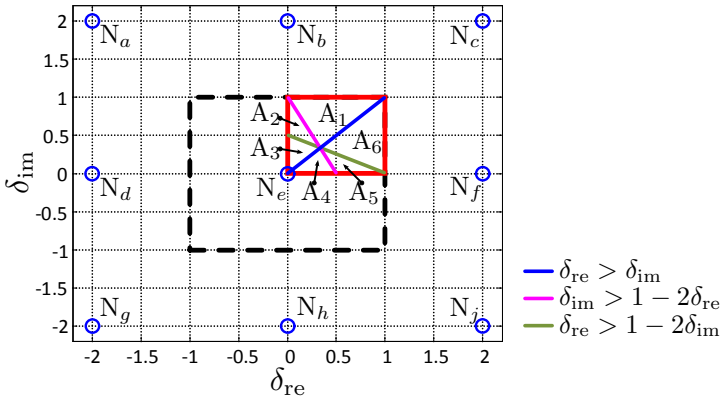


Figure 29: Illustration of the fast node enumeration scheme.

Table 7: Symbol sequences with respect to the position of $x_{p(i)}^{\mathrm{MMSE}}$.

Position of $x_{p(i)}^{\mathrm{MMSE}}$	A_1	A_2	A_3	A_4	A_5	A_6
Symbol sequence	N_e	N_e	N_e	N_e	N_e	N_e
	N_b	N_b	N_b	N_f	N_f	N_f
	N_f	N_f	N_f	N_b	N_b	N_b
	N_c	N_d	N_d	N_h	N_h	N_c
	N_d	N_c	N_h	N_d	N_c	N_h

1. $\delta_{\mathrm{re}} > \delta_{\mathrm{im}}$,
2. $\delta_{\mathrm{im}} > 1 - 2\delta_{\mathrm{re}}$,
3. $\delta_{\mathrm{re}} > 1 - 2\delta_{\mathrm{im}}$.

These comparisons correspond to three boundary lines inside the dashed box in Figure 29. Once $x_{p(i)}^{\mathrm{MMSE}}$ is positioned, all the required nearest sibling symbols are obtained. This is realized with the help of a look-up table, which stores all symbol sequences listed in Table 7 with different boundary check. Worth mentioning is that the adopted FNE scheme can be applied to other cases, where the search space is not at the center of the constellation map, for example, at corners or borders.

Figure 30 shows the block diagram of the node perturbation unit. It consists of N FNE units, one for each spatial layer, and a candidate vector generation unit for constructing candidate vectors by using the FE-CVG method (2.3.1).

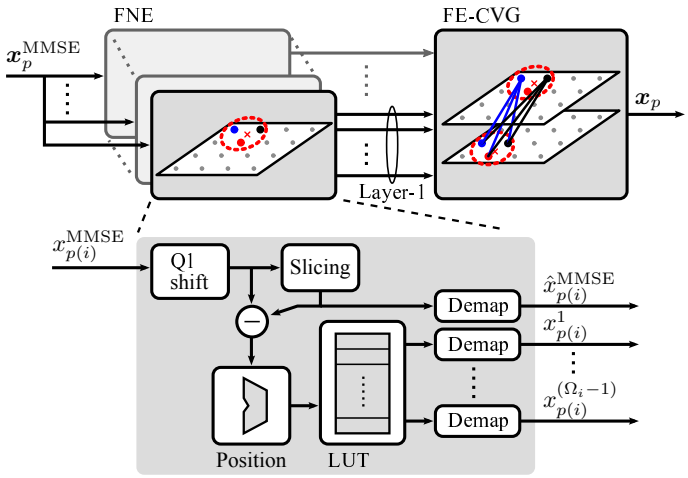


Figure 30: Block diagram of the node perturbation unit.

The FNE process starts by shifting the input $x_{p(i)}^{\mathrm{MMSE}}$ into the first quadrant. The initial hard-output symbol $\hat{x}_{p(i)}^{\mathrm{MMSE}}$, the closest symbol, is found by slicing $x_{p(i)}^{\mathrm{MMSE}}$ to the nearest constellation point. After calculating the distance (δ) between $\hat{x}_{p(i)}^{\mathrm{MMSE}}$ and $x_{p(i)}^{\mathrm{MMSE}}$, the position block carries out all comparisons of δ in parallel to determine the position of $x_{p(i)}^{\mathrm{MMSE}}$. The resulting zone number, ranging from A_1 to A_6, is used as an input to the following look-up table to obtain the remaining symbols. The FNE process is completed by shifting the expanded symbols back to the original quadrant, performed by the de-mapping units.

4.5 Concurrent Candidate Evaluation

In this section, a technique to further improve the implementation efficiency is presented. Among the MIMO signal processing, the ED calculation in (16) is the most compute-intensive operation, which needs to be performed at every data-carrying sub-carrier and for each of the L candidate vectors, e.g., $L = 24$ for $\boldsymbol{\Omega} = [\mathrm{F}, 4, 3, 2]$. A straightforward mapping of this on the SIMD core tends to incur low hardware utilization, at most 50% when computing $\underline{\boldsymbol{R}}\boldsymbol{x}_p$, since $\underline{\boldsymbol{R}}$ is an upper triangular matrix with real-valued diagonal elements. This is impermissible from the hardware efficiency point of view. To tackle this problem, the property of $\underline{\boldsymbol{R}}$ is utilized in such a way that two candidate vectors are concurrently evaluated, with the second $\underline{\boldsymbol{R}}\boldsymbol{x}_p$ operation mapped to the

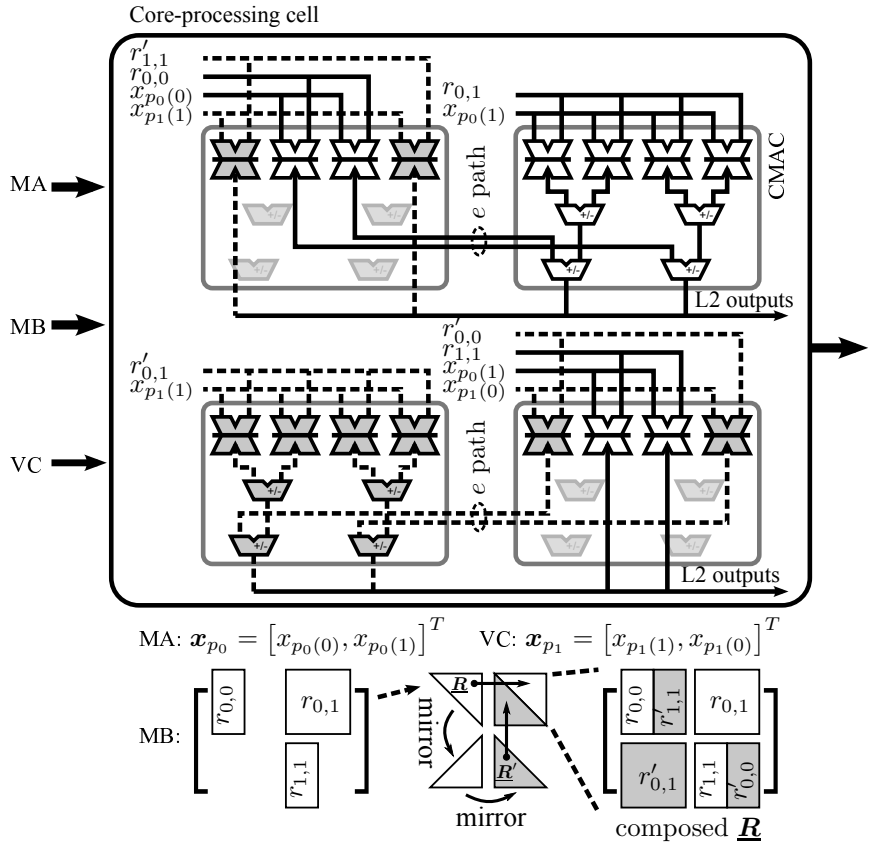


Figure 31: Concurrent candidate evaluation using the SIMD-based core-processing cell, an example of a 2×2 MIMO system. Internal data multiplexers are omitted for simplicity. Shaded blocks and dashed lines illustrate the computation of the second $\underline{\boldsymbol{R}}\boldsymbol{x}_p$ operation, denoted as $\underline{\boldsymbol{R}'}\boldsymbol{x}_{p_1}$.

lower triangular part of the SIMD core. In the following, computation of $\underline{\boldsymbol{R}}\boldsymbol{x}_p$ in a 2×2 MIMO system using a 2×2 SIMD core is given as an example to better illustrate the concurrent candidate evaluation.

To fully utilize the $N \times N$ CMAC units in the SIMD core, $\underline{\boldsymbol{R}}$ is duplicated to process two different $\boldsymbol{x}_p$ vectors at the same time. This matrix duplication is achieved by mirroring the $\underline{\boldsymbol{R}}$ matrix in both vertical and horizontal directions, such that the two matrices ($\underline{\boldsymbol{R}}$ and its counterpart $\underline{\boldsymbol{R}'}$) together compose a full square matrix, as illustrated at the bottom of Figure 31. The required

input vectors $\boldsymbol{x}_{p_0}$ and $\boldsymbol{x}_{p_1}$ are fed to the SIMD core via the matrix path MA and the vector path VC, respectively. Note that reserve-order permutation is required for the vector input $\boldsymbol{x}_{p_1}$ to match with the matrix orientation of $\underline{\boldsymbol{R}}'$ (flipped upside down). Figure 31 illustrates input data arrangement and internal data processing flow of the SIMD core. The ones associated with $\underline{\boldsymbol{R}}'\boldsymbol{x}_{p_1}$ computation are depicted in shaded blocks and dashed lines. In the diagonal CMAC units, processing of both $\boldsymbol{x}_{p_0}$ and $\boldsymbol{x}_{p_1}$ co-exist because of the real-valued diagonal elements in $\underline{\boldsymbol{R}}$. Additionally, both level-1 and level-2 adders are bypassed in these CMACs, shaded in light grey in Figure 31, since only real-valued multiplications are performed. Final vector outputs of both computations are conveyed to the following processing cell via level-2 outputs. As a result of this concurrent candidate evaluation, both hardware utilization and processing throughput are doubled for $\underline{\boldsymbol{R}}\boldsymbol{x}_p$ computations.

It is worth mentioning that various techniques have been presented in literature for improving the hardware efficiency of ED computations. For example, utilizing the property of the candidate vectors $\boldsymbol{x}_p$ (i.e., constellation points), [54] and [43] simplify the computation of $\underline{\boldsymbol{R}}\boldsymbol{x}_p$ by performing finite alphabet multiplications. However, applying those accelerator-based design techniques on vector processors may be infeasible or cost ineffective, as they require either fine-grained data manipulations (e.g., to realize finite alphabet multiplications) or increased data-path width (e.g., to hold multiple $\underline{\boldsymbol{R}}\boldsymbol{x}_p$ outputs). In contrast, the adopted scheme utilizes the existing structure of the SIMD core and only requires a few specialized signals for controlling diagonal CMAC units. Thus, it provides a balance between hardware efficiency and complexity.

5 Implementation Results and Comparison

To cope with different system configurations and design constraints on, for example, antenna size and processing throughput, the heterogeneous cell array is fully parametrizable at system design-time. Figure 32 shows the detailed architecture of the cell array configured for the target 20 MHz 4×4 MIMO LTE-A downlink. Processing and memory cells in the vector domain are labelled with 'VPC-$\mathcal{X}$' and 'VMC-$\mathcal{X}$' respectively, while counterparts in the scalar domain are denoted as 'SPC-$\mathcal{X}$' and 'SMC-$\mathcal{X}$'. All data computations are performed in 16 bits fixed-point arithmetic with 8 guard bits for accumulations. In Tile-0, the core-processing cell is configured to have 4×4 CMAC units. The post-processing cell contains two 3-bit barrel shifters deployed for handling data from matrix and vector bus respectively. The register bank consists of 16 general purpose vector registers, 16 VPRs, and 16 MMRs. Each distributed configuration memory can store up to 16 hardware configurations, and the program memory is capable of storing 256 micro-code sequences.

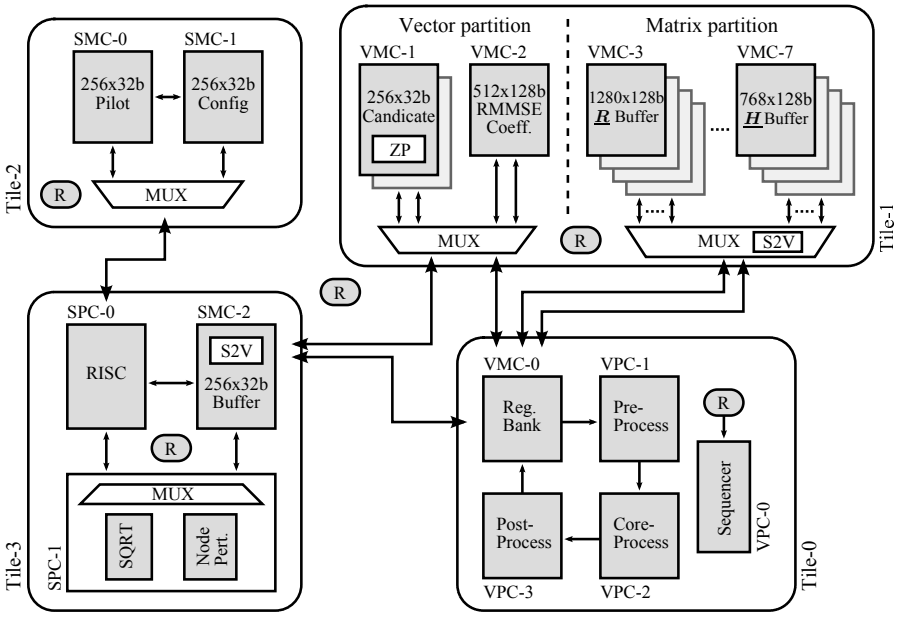


Figure 32: Architecture diagram of the heterogeneous cell array configured for baseband processing in a 20 MHz 4×4 MIMO LTE-A downlink.

Table 8: Memory configurations and usages.

Resource cell		Memory		Reserved	Usage
Tile-0	VPC-0	256×32b	8 Kb	N/A[a]	Program memory
	VPC-1−3 VMC-0	(16×32b)×7	3.5 Kb	N/A[a]	Configuration memory
Tile-1	VMC-1	(256×32b)×2	16 Kb	43.75%	Candidate vector buffer
	VMC-2	512×128b	64 Kb	43.75%	R.MMSE-SW $\mathcal{W}$ ROM
	VMC-3	(1280×128b)×4	640 Kb	6.25%	$\underline{\boldsymbol{R}}$ buffer
		1280×8b	10 Kb	6.25%	Access index
	VMC-4−7	(768×128b)×16	1536 Kb	8.85%	$\underline{\boldsymbol{H}}$ buffer
		(768×8b)×4	24 Kb	8.85%	Access index
Tile-2	SMC-0	256×32b	8 Kb	42.38%	Pilot ROM & data buffer
	SMC-1	256×32b	8 Kb	N/A[a]	Configuration memory
Tile-3	SPC-0	384×48b	18 Kb	N/A[a]	Program memory
	SPC-1	N/A	N/A	N/A	N/A
	SMC-2	256×32b	8 Kb	N/A[a]	Data buffer
Network		N/A	N/A	N/A	N/A

[a] Reserved space is not assessed for control and configuration memories.

The generic processor in Tile-3 is configured to have a 3-bit barrel shifter, 11-bit one-level inner loop controller, 16 general purpose scalar registers, and a 18 Kb program memory. The node perturbation unit in SPC-1 is able to extend each symbol with up to 5 nearest neighbors and generate one candidate vector in every clock cycle.

As for memory cells, each is configured to have 4 DSCs. In VMC-2, SMC-0, and SMC-2, the micro-block function is enabled. Data memories are deployed mainly to suffice the storage requirement of the target LTE-A setup. In the current design, the array contains 2.34 Mb of memory, in which 88% are data buffers for keeping data required in one LTE-A time slot (e.g., channel and decomposed matrices), 2% are control memories for storing instructions and resource configurations, and 10% are reserved space for facilitating flexible algorithm mappings and future system updates. Detailed memory configurations of all RCs are summarized in Table 8. Data transfers from vector to scalar RCs are bridged by memory cells using the micro-block function, whereas the reverse paths are handled by dedicated scalar-to-vector adapters, shown as 'S2V' in Figure 32. Each adapter contains a vector register and a FSM, capable of transmitting N scalar data packets in a vector form to the receiving end. Moreover, it should be mentioned that VMC-1 in Tile-1 is dedicated to storing candidate vectors $\boldsymbol{x}_p$ in symbol detection. Therefore, the wordlength of the memory is substantially reduced by only storing M-QAM values. For 64-QAM modulation, each symbol in $\boldsymbol{x}_p$ requires 2×4 bits (complex-valued) instead of 2×16, reducing the memory requirements by 4 times. During memory reading, the M-QAM values are extended to the vector format by padding zeros. This QAM-to-vector converter is denoted as 'ZP' in Figure 32.

5.1 Implementation Results

The cell array is modelled in VHDL, synthesized using Synopsys Design Compiler with a 65 nm CMOS standard digital cell library, and routed using Cadence SoC Encounter. Counting a two-input NAND gate as one equivalent gate, the whole array contains 2.76 M gates and has a core area of 8.88 mm^2 at 74% cell density in chip layout. Data buffers occupy more than 60% of the area, while the logic blocks, including control memories and the hierarchical network, share the rest. Excluding those data buffers, it shows in Table 9 that most of the logic gates are devoted to the vector processing domain (i.e., Tile-0 and 1) and the on-chip network takes less than 5%. At 1.2 V nominal core voltage supply, a maximum clock frequency of 500 MHz is obtained from post-layout simulations with back annotated timing information. At this frequency, the array is capable of performing 8.5 G CMACs per second, considering the 4×4 CMAC bank in Tile-0 and the CMAC unit in the generic processor in Tile-3.

Table 9: Area and power breakdown of the cell array with data buffers excluded.

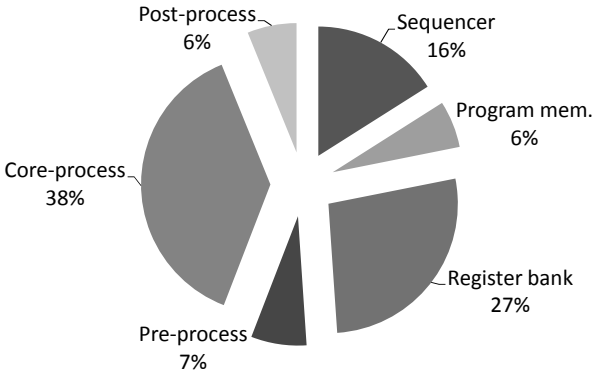
Resource cell		Gate count [KG]		Power [mW]	
Tile-0		367	34.77%	164.93	53.75%
Tile-1	Vector partition	96	9.12%	5.99	1.95%
	Matrix partition	365	34.60%	68.06	22.18%
Tile-2		47	4.44%	3.20	1.04%
Tile-3	Generic processor	70	6.60%	44.10	14.37%
	Others	61	5.83%	16.98	5.53%
Network		49	4.65%	3.56	1.16%
Total		1,055	100.00%	306.84	100.00%

Vector Dataflow Processor

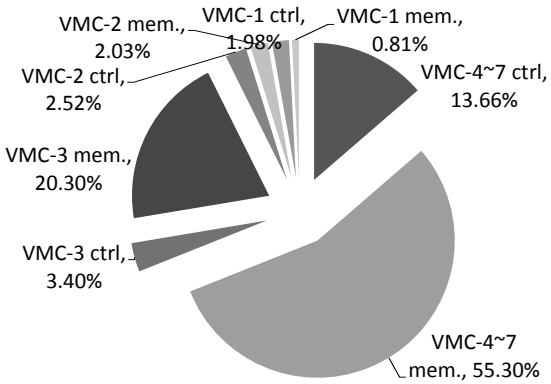
Figure 33(a) shows the area breakdown of Tile-0, the vector dataflow processor. The control logic of the processor, including the sequencer and the program memory, occupy 22% of the total area, while the processing cells take 51% and the register bank consumes 27%. Note that the distributed configuration memories are counted as part of the sequencing control in Figure 33(a). The relatively low area consumption of the control logic reveals the low control overhead of the processor, thanks to the adopted distributed micro-code execution scheme. Among the multi-stage computation path, the core-processing cell consumes most of the area due to the deployed homogeneous CMAC bank.

Vector Data Memory Tile

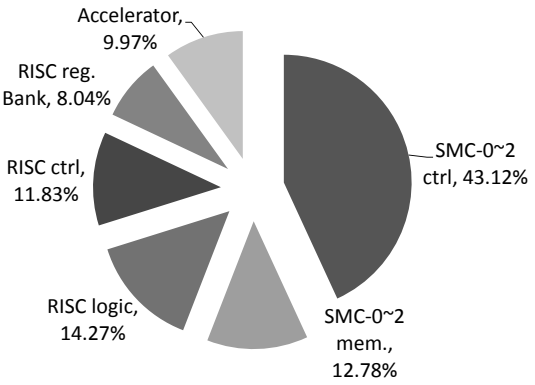
Tile-1 consists of two memory cells, VMC-1 to 2, and five memory pages, VMC-3 to 7. Because of the large storage requirements of the application, e.g., to store channel matrices $\underline{\boldsymbol{H}}$ and $\underline{\boldsymbol{R}}$ for all 1200 subcarriers and coefficients $\boldsymbol{\mathcal{W}}$ for the R.MMSE-SW estimator, most of the area in Tile-1 is consumed by memory macros, see Figure 33(b). Among the control logic, memory pages consume 80% of the area, due to the employed flexible access schemes such as matrix data transposition and access indexing. It is worth mentioning that VMC-2 is configured to have the micro-block function used to interface with RCs in the scalar processing domain. Therefore, it can be seen from Figure 33(b) that the control logic of VMC-2 consumes slightly more area than that of the VMC-1.



(a) Vector dataflow processor (Tile-0)



(b) Vector data memory tile (Tile-1)



(c) Tile-2 & 3 in the scalar processing domain

Figure 33: Area breakdown of RCs in the reconfigurable cell array.

Resource Cells in Scalar Processing Domain

In the scalar processing domain, the generic processor and accelerators occupy around 45% of the area, see Figure 33(c). Compared to the vector dataflow processor in Tile-0, the generic processor is equipped with a larger program memory and a simpler data path, since it is designed to mainly perform control related operations, such as conditional instruction execution and configuration of other RCs. This can be seen from the area partition in Figure 33(c), where the control logic of the processor takes almost the same area as its logic part, i.e., 11.83% versus 14.27%.

5.2 Task Mapping and Timing Analysis

The MIMO processing tasks, i.e., channel estimation, pre-processing, and symbol detection, are manually mapped onto the cell array with a primary focus on sufficing the stringent timing constraint and achieving high processing throughput. To this end, multi-subcarrier processing is adopted in all tasks and is scheduled based on the LTE-A resource block, i.e., 12 consecutive subcarriers. The number of blocks to process in each computation step is determined manually based on the computation and communication latency and available hardware resources. For example, MMSE-SQRD is programmed to operate on 2 LTE-A resource blocks in each step due to its high data dependency and the long latency involved in obtaining results from the inverse square root unit. In contrast, the other two tasks work with 1 resource block at a time. In addition to the multi-subcarrier processing, most of the data transfers are scheduled to utilize the low-latency high-bandwidth local interconnects, while the hierarchical network is mainly used for resource configurations and the streaming of external data such as receiving vector $\boldsymbol{y}$ and decoded $\hat{\boldsymbol{x}}$. In the following, detailed mapping of the MIMO processing tasks is described.

Channel Estimation

Recall that channel estimation contains two computation steps, LS estimation at pilot tones and $\boldsymbol{H}$ interpolation for data-carrying subcarriers. In this work, the LS computation is performed by the generic processor in Tile-3 and results are stored in an LS buffer (VMC4−7) in Tile-1. The computation starts immediately data at pilot positions have been received. Data transfers between SPC-0 and memory pages in Tile-1 are carried out through the hierarchical network. In Tile-1, the received scalar data packets are converted to the vector format before writing to the memory. This is accomplished by using the S2V unit deployed in the matrix partition of Tile-1. Steps ① to ⑦ in Figure 34 illustrate the processing flow of the LS computation. Using LS estimated channel coefficients, the data-tone $\boldsymbol{H}$ interpolation is performed by the

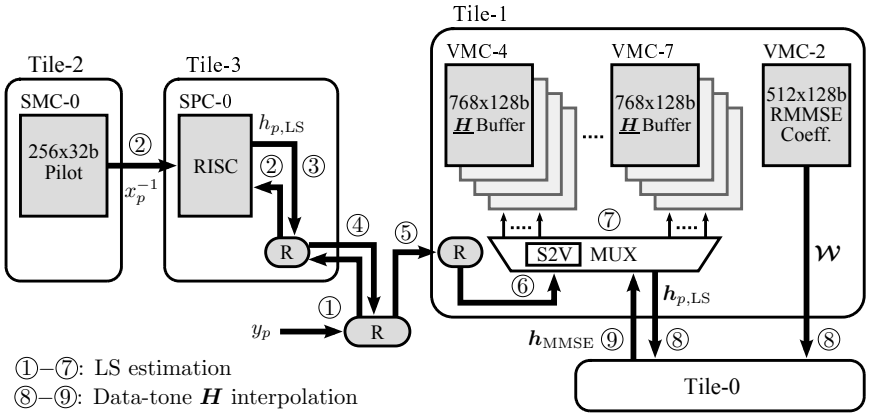


Figure 34: Processing flow of the channel estimation, performed by the generic processor in Tile-3 and the vector dataflow processor (Tile-0).

Table 10: Pseudo-code of the data-tone $\boldsymbol{H}$ interpolation performed in Tile-0.

```
for i = 1 : 100 do                 % Loop for 100 LTE-A resource blocks
  for j = 1 : 4 do                 % 4 groups per LTE-A resource blocks
    Copy H @ pilot position from "LS buffer" to "H buffer"
    for k = 1 : 2 do               % 2 data tones per group per resource block
      for l = 1 : 6 do             % 6 iterations per data tone for N_SW = 24
        for n = 1 : 4 do           % 4 Tx spatial layers, i.e., 4 columns of H
          'VDP(ab + e)' to compute h_MMSE = W h_p,LS
        end for
      end for
    end for
  end for
end for
```

vector dataflow processor. To fully utilize the 4×4 CMAC bank, four Rx spatial layers (i.e., rows of $\boldsymbol{H}$) are computed in parallel, one per processing lane, in each clock cycle. For $N_{\mathrm{SW}} = 24$, each interpolation process requires $24/4 = 6$ iterations to complete. The intermediate results are accumulated in the post-processing cell in Tile-0. Since the pilot tones reside in every third subcarrier in an OFDM symbol (Figure 2), processing of each LTE-A resource block is divided into 4 groups, each containing one pilot and two data tones. Table 10

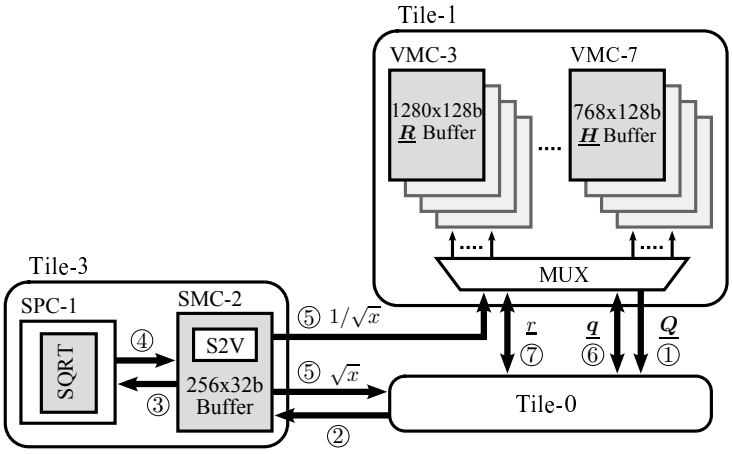


Figure 35: Processing flow of the MMSE-SQRD based channel pre-processing performed by the vector dataflow processor (Tile-0).

shows the pseudo-code of the data-tone $\boldsymbol{H}$ interpolation. Taking advantage of the adopted multi-level zero-delay inner loop controller, loop operations are used whenever possible, aiming to attain a modular program structure and to ease parameter updates.

Channel Pre-processing

The MMSE-SQRD based channel pre-processing is mainly performed by the vector dataflow processor, except that $1/\sqrt{x}$ and $\sqrt{x}$ operations are outsourced to the inverse square root unit in Tile-3. Taking the augmented channel matrix $\underline{\boldsymbol{H}}$ as an input, the MGS-based MMSE-SQRD performs matrix orthogonalization iteratively based on Algorithm 1. The corresponding task mapping on the vector dataflow processor and the layout of data storage in Tile-1 are illustrated in Figure 26. It should be pointed out that, among numerous task mapping schemes, the adopted approach focuses on the utilization of hardware resources in Tile-0 and the modularity of the program structure. Briefly, the processor works with one subcarrier in each clock cycle during the norm computation of $\underline{\boldsymbol{Q}}$, whereas two subcarriers are processed in parallel during other operations, i.e., column vector updates in $\underline{\boldsymbol{Q}}$ and the computation of $\underline{\boldsymbol{R}}$. Upon the completion of the matrix orthogonalization process, $\underline{\boldsymbol{H}}^{-1} = 1/\sigma_n \boldsymbol{Q}_b \boldsymbol{Q}_a^H$ is computed as a post-processing, required in the following symbol detection (12). Considering the latency of the inverse square root operation, i.e., 3 clock cycles computation latency (Section 4.4.1) plus 4 clock cycles communication

Table 11: Pseudo-code of MMSE-SQRD, computations performed in Tile-0.

for $i = 1 : 50$ **do** *% Loop for 50 pairs of LTE-A resource blocks*
  for $j = 1 : 4$ **do** *% 4 columns of* $\underline{\boldsymbol{H}}$
    for $k = 1 : 2 \times 12$ **do** *% Processing one subcarrier per clock cycle*
      'VDP($ab + e$)' to compute $\boldsymbol{\xi} = \left[\|\underline{\boldsymbol{q}}_{j+1}\|_2^2, \|\underline{\boldsymbol{q}}_{j+2}\|_2^2, \ldots, \|\underline{\boldsymbol{q}}_N\|_2^2\right]^T$
      Sorting to obtain $\boldsymbol{\xi}_{min}$
    end for
    for $k = 1 : 12$ **do** *% Processing two subcarriers per clock cycle*
      'bc' to compute $\underline{\boldsymbol{q}}_j = \underline{\boldsymbol{q}}_j / \underline{r}_{j,j}$
    end for
    for $k = 1 : 12$ **do** *% Processing two subcarriers per clock cycle*
      for $l = j + 1 : 4$ **do**
        'VDP($ab + e$)' to compute $\underline{r}_{j,l} = \underline{\boldsymbol{q}}_j^H \underline{\boldsymbol{q}}_l$
      end for
    end for
    for $k = 1 : 12$ **do** *% Processing two subcarriers per clock cycle*
      for $l = j + 1 : 4$ **do**
        '$a - bc$' to compute $\underline{\boldsymbol{q}}_l = \underline{\boldsymbol{q}}_l - \underline{r}_{j,l} \underline{\boldsymbol{q}}_j$
      end for
    end for
  end for
  for $j = 1 : 2 \times 12$ **do** *% Post-processing*
    'VDP($ab + e$)' & 'bc' to compute $\underline{\boldsymbol{H}}^{-1} = 1/\sigma_n \boldsymbol{Q}_b \boldsymbol{Q}_a^H$
  end for
end for

latency illustrated in Figure 35, two resource blocks are processed in each computation step of MMSE-SQRD. Table 11 shows the pseudo-code of the task mapping. Note that column permutations of matrices $\underline{\boldsymbol{Q}}$ and $\underline{\boldsymbol{R}}$ are realized by manipulating the accessing indexes (Section 4.3.2) of matrix pages in Tile-1, thus requiring no physical data exchanging.

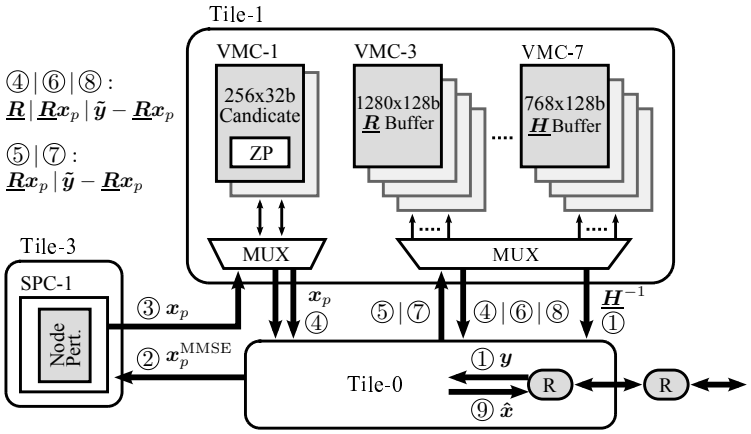


Figure 36: Processing flow of the MMSE-NP based symbol detection performed by the vector dataflow processor (Tile-0).

Symbol Detection

The MMSE-NP based symbol detection contains three main computation steps: initial MMSE detection, symbol expansion and candidate vector generation, and candidate evaluation. Among these, the second operation is performed by the node perturbation unit in Tile-3, while the other two are handled by the vector dataflow processor. Table 12 shows the pseudo-code of the computations performed in the vector processor. In view of the latency of communication between the vector processor and the node perturbation unit in Tile-3, see Figure 36, 12 adjacent data-carrying subcarriers are processed in each computation step of MMSE-NP. For the target 20 MHz LTE-A, each time slot has 7200 data-carrying subcarriers (Table 4), resulting in $7200/12 = 450$ blocks to be processed. To achieve high utilization of the CMAC bank in Tile-0, each ED computation (16) during the evaluation of candidate vectors is performed in three sub-steps: $\boldsymbol{\alpha} = \underline{\boldsymbol{R}}\boldsymbol{x}_p$, $\boldsymbol{\beta} = \tilde{\boldsymbol{y}} - \boldsymbol{\alpha}$, and vector norm $\|\boldsymbol{\beta}\|_2^2$. In each clock cycle, the computation of $\underline{\boldsymbol{R}}\boldsymbol{x}_p$ operates on two candidate vectors by using the concurrent candidate evaluation scheme (Section 4.5), while the other two vector-based operations work with four candidates at a time. Upon the completion of each vector norm computation $\|\boldsymbol{\beta}\|_2^2$, the four candidate vectors under evaluation are sorted in the post-processing cell in Tile-0. The one with the smallest ED value is temporarily stored in the register bank for further comparisons with other candidates. For $\boldsymbol{\Omega} = [\mathrm{F}, 4, 3, 2]$, there are in total 24 candidate vectors in each symbol detection, implying $24/4 = 6$ temporarily stored candidates at the end of the ED computation. The final detection out-

Table 12: Pseudo-code of symbol detection, computations performed in Tile-0.

for $i = 1 : 7200/12$ **do** *% Loop for 7200/12 blocks*
  for $j = 1 : 12$ **do** *% Loop for 12 subcarriers*
    'VDP$(ab + e)$' to compute $\boldsymbol{x}_p^{\mathrm{MMSE}} = \underline{\boldsymbol{H}}^{-1}\boldsymbol{y}$
  end for
  for $j = 1 : 12$ **do**
    'VDP$(ab + e)$' to compute $\tilde{\boldsymbol{y}} = \boldsymbol{Q}_a^H \boldsymbol{y}$
  end for
  for $j = 1 : 12$ **do**
    for $k = 1 : 24/2$ **do** *% Processing two $\underline{\boldsymbol{R}}\boldsymbol{x}_p$ per clock cycle*
      'VDP$(ab + e)$' to compute $\underline{\boldsymbol{R}}\boldsymbol{x}_p$
    end for
  end for
  for $j = 1 : 12$ **do**
    for $k = 1 : 24/4$ **do** *% Processing four $\tilde{\boldsymbol{y}} - \underline{\boldsymbol{R}}\boldsymbol{x}_p$ per clock cycle*
      '$a - b$' to compute $\tilde{\boldsymbol{y}} - \underline{\boldsymbol{R}}\boldsymbol{x}_p$
    end for
  end for
  for $j = 1 : 12$ **do**
    for $k = 1 : 24/4$ **do** *% Processing four $\|\cdot\|_2^2$ per clock cycle*
      'VDP$(ab + e)$' to compute $\|\tilde{\boldsymbol{y}} - \underline{\boldsymbol{R}}\boldsymbol{x}_p\|_2^2$ & sorting
    end for
  end for
  for $j = 1 : 12$ **do**
    for $k = 1 : \lceil 24/4/4 \rceil$ **do** *% Post-processing*
      Sorting to find $\hat{\boldsymbol{x}}_p = \arg\min_{\boldsymbol{x}_p \in \mathcal{S}} \|\tilde{\boldsymbol{y}} - \underline{\boldsymbol{R}}\boldsymbol{x}_p\|_2^2$
    end for
    $\hat{\boldsymbol{x}} = \boldsymbol{P}\hat{\boldsymbol{x}}_p$ *% Final detection output*
  end for
end for

put is obtained by comparing these 6 candidates, finding the one with the smallest value, and loading the corresponding candidate vector from VMC-1 together with the permutation matrix ($\boldsymbol{P}$ in (5)) from VMC4. Finally, the recovered transmitted vector $\hat{\boldsymbol{x}}$ with its original symbol sequence is sent out through the hierarchical network.

Table 13: Overhead analysis for the computation of MIMO signal processing.

	Execution time [Clock cycle]	Control time [Clock cycle]	Control overhead
Ch. Estimation	20,801	1,201	5.77%
QRD	22,451	851	3.79%
QRD[a]	15,101	701	4.64%
Detection	190,201	3,001	1.58%
Total	232,203	5,003	2.16%

[a] Without post-processing.

Miscellaneous Operations

Besides the aforementioned MIMO processing tasks, various miscellaneous operations are required, e.g., memory initialization for the permutation matrix $\boldsymbol{P}$ (5) and the augmented channel matrix $\underline{\hat{\boldsymbol{H}}}$ (6). These operations occupy only a fraction of the total processing time and are performed in the beginning of each processing iteration (i.e., time slot).

Results and Discussions

To assess the efficiency of the task mapping, control overhead is analyzed for program executions on the vector dataflow processor. Here the control is defined as non-computational operations, such as loop initializations and run-time program updates. Table 13 lists the total execution time and the number of control operations required for accomplishing three MIMO processing tasks. As can be seen, the total control overhead measured on the vector dataflow processor is only about 2% of the total execution time, thanks to the algorithm-architecture co-design. On the algorithm side, benefiting from the adopted streaming-based processing flow in each individual task, branching operations are completely eliminated, see Table 10−12. On the architecture side, with the employed configurable instruction set architecture, the number of hardware configurations (micro-codes) and program updates is substantially reduced, since the data-path of the processor can be dynamically configured to better suit target applications. Additionally, loop operations are assisted by the deployed multi-level zero-delay inner loop controller, thus requiring no manual loop manipulations from the user.

Table 14 summaries achieved performance of the three task mappings. Operating at 500 MHz, the total processing time for one LTE-A time slot is 469.72 μS. This fulfills the real-time requirement of the target LTE-A setup, i.e., $t_{\text{iter}} \leq t_{\text{slot}} = 0.5\,\text{mS}$ (see Section 3.4), and results in about 6% spare time

Table 14: Performance summary of the MIMO signal processing.

	Clock Cycle/Op	Time [μS]	Throughput	Power[b] [mW]	Energy[b]
Ch. Estimation	17.33	41.60	28.84 MEst/s	276.24	9.58 nJ/Est
QRD	18.71	44.90	26.72 MQRD/s	314.05	11.75 nJ/QRD
QRD[a]	12.63	30.30	39.60 MQRD/s	315.36	7.96 nJ/QRD
Detection	26.42	380.40	454.26 Mb/s	280.82	0.62 nJ/b
Miscellaneous	2.34	2.82	N/A	269.99	0.81 nJ/op
Total/Average	32.62	469.72	367.88 Mb/s	306.84	0.83 nJ/b

[a] Without post-processing.
[b] With data buffers excluded.

that can be used to map more advanced algorithms or upgrade system parameters such as the $\boldsymbol{\Omega}$ assignment in symbol detection. Based on the processing time and the number of tones/bits required to compute, Table 14 presents the corresponding throughput achieved in each task. On average, recovering one transmitted vector $\hat{\boldsymbol{x}}$, with all three processing tasks involved, requires 32.62 clock cycles, which is equivalent to a throughput of 367.88 Mb/s.

5.3 Computation Efficiency

To evaluate the computation efficiency of the array, resource utilization of the SIMD core in Tile-0 is measured as a representative, since it contributes to more than 90% of the total computation capacity. Thanks to the vector enhanced SIMD structure (Section 4.2.2) and the multi-stage computation chain (Section 4.2.3), an average utilization of 77% is achieved during the whole MIMO signal processing. Figure 37 reports a detailed utilization graph during the computation of two LTE-A resource blocks. Among those tasks, the miscellaneous processing shows the lowest utilization value, as the vector processor during that time interval only performs simple operations, such as vector scaling and data masking used for initializing registers and memory cells.

5.4 Power and Energy Consumption

Power consumption of the cell array is obtained from Synopsys PrimeTime using the post-layout design annotated with switching activities. At 500 MHz with 1.2 V supply voltage, the average power consumption for processing one data-carrying tone is 548.78 mW, including 306.84 mW from the logic blocks and 241.94 mW from the data buffers. The corresponding energy consumption for processing one information bit is 0.83 nJ/b and 1.49 nJ/b, without and

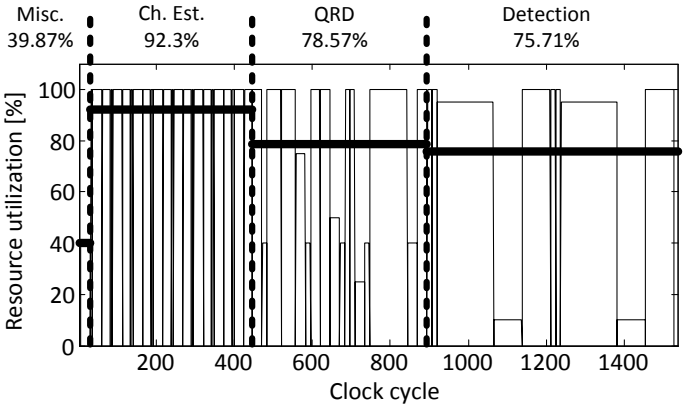


Figure 37: Utilization of the SIMD core in Tile-0 during MIMO signal processing of two LTE-A resource blocks (24 subcarriers). Horizontal lines in the figure show the average utilization of the corresponding task.

with data buffers respectively. Table 14 summaries average power and energy consumption of different tasks with data buffers excluded. As can be seen, power consumption of different task mappings is quite balanced because of the high computation efficiency of the cell array achieved by the algorithm-architecture co-design.

To acquire a more comprehensive understanding of the power distribution, a tile-level power breakdown of the array is listed in Table 9 and plotted in a pie-diagram in Figure 38(a). Among all, Tile-0 is the most power consuming block, because of the large area occupation and high resource utilization throughout the whole processing period. Further, the power distribution in Tile-0 is presented in Figure 38(b). The SIMD core accounts for 50% of the power. The register bank consumes only 16% thanks to that the developed multi-stage computation chain substantially reduces intermediate result buffering. Moreover, the hierarchical network of the array consumes only $\sim$1% of the power (Figure 38(a)), due to the high allocation of local data transfers. Worth mentioning is that no special low power physical design techniques, such as clock and power gating and multiple power islands, are adopted in the current array. Therefore, further power savings are possible with a more advanced back-end design. Additionally, it should be pointed out that simulated power figures from the post-layout design may be different for chip measurement results.

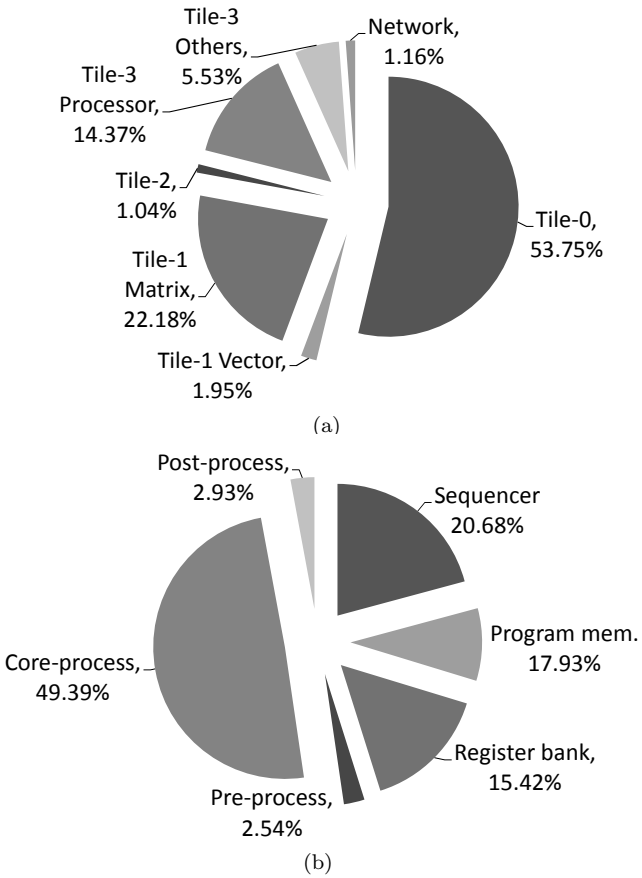


Figure 38: Power breakdown of (a) the reconfigurable cell array and (b) the vector dataflow processor (Tile-0).

5.5 Comparison and Discussion

In this section, implementation results and various performance metrics of the cell array are summarized and compared with previously reported designs in open literature. In fact, a fair quantitative comparison with related work is not an easy task due to many different design factors, such as flexibility, algorithm selection, performance, and operating scenario. Which platform/technology to choose is highly dependent on design specifications such as system setup, area and power budget, Quality of Service (QoS), and scalability requirements. Therefore, the following discussion only serves to give an overview of the design efficiency for related implementations and aims to position the proposed cell array with respect to different performance measures. To ease the discussion, related hardware architectures are divided into three broad categories: task specific accelerators (ASICs), programmable and reconfigurable platforms (e.g., DSPs, FPGAs, and GPUs), and domain-specific reconfigurable platforms (e.g., baseband processors).

To ensure a fair enough comparison, technology scaling is considered. Taking the employed setup "65 nm CMOS technology and 1.2 V supply voltage" as a reference, implementation results of all related work are normalized using a process scaling factor s [118]. The definition of s and the normalization of frequency, area, and power consumption are

$$\begin{aligned} s &= \frac{\text{Techology}}{65\,\text{nm}}, \\ \text{Frequency}_{\text{norm}} &\approx \text{Frequency} \times s, \\ \text{Area}_{\text{norm}} &\approx \text{Area} \times \frac{1}{s^2}, \\ \text{Power}_{\text{norm}} &\approx \text{Power} \times \frac{1}{s} \left(\frac{1.2\,\text{V}}{\text{Voltage}} \right)^2 . \end{aligned} \tag{26}$$

In Table 15-17, the cell array is compared with three aforementioned architecture categories, respectively. Their performances are evaluated by assessing area and power efficiency as well as hardware flexibility.

Table 15: Comparison of the cell array with reconfigurable baseband processors.

		09 [62]	10 [58]	10 [119]	08 [120]	11 [67]	10 [6]	09 [121]	This work	
Antenna		–	–	4×4	2×2	4×4	4×2	2×2	4×4	
Modulation (QAM)		64	N/A	–	64	64	N/A	N/A	64	
Mapping (CE | QRD | DT)		✓|–|✓	✓|–|✓	–|✓|–	–|–|✓	–|–|✓	✓|✓|✓	✓|✓|✓	✓|✓|✓	
Technology [nm]		120	130	65	90	130	65	90	65	
Area [mm^2]		11	11	N/A	N/A	N/A	16.06[c]	32	8.88	
Gate count [KG]		200[a]	N/A	824[a]	1200	71[a]	5969[c]	N/A	2760	1055[a]
Frequency [MHz]		70	300	234	600	277	400	400	500	
Power[b] [mW]		37.92	86.40	169[a]	642	20.48[a]	219[c]	240	549	307[a]
Throughput[b]	Ch. Est. [MEst/s]	N/A	N/A	–	–	–			28.84	
	QRD [MQRD/s]	–	–	10.64	–	–	N/A	N/A	39.60	
	Detection [Mb/s]	N/A	N/A	–	49.60	134			454.26	
	Total [Mb/s]	58.47	4	–	–	–	10.8	150	367.88	
Area. Eff.[b]	Ch. Est. [KEst/s/KG]	N/A	N/A	–	–	–			10.45	27.34[a]
	QRD [KQRD/s/KG]	–	–	12.91[a]	–	–	N/A	N/A	14.35	37.54[a]
	Detection [Kb/s/KG]	N/A	N/A	–	41.33	1890[a]			165	431[a]
	Total [Kb/s/KG]	292.34[a]	N/A	–	–	–	1.81	N/A	133	349[a]
Energy[b]	Ch. Est. [nJ/Est]	N/A	N/A	–	–	–			12.70	9.58[a]
	QRD [nJ/QRD]	–	–	15.85[a]	–	–	N/A	N/A	15.27	7.96[a]
	Detection [nJ/b]	N/A	N/A	–	1.49	0.3[a]			0.99	0.62[a]
	Total [nJ/b]	1.2	43.2	–	–	–	N/A	2.23	1.49	0.83[a]

[a] With data buffers excluded.
[b] Normalized to 65 nm with 1.2 V core voltage.
[c] Only counted relevant parts of the chip.

Table 16: Comparison of the cell array with programmable platforms.

		08 [122]		12 [123]		09 [124]	10 [125]	12 [126]	This work	
Platform		FPGA	DSP	CPU	GPU	GPU	GPU	GPU	Reconfig.	
Antenna		4×4		4×4		4×4	2×2	4×4	4×4	
Modulation (QAM)		16		64		64	16	64	64	
Mapping (CE | QRD | DT)		−|−|✓		−|−|✓		−|−|✓	−|−|✓	−|✓|✓	✓|✓|✓	
Technology [nm]		130	180	45	40	65	80	40	65	
Area [mm^2]		26[c]	96[c]	296	529	196	N/A	306.82[c]	8.88	
Gate count [KG]		N/A	N/A	1.94e5	7.75e5	1.26e5	2.39e5[c]	4.5e5[c]	2760	1055[a]
Frequency [MHz]		251	200	3070	1150	1900	920	1150	500	
Power[b] [mW]		624[c]	311[c]	101e3	557e3	137e3	9600[c]	323e3[c]	549	307[a]
Throughput[b]	Ch. Est. [MEst/s]	–	–	–	–	–	–	–	28.84	
	QRD [MQRD/s]	–	–	–	–	–	–	N/A	39.60	
	Detection [Mb/s]	163	10.14	0.18	66.09	12.68	44.38	10.58	454.26	
	Total [Mb/s]	–	–	–	–	–	–	–	367.88	
Area. Eff.[b]	Ch. Est. [KEst/s/KG]	–	–	–	–	–	–	–	10.45	27.34[a]
	QRD [KQRD/s/KG]	–	–	–	–	–	–	N/A	14.35	37.54[a]
	Detection [Kb/s/KG]	N/A	N/A	9.29e-4	0.0853	0.1	0.186[c]	0.0235[c]	165	431[a]
	Total [Kb/s/KG]	–	–	–	–	–	–	–	133	349[a]
Energy[b]	Ch. Est. [nJ/Est]	–	–	–	–	–	–	–	12.70	9.58[a]
	QRD [nJ/QRD]	–	–	–	–	–	–	N/A	15.27	7.96[a]
	Detection [nJ/b]	1.32[c]	9.15[c]	3.88e5	1.24e5	2.47e5	2.13e3[c]	3.79e6[c]	0.99	0.62[a]
	Total [nJ/b]	–	–	–	–	–	–	–	1.49	0.83[a]

[a] With data buffers excluded.
[b] Normalized to 65 nm with 1.2 V core voltage.
[c] Only counted relevant parts of the chip.

Table 17: Comparison of the cell array with ASIC implementations.

		11 [127]	13 [128]	10 [129]	13 [130]	10 [112]	13 [131]	11 [132]	This work	
Platform		ASIC							Reconfig.	
Antenna		–	–	4×4	4×4	4×4	4×4	4×4	4×4	
Modulation (QAM)		–	–	–	–	64	64	64	64	
Mapping (CE | QRD | DT)		✓|−|−	✓|−|−	−|✓|−	−|✓|−	−|−|✓	−|−|✓	✓|✓|✓	✓|✓|✓	
Technology [nm]		65	65	180	130	130	130	90	65	
Area [mm^2]		0.68^{a,c}	2.56^{a,c}	N/A	0.3^{a}	3.9^{a}	N/A	2.02^{a}	8.88	
Gate count [KG]		325^{a,c}	563^{a,c}	127.5^{a}	36^{a}	491^{a}	340^{a}	505^{a}	2760	1055^{a}
Frequency [MHz]		250	70	400	278	137.5	417	114	500	
Power^{b} [mW]		154^{a,c}	73.16^{a,c}	6.1^{a}	19.92^{a}	63.6^{a}	55^{a}	59.07^{a}	549	307^{a}
Throughput^{b}	Ch. Est. [MEst/s]	78	11.88	–	–	–	–	N/A	28.84	
	QRD [MQRD/s]	–	–	7.91	13.9	–	–	39.46	39.60	
	Detection [Mb/s]	–	–	–	–	2200	2000	N/A	454.26	
	Total [Mb/s]	–	–	–	–	–	–	947	367.88	
Area. Eff.^{b}	Ch. Est. [KEst/s/KG]	240^{a,c}	21.09^{a,c}	–	–	–	–	N/A	10.45	27.34^{a}
	QRD [KQRD/s/KG]	–	–	62.06^{a}	386^{a}	–	–	78.14^{a}	14.35	37.54^{a}
	Detection [Kb/s/KG]	–	–	–	–	4480^{a}	5882^{a}	N/A	165	431^{a}
	Total [Kb/s/KG]	–	–	–	–	–	–	1875^{a}	133	349^{a}
Energy ^{b}	Ch. Est. [nJ/Est]	1.97^{a,c}	6.84^{a,c}	–	–	–	–	N/A	12.70	9.58^{a}
	QRD [nJ/QRD]	–	–	0.53^{a}	2.87^{a}	–	–	N/A	15.27	7.96^{a}
	Detection [nJ/b]	–	–	–	–	0.058^{a}	0.055^{a}	N/A	0.99	0.62^{a}
	Total [nJ/b]	–	–	–	–	–	–	2.07^{a}	1.49	0.83^{a}

[a] With data buffers excluded.
[b] Normalized to 65 nm with 1.2 V core voltage.
[c] Scaled up to 4×4 MIMO configuration: $\{area, power\} \propto d$, where $d = 4/\#\text{Rx-antenna}$.

Area Efficiency

Area efficiency is calculated by normalizing the throughput of each processing task to the corresponding hardware consumption. The proposed solution accomplishes three tasks within the tight timing constraint of the 20 MHz 4×4 MIMO 64-QAM LTE-A downlink, thanks to the algorithm-architecture co-design, which has more than 98% of the total operations mapped onto the vector processor for exploiting extensive DLP and attaining high resource sharing. Compared to other implementations in Table 15, which adopt either lower-dimensions of MIMO configurations or mapping of a single task, the cell array achieves the highest throughput and shows superior area efficiency. Note that [62] and [58] in Table 15 are employed in single-antenna systems, Digital Video Broadcasting (DVB) and Wideband Code Division Multiple Access (WCDMA), respectively. According to [60], the performance required for WCDMA is less than 10% of the one needed for 4×2 MIMO 3GPP LTE delivering 10 Mbps. Thereby, their results cannot be directly compared with those of the baseband processing in MIMO systems. Here, they are included for references only.

Compared to programmable and reconfigurable platforms in Table 16, the processing throughput of the cell array is 2.8 and 45 times higher than that of the FPGA and the DSP solution [122], respectively. Besides, its area efficiency outperforms the GPU and CPU approaches by 3−5 orders of magnitude. It is interesting to note that GPU implementations only achieve a maximum of 66 Mb/s detection throughput, even though they are equipped with enormous parallel computing capacity. For example, the Nvidia Tesla C2070 GPU used in [126] consists of 14 stream multiprocessors, each containing 32 CUDA cores running at 1.15 GHz, and 6 GB of global memory. The low throughput of GPU implementations is mainly caused by the essential difference between wireless baseband processing and graphic computing. GPUs are competent for the latter one. In the wireless communication millions of vectors/matrices with small size have to be handled in parallel, whereas in the graphic computing a large matrix needs to be processed just once in a single application. According to [124], [125], and [126], bottlenecks in their design are the limited register resources and memory access bandwidth for each thread processing. In consequence, it is hard to get even half of the CUDA cores utilized for the mapped processing tasks. These show the importance of architectural customization for intended application domains, although algorithm selections and mapping optimizations may affect the implementation results.

Furthermore, compared to ASIC solutions in Table 17, 1.7−13.6 times of difference in area efficiency is observed for each individual task mapping. The result of [128] reveals a slightly lower efficiency value than that of the

cell array, since it performs data-tone channel estimation in the time-domain by reconstructing the channel impulse response. Compared to the adopted frequency-domain $\boldsymbol{H}$ interpolation that relies on the correlation properties between neighboring subcarriers, the time-domain channel reconstruction leads to better estimation results especially when the cyclic prefix is long. However, this performance gain comes at the cost of higher computational complexity and lower throughput, due to involved time-frequency domain transformations and iterative processing during channel reconstruction.

Energy Efficiency

Besides the area and throughput evaluation, energy consumption per operation is another important measure for baseband processing. In comparison to related implementations in Table 15, similar energy figures are observed. However, it should be mentioned that the cell array operates in a more complicated system setup (4×4 MIMO vs. 2×2 in [120] and [121] and 4×2 in [6]) and has more tasks assigned at the same time. Compared to ASICs, the cell array consumes 1.4−15 times more energy for performing each individual task, whereas a 1.3 and 9 times energy gain is obtained in comparison to the FPGA and the DSP solution, which support only up to 16-QAM detection. Moreover, its energy efficiency outperforms the GPU and CPU approaches by more than 5 orders of magnitude. Such high energy efficiency is achieved mainly by three key hardware developments in the array: the architecture partitioning for attaining efficient vector and scalar processing without frequent data alignments, the vector enhancements in Tile-0 for reducing register accesses, and the flexible memory access schemes in memory cells for relieving most of the non-computational operations from processing cores.

To better visualize the position of the cell array in comparison to other hardware platforms, implementations presented in Table 15-17 are plotted in an area-energy chart, see Figure 39. It clearly shows that the cell array is superior to programmable platforms and achieves an ASIC-like area and energy efficiency. It is worthwhile to re-emphasize here that the proposed solution, contrasts to other works, is capable of performing all three MIMO processing tasks in the target 4×4 MIMO 64-QAM 20 MHz LTE-A system.

Hardware Flexibility

In addition to the efficiency analysis, this section discusses the flexibility of the cell array. Among the three architecture categories, programmable platforms offer the greatest flexibility, while ASICs are designed for specific system setups but reveal the highest hardware efficiency. The baseband processors provide palatable flexibility-efficiency trade-offs between the two aforementioned

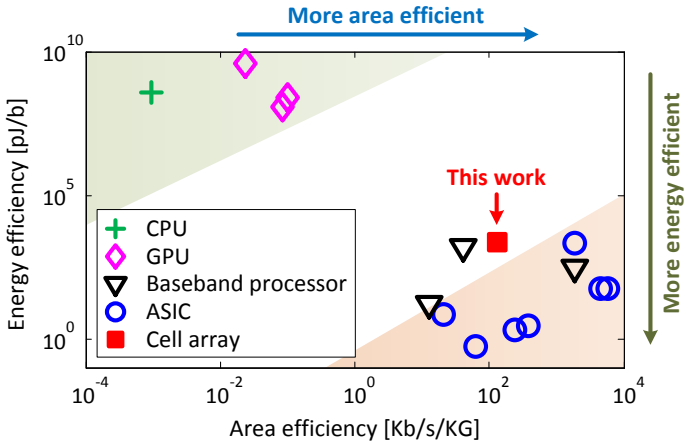


Figure 39: Area and energy efficiency of the cell array in comparison to other hardware platforms.

platforms. On one hand, they offer good flexibility as programmable platforms do, but often require more sophisticated software developments, such as low-level programming in domain-specific languages and manual algorithm mappings [62, 66, 67]. On the other hand, their hardware efficiency is largely improved compared to programmable platforms, thanks to their architecture customization and instruction-level accelerations [61]. In this work, the flexibility is demonstrated by time-multiplexing three different tasks onto one reconfigurable cell array. Additionally, by making use of the dynamic resource allocations, the platform has the potential to support other system configurations, such as processing of different modulation and antenna setups; support of different standards; mapping of different algorithms e.g., non-sorted or iterative-sorted QRD and linear MMSE or SSFE [53] detection; run-time adaption of system performance, e.g., adjusting the frequency of channel estimation and detection parameters.

In the following section, the flexibility of the cell array is further illustrated by mapping a hybrid decomposition scheme for performing channel preprocessing. It is aimed to provide a wide range of performance-complexity trade-offs for coping with constantly changing wireless channels. Briefly, the proposed scheme dynamically switches between the brute-force SQRD (Section 2.2) and a low-complexity group-sort QR-update scheme, based on instantaneous channel condition.

6 Adaptive Channel Pre-processor

For the discussions in the previous sections, the propagation channels were assumed to be quasi-static within one LTE-A time slot (0.5 ms). However, Channel State Information (CSI) of real-world radio channels are rarely constant because of Doppler induced channel changes and multi-path propagation. Outdated CSI introduces additional interferences to the following symbol detection, resulting in drastic degradation of MIMO performance. Thus, frequent CSI update and the corresponding channel pre-processing are highly desirable in wireless communication systems to provide symbol detectors with adequate channel knowledge.

Using the channel's time correlation, tracking of CSI changes can be achieved using low-complexity decision-directed algorithms such as Least Mean Square (LMS), Recursive Least Square (RLS), and Kalman filtering [133, 134]. Nevertheless, continuous CSI tracking has not been widely adopted in practical systems. This is due to the fact that each CSI update requires compute-intensive channel pre-processing, either QRD or channel matrix inversion, which has computational complexity of $\mathcal{O}(N^3)$ for an $N \times N$ MIMO system and consumes more energy than that of symbol detectors (see Table 17). To address the complexity and energy issue, [134] proposed an approximated QRD method to avoid exact tone-by-tone QRD computations during successive channel matrix updates. The presented method is based on the assumption that 1) the orthogonality of column vectors in $\boldsymbol{Q}$ remains unchanged during successive CSI updates; and 2) any change in channel matrix is represented as norm value variations in $\boldsymbol{R}$. Although this tracking-$\boldsymbol{R}$ (hold-$\boldsymbol{Q}$) scheme achieves a substantial complexity reduction, it results in a huge performance loss due to the out-of-date $\boldsymbol{Q}$ information, especially in fast-changing channels. Additionally, channel sorting was not considered in [134].

In this section, an adaptive channel pre-processor using a hybrid decomposition scheme with group-sort QR-update strategy is presented and mapped onto the cell array. By fully exploiting the property of the LTE-A pilot pattern, i.e., CSIs of only antenna port 0 and 1 are changed during half-H renewals (Figure 40), the proposed QR-update scheme computes exact $\boldsymbol{Q}$ and $\boldsymbol{R}$ matrices using only one Givens rotation. Compared to brute-force QRDs, this update strategy significantly reduces the computational complexity, while preserving the accuracy by avoiding the approximations as in the aforementioned tracking algorithms. To obtain the low-complexity benefit of the introduced update scheme in the context of SQRD, an effective group-sort algorithm is proposed for channel reordering. The underlying idea is to restrict the sorting into groups of antenna ports, wherein a two-step (intra- and inter-group) sorting is applied to approximate the optimal detection order. Using the group-sort

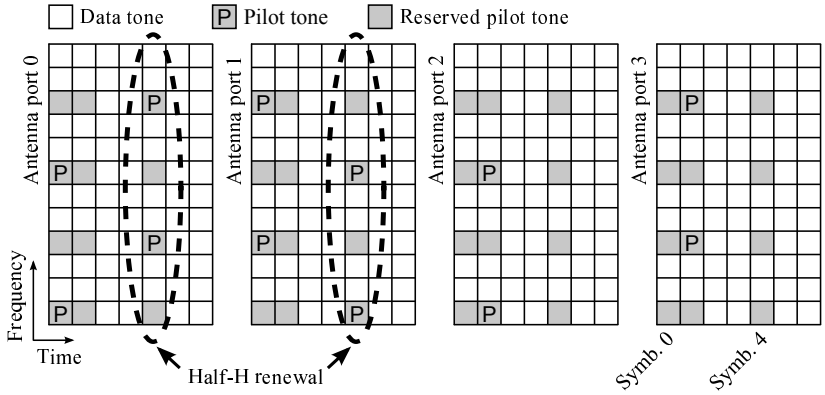


Figure 40: Scattered pilot pattern of LTE-A for four antenna ports. Symbol positions of half-H renewals are circled in dashed lines.

method, applicability of the QR-update is significantly expanded with negligible performance degradation compared to the precise sorting counterpart.

To ease the discussions, the SQRD algorithm is used as a case study in the following sections. However, the presented methods can be applied to the MMSE-SQRD case by working with the augmented matrix $\underline{\hat{\boldsymbol{H}}}$ instead of $\hat{\boldsymbol{H}}$.

6.1 QR-update Scheme

If only parts of the matrix columns alter over time, QRD of the new matrix can be performed in a more efficient way than a brute-force computation (referred to as Case-I), i.e., starting from scratch using Algorithm 1. Inspired by this, a low-complexity QR-update scheme is adopted during half-H renewals. Specifically, the proposed scheme starts with the brute-force SQRD during full-H renewals, expressed with a subscript "old" as

$$\hat{\boldsymbol{H}}_{p,\text{old}} = \boldsymbol{Q}_{\text{old}}\boldsymbol{R}_{\text{old}}, \tag{27}$$

where $\hat{\boldsymbol{H}}_p$ is the permuted channel matrix (5). During half-H renewals, $\hat{\boldsymbol{H}}_{p,\text{new}}$ is obtained by updating two columns of $\hat{\boldsymbol{H}}_{p,\text{old}}$. Although orthogonal vectors in $\boldsymbol{Q}_{\text{old}}$ may no longer triangularize $\hat{\boldsymbol{H}}_{p,\text{new}}$, it may still have vectors pointing in the correct directions. As a consequence, the new $\boldsymbol{R}$ matrix, denoted as $\tilde{\boldsymbol{R}}_{\text{new}}$, can be expressed using $\hat{\boldsymbol{H}}_{p,\text{new}}$ and $\boldsymbol{Q}_{\text{old}}$ as

$$\tilde{\boldsymbol{R}}_{\text{new}} = \boldsymbol{Q}_{\text{old}}^{H}\hat{\boldsymbol{H}}_{p,\text{new}}. \tag{28}$$

Due to the outdated $\boldsymbol{Q}_{\text{old}}$, $\tilde{\boldsymbol{R}}_{\text{new}}$ is no longer an upper-triangular matrix but may still reveal some upper-triangular properties depending on the positions of the two renewed columns. Specifically, if column changes take place at the right-most of $\hat{\boldsymbol{H}}_{p,\text{new}}$, only one element in the lower triangular part of $\tilde{\boldsymbol{R}}_{\text{new}}$ (i.e., $\tilde{r}_{\text{new}(4,3)}$ for 4×4 MIMO) becomes non-zero. This implies that triangularization of $\tilde{\boldsymbol{R}}_{\text{new}}$ can be significantly simplified by nulling the single non-zero element rather than operating on all columns afresh. Defining $\boldsymbol{G}$ as a complex-valued nulling matrix

$$\boldsymbol{G} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & c & s^* \\ 0 & 0 & -s^* & c^* \end{bmatrix}, \tag{29}$$

$\tilde{\boldsymbol{R}}_{\text{new}}$ can be triangularized by computing $\boldsymbol{G}\tilde{\boldsymbol{R}}_{\text{new}}$. In (29), $(\cdot)^*$ denotes the complex conjugate. c and s are defined as

$$\begin{aligned} c &= \tilde{r}^*_{\text{new}(3,3)}/z, \\ s &= \tilde{r}^*_{\text{new}(4,3)}/z, \\ z &= \left(|\tilde{r}_{\text{new}(3,3)}|^2 + |\tilde{r}_{\text{new}(4,3)}|^2\right)^{1/2}. \end{aligned} \tag{30}$$

This nulling process is commonly referred to as Givens rotation [108]. After triangularizing $\tilde{\boldsymbol{R}}_{\text{new}}$, exact $\boldsymbol{Q}_{\text{new}}$ and $\boldsymbol{R}_{\text{new}}$ are obtained, expressed as

$$\begin{aligned} \boldsymbol{Q}_{\text{new}} &= \left(\boldsymbol{G}\boldsymbol{Q}^H_{\text{old}}\right)^H, \\ \boldsymbol{R}_{\text{new}} &= \boldsymbol{G}\tilde{\boldsymbol{R}}_{\text{new}} = \boldsymbol{G}\left(\boldsymbol{Q}^H_{\text{old}}\hat{\boldsymbol{H}}_{p,\text{new}}\right). \end{aligned} \tag{31}$$

It should be pointed out that the lower-right diagonal element of $\boldsymbol{R}_{\text{new}}$ in (31), i.e., $r_{\text{new}(4,4)}$, has been transformed from real to complex-valued domain during the QR updates. This can be easily resolved by performing an additional real-valued domain-transformation using another nulling matrix $\boldsymbol{G}'$ [107], if real-valued diagonal elements are required. The matrix $\boldsymbol{G}'$ is defined as

$$\boldsymbol{G}' = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & c' \end{bmatrix}, \tag{32}$$

where

$$c' = \tilde{r}^*_{\text{new}(4,4)}/|\tilde{r}_{\text{new}(4,4)}|. \tag{33}$$

By combining the traditional brute-force approach and the QR-update scheme, a hybrid decomposition algorithm is formed. Depending on run-time conditions of the channel reordering, the two schemes can be dynamically switched to reduce the computational complexity. Obviously, the complexity reduction depends on the applicability of the QR-update. Intuitively, the position of antenna port 0 and 1 can be fixed to the right-most part of $\hat{\boldsymbol{H}}_{p,\text{new}}$ in order to obtain a maximum complexity gain, since it completely avoids brute-force computation during half-H renewals. However, the advantage of channel reordering, for improving detection performance, is lost. This approach is referred to as Case-II. On the other hand (Case-III), the applicability of the QR-update is dramatically reduced if channel columns are permuted based on the optimal detection order without considering the position of renewed channel columns. For example, considering the 4×4 MIMO LTE-A, only $(2!2!)/4! = 1/6$ of sorting combinations meet the required update condition, thus limiting the complexity reduction. As a consequence, a smart scheduling strategy is needed to explore the low-complexity potential of the QR-update, while still retaining the performance gain of the optimal channel reordering.

6.2 Group-sort Algorithm

To fulfil the aforementioned requirement, an effective group-sort algorithm is proposed for channel reordering. Instead of operating on individual columns, sorting of $\hat{\boldsymbol{H}}$ is applied on two virtual groups, wherein columns associated with antenna ports 0 and 1 are tied together. This way, the number of combinations of "columns" is reduced from 4! to 2!. Consequently, the probability of having both altered columns at the right-most part of $\hat{\boldsymbol{H}}_{p,\text{new}}$ is increased by 3 times, from 1/6 to 1/2. To reduce errors caused by sub-optimal sorting sequences, a two-step sorting scheme is adopted. First, the sorting between groups is based on the total energy of bundled columns as

$$\mathcal{I} = \arg\max_{i=\{0,1\},\{2,3\}} \sum\nolimits_i \|\hat{\boldsymbol{h}}_{p(i)}\|^2, \tag{34}$$

where $\mathcal{I}$ contains inter-sorted group indexes, e.g., $\mathcal{I} = \{0, 1\}$ if antenna ports 0 and 1 correspond to the strongest channels. Second, the two columns within each group, e.g., indexes within $\mathcal{I}$, are intra-sorted based on the energy of individual columns. To summarize, Table 18 lists applicability of all the four cases of the hybrid decomposition algorithm. The proposed group-sort method is denoted by Case-IV.

6.3 Algorithm Evaluation and Operation Analysis

To illustrate the effectiveness of the proposed algorithm, the same simulation setup as the one presented in Section 3.1 with parameters in Table 1 is em-

Table 18: Case-I–IV of the proposed hybrid decomposition algorithm

	Channel reordering	Brute-force	QR-update
Case-I	Optimal ordering with precise sorting	100%	0%
Case-II	Fixed order for antenna port 0 and 1	0%	100%
Case-III	Optimal ordering with precise sorting	83.33%	16.67%
Case-IV	Group-sort	50%	50%

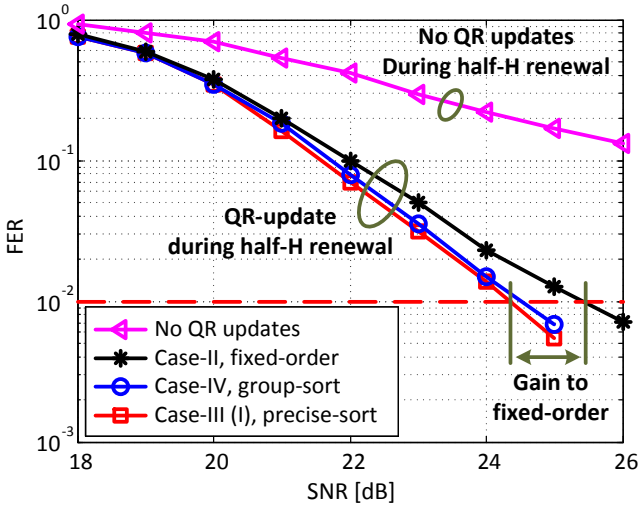


Figure 41: Simulated FERs in a 4×4 MIMO LTE-A downlink using 3GPP EVA-70 channel model with 64-QAM modulation.

ployed, but the variant channel modelling is used in this case for emulating time-variations of radio channels. At 2.6 GHz carrier frequency, the maximum Doppler frequency of 70 Hz corresponds to a speed of 29 Km/hour. To minimize performance influences from channel estimation and symbol detection, perfect channel knowledge is assumed at the receiver and the near-ML FSD algorithm is used for detecting symbols.

Performance of the proposed group-sort QR-update and aforementioned cases are shown in Figure 41. Note that Case-III has the same performance as the brute-force approach and is used as a reference for FER measurements. The difference in performance is remarkable between Case-III and the case where no QRDs are performed during half-H renewals (the upper curve in Figure 41), indicating the importance of performing CSI and QR updates even for channels

Table 19: Complexity of computations in the hybrid decomposition algorithm

Complexity	Computation	Multiplication	$1/\sqrt{x}$
$\mathcal{C}_1$	QRD (27)	$N^3 + 2N^2$	N
$\mathcal{C}_2$	$\boldsymbol{Q}_{\text{old}}^H \hat{\boldsymbol{H}}_{p,\text{new}}$ (28)	$\frac{1}{2}N^3$	0
$\mathcal{C}_3$	Triangularization (29)−(31)	$N^2 + 2N$	1
$\mathcal{C}_4$	Sorting (Algorithm 1 line 3 and 12)	$6N$	0
$\mathcal{C}_5$	Sorting (34)	$4N$	0

Table 20: Complexity and performance comparisons of Case-I−IV

	Complexity	Complexity reduction	SNR degradation
Case-I	$\mathcal{C}_1 + \mathcal{C}_4$	− (ref.)	− (ref.)
Case-II	$\mathcal{C}_2 + \mathcal{C}_3$	53%	1.1 dB
Case-III	$\frac{5}{6}\mathcal{C}_1 + \frac{1}{6}(\mathcal{C}_2 + \mathcal{C}_3) + \mathcal{C}_4$	6%	0 dB
Case-IV	$\frac{1}{2}\mathcal{C}_1 + \frac{1}{2}(\mathcal{C}_2 + \mathcal{C}_3) + \mathcal{C}_5$	23%	0.2 dB

with moderate Doppler shifts. Additionally, adoption of channel reordering during QR decomposition improves on the fixed-order approach in performance, e.g., 1.1 dB difference between Case-II and III at FER=10^{-2}. Furthermore, the group-sort approach has only small performance degradation of about 0.2 dB compared to Case-III, however, with a large complexity reduction as analyzed in the following.

Table 19 summaries complexity ($\mathcal{C}$) of computations (27)−(31) and (34) for an $N \times N$ MIMO system. To perform the brute-force decomposition (27), MGS algorithm (Algorithm 1) is considered which has a complexity of $\mathcal{C}_1$. Computations required for both (28) and (31) have a total complexity of $\mathcal{C}_2$+$\mathcal{C}_3$, which is significantly lower than $\mathcal{C}_1$, e.g., by about 42% for $N = 4$. Note that the product of $\boldsymbol{Q}_{\text{old}}^H \hat{\boldsymbol{H}}_{p,\text{new}}$ in (28) requires only half of the matrix computations during QR updates, since only two columns change in $\hat{\boldsymbol{H}}_{p,\text{new}}$. The complexity of the precise-iterative-sort and the group-sort approach is denoted as $\mathcal{C}_4$ and $\mathcal{C}_5$, respectively. Based on this analysis and in reference to Case-I, Table 20 shows the complexity reduction versus performance degradation of Case-II−IV for a 4×4 MIMO system. It shows that Case-II reduces the computational complexity by 53%. Additionally, combining the group-sort and the QR-update scheme results in more palatable trade-offs, i.e., 23% complexity reduction for only 0.2 dB performance degradation.

Table 21: Operation profile of the hybrid decomposition algorithm for $N = 4$

	Vector operations			$1/\sqrt{x}$
	$\boldsymbol{A} \cdot \boldsymbol{B}$	$\boldsymbol{A} \bigodot \boldsymbol{B}$	$\boldsymbol{A} \pm \boldsymbol{B}$	
QRD (27)	17	4	6	4
$\boldsymbol{Q}_{\text{old}}^{H} \hat{\boldsymbol{H}}_{p,\text{new}}$ (28)	8	0	0	0
Triangularization (29)−(31)	10	0	0	1
Sorting (34)	4	0	0	0

To further evaluate the hardware friendliness of the proposed algorithm and the possibility of being mapped onto the cell array, operations required in the four computations (Table 19) are profiled, see Table 21. It clearly shows that all operations required in the group-sort QR-update algorithm are shared with that of the brute-force method. This implies that extensive hardware reuse is possible. Additionally, over 95% of the operations are at vector level, representing a high degree of DLP that can be exploited to achieve high processing throughput.

6.4 Implementation Results and Discussion

It is straightforward to implement the adaptive channel pre-processor onto the cell array using the hybrid decomposition scheme, Case-I−IV in Table 18, since all the required operations (Table 21) have already been mapped for the SQRD computation. Similar to the task mapping scheme presented in the previous section, multi-subcarrier processing is adopted for attaining high throughput and is scheduled based on the LTE-A resource block. Worth mentioning is that Givens rotation (31) can be implemented in different ways, such as using conventional arithmetic or through a series of CORDIC operations [107]. In Part II, the CORDIC algorithm has been mapped onto the scalar processing cell for computing the phase value of received symbols. However, each computation required several clock cycle to complete because of the iterative nature of the CORDIC. Evidently, adopting the CORDIC approach in the QR-update computation would make the triangularization process an implementation bottleneck. Hence, the Givens rotation is realized by using conventional arithmetics, where c and s in (30) are computed by the dedicated inverse square root unit available in Tile-3.

Table 22 summaries implementation results for the brute-force and the QR-update computations. Operating at 500 MHz, processing throughput of the QR-update is 2.6 times higher than that of the brute-force approach. Moreover,

Table 22: Performance summary of the hybrid decomposition scheme.

	Clock Cycle/Op	Throughput [MQRD/s]	Energy[a] [nJ/QRD]
Brute-force SQRD	12.63	39.60	7.96
Group-sort QR-update	4.83	103.45	4.29

[a] With data buffers excluded.

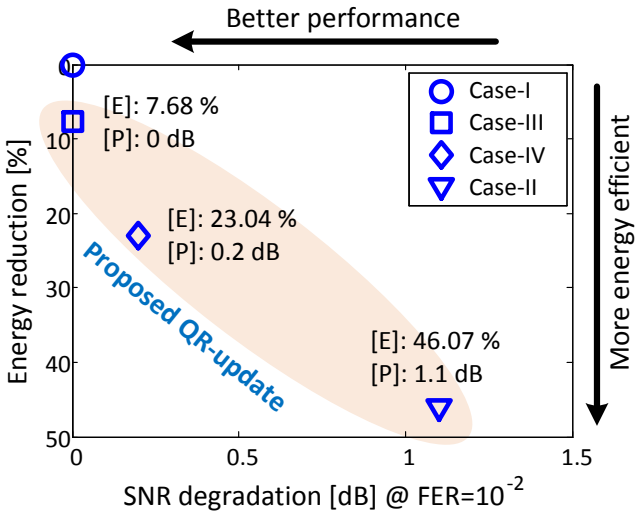


Figure 42: Energy (E) and performance (P) trade-off for Case-I–IV of the hybrid decomposition scheme.

it reduces the energy consumption by 1.9 times. It should be pointed out that further energy reduction is possible if fine-grained low power design techniques are employed. For example, the computation of $\boldsymbol{Q}_{\text{new}}$ (31) can be performed more efficiently if half of the SIMD core in Tile-0 could be clock gated, because of the zero elements in the nulling matrix $\boldsymbol{G}$.

Figure 42 presents design trade-offs between energy and performance for Case-I–IV of the hybrid decomposition scheme. Taking the brute-force QRD (Case-I) as a reference, numbers on the horizontal axis measures the SNR degradation for reaching the target 10^{-2} FER, while the percentage of energy reduction is shown on the vertical axis. Accordingly, algorithms having their coordinates towards the bottom-left corner are desired. Figure 42 clearly shows that the proposed group-sort QR-update scheme (Case-IV) achieves a good

compromise, i.e., trading 0.2 dB performance for 23% energy reduction. In the case of energy-constrained systems, the fixed-order scheme (Case-II) can be adopted to further reduce the energy consumption, i.e., by 46% in total, whereas the precise-sort scheme (Case-III) can be used if high performance is demanded. Benefiting from the flexibility of the cell array, the selection of the algorithm can be made at system run-time, depending on instantaneous channel condition, performance requirement, and power budget.

7 Summary

This part presents a reconfigurable baseband processor designed based on the heterogeneous cell array architecture. The performance and flexibility of the proposed solution is exhibited by mapping three crucial baseband processing blocks onto the processor, while the capability of real-time processing in an LTE-A downlink is demonstrated. Such high processing performance is achieved by algorithm-architecture co-design. On the algorithm side, more than 98% of total operations in all three tasks are unified to a vector-level, enabling extensive parallel processing and resource sharing for attaining high hardware efficiency. Further achievements in area and energy efficiency are enabled by architectural developments, including the heterogeneous resource deployments, vector enhancements of the processing core, and flexible self-governed memory data access schemes. Implementation results show that the proposed processor bridges the gap between conventional platforms. The processor provides enormous design flexibility and scalability like programmable platforms, while approaching the area and energy efficiency of task specific ASIC solutions. In addition to the multi-task MIMO processing, the flexibility of the cell array is demonstrated by mapping an adaptive channel pre-processor. Taking advantage of dynamic resource allocations, a wide range of performance-complexity trade-offs are provided, so that an appropriate pre-processing algorithm can be selected at run-time based on instantaneous channel condition.

Conclusion and Outlook

Coarse-Grained Reconfigurable Architectures (CGRAs) emerge as a new class of hardware platforms, designed to bridge the gap of computational performance, hardware efficiency and flexibility among conventional architectures such as Application-Specific Integrated Circuits (ASICs), Field-Programmable Gate Arrays (FPGAs), and Digital Signal Processors (DSPs). The strength of CGRAs lies in the capability of allocating hardware resources dynamically to accomplish current computational demands. In addition, hardware efficiency with respect to area and power consumption is substantially improved in comparison to FPGAs, thanks to the word-level data manipulations.

In this thesis, a dynamically reconfigurable cell array architecture is proposed, developed, and verified in silicon with a primary focus on digital baseband processing in wireless communication. The proposed cell array architecture is constructed from an array of processing and memory cells interconnected through a hierarchical easily-scalable on-chip network. High hardware efficiency is attained by conducting algorithm-architecture, hardware-software, and processing-memory co-design. The performance and flexibility of the cell array are demonstrated through two case studies.

In the first study, the cell array is designed to process multiple radio standards concurrently, aiming to demonstrate the flexibility of the architecture and evaluate the control overhead, in terms of clock cycles and area consumption, of hardware reconfigurations. With a 2×2 cell array, three contemporary wireless communication standards are supported and two independent data streams from any of the three standards can be processed concurrently. Depending on the number of receiving data streams, the cell array dynamically adjusts its underlying hardware resources to maximize hardware usage for achieving either high computational accuracy or processing concurrence. In addition to resource sharing among multiple radio standards, hardware flexibility is demonstrated by mapping a different algorithm onto the same platform after chip fabrication. The adoption of the new algorithm extends the coverage of standards to be supported. Thanks to the employed in-cell configuration scheme, run-time context switching between different operation scenarios requires at most 11 clock cycles, which correspond to the configuration time of ∼34 nS at 320 MHz. Implementation results show that the adoption of the cell array in a digital front-end receiver requires only about 16% area overhead in comparison to its ASIC counterpart.

The second study deals with multi-task processing, aiming to demonstrate the flexibility and real-time processing capability of the cell array as well as to

evaluate the area and energy efficiency. To this end, three crucial and compute-intensive baseband processing blocks in 4×4 MIMO-OFDM systems, namely channel estimation, pre-processing, and symbol detection, are mapped onto the cell array. Benefiting from algorithm-architecture co-design, the cell array is capable of processing all the target tasks in real-time for a 20 MHz 64-QAM 3GPP LTE-A downlink. On the the algorithm side, most of the operations are unified to vector-level, which enables extensive parallel processing and resource sharing for attaining high hardware efficiency. On the architecture side, the cell array is extended with extensive vector computing capabilities, including vector-enhanced SIMD cores and VLIW-style multi-stage computation chain. This is done in order to achieve low-latency high-throughput vector computing and reduce register/memory access for loading and storing intermediate results. In addition, flexible memory access schemes are adopted to relieve processing cores from non-computational address manipulations. Implementation results show that the proposed cell array outperforms related programmable platforms by up to 6 orders of magnitude in energy efficiency, and is 1.7−13.6 and 1.4−15 times less efficient than ASICs in terms of area and energy.

In conclusion, the CGRA-based cell array demonstrates a good design trade-off between the contradictory requirements of flexibility, performance, and hardware efficiency. Thus, it is a promising and feasible solution to bridge the huge gap between conventional platforms. Looking forward, adopting the cell array in a wide range of applications is a natural continuation. However, this requires a series of system-level exploration tools to model, simulate and evaluate the use of the platform as well as application mapping tools to automate task profiling, scheduling, mapping, and compilation process. Despite the system-level developments to be explored, the future of the reconfigurable cell array is certainly bright.

Bibliography

[1] A. Nilsson, "Design of Programmable Multi-Standard Baseband Processors," Ph.D. dissertation, Department of Electrical Engineering, Linköping University, 2007.

[2] M. Dillinger, K. Madani, and N. Alonistioti, *Software Defined Radio: Architectures, Systems and Functions*, 1st ed. Wiley, June 2003.

[3] Ericsson, "White Paper: More Than 50 Billion Connected Devices - Taking Connected Devices to Mass Market and Profitability," Feb. 2011. [Online]. Available: http://www.ericsson.com/res/docs/whitepapers/wp-50-billions.pdf

[4] Broadcom, "Facts at a Glance," Apr. 2014. [Online]. Available: https://www.broadcom.com/docs/company/BroadcomQuickFacts.pdf

[5] E. Dahlman, S. Parkvall, and J. Skold, *4G: LTE/LTE-Advanced for Mobile Broadband*, 1st ed. Academic Press, May 2011.

[6] F. Clermidy *et al.*, "A 477mW NoC-Based Digital Baseband for MIMO 4G SDR," in *IEEE International Solid-State Circuits Conference (ISSCC)*, Feb. 2010, pp. 278–279.

[7] T. Lenart, "Design of Reconfigurable Hardware Architectures for Real-time Applications," Ph.D. dissertation, Department of Electrical and Information Technology, Lund University, May 2008.

[8] H. Svensson, "Reconfigurable Architectures for Embedded Systems," Ph.D. dissertation, Department of Electrical and Information Technology, Lund University, Oct. 2008.

[9] S. Borkar, "Thousand Core Chips - A Technology Perspective," in *44th Annual Design Automation Conference (DAC)*, 2007, pp. 746–749.

[10] M. B. Taylor, "A Landscape of the New Dark Silicon Design Regime," *IEEE Micro*, vol. 33, no. 5, pp. 8–19, Sep. 2013.

[11] M. Woh, S. Mahlke, T. Mudge, and C. Chakrabarti, "Mobile Supercomputers for the Next-Generation Cell Phone," *IEEE Computer*, vol. 43, no. 1, pp. 81–85, Jan. 2010.

[12] G. Miao, N. Himayat, Y. Li, and A. Swami, "Cross-layer Optimization for Energy-efficient Wireless Communications: A Survey," *Wireless Communications & Mobile Computing*, vol. 9, no. 4, pp. 529–542, Apr. 2009.

[13] G. Estrin, "Organization of Computer Systems: The Fixed Plus Variable Structure Computer," in *Western Joint IRE-AIEE-ACM Computer Conference*, May 1960, pp. 33–40.

[14] C. C. Wang, F. L. Yuan, H. Chen, and D. Marković, "A 1.1 GOPS/mW FPGA Chip with Hierarchical Interconnect Fabric," in *IEEE Symposium on VLSI Circuits (VLSIC)*, Jun. 2011, pp. 136–137.

[15] I. Kuon, R. Tessier, and J. Rose, "FPGA Architecture: Survey and Challenges," *Foundations and Trends in Electronic Design Automation*, vol. 2, no. 2, pp. 135–253, Feb. 2008.

[16] R. Hartenstein, "A Decade of Reconfigurable Computing: a Visionary Retrospective," in *Design, Automation Test in Europe Conference Exhibition (DATE)*, 2001, pp. 642–649.

[17] R. Tessier and W. Burleson, "Reconfigurable Computing for Digital Signal Processing: A Survey," *Journal of VLSI Signal Processing Systems*, vol. 28, pp. 7–27, May 2001.

[18] K. Compton and S. Hauck, "Reconfigurable Computing: A Survey of Systems and Software," *ACM Computing Survey*, vol. 34, pp. 171–210, June 2002.

[19] T. J. Todman, G. A. Constantinides, S. J. E. Wilton, O. Mencer, W. Luk, and P. Y. K. Cheung, "Reconfigurable Computing: Architectures and Design Methods," in *Computers and Digital Techniques*, vol. 152, no. 2, Mar. 2005, pp. 193–207.

[20] Z. Abdin and B. Svensson, "Evolution in Architectures and Programming Methodologies of Coarse-Grained Reconfigurable Computing," *Microprocessors and Microsystems: Embedded Hardware Design*, vol. 33, pp. 161–178, May 2009.

[21] A. Chattopadhyay, "Ingredients of Adaptability: A Survey of Reconfigurable Processors," *VLSI Design*, vol. 2013, Jan. 2013.

[22] G. E. Moore, "Cramming More Components onto Integrated Circuits," *Electronics*, vol. 38, no. 8, pp. 114–117, Apr. 1965.

[23] R. W. Keyes, "The Impact of Moore's Law," *IEEE Solid-State Circuits Society Newsletter*, vol. 11, no. 5, pp. 25–27, Sep. 2006.

[24] J. L. Hennessy and D. A. Patterson, *Computer architecture: A Quantitative Approach*, 4th ed. Morgan Kaufmann Publishers, 2003.

[25] D. Marr *et al.*, "Hyper-threading Technology Architecture and Microarchitecture: a Hypertext History," *Intel Technical Journal*, vol. 1, no. 1, Feb. 2002.

[26] S. H. Fuller and L. I. Millett, "Computing Performance: Game Over or Next Level?" *Computer*, vol. 44, no. 1, pp. 31–38, Jan. 2011.

[27] K. K. Parhi, *VLSI Digital Signal Processing Systems: Design and Implementation*, 1st ed. Wiley, 1999.

[28] NVIDIA, "Tesla C2050/C2070 GPU Computing Processor," Jul. 2010. [Online]. Available: http://www.nvidia.com/docs/IO/43395/NV_DS_Tesla_C2050_C2070_jul10_lores.pdf

[29] M. Garland *et al.*, "Parallel Computing Experiences with CUDA," *IEEE Micro*, vol. 28, no. 4, pp. 13–27, Jul. 2008.

[30] NVIDIA, "MATLAB Acceleration on NVIDIA Tesla and Quadro GPUs," 2014. [Online]. Available: http://www.nvidia.com/object/tesla-matlab-accelerations.html

[31] D. Liu, *Embedded DSP Processor Design: Application Specific Instruction Set Processors*, 1st ed. Morgan Kaufmann, May 2008.

[32] J. Byrne, "Tensilica DSP Targets LTE Advanced," Mar. 2011. [Online]. Available: http://www.tensilica.com/uploads/pdf/MPR_BBE64.pdf

[33] J. Rabaey, *Low Power Design Essentials*, 1st ed. Springer, May 2009.

[34] A. Chandrakasan, S. Sheng, and R. Brodersen, "Low-Power CMOS Digital Design," *IEEE Journal of Solid-State Circuits*, vol. 27, no. 4, pp. 473–484, Apr. 1992.

[35] United Nations Foundation and The Vodafone Foundation, "mHealth for Development: The Opportunity of Mobile Technology for Healthcare in the Developing World," 2011. [Online]. Available: http://www.vitalwaveconsulting.com/pdf/2011/mHealth.pdf

[36] P. J. and M. Salehi, *Digital Communications*, 5th ed. McGraw-Hill Science, Nov. 2007.

[37] A. F. Molisch, *Wireless Communications*, 2nd ed. Wiley, Dec. 2010.

[38] R. W. Chang, "Synthesis of Band-Limited Orthogonal Signals for Multichannel Data Transmission," *Bell System Technical Journal*, vol. 45, no. 10, pp. 1775–1796, 1966.

[39] A. Paulraj, R. Nabar, and D. Gore, *Introduction to Space-Time Wireless Communications*, 1st ed. Cambridge University Press, June 2008.

[40] C. Yuen and B. Hochwald, "Achieving Near-Capacity at Low SNR on a Multiple-Antenna Multiple-User Channel," *IEEE Transactions on Communications*, vol. 57, no. 1, pp. 69–74, Jan. 2009.

[41] C. Spiegel, J. Berkmann, Z. Bai, T. Scholand, and C. Drewes, "MIMO Schemes in UTRA LTE, A Comparison," in *IEEE Vehicular Technology Conference (VTC)*, May 2008, pp. 2228–2232.

[42] G. Bauch and G. Dietl, "Multi-User MIMO for Achieving IMT-Advanced Requirements," in *International Conference on Telecommunications (ICT)*, June 2008, pp. 1–7.

[43] L. Liu, J. Löfgren, and P. Nilsson, "Area-Efficient Configurable High-Throughput Signal Detector Supporting Multiple MIMO Modes," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 59, no. 9, pp. 2085–2096, Sep. 2012.

[44] E. Tell, "Design of Programmable Baseband Processors," Ph.D. dissertation, Department of Electrical Engineering, Linköping University, 2005.

[45] R. Gallager, "Low-Density Parity-Check Codes," *IRE Transactions on Information Theory*, vol. 8, no. 1, pp. 21–28, Jan. 1962.

[46] A. Viterbi, "Error Bounds for Convolutional Codes and an Asymptotically Optimum Decoding Algorithm," *IEEE Transactions on Information Theory*, vol. 13, no. 2, pp. 260–269, Apr. 1967.

[47] C. Berrou and A. Glavieux, "Near Optimum Error Correcting Coding and Decoding: Turbo-Codes," *IEEE Transactions on Communications*, vol. 44, no. 10, pp. 1261–1271, Oct 1996.

[48] F. Horlin and A. Bourdoux, *Digital Compensation for Analog Front-Ends: A New Approach to Wireless Transceiver Design*, 1st ed. Wiley, June 2008.

[49] I. Diaz, "Algorithm-Architecture Co-Design for Digital Front-Ends in Mobile Receivers," Ph.D. dissertation, Department of Electrical and Information Technology, Lund University, 2014.

[50] A. Burg *et al.*, "VLSI Implementation of MIMO Detection Using the Sphere Decoding Algorithm," *IEEE Journal of Solid-State Circuits (JSSC)*, vol. 40, no. 7, pp. 1566–1577, July 2005.

[51] Z. Guo and P. Nilsson, "Algorithm and Implementation of the K-best Sphere Decoding for MIMO Detection," *IEEE Journal on Selected Areas in Communications*, vol. 24, no. 3, pp. 491–503, Mar. 2006.

[52] L. G. Barbero and J. S. Thompson, "Fixing the Complexity of the Sphere Decoder for MIMO Detection," *IEEE Transactions on Wireless Communications*, vol. 7, no. 6, pp. 2131–2142, June 2008.

[53] M. Li *et al.*, "Optimizing Near-ML MIMO Detector for SDR Baseband on Parallel Programmable Architectures," in *Design, Automation and Test in Europe (DATE)*, Mar. 2008, pp. 444–449.

[54] C. Yang and D. Marković, "A Flexible DSP Architecture for MIMO Sphere Decoding," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 56, no. 10, pp. 2301–2314, Oct. 2009.

[55] M. Shami and A. Hemani, "Classification of Massively Parallel Computer Architectures," in *IEEE 26th International Parallel and Distributed Processing Symposium Workshops PhD Forum (IPDPSW)*, May 2012, pp. 344–351.

[56] R. Baines and D. Pulley, "A Total Cost Approach to Evaluating Different Reconfigurable Architectures for Baseband Processing in Wireless Receivers," *IEEE Communications Magazine*, vol. 41, no. 1, pp. 105–113, Jan. 2003.

[57] Y. Lin *et al.*, "SODA: A Low-power Architecture For Software Radio," in *International Symposium on Computer Architecture (ISCA)*, 2006, pp. 89–101.

[58] H. Lee, C. Chakrabarti, and T. Mudge, "A Low-Power DSP for Wireless Communications," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 18, no. 9, pp. 1310–1322, 2010.

[59] R. Airoldi, F. Garzia, O. Anjum, and J. Nurmi, "Homogeneous MPSoC as Baseband Signal Processing Engine for OFDM Systems," in *International Symposium on System on Chip (SoC)*, Sep. 2010, pp. 26–30.

[60] C. Bernard and F. Clermidy, "A Low-Power VLIW Processor for 3GPP-LTE Complex Numbers Processing," in *Design, Automation Test in Europe Conference Exhibition (DATE)*, Mar. 2011, pp. 1–6.

[61] R. Fasthuber *et al.*, "Exploration of Soft-Output MIMO Detector Implementations on Massive Parallel Processors," *Journal of Signal Processing Systems*, vol. 64, pp. 75–92, 2011.

[62] A. Nilsson, E. Tell, and D. Liu, "An 11 mm^2, 70 mW Fully Programmable Baseband Processor for Mobile WiMAX and DVB-T/H in 0.12μm CMOS," *IEEE Journal of Solid-State Circuits (JSSC)*, vol. 44, no. 1, pp. 90–97, Jan. 2009.

[63] B. Bougard, B. De Sutter, D. Verkest, L. Van der Perre, and R. Lauwereins, "A Coarse-Grained Array Accelerator for Software-Defined Radio Baseband Processing," *IEEE Micro*, vol. 28, no. 4, pp. 41–50, July 2008.

[64] K. van Berkel, F. Heinle, P. P. E. Meuwissen, K. Moerman, and M. Weiss, "Vector Processing As an Enabler for Software-defined Radio in Handheld Devices," *EURASIP Journal on Applied Signal Processing*, vol. 2005, pp. 2613–2625, Jan. 2005.

[65] V. Baumgarte, G. Ehlers, F. May, A. Nückel, M. Vorbach, and M. Weinhardt, "PACT XPP-A Self-Reconfigurable Data Processing Architecture," *The Journal of Supercomputing*, vol. 26, pp. 167–184, Sep. 2003.

[66] M. Thuresson *et al.*, "FlexCore: Utilizing Exposed Datapath Control for Efficient Computing," *Journal of Signal Processing Systems*, vol. 57, no. 1, pp. 5–19, 2009.

[67] J. Janhunen, T. Pitkanen, O. Silven, and M. Juntti, "Fixed- and Floating-Point Processor Comparison for MIMO-OFDM Detector," *IEEE Journal of Selected Topics in Signal Processing*, vol. 5, no. 8, pp. 1588–1598, 2011.

[68] M. Shami and A. Hemani, "Morphable DPU: Smart and Efficient Data Path for Signal Processing Applications," in *IEEE Workshop on Signal Processing Systems (SiPS)*, Oct. 2009, pp. 167–172.

[69] B. Plunkett and J. Watson, *Adapt2400 ACM Architecture Overview*, Quicksilver, 2004, a Technology White paper. [Online]. Available: http://vada.skku.ac.kr/ClassInfo/system_level_design/sdr_slides/Adapt2400_Whitepaper_0404.pdf

[70] J. Eker and J. W. Janneck, "CAL Language Report: Specification of the CAL Actor Language," University of California at Berkeley, Tech. Rep., Nov. 2003. [Online]. Available: http://ptolemy.eecs.berkeley.edu/papers/03/Cal/

[71] G. Kahn, "The Semantics of a Simple Language for Parallel Programming," in *Information Processing*. North-Holland Publishing Company, 1974, pp. 471–475.

[72] R. S. Patti, "Three-Dimensional Integrated Circuits and the Future of System-on-Chip Designs," *Proceedings of the IEEE*, vol. 94, no. 6, pp. 1214–1224, June 2006.

[73] C. Zhang, T. Lenart, H. Svensson, and V. Öwall, "Design of Coarse-Grained Dynamically Reconfigurable Architecture for DSP Applications," in *International Conference on Reconfigurable Computing and FPGAs (ReConFig)*, Dec 2009, pp. 338–343.

[74] M. B. Taylor *et al.*, "A 16-Issue Multiple-Program-Counter Microprocessor with Point-to-Point Scalar Operand Network," in *IEEE International Solid-State Circuits Conference*, Feb. 2003, pp. 170–171 vol.1.

[75] Z. Yu and B. M. Baas, "A Low-Area Multi-Link Interconnect Architecture for GALS Chip Multiprocessors," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 18, no. 5, pp. 750–762, 2010.

[76] A. Rahimi, I. Loi, M. R. Kakoee, and L. Benini, "A Fully-Synthesizable Single-Cycle Interconnection Network for Shared-L1 Processor Clusters," in *Design, Automation Test in Europe Conference Exhibition (DATE)*, 2011, pp. 1–6.

[77] M. Coppola, R. Locatelli, G. Maruccia, L. Pieralisi, and A. Scandurra, "Spidergon: A Novel on-chip Communication Network," in *International Symposium on System-on-Chip*, 2004, p. 15.

[78] T. Bjerregaard and S. Mahadevan, "A Survey of Research and Practices of Network-on-Chip," *ACM Computing Surveys*, vol. 38, no. 1, June 2006.

[79] A. Y. Dogan, J. Constantin, M. Ruggiero, A. Burg, and D. Atienza, "Multi-core Architecture Design for Ultra-Low-Power Wearable Health Monitoring Systems," in *Design, Automation Test in Europe Conference Exhibition (DATE)*, 2012, pp. 988–993.

[80] "AMBA 4 AXI4-Stream Protocol Specification v1.0," Mar. 2010. [Online]. Available: http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.ihi0051a/index.html

[81] H. Svensson, T. Lenart, and V. Öwall, "Modelling and Exploration of a Reconfigurable Array Using SystemC TLM," in *IEEE International Symposium on Parallel and Distributed Processing*, Apr. 2008, pp. 1–8.

[82] X. Yang, "IEEE 802.11n: Enhancements for Higher Throughput in Wireless LANs," *IEEE Wireless Communications*, vol. 12, no. 6, pp. 82–91, Dec. 2005.

[83] U. H. Reimers, "DVB-The Family of International Standards for Digital Video Broadcasting," *Proceedings of the IEEE*, vol. 94, no. 1, pp. 173–182, Jan. 2006.

[84] C. Ebeling, C. Fisher, G. Xing, M. Shen, and H. Liu, "Implementing an OFDM Receiver on the RaPiD Reconfigurable Architecture," *IEEE Transactions on Computers*, vol. 53, no. 11, pp. 1436–1448, Nov. 2004.

[85] A. Baschirotto *et al.*, "Baseband Analog Front-end and Digital Back-end for Reconfigurable Multi-standard Terminals," *IEEE Circuits and Systems Magazine*, vol. 6, no. 1, pp. 8–28, Jan. 2006.

[86] "Multi-Base−Scalable Multi-tasking Baseband for Mobile Communications ," Feb. 2008. [Online]. Available: ftp://ftp.cordis.europa.eu/pub/fp7/ict/docs/future-networks/projects-multibase-factsheet-20080206_en.pdf

[87] L. v. d. Perre *et al.*, "D4.2 Identification and Architectural Development of Key Building Blocks," Multibase Consortium, Tech. Rep., Jan. 2009.

[88] C. W. Farrow, "A Continuously Variable Digital Delay Element," in *IEEE International Symposium on Circuits and Systems*, Jun. 1988, pp. 2641–2645 vol.3.

[89] M. Speth, S. Fechtel, G. Fock, and H. Meyr, "Optimum Receiver Design for OFDM-Based Broadband Transmission – Part II: A Case Study," *IEEE Transactions on Communications*, vol. 49, no. 4, pp. 571–578, Apr. 2001.

[90] J. J. van de Beek, M. Sandell, and P. O. Borjesson, "ML Estimation of Time and Frequency Offset in OFDM Systems," *IEEE Transactions on Signal Processing*, vol. 45, no. 7, pp. 1800–1805, Jul. 1997.

[91] IEEE, "IEEE P802.11N/D2.00," IEEE LAN/MAN Standards Committee, Tech. Rep., Feb. 2007.

[92] B. Parhami, *Computer Arithmetic: Algorithm and Hardware Designs*. Oxford University Press, 2000.

[93] I. Diaz, L. Wilhelmsson, J. Rodrigues, J. Lofgren, T. Olsson, and V. Öwall, "A Sign-bit Auto-correlation Architecture for Fractional Frequency Offset Estimation in OFDM," in *IEEE International Symposium on Circuits and Systems (ISCAS)*, May 2010, pp. 3765–3768.

[94] L. Wilhelmsson, I. Diaz, T. Olsson, and V. Öwall, "Performance Analysis of Sign-Based Pre-FFT Synchronization in OFDM Systems," in *IEEE 71st Vehicular Technology Conference*, May 2010, pp. 1–5.

[95] J. W. Cooley and J. W. Tukey, "An Algorithm for the Machine Calculation of Complex Fourier Series," *IEEE Journal of Solid-State Circuits*, vol. 19, no. 90, pp. 297–301, 1965.

[96] Xilinx, "Application Note XAPP224: Data Recovery," Jul. 2005. [Online]. Available: http://www.xilinx.com/support/documentation/application_notes/xapp224.pdf

[97] ——, "MicroBlaze Processor Reference Guide (v14.1)," Apr. 2012. [Online]. Available: http://www.xilinx.com/support/documentation/sw_manuals/xilinx14_1/mb_ref_guide.pdf

[98] S. Haene, D. Perels, and A. Burg, "A Real-Time 4-Stream MIMO-OFDM Transceiver: System Design, FPGA Implementation, and Characterization," *IEEE Journal on Selected Areas in Communications*, vol. 26, no. 6, pp. 877–889, 2008.

[99] M. Šimko, D. Wu, C. Mehlfüehrer, J. Eilert, and D. Liu, "Implementation Aspects of Channel Estimation for 3GPP LTE Terminals," *11th European Wireless Conference*, pp. 1–5, Apr. 2011.

[100] M. H. Hsieh and C. H. Wei, "Channel Estimation for OFDM Systems Based on Comb-type Pilot Arrangement in Frequency Selective Fading Channels," *IEEE Transactions on Consumer Electronics*, vol. 44, no. 1, pp. 217–225, Feb. 1998.

[101] "3GPP TS 36.101 V11.4.0: User Equipment (UE) radio transmission and reception (Release 11)," Mar. 2013. [Online]. Available: http://www.3gpp.org/ftp/Specs/archive/36_series/36.101/36101-b40.zip

[102] O. Edfors, M. Sandell, J. J. van de Beek, S. K. Wilson, and P. O. Börjesson, "OFDM Channel Estimation by Singular Value Decomposition," *IEEE Transactions on Communications*, vol. 46, no. 7, pp. 931–939, July 1998.

[103] F. Edman and V. Öwall, "A Scalable Pipelined Complex Valued Matrix Inversion Architecture," in *IEEE International Symposium on Circuits and Systems (ISCAS)*, vol. 5, May 2005, pp. 4489–4492.

[104] D. Wübben, J. Rinas, R. Böhnke, V. Kühn, and K. K. D., "Efficient Algorithm for Detecting Layered Space-Time Codes," in *4th International ITG Conference on Source and Channel Coding (SCC)*, Jan 2002, pp. 399–405.

[105] R. W. Heath and A. Paulraj, "Antenna Selection for Spatial Multiplexing Systems Based on Minimum Error Rate," in *IEEE International Conference on Communications (ICC)*, vol. 7, 2001, pp. 2276–2280 vol.7.

[106] D. Wübben, R. Böhnke, V. Kühn, and K. D. Kammeyer, "MMSE Extension of V-BLAST Based on Sorted QR Decomposition," in *IEEE 58th Vehicular Technology Conference (VTC)*, vol. 1, 2003, pp. 508–512.

[107] P. Luethi, A. Burg, S. Haene, D. Perels, N. Felber, and W. Fichtner, "VLSI Implementation of a High-Speed Iterative Sorted MMSE QR Decomposition," in *IEEE International Symposium on Circuits and Systems (ISCAS)*, 2007, pp. 1421–1424.

[108] G. H. Golub and C. F. Van Loan, *Matrix Computations*, 3rd ed. Johns Hopkins University Press, Feb. 1996.

[109] Z.-Y. Huang and P.-Y. Tsai, "Efficient Implementation of QR Decomposition for Gigabit MIMO-OFDM Systems," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 58, no. 10, pp. 2531–2542, Oct. 2011.

[110] "3GPP TS 36.212 V11.2.0: Multiplexing and channel coding (Release 11)," Feb. 2013. [Online]. Available: http://www.3gpp.org/ftp/Specs/archive/36_series/36.212/36212-b20.zip

[111] L. Bahl, J. Cocke, F. Jelinek, and J. Raviv, "Optimal Decoding of Linear Codes for Minimizing Symbol Error Rate," *IEEE Transactions on Information Theory*, vol. 20, no. 2, pp. 284–287, 1974.

[112] L. Liu, F. Ye, X. Ma, T. Zhang, and J. Ren, "A 1.1-Gb/s 115-pJ/bit Configurable MIMO Detector Using 0.13-μCMOS Technology," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 57, no. 9, pp. 701–705, Sep. 2010.

[113] M. D. Ercegovac, L. Imbert, D. W. Matula, J. M. Muller, and G. Wei, "Improving Goldschmidt Division, Square Root, and Square Root Reciprocal," *IEEE Transactions on Computers*, vol. 49, no. 7, pp. 759–763, Jul. 2000.

[114] Y. Kim, R. N. Mahapatra, I. Park, and K. Choi, "Low Power Reconfiguration Technique for Coarse-Grained Reconfigurable Architecture," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 17, no. 5, pp. 593–603, May 2009.

[115] S. Ye, S. H. Wong, and C. Worrall, "Enhanced Physical Downlink Control Channel in LTE Advanced Release 11," *IEEE Communications Magazine*, vol. 51, no. 2, pp. 82–89, 2013.

[116] Y. Xie, W. Wolf, and H. Lekatsas, "Code Compression for Embedded VLIW Processors Using Variable-to-fixed Coding," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 14, no. 5, pp. 525–536, 2006.

[117] C. Kozyrakis and D. Patterson, "Vector vs. Superscalar and VLIW Architectures for Embedded Multimedia Benchmarks," in *35th Annual IEEE/ACM International Symposium on Microarchitecture*, 2002, pp. 283–293.

[118] J. M. Rabaey, A. Chandrakasan, and B. Nikolic, *Digital Integrated Circuits - A Design Perspective*, 2nd ed. Prentice Hall, 2002.

[119] K. Mohammed and B. Daneshrad, "A MIMO Decoder Accelerator for Next Generation Wireless Communications," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 18, no. 11, pp. 1544–1555, Nov. 2010.

[120] J. Janhunen, O. Silven, M. Juntti, and M. Myllyla, "Software Defined Radio Implementation of K-Best List Sphere Detector Algorithm," in *International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS)*, July 2008, pp. 100–107.

[121] V. Derudder *et al.*, "A 200Mbps+ 2.14nJ/b Digital Baseband Multi Processor System-on-Chip for SDRs," in *IEEE Symposium on VLSI Circuits (VLSIC)*, 2009, pp. 292–293.

[122] X. Huang, C. Liang, and J. Ma, "System Architecture and Implementation of MIMO Sphere Decoders on FPGA," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 16, no. 2, pp. 188–197, 2008.

[123] D. Sui, Y. Li, J. Wang, P. Wang, and B. Zhou, "High Throughput MIMO-OFDM Detection with Graphics Processing Units," in *IEEE International Conference on Computer Science and Automation Engineering (CSAE)*, vol. 2, May 2012, pp. 176–179.

[124] M. Wu, S. Gupta, Y. Sun, and J. R. Cavallaro, "A GPU Implementation of a Real-Time MIMO Detector," in *IEEE Workshop on Signal Processing Systems (SiPS)*, Oct. 2009, pp. 303–308.

[125] T. Nylanden, J. Janhunen, O. Silven, and M. Juntti, "A GPU Implementation for Two MIMO-OFDM Detectors," in *International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS)*, July 2010, pp. 293–300.

[126] S. Roger, C. Ramiro, A. Gonzalez, V. Almenar, and A. M. Vidal, "Fully Parallel GPU Implementation of a Fixed-Complexity Soft-Output MIMO Detector," *IEEE Transactions on Vehicular Technology*, vol. 61, no. 8, pp. 3796–3800, 2012.

[127] I. Diaz, B. Sathyanarayanan, A. Malek, F. Foroughi, and J. N. Rodrigues, "Highly Scalable Implementation of a Robust MMSE Channel Estimator for OFDM Multi-Standard Environment," in *IEEE Workshop on Signal Processing Systems (SiPS)*, 2011, pp. 311–315.

[128] J. Löfgren, L. Liu, O. Edfors, and P. Nilsson, "Improved Matching-Pursuit Implementation for LTE Channel Estimation," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 61, no. 1, pp. 226–237, Jan. 2014.

[129] R. C. H. Chang, C. H. Lin, K. H. Lin, C. L. Huang, and F. C. Chen, "Iterative QR Decomposition Architecture Using the Modified GramC-Schmidt Algorithm for MIMO Systems," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 57, no. 5, pp. 1095–1102, May 2010.

[130] M. Shabany, D. Patel, and P. G. Gulak, "A Low-Latency Low-Power QR-Decomposition ASIC Implementation in 0.13 μm CMOS," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 60, no. 2, pp. 327–340, Feb. 2013.

[131] M. Mahdavi and M. Shabany, "Novel MIMO Detection Algorithm for High-Order Constellations in the Complex Domain," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 21, no. 5, pp. 834–847, 2013.

[132] P. L. Chiu, L. Z. Huang, L. W. Chai, C. F. Liao, and Y. H. Huang, "A 684Mbps 57mW Joint QR Decomposition and MIMO Processor for 4×4 MIMO-OFDM Systems," in *IEEE Asian Solid State Circuits Conference (ASSCC)*, Nov. 2011, pp. 309–312.

[133] S. Gifford, C. Bergstrom, and S. Chuprun, "Adaptive and Linear Prediction Channel Tracking Algorithms for Mobile OFDM-MIMO Applications," in *IEEE Military Communications Conference (MILCOM)*, vol. 2, Oct. 2005, pp. 1298–1302.

[134] L. Gor and M. Faulkner, "Power Reduction Through Upper Triangular Matrix Tracking in QR Detection MIMO Receivers," in *IEEE 64th Vehicular Technology Conference (VTC)*, Sept. 2006, pp. 1–5.

Appendix

Appendix A

Dataflow Processor Architecture

This appendix includes some detailed hardware development of the 2×2 cell array presented in Part II. The cell array consists of two dataflow processors and two memory cells. The dataflow processors are RISC cores with extended computational units in both "instruction decode" and "write back" stage. Each processor contains 16 general-purpose 16-bit registers and uses a 48-bit fixed-length instruction set. Some of the key features of the processor are:

- SIMD-like operation
- Run-time control and data path configuration
- Conditional instruction execution
- Single-cycle delayed branch
- Zero-delay inner loop control
- Direct I/O port addressing and multi-port data streaming
- In-cell RC supervision and configuration

Figure 1-3 and Table 1-2 present the instruction set of the dataflow processor. Figure 4 and Table 3-6 illustrate the data arrangement blocks and list the configuration set of each pipeline stage. Figure 5-7 present the configuration generation tool developed in-house for the 2×2 cell array. Table 7 and Table 8 describe user commands in the UART and the MATLAB interface, respectively, for controlling the cell array at run-time.

Mnemonics	Operands	Type	Description	Operation	Field 47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
		A			Cond.			OpCode						I	P	Index		Add.																																
		B			Cond.			OpCode						I	P	Index		Add.									Imm.																								
		C			Cond.			OpCode						I	P			D0									32-bit memory data read/write request																							
		D			Cond.			OpCode						I	P	Index		Add.									21-bit configuration																							
		E			Cond.			OpCode						I	P	Index		Add.									21-bit configuration																							
NOP		D	No operation/Configuration updates	—	Cond.	0	0	0	0	0	0	0	0	P	Index		Add.								21-bit configuration																							
GID	Imm	E	Global IO TX port destination ID write	$G0.dst ← Imm	Cond.	0	0	0	0	0	0	0	1	P	Index		Add.								21-bit configuration																					Imm.			
END	Imm	E	Stop execution and return ending code	$END_CODE ← Imm	Cond.	0	0	0	1	0	1	0	1	P	Index		Add.								21-bit configuration																					Imm.			
ILC	S1, D1	D	Inner loop control with register	$ILP ← $PC + 1; D1 ← $ILC; $ILC ← S1	Cond.	0	0	0	1	1	0	P	Index		Add.								21-bit configuration																S1								D1		
ILCI	Imm	E	Inner loop control with imm. Infinite loop control when imm. is all "1"	$ILP ← $PC + 1; $ILC ← Imm	Cond.	0	0	0	1	1	1	P	Index		Add.								21-bit configuration																								Imm.		
LDR	D0, D1, S0, S1	A	MC (RAM) data read request	D0 ← S0 & S1	Cond.	0	0	1	0	0	0	0				D0																			32-bit read request				S0			S1				D1	
LDRI	D0, Imm	C	MC (RAM) data read request	D0 ← Imm & S1	Cond.	0	0	1	0	0	1	0				D0																			32-bit read request				S0			S1				D1	
STR	D0, D1, S0, S1	A	MC (RAM) data write request	D0 ← S0 & S1	Cond.	0	0	1	1	0	0	1	0	0				D0																	D0				S0			S1				D1	
STRI	D0, Imm	C	MC (RAM) data write request	D0 ← Imm	Cond.	0	0	1	1	0	1	1	0	0				D0																			32-bit write request				S0			S1				D1	
CONFI	D0, Imm	C	Resource cell configuration	D0 ← Imm	Cond.	0	0	1	1	0	1	1	0	0				D0																			32-bit write request											
BR		D	Branch register (procedure return)	$PC ← $R_link	Cond.	0	1	0	0	0	0	0	P		Index		Add.								21-bit configuration																					$R_link			
B	S0, S1, Imm	B	Branch relative on condition	$PC ← $PC + SXT(Imm)	Cond.	0	1	0	0	1	0	0										U					Imm.												S0			S1					
BL	S0, S1, Imm	B	Branch and Link (procedure call) on condition	$PC ← $PC + SXT(Imm); $R_link ← $PC + 1	Cond.	0	1	0	1	0	1	0										U					Imm.												$R_link			S0			S1				

Figure 1: Instruction set of the dataflow processor, control-related operations.

Mnem-onics	Operands	Type	Description	Operation	Field 47 46 45 44	43	42 41 40 39	38 37 36 35	34	33	32	31 30 29 28 27 26	25	24 23 22	21 20	19 18 17 16	15 14 13 12 11 10	9 8 7 6 5 4 3 2 1 0
		F			Cond.		OpCode1	OpCode2	I	U	L	V0 V1 A0 A1 R0 R1	ID	EXE	WD	D0	S0	S1 D1
		G			Cond.		OpCode1	OpCode2	I	U	L	V0 V1 A0 A1 R0 R1	ID	EXE	WD	D0	S0	Imm.
ADD	D0, D1, S0, S1	F	Data add between registers	D0 <- S0 + S1	Cond.	1	0 0 0 0	OpCode2	0	U	L	V0 V1 A0 A1 R0 R1	ID	EXE	WD	D0	S0	S1 D1
ADDI	D0, S0, Imm	G	Add register data and imm.	D0 <- S0 + SXT(Imm)	Cond.	1	0 0 0 0		1	U	L	V0 V1 A0 A1 R0 R1	ID	EXE	WD	D0	S0	Imm.
ADC	D0, D1, S0, S1	F	Data add between registers	D0 <- S0 + S1 + Carry	Cond.	1	0 0 0 1		0	U	L	V0 V1 A0 A1 R0 R1	ID	EXE	WD	D0	S0	S1 D1
ADCI	D0, S0, Imm	G	Add register data and imm.	D0 <- S0 + SXT(Imm) + Carry	Cond.	1	0 0 0 1		1	U	L	V0 V1 A0 A1 R0 R1	ID	EXE	WD	D0	S0	Imm.
SUB	D0, D1, S0, S1	F	Data subtract between registers	D0 <- S0 - S1	Cond.	1	0 0 1 0		0	U	L	V0 V1 A0 A1 R0 R1	ID	EXE	WD	D0	S0	S1 D1
SBC	D0, D1, S0, S1	F	Data subtract between registers	D0 <- S0 - S1 - Carry	Cond.	1	0 0 1 1		0	U	L	V0 V1 A0 A1 R0 R1	ID	EXE	WD	D0	S0	S1 D1
ADS	D0, D1, S0, S1	F	Data add and subtract between registers	D0.lo <- S0.lo + S1.lo; D0.hi <- S0.hi - S1.hi	Cond.	1	0 1 0 0		0	U	L	1 1 A0 A1 R0 R1	ID	EXE	WD	D0	S0	S1 D1
ADSC	D0, D1, S0, S1	F	Data add and subtract between registers	D0.lo <- S0.lo + S1.lo + Carry; D0.hi <- S0.hi - S1.hi - Carry	Cond.	1	0 1 0 1		0	U	L	1 1 A0 A1 R0 R1	ID	EXE	WD	D0	S0	S1 D1
MUL	D0, D1, S0, S1	F	Data multiplication between registers	D0 <- S0 * S1	Cond.	1	0 1 1 0	0 0 0 0	0	U	L	V0 V1 A0 A1 R0 R1	ID	EXE	WD	D0	S0	S1
MULI	D0, S0, Imm	G	Multiply register data and imm.	D0 <- S0 * SXT(Imm)	Cond.	1	0 1 1 0	0 0 0 0	1	U	L	V0 V1 A0 A1 R0 R1	ID	EXE	WD	D0	S0	Imm.
CMP	D0, D1, S0, S1	F	Compare date between registers	As SUB without writing result(s)	Cond.	1	1 0 1 0		0	1	L	V0 V1 A0 A1 R0 R1	ID	EXE	WD	D0	S0	S1 D1
CMPI	D0, S0, Imm	G	Compare register data and imm.	As ADDI without writing result	Cond.	1	1 0 0 0		1	1	L	V0 V1 A0 A1 R0 R1	ID	EXE	WD	D0	S0	Imm.
TST	D0, D1, S0, S1	F	Register data test	As AND without writing result(s)	Cond.	1	1 0 0 1		0	1	L	V0 V1 A0 A1 R0 R1	ID	EXE	WD	D0	S0	S1 D1
TSTI	D0, S0, Imm	G	Imm. data test	As ANDI without writing result	Cond.	1	1 0 0 1		1	1	L	Imm. - High	ID	EXE	WD	D0	S0	Imm. - Low
TEQ	D0, D1, S0, S1	F	Register data equality test	As XOR without writing result(s)	Cond.	1	1 0 1 1		0	1	L	V0 V1 A0 A1 R0 R1	ID	EXE	WD	D0	S0	S1 D1
TEQI	D0, S0, Imm	G	imm. data equality test	As XORI without writing result	Cond.	1	1 0 1 1		1	1	L	Imm. - High	ID	EXE	WD	D0	S0	Imm. - Low
AND	D0, D1, S0, S1	F	Logical AND two register data	D0 <- S0 & S1	Cond.	1	1 1 0 0	OpCode2	0	U	L	V0 V1 A0 A1 R0 R1	ID	EXE	WD	D0	S0	S1 D1
ANDI	D0, S0, Imm	G	Logical AND register data with imm.	D0 <- S0 & Imm	Cond.	1	1 1 0 0		1	U	L	Imm. - High	ID	EXE	WD	D0	S0	Imm. - Low
OR	D0, D1, S0, S1	F	Logical OR two register data	D0 <- S0 | S1	Cond.	1	1 1 0 1		0	U	L	V0 V1 A0 A1 R0 R1	ID	EXE	WD	D0	S0	S1 D1
ORI	D0, S0, Imm	G	Logical OR register data with imm.	D0 <- S0 | Imm	Cond.	1	1 1 0 1		1	U	L	Imm. - High	ID	EXE	WD	D0	S0	Imm. - Low
XOR	D0, D1, S0, S1	F	Logical XOR two register data	D0 <- S0 ⊕ S1	Cond.	1	1 1 1 1		0	U	L	V0 V1 A0 A1 R0 R1	ID	EXE	WD	D0	S0	S1 D1
XORI	D0, S0, Imm	G	Logical XOR register data with imm.	D0 <- S0 ⊕ Imm	Cond.	1	1 1 1 1		1	U	L	Imm. - High	ID	EXE	WD	D0	S0	Imm. - Low
MOV	D0, D1, S0, S1	F	Data move between registers	D0 <- S0	Cond.	1	1 1 1 0		0	U	L	V0 V1 A0 A1 R0 R1	ID	EXE	WD	D0	S0	S1 D1
MOVI	D0, S0/D1, Imm	G	Move imm. to a register	D0, D1 <- Imm	Cond.	1	1 1 1 0	1 1 1 0	1	U	L	Imm. - High	ID	EXE	WD	D0	D1	Imm. - Low

Figure 2: Instruction set of the dataflow processor, arithmetic and logic operations.

Operation	Description	Direction	Data package 31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9-3	2	1	0
Program download Header	Package header for downloading program into ProGram Memory (PGM). Address 0 is reserved for the Control register, therefore instructions start from address 1.	Host -> PC	Count									Starting address													Conditional configuration address	CC	P	Write
Program download	Instruction to be downloaded into ProGram Memory. This is a consecutive operation of the "Program download header".	Host -> PC	Instruction																									
PC counter update	Update PC counter starting address, to select program section inside ProGram Memory (PGM).	Host -> PC	0									Starting address													Conditional configuration address	CC	P	Write
Control register update	Update Control register to control the operations of the Processor Cell.	Host -> PC				Stop	Run to	Step	Reset	Pause	Start	Ending address													Conditional configuration address	CC	C	Write
Control register read request	Control register status reading request.	Host -> PC																								CC	C	Read
Control register read data 1	Sending back Control register status, data package 1. This is a consecutive operation of "Control register read request".	PC -> Host				Stop	Run to	Step	Reset	Pause	Start	Ending address													Current PC counter			
Control register read data 2	Sending back Control register status, data package 2. This is a consecutive operation of "Control register read data 1".	PC -> Host																							Instruction END code			

Figure 3: Control instruction set of the dataflow processor.

Table 1: Conditional field of the instruction set.

Opcode	ALU-based condition	Co-ALU-based condition
0000	Equal	Equal
0001	Not equal	Not equal
0010	Negative	Less than
0011	Non-negative	Less than or equal
0100	Carry out	Greater than
0101	Non-carry out	Greater than or equal
0110	Overflow	Positive operand ‘a’
0111	Non-overflow	Negative operand ‘a’
1000	Less than	Positive operand ‘b’
1001	Less than or equal	Negative operand ‘b’
1010	Greater than	Positive operand ‘c’
1011	Greater than or equal	Negative operand ‘c’
1100	Reserved	Positive operand ‘d’
1101	Reserved	Negative operand ‘d’
1110	Reserved	Branch not taken
1111	Always	Always

Table 2: Summary of register address.

Register	Address	Register	Address
General-purpose registers		*Special purpose registers*	
$1	00100	ZERO	00000
$2	00101	PC	00001
$3	00110	Link	00010
$4	00111	Stack	00011
$5	01000		
$6	01001	*Hierarchical IO registers*	
$7	01010	G0	10111
$8	01011		
$9	01100	*Local IO registers*	
$10	01101	L0	11000
$11	01110	L1	11001
$12	01111	L2	11010
$13	10000	L3	11011
$14	10001	L4	11100
$15	10010	L5	11101
$16	10011	L6	11110
$17	10100	L7	11111
$18	10101		
$19	10110		

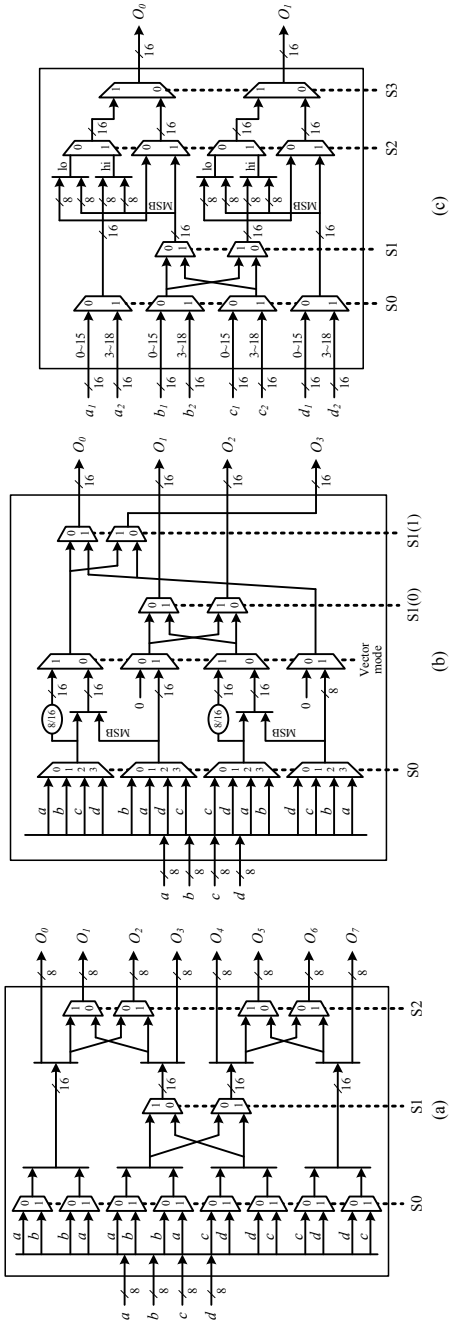


Figure 4: Architecture of data arrangement blocks, (a) block-I, (b) block-II, (c) block-III.

Table 3: Configuration set for function units in Instruction Decoding (ID) stage.

Bit	Field	Description
20-19	opcode_a	Operation code (Table 4) for Barrel shifting, operand A
18-17	opcode_b	Operation code (Table 4) for Barrel shifting, operand B
16	bs_inc_en_a	Enable of automatic shifting increment, operand A
15	bs_en_a	Barrel shifting enable for operand A
14-12	bs_imm_a	Shifting bit count for operand A
11	bs_inc_en_b	Enable of automatic shifting increment, operand B
10	bs_en_b	Barrel shifting enable for operand B
9-7	bs_imm_b	Shifting bit count for operand B
6-3	neg_en	Enable of input data negation (d, c, b, a)
2	sel_alu	ALU status register selection, 0: Co-ALU; 1: ALU
1-0	sel_lane	Processing lane selection

Table 4: Operation code for barrel shifter.

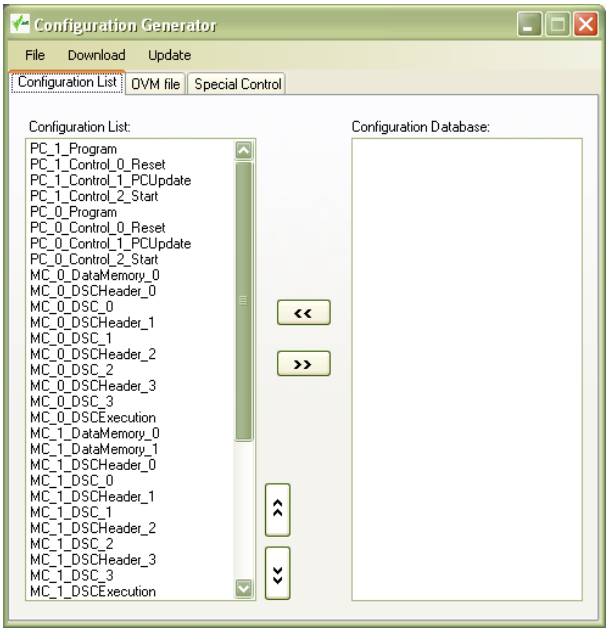
Opcode	Operation
00	Logical Shift Left (LSL)
01	Logical Shift Right (LSR)
10	Arithmetic Shift Right (ASR)
11	ROtate Right (ROR)

Table 5: Configuration set for function units in EXEcution (EXE) stage.

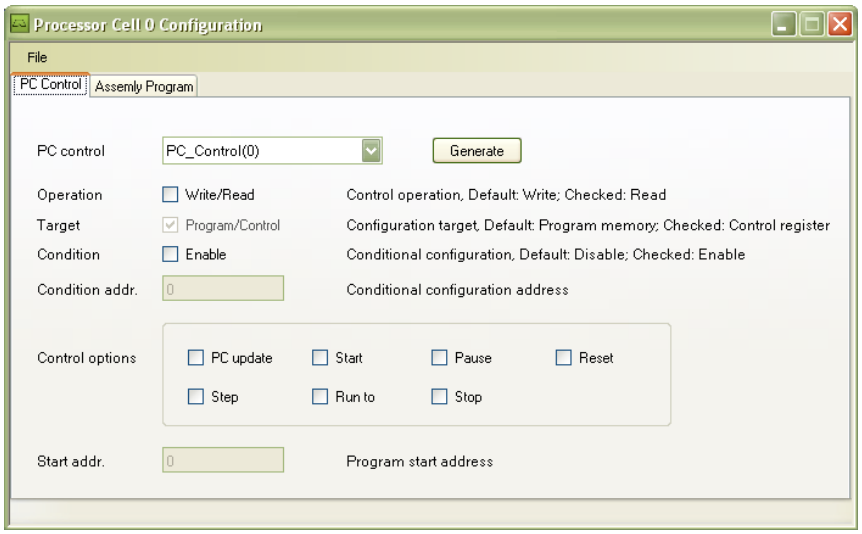
Bit	Field	Description
20-13	mux_1_s0	Control bits for data arrangement block-I, stage 0
12	mux_1_s1	Control bits for data arrangement block-I, stage 1
11-10	mux_1_s2	Control bits for data arrangement block-I, stage 2
9-2	mux_2_s0	Control bits for data arrangement block-II, stage 0
1-0	mux_2_s1	Control bits for data arrangement block-II, stage 1

Table 6: Configuration set for function units in Write Back (WB) stage.

Bit	Field	Description
14-11	add_sub	Accumulator ADD/SUB select (d, c, b, a), 0: addition; 1: subtraction
10-7	mux_3_s0	Control bits for data arrangement block-III, stage 0
6	mux_3_s1	Control bits for data arrangement block-III, stage 1
5-2	mux_3_s2	Control bits for data arrangement block-III, stage 2
1-0	mux_3_s3	Control bits for data arrangement block-III, stage 3



(a)



(b)

Figure 5: Configuration generation tool, (a) bit stream generation, (b) configuration of a processing cell.

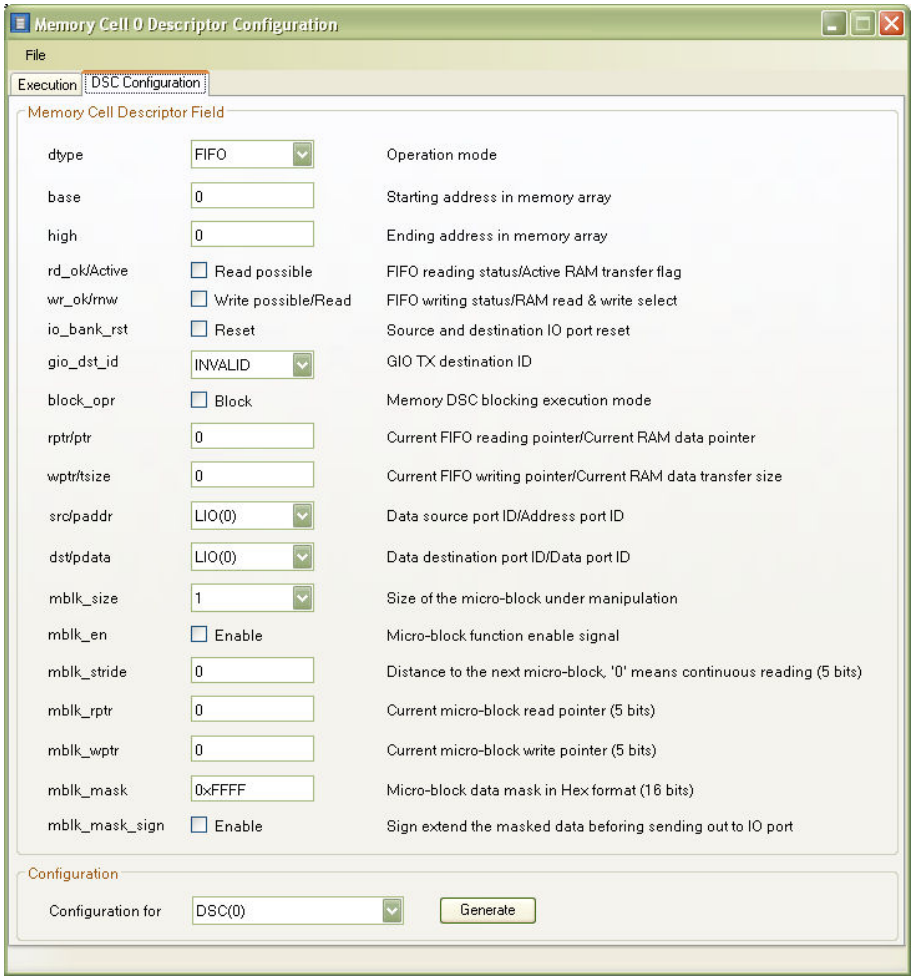


Figure 6: Configuration generation tool, descriptor configuration of a memory cell.

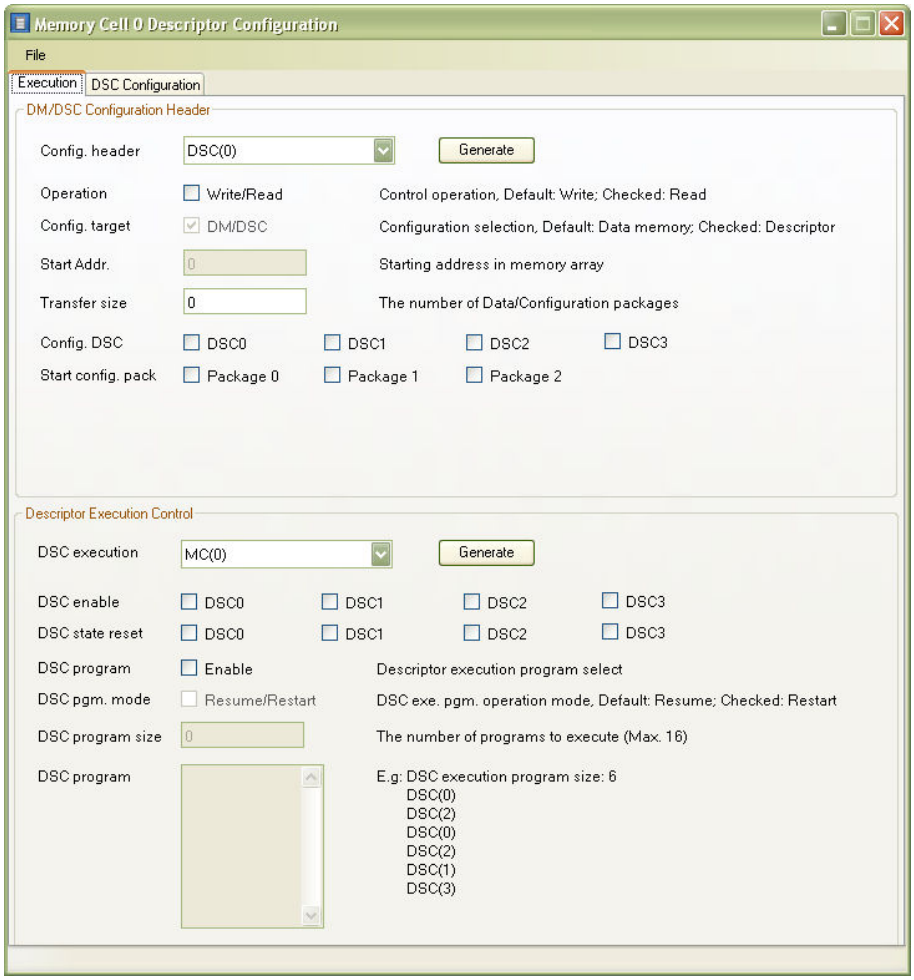


Figure 7: Configuration generation tool, configuration header and descriptor execution program of a memory cell.

Table 7: User commands in UART interface.

CMD	Description	Parameter
g	Destination cell ID for hierarchical I/O communication	Resource cell destination ID (number input): 0∼4
d	Send data inputs via UART to the selected cell	a) Data count (number input) b) Data inputs (string input)
D	Send data inputs via Ethernet to the selected cell	None
i	Send inst./config. packages via UART to the selected cell	a) Inst. count (number input) b) Inst./Config. Inputs (string input)
I	Send inst./config. packages via Ethernet to the selected cell	None
s	Send "start" command to the selected processing cell	None
p	Send "pause" command to the selected processing cell	None
r	Send "reset" command to the selected processing cell	None
e	Send "step" command to the selected processing cell	None
u	Send "run to" command to the selected processing cell	Destination instruction to run to (number input)
o	Send "stop" command to the selected processing cell	None
t	Status tracing of the selected cell	None
c	Send user command to the selected cell	User command (string input)
f	Memory data initialization (zero filling)	Memory cell destination ID (number input): 1, 2
q	Processing cell initialization	Processing cell destination ID (number input): 0, 3
z	Test data set input	None
0	Demo full config. script	None
1	Demo partial config. script	None
h	Command help printout	None

Table 8: User commands in MATLAB interface.

CMD	Description	Parameter
cmd	User command input in UART interface	UART commands
config	Send inst./config. packages from a script file	None
data	Send data inputs from a script file	None
demo	Run a script demo, IEEE 802.11n Sync.	None
rxbuf	Flush UART Rx buffer of the host	None
help	Command help printout	None
exit	Exit user interface in MATLAB	None

Appendix B

Vector Dataflow Processor Architecture

This appendix includes detailed micro-code and instruction set of the vector dataflow processor (Tile-0) in the reconfigurable cell array presented in Part III. The processor is composed of three processing cells for data computations, one memory cell for local data buffering, and a sequencer for control-flow managements. Figure 1 illustrates seven-stage pipeline of the processor. Note that operations mapped onto two function units in the pre-processing stage can be executed concurrently. Table 1-8 presents the micro-code set for each processing and memory cell and Table 13 describes the instruction set of the sequencer.

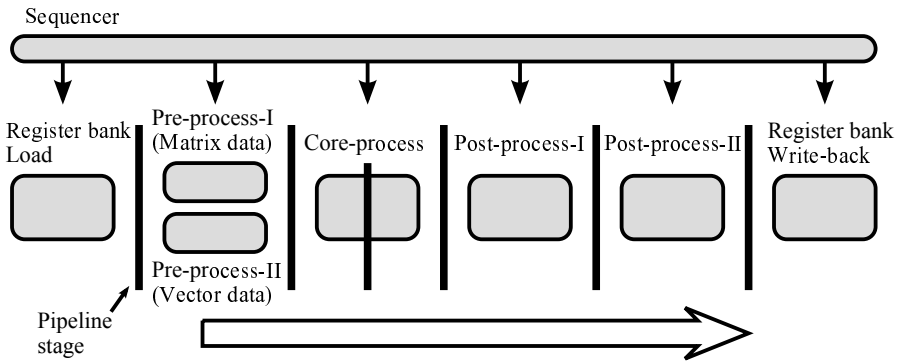


Figure 1: Microarchitecture of the vector dataflow processor, a view of pipeline stages.

Table 1: Micro-code set for the data loading stage of register bank.

Bit	Division	Field	Description
31-22	Reserved		
21	Matrix data operand	reg_mopa_en	I/O port enable for operand A reading, 0: disable; 1: enable
20		reg_mopa_src	Data source for operand A reading, 0: GPR; 1: I/O
19-18		reg_mopa_idx[a]	I/O port number/GPR index for operand A reading
17		reg_mopb_en	I/O port enable for operand B reading
16		reg_mopb_src	Data source for operand B reading, 0: GPR; 1: I/O
15-14		reg_mopb_idx[a]	I/O port number/GPR index for operand B reading
13	Vector data operand	reg_vop_en	I/O port enable for vector operand reading
12		reg_vop_src	Data source for vector operand reading, 0: GPR; 1: I/O
11-8		reg_vop_idx	I/O port number/GPR index for vector operand reading
7-4	VPR	vpr_idx	Index of Vector Permutation Register (VPR)
3-0	MMR	mmr_idx	Index of Matrix Mask Register (MMR)

[a] "Multiple-of-four" addressing (2 LSBs are discarded) for loading matrix data from GPR, e.g., accessing $0, $4, $8, etc.

Table 2: Micro-code set for pre-processing cell I (matrix data).

Bit	Division	Field	Description
31-22	Reserved		
21-20	Lane 0	src_l0	Source operand for processing lane 0, 0: operand A; 1: operand B
19-17		opcode_l0	Operation code (Table 3) for data pre-processing in lane 0
16-15	Lane 1	src_l1	Source operand for processing lane 1, 0: operand A; 1: operand B
14-12		opcode_l1	Operation code (Table 3) for data pre-processing in lane 1
11-10	Lane 2	src_l2	Source operand for processing lane 2, 0: operand A; 1: operand B
9-7		opcode_l2	Operation code (Table 3) for data pre-processing in lane 2
6-5	Lane 3	src_l3	Source operand for processing lane 3, 0: operand A; 1: operand B
4-2		opcode_l3	Operation code (Table 3) for data pre-processing in lane 3
1-0	Mask	mask_src	Source operand for data masking, 0: operand A; 1: operand B

Table 3: Operation code for matrix data pre-processing.

Opcode	Function	Operand
000	None	–
001	Negation	the real part of the complex-valued input
010		the imaginary part of the complex-valued input
011		both real and imaginary part of the complex-valued input
100	Reserved	
101	Absolute	the real part of the complex-valued input
110		the imaginary part of the complex-valued input
111		both real and imaginary part of the complex-valued input

Table 4: Micro-code set for pre-processing cell II (vector data).

Bit	Division	Field	Description
31-14	Reserved		
13	Permutation	perm_en	Permutation enable
12-11		imm_l0	Immediate value input for permutation sequence, processing lane 0
10-9		imm_l1	Immediate value input for permutation sequence, processing lane 1
8-7		imm_l2	Immediate value input for permutation sequence, processing lane 2
6-5		imm_l3	Immediate value input for permutation sequence, processing lane 3
4	Swap	swap_en	Enable for swapping the real and imaginary part of each data operand
3-1	Pre-process	opcode	Operation code (Table 3) for vector data pre-processing
0	Mask	mask_en	Vector data mask enable

Table 5: Micro-code set for core-processing cell.

Bit	Division	Field	Description
31-17	Reserved		
16-15	Shuffle	opcode_shuf	Operation code (Table 9) for input data shuffling
14		swap_en_opa	Enable for swapping the real and imaginary part of matrix operand A
13		swap_en_opb	Enable for swapping the real and imaginary part of matrix operand B
12	SIMD	mul_en	Enable for complex-valued multiplication
11		mul_sign	Signed/unsigned multiplication, 0: signed; 1: unsigned
10		add_en_l1	Enable for addition, level-1 adders
9		add_en_l2	Enable for addition, level-2 adders
8		add_sign_l1	Signed/unsigned addition, level-1 adders, 0: signed; 1: unsigned
7		add_sign_l2	Signed/unsigned addition, level-2 adders
6		add_sub_l1a	Addition/subtraction selection, level-1 adder A, 0: addition; 1: subtraction
5		add_sub_l1b	Addition/subtraction selection, level-1 adder B
4		add_sub_l2a	Addition/subtraction selection, level-2 adder A
3		add_sub_l2b	Addition/subtraction selection, level-2 adder B
2-1		opcode_v	Operation code (Table 10) for vector data operand
0		v_duplicate	Duplication of vector data operand, 0: column-wise; 1: row-wise

Table 6: Micro-code set for post-processing cell I.

Bit	Division	Field	Description
31-18	Reserved		
17	Accumulation	acc_en_l3	Enable for data accumulation, processing lane 3
16		acc_en_l2	Enable for data accumulation, processing lane 2
15		acc_en_l1	Enable for data accumulation, processing lane 1
14		acc_en_l0	Enable for data accumulation, processing lane 0
13		acc_init_l3	Register initialization, processing lane 3, 0: init.; 1: acc.
12		acc_init_l2	Register initialization, processing lane 2
11		acc_init_l1	Register initialization, processing lane 1
10		acc_init_l0	Register initialization, processing lane 0
9	Summation	mux_sum_in	Input multiplexing for lane 3 & 2, 0: straight; 1: swapped.
8		mux_sum_in	Input multiplexing for lane 1 & 0
7		mux_sum_out	Output multiplexing for lane 3 & 2, 0: acc.; 1: sum
6		mux_sum_out	Output multiplexing for lane 1 & 0
5-4	Barrel shifting	bs_opcode	Operation code (Table 11) for barrel shifting
3-0		bs_imm	Shifting bit count

Table 7: Micro-code set for post-processing cell II.

Bit	Division	Field	Description
31-14	Reserved		
13	Sorting	sort_en	Enable for vector data sorting
12		sort_order	Sorting order, 0: ascending; 1: descending
11		sort_sign	Signed/unsigned sorting, 0: signed; 1: unsigned
10-9	Permutation	perm_src	Permutation sequence input, control code in Table 12
8-7		perm_imm_l0	Immediate value input for permutation sequence, processing lane 0
6-5		perm_imm_l1	Immediate value input for permutation sequence, processing lane 1
4-3		perm_imm_l2	Immediate value input for permutation sequence, processing lane 2
2-1		perm_imm_l3	Immediate value input for permutation sequence, processing lane 3
0	Mask	mask_en	Enable for vector data mask

Table 8: Micro-code set for the write-back stage of register bank.

Bit	Division	Field	Description
31-19	Reserved		
18	Matrix data writing	reg_m_wen	Enable for matrix data writing
17		reg_m_src	Data source for register writing, 0: matrix bus; 1: vector bus
16		reg_m_vdup	Duplication of vector data output, 0: row-wise; 1: column-wise
15		reg_m_dst	Destination for matrix data writing, 0: GPR; 1: I/O
14-13		reg_m_idx[a]	I/O port number/GPR index for matrix data writing
12	Vector data writing	reg_v_wen	Enable for vector data writing
11		reg_v_dst	Destination for register writing, 0: GPR; 1: I/O
10-7		reg_v_idx	I/O port number/GPR index for vector data writing
6	VPR/ MMR	reg_s_wen	Register write enable
5		reg_s_dst	Destination for data writing, 0: VPR; 1: MMR
4		reg_s_src	Data source for register writing, 0: sorted data; 1: vector data
3-0		reg_s_idx	Register index

[a] "Multiple-of-four" addressing (2 LSBs are discarded) for writing matrix data to GPR, e.g., accessing $0, $4, $8, etc.

Table 9: Operation code for input data shuffling in the SIMD core.

Opcode	Operation
00	Complex-valued arithmetic
01	Real-valued arithmetic
10	Complex- & real-valued square operation
11	Reserved

Table 10: Operation code for vector data operand in the SIMD core.

Opcode	Operation
00	None
01	Constant multiplication
10	Constant addition
11	Reserved

Table 11: Operation code for barrel shifter.

Opcode	Operation
00	None
01	Arithmetic Shift Right (ASR)
10	Logical Shift Left (LSL)
11	Logical Shift Right (LSR)

Table 12: Data source for permutation sequence.

Opcode	Operation
00	No permutation
01	Using sorting output as a permutation sequence
10	Using sequence loaded from VPR
11	Using an immediate value input as a permutation sequence

Table 13: Instruction set for sequencer.

	31-29	28	27-24	23-20	19-16	15-12	11-8	7-4	3-0
Instruction	Opcode	Field							
NOP	000	L[a]	–						
Normal[b]	001	L	offset_1	offset_2	offset_3	offset_4	offset_5	offset_6	offset_7
Loop push[c]	010	–	Loop count			–			
Base config.[d]	011	–	base_1	base_2	base_3	base_4	base_5	base_6	base_7
End of program	100	–	–						ID
Reserved	101	–							
	110	–							
	111	–							

[a] "end-of-loop" flag.
[b] Normal instruction, controls the address ("address = base + offset") of distributed configuration memories.
[c] Pushing a loop into the stack of the inner loop controller, including link address and loop count.
[d] Base address configuration for the distributed configuration memories.