Waqar Hussain · Jari Nurmi
Jouni Isoaho · Fabio Garzia *Editors*

Computing Platforms for Software-Defined Radio

Springer

Computing Platforms for Software-Defined Radio

Waqar Hussain • Jari Nurmi • Jouni Isoaho
Fabio Garzia
Editors

Computing Platforms for Software-Defined Radio

Springer

Editors

Waqar Hussain
Department of Electronics
and Communications Engineering
Tampere University of Technology
Tampere, Finland

Jari Nurmi
Department of Electronics
and Communications Engineering
Tampere University of Technology
Tampere, Finland

Jouni Isoaho
Communication Systems laboratory,
Department of Information Technology
University of Turku
Turku, Finland

Fabio Garzia
Fraunhofer IIS
Nürnberg, Germany

ISBN 978-3-319-49678-8 ISBN 978-3-319-49679-5 (eBook)
DOI 10.1007/978-3-319-49679-5

Library of Congress Control Number: 2016963251

Printed on acid-free paper

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

Preface

Back in 2008, Prof. Jari Nurmi hosted the International System-on-Chip Symposium in Tampere, Finland, with a key focus on software-defined radio (SDR) technology. The symposium was attended by a large audience traveling from different parts of the world and in particular by Dr. Joseph Mitola who coined the term "the SDR." At the end of the conference, the editors felt the absolute need to compile a comprehensive book that particularly addresses the baseband processing engines for SDR—a book still largely required by many in academia and industry.

SDR has revolutionized how we deal with radio electronics today. SDR has impacted almost every field of life and spawned newer ideas like cognitive and intelligent radios. It is evident from the many chapters submitted to this book that software support for signal processing is equally important than the efforts put in place to increase the transistor density on the chip. The editors gratefully acknowledge the contributions from all the authors and their patience during the stringent review process. The editors are thankful for their cooperation and timely responses.

Tampere, Finland	Waqar Hussain
Tampere, Finland	Jari Nurmi
Turku, Finland	Jouni Isoaho
Nuremberg, Germany	Fabio Garzia
September 2016	

Acknowledgements

The editors sincerely thank the many agencies and organizations providing funding during the past many years. Special acknowledgments to the Academy of Finland, the European Union, Nokia, TEKES, TISE, GETA, and Tampere University of Technology, Finland, for consistent funding and support for projects that led to the compilation of this book.

Contents

1 The Evolution of Software-Defined Radio: An Introduction 1
Waqar Hussain, Jouni Isoaho, and Jari Nurmi

Part I Architectures, Designs and Implementations

2 Design Transformation from a Single-Core to a Multi-Core Architecture Targeting Massively Parallel Signal Processing Algorithms 7
Waqar Hussain, Henry Hoffmann, Tapani Ahonen, and Jari Nurmi

3 The CoreVA-MPSoC: A Multiprocessor Platform for Software-Defined Radio 29
Gregor Sievers, Boris Hübener, Johannes Ax, Martin Flasskamp, Wayne Kelly, Thorsten Jungeblut, and Mario Porrmann

4 Design and Implementation of IEEE 802.11a/g Receiver Blocks on a Coarse-Grained Reconfigurable Array 61
Sajjad Nouri, Waqar Hussain, Diana Göhringer, and Jari Nurmi

5 Reconfigurable Multiprocessor Systems-on-Chip 91
Diana Goehringer

6 Ninesilica: A Homogeneous MPSoC Approach for SDR Platforms .. 107
Roberto Airoldi, Fabio Garzia, Tapani Ahonen, and Jari Nurmi

Part II Software-Based Radio Cognition and Implementation Tools

7 Application of the Scalable Communications Core as an SDR Baseband 123
Anthony Chun and Jeffrey D. Hoffman

8 HW/SW Co-design Toolset for Customization of Exposed Datapath Processors 147
Pekka Jääskeläinen, Timo Viitanen, Jarmo Takala, and Heikki Berg

9 FPGA-Based Cognitive Radio Platform with Reconfigurable Front-End and Antenna 165
Aitor Arriola, Pedro Manuel Rodríguez, Raúl Torrego, Félix Casado, Zaloa Fernández, Mikel Mendicute, Eñaut Muxika, Juan Ignacio Sancho, and Iñaki Val

10 Synchronization in NC-OFDM-Based Cognitive Radio Platforms ... 189
Farid Shamani, Tapani Ahonen, and Jari Nurmi

11 Towards Adaptive Cryptography and Security with Software Defined Platforms 209
Antti Hakkala, Jouni Isoaho, and Seppo Virtanen

12 The Future of Software-Defined Radio: Recommendations 237
Waqar Hussain, Jouni Isoaho, and Jari Nurmi

Bibliography 239

Contributors

Tapani Ahonen Department of Electronics and Communications Engineering, Tampere University of Technology, Tampere, Finland

Roberto Airoldi Department of Electronics and Communications Engineering, Tampere University of Technology, Tampere, Finland

Aitor Arriola ICT Department, IK4-IKERLAN, Arrasate-Mondragón, Spain

Johannes Ax Cognitronics and Sensor Systems Group, CITEC, Bielefeld University, Bielefeld, Germany

Heikki Berg Nokia Technologies, Tampere, Finland

Félix Casado ICT Department, IK4-IKERLAN, Arrasate-Mondragón, Spain

Anthony Chun Internet of Things Group, Intel Corporation, Santa Clara, CA, USA

Zaloa Fernández ICT Department, IK4-IKERLAN, Arrasate-Mondragón, Spain

Martin Flasskamp Cognitronics and Sensor Systems Group, CITEC, Bielefeld University, Bielefeld, Germany

Fabio Garzia Department of Electronics and Communications Engineering, Tampere University of Technology, Tampere, Finland

Diana Göhringer Ruhr-Universität Bochum (RUB), Bochum, Germany

Antti Hakkala Department of Information Technology, Communication Systems laboratory, University of Turku, Turku, Finland

Jeffrey D. Hoffman Internet of Things Group, Intel Corporation, Hillsboro, OR, USA

Henry Hoffmann Department of Computer Science, The University of Chicago, Chicago, IL, USA

Boris Hübener Cognitronics and Sensor Systems Group, CITEC, Bielefeld University, Bielefeld, Germany

Waqar Hussain Department of Electronics and Communications Engineering, Tampere University of Technology, Tampere, Finland

Jouni Isoaho Department of Applied Physics, Laboratory of Electronics and Information Technology, University of Turku, Turku, Finland

Pekka Jääskeläinen Tampere University of Technology, Tampere, Finland

Thorsten Jungeblut Cognitronics and Sensor Systems Group, CITEC, Bielefeld University, Bielefeld, Germany

Wayne Kelly Cognitronics and Sensor Systems Group, CITEC, Bielefeld University, Bielefeld, Germany

Mikel Mendicute Electronics and Computing Department, University of Mondragón, Arrasate-Mondragón, Spain

Eñaut Muxika Electronics and Computing Department, University of Mondragón, Arrasate-Mondragón, Spain

Sajjad Nouri Tampere University of Technology, Tampere, Finland

Jari Nurmi Department of Electronics and Communications Engineering, Tampere University of Technology, Tampere, Finland

Mario Porrmann Cognitronics and Sensor Systems Group, CITEC, Bielefeld University, Bielefeld, Germany

Pedro Manuel Rodríguez ICT Department, IK4-IKERLAN, Arrasate-Mondragón, Spain

Juan Ignacio Sancho Department of Electronics and Communications, CEIT and Tecnun - University of Navarra, San Sebastián, Spain

Farid Shamani Tampere University of Technology, Tampere, Finland

Gregor Sievers Cognitronics and Sensor Systems Group, CITEC, Bielefeld University, Bielefeld, Germany

Jarmo Takala Tampere University of Technology, Tampere, Finland

Raúl Torrego ICT Department, IK4-IKERLAN, Arrasate-Mondragón, Spain

Iñaki Val ICT Department, IK4-IKERLAN, Arrasate-Mondragón, Spain

Timo Viitanen Tampere University of Technology, Tampere, Finland

Seppo Virtanen Department of Information Technology, Communication Systems laboratory, University of Turku, Turku, Finland

Chapter 1
The Evolution of Software-Defined Radio: An Introduction

Waqar Hussain, Jouni Isoaho, and Jari Nurmi

The Software-Defined Radio (SDR) concept was originally developed by the combined efforts of various research groups in the private and government organizations of the United States (US) in 1970s–1980s. The important ones to mention are the US Department of Defense Laboratory and a team at the Garland, Texas Division of E-Systems Inc. In 1991, Joe Mitola independently reinvented the term 'Software Radio' (SR) in cooperation with E-Systems as a plan to build a true software-based GSM transceiver [1]. The SR platform essentially processes almost all the transceiver algorithms as software for a processor. This includes nearly all layers of transmission. However, an optimal implementation of physical layer is always challenging due to an enormous amount of mathematical computation. Over the period of time, many developmental changes occurred and an interesting feature of cognition was added to existing SDR platforms, thereby inventing the term 'Cognitive Radio'. The main idea was to reduce over-sampling by the analog to digital converter, reduce on-chip processing and to target only the spectrum of interest. This book also touches the CR feature in the large SDR field in some of its selected chapters. Since, the very first few articles of J. Mitola, there has a been a tremendous amount of research work conducted in industry and academia. The evolution in SDRs is continuous with time and provides a number of excellent opportunities to researcher for exploration and to come up with their findings. The present day SDR implementations are such that the designers are focused mostly on

W. Hussain (✉) • J. Nurmi
Department of Electronics and Communications Engineering, Tampere University of Technology, Tampere, Finland
e-mail: waqar.hussain@tut.fi; jari.nurmi@tut.fi

J. Isoaho
Department of Applied Physics, Laboratory of Electronics and Information Technology, University of Turku, Turku, Finland
e-mail: jisoaho@utu.fi

W. Hussain et al. (eds.), *Computing Platforms for Software-Defined Radio*,
DOI 10.1007/978-3-319-49679-5_1

the design of hardware and software, their interfacing and optimizations for varying architectural choices. It includes multiple cases of application-specific general-purpose acceleration platforms that are scalable, homogeneous, and heterogeneous in nature while providing multiple programmable cores on a single chip computing system.

There are ten chapters included in this book written by different authors who designed platforms to support the execution of specific algorithms. The platforms were then synthesized for different process sizes or their images were placed and routed over the FPGAs to measure or estimate the performance and then to conduct comparisons with the other state of the art. However, among different architectural choices, none has been observed as an absolute SDR processing platform. However, it is important to have a collective look from the architectures and in future the design choice can be influenced from already existing platforms—so a nearly approaching design of a platform can be made that falls in the abstract defining the vague boundaries of an absolute SDR processor. There are a number of radio protocols that need to be supported by a single transceiver system. Nevertheless, most of them are computation-intensive and require parallel processing while containing some sub-programs that are serial in nature.

We have divided the contents of the book into two parts and each part contains five chapters. The first part is a compilation of Architectures, Designs, and Implementations that are widely accepted for SDR realization. The second part is focused on the radio protocols implementation in software while introducing cognition mechanism for RF sensing of available spectrum and implementation of some of the customized tools from the academia. The details related to software implementation are presented in connection with the underlying hardware platform. Let's discuss highlights from the chapters included and the key points authors have mentioned. We will then try to summarize and make points to some conclusions.

Part-I: Architectures, Designs, and Implementations

This part includes some of the latest SDR architectures and their design implementations. The first chapter 'Design Transformation from a Single-Core to a Multi-Core Architecture targeting Massively Parallel Signal Processing Algorithms' describes an evolution from single core Coarse-Grained Reconfigurable Array (CGRA) architectures to a Heterogeneous Accelerator-Rich Platform (HARP) which contained multiple CGRA connected with each other over a Network-on-Chip. The HARP platform was subjected to some of the most stringent test comprising of many computationally intensive signal processing algorithms. The second chapter, 'The CoreVA-MPSoC: A Multiprocessor Platform for Software-Defined Radio' is about a hierarchical system architecture composed of multiple VLIW processors and the overall platform supports fine-grained instruction-level parallelism thereby benefiting important SDR related algorithms. The authors introduce LLVM-based C-compiler and a StreamIt-based compiler to map applications on multiple processors. The third chapter, ‘Design and Implementation of IEEE 802.11a/g Receiver Blocks on a Coarse-Grained Reconfigurable Array’ is about the implementation of almost all the kernels of a single input single out WiFi receiver on a processor-CGRA model. Almost 80 % of the computational tasks were assigned to the CGRA and some serial processing tasks were left of the RISC processor to solve. The case to conduct such a work was to know the feasibility of a CGRA for processing a large set of different signal processing related algorithms. In the fourth chapter, the author emphasizes high flexibility, high scalability, high performance while low power dissipation for SDR solutions to fulfil modern day requirements and uses a ‘Reconfigurable Multiprocessor Systems-on-Chip’ architecture to prove her point. In the last chapter of Part I, the authors present a Multi-Processor System-on-Chip architecture composed on nine RISC cores arranged as two dimensional array of three rows and three columns. The platform is tested for an FFT algorithm where the computational load is almost equally distributed over the nine cores. From this first part, the reader can see how single core platforms, homogeneous and heterogeneous multicores contribute to the solution of the SDR design problem. It can be beneficial further to the solution, if the next generation platforms contain flavor from all the three design philosophies.

Part-II: Software-Based Radio Cognition and Implementation Tools

In the first chapter of this part, ‘Application of the Scalable Communications Core as an SDR Processor’, the authors emphasize on the efficient support of a wide range of wireless standards and the cognitive radios that can perform RF sensing for the available spectrum. The authors present a scalable communications core, developed by Intel Labs which was programmable and allowed resource sharing for efficient scheduling of radio threads. The tape-out was performed at 65 nm CMOS. The chip supported the processing of multiple protocols including WiFi, WiMAX, GPS, Bluetooth, and DVB-H. In the next chapter, ‘HW/SW Co-Design Toolset for Customization of Exposed Datapath Processors’, the authors present an open source toolset for hardware and software co-design to support different types of parallelisms in applications. The toolset is based on a re-targetable high-level language compiler and a scalable datapath template. The third chapter, ‘FPGA-based Cognitive Radio Platform with Reconfigurable Front-End and Antenna’ presents an SDR platform that is prototyped on an FPGA as a proof-of-concept for radio cognition. The platform works on a 12.8 Mbps RF-to-Ethernet bridge to support industrial, scientific, and medical bands of 868 MHz and 2.45 GHz. Another interesting chapter, ‘Synchronization in NC-OFDM-Based CR Platforms’ is presented which discusses the implementation of a reconfigurable synchronizer and its multiple derivatives. The author highlights different architectural choices affecting resource consumption, operating frequency, and power dissipation on a FPGA device. The last chapter highlights cryptography and security issues in modern communications system and highlights the potential role of SDR platforms towards contribution.

In the end, a concluding chapter is presented summarizing the key points from all the chapters and recommendations for the reader. The chapter indicates important characteristics that an SDR platform should contain in future to support a wide range of radio applications with ever increasing performance levels with minimum costs and investments. It further discusses the future SDR as a key enabler for many upcoming technologies that can change the human society and our planet Earth.

Reference

1. Mitola, J.: Software radios-survey, critical evaluation and future directions. In: Telesystems Conference, 1992. NTC-92, National, Washington, DC, pp. 13/15–13/23 (1992). doi:10.1109/NTC.1992.267870

Part I
Architectures, Designs and Implementations

Chapter 2
Design Transformation from a Single-Core to a Multi-Core Architecture Targeting Massively Parallel Signal Processing Algorithms

Waqar Hussain, Henry Hoffmann, Tapani Ahonen, and Jari Nurmi

2.1 Introduction

Parallel processing of digital signals demands an ever increasing throughput and is expected to increase in future. Considering the upcoming 5G standards, the chip designs accelerating such computationally intensive tasks are subject to many challenges. This includes but not limited to achieve specified execution-time constraints, resource/area consumption on the chip and to deal with power/energy walls. The Dennardian scaling of transistor feature size was promising to increase the number of transistors on the chip while keeping the power dissipation constant [8]. This scaling was supporting Moore's law, i.e., the number of transistors on a silicon chip will double after every 2 years. However, the down-scaling of the transistor size and increasing the operating frequency increased the power density of the chip to unprecedented thermal levels and further packing of more transistors while keeping the power dissipation constant became infeasible. A solution was found in multi-core scaling which lasted not for long, where multiple cores are combined together on a chip and the workload is distributed over all the cores. Studies by the research group of Michael B. Taylor at the University of California, San Diego, have shown with an experimental evidence that the power-wall is obstructing again the existence of Moore's law. The continuous scaling in the number of cores is not consistent for keeping the power dissipation constant. The study shows that only a 7.0 % area of a 300 mm^2 homogeneous multi-core chip can

W. Hussain • T. Ahonen • J. Nurmi
Department of Electronics and Communications Engineering, Tampere University of Technology, Tampere, Finland
e-mail: waqar.hussain@tut.fi; ukaata@gmail.com; jari.nurmi@tut.fi

H. Hoffmann (✉)
Department of Computer Science, The University of Chicago, Chicago, IL, USA
e-mail: hankhoffmann@cs.uchicago.edu

W. Hussain et al. (eds.), *Computing Platforms for Software-Defined Radio*,
DOI 10.1007/978-3-319-49679-5_2

be operated at its full frequency within a power budget of 80 W [21]. Hence most of the part of the chip stays under-utilized, i.e., switched-off (Dark) or being operated at a very low frequency (Dim). The overall terminology used for dark or dim part of chip is known as Dark Silicon. Different research groups across the world are addressing this problem in different domains. For example, Hank Hoffmann at the Computer Science and Artificial Intelligence Laboratory, Massachusetts Institute of Technology, USA, presented a Self-Aware Computing model to dynamically control the voltage and operating frequency for power mitigation of a video decoder using an adaptive feedback control system. Efforts have been placed to design 3D integrated circuits for energy efficiency as well as the traditional Dynamic Voltage and Frequency Scaling (DVFS) approach is also actively pursued. The author in [19] has suggested the use of Coarse-Grained Reconfigurable Arrays (CGRAs) as one of the possible solutions to deal with this issue. Since CGRAs provide a large number of Processing Elements (PEs) to work in parallel, they can process computationally intensive signal processing tasks at a high throughput while being operated at a low frequency. The parts of the silicon which are dark can be replaced by custom-computing CGRAs, and since they are reconfigurable, they can adapt to the needs to multiple applications.

In this chapter, we introduce a scalable model of a template-based CGRA and a brief description of the crafting mechanism to generate application-specific CGRA-based accelerators. Then using the CGRAs for integration over a Network-on-Chip (NoC) to establish a Heterogeneous Accelerator-Rich Platform (HARP). HARP is also a template-based design and its instances can contain a variable number of Reduced Instruction-Set Computing (RISC) cores as well as user-specified number of CGRAs. The overall design demonstrates the advantages of loose-coupling where all the cores over the NoC can access each other's communication and computation resources. The design of HARP is an effort to maximize the number of reconfigurable processing resources on a single chip. HARP has been evaluated for a proof-of-concept test as well as for a relatively realistic design problem, i.e., satisfying the execution-time constraints for Fast Fourier Transform (FFT) processing required in the baseband processing of IEEE-802.11n demodulators. In addition to this, HARP has been subjected to constraint-aware workload balancing in conjunction with dynamic frequency scaling. This work showed interesting results and the importance of adopting cost effective self-aware computing models in heterogeneous multi-core architectures. The HARP design has also been evaluated for suitability to 3D Integrated Circuit (IC) design and as a competitive contender in the dark silicon era.

This chapter is organized as follows. The next section describes the existing multi-core platforms that have a similar architecture as of HARP. Section 2.3 explains the design and usage of the CGRAs to instantiate application-specific accelerators to be integrated to HARP nodes. The next following section describes the NoC which is used as a backbone for communication and control between all the nodes. The overall HARP architecture is described in Sect. 2.6 followed by hardware and software integration details presented in Sect. 2.5. The measurement and estimation of different basic and advanced level performance metrics are described in Sect. 2.8. Section 2.9 establishes comparisons between HARP and the state-of-the-art platforms described in Sect. 2.2. In the last section, conclusions are presented.

2.2 Existing State of the Art

The multi-core platforms exist both in homogeneous and heterogeneous forms. A homogeneous platform is generally composed of similar cores loosely connected with each other while a heterogeneous platform is a loose or tight integration of different cores over a communication infrastructure. Both forms of the multi-core platforms are programmable in C but for heterogeneous platforms, additional custom tools may be required than a simple C compiler. In a heterogeneous platform, cores are generally application-specific targeting only a set of applications. For example, reconfigurable Application-Specific Instruction-set Processors (rASIPs), Fine-Grained, Mid-Grained, and Coarse-Grained Reconfigurable Array devices, Very Long Instruction Word (VLIW) processors, and nevertheless to mention the Digital Signal Processors (DSPs).

2.2.1 MORPHEUS

Morpheus is an integrated platform made of a Fine-, Mid-, and a Coarse-Grained Reconfigurable Array (CGRA) [23]. The platform supports an operating system and uses a custom-designed tool chain for reconfigurable accelerators. The Fine-, Mid-, and Coarse-Grained devices are called as FlexEOS [22], DREAM [6], and XPP-III [7] and these devices communicate over an NoC. FlexEOS is a high-density multi-logic FPGA fabric made of SRAM cells. DREAM is a reconfigurable DSP core which can perform general-purpose processing using a 32-bit RISC processor and the reconfiguration feature is supported by PiCoGA [16]. XPP-III is a CGRA integrated into the datapath of a VLIW processor.

2.2.2 P2012

P2012 is a homogeneous multi-core platform composed of 16 processors equally divided into four clusters [18]. All the processors in a cluster are locally synchronous and globally asynchronous with processors in other clusters. The P2012 platform is tested for image processing algorithms and shows a performance of 0.04 Giga Operations Per Second per Milli Watt (GOPS/mW) on a 90 nm Complimentary Metal-Oxide Semiconductor (CMOS) technology.

2.2.3 *NineSilica*

NineSilica is a homogeneous Multi-Processor System-on-Chip (MPSoC) platform, designed and developed at Tampere University of Technology, Finland [2]. The platform contains nine RISC cores integrated together over an NoC in a topology of three rows and three columns. Each RISC core has an instruction and a data memory. The synchronization between cores is established based on using a shared-memory architecture, where the data is exchanged using flags. NineSilica has a track record of processing many computationally intensive signal processing related algorithms, e.g., Fast Fourier Transform (FFT).

2.2.4 *RAW*

Reconfigurable Architecture Workstation (RAW) is composed of sixteen 32-bit MIPS2000 processors placed in an order of four rows and four columns [20] and the communication between the processors is facilitated by an NoC. RAW supports static and dynamic scheduling hence being similar to a reconfigurable fabric as well as a multi-core platform.

2.2.5 *CRISP*

CRISP was a European Union funded project that stands for Cutting edge Reconfigurable ICs for Stream Processing [1]. It presents a general stream processor which is a highly scalable platform with core-level coarse-grained granularity. The platform targets streaming DSP applications and is designed on the principle of locality-of-reference which indicates different levels of storage. An instance of this platform was generated which was composed of a general-purpose processor and a reconfigurable fabric. The system also contains an ARM core and 45 Xentium DSP cores, where all the cores are integrated with each other using an NoC. The Xentium processor can operate at 200.0 MHz using a 90 nm CMOS technology. The overall general-purpose processor device is manufactured using UMC 130 nm CMOS technology, yielding an operating frequency of 200.0 MHz.

2.2.6 *Intel's Single-Chip Cloud Computer*

A 48 core x86-based Single-Chip Cloud Computer (SCC) was designed by Intel to provide an opportunity for design-space exploration with custom-application implementation and to explore future design challenges and opportunities [17].

The communication between cores is provided using a mesh network and memory controllers. The SCC is implemented using a 45 nm CMOS technology and all the processors can be tuned to different operating frequencies between 100.0 and 800.0 MHz. The communication network can be operated from 0.8 to 1.6 GHz and the platform has been benchmarked by different research groups across the globe.

2.2.7 TILE64™

TILE64™is a homogeneous multi-processor composed of 64 cores arranged in an order of eights rows and eight columns and covers a wide range of embedded applications [4]. Each processor core is a 64-bit VLIW machine but the integer datapath is 32-bit wide to support subword computations. All the cores support SMP Linux kernel and run at 750.0 MHz while delivering a performance of 384 GOPS. All the cores communicate with each using an NoC which provides a bandwidth of 120.0 GB/s. The processor system has three layers of caches and each core has a autonomous DMA engine.

2.3 Scalable CGRAs

In this section, special emphasis is given to briefly describe the family of template-based CGRAs used in the integrated design of HARP. In particular, the CGRAs used are CREMA [9], AVATAR [11], and SCREMA [12]. Figure 2.1 shows combined representation of all the three CGRAs. The unit of processing in all the CGRAs is a Processing Element (PE) which is a 32-bit Arithmetic and Logic Unit (ALU) capable to perform arithmetic and logic operations in integer and IEEE-754 format. Each PE has two inputs (a, b) and two outputs (A, B). The inputs are supplied with operands which reside in the corresponding registers inside each PE. An arithmetic or logic operation can be performed on the operands received by the registers and the output A provides the results. The output B provides the registered value of input b, so to be multiplexed to the inputs of other PEs for the maximum availability of data. The template feature of the CGRAs allows all the processing elements to instantiate only those computational resources that are required by the set of target applications. This is specified by the user manually during the system design time using a custom-made Graphical User Interface (GUI) tool. It optimizes resource utilization on FPGA or area on an ASIC, increases operating frequency, and reduces power and energy consumption. The PEs can exchange data with each other using point-to-point local and global interconnections. Local interconnections are with the neighboring PEs while the global ones are with the PEs that are located relatively farther than the neighboring ones. CREMA is an R×8, AVATAR is an R×16, and SCREMA is an R×C PE CGRA, where R can be any positive integer and C can only have values, i.e., 4, 8, 16, and 32. The CGRAs are equipped with two local

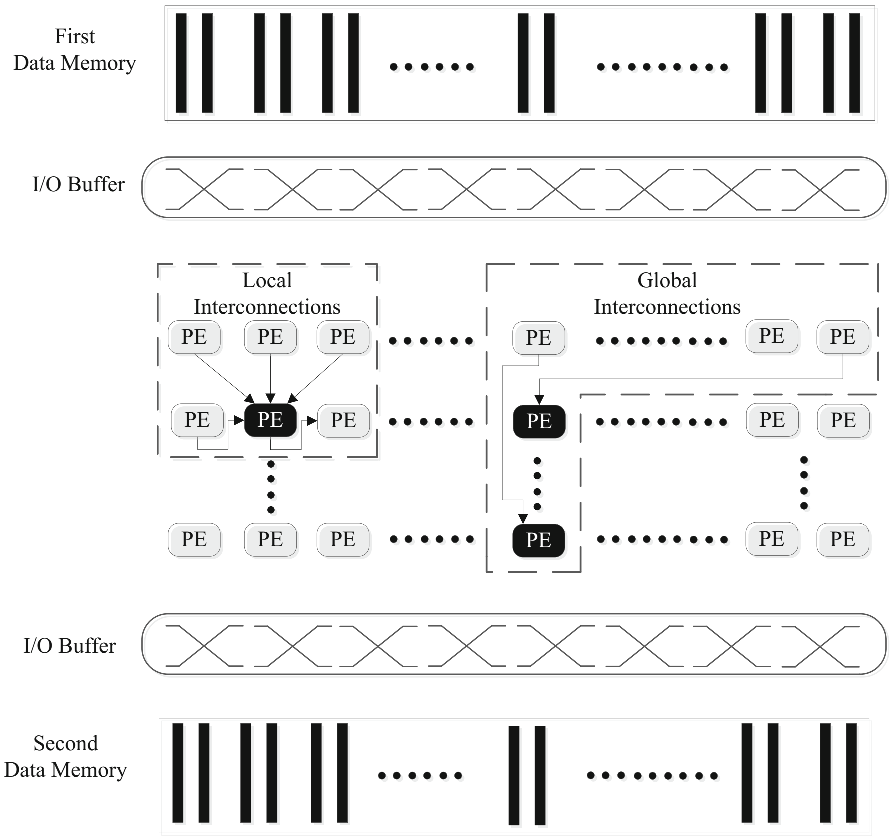


Fig. 2.1 A combined representation of CREMA, AVATAR, and SCREMA. The figure shows local and global interconnections towards destined PEs shown in *black*, I/O buffers, and the data memories [13]

memories to contain the data to be processed as shown in Fig. 2.1. The figure also shows two I/O buffers but they are only instantiated in the designs of CREMA and AVATAR. The I/O buffers make each memory bank in the local memory accessible to almost each and every input of the PE array for read and write operations. I/O buffers are composed of multiplexers and registers. The size of the multiplexer will always be 2C×1 and there will always be 2C number of multiplexers in each I/O buffer. SCREMA does not carry I/O buffers as in case of C = 32, the number and size of each multiplexer will be very large thus consuming a large portion of the overall design space. These CGRAs were initially designed to work as accelerators with an RISC core. A Direct Memory Access (DMA) device was tightly integrated with each CGRA to transfer configuration and data stream from the main memory of the system to the local data memories of the CGRA during the system startup time. A new configuration can also be loaded as required during the run-time making the

CGRA dynamically reconfigurable. Similarly, for continuous processing, new data can be loaded whenever required. The user can write the custom program flow which is essentially composed of three major portions. (1) Loading the configuration and data to be processed; (2) Enabling different context for cycle-accurate processing; and (3) Transferring the computed results from the CGRA to the main memory of the system. A context means a pattern of interconnections among the PEs and the operation to be performed by each PE at a particular cycle. These CGRAs have a track record of processing a number of computationally intensive signal processing related algorithms, for example, different length and types of FFTs, complex and real Matrix-Vector Multiplication (MVM), and Correlations. The implementation details can be found in [10, 12] and [11], respectively.

2.4 The Network-on-Chip

As described earlier, the communication over HARP is supported by an NoC, a network of nine nodes placed in a geometry of three rows and three columns. The NoC nodes in HARP can act as supervisors or workers based on the user-specified custom design. The overall architecture of HARP is template based, so the user can select particular nodes to be instantiated at the system design time. All the nodes can be integrated to either an RISC core or to the CGRAs mentioned in Sect. 2.3. A combined view of the NoC node integrated with RISC or the DMA is shown in Fig. 2.2. The NoC nodes function as routers for a master and two slave devices. If a node is connected to an RISC core, the master interface of the node is integrated with an RISC core and rest of the two slaves are interfaced with the instruction and data memory. The RISC can access these memories locally within the node and can also write to the network to send data to other devices connected to other nodes. A node can also be interfaced with the CGRA which is tightly integrated with a DMA device. The DMA device has master and slave ports which are interfaced with the master and slaves ports of the NoC node. The other slave interface of the node is integrated with the data memory. The DMA slave can be written from the NoC to invoke the DMA master, so to access the data memory, conduct CGRA related operations, and also to write to the NoC.

2.5 Hardware/Software Integration

The HardWare/SoftWare (HW/SW) integration is implemented by addressing special-purpose registers and dedicated shared-memory locations in the overall architecture. The addressing and data transfers are carried out by implementing special Application Programming Interfaces (APIs) which can be repetitively used

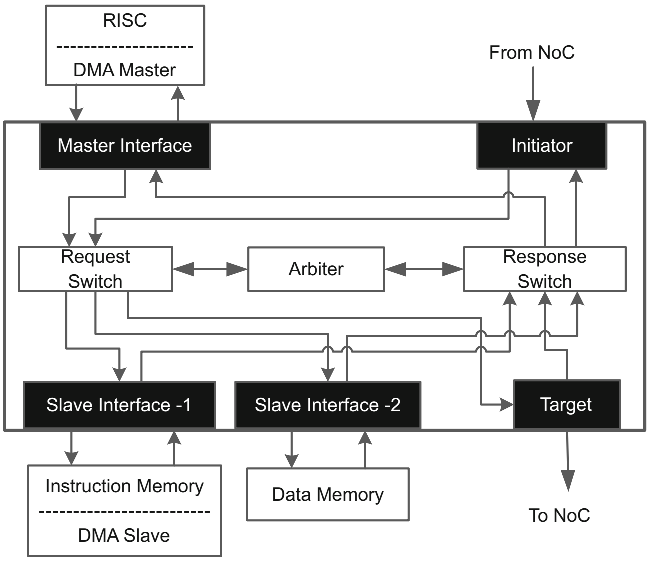


Fig. 2.2 An NoC node interfaced with RISC core or the DMA device [13]

by the programmer for algorithm mapping. The HW/SW integration is divided into three layers fundamentally required for processing, i.e., Loading the Configuration and Data, Execution, and Synchronization. The general description of APIs related to these procedures is presented in the following subsection.

2.5.1 Loading Configuration and Data

The major part of the configuration stream required for processing is generally loaded at system startup time. A configuration word is composed of address and data fields. The configuration can be injected straight into a CGRA and based on the address fields, it reaches the destination PEs after passing through a pipelined distribution infrastructure.

1. **Configuration Transfer**: The API developed for this task requires the base addresses of the configuration stream in the main memory and the configuration memories of the CGRA. Since the data and configuration memories operate at the same frequency across the platform, there is one word per clock cycle transfer between the nodes, therefore no particular synchronization protocol is required.
2. **Data Transfer**: The data transfer between two nodes is either between the data memories of the respective nodes or if the DMA of an associated CGRA transfers the data from the local memories of the CGRA to the data memory of another node. During the system startup time, the data from the supervisor node's

(RISC processor) data memory is fetched and sent to the data memory of a CGRA node. As explained in the previous point, this step does not need any synchronization, but the data from the node's memory has to be placed in a time-multiplexed manner in the local memories of the CGRA for custom processing. This task may require several clock cycles. To conduct this task, the supervising RISC processor sends a control word to the DMA of the associated CGRA node which starts reading the data from the memory and sends the data to the local memories of the CGRA. The control word contains the source and destination addresses, the number of read and write cycles, and information related to time-division multiplexing for the placement of data to be processed. When an internal data transfer is in progress within a node, i.e., between the node's data memory and the local data memories of the CGRA, the supervising RISC node should wait for the current data transfer to complete before invoking another internal data transfer through the DMA.

2.5.2 Context Enabling and Execution

As soon as the configuration and data transfer complete, the supervising RISC node enables CGRA execution by sending a control word for cycle-accurate processing. The control word contains information for the CGRA, i.e.,

1. Direction of data flow for processing, that is from the first local data memory of the CGRA to the other or vice-versa;
2. The context to be enabled;
3. Read and write latency, cycles, and offsets;
4. Total number of operation cycles.

During this task, different user-designed contexts are enabled in a sequence for a specified number of clock cycles for the custom computing of the target application.

2.5.3 Synchronization

There are two types of data transfers: (1) between data memories of NoC nodes and (2) between the node's data memory and the local memory of the CGRA and vice-versa. The data memories of the NoC nodes operate on the same clock frequency, so there is no specific synchronization required. The transfer rate is about one word per clock cycle after a latency equal to the number of nodes encountered in the transfer path. The transfer between the node's data memory and the data memory of the CGRA is established by the RISC core which sends a control word to the DMA device tightly integrated with the CGRA. The control word is packed with cycle-accurate information that enables data transfer in a specific time-multiplexed pattern. This internal transfer may require several clock cycles and the

RISC processor must not invoke another DMA transfer for this node, unless the current one is active. During this time, the RISC processor can send commands to other nodes and can be busy with subsequent tasks. The RISC processor before invoking a DMA transfer within a node sets a reserved location in its data memory. As soon as the DMA transfer completes, the DMA writes to the NoC to reset the same reserved location in the data memory of the supervising RISC processor. Meanwhile, the RISC processor continuously monitors the reserved location and as soon as it finds it being reset, it assigns another DMA transfer for the same node.

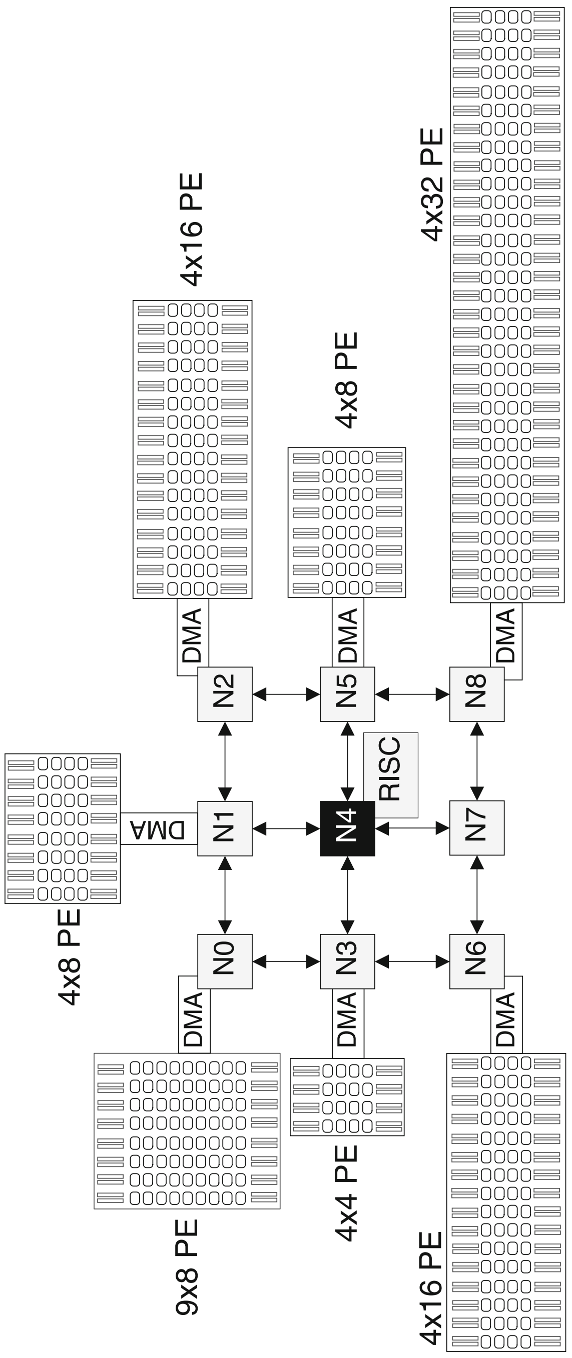
2.6 Heterogeneous Accelerator-Rich Reconfigurable Platform

Currently, HARP template has been used to instantiate four different instances. The first instance is for the proof-of-concept where each application-sized CGRA integrated with the node performs a computationally intensive signal processing task, i.e., 64- and 128-point FFT, 64th-order complex Matrix-Vector Multiplication (MVM), and 32nd-order real MVM. This instance is composed of six CGRA nodes and one central master node integrated with the RISC processor. Node-7 is not instantiated in this design to prove the template feature of the platform.

The second instance is relatively a realistic design problem, that is processing of four different streams of 64- and 128-point of FFT. The third instance is a hybrid architecture where the middle horizontal layer of the NoC is integrated with RISC cores and rest of the nodes are interfaced with CGRAs of application-specific sizes. This hybrid architecture was motivated to organize a distributed control among the RISC processors so that the program workload distribution for the CGRAs can be divided over the three processors. In this way, the tasks related to the distribution of configuration and data streams can be made faster than the instance presented in Fig. 2.3. The hybrid instance of HARP is shown in Fig. 2.4. The layered architecture of this instance is favorable for 3D IC integration. Simulation results have shown that the RISC processors are active most of the time, therefore contributing a major part to the average dynamic power dissipation. In this context, the layer of the RISC cores can be spatially closer towards the heat-sink and the airflow than the layers of the 3D IC containing the CGRAs.

Another instance of HARP was subjected to Dynamic Frequency Scaling (DFS) while considering the computational workload to be equally distributed over all the CGRAs. In this context, the central supervising node integrated to the RISC processor continuously monitors the execution time of each CGRA and then issues instructions to fine tune the operating frequency of the CGRAs until they approach the same execution-time constrained as a user-stated goal. This task is achieved by implementing a feedback control system in both HW and SW. The HW part constitutes a dedicated frequency control register and software part includes the API addressing the register as soon as the frequency of a particular CGRA needs

Fig. 2.3 A proof-of-concept HARP instance [13]



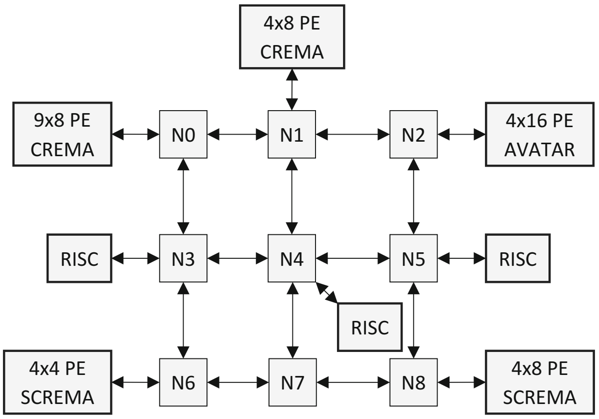


Fig. 2.4 A hybrid instance of HARP [14]

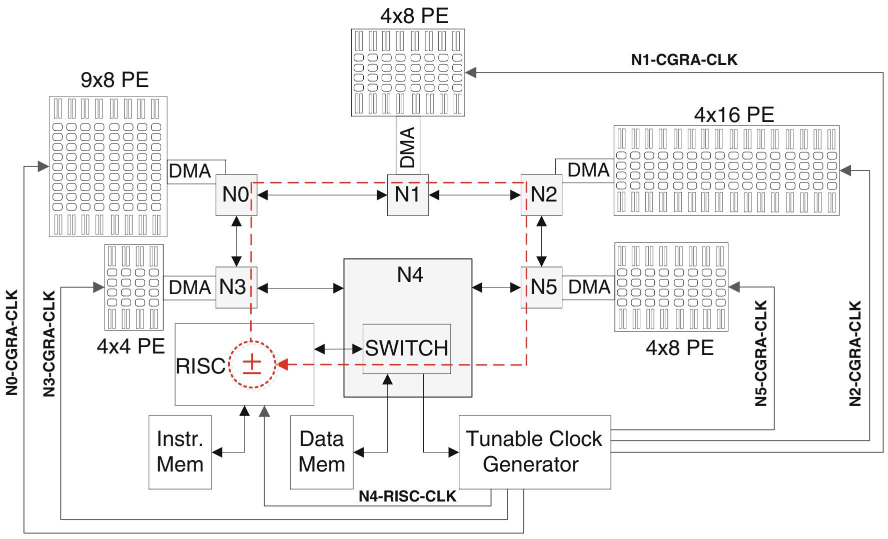


Fig. 2.5 An instance of HARP subjected to DVFS for workload balancing [13]

to be updated. The CGRAs are performing signal processing related tasks similar to the ones mentioned for the first instance of HARP. Figure 2.5 shows the HARP instance being subjected to DFS using a feedback control loop specified in dashed red colored lines. A larger CGRA is designed to process a large amount of data and its overall execution time is relatively lower than the smaller CGRAs. However, if all the CGRAs over the platform are supposed to exchange data with each other, they are required to process their respective tasks in the same time frame which is needed by the most work-loaded CGRA. In this case, the RISC processor while identifying the worst performing CGRA tunes the clock frequency of other CGRAs, so that all the CGRAs perform closer to the performance of the worst-case. The DFS applied during this process shows that this particular workload balancing scheme mitigates power dissipation. Figure 2.5 also shows a frequency tunable clock generator which is a piece of non-synthesizable VHDL, but when actually implemented on a printed circuit board, or an FPGA, Phase-Locked Loop (PLL) based clock generators need to be adopted.

2.7 Application Mapping

The programming of an instance of HARP can be broadly categorized at two different levels. The first one is at the level of the CGRA, where the application data flow is organized based on the spatial location of the processing elements and establishing the interconnections between them. The second is purely writing the software for the application program flow and control. It includes calling the API functions with specific parameters to address the DMA and the CGRA control registers for cycle-accurate processing. The configuration streams are generated by the GUI tool (mentioned in Sect. 2.3) upon completion of the design of a context which can be loaded at the system start time. In fact, for any instance of HARP, all the configuration streams related to integrated CGRAs are loaded before the overall system starts processing. For any multi-core platform, it is important to demonstrate that the cores can exchange data with each other and also process the data simultaneously and independently of each other. The proof of this concept is implemented so that the CGRA integrated at Node N0 of HARP instance depicted in Fig. 2.3 processes 64-point FFT in radix-4 scheme and then transfers the results to the data memory of node N1. The DMA engine in node N1 fetches this data and sends it to the local data memory of node N1 CGRA which subsequently processes 64th-order complex MVM on the 64-point vector data obtained from FFT processing. The node N1 transfers the results of complex MVM processing back to node N4. The transfer and processing of data between these nodes show that nodes can exchange data with each other for dependent processing. The other nodes N2, N3, N5, N6, and N8 are simultaneously processing the data and independently of each other. The node N2 CGRA processes 128-point FFT in mixed-radix(2, 4) scheme while rest of the CGRA nodes are processing integer MVM of different orders. The second instance of HARP was processing only FFT for IEEE-802.11n demodulator case which requires four different streams of 128-point data to be processed simultaneously. In this case, nodes N1, N2, N5, and N8 are instantiated with mixed-radix(2, 4) FFT CGRAs to perform this task. The details of application mapping on these first two instances of HARP are well described in [13]. Table 2.1 shows the type of the CGRA integrated with a node of HARP and the application mapped on it. The table is a combined representation of the first two instances of HARP, i.e., the homogeneous (hom) and the heterogeneous (het).

The third instance of HARP is a hybrid platform which has a flavor of both homogeneous and heterogeneous cores. In it, the nodes N3, N4, and N5 are

Table 2.1 Types and sizes of the CGRAs integrated with the nodes of HARP and the applications mapped on them [13]

Node	PEs	CGRA type	Kernel
HARP-het-N0	8×9	CREMA	Radix-4, 64-point FFT
HARP-het-N1	4×8	CREMA	Complex MVM, 64th-order
HARP-het-N2	4×16	AVATAR	Radix-(2, 4), 128-point FFT
HARP-het-N3	4×4	SCREMA	Real MVM 32nd-order
HARP-het-N5	4×8		
HARP-het-N6	4×16		
HARP-het-N8	4×32		
HARP-hom-(N0, N2, N6, N8)	4×16	AVATAR	128-point FFT

In the table, hom and het stand for homogeneous and heterogeneous, respectively

integrated with COFFEE RISC cores and rest of the nodes are integrated with CGRAs of application-specific sizes. The three RISC cores communicate with each other to establish distributed supervisory control among all the CGRA cores. The RISC at node N3 supervises the CGRAs at node N0 and N6, the N4 does it for N1 and N7 CGRAs while the N5 performs supervision for N2 and N8 CGRAs. The synchronization and control are established as the RISC cores write and read flags in each others shared-memory locations. Meanwhile, the CGRAs on the completion of a task write acknowledgments in specific memory locations of their supervising RISC cores. The advantage of using the three RISC cores is the simultaneous configuration transfer to all the cores which otherwise in the case of the first instance of HARP is the responsibility of a single RISC core—a situation where the NoC bandwidth was under-utilized. The application mapping in this instance of HARP is similar to the two instances and the CGRA nodes are processing different lengths and types of FFTs, complex and real MVM [13, 14].

The fourth instance of HARP was designed as a six core platform organized in a geometry of two rows and three columns. The nodes N0, N1, N2, N3, and N5 are integrated with CGRAs while the node N4 contains the RISC core acting as a supervisor. The platform is subjected to Dynamic Frequency Scaling (DFS) in an effort to equalize performance on all the CGRAs, thereby mitigating instantaneous dynamic power dissipation. The RISC has a special-purpose counter which counts the cycles elapsed for the execution of an algorithm by the CGRA and stores the measurement in the data memory. This counter is not the part of the main datapath of the processor, so if the RISC processor is busy in general-purpose tasks, its counting remains unobstructed. Once the clock-cycle count is measured for all the CGRA integrated to the HARP platform, then the supervising RISC processor searches for the largest number (worst-case). Based on this worst-case, the RISC sends the data to its frequency control register which updates the operating frequencies of all the CGRAs other than the worst performing. This process is made iterative so in the next iteration, the clock-cycle count is measured for all the CGRAs again and fine tuning is performed in form of a feedback loop. In this way, the CGRAs gradually approach the performance of the worst-case. Once all the CGRAs degrade their execution time, the instantaneous power consumption is reduced, therefore the overall heat dissipation. This form of DFS is implemented for the proof-of-concept when all the cores on a platform are supposed to exchange data with each other and it is not a requirement if one core needs to operate faster than the other. However, the overall operating frequency control system is software-defined which allows any custom power mitigation technique to be implemented in software.

2.8 Measurement and Estimation

This section presents the numbers related to different performance metrics for all the four instances of HARP platform, i.e., clock cycles required to perform data transfers and execution, resource utilization on the FPGA device, synthesis frequencies and estimates related to power dissipation and energy consumption. Table 2.2 presents clock cycles required for transfer between the data memories of NoC nodes and from the data memory of an NoC to one of the local data memories of the CGRA. The table also shows the cycles required for execution by the CGRA.

Table 2.3 shows the DSP resource consumption on the FPGA at the node level. The Stratix-V FPGA device has 18-bit DSP elements and to implement one 32-bit DSP element, two 18-bit DSP resources are required. For example, in the first row of the table, 12 32-bit multipliers consume 24 DSP resources. It can also be observed, the DSP resource consumption increases as the complexity of algorithm increases, for example, radix-(2, 4) FFT accelerator requires 56 DSP elements than the 24 required by the radix-4 FFT algorithm.

Table 2.4 shows the resources required by the first two instances of HARP, i.e., hom and het on the FPGA device. As it has been discussed before that the HARP-het and HARP-hom are the instances with 7 and 4 nodes, respectively, but the Adaptive Logic Module (ALM) consumption by homogeneous version is significantly higher

Table 2.2 The number of clock cycles for data transfer and execution at different stages measured with reference to clock frequency of the supervising node [13]

Node-to-node	D. mem to D. mem	D. mem to CGRA	Trans. total	Exe. total
het(N4-N0)	1017	659	1676	420
het(N0-N1)	1036	446*	1482	457
het(N1-N4)	–	448*	–	–
het(N4-N2)	2042	1033	3075	571
het(N4-N3)	75,622	–	75,622	728
het(N4-N5)				609
het(N4-N6)				340
het(N4-N8)				211
hom(N4- (N0, N2, N6, N8))	4× 2042	4× 1033	4× 3075	587

D. Mem, Trans., and Exe. stand for Data Memory, Transfer, and Execution, respectively
* Sign shows data transfer from CGRA to node's data memory

Table 2.3 DSP resource consumption on the FPGA device at the node level for both of the versions of HARP [13]

Node	Accelerator type	Multipliers(32-bit)	DSPs
HARP het			
Node-0	Radix-4 FFT	12	24
Node-1	Complex MVM	12	24
Node-2	Radix-(2, 4) FFT	28	56
Node-3	Real MVM	4	8
Node-4	RISC Core	6	12
Node-5	Real MVM	8	16
Node-6	Real MVM	16	32
Node-7	–	–	–
Node-8	Real MVM	32	64
Total	–	118	236
HARP hom			
Node-0	Radix-(2, 4) FFT	28	56
Node-2	Radix-(2, 4) FFT	28	56
Node-4	RISC Core	6	12
Node-6	Radix-(2, 4) FFT	28	56
Node-8	Radix-(2, 4) FFT	28	56
Total	–	118	236

Table 2.4 Resource utilization by HARP instances on Stratix-V FPGA device (5SGXEA4H1F35C1) [13]

HARP het		
ALMs	78,845/158,500	50 %
Registers	64,436/634,000	10 %
Memory bits	21,000,928/38,912,000	54 %
DSP	236/256	92 %
HARP hom		
ALMs	120,823/158,500	76 %
Registers	67,475/634,000	10.6 %
Memory bits	13,679,616/38,912,000	35 %
DSP	236/256	92 %

than heterogeneous version. The ALM consumption depends on the complexity of the algorithm mapped on the CGRA and in comparison, the complexity of radix-(2, 4) FFT algorithm is the highest and having four of those makes the hom version to have the largest number of ALMs despite the fact that it has less number of CGRAs integrated than the het version.

The first two instances of HARP operate on a single clock source. The HARP-het version synthesizes at 214.64 MHz at 0 °C and 186.22 MHz at 85 °C while the functional voltage is set to 900 mV. These measurements were observed for slow timing model. The fast time model shows 274.65 MHz at 0 °C and 251.0 MHz at 85 °C. The HARP-hom instance synthesizes for slow timing model at 208.42 MHz at 0 °C and 196.89 MHz at 85 °C with the functional voltage equal to 900 mV. The operating frequencies for fast timing model are observed to be 299.31 MHz at 0 °C and 272.63 MHz at 85 °C. The HARP-hom processes four streams of 128-point FFT within 587 CC ×1/186.22 MHz = 3.15 µs while considering the operating frequency of 186.22 MHz and the cycles presented in Table 2.2, therefore satisfying FFT execution-time constraints of IEEE-802.11n standard.

Table 2.5 shows the dynamic power and energy consumption by the CGRAs integrated with the nodes of HARP-hom and -het versions. It is evident that a larger CGRA in size has more logic to toggle per unit time and therefore dissipates more power but offers lower execution times as in the case of processing 32nd-order MVM. As energy is the product of power dissipation and execution time, so in this specific case, the dynamic power dissipation does not appear to deviate considerably.

2.9 Evaluation and Comparisons

The HARP instances are synthesized for a 28 nm FPGA device. According to a document produced by Altera Corporation, the average operating frequency of a Stratix FPGA device scales by 20 % as the process size scales [3], which means that a 28 nm FPGA is capable to operate with a 20 % higher operating frequency than a 40 nm FPGA device. On the other hand, Xilinx. Inc claims to provide 50 % higher operating frequency for the synthesized designs with their latest 28 nm FPGA device in comparison to their 40 nm one, but such a performance comes at the cost of increasing the on-chip resources by 2× [24]. The designer can implement 2× parallel resources to achieve 2× speed-up. Since this argument is out of context, therefore Altera's scaling trend can also be estimated for Xilinx devices as well. A 90 nm ASIC implementation shows 3.4× to 4.6× (average 4.0×) higher operating frequency than an equivalent FPGA implementation as well as consumes 14× lesser dynamic power dissipation [15]. Based on this information, cross-technology comparisons are established in comparison to a HARP instance as shown in Table 2.6. The state-of-the-art platforms selected for comparisons are already described in Sect. 2.2.

The node N0 of HARP-het version computes 64-point FFT in 420 CC ×1/100.0 MHz = 4.2 µs. NineSilica computes the same task in 10.3 µs on a 40 nm FPGA using all of its RISC nodes. Since HARP-het is synthesized on a 28 nm FPGA device, scaling on NineSilica's performance is required to establish comparisons. After scaling, NineSilica is estimated to compute the same task in 8.25 µs, hence HARP-het shows a 1.96× gain in comparison.

A homogeneous MIMD MPSoC delivers a performance of 19.2 GOPS while being synthesized on a 90 nm FPGA. After appropriate scaling to establish comparisons, the performance estimate increases to 26.88 GOPS. HARP-het performs at 40.8 GOPS and shows 1.51× of gain in comparison.

A homogeneous MPSoC platform called P2012 is synthesized on a 28 nm CMOS technology. The performance gap ratios discussed above between an ASIC and FPGA implementation are supposed to remain similar even between a 28 nm ASIC

Table 2.5 Estimation of dynamic power/energy for each CGRA integrated in a node for both of HARP instances [13]

Node CGRA size Other HW	Algorithm (Type)	Dynamic power (mW)	Active time (µs)	Dynamic energy (µJ)
HARP-het				
N0 8×9 PE	Radix-4 FFT (64-point)	129.68	4.20	0.54
N1 4×8 PE	Complex MVM (64-Order)	116.75	4.57	0.53
N2 4×16 PE	Radix-(2, 4) FFT (128-point)	213.12	5.71	1.21
N3 4×4 PE	Real MVM (32nd order)	61.55	7.28	0.44
N4 RISC	General flow control	59.73	–	–
N5 4×8 PE	Real MVM (32nd order)	98.97	6.09	0.60
N6 4×16 PE	Real MVM (32nd order)	175.80	3.40	0.59
N7	–	0.37	–	–
N8 4×32 PE	Real MVM (32nd order)	327.36	2.11	0.69
NoC integration	–	7.12	–	–
Logic	–	302.07	–	–
Total	–	1492.52	–	4.6
HARP-hom				
N0 4×16 PE	Radix-(2, 4) FFT (128-point)	214.68	5.87	7.11
N2 4×16 PE	Radix-(2, 4) FFT (128-point)	214.35		
N4 RISC	General flow control	59.75		
N6 4×16 PE	Radix-(2, 4) FFT (128-point)	212.97		
N8 4×16 PE	Radix-(2, 4) FFT (128-point)	213.89		
NoC integration	–	0.67		
Logic	–	295.64		
Total	–	1211.95		

Table 2.6 Multiple performance metric comparison with HARP Implementation at 100.0 MHz on a 28 nm FPGA [13]

Platform/technology	Performance metric	Platform's value (PV)	Scaled PV	HARP-het	
				Value	Gain
NineSilica [2] /FPGA 40 nm	FFT execution time (μs)	10.3	Exe. time − Exe. time×20% SU_fTfs = 10.3 − 10.3 × 0.2 = 8.24	4.2	1.96×
[5] /FPGA 90 nm	GOPS	19.2	PV×40 % SU_fTfs = 19.2 × 1.4 = 26.88	40.8	1.51×
P2012/ [18] CMOS 28 nm	GOPS/mW	0.04	PV×SD_atfs×Ps = 0.04 × 1/4 × 1/2 = 0.005	0.0153	3.0×
MORPHEUS /[23] CMOS 90 nm	GOPS/mW	0.02	(PV×SD_aTfs×Ps)×60 % SU_fTfs = (0.02 × 1/4 × 1/2) × 1.6 = 0.004	0.0153	3.8×

In the table, SU_fTfs, SD_aTfs, and Ps stand for Speed-Up for FPGA to FPGA scaling, Speed-Down for ASIC to FPGA scaling, and Power scaling, respectively and a 28 nm FPGA implementation. The gap shows that performance degrades by a factor of 4 but the dynamic power dissipation increases by a factor of 14 on an FPGA implementation than the ASIC for the same process size. Considering the worst-case scenario, where the static power can be equal to dynamic power dissipation, the factor of 14 can be almost equal to 2. In such a situation, the scaled performance of P2012 is estimated to be 0.005 GOPS/mW as shown in Table 2.6 while HARP shows a performance gain of 3.0×.

The 0.02 GOPS/mW performance of MORPHEUS at 90 nm CMOS can be first scaled for a 90 nm FPGA and then be scaled down to a 28 nm FPGA to establish comparison with HARP. The scaling is also shown in the last row of Table 2.6. In comparison, HARP shows a performance of 3.8×.

2.10 Conclusions

In this chapter, an extra large-scale template-based homogeneous/heterogeneous accelerator-rich architecture is presented which is capable to process massively parallel computationally intensive signal processing related algorithms. In particular, three instances from the architecture's template were generated in which one is completely heterogeneous, the other one homogeneous, and the third one is hybrid in nature. All of these instances were tested for a proof-of-concept to verify the functionality and architectural features. From the discussion presented in the overall and particularly in the section of evaluation and comparisons, the platform outperforms many state-of-the-art platforms in several performance metrics due to its high density in the number of on-chip computational resources and efficient utilization of the communication channel, i.e., a Network-on-Chip.

References

1. Ahonen, T., ter Braak, T.D., Burgess, S.T., GeiBler, R., Heysters, P.M., Hurskainen, H., Kerkhoff, H.G., Kokkeler, A.B.J., Nurmi, J., Raasakka, J., Rauwerda, G.K., Smit, G.J.M., Sunesen, K., van Zonneveld, H., Vermeulen, B., Zhang, X.: CRISP: cutting edge reconfigurable ICs for stream processing. In: Cardoso, J.M.P., Hübner, M. (eds.) Reconfigurable Computing, pp. 211–237. Springer, New York (2011). ISBN: 978-1-4614-0060-8
2. Airoldi, R., Garzia, F., Anjum, O., Nurmi, J.: Homogeneous MPSoC as baseband signal processing engine for OFDM systems. In: International Symposium on System on Chip (SoC), 2010, pp. 26–30, Sept 2010. doi:10.1109/ISSOC.2010.5625562
3. Altera Product Catalog. Release Date: July 2014, Version 15.0, p. 2 (2015). www.altera.com
4. Bell, S., Edwards, B., Amann, J., Conlin, R., Joyce, K., Leung, V., MacKay, J., Reif, M., Liewei, B., Brown, J., Mattina, M., Chyi-Chang, M., Ramey, C., Wentzlaff, D., Anderson, W., Berger, E., Fairbanks, N., Khan, D., Montenegro, F., Stickney, J., Zook, J.: TILE64 - processor: a 64-Core SoC with mesh interconnect. In: IEEE International Solid-State Circuits Conference (2008). ISSCC 2008. Digest of Technical Papers, pp. 88–598, 3–7 Feb 2008

5. Bonnot, P., Lemonnier, F., Edelin, G., Gaillat, G., Ruch, O., Gauget, P.: Definition and SIMD implementation of a multi-processing architecture approach on FPGA. In: Proceedings of Design, Automation and Test in Europe (DATE '08), pp. 610–615. ACM, New York, NY (2008)
6. Campi, F., Deledda, A., Pizzotti, M., Ciccarelli, L., Rolandi, P., Mucci, C., Lodi, A., Vitkovski, A., Vanzolini, L.: A dynamically adaptive DSP for heterogeneous reconfigurable platforms. In: Proceedings of Design Automation and Test in Europe (DATE '07), pp. 9–14. EDA Consortium, San Jose, CA (2007)
7. Campi, F., Konig, R., Dreschmann, M., Neukirchner, M., Picard, D., Juttner, M., Schuler, E., Deledda, A., Rossi, D., Pasini, A., Hübner, M., Becker, J., Guerrieri, R.: RTL-to-layout implementation of an embedded coarse grained architecture for dynamically reconfigurable computing in systems-on-chip, SOC 2009. In: International Symposium on System-on-Chip, 2009, pp. 110–113, 5–7 Oct (2009)
8. Dennard, R., Gaensslen, F.H., Rideout, V.L., Bassous, E., LeBlanc, A.R.: Design of ion-implanted MOSFET's with very small physical dimensions. In: JSSC, Oct (1974)
9. Garzia, F., Hussain, W., Nurmi, J.: CREMA, a coarse-grain re-configurable array with mapping adaptiveness. In: Proceedings of 19th International Conference on Field Programmable Logic and Applications (FPL 2009). Prague, IEEE, New York (2009)
10. Garzia, F., Hussain, W., Airoldi, R., Nurmi, J.: A reconfigurable SoC tailored to software defined radio applications. In: Proceedings of 27th Norchip Conference, Trondheim, NO (2009)
11. Hussain, W., Garzia, F., Ahonen, T., Nurmi, J.: Designing fast Fourier transform accelerators for orthogonal frequency-division multiplexing systems. J. Signal Process. Syst. Signal Image Video Technol. **69**, 161–171 (2012). Springer, Berlin
12. Hussain, W., Ahonen, T., Nurmi, J.: Effects of scaling a coarse-grain reconfigurable array on power and energy consumption. In: 2012 International Symposium on System on Chip (SoC), pp. 1-5. Tampere (2012). doi:10.1109/ISSoC.2012.6376372
13. Hussain, W., Airoldi, R., Hoffmann, H., Ahonen, T., Nurmi, J.: HARP2: an X-scale reconfigurable accelerator-rich platform for massively-parallel signal processing algorithms. J. Signal Process. Syst. **85**, 341–353 (2015). Springer, New York
14. Hussain, W., Hoffmann, H., Ahonen, T., Nurmi, J.: Design of a hybrid multicore platform for high performance reconfigurable computing. In: Nordic Circuits and Systems Conference (NORCAS): NORCHIP & International Symposium on System-on-Chip (SoC), pp. 1–8. Oslo (2015). doi:10.1109/NORCHIP.2015.7364376
15. Ian, K., Rose, J.: Measuring the gap between FPGAs and ASICs. IEEE Trans. Comput. Aided Des. Integr. Circuits Syst. **26**(2), 203–215 (2007). doi:10.1109/TCAD.2006.884574
16. Lodi, A., Mucci, C., Bocchi, M., Cappelli, A., De Dominicis, M., Ciccarelli, L.: A Multi-context pipelined array for embedded systems. In: International Conference on Field Programmable Logic and Applications, Aug 2006. FPL'06, pp. 1–8 (2006)
17. Mattson, T.G., Riepen, M., Lehnig, T., Brett, P., Haas, W., Kennedy, P., Howard, J., Vangal, S., Borkar, N., Ruhl, G., Dighe, S.: The 48-core SCC processor: the programmer's view. In: Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis (SC '10). IEEE Computer Society, Washington, DC (2010)
18. Melpignano, D., Benini, L., Flamand, E., Jego, B., Lepley, T., Haugou, G., Clermidy, F., Dutoit, D.: Platform 2012, a many-core computing accelerator for embedded SoCs: performance evaluation of visual analytics applications. In: Proceedings of 49th Annual Design Automation Conference (DAC '12), pp. 1137–1142. ACM, New York, NY (2012)
19. Taylor, M.B.: Is dark silicon useful: harnessing the four horsemen of the coming dark silicon apocalypse. In: Proceedings of the 49th Annual Design Automation Conference (DAC-2012), pp. 1131–1136. ACM, New York (2012)

20. Taylor, M.B., Kim, J., Miller, J., Wentzlaff, D., Ghodrat, F., Greenwald, B., Hoffman, H., Johnson, P., Lee, J.-W., Lee, W., Ma, A., Saraf, A., Seneski, M., Shnidman, N., Strumpen, V., Frank, M., Amarasinghe, S., Agarwal, A.: The raw microprocessor: a computational fabric for software circuits and general-purpose programs. Micro IEEE **22**(2), 25–35 (2002)
21. Venkatesh, G., Sampson, J., Goulding, N., Gracia, S., Bryksin, V., Martinez, J.L., Swanson, S., Taylor, M.B.: Conservation cores: reducing the energy of mature computations. In: ASPLOS'10, pp. 205–218 (2010)
22. Voros, N.S., Rosti, A., Hübner, M.: Flexeos embedded FPGA solution. In: Dynamic System Reconfiguration in Heterogeneous Platforms. Lecture Notes in Electrical Engineering, vol. 40, pp. 39–47. Springer Netherlands, Berlin (2009). ISBN 978-90-481-2426-8
23. Voros, N.S., Hübner, M., Becker, J., Kühnle, M., Thomaitiv, F., Grasset, A., Brelet, P., Bonnot, P., Campi, F., E. Schüler, H. Sahlbach, S. Whitty, R. Ernst, E. Billich, C. Tischendorf, U. Heinkel, F. Ieromnimon, D. Kritharidis, A. Schneider, J. Knaeblein, W. Putzke-Röming. MORPHEUS: a heterogeneous dynamically reconfigurable platform for designing highly complex embedded systems. ACM Trans. Embed. Comput. Syst. **12**(3), 33 pp. (2013).Article 70
24. Wu, X., Gopalan, P.: Xilinx next generation 28 nm FPGA technology overview. White Paper: 28nm Technology, July 23 Version 1.1.1, p. 5 (2013). www.xilinx.com

Chapter 3
The CoreVA-MPSoC: A Multiprocessor Platform for Software-Defined Radio

Gregor Sievers, Boris Hübener, Johannes Ax, Martin Flasskamp, Wayne Kelly, Thorsten Jungeblut, and Mario Porrmann

3.1 Introduction

The advancement in performance of mobile devices goes hand in hand with increasing demand for communication bandwidth. In the past decade an almost unmanageable number of different wireless communication standards have emerged. In addition, the complexity of many of those standards has led to a steadily increasing demand for high performance of modem signal processing of the devices. The computational throughput required for processing state-of-the-art algorithms ranges from 100 GOPS for the current generation (3.5G) to 1000 GOPS for next generation communication systems (4G and above) [7, 35]. Additionally, a high number of different configurations per single protocol expand the huge design-space to be taken into account by developers. Seamless handover between the different protocols has to be supported and a fast run-time reconfiguration is essential.

Besides the usual physical requirements such as low silicon area or energy consumption, new requirements such as programmability and flexibility arise. In the history of mobile communication, multi-standard communication-interfaces relied on multiple fixed application-specific implementations. The need to support a huge number of different standards can lead to a large increase of device complexity if each is implemented separately in silicon. While a fine-tuned power management may avoid the drawback of a possible increase of the power consumption, the large silicon area may undo these efficiencies. Therefore, as new standards emerge or

G. Sievers • B. Hübener • J. Ax • M. Flasskamp • T. Jungeblut (✉) • M. Porrmann
Cognitronics and Sensor Systems Group, CITEC, Bielefeld University, Bielefeld, Germany
e-mail: jungeblut@cit-ec.uni-bielefeld.de

W. Kelly
Science and Engineering Faculty, Queensland University of Technology, Brisbane, QLD, Australia

W. Hussain et al. (eds.), *Computing Platforms for Software-Defined Radio*,
DOI 10.1007/978-3-319-49679-5_3

changes in standards evolve, reprogrammable devices allow us to realize silicon optimized chips that need only meet the resource needs of the most complex communication standard. In addition, it allows a reduction in time-to-market by providing the ability to easily adapt to changes of standards. Additional features can be implemented in late design cycles and even post-silicon bugs can be handled more easily. Although this challenge mainly applies to high volume mobile devices, flexible hardware architectures can also be used in stationary base stations especially when including pico-cell concepts of state-of-the art protocols like LTE.

As the computational requirements of baseband processing steadily increasing, the performance demands of the application layer also add to the problem. Multimedia applications such as 3D graphics processing or video en-/decoding (e.g., H.265) increase performance demand even more. Uniform architectures are capable to handle baseband processing together with the application layer. To allow for the required flexibility on one hand and a sustainable performance on the other hand, several approaches have evolved towards SDR-based architectures in the past. Research moved from ASIC-centered blocks towards heterogeneous DSP-centered designs aided by specialized hardware accelerators. Such architectures provide a higher flexibility but still lack in terms of programmability, flexibility, and scalability for new communication standards in the future. Flexibility and ease of programming is enabled by use of homogeneous many-core architectures with uniform communication infrastructure as they allow for simpler and more generic programming models. The programmability of such architectures enables the efficient mapping of the particular baseband core algorithms as well as high-level applications like operating systems. Due to their regular structure homogeneous many-cores also allow us to higher levels of performance by merely increasing the number of processor cores. Furthermore, a uniform communication infrastructure and a general programmability should facilitate the use of high-level mapping strategies to allow for the required flexibility while maintaining the maximum possible efficiency.

Recently, various homogeneous many-core architectures designed for high performance signal processing have been proposed [6, 8, 11]. While their sustained performance makes them suitable for baseband signal processing, they are impractical for use in mobile user equipment as their energy needs exceed the typical mobile restriction of 1 W [7, 35]. They are therefore more suitable for software-defined radio in stationary base stations with a wire-connected power supply.

In this work, we try to depict the future path of performing SDR baseband processing via the use of homogeneous many-core architectures. To illustrate the architectural challenges, we present our CoreVA-MPSoC as an example of an embedded hierarchical multiprocessor architecture for SDR processing. The massive parallelism is introduced at different levels. Our basic building block is the resource-efficient VLIW[1] processor CoreVA providing fine-grained concurrency at the instruction level. Multiple CoreVA CPUs are then combined within a CPU

[1]VLIW: Very Long Instruction Word.

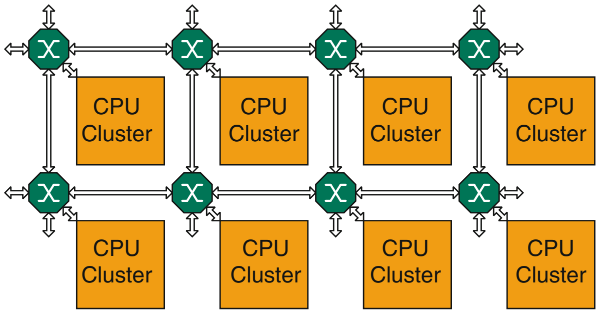


Fig. 3.1 Hierarchical CoreVA-MPSoC with 4×2 mesh NoC. ©IEEE [32]

cluster and connected via a high speed, low latency interconnect. On the next level of hierarchy a Network-on-Chip[2] is then used to connect together any number of CPU clusters all located on a single chip (cf. Fig. 3.1).

Mapping an SDR application to dozens or hundreds of CPUs under throughput, latency, energy, and memory constraints is a challenging task. Therefore, in addition to our MPSoC hardware architecture, we present an MPSoC compiler for streaming applications.

This chapter is organized as follows. In Sect. 3.2 we present related work and the current state of the art. Section 3.3 describes our CoreVA-MPSoC hardware architecture including the different levels of the hierarchy: CPU core, cluster, and NoC. Section 3.4 shows synthesis results using a 28 nm FD-SOI standard-cell technology. In Sect. 3.5, we propose our software tool-chain including VLIW and MPSoC compiler, simulator, and MPSoC programming model. In Sect. 3.6 we present results of mapping typical SDR algorithms to both a single CoreVA and different configurations of MPSoCs. The work is concluded in Sect. 3.7 with a summary.

3.2 Related Work

Many different multicore architectures for software-defined radio have been proposed by academia and industry in the last years. A short overview is given in the following.

The MAGALI [7] is a heterogeneous SDR MPSoC with a 15-router NoC. The NoC routers are asynchronous to reduce dynamic power consumption. Five general purpose VLIW cores perform channel estimation and MIMO[3] decoding. Four

[2]NoC: Network-on-Chip.

[3]MIMO: Multiple Input, Multiple Output.

NoC nodes integrate 1 Mbit memory each and a DMA[4] engine for data transfers. An ARM1176 CPU with floating-point support is used for control and scheduling. In addition, the MPSoC features four Fast Fourier Transform (FFT), three channel decoding, and two bit-level hardware accelerators. The MPSoC has a peak performance of 37.3 GOPS. A chip prototype manufactured in 65 nm consumes 477 mW while executing a 4×2 LTE receiver with 10.8 MBit/s throughput.

Noethen et al. [24] present Tomahawk2, a heterogeneous MPSoC with 20 CPU cores targeting 4×4 MIMO LTE baseband SDR. The hierarchical MPSoC contains two CPU clusters with four general purpose CPUs (Duo-PE) each and two additional CPU clusters with I/O interfaces, memory controller, and application-specific cores. A Duo-PE consists of a fourfold 16 bit SIMD[5] core and a Tensilica LX4 CPU with floating-point support. Tasks can be scheduled dynamically either in software or via a dedicated hardware unit (CoreManager). The MPSoC features hierarchical power management. The area requirements of the chip are 36 mm^2 and it consumes 480 mW in a 65 nm process. Peak performance is 105 GOPS and 3.8 GFLOPS.

The Infineon X-GOLD SDR20 [29] is an SDR baseband processor for communication standards such as UMTS, HSPA, and LTE. The SDR20 consists of 12 SIMD cores for baseband processing and additional hardware accelerators. In addition, an ARM-based subsystem can execute a protocol stack. A SIMD core contains four processing elements which provide 16 bit and 32 bit integer calculations, 4 kB scratchpad memory, a DMA controller, and an additional scalar core to handle control and synchronization. Four SIMD cores, two additional scalar cores, and 512 kB shared memory are grouped in a cluster and connected via a multi-layer interconnect. Three of those clusters, an additional accelerator cluster, and the ARM subsystem are connected via a full crossbar. Manufactured in a 65 nm process, the SDR20 operates at 300 MHz (SIMD cores) and 412 MHz (ARM core). Peak performance of the SDR subsystem is 17.7 GOPS.

The SDR MPSoCs that we presented in this section feature programmable DSP-like processing cores and specialized hardware accelerators. Using today's chip technology, these accelerators are mandatory to meet performance and energy requirements. Considering the execution of an increasing number of communication standards on a single MPSoC and due to improvements in semiconductor technology, more and more homogeneous many-core architectures like our CoreVA-MPSoC will emerge.

The Kalray MPPA-256 [8] is a hierarchical 288-core MPSoC targeting embedded applications. Each CPU cluster contains a system CPU, 16 processing CPUs, and 2 MB shared L2 memory.

Adapteva's Epiphany E64G401 [11] is a multiprocessor with 2 MB distributed shared memory and 64 CPU cores. The maximum operation frequency is 800 MHz.

[4]DMA: Direct Memory Access.

[5]SIMD: Single Instruction, Multiple Data.

The interconnect fabric features a 3-layer 2D-mesh NoC. The Epiphany does not introduce a cluster-level hierarchy but solely relies on NoC communication.

The Cadence Tensilica Xtensa processor-suite [10, 13] provides a customizable CPU architecture that targets a variety of different fields of applications. The processor-suite is based on a configurable 32-bit RISC[6] processor core that can be used as an embedded controller, e.g., for memories and interfaces. This controller can be extended by a multiplicity of DSP IP cores that enable the processor to be a high performance data-plane processing unit (DPU). The DSP cores mainly provide VLIW and SIMD extensions of different data widths to speedup the baseband and RF signal processing in, e.g., LTE and LTE-Advanced wireless modems. The latest Cadence Tensilica DSPs provide fourfold VLIW instructions and 32-way SIMD calculations on vectors with up to 512 bits. In combination with specialized vector-MAC[7]-units this enables the processor can calculate up to 128 MAC operations per cycle.

3.3 The CoreVA-MPSoC Architecture

This section provides architectural details about CoreVA CPU core, CPU cluster, and NoC that represent the three main building blocks of the CoreVA-MPSoC. The architecture is highly configurable at design-time which leads to a huge design-space but carefully to be explored to select the configuration with the highest resource efficiency for the execution of SDR applications.

3.3.1 CoreVA VLIW CPU

The configurable 32 bit VLIW CPU CoreVA is designed to be used as an embedded processor for signal processing purposes. It has a typical Harvard architecture with separate instruction and data memories. The CoreVA is based on an ARM-like RISC instruction set that consists of 41 scalar and 13 vector instructions. The CoreVA comprises parallelism-schemes like VLIW and SIMD to exploit inherent instruction- and data-level-parallelism (ILP and DLP). The VLIW concept benefits from ILP within the applications by executing data-independent instructions in parallel. For this purpose, the CoreVA processor integrates multiple processing elements that form independent vertical issue slots executing multiple instructions streams in the pipeline (Multiple Instruction, Multiple Data—MIMD). SIMD benefits from DLP and uses multiple functional units within one processing engine to perform the same operation on different input data elements in parallel. As depicted

[6]RISC: Reduced Instruction Set Computing.

[7]MAC: Multiply Accumulate.

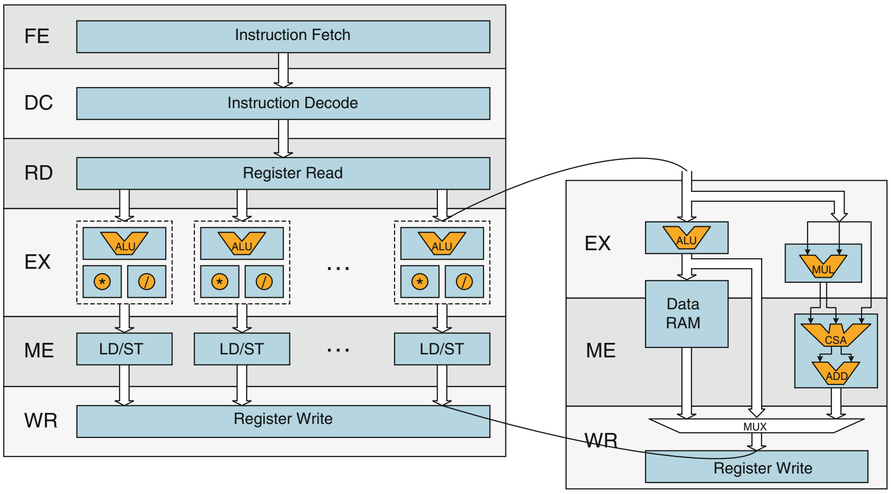


Fig. 3.2 Pipeline of the CoreVA processor

in Sect. 3.4.1, the CoreVA allows for the fine-grained configuration of the number of issue slots and processor's functional units at design-time, e.g., ALUs,[8] MACs, or LD/ST units.

The CoreVA CPU is based on a six-staged pipeline that consists of the instruction-fetch (FE), instruction-decode (DE), register-read (RD), execute (EX), memory (ME), and register-write stage (WR) (see Fig. 3.2). Besides these stages, the processor comprises a unified register file that provides the operands for the calculations and keeps the intermediate results until they are stored to the data memory.

The register file is directly connected to the processor pipeline to ensure that the operands can be read inside the RD stage and written by the WR stage without any delay. It contains 32 general purpose registers with a size of 32 bits each. The 32'nd register is used as a special purpose register holding the program counter. If multiple VLIW issue slots are configured, all of them are able to read and write every register.

The number of the register file's read and write ports depends on the chosen number of issue slots and functional units. Each VLIW slot requires two read and one write port to provide the operands for the ALU and to store the results. If an MAC unit is present, a third read port is required to provide the accumulator. LD/ST units require a second write port to store the updated address in pre- and post-modify addressing mode. Besides these general purpose registers, the register file contains eight 1-bit condition registers that allow for the conditional execution of instructions.

[8]ALU: Arithmetic Logic Unit.

The FE and DE stages are responsible for reading and decoding the instruction words that are stored in the instruction memory. All instructions that are scheduled to be executed in parallel are combined to an instruction group. To avoid performance degradation, the number of instructions read from the memory in the FE stage has to at least match the number of VLIW slots. However, this would imply a waste of instruction memory if the compiler is not able to utilize all available VLIW slots. For this reason, a stop-bit is used to compress partial filled instruction groups by marking the last instruction. By shortening some instruction groups, the memory alignment of instructions would be disrupted leading to additional memory stalls. Therefore, the FE stage contains an alignment register that stores at least two instruction groups and guarantees the availability of one group per clock cycle. If the size of the alignment register is configured to be greater than necessary, this register can be seen as a level zero (L0) cache to buffer instructions and save energy by reducing the number of memory accesses [27].

The DC stage is used to determine the type of the instruction (e.g., data processing, load/store, branch) and to extract the containing operands and register addresses. Furthermore, the DC stage is responsible for the execution of conditional and non-conditional branches. To avoid stall cycles while executing conditional branches, this stage contains a simple branch prediction and functionality to revert mistaken branches. The branch prediction follows the assumption that backward branches are predicted to be taken (e.g., in loops) and forward branches will not be taken [Backward Taken Forward Not Taken (BTFNT)].

After the data registers have been read by the RD stage, the containing operands are forwarded to the EX stage where the actual data operation is performed. Each VLIW slot contains a dedicated 32-bit ALU that comprises an add/sub and a shift unit. Besides the handling of data processing operations, the ALUs are used for memory address calculation. The results of the ALU operations are valid in the end of the EX stage and thus have a latency of one cycle.

In addition to the ALU that is present in every issue slot, each slot can be extended by dedicated multiply accumulate (MAC) and division step (DIV) units. The MAC unit multiplies two 32-bit values and adds a third operand (accumulator) within one MAC instruction. This functionality is an essential part of the CoreVA since MAC operations are very useful for the processing of matrix multiplications which are used in most SDR algorithms. Due to the high combinatorial delay of the multiplier the MAC units are segmented into an EX and ME stage part (cf. Fig. 3.2). On this account, they achieve a sustainable throughput of one instruction per clock cycle with a latency of two clock cycles.

Data memory accesses are handled by dedicated LD/ST units implemented in parallel to the EX-ME pipeline registers (cf. Fig. 3.2). The addresses of load and store operations and the operands that shall be written to the memory are determined by the ALU and provided to the LD/ST unit in the EX stage. The data read from the memory is stored to an ME-WR pipeline register which results in a latency of two cycles for load operations. Divisions are accelerated by using a radix-2 division which is able to calculate a 32-bit division within 32 clock cycles.

To exploit DLP, the CoreVA processor can be enabled to process two 16-bit SIMD calculations within each VLIW slot. If SIMD support is enabled, each VLIW slot is extended by two 16-bit ALUs and two 16-bit MAC units where applicable. The results of the data processing and memory operations are finally stored to the register file in WR stage.

Since the results of the operations are already available at the end of the EX or ME stage, a complete pipeline bypass is implemented to forward those results to directly succeeding operations. The pipeline bypass provides paths from the last three pipeline stages (EX, ME, WR) to all operand registers in RD stage (data bypass) and condition registers in DC stage (control bypass). The number of bypass paths highly depends on the number of implemented VLIW slots. Since the utilization of this bypass path is very application-specific and some of them may only rarely be used, the CoreVA architecture allows the configuration of whether a pipeline bypass is implemented or replace by a stalling mechanism. The potential of such an application-specific configuration is shown in [17].

In addition to the high configurability of the processing elements and the control/data flow of the CoreVA processor, instruction and data memories are also widely configurable at design-time (cf. Fig. 3.3). The memories can be configured as caches or on-chip scratchpad memories and their size and the number of read- and write-ports can be adapted to application requirements. Furthermore, if two LD/ST units are configured, the data memory can either be implemented as a true dual-port memory or as a two-banked multi-port memory.

In addition to the CPU pipeline, the CoreVA can be extended via a generic interface to connect to additional components like a CPU system interface (interface to the cluster interconnect), UART or Ethernet MAC, or dedicated hardware accelerators (cf. Fig. 3.3). Those interfaces can be accessed via memory-mapped-IO (MMIO) supported by an address-decoder located in the EX-stage.

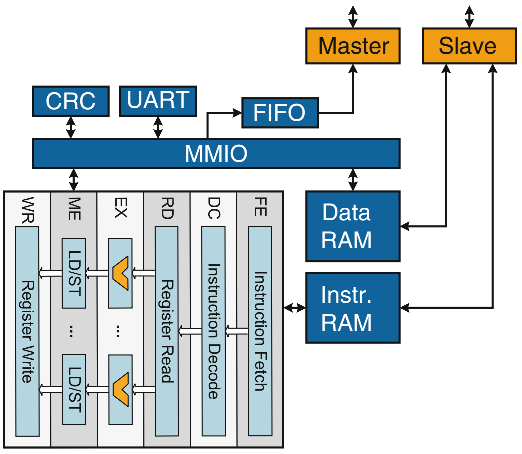


Fig. 3.3 Block diagram of the CoreVA processor environment

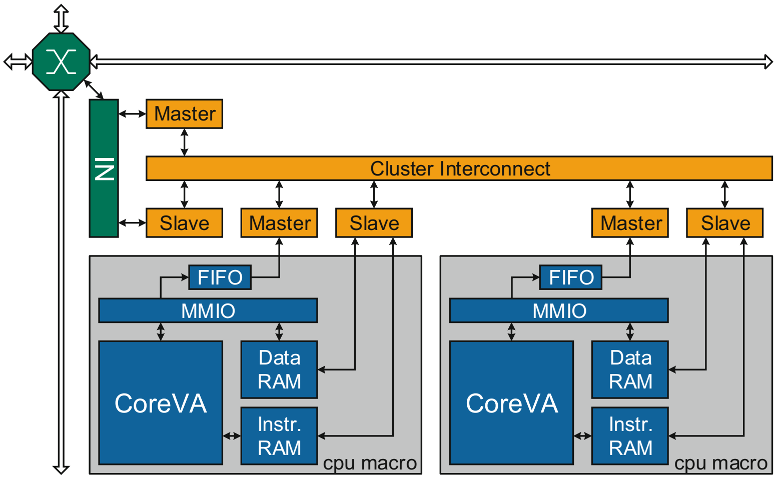


Fig. 3.4 CPU cluster with Network Interface (NI) ©IEEE [32]

3.3.2 *CPU Cluster*

The last section presented the different types of parallelism provided by the CoreVA-MPSoC on core level (i.e., essentially ILP). In this section it is shown how the computing capacity can be further increased by tightly coupling several CoreVA CPUs in a CPU cluster. The CPUs communicate via low latency, high bandwidth interconnects by reading and writing distributed shared memory (cf. Fig. 3.4). Each CPU can access the local data memories of all other CPUs in an NUMA[9] fashion using the cluster interconnect [32]. Therefore, each CPU has a master interface to access the other CPUs, the network interface (NI, cf. Sect. 3.3.3), and an optional common shared memory [33]. In addition, each CPU has a slave interface for initialization and the external access of its local memories.

The master and slaves of a cluster are connected via an interconnect fabric. The highest possible performance and lowest latency can be achieved if a full crossbar is used [28]. All bus masters are directly connected to all bus slaves. This results in large area requirements and power consumption. In contrast, a shared bus allows only one master to access the interconnect at the same time [5]. This increases the probability of bus conflicts, but decreases area and power requirements. A partial crossbar combines the advantages of shared bus and full crossbar by grouping some slaves using a shared bus. The masters are connected to these segments via a full crossbar. Several other interconnect topologies like a ring are discussed in the literature [5, 28] but not considered in this work.

[9]NUMA: Non-Uniform Memory Access.

The configurable architecture of the CoreVA-MPSoC allows for the integration of different interconnect standards depending on the communication requirements of the target application. The CPU cores feature a generic bus interface that can be configured to be compliant to certain bus standards. We have implemented OpenCores Wishbone (WB) [26] as an example for a light-weight bus-based interconnect and ARM AMBA AXI4 [1] which has a more advanced architecture with separated address and data channels. WB is maintained by the community project OpenCores and is used in a broad range of academic and open-source projects.

AMBA AXI4 is used by ARM Inc. as interconnect standard for their latest Cortex-A CPUs. In addition, AXI is used by a broad range of companies for their products. AXI defines separate channels for read and write transfers. This allows for parallel read and write requests even if a shared bus is used. In addition, address and data channels are separated to allow for efficient burst handling. This results in five channels in total (read and write address, read and write data, and write response). Each channel is uni-directional which allows for an easy integration of register stages to achieve higher clock frequencies. Our AXI implementation can be configured to use, e.g., a crossbar for the data channels and a shared bus for the address channels. This allows for high throughput if bursts are used and saves area because the address channels are only rarely used. In this work, we use a non-pipelined round robin arbiter. The AXI shared bus requires 5 arbiters (1 per channel) while the AXI crossbar interconnect requires 2 arbiters per slave (read and write address channel). The WB shared bus requires 1 arbiter in total and the WB crossbar 1 arbiter per slave. The data channels do not require extra arbitration, because our interconnect does not support outstanding transactions.

Depending on the size and the number of connected CPU cores, bus-based interconnects may have long combinatorial paths. This limits the maximum clock frequency of the interconnect. To overcome this drawback, register stages can be added to the design. By default our interconnect implementations are not registered. This results in a combinatorial path from the system interface of one CPU within a cluster to the memories of another CPU. Register stages can be added to master and slave ports. We have implemented two different types of register stages. Type 1 registers all signals but the stall (WB) or ready (AXI) signals. The register can store a request only if the stall signal of the successor is not set. To avoid this combinatorial path, the second implementation (type 2) contains two sets of registers. An incoming request is stored in the first set of registers. In the next cycle, another request can be stored in the second set. If both sets of registers are full, a register is set to stall the predecessor for the next cycle.

The minimum read latency to access a data memory of another CPU within a cluster is four cycles (without register stages). In contrast, the read latency to access the CPU's local data memory is two cycles (cf. Sect. 3.3.1). Outstanding read requests are not required to be supported as our CoreVA CPU does not implement out-of-order execution. The data bus width of both interconnect implementations is configurable at design-time. In this work, we consider a data bus width of 32 bits.

A cluster can optionally integrate a common shared memory that can be split into several banks. This memory can be used to fulfill the memory requirements of state-of-the-art SDR applications like LTE [29]. In combination with a (partial) crossbar, memory banks allow CPUs to access the shared memory in parallel. A DMA controller can be used to copy data between the local data memory of a CPU to the common shared memory or the local memory of another CPU. The CPU is informed (interrupt or mutex) when the transfer has been finished. This reduces the CPU cycles due to copy operations. Our implementation is connected to the cluster interconnect and has one slave port for configuration. Two master ports are used to read from and write to the memories within a cluster. The synchronization of our DMA controller implementation is based on our CoreVA-MPSoC communication model (cf. Sect. 3.5.1).

3.3.3 Network-on-Chip

As discussed in Sect. 3.1, SDR applications are highly computationally intensive. To provide the required throughput the many-core architecture has to scale towards a high number of processing cores. Shared-bus-based interconnects lack a linear speedup for a high number of connected nodes, because the resource requirements of the interconnect network and/or stalls due to bus arbitration limit the expected gain in performance. To handle this, an additional interconnect hierarchy called Network-on-Chip (NoC) is introduced to the CoreVA-MPSoC. Because of its modularity and an explicit packet-based communication within an NoC, additional CPU cores scale with a higher and more predictable performance compared to a bus-based interconnect [9, 32].

The NoC used in the CoreVA-MPSoC features packet wormhole switching. In contrast to circuit switched networks, packet switched networks transmit the data packets without allocating the entire physical path for a single packet [2]. This allows for the usage of physical links for different data streams at the same time. In wormhole switching each packet is segmented into small pieces called flits (flow control digits). All flits of a data packet are sent sequentially from the source node to the destination node, along one path of the NoC. In wormhole switching only small flits need to be buffered at each node. This requires less amount of buffer space compared to a “store and forward” or “virtual cut through” switching network. For embedded devices tailored for SDR applications this is more efficient in terms of chip area and power consumption.

3.3.3.1 Switch Box

The NoC is built up of routing nodes called switch boxes (cf. Fig. 3.1), which forward the flits along a path of the network links. Each switch box has a configurable number of ports.

One port of each switch box is connected to a CPU cluster or a single CPU core using a network interface (NI). All other ports are connected via communication links to further switch boxes to build up the network topology. The configurable number of ports allows for the implementation of different topologies at design-time. Until now the CoreVA-MPSoC supports the configuration of three different topologies: 2D-mesh, honeycomb, and ring. The number of ports per switch box (5 for 2D-mesh, 4 for honeycomb, and 3 for ring) has a major effect on-chip area and power consumption of the NoC. A higher number of ports lead to a better bisection bandwidth of the network, which results in a higher maximum data throughput. For the further description of the CoreVA-MPSoC, a 2D-mesh topology (cf. Fig. 3.1) in combination with a static XY-routing algorithm is used. The static XY-routing is a simple routing and always follows a deterministic path. Due to XY-routing, the 2D-mesh topology is interpreted as a cartesian coordinate system. In this context each destination node (a CPU cluster or a single CPU) of the network has a unique *XY*-coordinate. Flits are sent along the shortest path of the network, first in *X*-direction and afterwards in *Y*-direction. Another more powerful routing algorithm would be an adaptive routing. Adaptive routing is able to avoid congested paths, but the implementation is more complex compared to static routing. This results in an increase of chip area and power consumption, which is in conflict with the requirements for an embedded MPSoC platform executing SDR applications.

Figure 3.5a shows a single switch box and gives more details about a port. At first an incoming flit is registered to decouple the critical path of the circuit from other switch boxes within the network. The second stage features an FIFO[10], which buffers flits if it is not possible to forward them directly to the next switch box (e.g., high traffic on the outgoing link). To guarantee that no flits get lost, a flow control between two switch boxes is required. Within the CoreVA-MPSoC a stall/go flow control is used. If there is no buffer space available at the receiving switch box, a stall signal is activated. In contrast to a handshake-based flow control, this leads to a higher throughput, because a flit can be sent every cycle while no stall signal is active. In addition, the stall signal is registered at the output port to decouple the critical path of the circuit from other switch boxes. The registration of flits and the

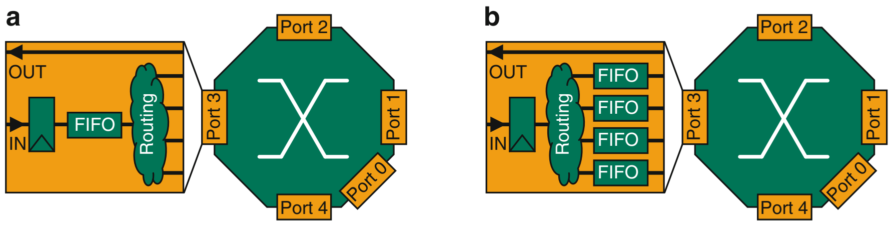


Fig. 3.5 Switch box (**b**) with and (**a**) without virtual channels

[10]FIFO: First In First Out.

stall signals require an FIFO with almost-full functionality to avoid a flit loss. If a flit is waiting in the FIFO, a routing component decides to which output port the flit needs to be forwarded. A crossbar with round robin arbitration per output port decides from which input port a flit could be sent.

Due to the input register and the FIFO, each switch box has a latency of two clock cycles. For the minimal latency of a whole packet, this implies a latency of two clock cycles per hop for the first flit of the packet. All remaining flits follow one-by-one.

To increase the average throughput, virtual channels can be introduced to the NoC. The virtual channels are realized with dedicated input buffers at each input port (cf. Fig. 3.5b). This allows flits to pass a waiting flit, as long as they are forwarded via a different output port. On the one hand, virtual channels increase the average throughput, on the other hand, chip area and power consumption increase, which makes it less attractive for embedded systems.

3.3.3.2 Network Interface

The network interface (NI, cf. Fig. 3.4) represents the interface between the CPU cluster and the NoC [4]. The NI bridges between the address-based communication of the cluster and the flow- and packet-based communication of the NoC to provide a communication between two CPU cores of different clusters. This communication is based on the CoreVA-MPSoC communication model presented in Sect. 3.5.1. It acts like a DMA controller by sending and receiving packets concurrently to CPU processing. Communication between two CPUs is done via uni-directional channels synchronized by mutexes.

Figure 3.6 gives an abstract overview of the NI architecture. One end of the NI is connected with the cluster interconnect. Via Slave Control all CPUs can configure the NI by writing information about incoming packets into a look up table (LUT). On the other end, the Recv. Control handles incoming flits from the NoC interconnect. All flits of a packet have the same unique flow ID to identify their origin. With this flow ID and address information, stored in an LUT, the NI is

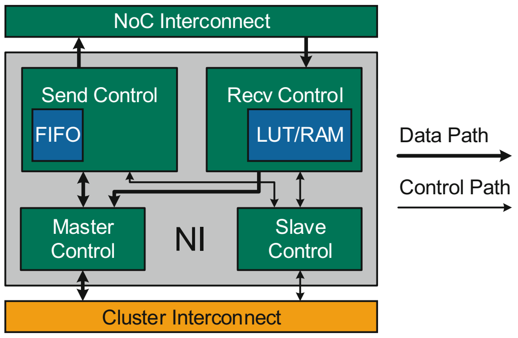


Fig. 3.6 NoC interface (NI)

able to store incoming flits directly in the data memory of the target CPU via the Master Control. The Send Control handles the sending of packets, which is initiated by the source CPU.

With the help of an FIFO the NI is able to handle concurrent sending requests from several CPUs of a cluster without any blocking of the CPUs. Requests within the FIFO are executed successively by the Send Control. It reads packets directly from the data memory of the source CPU, segments it in flits, and sends it to the switch box. Each flit contains a 23-bit header for control information (e.g., flow ID or X and Y coordinates) and 64-bit of payload data. The maximum payload size of a whole packet is configurable at design-time and set to 4 kB for the proposed CoreVA-MPSoC implementation.

3.3.3.3 Globally Asynchronous Locally Synchronous

The implementation of large-scaled MPSoCs results in a very complex design process when implementing fully synchronous circuits. Huge clock trees require a high effort to minimize clock skew which leads to an increasing chip area and power consumption. In addition, a large synchronous circuit impacts the electromagnetic interference (EMI) of the system due to high peak currents on the clock edges.

The NoC can be extended by Globally Asynchronous Locally Synchronous (GALS)-based network links for large-scaled CoreVA-MPSoCs. To support GALS links the system is divided into sub-modules with small autonomous clock trees. In the CoreVA-MPSoC each sub-module consists of a cluster and a single switch box (cf. Fig. 3.4), hereinafter called node. To divide the system into these nodes with autonomous clock trees, mesochronous links are introduced between the switch boxes. In this context, mesochronous indicates that the clocks of the different nodes run with the same frequency but unknown phase shifts. This phase shift can distribute the switching activity of the MPSoC over the whole clock period and therefore minimize electromagnetic interference of the design.

To avoid metastability at the input ports of a switch box, a mechanism for synchronization needs to be integrated. In the CoreVA-MPSoC the Tightly Coupled Mesochronous Synchronizer (TCMS) proposed in [20] is used. The TCMS replaces the input register within each network port of a switch box. It is based on three latches, which alternately store the incoming data controlled by a 2-bit counter, clocked by the sender clock. A second 2-bit counter, clocked by the receiver clock, controls the forwarding of stable data out of the latches. As shown in [16], the TCMS is more efficient in chip area and latency compared to asynchronous FIFOs.

3.4 Implementation Results in a 28 nm FD-SOI Technology

To provide the required high resource efficiency for an embedded many-core architecture executing SDR applications, the optimal trade-off between resource requirements and performance has to be found. Resource costs are given by the production- and processing-costs, namely die area and power consumption of the system.

Therefore, this section analyzes the resource requirements for different processor, cluster, and NoC configurations in terms of area and power consumption. The results are generated using a highly automated standard-cell design-flow based on Cadence Encounter Digital Implementation System. A 28 nm FD-SOI standard-cell library[11] provided by STMicroelectronics is used for the physical implementation. This technology offers a maximum clock frequency of 900 MHz for our CPU core. For the proposed implementation we chose a slightly relaxed clock frequency of 800 MHz. The presented results are based on RTL synthesis. RTL Synthesis features a much shorter round-trip time compared to physical implementation (place and route) and allows an accurate comparison of different CPU and MPSoC configurations. Nevertheless, we present example results at the end of each subsection to show the exact area requirements for certain CPU and MPSoC configurations. All power results considered in this section are determined on the basis of gate-level netlists assuming switching activities of 10 %.

3.4.1 CoreVA VLIW CPU Implementation Results

Since the configurability of the CoreVA CPU enables the implementation of several ten-thousand different processor configurations, the analysis of the complete design-space would be very time consuming. Therefore, this chapter evaluates the impact of single configuration parameters on the area and power consumption. Thus, only one of the most important processor core parameters is varied at once while the others are fixed. The resource requirements of many other processor configurations can be seen in Sect. 3.6.1 and in [15]. For this analysis, the design-space is limited to 1–8 VLIW-issue-slots, 1–8 MAC-units, and 1 or 2 LD/ST units. If 2 LD/ST units are implemented, either a true dual-port or a two-banked multi-port memory can be used. The number of DIV units is fixed to 1 since divisions are rarely used in SDR algorithms. The size of the instruction- and data-memory is fixed to 16 KB. The width of the instruction memory is set to 256 bits. The EX-DC control bypass is disabled as proposed in our previous work [17].

The basic RISC-like configuration that consists of one ALU, MAC, and LD/ST unit has an area requirement of 0.144 mm^2 and a power consumption of 21,243 mW.

[11]STMicroelectronics, 10 metal layer, Worst Case Corner: 1.0 V, 125 °C.

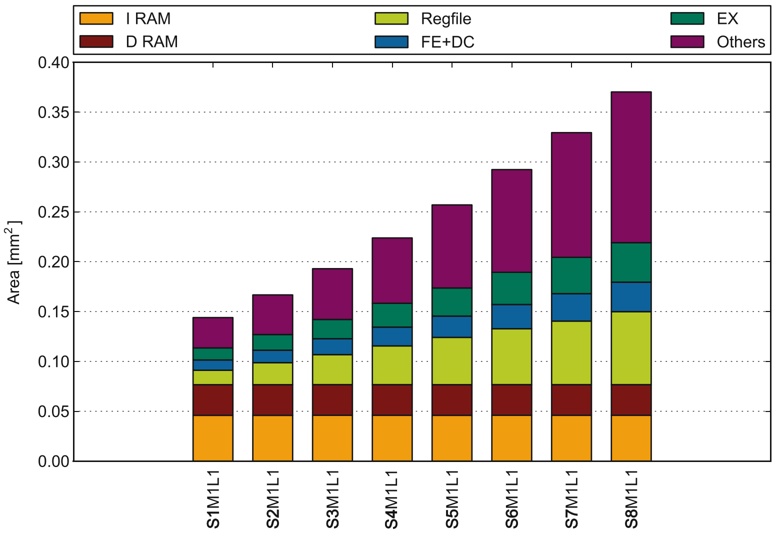


Fig. 3.7 Area requirements of CoreVA CPU configurations with varying number of slot (S, M, L = number of slots, MAC, LD/ST)

As depicted in Fig. 3.7, the area increase for every additional VLIW slots is about 0.032 mm^2. The power consumption increases by 3.732 mW, respectively. This overhead is mainly caused by additional ALUs in the EX stage, more read and write ports of the register file, and by the adaption of the pipeline registers.

A comparison of the first five bars in Fig. 3.8 shows the overhead for adding up to eight MAC units to a 8-slot CoreVA with one LD/ST-Unit. The resulting average increases are 0.003 mm^2 and 0.330 mW per MAC. In addition to an increasing size of the EX stage, the area of the register file increases due to the additional read ports for the accumulator of the MAC unit.

Finally, the resource requirements for a processor with eight slots and one MAC unit increase by 0.02 mm^2 and 5.48 mW (multi-bank memory) or 0.08 mm^2 and 6.08 mW (dual-port memory), if a second LD/ST unit is implemented. This increase is caused by the impact of the additional LD/ST unit and by the overhead of the true dual-port memory compared to the single-port or multi-bank memory with same capacity.

Figure 3.9a shows the layout of the CoreVA VLIW CPU in a 2-issue configuration with 2 ALUs, 1 MAC, and 1 LD/ST unit. For all MPSoC configurations a fixed CoreVA CPU hard macro with 2 VLIW slots, 16 kB data-, and 16 kB instruction-memory is used. This macro is completely pre-placed and pre-routed based on the synthesis results presented in this section. The area requirements are 0.130 mm^2 and the estimated power consumption is 18.57 mW.

Up to now, two chip prototypes of the CoreVA CPU have been realized based on a 65 nm low-power technology. Data and instruction caches have a size of 16 kB each. The CoreVA-VLIW is based on a conventional standard-cell library and contains

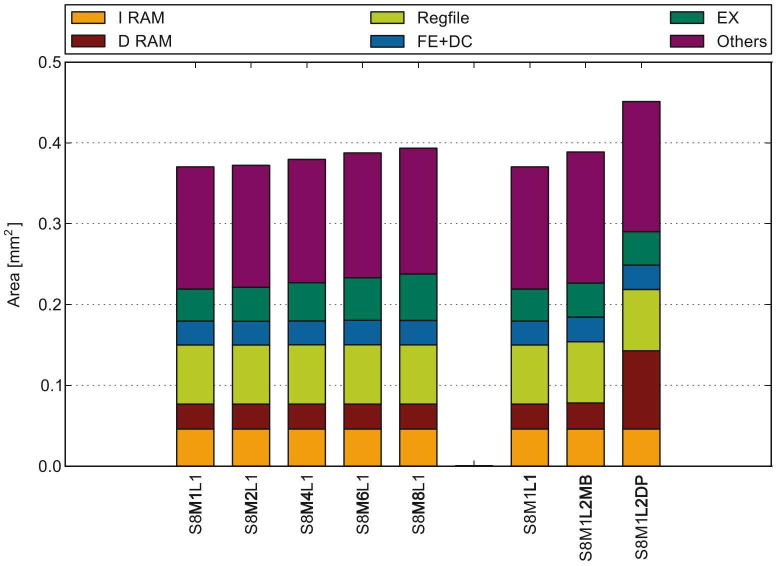


Fig. 3.8 Area requirements of CoreVA CPU configurations with varying number of MAC- and LD/ST-units (S, M, L = number of slots, MAC, LD/ST)

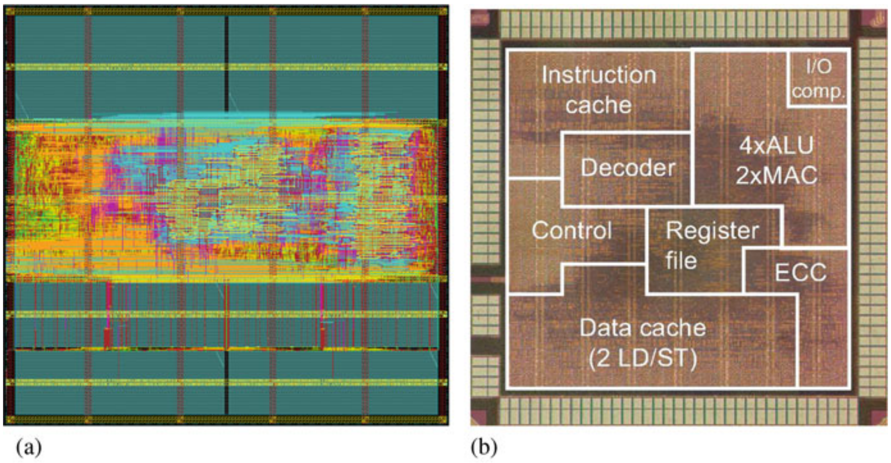


Fig. 3.9 Annotated floorplan and die photograph of the CoreVA VLIW CPU. (**a**) 2-issue CoreVA CPU in 28 nm FD-SOI. (**b**) 4-issue CoreVA in 65 nm technology ©IEEE [31]

4 ALU, 2 MAC, 2 DIV, and 2 LD/ST units (cf. Fig. 3.9b). Power consumption is about 100 mW at a clock frequency of 300 MHz. The CoreVA-ULP[12] [21] features a RISC-like one-slot processor with 1 ALU, 1 MAC, 1 DIV, and 1 LD/ST units.

[12]ULP: Ultra Low Power.

This ASIC is based on a custom standard-cell library optimized for sub-threshold operations. The operation voltage of the CoreVA-ULP ranges from 1.2 V down to 200 mV. The lowest energy consumption (9.94 pJ/cycle) is observed at a supply voltage of 325 mV [22].

3.4.2 MPSoC Implementation Results

This section shows synthesis results for different MPSoC configurations comprising 8, 16, and 32 CoreVA CPUs. For all MPSoC configurations the CoreVA CPU hard macro (2 VLIW slots, 16 kB data-, and 16 kB instruction-memory) presented above is used. For a CPU cluster we consider different topologies (shared bus, partial crossbar, and full crossbar; see Sect. 3.3.2). The partial crossbar configurations feature 4 (*N*/4) and 2 (*N*/2) CPUs per shared bus with *N* representing the number of shared buses of the partial crossbar interconnect. For 8 CPUs this results in a partial crossbar configuration with 2 shared buses and 4 CPUs per shared bus (2/4) and another configuration with 4 shared buses and 2 CPUs per bus (4/2, cf. Fig. 3.10). The interconnect within a cluster is compliant to the AXI standard. In addition, we consider different NoC configurations with 1, 4, and 8 CPUs per cluster. All NoC configurations use a full crossbar as the cluster interconnect.

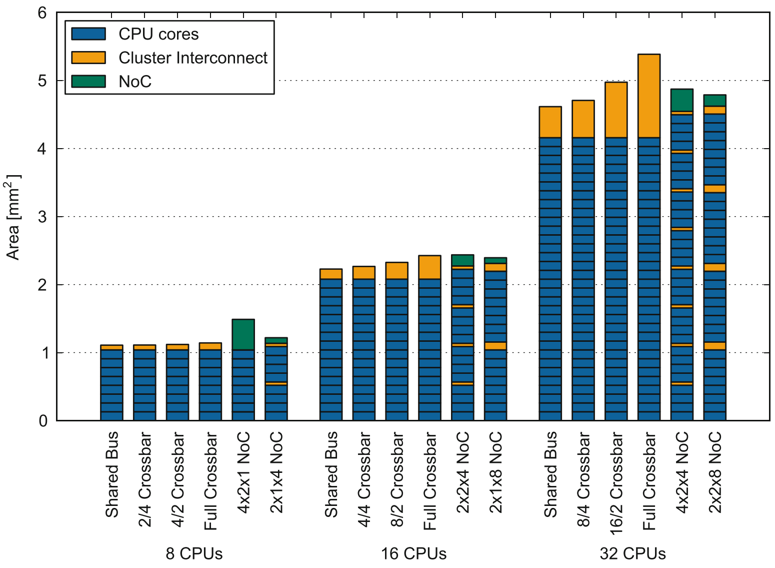


Fig. 3.10 Area requirements of different MPSoC topologies, 2-issue VLIW CPU, and 32 kB memory per CPU. The partial crossbar configurations features 4 (*N*/4) and 2 (*N*/2) CPUs per shared bus where *N* is the number of shared buses ©IEEE [32]

To match the target frequency of 800 MHz all configurations with 32 CPUs require 1 master and 1 slave register stage to meet the timing (cf. Sect. 3.3.2). The 16 CPU full crossbar configuration requires one master register stage of type 2 (double register). All other configurations met timing with 1 master register stage of type 1 or type 2. The type 1 configurations are shown in Fig. 3.10 due to smaller area requirements.

The 8 CPU cluster with a shared bus has a total area of 1.11 mm^2, the interconnect requires only 6.26 % (0.069 mm^2) of this area. The 2/4, 4/2, and full crossbar configurations consume only 0.071 mm^2, 0.080 mm^2, and 0.102 mm^2 respectively. A 4×2 mesh NoC with 1 CPU per cluster (4×2×1, NoC_dimesion_1 × NoC_dimension_2 × #CPU_per_cluster) has an area of 1.49 mm^2 with 0.448 mm^2 (30.13 %) for the NoC. This shows that a pure NoC without a CPU cluster as an additional level of hierarchy has a large area overhead compared to bus-based interconnects. The NoC with two 4-CPU clusters (2×1×4) requires 1.22 mm^2 with 0.097 mm^2 for the two AXI crossbar clusters and 0.082 mm^2 for the NoC. Area requirements for the 16 CPU clusters vary from 2.23 mm^2 (shared bus) to 2.43 mm^2 (full crossbar). The 4/4 partial crossbar is 26 % larger than the shared bus whereas the 8/2 partial crossbar requires 66 % more area. Both considered NoC configurations have approximately the same size compared to the full AXI crossbar (2×2×4: 2.44 mm^2, 2×1×8: 2.39 mm^2). Considering 32 CPU cores, the shared bus implementation scales quite well and requires only 9.89 % of the overall area. The crossbars interconnects require 11.64 % (8/4), 16.40 % (16/4), and 22.77 % (full crossbar) of the overall area. A 4×2×4 MPSoC consumes 14.65 % of the area whereas the 2×2×8 MPSoC requires 13.14 % of the overall area for the interconnect.

In the following we present power estimations for the MPSoC configurations with 16 CPUs. The shared bus cluster consumes 331 mW in total including 10 % for the interconnect. The 4/4 partial crossbar dissipates 345 mW, the 8/2 partial crossbar 358 mW and the full crossbar 383 mW. The 2×2×4 NoC consumes 433 mW and the 2×1×8 NoC 394 mW.

Figure 3.11 shows the physical implementation (layout) of a 3×2×4 NoC with 24 2-issue VLIW cores and 768 kB distributed shared memory in total. The area of the CPU clusters with four CPUs is 0.755 mm^2 and the total area is 4.53 mm^2 [32].

3.5 Programming the CoreVA-MPSoC

Most SDR applications can be split into smaller parts or algorithms, e.g., different filters, FFT, etc. A typical SDR development flow starts with a specification of the SDR application using a high-level language, e.g., MATLAB. In a second step each algorithm is implemented in the target language, e.g., C or C++ for a single CPU of the target platform. Within this step it is important to consider limitations of the target platform, like floating-point support or the availability of libraries. We use the compiler framework LLVM [19] with a CoreVA-specific backend to map algorithms

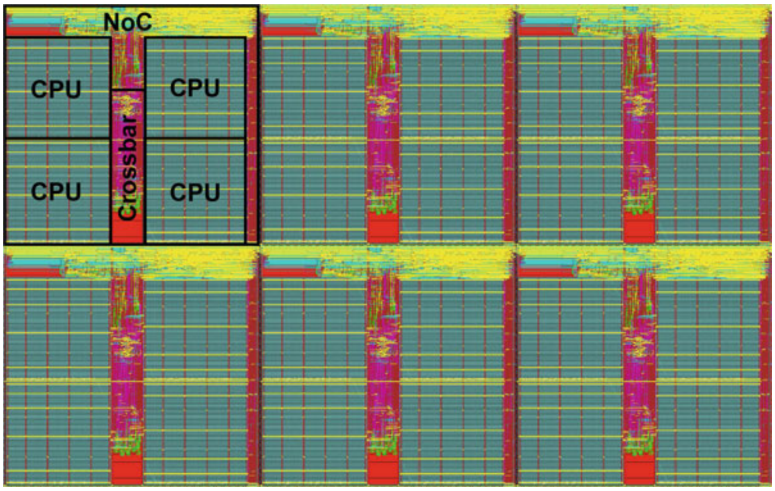


Fig. 3.11 Layout of a 3×2×4 NoC with 4.53 mm^2 area requirements

written in C to a single CoreVA VLIW CPU. In addition, we have developed a cycle accurate simulator for both single CPU and MPSoC simulations. This allows for an accurate estimation of execution time for each algorithm at the early design stage.

3.5.1 Communication Model

The different algorithms of the SDR application are connected via a directed data flow graph. Based on early performance estimation, some algorithms can be executed on the same CPU and some algorithms may have to be split into multiple CPUs to satisfy the throughput and latency constrains of the SDR application. To allow communication between the algorithms executing on different CPUs, a communication model is required. Within the CoreVA-MPSoC we use a communication model with uni-directional communication channels. This approach is more efficient and scalable than shared memory concepts where the memory access can be the bottleneck [23].

In general, an algorithm will read from one or more input channels and write to one or more output channels. Each channel manages one or more read/write buffers. The application can ask its input channels for a buffer to read from and ask its output channels for a buffer to write. The channels manage synchronization at the granularity of a buffer. The operation of requesting a buffer from a channel may block if sufficient data has not yet arrived or if there is not a free buffer available to write. However, once the application is handed a buffer by channel, it is free to read or write (as appropriate) any memory location within that buffer without need for any further synchronization. When the application finishes using a buffer, it must inform the channel so that it can be reused.

There are three types of channels:

- *MemoryChannel* for communication between algorithms mapped to the same core.
- *ClusterChannel* for communication between cores of same cluster.
- *NoCChannel* for communication between cores of different clusters.

To simplify the work for the programmer, a unified interface is provided regardless of the channel type. Internally inter-CPU channels (ClusterChannel and NoCChannel) maintain at least two separate buffers so that latency can be hidden. This is called double buffering—one buffer can be filled while a previous buffer is still being read.

In case of the ClusterChannel, the buffers are allocated in the receiving processor's memory to avoid the latency of a read over the bus. In addition, write accesses to the bus are made asynchronous via a hardware FIFO queue attached to each CPU. For synchronization, two mutexes are used per buffer: one mutex signals that the producer can start writing to the buffer and the second one signals that the consumer can start reading from the buffer. To minimize latency for checking the mutexes, the *ReadyToWrite* mutex is allocated in the producer's memory and the *ReadyToRead* mutex is allocated in the consumer's memory. A hardware mutex can be used to switch off the CPU and save energy until all required buffers are available.

From a programmer's perspective the interface remains the same for an NoCChannel. Internally an NoCChannel is implemented as a pair of ClusterChannels, one on the sending cluster and the other on the receiving cluster. The NCI on the sending cluster acts as a consumer and the NCI on the receiving cluster acts as a producer. For an NoCChannel, data buffers are allocated at the sending and at the receiving cluster. Therefore, the memory footprint of an NoCChannel is doubled compared to a cluster channel.

3.5.2 StreamIt Language and Compiler

Programming and partitioning an application efficiently on a large number of CPUs is a non-trivial task. Although a unified programming model exists for our CoreVA-MPSoC, a lot of effort is required to adapt an existing program to a different number of CPU cores or another topology. By automating the partitioning process the programmer is able to write a program independently from a specific CoreVA-MPSoC configuration.

Streaming applications are a typical kind of program in the SDR domain. A popular stream-based programming language is StreamIt developed at MIT [34]. A StreamIt program describes a data flow graph consisting of filters which are connected via pipeline, split-join, and feedback loop structures. Hereby, the inherent parallelism contained in the program can be expressed in an abstract way. Each filter is an independent block that receives an input data stream, processes the data, and produces an output data stream. In [18], we presented a compiler for the StreamIt language targeting the CoreVA-MPSoC architecture.

Our compiler conforms with the traditional compiler structure. Source code written in the StreamIt language is parsed and analyzed to create a corresponding abstract syntax tree. The static part of the StreamIt program is then “executed” (at “compile time”) to dynamically generate a hierarchical data flow graph consisting of primitive filter instances connected via pipelines, split-join, and feedback loop structures. Every filter must specify the number of data items that it will consume and produce each time its work function is executed. Based on these I/O rates, the compiler determines a schedule by computing the minimum number of times each filter will need to be executed in each iteration of the application’s steady state loop. We require that during each iteration of the steady state loop each producer produces exactly the amount of data required by its consumers, which allows us to allocate only a minimal fixed amount of buffer space. A prime pump phase is executed prior to the start of the steady state loop to fill the buffers so that each filter will always receive data reads from a buffer which was written in a previous iteration of the steady state and writes to a buffer which is not needed before the next steady state iteration. This helps to hide internal communication latency and thereby maximize throughput.

The hierarchical data flow graph consisting of pipelines, split-join, and feedback loop structures is flattened into a graph consisting of just primitive filter instances which are directly connected to one another via edges which represent actual communication channels at runtime.

The compiler partitions this flattened graph by mapping each filter instance to a specific CPU core of the CoreVA-MPSoC. Our compiler tries to balance the computational load as evenly as possible across the available CPUs, while minimizing the communication cost.

Secondly, we can adjust the granularity of work done in each iteration of the steady state loop. If each iteration of the steady state loop is not doing much work, then communication and synchronization overheads may outweigh the benefits from parallelism. To overcome this, we can simply do more work in each iteration of the steady state loop by effectively unrolling it a fixed number of times. If according to the original schedule filter A was executed m times and filter B was executed n times, then we can instead execute them $\alpha \cdot m$ and $\alpha \cdot n$ times, respectively, since the I/O rates remain matched. So the synchronization overhead remains unchanged but more data items are processed. This is referred to as increasing the multiplicity. The buffer sizes increase as a result, so multiplicity must be restricted as the system’s memory is a limited resource.

Thirdly, all three forms of parallelism [14], namely pipeline, task, and data parallelism, need to be exploited by the compiler. A StreamIt program allows the compiler to directly exploit pipeline parallelism by executing all filters in a pipeline on different CPUs. Our compiler benefits from task parallelism by executing the branches of split-join structures on distinct CPUs. Both forms of parallelism offer typically only a constant amount of parallelism. To enable the scalability required by SDR applications, the use of data parallelism is required.

Data parallelism is achieved by the compiler in terms of replacing a stateless filter instance by multiple clones of that filter and executing them in parallel. This transformation is called fission.

The compiler has to choose: the placement of all filters, the multiplicity, and the amount of fission for individual filters. Currently these decisions are made with the goal of optimizing throughput. To handle such a large search space, a simulated annealing optimization algorithm was implemented [18]. The algorithm starts with an initial solution generated by a simple greedy bin packing algorithm. Afterwards, it tries to improve the throughput by generating mutations of this solution. Meanwhile the compiler has to verify if a solution is valid by taking care of the limited resources of the CoreVA-MPSoC [12]. Typically for an embedded system memory is not only a bounded resource but also the number of individual channels supported by the Network-on-Chip. Finally, C code for every CPU of the CoreVA-MPSoC is generated for the partitioned program. It is automatically determined what kind of communication channel is needed between two connected filters depending on their placement.

To date, all of our optimizations have aimed to only optimize system throughput. However many application scenarios, especially in the context of SDR, have real-time requirements. Therefore, we are currently integrating latency as an optimization target into the compiler.

3.6 Mapping SDR Algorithms to the CoreVA-MPSoC

A typical application of the CoreVA-MPSoC is a software-defined radio implementation of an OFDM-receiver in, e.g., LTE, IEEE-802.11n WiFi, or DVB-T receivers [25]. The following paragraph presents the main processing elements and their underlying algorithms for the inner-part of such an OFDM-receiver (see Fig. 3.12).

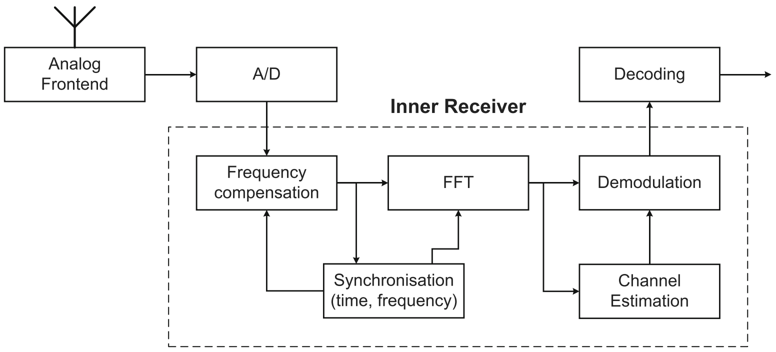


Fig. 3.12 OFDM-receiver [25]

The first processing element of an OFDM[13]-receiver after A/D conversion is the frequency compensation. This element is needed to detect and correct phase shifts that might have happened during transmission. This can be done using cordic or cross-correlation algorithms that are widely used in DSP-Systems [3, 25].

The time and frequency synchronization take place in cooperation with the frequency compensation. These synchronizations are required to detect the beginning of an OFDM-packet and are also based on correlation-techniques [25]. Thereafter the processed signal is transferred to the frequency-domain using Fast Fourier Transformation (FFT) and then demodulated [25]. The demodulation is supported by a channel estimation that determines whether the signal has been distorted by the transmission channel. This is done by extracting and analyzing specially transmitted pilot signals.

In discrete SDR systems, these methods are mainly based on filtering-techniques like Finite-Impulse-Response filter (FIR) [30]. The demodulation reverts the amplitude or phase modulation that has been applied by the OFDM sender. This demodulation is less calculation-intensive compared to the other parts of the receiver [3].

A possible mapping of the processing elements to a many-core architecture like the CoreVA-MPSoC is shown in Fig. 3.13. In this example the calculation-intensive FFT processing is mapped to four CPU clusters whereas the other algorithms require fewer CPUs.

This chapter presents the mapping of several algorithms that are common in most SDR systems to the CoreVA-MPSoC. We use FFT, FilterBank (a series of FIR-filters), AutoCor (autocorrelation), and LowPassFilter that represent algorithms used in an OFDM-receiver. Apart from OFDM, we consider MatrixMult (matrix

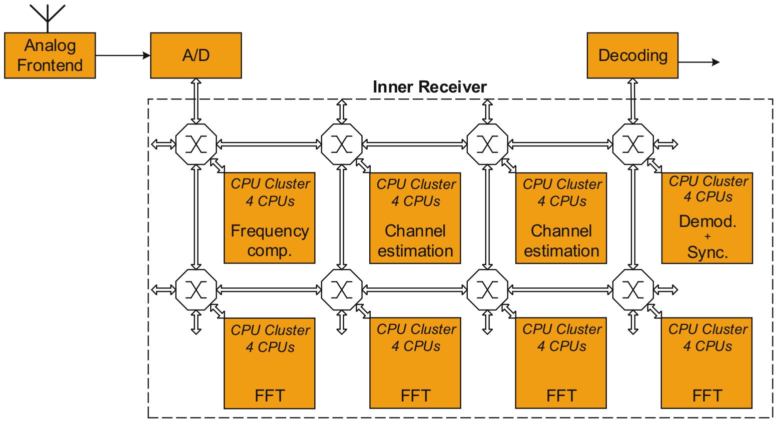

Analog Frontend
A/D
Decoding
Inner Receiver
CPU Cluster 4 CPUs Frequency comp.
CPU Cluster 4 CPUs Channel estimation
CPU Cluster 4 CPUs Channel estimation
CPU Cluster 4 CPUs Demod. + Sync.
CPU Cluster 4 CPUs FFT
CPU Cluster 4 CPUs FFT
CPU Cluster 4 CPUs FFT
CPU Cluster 4 CPUs FFT

Fig. 3.13 Mapping of an OFDM-receiver to the CoreVA-MPSoC

[13]OFDM: Orthogonal Frequency-Division Multiplexing.

multiplication), the encryption algorithm DES, Lattice, and Sender80211. Lattice is a filter algorithm used in devices for the GSM standard. Sender80211 is a baseband implementation of a WiFi sender. We evaluate the mapping to different single CPU configurations with up to four issue slots and MPSoC configurations with up to 16 CPUs.

3.6.1 Mapping SDR Algorithms to a Single CPU VLIW Core

This section shows the performance gains that can be achieved by using additional VLIW issue slots and functional units as well as the energy requirements that represent the trade-off between the resource costs and performance gains.

The analyses treat the algorithms AutoCor, FFT, FilterBank, and LowPassFilter. The LLVM compiler is used to map the algorithms to the CPU configurations presented in Sect. 3.4.1. It uses a configuration-specific scheduler that is aware of the availability of issue slots and MAC- and LD/ST-units. The execution time is measured using our cycle accurate instruction set simulator. The performance and energy gains are determined by calculating the speedup or energy reduction of the individual configurations compared to the 1-slot configuration. Figure 3.14 shows that adding an additional VLIW slot results in a speedup of 1.2 (AutoCor), 1.6 (FFT), 1.5 (FilterBank), and 1.7 (LowPassFilter). This speedup improves the energy

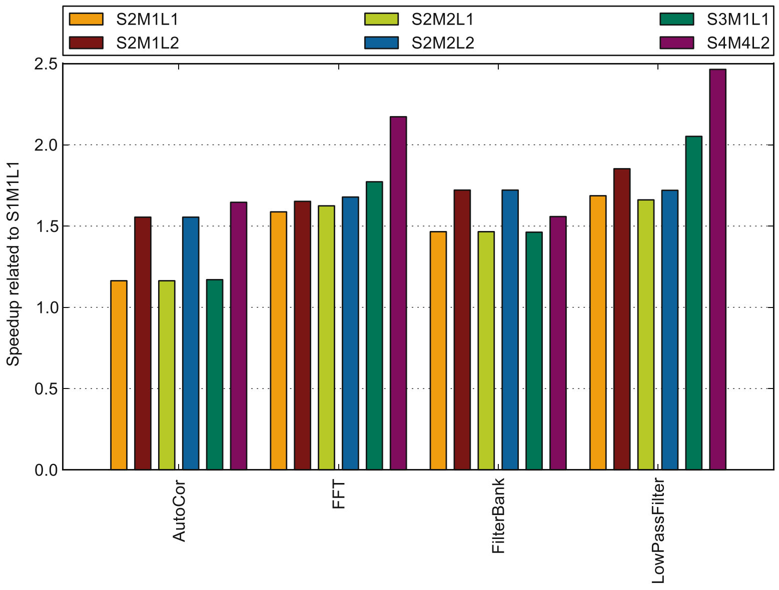


Fig. 3.14 Speedup of different CPU configurations related to S1M1L1 (S, M, L = number of slots, MAC, LD/ST)

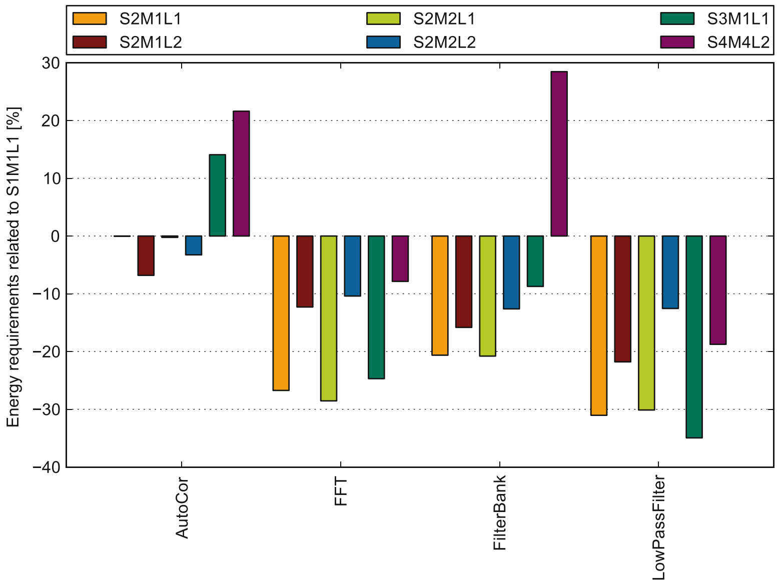


Fig. 3.15 Energy requirements of different CPU configurations related to S1M1L1 (S, M, L = number of slots, MAC, LD/ST)

consumption by 27 % (FFT), 21 % (FilterBank), and 31 % (LowPassFilter) since the number of cycles falls more than the power requirements increase (see Fig. 3.15). The energy requirements of the AutoCor algorithm stay constant.

By adding a second LD/ST and a second MAC unit (S2M2L2), the performance can be increased even further. The speedup increases by a factor of 1.6 (AutoCor), 1.7 (FFT), 1.5 (FilterBank), and 1.7 (LowPassFilter) compared to the 1-slot configuration. Unfortunately this improvement comes at a high power and area consumption that is mainly caused by the overhead of the true dual-port memory (c.f. 3.4.1). Therefore the resulting energy requirements worsen in most cases, i.e., AutoCor 3 %, FFT 10 %, FilterBank 13 %, and LowPassFilter 13 % in comparison to the 1-slot configuration. Thus only AutoCor can benefit from these additional functional units since it is based on a comparatively large amount of load-store operations.

Finally, increasing the number of VLIW issue slots and functional units to the maximum value (S4M4L2) shows a speedup of 1.6 (AutoCor), 2.2 (FFT), 1.6 (FilterBank), and 2.5 (LowPassFilter). It can be seen that the compiler's ability in utilizing additional functional units highly differs from one algorithm to another due to data dependencies between succeeding instructions. FFT and LowPassFilter can benefit from a third and fourth issue slot while the speedup of AutoCor and FilterBank remains nearly the same like for S2M2L2.

The search for the configuration that achieves the lowest energy consumption while processing the given algorithms results in the configuration S2M1L2 for

AutoCor, S2M2L1 for FFT and FilterBank, and S3M1L1 for LowPassFilter. Those configurations decrease the energy requirements by 6 % (AutoCor), 29 % (FFT), 21 % (FilterBank), and 35 % (LowPassFilter) in contrast to the 1-slot configuration.

3.6.2 Benchmark Results for CoreVA-MPSoC

Certainly the huge performance requirements of SDR applications exceed the throughput provided by a single CPU. Therefore most applications have to be partitioned on multiple CPUs. To determine the overhead of the communication and parallelizing the algorithms, we analyzed the speedup of the presented algorithms compared to a single CPU. Figure 3.16 shows the speedup of throughput using 4, 8, and 16 CPUs for different MPSoC configurations. All CoreVA-MPSoC configurations use a CoreVA CPU with two issue slots (S2M1L1). The partitioning of the algorithms to the different CoreVA-MPSoC configurations is done by our compiler for the StreamIt language.

The average speedup of all algorithms is 4.0 for a cluster with 4 CPUs (1×1×4) compared to a single CPU. For clusters with 8 (1×1×8) and 16 (1×1×16) CPUs the average speedup factor is 7.1 and 11.0, respectively. A hierarchical MPSoC configuration with a 2×1 NoC and 8 CPUs per cluster (2×1×8) shows similar results (average speedup factor of 10.5). With a speedup between 11 and 15, the DES, FFT,

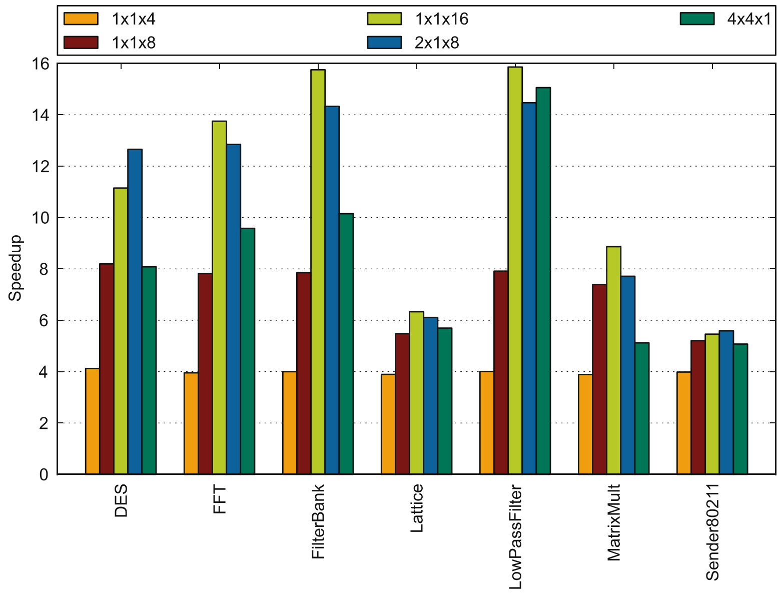


Fig. 3.16 Speedup of SDR applications on different CoreVA-MPSoC configurations compared to a single CoreVA CPU

FilterBank, and LowPassFilter benefit most from 16 CPUs. The implementations of Lattice and MatrixMult do not benefit significantly from more than 8 CPUs, because CPU-to-CPU communication becomes a bottleneck. This shows that the most efficient number of CPUs highly depends on the considered algorithm.

The effect of the overhead of NoC communication is clearly shown in a 4×4 NoC with only 1 CPU per node (4×4×1). NoCChannels are more expensive in terms of memory consumption (see Sect. 3.5.1), which limits the average speedup to 8.4. Nevertheless, an NoC is essential to build MPSoCs with hundreds of CPUs due to the shown hardware limitations of bus-based interconnects. The benchmark results presented in this section combined with hardware analysis shown in Sect. 3.4 and [32] indicate that a hierarchical MPSoC (CPU cluster and NoC) provides the most efficient trade-off between performance and resource requirements. We presented the speedup gains provided by a massively parallel architecture for sub-tasks of a next generation SDR application. For a holistic scenario, the application will consist of dozens of this sub-tasks and therefore will benefit even more from large-scaled MPSoCs providing the required computational throughput.

Figure 3.17 gives an example how the Sender80211 algorithm is mapped to a 1×1×4 CoreVA-MPSoC. The Sender80211 is the sender part of an IEEE 802.11b algorithm. It performs scrambling, differential encoding, symbol mapping, and an FIR-filter which is split into inphase (I) and quadrature (Q) components. Additionally an input filter generates a data stream and an output filter joins the data from FIR-filters to send it out of chip. The most computational work is done by the FIR-filters. A native programmer would partition the two FIR-filters (I and Q) to two different CPUs and all other filters to the two remaining CPUs. In this partition an FIR-filter will still be the bottleneck and the speedup is limit to 3. However, our StreamIt compiler identifies the bottleneck and performs fission by cloning the FIR-filters, to distribute them to multiple CPU cores (see Fig. 3.17). This results to a better load balancing and the mapping to a cluster with 4 CPUs,

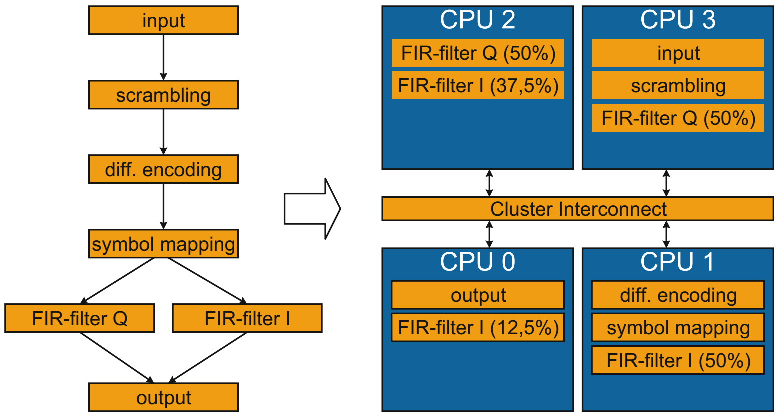


Fig. 3.17 Mapping of a 802.11b sender to a 1×1×4 CoreVA-MPSoC configuration

resulting to a speedup of 4 compared to a single CPU. Because of massive data interpolation, Sender80211 produces a lot of data and the output filter becomes the system's bottleneck limiting further scaling. In future work, this bottleneck will be removed.

3.7 Summary

In this chapter, we presented the CoreVA-MPSoC, an efficient multiprocessor architecture for software-defined radio applications. The CoreVA-MPSoC is based on a hierarchical system architecture comprising multiple resource-efficient processor cores connected via a high performance communication infrastructure. On CPU level, the CoreVA VLIW processor supports fine-grained ILP for the execution of SDR key algorithms. On the second level of hierarchy (i.e., the cluster level), multiple CPUs can be tightly connected via a bus-based interconnect. The tight coupling of CPUs within a cluster is required to allow for an efficient scaling of complex algorithms like FFT because a single CPU is not fast enough. On the third level of hierarchy a network-on-chip is introduced to cope with scaling challenges of the bus-based interconnect when integrating a high number of CPUs and thus meet the processing requirements for SDR.

For the efficient programming, we introduced an LLVM-based C-compiler for a single CPU and a StreamIt-based compiler for mapping of applications to multiple CPUs. Our work has shown that it is possible to realize many-core architectures suited for an efficient SDR processing. Integrating 64 CPUs in 28 nm FDSOI technology provides a peak performance of 200 GOPS at a power consumption of 1.7 W. To further increase the energy-efficiency future work will include a fine-grained power-management to turn-off unused system-parts in phases of lower performance requirements. In addition, we will explore different memory architectures because memory limits the scaling of some of the proposed SDR algorithms.

Acknowledgements This research was supported by the ATN—DAAD Joint Research Co-operation Scheme: Tightly Coupled Software Tools and Adaptable Hardware for Resource Efficient Multiprocessor Architectures, the DFG CoE 277: Cognitive Interaction Technology (CITEC), and the German Federal Ministry of Education and Research (BMBF) within the projects KogniHome (16SV7054K), Treufunk (16KISO236), and the Leading-Edge Cluster "Intelligent Technical Systems OstWestfalenLippe" (it's OWL), managed by the Project Management Agency Karlsruhe (PTKA). The authors are responsible for the contents of this publication.

References

1. AMBA AXI and ACE Protocol Specification (2013). http://www.arm.com/products/system-ip/amba/
2. Agarwal, A., Iskander, C., Shankar, R.: Survey of network on chip (NoC) architectures & contributions. J. Eng. Comput. Archit. **3**(1), 21–27 (2009)

3. Airoldi, R., Garzia, F., Anjum, O., Nurmi, J.: Homogeneous MPSoC as Baseband Signal Processing Engine for OFDM Systems. In: International Symposium on System on Chip (SoC), pp. 26–30 (2010). doi:10.1109/ISSOC.2010.5625562
4. Ax, J., Sievers, G., Flasskamp, M., Kelly, W., Jungeblut, T., Porrmann, M.: System-level analysis of network interfaces for hierarchical MPSoCs. In: International Workshop on Network on Chip Architectures (NoCArc). ACM Press, New York (2015). doi:10.1145/2835512.2835513
5. Baer, J.L.: Microprocessor Architecture: From Simple Pipelines to Chip Multiprocessors. Cambridge University Press, Cambridge (2009)
6. Benini, L., Flamand, E., Fuin, D., Melpignano, D.: P2012: building an ecosystem for a scalable, modular and high-efficiency embedded computing accelerator. In: Design, Automation & Test in Europe Conference & Exhibition (DATE), pp. 983–987. IEEE, New York (2012). doi:10.1109/DATE.2012.6176639
7. Clermidy, F., Bernard, C., Lemaire, R., Martin, J., Miro-Panades, I., Thonnart, Y., Vivet, P., Wehn, N.: A 477mW NoC-based digital baseband for MIMO 4G SDR. In: International Solid-State Circuits Conference (ISSCC), pp. 278–279 (2010). doi:10.1109/ISSCC.2010.5433920
8. de Dinechin, B.D., et al.: A distributed run-time environment for the kalray MPPA-256 integrated manycore processor. In: Procedia Computer Science, pp. 1654–1663. Elsevier, Amsterdam (2013). doi:10.1016/j.procs.2013.05.333
9. De Micheli, G., Seiculescu, C., Murali, S., Benini, L., Angiolini, F., Pullini, A.: Networks on chips: from research to products. In: Design Automation Conference (DAC), pp. 300–305. IEEE (2010). doi:10.1145/1837274.1837352
10. Demler, M.: Xtensa 10 plays well with ARM. Microprocess. Rep. **27**(10), 25–27 (2013)
11. E64G401 Epiphany 64-Core Microprocessor. Tech. rep., Adapteva, Inc. (2014). http://www.adapteva.com/epiphanyiv
12. Flasskamp, M., Sievers, G., Ax, J., Klarhorst, C., Jungeblut, T., Kelly, W., Thies, M., Porrmann, M.: Performance estimation of streaming applications for hierarchical MPSoCs. In: Workshop on Rapid Simulation and Performance Evaluation (RAPIDO). ACM Press, New York (2016). doi:10.1145/2852339.2852342
13. Gardner, J.S.: Tensilica sets its sights on vision. Microprocess. Rep. **27**(3), 19 (2013)
14. Gordon, M.I., Thies, W., Amarasinghe, S.: Exploiting coarse-grained task, data, and pipeline parallelism in stream programs. In: ACM SIGPLAN Notices, vol 41, p 151. ACM, New York (2006). doi:10.1145/1168918.1168877
15. Hübener, B., Sievers, G., Jungeblut, T., Porrmann, M., Rückert, U.: CoreVA: a configurable resource-efficient VLIW processor architecture. In: International Conference on Embedded and Ubiquitous Computing (EUC), pp. 9–16 (2014). doi:10.1109/EUC.2014.11
16. Jungeblut, T., Ax, J., Porrmann, M., Rückert, U.: A TCMS-based architecture for GALS NoCs. In: International Symposium on Circuits and Systems (ISCAS), pp. 2721–2724. IEEE, New York (2012). doi:10.1109/ISCAS.2012.6271870
17. Jungeblut, T., Hübener, B., Porrmann, M., Rückert, U.: A systematic approach for optimized bypass configurations for application-specific embedded processors. ACM Trans. Embed. Comput. Syst. **13**(2) (2013). doi:10.1145/2514641.2514645
18. Kelly, W., Flasskamp, M., Sievers, G., Ax, J., Chen, J., Klarhorst, C., Ragg, C., Jungeblut, T., Sorensen, A.: A communication model and partitioning algorithm for streaming applications for an embedded MPSoC. In: International Symposium on System-on-Chip (SoC). IEEE, New York (2014). doi:10.1109/ISSOC.2014.6972436
19. Lattner, C., Adve, V.: LLVM: a compilation framework for lifelong program analysis & transformation. In: International Symposium on Code Generation and Optimization (CGO), pp. 75–86. IEEE, New York (2004). doi:10.1109/CGO.2004.1281665
20. Ludovici, D., Strano, A., Bertozzi, D., Benini, L., Gaydadjiev, G.N.: Comparing tightly and loosely coupled mesochronous synchronizers in a NoC switch architecture. In: International Symposium on Networks-on-Chip, pp. 244–249. IEEE Computer Society, Silver Spring, MD (2009). doi:10.1109/NOCS.2009.5071473

21. Lütkemeier, S., Jungeblut, T., Porrmann, M., Rückert, U.: A 200 mV 32b subthreshold processor with adaptive supply voltage control. In: International Solid-State Circuits Conference (ISSCC), vol. 57, pp. 484–485 (2012). doi:10.1109/ISSCC.2012.6177101
22. Lütkemeier, S., Jungeblut, T., Kristian, H., Berge, O., Aunet, S., Porrmann, M., Rückert, U.: A 65 nm 32 b subthreshold processor with 9T multi-Vt SRAM and adaptive supply voltage control. J. Solid-State Circ. **48**(1), 8–19 (2013). doi:10.1109/JSSC.2012.2220671
23. Marescaux, T., Brockmeyer, E., Corporaal, H.: The impact of higher communication layers on NoC supported MP-SoCs. International Symposium on Networks-on-Chip (NOCS) pp. 107–116 (2007). doi:10.1109/NOCS.2007.41
24. Noethen, B., Arnold, O., Adeva, E.P., Seifert, T., Fischer, E., Kunze, S., Matus, E., Fettweis, G., Eisenreich, H., Ellguth, G., Hartmann, S., Hoppner, S., Schiefer, S., Schlusler, J.U., Scholze, S., Walter, D., Schuffny, R.: A 105GOPS 36mm2 heterogeneous SDR MPSoC with energy-aware dynamic scheduling and iterative detection-decoding for 4G in 65nm CMOS. In: International Solid-State Circuits Conference (ISSCC), pp. 188–189. IEEE, New York (2014). doi:10.1109/ISSCC.2014.6757394
25. Olsson, T., Carlsson, A., Wilhelmsson, L., Eker, J., von Platen, C., Diaz, I.: A reconfigurable OFDM inner receiver implemented in the CAL dataflow language. In: International Symposium on Circuits and Systems (ISCAS), pp. 2904–2907 (2010). doi:10.1109/ISCAS.2010.5538042
26. OpenCores Project. http://opencores.org/
27. Park, J., Balfour, J., Dally, W.J.: Fine-grain dynamic instruction placement for L0 scratch-pad memory. In: International Conference on Compilers, Architectures and Synthesis for Embedded Systems (CASES), p. 137. ACM Press, New York (2010). doi:10.1145/1878921.1878943
28. Pasricha, S., Dutt, N.: On-Chip Communication Architectures: System on Chip Interconnect. Morgan Kaufmann, Los Altos, CA (2009)
29. Ramacher, U., Raab, W., Hachmann, U., Langen, D., Berthold, J., Kramer, R., Schackow, A., Grassmann, C., Sauermann, M., Szreder, P., Capar, F., Obradovic, G., Xu, W., Bruls, N., Lee, K., Weber, E., Kuhn, R., Harrington, J.: Architecture and implementation of a software-defined radio baseband processor. In: International Symposium on Circuits and Systems (ISCAS), pp. 2193–2196. IEEE, New York (2011). doi:10.1109/ISCAS.2011.5938035
30. Sankarayya, N., Roy, K., Bhattacharya, D.: Algorithms for low power and high speed FIR filter realization using differential coefficients. Trans. Circ. Syst. II: Anal. Digit. Signal Process. **44**(6), 488–497 (1997). doi:10.1109/82.592582
31. Sievers, G., Christ, P., Einhaus, J., Jungeblut, T., Porrmann, M., Rückert, U.: Design-space exploration of the configurable 32 bit VLIW processor coreVA for signal processing applications. In: Norchip Conference, November 2013. IEEE, New York (2013). doi:10.1109/NORCHIP.2013.6702002
32. Sievers, G., Ax, J., Kucza, N., Flasskamp, M., Jungeblut, T., Kelly, W., Porrmann, M., Rückert, U.: Evaluation of interconnect fabrics for an embedded MPSoC in 28 nm FD-SOI. In: International Symposium on Circuits and Systems (ISCAS) (2015). doi:10.1109/ISCAS.2015.7169049
33. Sievers, G., Daberkow, J., Ax, J., Flasskamp, M., Kelly, W., Jungeblut, T., Porrmann, M., Rückert, U.: Comparison of shared and private L1 data memories for an embedded MPSoC in 28 nm FD-SOI. In: International Symposium on Embedded Multicore/Many-core Systems-on-Chip (MCSoC), September 2015. IEEE, New York (2015). doi:10.1109/MCSoC.2015.25
34. Thies, W., Karczmarek, M., Amarasinghe, S.: StreamIt: a language for streaming applications. In: International Conference on Compiler Construction, vol. 2304, pp. 179–196. Springer, Berlin (2002). doi:10.1007/3-540-45937-5_14
35. van Berkel, C.H.: Multi-core for mobile phones. In: Design, Automation & Test in Europe Conference & Exhibition (DATE), pp. 1260–1265. IEEE, New York (2009)

Chapter 4
Design and Implementation of IEEE 802.11a/g Receiver Blocks on a Coarse-Grained Reconfigurable Array

Sajjad Nouri, Waqar Hussain, Diana Göhringer, and Jari Nurmi

4.1 Introduction

During the last decade, wireless technologies have experienced significant development, most notably in the form of mobile cellular radio evolution from GSM to UMTS/HSPA and thereon to Long-Term Evolution (LTE) for increasing the capacity and speed of wireless data networks. The continuous evolution in wireless communication systems has yielded higher data rates to improve the quality of service. The concept of Multiple-Input-Multiple-Output (MIMO) Orthogonal Frequency-Division Multiplexing (OFDM) was introduced to support higher data rates where multiple streams are required to be processed in parallel. On the other hand, with increasing demand for additional bandwidth, new wireless communication technologies have to seek solutions for resolving spectrum scarcity issue. One proposed solution for efficiently utilizing the spectrum across frequency, space, and time is Dynamic Spectrum Access (DSA) [21] which can be realized by using reconfigurable wireless platforms. This purpose can be achieved using Software Defined Radio (SDR) technology and its intelligent version called Cognitive Radio (CR) [14] where a high degree of flexibility is introduced by the software which allows the implementation of multi-mode,

S. Nouri (✉)
Tampere University of Technology, Tampere, Finland
e-mail: sajjad.nouri@tut.fi

W. Hussain • J. Nurmi
Department of Electronics and Communications Engineering,
Tampere University of Technology, Tampere, Finland
e-mail: waqar.hussain@tut.fi; jari.nurmi@tut.fi

D. Göhringer
Ruhr-Universität Bochum, Bochum, Germany
e-mail: diana.goehringer@rub.de

W. Hussain et al. (eds.), *Computing Platforms for Software-Defined Radio*,
DOI 10.1007/978-3-319-49679-5_4

multi-standard transceivers. In an SDR platform, most of the kernels related to signal processing can be performed in programmable processing technologies including General-Purpose Processor (GPP), Digital Signal Processor (DSP), Coarse-Grained Reconfigurable Array (CGRA), and Field Programmable Gate Array (FPGA). Different types of data transmission schemes are employed in wireless communication systems such as OFDM which is important in SDR because of providing high data rates. OFDM systems split an input high-speed data stream into several parallel streams which are simultaneously transmitted across a contiguous collection of non-overlapping subcarriers.

In order to provide a large number of computational resources to process multiple parallel streams and maintaining the power consumption at an acceptable level, reconfigurable architectures are playing a significant role in embedded systems as they provide high computation parallelism and throughput. Furthermore, reducing the execution time by tailoring the platform for a specific algorithm and achieving higher performance level with minimum resources has always been a matter of concern for designers. The reconfigurable devices can be classified according to the level of granularity into Fine-Grained ($<$ 4-bits), Middle-Grained ($\leq$ 8-bits), and Coarse-Grained ($>$ 8-bits) [20]. One of the most powerful classes of reconfigurable platforms is CGRA which offers a high level of parallelism and throughput due to its array-based structure. The symmetry feature in their structure makes them ideal for digital signal processing of multiple parallel streams. CGRAs have a proven track-record of processing massively parallel workloads of critical-priority applications, e.g., Fast Fourier Transform (FFT) [7], Wideband Code Division Multiple Access (WCDMA) cell search [5], Finite Impulse Response Filtering [9], Image Processing, and Video Decoding applications [2, 12]. Some state-of-the-art CGRAs are ADRES [13], MorphoSys [11], PACT-XPP [1], and MORPHEUS [20]. General-purpose CGRAs consume an area of a few million gates due to their ability to perform several tasks which are not financially feasible in lightly loaded scenarios. Furthermore, they can have potentially high transient power dissipation. As a solution, template-based CGRAs were developed which are able to generate application-specific accelerators on a specified mapping tailored for the computationally intensive tasks. In this context, a 32-bit template-based CGRA, CREMA, was generated to reduce area and resource utilization by instantiating only those hardware resources which are required for a particular application [4].

In this chapter, the implementation of application-specific accelerators targeting OFDM receiver baseband processing based on IEEE 802.11a/g (WiFi) specifications on template-based CGRAs is presented. In an OFDM transceiver, the processing of some blocks like Time Synchronization and Fast Fourier Transform (FFT) is computationally intensive for a processor core since each received data symbol has to be processed within 4.0 μs. The chapter is organized as follows: the architecture of the platform is described in the next section. Then the design and implementation of OFDM receiver blocks using template-based CGRA and execution details of baseband algorithms are presented. The last section presents the simulation and synthesis results along with the conclusion.

4.2 Platform Architecture

The experimental platform is a processor/coprocessor model where COFFEE, a Reduced Instruction-Set Computing (RISC) core, is used as a general-purpose processor while the generated accelerators are used as coprocessors for computationally intensive tasks. CREMA is 32-bit CGRA template which is working as an accelerator with COFFEE RISC core. Both COFFEE and CREMA are designed and developed at Tampere University of Technology. As it is shown in Fig. 4.1, CREMA is composed of a matrix of 4 rows $\times$ 8 columns of PEs and two 32-bit local memories each of size 16 columns $\times$ 256 rows. There are also two I/O buffers

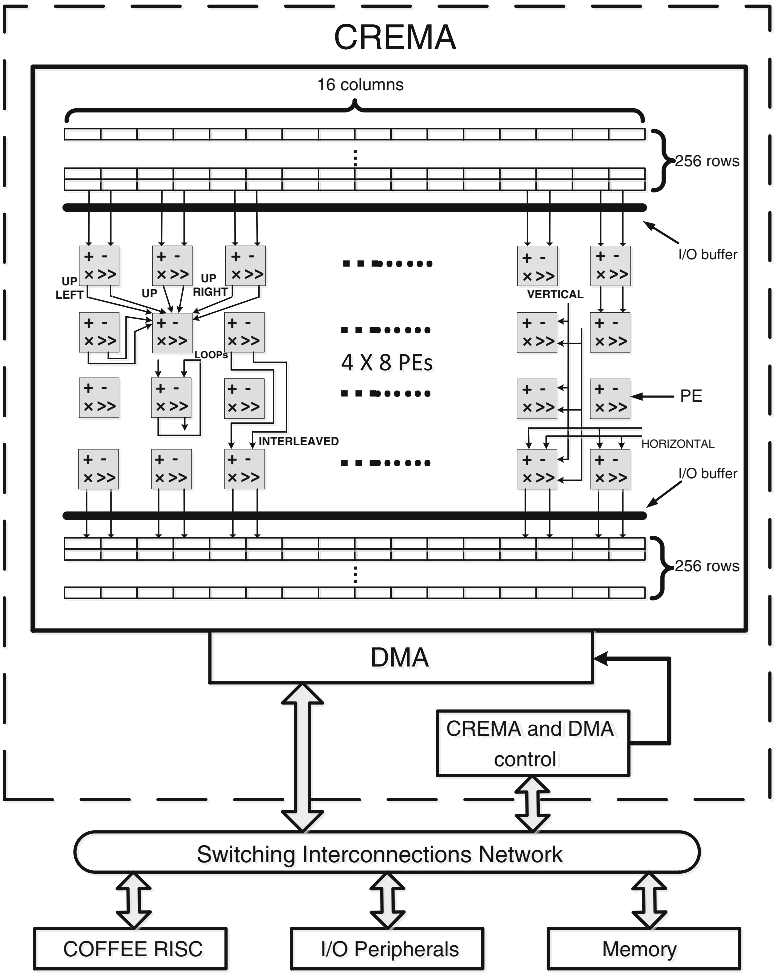


Fig. 4.1 CREMA template © 2009 IEEE [4]

for interleaving data between local memories and PE array. I/O buffers are made of 16 of the 16 × 1 multiplexers and 32 of 32-bit registers. The multiplexers are used for distributing the data from the local memory to the processing array and vice versa where their outputs can be delayed for one clock cycle by the register. The first I/O buffer interleaves the data from the first local memory to the PEs and vice versa for the second I/O buffer. During an operation, the data to be processed is loaded into the local memories by the Direct Memory Access (DMA) device and then processed over the PE array. The results are stored sequentially in the receiving memories. The written data on the second local memory can be directly read during the next iteration by changing the direction of data flow.

As it is shown in Fig. 4.2, every PE receives two operands and can perform 32-bit integer and IEEE-754 format floating-point operations. A PE is composed of a Look-Up Table (LUT), adder, multiplier, shifter, immediate register, and floating-point logic. The blocks with the dashed borders can be selected at design time according to the processing requirements of the applications. The connections among the PEs are in a point-to-point fashion with multiple routing possibilities divided into three groups: local, interleaved, and global. Local interconnections are with the nearest-neighboring PEs such as up-left, up, up-right, and loops. The global connections are used for transferring the data among PEs in both vertical and horizontal directions by using specific point-to-point connections.

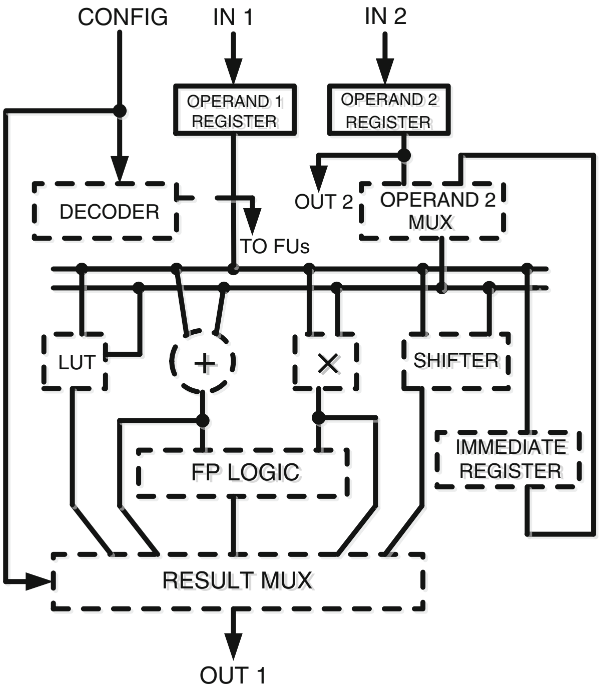


Fig. 4.2 PE core template © 2009 IEEE [4]

Each particular application has to be performed according to its mathematical expressions with the most optimum placement and routing. At any clock cycle, the pattern of interconnections among all PEs and the operations to be performed by the PEs is called context. The functionality and routing of the existing PEs can be switched as required for application processing. Enabling a different context is based on the configuration words stored in the local configuration memories. Each configuration word is composed of an operation filed for specifying the task of each PE and also address field to specify its destination address. Each configuration word includes an operation and address field to specify the task of each PE and its destination address, respectively. The configuration words are sequentially injected into the configuration memories and distributed among the PE array in a pipelined fashion with the help of a DMA device. The CREMA-generated accelerators are programmable in C and COFFEE RISC core performs the control operation by writing control words in the accelerator control registers. As the first step, a graphical platform called Firetool can be utilized for defining the functionalities of the PEs and routing among them. Then configuration files will be generated for mapping and run-time reconfiguration. At the system start-up time, the configuration data is loaded using the DMA device. After configuring the array with a set of contexts, the data to be processed should be loaded into the local memories of CREMA. Then a context should be enabled for configuring the functionalities of the PEs and the routing among them. Once the data is processed over the PE array, a new set of data can be loaded in one of the local memories based on the algorithm flow. These steps may be repeated until the algorithm completes its execution. In this chapter, we further scaled up CREMA to AVATAR (a 4×16 array) [8], as shown later.

4.3 Design Implementation and Algorithm Mapping

In this section, design and implementation of OFDM receiver shown in Fig. 4.3 using template-based CGRAs are studied in detail and fully described. In this experimental work, OFDM data symbols are generated based on IEEE 802.11a/g specifications. Since the experimental platform is a processor/coprocessor model, COFFEE core is the supervisory module in the system besides its general-purpose functionality. Thus, the reference measurements are based on COFFEE (system) clock cycle. In the following, the design and implementation of application-specific accelerators for OFDM receiver blocks on a CGRA is presented.

4.3.1 Time Synchronization

The first block of OFDM receiver after Analog-to-Digital Converter (ADC) is Time Synchronization which is essential for detecting the arrival time of the received OFDM symbols and the right position of Fast Fourier Transform (FFT) window

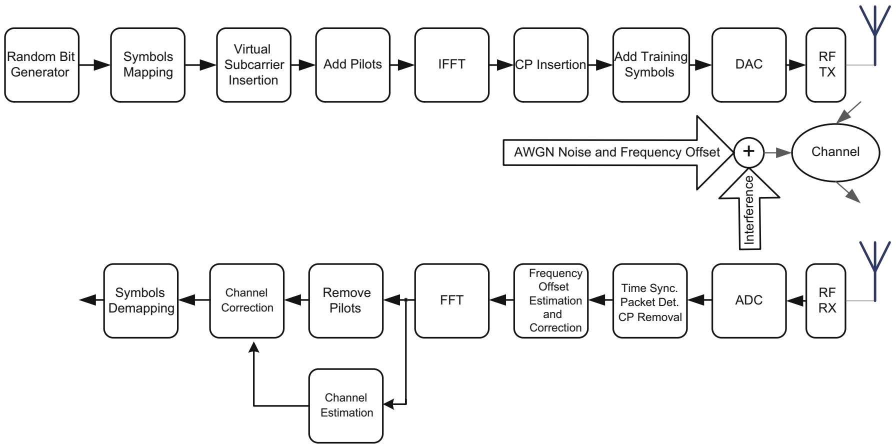


Fig. 4.3 IEEE 802.11a/g simplified transceiver architecture

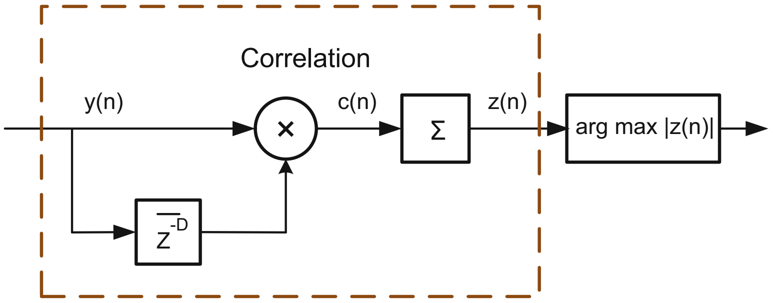


Fig. 4.4 Signal flow structure of time synchronization block by using delay and correlation algorithm © 2015 IEEE

precisely. In different wireless standards, synchronization requires the calculation of the correlation for determining similarity between two signals. The timing synchronization can be performed by using Cyclic Prefix (CP) or Guard Interval (GI) correlation method [18]. In telecommunications, the term CP is introduced for coping with Inter-Symbol Interference (ISI) which refers to the prefixing of a data symbol with a repetition of the end. According to the CP correlation method shown in Fig. 4.4, the received signal (y_n) is correlated with its delayed version. The delay z^{-D} is equal to the length of CP which is 16 based on IEEE 802.11a/g standard specifications. It should be considered that the length of received data symbol according to the mentioned standard is equal to 80 (64 FFT and 16 CP), if there is no multipath propagation. This method will produce outputs c_n and z_n shown in Fig. 4.2 and expressed in (4.1) and (4.2), respectively.

$$c_n = c_n y^*_{n-D} \tag{4.1}$$

$$z_n = \sum_{i=0}^{L-1} c_{i+n} \tag{4.2}$$

In order to have optimal placement and routing over the PE array, (4.1) can be simplified as (4.3) where R and I stand for Real and Imaginary parts of the received signal, respectively.

$$\begin{aligned} c_n = {} & \underbrace{((c_{n(R)} \times y_{n-D(R)}) + (c_{n(I)} \times y_{n-D(I)}))}_{\text{Real}} \\ & + \underbrace{((c_{n(I)} \times y_{n-D(R)}) - (c_{n(R)} \times y_{n-D(I)}))}_{\text{Imaginary}} \end{aligned} \tag{4.3}$$

The mapping of an 80-point correlation algorithm on CREMA according to the above equation can be performed by using two contexts shown in Figs. 4.5 and 4.6. These two contexts must be enabled in a sequence for complete implementation of the above equation. The functional contexts are the same for all PEs. However, the only difference between the two contexts is related to their I/O buffers. Furthermore, some PEs should be switched off for each context which can be performed by disabling the specific parts of I/O buffers. The first context is related to the loading of immediate values required for performing shift operation after each multiplication to prevent any overflows. As the first step, the received data symbols must be loaded into the first local memory and multiplied by the complex conjugation of its delayed version. In order to perform Time Synchronization for 80 data symbols, 80 correlations are required. Both contexts are indicated by the indexes from 0 to 79 which belong to 80-point correlation. For each correlation, a sum-of-products of the results of complex multiplication is required for calculating the final result of each correlation. The summation operation can be performed using feedback addition. During this process, the data symbols are also shifted by one position using Unregistered-Feed Through (URF) feature of the PE. A shifted version of the data is transmitted to the second local memory which allows accomplishment of the next step. In order to achieve higher processing speed and reduce the execution time, the generated accelerators related to the correlation algorithm can be redesigned by scaling-up CREMA which is called AVATAR. AVATAR is composed of a matrix of 5 rows $\times$ 16 columns of PEs and two 32-bit local memories each of size 32 columns $\times$512 rows. I/O buffers are also made of 32 of the 32 $\times$ 1 multiplexers and 32 of 32-bit registers.

The mapping of an 80-point correlation algorithm on AVATAR is depicted in Figs. 4.7 and 4.8. Similarly, as the first step, data should be loaded into the first local memory of AVATAR. Second context has been designed for performing two different tasks which is only used once for the first correlation. The first task is multiplication between the received signal and the complex conjugation of its delayed version, indicated by the indexes from 0 to 79 which belong to 80-point correlation. The second task is related to data distribution for maximizing the

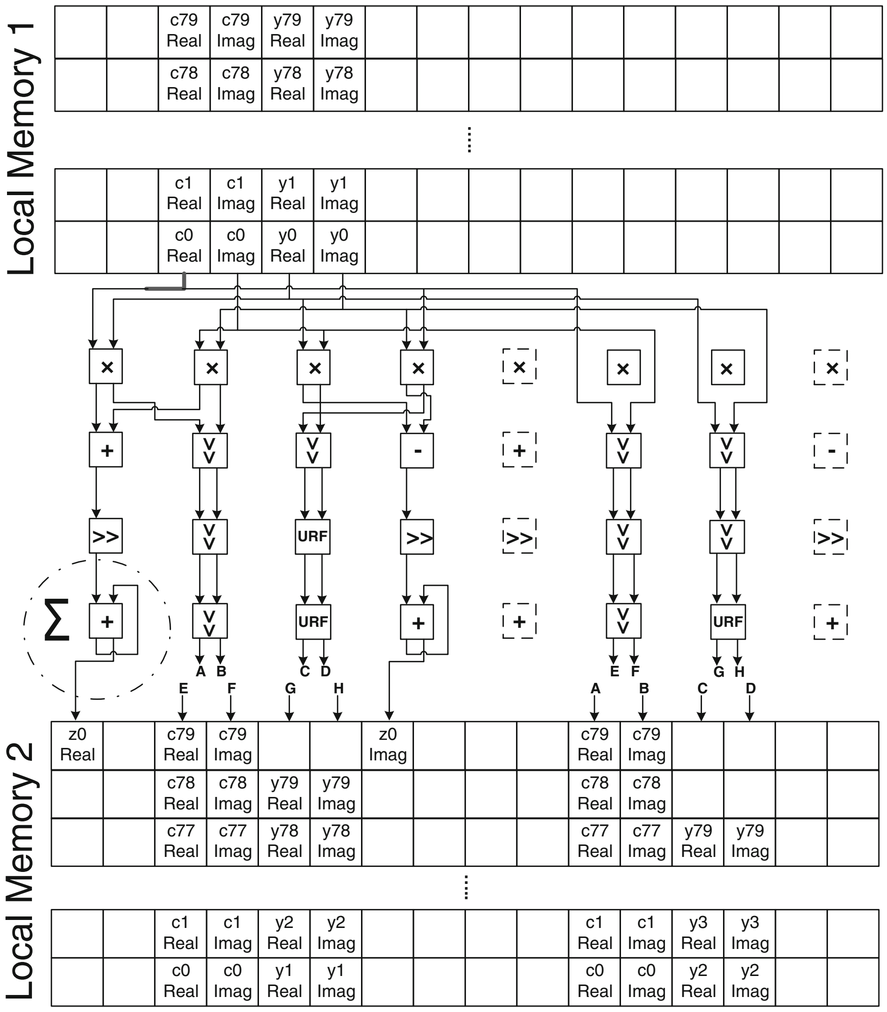


Fig. 4.5 First context for the calculation of the correlations [15] © 2015 IEEE

parallel usage of resources. Then the direction of the data flow will be changed from the second local memory to the first one for processing four correlations in parallel. As it can be observed in Fig. 4.8, four URFs are used in order to shift the delayed version of received data symbols by four-cycle delay during each iteration which allows accomplishment of the next step. It means that each four correlations and data shifting can be performed in parallel using just one context until all 80 correlations complete their execution.

In the next step of Time Synchronization block, the largest value has to be searched by the RISC core as it will indicate the index of the time offset specifying the edge of the first FFT window.

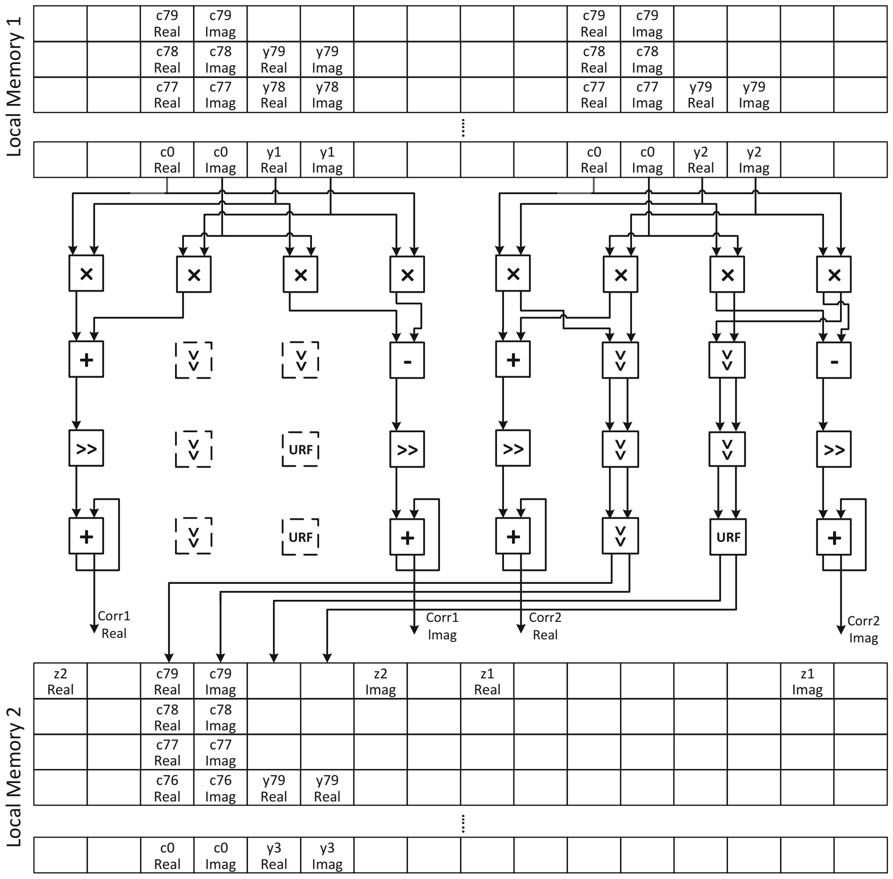


Fig. 4.6 Second context for the calculation of the correlations [15] © 2015 IEEE

$$\hat{\tau}_s = \underset{n}{\operatorname{argmax}} \mid z_n \mid$$
$$= \underset{n}{\operatorname{argmax}} \mid z_{n(R)} \times z_{n(R)} + z_{n(I)} \times z_{n(I)} \mid \quad (4.4)$$

As it can be seen from (4.4), $\hat{\tau}_s$ specifies the largest value among the 80 values of the performed correlations where R and I represent the Real and Imaginary parts. Since the results are complex, the Square Modulus (SM_n) is required to calculate the magnitude of complex numbers. The SM can be mapped on a template-based CGRA by using just two columns of PEs. As it is shown in Fig. 4.9, $n = 0, 1, 2, \ldots, 79$ are representing the results of square modulus stored in the second local memory. Once the computation of square modulus is completed, the results are returned to the RISC core for probing the largest peak as well as the index of time offset. Then the COFFEE RISC core performs search algorithms for this task.

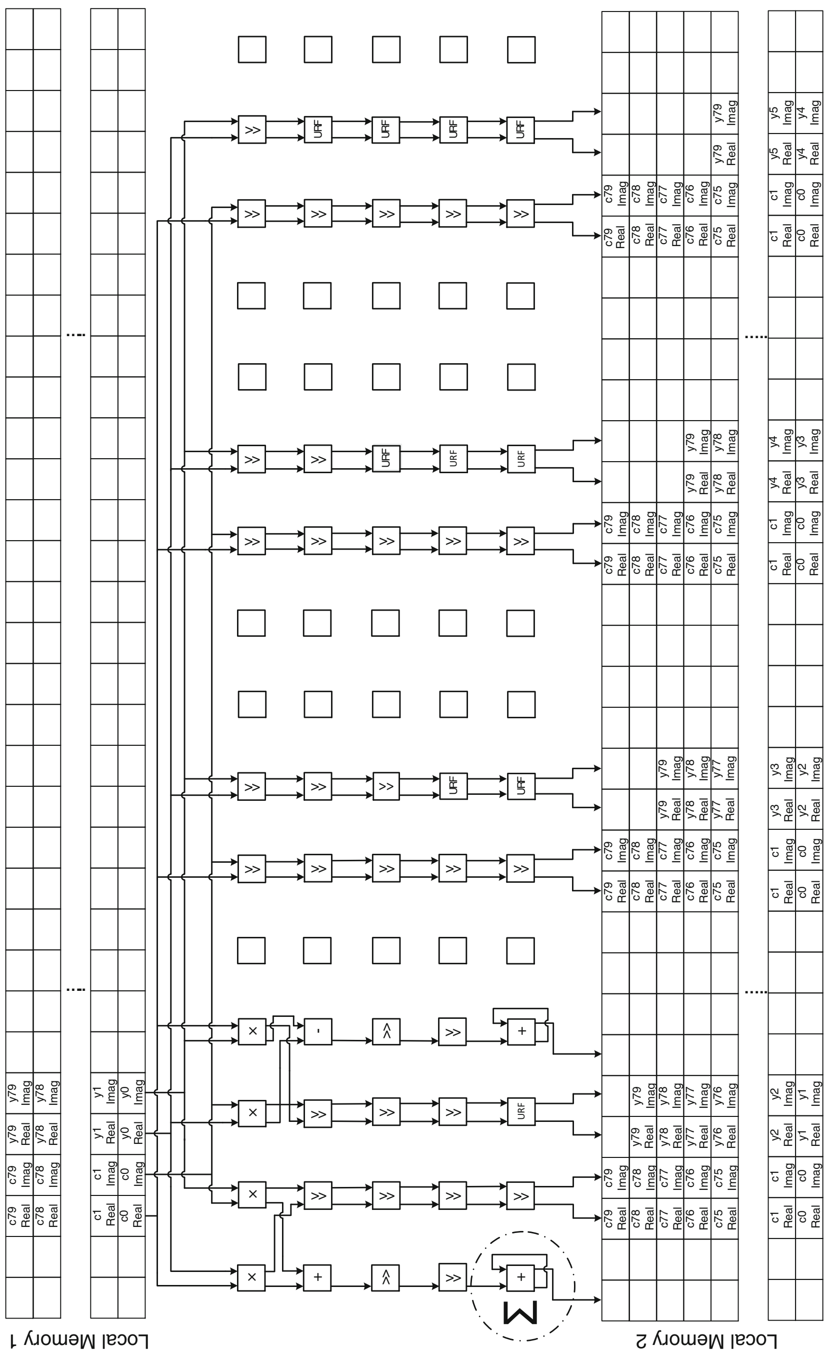


Fig. 4.7 Scaled-up version of second context for the calculation of the correlations

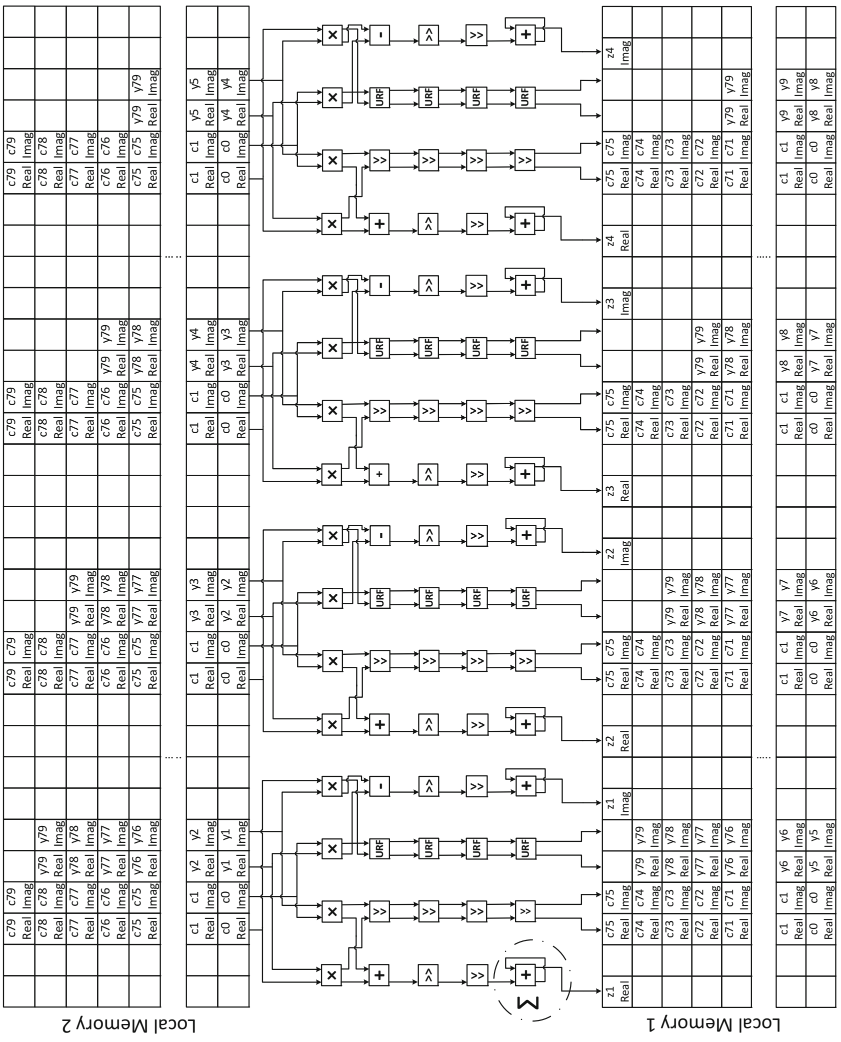


Fig. 4.8 Scaled-up version of third context for the calculation of the correlations

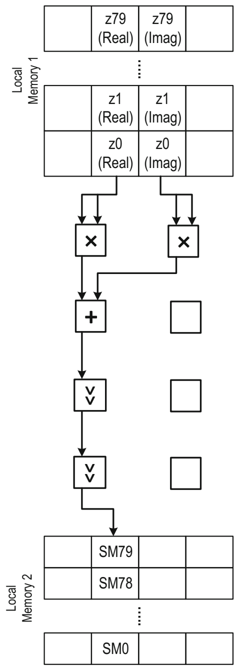


Fig. 4.9 Two columns of the context for the square modulus [15] © 2015 IEEE

The mapping of Time Synchronization on CREMA platform requires three contexts. The immediate values used for performing the shift operation after each multiplication can be loaded in the first context. The second context is used for correlations and the third one is required for SM. The number of clock cycles required depends on the size of the correlation window and therefore they decrease as the correlation window shifts. For example, 80 clock cycles are required (excluding latency due to PE arrays) to perform 80 points correlation. However, in the next iteration, 79 clock cycles are needed to compute 79 points correlation and so on. The 80 correlations are implemented in 4015 system clock cycles in total. There is an overhead between each correlation during the entire correlation kernel because of the control operations that are performed by COFFEE RISC core. Control operations are composed of context switching, reloading of I/O buffers, reconfiguration, and transferring the data from the main memory to local memories or vice versa. Calculation of the square modulus is performed in 192 system clock cycles including communication overhead. Then, the maximum value along with its index (time offset) can be found using COFFEE RISC core software in 835 clock cycles. The overall process of time synchronization can be executed in a total of 5042 system clock cycles by using both CREMA and the processor software. In the case of AVATAR, the total number of clock cycles for the complete Time Synchronization implementation is equal to 1681 which gives us 4.0× speed-up in comparison with CREMA CGRA implementation.

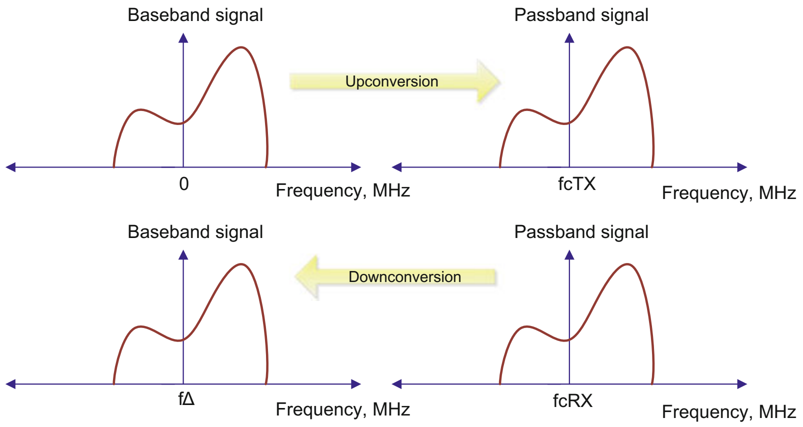


Fig. 4.10 Upconversion and downconversion of signal in transceiver

4.3.2 Frequency Offset Estimation

OFDM waveform is composed of multiple sinusoidal components. As it is depicted in Fig. 4.10, a transmitted signal should be upconverted to carrier frequency before transmission and then downconverted from the same carrier frequency at the receiver side. Because of the device impairments [6], OFDM is sensitive to Carrier Frequency Offset (CFO). Therefore, the received baseband signal might be centered at f_Δ instead of zero which is equal to the difference between the carrier frequencies in transmitter and receiver side. One of the methods for estimating the CFO is using short training sequences that are added by the transmitter. Lets assume that r_n is the received signal and can be expressed as

$$r_n = x_n e^{j2\pi f_\Delta n T_s}, \tag{4.5}$$

where f_Δ stands for the frequency offset and x_n is the transmitted signal. Frequency offset can be estimated by using delay and correlation method based on (4.6).

$$\begin{aligned} z &= \sum_{n=0}^{L-1} r_n r_{n+D}^* \\ &= \sum_{n=0}^{L-1} ((\underbrace{(r_{n(R)} \times r_{n+D(R)}) + (r_{n(I)} \times r_{n+D(I)})}_{\text{Real}}) \\ &\quad + \underbrace{((r_{n(I)} \times r_{n+D(R)}) - (r_{n(R)} \times r_{n+D(I)})}_{\text{Imaginary}})) \end{aligned} \tag{4.6}$$

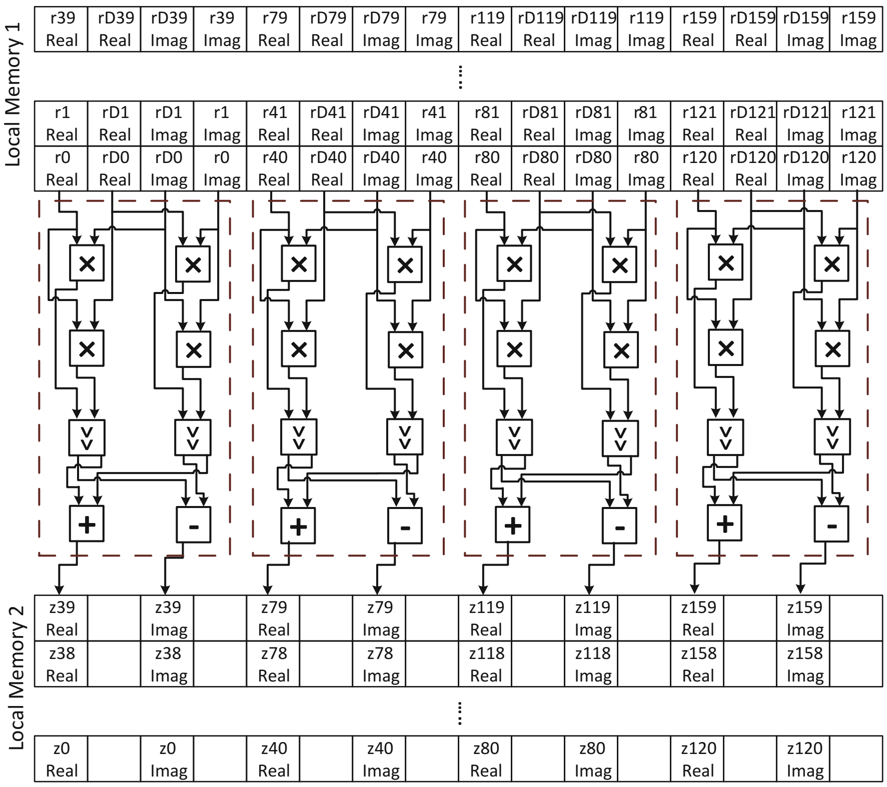


Fig. 4.11 Context for the multiplication between a signal and its complex conjugation [15] © 2015 IEEE

According to IEEE 802.11a/g specifications, the value of delay D is equal to 16 which is calculated by multiplying the period of short training symbols and frequency space (0.8 μs ×20 MHz). The short training symbols can be used for frequency offset estimation and are composed of 160 predefined data symbols added by the transmitter. Accordingly, there is a need to compute 160 complex multiplications. The main processing context of the accelerator generated is shown in Fig. 4.11, where r and rD stand for received signal and its delayed version, respectively. Subsequent to the multiplication among the received signal and the complex conjugation of its delayed version, the frequency offset estimation can be expressed as

$$\hat{f_\Delta} = -\frac{1}{2\pi D T_s} \angle z, \tag{4.7}$$

where T_s is the sampling period and $\angle$ takes the angle of z. For a complex number $x + iy$, the phase angle equation can be expressed as (4.8).

$$\hat{\theta} = \text{atan}\left(\frac{y}{x}\right) \tag{4.8}$$

At first, an imaginary part should be divided by the real part and then, the phase angle can be computed by using ATAN function. It is feasible to execute the division operation in processor software using COordinate Rotation DIgital Computer (CORDIC) algorithm [19]. It is an efficient algorithm for the calculation of trigonometric and hyperbolic functions (functions of an angle), including exponential and logarithmic. In addition, it might be used for other purposes containing complex number multiplication and division. The CORDIC algorithm for calculating the division operation can be written based on the following C code. The accuracy can also be increased by increasing the number of *MaxBits* at the cost of more execution time.

```
for (i = 0; i < MaxBits; i++)
{
if (y < 0 || z >= 0)
{
y = y + x*t;
z = z - t;
}
else
{
y = y - x*t;
z = z + t;
}
t = t >> 1;
}
```

Once the division operation is performed, the phase angle should be computed. In this case, *atan* function can be implemented by using several methods such as CORDIC and Taylor series. However, Taylor series is selected due to its simplicity and shorter execution time in COFFEE RISC processor software. Taylor series is an expansion of a particular function into an infinite sum of terms about a point which can be written as (4.9).

$$\text{atan}\, x = \sum_{n=0}^{m} \frac{-1^n}{2n+1} x^{2n+1} \text{ for } -1 < x < 1 \tag{4.9}$$

Here, the value of m can be chosen based on the proposed accuracy. Finally, the frequency offset correction can be implemented by multiplying the estimated frequency offset and the received signal as represented in (4.10) where $r_n{}'$ is the corrected signal, n is the sample index, and N is the number of samples in a symbol.

$$r_n{}' = r_n \times e^{-j2\pi f_\Delta \frac{n}{N}} \tag{4.10}$$

The exponent of $-j2\pi f_{\Delta}\frac{n}{N}$ can be expanded by using Taylor series as (4.11).

$$\begin{aligned} e^x &= 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \frac{x^4}{4!} + \cdots \text{ for } -\inf < x < \inf \\ &= \sum_{n=0}^{\infty} \frac{x^n}{n!} \end{aligned} \tag{4.11}$$

The above equation can be rewritten for a complex number z as

$$e^z = e^x(\cos(y) + i\sin(y)), \tag{4.12}$$

where z is composed of real part (x) and imaginary part (y). The cos and sin functions can also be expanded by using Taylor series as a sum series which are expressed in (4.13) and (4.14), respectively.

$$\begin{aligned} \cos\ y &= 1 - \frac{y^2}{2!} + \frac{y^4}{4!} - \frac{y^6}{6!} + \cdots \text{ for } -\inf < y < \inf \\ &= \sum_{n=0}^{\infty} \frac{(-1)^n}{(2n)!} y^{2n} \end{aligned} \tag{4.13}$$

$$\begin{aligned} \sin\ y &= y - \frac{y^3}{3!} + \frac{y^5}{5!} - \frac{y^7}{7!} + \cdots \text{ for } -\inf < y < \inf \\ &= \sum_{n=0}^{\infty} \frac{(-1)^n}{(2n+1)!} y^{2n+1} \end{aligned} \tag{4.14}$$

Finally, the received signal must be multiplied by the correction factor calculated above using CREMA-generated accelerator with almost the same context as the first one. As it is shown in Fig. 4.11, 160 complex numbers (320 real integer numbers) are injected into the local memories in a way that the four complex multiplications can be performed in parallel. Thus, the 160-point complex multiplications are performed in 40 clock cycles in addition to 4 clock cycles latency of the PE arrays. The complex multiplication results (320 real integers) are transferred back to the main memory using the DMA device which requires 429 system clock cycles. The division operation for calculating the phase angle of a single complex number requires 163 clock cycles. In the total of 160 complex numbers, the first 16 values are zero due to the delay D mentioned in Eq. (4.6). As the remaining complex numbers are 144, so the total number of system cycles required for all the phase angle calculations are $144 \times 163 = 23{,}472$ clock cycles. The correction factor for 80 complex data symbols mentioned in Eq. (4.8) is also implemented in processor software. This process requires 5868 system clock cycles. The 80-point complex multiplication for performing the frequency offset correction also requires 30 system clock cycles on CREMA-generated accelerator.

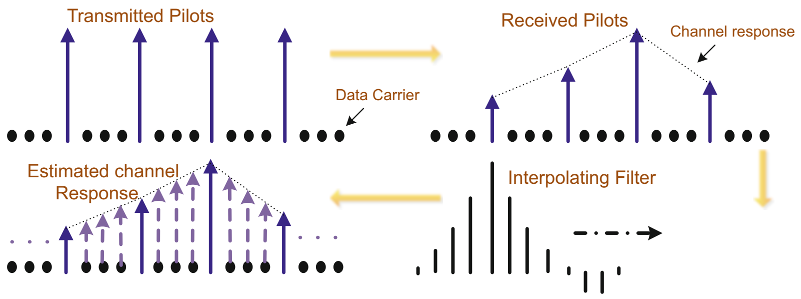


Fig. 4.12 Channel estimation based on linear interpolation

4.3.3 Channel Estimation

Transmitted data symbols after passing through the wireless channel may get destroyed because of various impairments which require correction. Subsequent to the recovering of the data symbols in FFT block [7], the channel frequency response should be estimated. This issue can be performed in channel estimation block by using the long training sequences or the pilots (shown in Fig. 4.12) which are added in the transmitter side and known to the receiver.

The received signal Y_n can be expressed as follows:

$$Y_n = X_n H_n + N_n, \tag{4.15}$$

where n is representing the number of subcarriers, H_n stands for channel impulse response, and N_n is the additive noise. In the case of a linear channel, there are two steps for performing channel correction [16]:

- Channel estimation attempts to estimate H_n
- Channel equalization attempts to correct Y_n based on X_n

One of the methods is pilot-assisted linear interpolation algorithm. There are some known training symbols for the receiver like pilots in WLAN OFDM which can be employed for various targets. According to the IEEE 802.11a/g specifications, four specific values are inserted between data subcarriers as pilots prior to transmitting the signal. The channel impulse response can be mathematically expressed as

$$\tilde{H}_k = M^{-1} P_{Rx}, \tag{4.16}$$

where k is the number of pilots, P_{Rx} represent the received noise-impaired pilots, and $\tilde{H}_k$ is representing the channel impulse response of received pilots. Furthermore, M can be represented as a diagonal matrix formed from transmitted pilots according to (4.17).

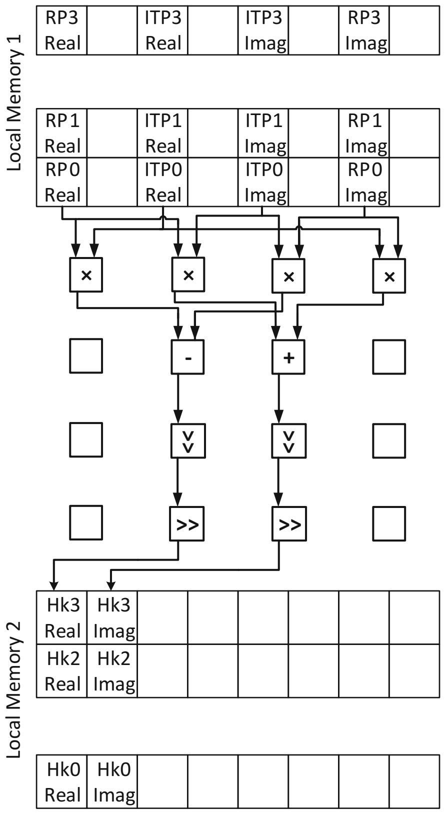


Fig. 4.13 First context for the channel estimation [15] © 2015 IEEE

$$M = \begin{bmatrix} M_{1,1} & 0 & \cdots & 0 \\ 0 & M_{2,2} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & M_{k,k} \end{bmatrix} \tag{4.17}$$

As it is shown in Fig. 4.13, the first context of channel estimation can be mapped on CREMA by using only four columns of PEs in order to calculate the complex multiplication between the Received Pilots (RP) and Inverse of Transmitted Pilots (ITP). As the next step, the channel frequency response related to the other subcarriers should to be calculated. It can be performed by using linear interpolation method for approximating the value among two data samples. Equation (4.18) is used for linear interpolation by expanding the channel frequency response of four received pilots $\hat{H}_n$ and then estimating the channel frequency response of the remaining subcarriers $\tilde{H}_k$.

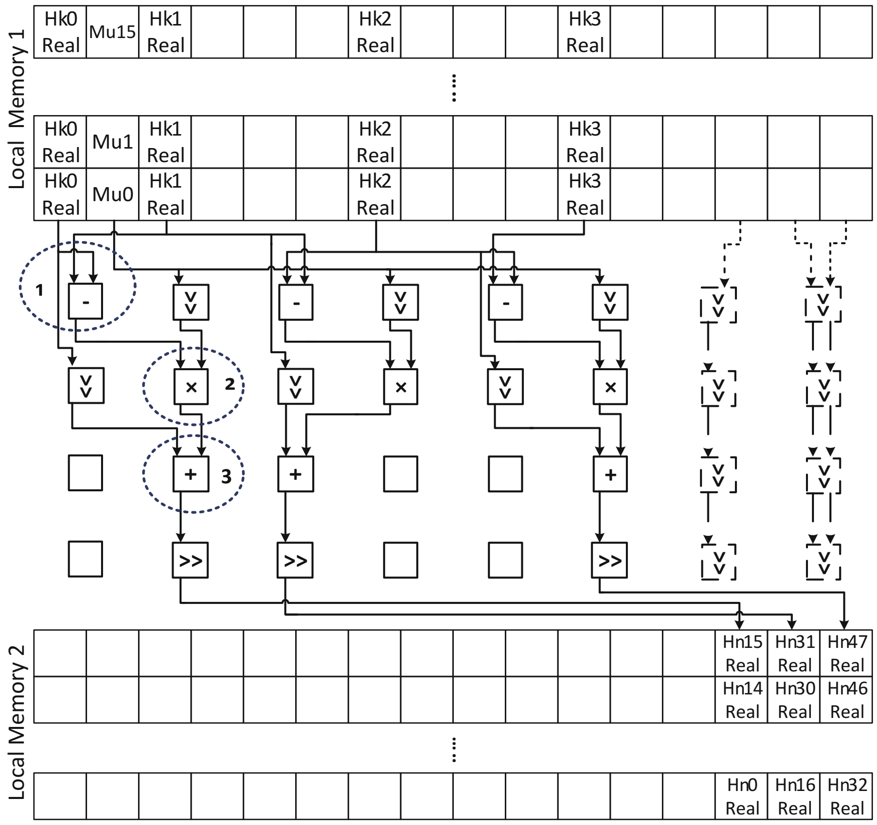


Fig. 4.14 First context for the linear interpolation [15] © 2015 IEEE

$$\hat{H}_n = \sum_{i=1}^{N_p-1}\sum_{j=1}^{N_s}\tilde{H}_k(i) \underbrace{+}_{3} \left((\tilde{H}_k(i+1) \underbrace{-}_{1} \tilde{H}_k(i)) \underbrace{\times}_{2} \underbrace{\frac{j-1}{N_s}}_{\mu} \right) \tag{4.18}$$

Here N_p, N_s, and μ are representing the number of pilots, the number of samples between the two pilots, and the step size, respectively.

As it is shown in Figs. 4.14 and 4.15, the step size has to be loaded along with the real and imaginary parts which can take 16 different fixed values. Equation (4.18) can be mapped on CREMA in three steps that is also shown in Fig. 4.14. Furthermore, delay operations are used for preprocessing the data. The last two columns of PEs (dashed border) are not active in the first context and can be instantiated in the second one for transferring the results of the real part of

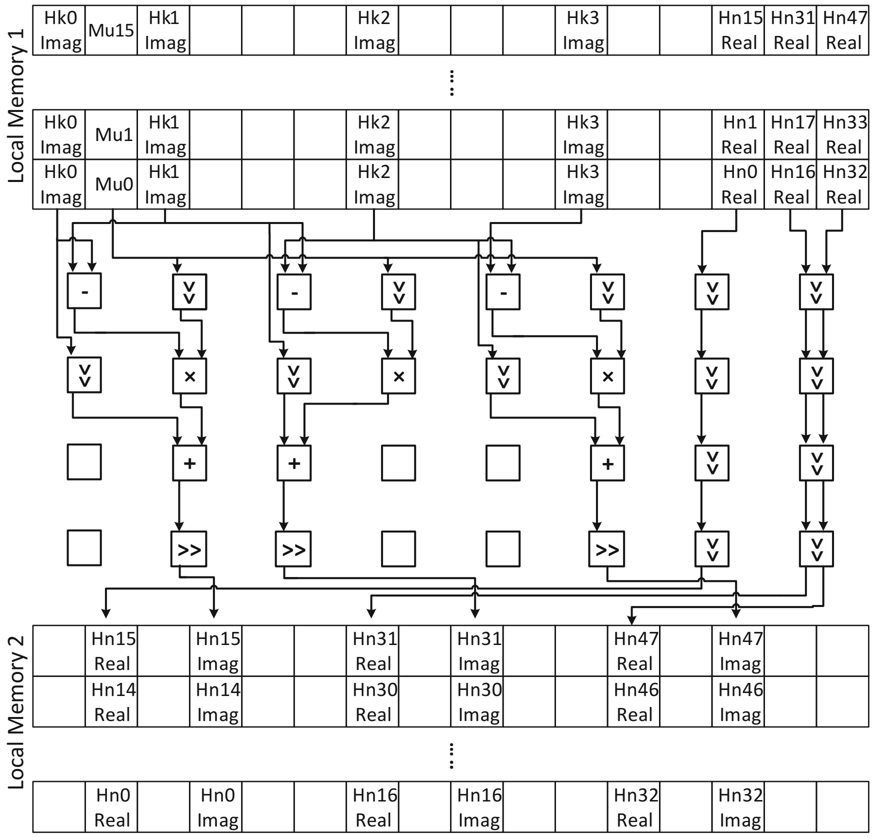


Fig. 4.15 Second context for the linear interpolation

linear interpolation. Once linear interpolation is performed, the computed channel frequency response is returned to the main memory of RISC core as it will be needed again for performing channel equalization.

After estimating the channel frequency response, channel equalization is the next step for correcting the noisy data symbols Y_n the closest possible to X_n. Therefore, the received signal should be divided by its channel frequency response as expressed in (4.19).

$$\hat{Y}_n = \frac{Y_n}{\hat{H}_n} \tag{4.19}$$

Although the CORDIC algorithm can be employed for performing the division operation, but another algorithm is proposed to be used in this case since the mapping of CORDIC algorithm on template-based CGRA is not efficient due to using iteration in that. In this case, the denominator can be assumed a fixed value due to the specific pilots which are added in the transmitter. The selected algorithm for executing the division operation is **Newton–Raphson** method [17] as an algorithm for computing the root of an equation expressed as

$$x_{n+1} = x_n + \frac{f(x_n)}{f'(x_n)}, \tag{4.20}$$

where n is the number of iteration started from zero and x_n is the initial guess of the root. In order to use the above equation for the division operation, it has to be modified according to Eq. (4.21) where D stands for the denominator.

$$\begin{aligned} x_{n+1} &= x_n + \frac{\frac{1}{x_n} - D}{-\frac{1}{x_n^2}} \\ &= x_n + \left(\frac{\frac{1}{x_n} - D}{-\frac{1}{x_n^2}} \times \frac{-x_n^2}{-x_n^2} \right) \\ &= x_n - (-x_n + Dx_n^2) \\ &= 2x_n - Dx_n^2 \\ &= x_n.(2 - Dx_n) \end{aligned} \tag{4.21}$$

Since the denominator might be a complex number because of the noisy channel, it would be simplified to a number for mapping on CGRA based on (4.22).

$$\frac{x+iy}{a+ib} \times \frac{a-ib}{a-ib} = \frac{(x+iy)\times(a-ib)}{a^2+b^2} \tag{4.22}$$

Here $x + iy$, $a + ib$, and $a - ib$ stand for demodulated data symbols after FFT block, estimated channel response, and complex conjugation of the channel response, respectively.

As it is shown in Figs. 4.16 and 4.17, in order to perform channel equalization, Newton–Raphson method should be mapped on CREMA for computing the value of $\frac{1}{a^2+b^2}$ where D is equivalent to $(a^2 + b^2)$ based on Eq. (4.21). Here a and b are representing the real and imaginary values of the channel frequency response. The execution of Newton–Raphson method according to Eq. (4.21) can be implemented by tailoring the CREMA as two contexts. During the first context related to Newton–Raphson algorithm, multiplication among the real and imaginary values, addition, and then multiplication between the result of addition and the initial guess (Dx_n) have to be performed, respectively. During the second context, the final result of division $(\frac{1}{a^2+b^2})$ is computed and then transferred back to the main memory of RISC core using the DMA device. In the last step, the complex multiplication among the demodulated data symbols and the complex conjugation of estimated channel frequency response should be performed in another context shown in Fig. 4.18. Once the complex multiplication is performed, the results of Newton–Raphson

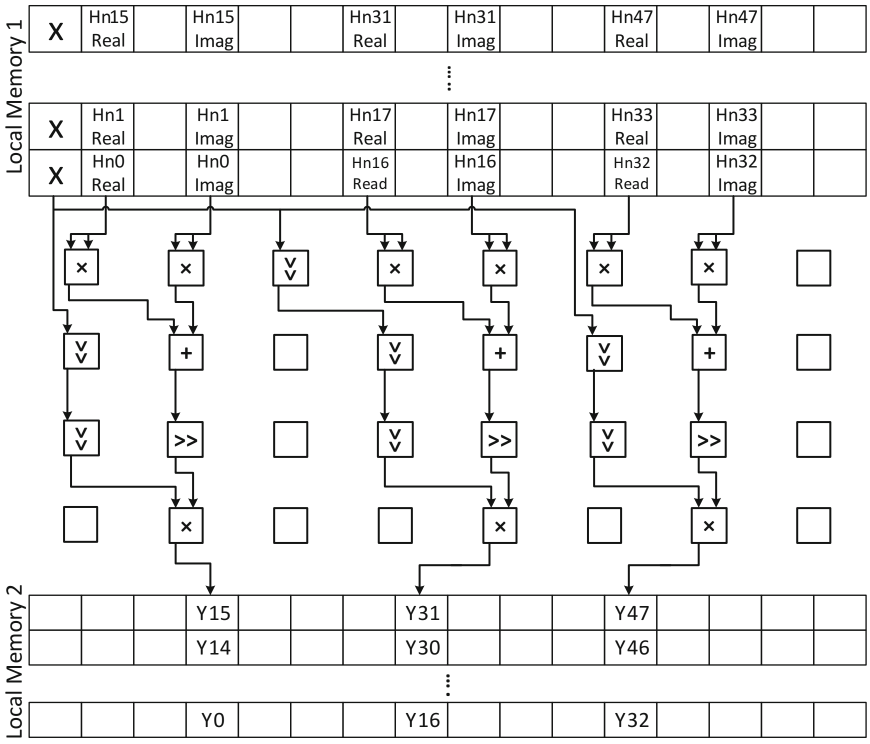


Fig. 4.16 First context for the Newton–Raphson method [15] © 2015 IEEE

division are transmitted back from the main memory to the CGRA to perform channel equalization. Within this stage, multiplication as well as the shift operation is required to produce the final product shown in Figs. 4.19 and 4.20. Here Res_i stands for equalized received data.

The design and implementation of an application-specific accelerator for channel estimation block is performed completely using CREMA with eight contexts and seven I/O buffers. As it is shown in Table 4.1, the overall processing of channel estimation requires 686 system clock cycles.

4.4 Experimental Results and Conclusion

The generated accelerators are synthesized for Altera Stratix-V 5SGXEA4H1F35C1 FPGA device. Table 4.2 represents the resource utilization summaries of each of the designed accelerator on FPGA device. The consumption of ALMs on the FPGA device in case of AVATAR is almost three times more than CREMA because

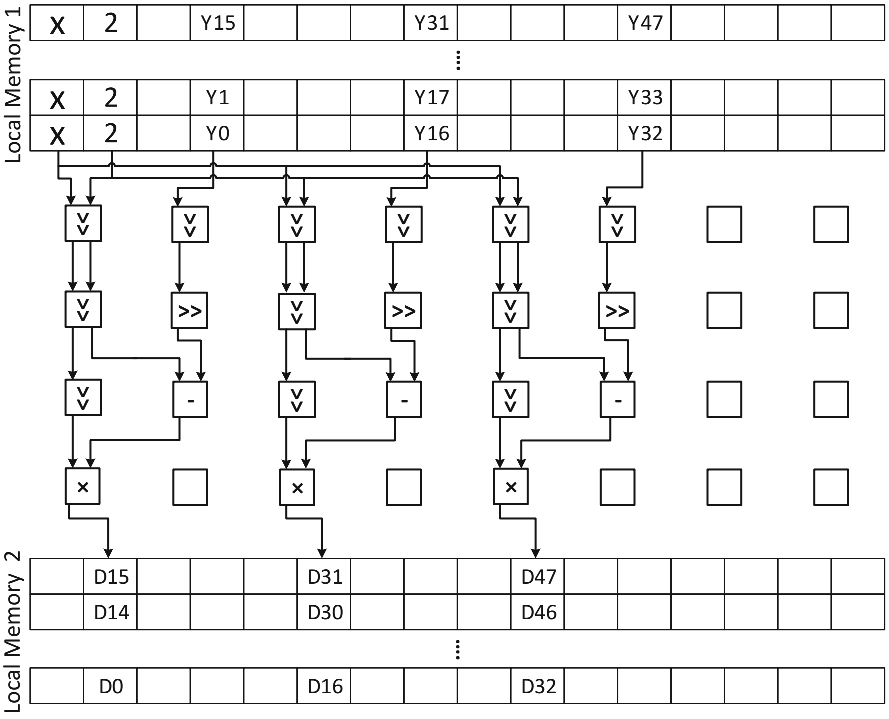


Fig. 4.17 Second context for the Newton–Raphson method [15] © 2015 IEEE

of increasing the size of the PE array and the multiplexers in the I/O buffers. Furthermore, the DSP consumption of AVATAR is twice in comparison with CREMA which is related to the number of 32-bit multiplication. The maximum operating frequencies of the accelerators generated are shown in Table 4.3 at two different temperature in the fast and slow timing models and in two different categories, i.e., the clock frequency of PE array and system. It can be observed that the operating frequency of CREMA-generated accelerator is roughly 1.3× higher than the operating frequency of AVATAR.

Power estimation has been implemented using PowerPlay Power Analyzer Tool of Quartus II 15.0. To estimate power dissipation, the postfit gate-level netlist for each accelerator is generated and then the timing simulation is performed which is the most accurate method. A Value Change Dump (VCD) file is used for power estimation which is generated using ModelSim software. The VCD file contains signal transition activity information of the design [3]. The estimated power dissipation for each designed accelerator is shown in Table 4.4. The power analyzer tool showed a "HIGH" level of confidence for all estimations. From the table, it can be concluded that the static power which is required to keep the device in the ON state is almost the same for both platforms due to the unused portion of the FPGA

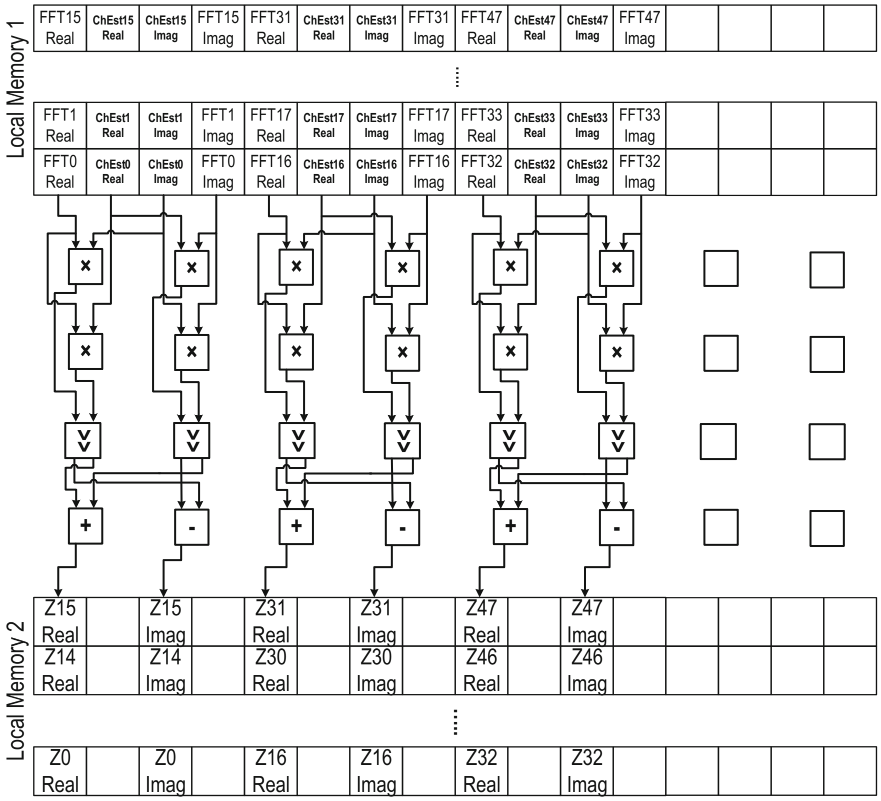


Fig. 4.18 Sixth context for the channel estimation

chip. However, AVATAR requires 1.5× dynamic power consumption in comparison with the CREMA as the size of CGRA increases which results to increase the number of signals those have switching activity at a specific time instant.

The speed-up for each designed accelerator is computed according to the product of the total number of clock cycles and clock period which is called execution time. Thus, the overall speed-up is equal to the ratio of old execution time to the new one for a system. Table 4.5 depicts that the performance of CREMA and in its scaled-up version, AVATAR, gives a significant speed-up compared to the COFFEE RISC software.

In this chapter, three processing blocks related to OFDM baseband receiver are designed and implemented. The implementation is performed using a template-based CGRA which can be tailored to the computational requirements of the target algorithm. In comparison with a Reduced Instruction-Set Computing (RISC) processor software, the implementation of Time Synchronization and Frequency Offset Estimation on the CGRA shows speed-ups for different subsets of their algorithms. Furthermore, the Channel Estimation implementation on CGRA shows

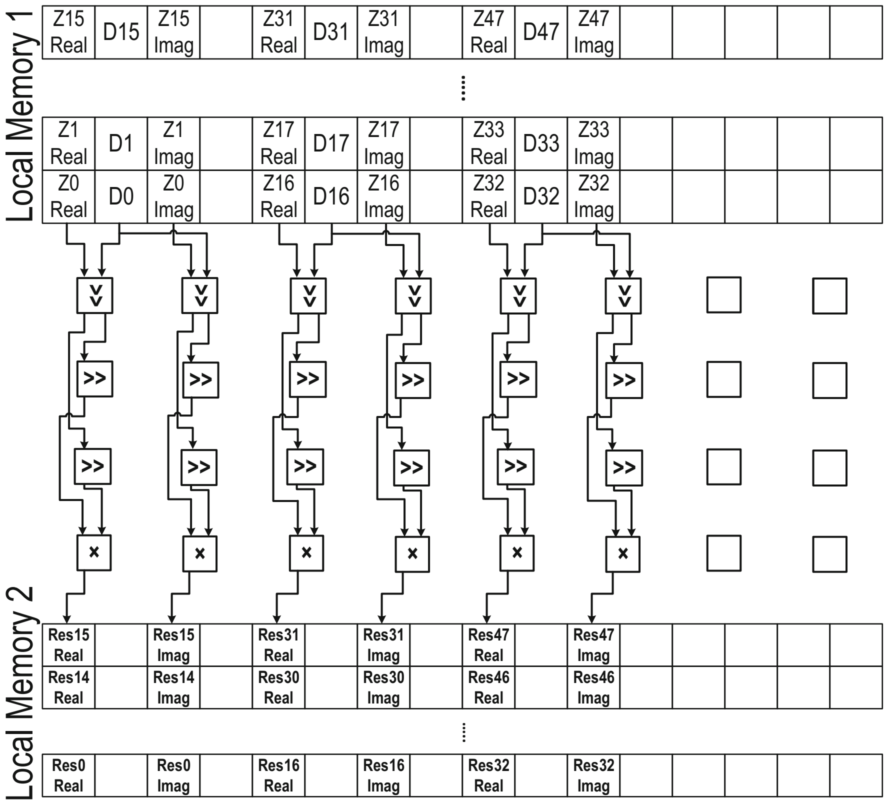


Fig. 4.19 Seventh context for the channel estimation

an overall speed-up 9.2× in comparison with the RISC core. The measured power dissipation for designed accelerators are also presented in this experimental work. In future, the generated accelerators can be redesigned by using different CGRA templates made by scaling-up CREMA for achieving higher processing speed and reducing the execution time at the cost of more resource utilization and higher transient power dissipation.

Acknowledgements This research work is conducted by the Department of Electronics and Communications Engineering, Tampere University of Technology, Finland. It was funded by the Academy of Finland under contract # 258506 (DEFT: Design of a Highly parallel Heterogeneous MP-SoC Architecture for Future Wireless Technologies).

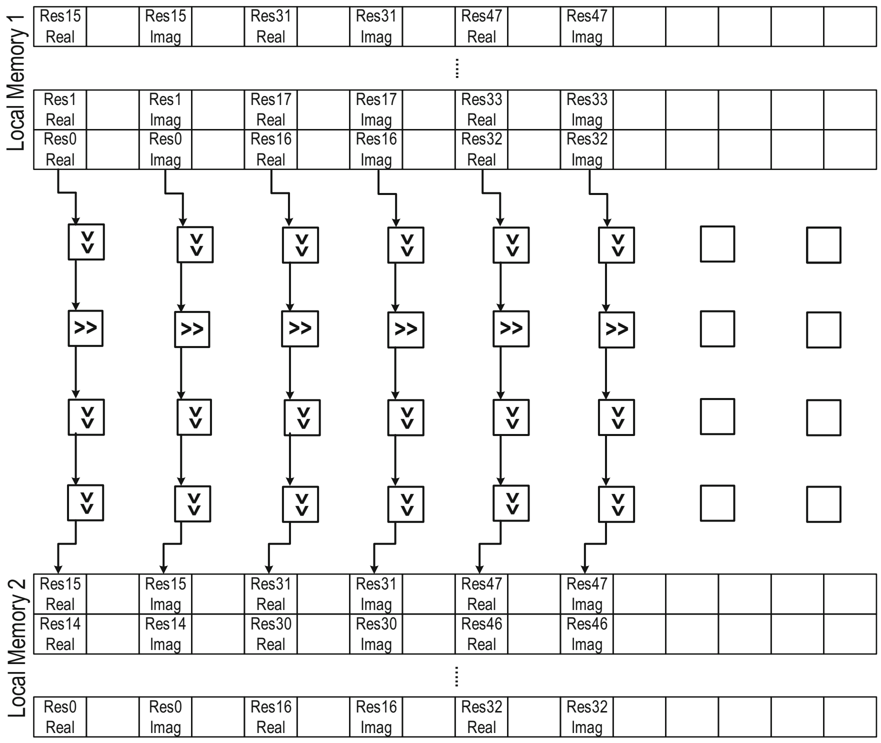


Fig. 4.20 Eighth context for the channel estimation

Table 4.1 Cost for different steps of channel estimation [15] © 2015 IEEE

S/No	Execution steps	Clock cycles (CC)
1	Overhead	31 CC
2	Stage-1 complex multiplication	5 CC
3	Overhead	342 CC
4	Stage-2 linear interpolation	30 CC
5	Overhead	15 CC
6	Stage-3 Newton–Raphson method	30 CC
7	Overhead	163 CC
8	Stage-4	14 CC
9	Overhead	16 CC
10	Stage-5	14 CC
11	Overhead	12 CC
12	Stage-6	14 CC
	Total	686 CC

Table 4.2 Resource utilization summary of proposed accelerators

Accelerator	Resource utilization			
	ALMs	Registers	Memory bits	DSP
Time synchronization (CREMA)	6777 (4 %)	8196	267,520 (<1 %)	16
Time synchronization (AVATAR)	23,417 (15 %)	16,072	536,320 (1 %)	32
Frequency offset estimation	9316 (6 %)	9188	267,520 (<1 %)	32
Channel estimation	10,075 (6 %)	9566	267,968 (<1 %)	40

Table 4.3 Synthesis frequencies of designed accelerators

Accelerator	Max frequency 85 °C | 0 °C			
	Slow model (MHz)		Fast model (MHz)	
	clk_crema	clk_sys	clk_crema	clk_sys
Time synchronization (CREMA)	271 | 291	347 | 397	376 | 419	504 | 531
	clk_avatar	clk_sys	clk_avatar	clk_sys
Time synchronization (AVATAR)	220 | 249	300 | 336	262 | 289	496 | 526
	clk_crema	clk_sys	clk_crema	clk_sys
Frequency offset estimation	168 | 196	285 | 330	271 | 297	426 | 468
	clk_crema	clk_sys	clk_crema	clk_sys
Channel estimation	191 | 204	267 | 310	274 | 305	391 | 426

Table 4.4 Power dissipation of designed accelerator

Accelerator	Static power (mW)	Dynamic power (mW)	I/O power (mW)	Total power (mW)
Time synchronization (CREMA)	990	789	30	1809
Time synchronization (AVATAR)	1016	1202	31	2249
Frequency offset estimation	985	639	43	1667
Channel estimation	975	348	55	1378

Table 4.5 Performance comparison in clock cycles between COFFEE RISC core software and CREMA by considering the clock frequency of system at 100 MHz [10]

Accelerator	Application	RISC	CGRA	Gain
Time synchronization	Correlation (CREMA)	12,800	4015	3.2×
Time synchronization	Correlation (AVATAR)	12,800	453	28.3×
Time synchronization	Square modulus	1680	192	8.8×
Time synchronization	Overall (By considering DMA transfer on CREMA)	15,315	5042	3.0×
Time synchronization	Overall (By onsidering DMA transfer on AVATAR)	15,315	1681	9.1×
Frequency offset estimation	160-point complex multiplication	4338	467	9.3×
Frequency offset estimation	80-point complex multiplication	2338	842	2.8×
Channel estimation	Overall	6274	686	9.2×

References

1. Baumgarte, V., Ehlers, G., May, F., Nuckel, A., Vorbach, M., Weinhardt, M.: PACT-XPP A self-reconfigurable data processing architecture. J. Supercomput. **26**(2), 167–184 (2003)
2. Brunelli, C., Garzia, F., Nurmi, J.: A coarse-grain reconfigurable architecture for multimedia applications featuring subword computation capabilities? J. Real-Time Image Proc. **3**(1–2), 21–32 (2008). doi:10.1007/s11554-008-0071-3
3. Altera Corporation: Design implementation and optimization quartus-II handbook version 13.1, vol. 2. Altera Corporation (2013)
4. Garzia, F., Hussain, W., Nurmi, J.: CREMA, A coarse-grain re-configurable array with mapping adaptiveness. In: Proceedings of 19th International Conference on Field Programmable Logic and Applications (FPL 2009). IEEE, Prague (2009)
5. Garzia, F., Hussain, W., Airoldi, R., Nurmi, J.: A reconfigurable SoC tailored to software defined radio applications. In: Proceedings of 27th Norchip Conference, Trondheim (2009)
6. Heiskala, J., Terry, J.: OFDM Wireless LANs: A Theoretical and Practical Guide. Sams Publishing, Indianapolis, IN (2002)
7. Hussain, W., Garzia, F., Nurmi, J.: Exploiting control management to accelerate Radix-4 FFT on a reconfigurable platform. In: Proceedings of International Symposium on System-on-Chip 2010, pp. 154–157. IEEE, Tampere (2010). ISBN: 978-1-4244-8276-4
8. Hussain, W., Ahonen T., Garzia, F., Nurmi, J.: Energy and power estimation of coarse-grain reconfigurable array based fast Fourier transform accelerators. In: Proceedings of 7th International Workshop on Reconfigurable Communication-Centric Systems-on-Chip (ReCoSoC), pp. 1–4 (2012)
9. Kunjan, P., Bleakley, C.: Systolic algorithm mapping for coarse grained reconfigurable array architectures. In: Reconfigurable Computing: Architectures, Tools and Applications. Lecture Notes in Computer Science, vol. 5992, pp. 351–357. Springer, Berlin, Heidelberg (2010). doi:10.1007/978-3-642-12133-3 33
10. Kylliäinen, J., Ahonen, T., Nurmi, J.: General-purpose embedded processor cores - the COFFEE RISC example. In: Nurmi, J. (ed.) Processor Design: System-on-Chip Computing for ASICs and FPGAs, Chap. 5, pp. 83–100. Kluwer/Springer, Dordrecht/Berlin (2007). ISBN-10: 1402055293, ISBN-13: 978-1-4020-5529-4
11. Lee, M.-H., Singh, H., Guangming, L., Bagherzadeh, N., Kurdahi, F.J., Filho, E.M.C., Vladimir, C.A.: Design and implementation of the morphosys reconfigurable computing processor. J. VLSI Sig. Proc. Systems Signal Image Video Technol. **24**, 147–164 (2000)
12. Lo, C.-C., Tsai, S.-T., Shieh, M.-D.: A reconfigurable architecture for entropy decoding and IDCT in H.264. In: International Symposium on VLSI Design, Automation and Test, VLSI-DAT '09, 28–30 April 2009, pp. 279–282. doi:10.1109/VDAT.2009.5158149. ISBN: 978-1-4244-2781-9
13. Mei, B., Vernalde, S., Verkest, D., Man, H.D., Lauwereins, R.: ADRES: an architecture with tightly coupled VLIW processor and coarse-grained reconfigurable matrix. In: Field-Programmable Logic and Applications, vol. 2778, pp. 61–70 (2003). ISBN 978-3-540-40822-2
14. Mitola, J., Maguire, G.Q.: Cognitive radio: making software radios more personal. IEEE Pers. Commun. **6**(4), 13–18 (1999)
15. Nouri, S., Hussain, W., Nurmi, J.: Implementation of IEEE-802.11a/g receiver blocks on a coarse-grained reconfigurable array. In: 2015 Conference on Design and Architectures for Signal and Image Processing (DASIP), pp. 1–8 (2015). doi:10.1109/DASIP.2015.7367254
16. Pun, M.-O., Morelli, M., Kuo, C.-C.J.: Multi-Carrier Techniques for Broadband Wireless Communications: A Signal Processing Perspective. Imperial College Press, London (2007)
17. Ryaben'kii, V.S., Tsynkov, S.V.: A Theoretical Introduction to Numerical Analysis, p. 243. CRC, Boca Raton, FL (2006)
18. van de Beek, J.-J., Borjesson, P.O., Boucheret, M.-L., Landstrom, D., Arenas, J.M., Odling, P., Ostberg, C., Wahlqvist, M., Wilson, S.K.: A time and frequency synchronization scheme for multiuser OFDM. IEEE J. Sel. Areas Commun. **17**(11), 1900–1914 (1999)

19. Volder, J.E.: The CORDIC trigonometric computing technique. IRE Trans. Electron. Comput. **8**(3), 330–334 (1959)
20. Voros, N.S., Hübner, M., Becker, J., Kühnle, M., Thomaitiv, F., Grasset, A., Brelet, P., Bonnot, P., Campi, F., Schler, E., Sahlbach, H., Whitty, S., Ernst, R., Billich, E., Tischendorf, C., Heinkel, U., Ieromnimon, F., Kritharidis, D., Schneider, A., Knaeblein, J., Putzke-Rming, W.: MORPHEUS: a heterogeneous dynamically reconfigurable platform for designing highly complex embedded systems. ACM Trans. Embed. Comput. Syst. **12**(3), Article 70, 33pp. (2013)
21. Zhao, Q., Sadler, B.M.: A survey of dynamic spectrum access. IEEE Signal Process. Mag. **24**(3), 79–89 (2007)

Chapter 5
Reconfigurable Multiprocessor Systems-on-Chip

Diana Goehringer

5.1 Introduction and Motivation

Software Defined Radio (SDR) [1] is one of the most predestinated applications for re-configurable computing. The reason for this is the diversity of algorithms within the application, which changes from one use case to another. These changes of algorithms, such as adapting a codec, encryption and error correction according to the demands of the current signal to be processed, make reconfigurable hardware the ideal platform for this kind of application. The benefit is obvious: hardware which can be adapted to the demands of an algorithm can be reused and does not need to be provided in parallel on the chip. This re-use leads to a reduction of chip area which simultaneously reduces costs and power consumption. Looking more deeply to the SDR algorithms brings up, that besides the reconfiguration also parallelism can be exploited to improve the efficiency. Here the digital front end and the baseband processing include many opportunities to parallelize the signal processing by using several specific cores realizing a submodule of the general algorithm. The combination of parallelism and reconfiguration of hardware blocks enables now to adapt to changing requirements by substituting a module of the parallel hardware architecture. A simple example is to exchange an encryption block which is used in a secure communication environment. In a further scenario, this block might not be used anymore, but a different codec for a protocol is reconfigured and substitutes the encryption block. This is only one small example of the powerful methods,

D. Goehringer (✉)
Ruhr-University Bochum (RUB), Bochum, Germany
e-mail: diana.goehringer@rub.de

W. Hussain et al. (eds.), *Computing Platforms for Software-Defined Radio*,
DOI 10.1007/978-3-319-49679-5_5

which can be used with reconfigurable Multi-Processor System-on-Chip (MPSoC) architectures. In the next sections, a detailed view to the hardware architecture and the mechanisms of dynamic and partial reconfiguration will be presented. It motivates the usage within the SDR domain and enables to develop many more use cases for future applications.

5.2 Background: Reconfigurable Hardware

There exist a wide variety of different types of reconfigurable hardware architectures. They can be classified into coarse-, medium-, and fine-grained architectures (see [2, 3] and [4]). This classification depends on the bitwidth of the logic blocks as well as on the bitwidth of the routing resources between the logic blocks. As the separation between those classes is not exactly defined, it can occur that a reconfigurable hardware cannot be classified as a specific type of granularity. An example for such a difficult case is an architecture where the bitwidth of the logic blocks and the one of the routing differ from each other. As rule of thumb, reconfigurable architectures with a bitwidth of less than 6 bits are classified as fine-grained, those with a bitwidth between 6 and 8 bits are classified as medium-grained, and all those with a bitwidth of more than 8 bits are classified as coarse-grained.

Coarse-grained architectures are less flexible than fine-grained ones and often have a similar structure as Digital Signal Processors (DSPs), i.e., they consist of an array of DSP-like processing elements with a fixed bitwidth each. Because of this similarity with DSPs their main target application domain is signal processing. A well-known commercial example for this kind of architectures is the PACT-XPP [5]. Their advantage is a simpler programming paradigm compared to fine-grained architectures and also a faster context switch. This means a part or the complete functionality of the PACT-XPP architecture can be adapted at runtime within a few clock cycles.

Fine-grained reconfigurable architectures are often called Field Programmable Gate Arrays (FPGAs). They consist of an array of Look-Up Tables (LUTs), which are connected with each other using a routing network of so called switchboxes. Due to this, the fine-grained architecture has many choices for the user in terms of bitwidth and application realization. Frequently used IP cores such as digital signal processing elements and processors are implemented in some FPGA families as specific on-chip hard-coded IP blocks. The advantage of this realization is an improved performance compared to an implementation of the same functionality using LUTs. FPGAs can be further classified into subgroups based on the type of memory used for containing the configuration information. These subgroups are: SRAM-, flash-, and fuse-based FPGAs. Some of the well-known vendors are Xilinx, Altera, Lattice, and Microsemi. There also exist more and more startup companies such as Tabula and Tier Logic. Xilinx and Altera are the market leaders for SRAM-based FPGAs. SRAM- and flash-based FPGAs can be reconfigured many times, while fuse-/antifuse-based FPGAs can be configured only once. Due to their reconfigurability, SRAM- and flash-based FPGAs are more flexible, because they can be reused for different application scenarios. Due to their fine-grained and flexible structure FPGAs are used for a wide variety of applications, e.g. aerospace/defense, digital signal processing, automotive and medical applications and are therefore more widely deployed than coarse-grained reconfigurable architectures. Their flexibility comes with a longer reconfiguration time compared to coarse-grained reconfigurable architectures. SRAM-based FPGAs from Xilinx and Altera provide currently the highest number of LUTs. Therefore, these FPGAs are frequently used in rapid prototyping systems, such as Synopsis ChipIT [6] and Cadence RP Platform [7], to evaluate the correct functionality of integrated circuits before an ASIC (Application Specific Integrated Circuit) is fabricated. In addition, FPGAs are frequently used in commercial products, for which the production of an ASIC would be too expensive. These are typically products with either medium sales volumes or systems which require a lot of field-updates, due to changing protocols or functionality. Examples are entertainment systems in automotive-, as well as in avionics systems and software defined radios. The reduced power consumption of latest FPGA families and their support for partial reconfiguration make them an attractive platform for a variety of applications.

5.3 Dynamic and Partial Reconfiguration

SRAM-based FPGAs, such as the ones offered by Xilinx, consist of two layers: the first layer (see Fig. 5.1a) contains the reconfigurable hardware (LUTs grouped into Configurable Logic Blocks (CLBs), wiring resources, etc.) and the second layer (see Fig. 5.1b) is the SRAM-based configuration memory containing the configuration frames. A frame is 1-bit wide and is the smallest piece of configuration information that can be written to an FPGA. By writing configuration frames to the SRAM-based configuration layer the hardware structure on the hardware layer, i.e. the function of the LUTs or the wiring, can be changed before the initial start and at runtime. For writing to and reading from the configuration layer, specific configuration and control logic exists. The reading functionality can be used, e.g., to detect errors in the hardware layer, due to event upsets. While in former FPGA families from Xilinx, such as Virtex-II, the frame covered the complete height of the FPGA, this has been changed on the newer architectures starting from Virtex-4. Here a frame only covers a specific height, as shown in the tiled structure of Fig. 5.1b. This height depends on the FPGA family, e.g. Virtex-4 uses 16 CLB-high and Virtex-5 uses 20 CLB-high configuration frames.

Besides configuring the complete FPGA, some SRAM-based FPGAs such as the ones from Xilinx and now also from Altera support a special feature called dynamic and partial reconfiguration. This feature allows exchanging a subset of the FPGA hardware architecture on the first layer, by overwriting only the dedicated configuration frames in the corresponding SRAM configuration memory of the second layer. During this partial configuration, the remaining hardware architecture

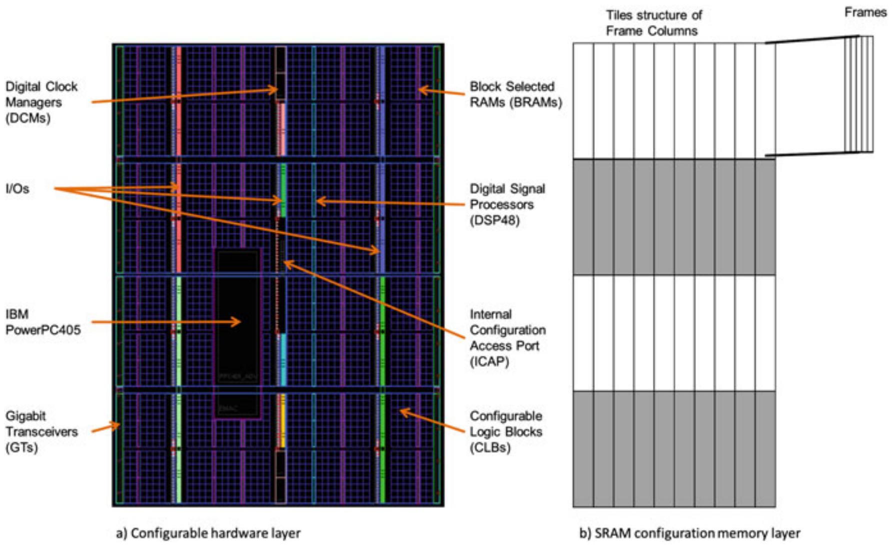

Digital Clock Managers (DCMs)
I/Os
IBM PowerPC405
Gigabit Transceivers (GTs)
Block Selected RAMs (BRAMs)
Digital Signal Processors (DSP48)
Internal Configuration Access Port (ICAP)
Configurable Logic Blocks (CLBs)
a) Configurable hardware layer
Tiles structure of Frame Columns
Frames
b) SRAM configuration memory layer

Fig. 5.1 Layered-Structure of an SRAM-based Xilinx FPGA: (**a**) configurable hardware layer of a Virtex-4FX20 FPGA; (**b**) SRAM configuration memory layer

stays operative and unaffected. Therefore, this process is called dynamic and partial reconfiguration and allows using the FPGA hardware resources in a time-multiplexed fashion for different application tasks. To write the configuration frames into the second layer (SRAM configuration layer) Xilinx provides several external (e.g., PCAP (Processor Configuration Access Port), JTAG) interfaces and an internal interface called ICAP (Internal Configuration Access Port). Therefore, the dynamic and partial reconfiguration can be controlled either from outside of the FPGA, e.g. by using an external processor connected to one of the external configuration interfaces (see, e.g., [8]) or from within the FPGA itself using the ICAP interface (see, e.g., [9]). The internal interface only allows doing partial reconfiguration, i.e. the control processor cannot reconfigure itself using this interface, while the external interface allows both a complete reconfiguration of the FPGA and just loading partial configuration bitstreams. This makes the PCAP, which is available on the Xilinx Zynq FPGAs, such an interesting solution. The main benefits of systems supporting dynamic and partial reconfiguration compared to static FPGA systems are:

- Smaller FPGAs can be used, as the area is reused in a time-multiplexed fashion. This means only the currently needed functionality has to be configured on the FPGA at one point in time.
- Lower power consumption due to the smaller FPGA.
- Reduced costs due to the smaller FPGA.
- Shorter reconfiguration time because the reconfiguration time is proportional to the size of the reconfigured FPGA area.

- No down-time. The non-affected part of the FPGA design stays operative during dynamic and partial reconfiguration.
- Higher flexibility, as the FPGA can be adapted at runtime to the current needs of the application (algorithm, performance, power consumption).
- Higher fault-tolerance, as erroneous parts can be overwritten or exchanged at runtime.

5.3.1 Benefits of Dynamic and Partial Reconfiguration for Software Defined Radio Applications

Due to their configurability, SRAM-based FPGAs allow to receive an in-field update of their hardware architecture, e.g., to support a new communication standard in Software Defined Radio (SDR) applications. During this update typically the complete FPGA is shut down and then a new full bitstream with the updates is configured onto the FPGA. This results in a down-time of the system and a connection loss. However, using dynamic and partial reconfiguration the FPGA hardware can be updated without shutting down the FPGA. This is very important for SDR, as the connection will not be lost if reconfiguration happens within a call or session. Therefore, SDR was one of the first applications used by Xilinx in 2005 (see [10] and [11]) to show the benefits of dynamic and partial reconfiguration in terms of flexibility, reduced system power and costs. Of course, to fully exploit the reconfiguration during a call or session, it has to be assured that the reconfiguration will occur fast enough and sufficient buffer space is available to store the incoming data. An analysis of four SDR scenarios showing the benefits of dynamic and partial reconfiguration and the requirements in terms of reconfiguration time, buffering time and processing performance is given in [12]. The goal of this analysis is to give developers a guideline in designing dynamic and partial reconfigurable systems for SDR. Until now, several research groups all over the world investigate different approaches for exploiting dynamic and partial reconfiguration for software defined radio (see, e.g., [13] and [14]). The availability of open-source libraries, such as GNU radio (www.gnuradio.org), further motivates the development of novel flexible architectures for SDR.

5.4 Reconfigurable Multiprocessor Systems-on-Chip

During the last years, the available hardware resources on Xilinx and Altera FPGAs have increased a lot allowing the implementation of complex systems, such as a Multi-Processor System-on-Chip (MPSoC) consisting of 20 or more cores. Both major FPGA vendors have seen the benefits of combining powerful processor cores with the flexibility of reconfigurable hardware. As a result, both are cooperating with

ARM, famous for low power embedded processors. Out of this, powerful reconfigurable SoCs were generated, such as the Xilinx Zynq (www.xilinx.com) and the Altera SoC (www.altera.com), both combining an FPGA with an ARM dual-core (www.arm.com) processor. Xilinx already announced the follow-up of Zynq called UltraScale MPSoC, which contains even more specialized hardware IP (Intellectual Property) cores combined with the flexible hardware architecture of an FPGA. Due to the high amount of available reconfigurable resources on current FPGAs and the available tool flows for designing dynamically and partially reconfigurable systems, more and more academic and industrial solutions for reconfigurable single- and multiprocessor systems have been introduced. To differentiate these systems from each other with respect to their reconfigurability, well-known taxonomies, such as the one from Flynn [15] which classifies systems based on single/multiple data and instruction streams, are not sufficient anymore. Therefore, Goehringer et al. proposed a new taxonomy (see [16] and [17]), which extends Flynn's taxonomy by differentiating between static (SI/MI) and reconfigurable instruction stream (rSI/rMI) as well as static (SD/MD) and reconfigurable data stream (rSD/rMD) systems. Systems with a reconfigurable instruction stream can reconfigure either their instruction memory or their instruction set, e.g. using reconfigurable hardware accelerators, or both. Reconfigurable data stream systems can either reconfigure their data memory or modify the data paths or both. Figure 5.2 shows the taxonomy. The four classes in the center (SISD, MISD, SIMD, and MIMD) are the ones known from Flynn taxonomy and resemble systems that do not support reconfiguration. Example architectures for each of the classes can be found in [16].

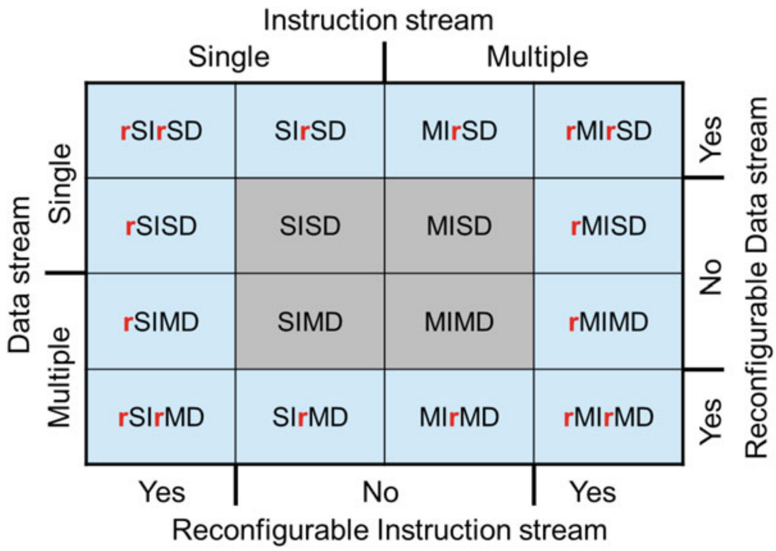


Fig. 5.2 Goehringer's taxonomy [17]

A further extension of this taxonomy for datastream (anti-machine) no instruction stream systems has been provided by Hartenstein (see [18]) resulting in four additional classes (SD, MD, rSD, rMD). Until now, many different reconfigurable processor systems have been proposed, which can be classified based on this taxonomy. However, the focus in the following is on reconfigurable multiprocessor systems, as they provide the highest parallelism (MIMD) combined with reconfiguration, resulting in the three classes: rMIMD, MIrMD, rMIrMD. The class rMIrMD provides hereby the highest flexibility, because it contains multiple instructions and data streams and both are reconfigurable, i.e., they can be adapted at runtime to the requirements of the applications.

5.4.1 rMIMD: Multiprocessors with Reconfigurable Instruction Streams

The first system approaches have been developed for the rMIMD class which stands for MPSoCs with reconfigurable instruction streams, such as reconfigurable instruction memories or reconfigurable accelerators. One of the first systems of this kind was presented by Paulsson et al. [19] and Huebner et al. [20]. This homogeneous MPSoC consisted of several Xilinx MicroBlaze processors and supported the reconfiguration of the instruction memories of the processors in order to load new software applications from the external flash memory at runtime. The system was developed to target automotive applications.

Claus et al. [21] and Bobda et al. [22] also developed FPGA-based multiprocessor systems which can be classified as rMIMD systems. In both systems, the processors are fixed but the accelerators can be reconfigured using dynamic and partial reconfiguration. The system developed by Claus et al. [21] is called Autovision and targets image processing applications for advanced driver assistance systems. The system developed by Bobda et al. [22] is called AMoC (Adaptive Multiprocessor-on-Chip) and targets image processing applications in general.

The multiprocessor system proposed by Dobson et al. [14] consists of five Xilinx Zynqs and targets SDR applications. The ARM dual-core A9 processors are static, but the accelerators implemented on the FPGAs are reconfigurable, thus resulting in rMIMD system.

Tumeo et al. [23] present a homogeneous FPGA-based MPSoC which supports dynamic and partial reconfiguration of the processors. The goal behind this is to increase the reliability by using dynamic and partial reconfiguration to exchange a faulty processor.

Nguyen et al. [24] introduced an FPGA-based MPSoC called PR-HMPSoC. This system consists of several tiles connected over an NoC [25]. A tile can be a processor or an accelerator. Each tile can be exchanged using dynamic and partial reconfiguration.

A different approach is the XiRisc reconfigurable processor [26], which consists of a dual-datapath VLIW RISC core. An additional datapath of this VLIW RISC core is implemented as a closely coupled runtime reconfigurable datapath, called PiCoGA (Pipelined Configurable Gate-Array). By reconfiguring the PiCoGA, the instruction set of the processor and therefore the instruction stream are reconfigurable. The XiRisc was designed and optimized for DSP applications.

An additional example for a VLIW processor with a runtime reconfigurable data path consisting of a coarse-grained reconfigurable array is the ADRES architecture [27]. This reconfigurable array is used to extend the instruction set of the VLIW RISC core and to increase the performance for special instructions, for example related to multimedia applications.

All the above-mentioned architectures are just a subset of the MPSoCs falling into this category and more and more are currently being developed due to the provided flexibility by adapting the instruction stream of these architectures at runtime.

5.4.2 MIrMD: Multiprocessors with Reconfigurable Data Streams

MPSoCs belonging to the MIrMD class have static instruction streams and reconfigurable data streams, i.e. they support reconfigurable data memories and/or have a reconfigurable communication infrastructure.

Sander et al. [28] were the first to demonstrate the transfer of data from one set of on-chip block memories (BRAMs) to another set of BRAMs using partial reconfiguration over the ICAP interface. Shelbourne et al. [29] proposed Metawire, which is an NoC architecture using the same principle as proposed by Sander et al. [28]. This means, the NoC routers are not directly connected with each other on the first layer of the FPGA. Instead the ICAP is used to read out the respective data stored within a BRAM of the source router and transfer this data to the BRAM of the destination router. The functionality was proven with an AM radio application.

Several reconfigurable communication infrastructures have been proposed. Pionteck et al. [30] presented CoNoChi, which is a packet-based Network-on-Chip (NoC) with a 2D mesh topology. It supports dynamic and partial reconfiguration of the processing elements and the routers, i.e., the size of the NoC can be scaled at runtime to fit to the current application requirements.

CSRA-NoC [31] is a circuit-switching NoC with a 2D mesh topology. It also supports dynamic reconfiguration of both the routers and the processing elements. The advantage of this NoC is the small size of the routers compared to a packet-based NoC such as CoNoChi. However, the drawback is the comparable long setup time to establish a communication channel between a sender and its destination, as this is done using partial reconfiguration.

The Star-Wheels NoC [32] combines packet- and circuit-switching. It has a special topology consisting of several subnets using a wheel topology that are connected over a central star topology. This NoC supports partial reconfiguration of routers and the processing elements.

Recently, the RAR-NoC [33] using wormhole routing and a 2D mesh topology has been proposed. As the previous NoCs it supports partial reconfiguration of the routers and the processing elements. In addition the routing algorithm can be adapted at runtime to avoid hotspots.

Also for this category more examples exist than can be mentioned here. Especially, in the area of reconfigurable NoCs, a lot of research is being done.

5.4.3 rMIrMD: Multiprocessors with Reconfigurable Instruction and Reconfigurable Data Streams

The rMIrMD class combines both the rMIMD and the MIrMD class. Basically they consist of MPSoCs from the rMIMD class using either the ICAP for transferring data from one PE to another or using a reconfigurable communication in-frastructure, such as the NoCs mentioned in the MIrMD class. The benefits of this architecture are the high flexibility, as both the data transfer and the processing elements can be reconfigured. However, designing and programming such a flexible architecture is very complex. Therefore, so far only a few systems exist that fall into this category. Two of them will be introduced more in detail in the next subsections.

5.4.3.1 RAMPSoC

The first MPSoC fulfilling these features has been RAMPSoC [34], which supports dynamic and partial reconfiguration of the processing elements (processors and hardware accelerators) and the communication infrastructure. For the communication infrastructure it uses the before mentioned Star-Wheels NoC [32]. Figure 5.3 shows an example of the RAMPSoC architecture at one point in time.

RAMPSoC has one master processor called RAMPSoCVM (RAMPSoC Virtual Machine) [35] and several slave processing elements. RAMPSoCVM is responsible for scheduling and allocation of the application task to the processing elements. It is also responsible for handling all reconfiguration decisions. Therefore, it is connected to the ICAP using an own developed IP core called FSL-ICAP [9]. On this processor an embedded Linux operating system is running to abstract from the complexity of the MPSoC and to provide a well-known interface to the user. The user then can start his/her applications within the Linux interface and the RAMPSoCVM will manage then all the required scheduling, allocation, and reconfiguration decisions for this application. The slave processing elements can be either processors with/without hardware accelerators or a Finite State Machine

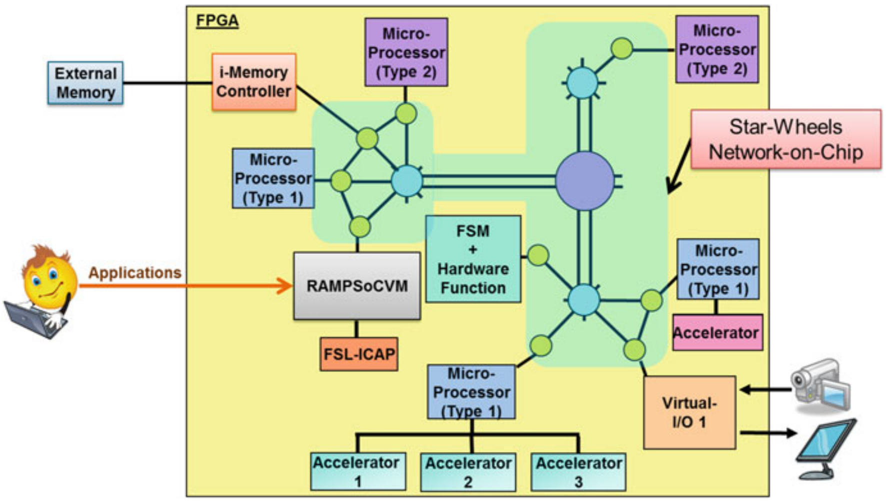


Fig. 5.3 RAMPSoC architecture at one point in time [17]

(FSM) with a hardware accelerator. The slave processors only run a small boot program to receive incoming application tasks over the Network-on-Chip and store them into their memory before execution. RAMPSoC uses a distributed memory approach. Due to the limited on-chip memory of the FPGA, RAMPSoCVM uses external memory for the Linux OS, while the slave processors use on-chip memory and therefore do not have an OS but rather use an area optimized boot program. Figure 5.3 also shows the Star-Wheels Network-on-Chip. As can be seen, only the currently required number of routers are present on the FPGA. In addition, the routers have the functionality to detect, if the connected processing element has been exchanged or removed. Two additional components should be mentioned here as well. The first one is the Virtual-IO, which allows connecting sensors (e.g., a camera) and actors (e.g., a screen) to the Star-Wheels NoC. The second one is the intelligent memory controller called i-memory controller [36], which manages the access of the varying number of processing elements to the external memory. An additional feature of this architecture is that it can be easily distributed over several FPGA boards, as it was shown in [37].

Designing and programming such a system is not easy due to the huge design space. The number and type of the required processing elements and their connections have to be specified. It has to be selected which of these elements need to be reconfigurable. In addition, the applications have to be partitioned and optimized to achieve a good performance on the selected processing elements. To support the user, a semi-automatic design methodology, see Fig. 5.4, has been introduced in [38] and extended in [17]. It consists of three phases and uses a combination of commercial and own developed tools. The first phase is responsible for suggesting an application partitioning, the number of required processors and their placement within the Star-Wheels NoC. Therefore, this first phase requires the application

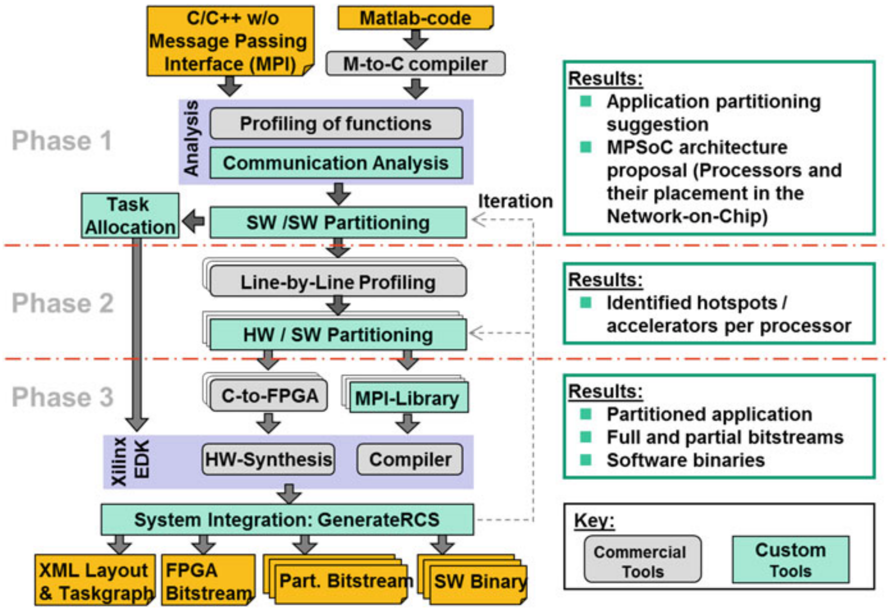


Fig. 5.4 RAMPSoC design methodology [17]

described either as C/C++ code with or without the usage of MPI (Message Passing Interface) or alternatively embedded Matlab code which has been translated to C code using the M-to-C compiler from Matlab [39]. The functions of the resulting C/C++ code are then analyzed with a commercial profiler, such as GNU gprof [40]. In addition, an own developed tool is used to analyze the required communication between the functions. With this analysis, all required information needed (relative computation time of the functions, communication effort between the functions) for the Software (SW)/SW Partitioning tool is available. The SW/SW Partitioning tool uses the hierarchical clustering algorithm to partition the application in such a way that the computation load is well balanced and that the required communication between the processors is reduced. Due to the heterogeneous architecture of the Star-Wheels NoC a further optimization is done using the Task Allocation tool. This tool will analyze how close the processing elements should be placed to each other to reduce the communication delay over the Star-Wheels NoC. For this step a heuristic is used, which combines a greedy algorithm with an optimization heuristic, which is based on the Kernighan-Lin [41] and the Fiduccia-Matheyses [42] algorithms.

The second phase uses then again a commercial profiler to do a line-by-line profiling for the function sets assigned to each individual processor. Based on this line-by-line profiling, a hardware/software partitioning is done to move compute-intensive functionality into hardware accelerators.

The third phase is responsible for generating all the required hardware and software files needed for the RAMPSoCVM processor to manage the system.

C-to-FPGA tools, such as VivadoHLS, are used to generate the code for the hardware accelerators identified in Phase 2. An own developed MPI-library is linked to the software functions to abstract from the low-level commands of the Star-Wheels NoC. The Xilinx Embedded Development Kit (EDK) tools are then used to design and synthesize the MPSoC and to compile the software executables. To generate the full and partial bitstreams an own developed tool called GenerateRCS is used, which calls the Xilinx tools for generating the full and partial bitstreams. In addition, the design methodology generates an XML description, which includes the description of the task graphs and their ideal relevant placement to each other within the Star-Wheels NoC. The resulting files of the design methodologies are then given to the RAMPSoCVM at system startup.

RAMPSoC has been evaluated with a variety of different applications such as image processing, 3D Ultrasound Computer Tomography and Bioinformatics.

5.4.3.2 RAR-MPSoC

A second architecture of this kind called RAR-MPSoC is built around the reconfigurable RAR-NoC [33]. Besides the reconfiguration of the NoC, also here the processing elements (processors and hardware accelerators) can be reconfigured at runtime. Figure 5.5a shows the system architecture and Fig. 5.5b the corresponding FPGA layout with the reconfigurable regions for the PEs and the routers.

As can be seen, this system uses the Xilinx Zynq architecture. Also here a distributed memory architecture with one master and multiple slave processors is used. The ARM functions as the master processor and is responsible for

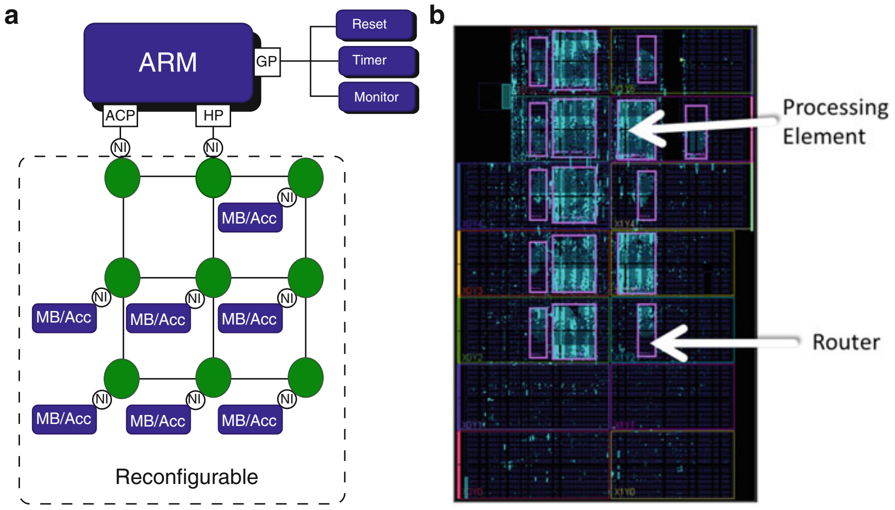


Fig. 5.5 RAR-MPSoC [33]: (a) system architecture, (b) FPGA layout

reconfiguring the other processors and routers. For the reconfiguration, the PCAP interface is used. Due to the higher clock rate and higher performance of the ARM processor compared to the reconfigurable fabric, two high performance ports, Accelerator Coherence Port (ACP) and High Performance Port (HP), are used to connect the ARM to the RAR-NoC.

The system has been evaluated so far with image processing applications and an automotive use case. In the future software defined radio will be implemented to evaluate the flexibility, scalability and high energy efficiency of RAR-MPSoC for this kind of application.

5.5 Conclusion and Outlook

Reconfigurable MPSoCs combine the benefits of traditional MPSoCs and FPGAs. Due to dynamic and partial reconfiguration, these systems can adapt at runtime both the software and the hardware according to the application requirements. This way high energy efficiency as well as a high flexibility is achieved. In addition, these systems are highly scalable, both on-chip and by connecting several chips to a cluster. This makes them ideal candidates for hosting software defined radio applications. In addition, open-source libraries such as GNU radio will even increase the interest of the reconfigurable computing community to develop efficient systems for this kind of application.

References

1. Reed, J.: Software Radio - A Modern Approach to Radio Engineering. Prentice-Hall, Englewood, Cliffs, NJ (2002)
2. Compton, K., Hauck, S.: Reconfigurable computing: a survey of systems and software. ACM Comput. Surv. **34**(2), 171–210 (2002)
3. Radunovic, B.: An overview of advances in reconfigurable computing systems. In: Proceedings of the 32nd Hawaii International Conference on System Science (1999)
4. Hauck, S., DeHon, A.: Reconfigurable Computing: The Theory and Practice of FPGA-Based Computation. Morgan Kaufmann Series in Systems on Silicon. Morgan Kaufmann Publishers Inc., San Francisco, CA (2007)
5. Becker, J.: Configurable systems-on-chip: commercial and academic approaches. In: Proceedings of the IEEE 9th International Conference on Electronics, Circuits and Systems (ICECS 2002), Dubrovnik (Kroatien), vol. 2, pp. 809–812, September 2002
6. Synopsys Inc.: CHIPit Platinum Edition and HAPS-600 Series Hardware Reference Manual (v.1.9) (2012). Available at: www.synopsys.com
7. Cadence: Cadence Rapid Prototyping Platform. Datasheet (2011). Available at: www.synopsys.com
8. Angermeier, J., Goehringer, D., Majer, M., Teich, T., Fekete, S., van der Veen, J.: The Erlangen slot machine-a platform for interdisciplinary research in reconfigurable computing. Inf. Technol. J. Oldenbourg Wissenschaftsverlag **49**(3), 143–148 (2007)

9. Huebner, M., Goehringer, D., Noguera, J., Becker, J.: Fast dynamic and partial reconfiguration Data Path with low Hardware overhead on Xilinx FPGAs. In: Proceedings of the IEEE International Symposium on Parallel and Distributed Processing, Workshops and PhD Forum (IPDPSW), April 2010
10. Kao, C.: Benefits of partial reconfiguration. Xcell J. **Fourth Quarter**, 65–67 (2005)
11. Uhm, M.: Software-defined radio: the new architectural paradigm. Xilinx DSP Magazin, 40–42 (2005)
12. Becker, T., Luk, W., Cheung, P.Y.K.: Parametric design for reconfigurable software-defined radio. In: Proceedings of the International Symposium on Applied Reconfigurable Computing (ARC), pp. 15–26, March 2009
13. Delahaye, J.P., Palicot, J., Moy, C., Leray, P.: Partial reconfiguration of FPGAs for dynamical reconfiguration of a software radio platform. In: Proceedings of the 16th IST Mobile and Wireless Communications Summit, pp. 1–5 , July 2007
14. Dobson, C., Rooks, K., Athanas, P.: A power-efficient FPGA-based self-adaptive soft-ware defined radio. In: Proceedings of the 24th International Workshop on Power and Timing Modeling, Optimization and Simulation (PATMOS), pp. 1–8 (2014)
15. Flynn, M.J.: Very high-speed computing systems. Proc. IEEE **54**(12), 1901–1909 (1966)
16. Goehringer, D., Huebner, M., Perschke, T., Becker, J.: A taxonomy of reconfigurable single/multi-processor systems-on-chip. Hindawi Int. J. Reconfig. Comput., **2009**(395018), pp. 1–11 (2009)
17. Goehringer, D.: Flexible Design and Dynamic Utilization of Adaptive Scalable Multi-Core Systems. Dissertation, Karlsruhe Institute of Technology, Verlag Dr. Hut Muenchen (2011)
18. Hartenstein, R.: Stonewalled progress of computing efficiency: why we must reinvent computing. In: Keynote, 25th Symposium on Integrated Circuits and Systems Design (SBCCI) (2012)
19. Paulsson, K., Huebner, M., Zou, H., Becker, J.: Realization of real-time control flow oriented automotive applications on a soft-core multiprocessor system based on Xilinx Virtex-II FPGAs. In: Proceedings of International Workshop on Applied Reconfigurable Computing (ARC), pp. 103–110 (2005)
20. Huebner, M., Paulsson, K., Becker, J.: Parallel and flexible multiprocessor system-on-chip for adaptive automotive applications based on Xilinx MicroBlaze Soft-Cores. In: Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS) (2005)
21. Claus, C., Stechele, W., Herkersdorf, A.: Autovision - a run-time reconfigurable MPSoC architecture for future driver assistance systems? Inf. Technol. J. **49**(3), 181–187 (2007)
22. Bobda, C., Haller, T., Muehlbauer, F., Rech, D., Jung, S.: Design of adaptive multiprocessor on chip systems. In: Proceedings of the 20th Annual Conference on Integrated Circuits and Systems Design (SBCCI), pp. 177–183 (2007)
23. Tumeo, A., Regazzoni, F., Palermo, G., Ferrandi, F., Sciuto, D.: A reconfigurable multiprocessor architecture for a reliable face recognition implementation. In: Proceedings of the Conference on Design, Automation and Test in Europe (DATE) (2010)
24. Nguyen, T.D.A., Kumar, A.: PR-HMPSoC: a versatile partially reconfigurable heterogeneous multiprocessor system-on-chip for dynamic FPGA-based embedded systems. In: Proceedings of the International Conference on Field Programmable Logic and Applications (FPL) (2014)
25. Yang, Z.J., Kumar, A., Ha, Y.: An area-efficient dynamically reconfigurable spatial division multiplexing network-on-chip with static throughput guarantee. In: Proceedings of the International Conference in Field-Programmable Technology (FPT), pp. 389–392 (2010)
26. Cappelli, A., Lodi, A., Mucci, C., Toma, M., Campi, F.: A dataflow control unit for C-to-configurable pipelines compilation flow. In: Proceedings of IEEE 12th International Symposium on Field-Programmable Customs Computing Machines (FCCM), Napa Valley (CA), pp. 332–333 (2004)
27. Berekovic, M., Kanstein, A., Mei, B.: Mapping MPEG video decoders on the ADRES reconfigurable array processor for next generation multi-mode mobile terminals. In: Proceedings of Global Signal Processing Conferences and Expos for the Industry: TV to Mobile (GSPX) (2006)

28. Sander, O., Braun, L., Becker, J.: An exploitation of data reallocation by performing internal FPGA self-reconfiguration mechanisms. In: Proceedings of Reconfigurable Computing: Architectures, Tools and Applications (ARC), pp. 312–317 (2008)
29. Shelburne, M., Patterson, C., Athanas, P., Jones, M., Martin, B., Fong, R.: Metawire: using FPGA configuration circuitry to emulate a network-on-chip. In: Proceedings of Field Programmable Logic and Applications (FPL), pp. 257–262 (2008)
30. Pionteck, T., Albrecht, C., Koch, R., Maehle, E., Huebner, M., Becker, J.: Communication architectures for dynamically reconfigurable FPGA designs. In: Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS) (2007)
31. Braun, L., Huebner, M., Becker, J., Perschke, T., Schatz, V., Bach, S.: Circuit-switched runtime adaptive network-on-chip for image processing applications. In: Proceedings of the International Conference on Field Programmable Logic and Application (FPL), pp. 688–691 (2007)
32. Goehringer, D., Oey, O., Huebner, M., Becker, J.: Heterogeneous and runtime parameterizable star-wheels network-on-chip. In: Proceedings of 11th International Conference on Embedded Computer Systems: Architectures, Modeling and Simulation (SAMOS X) (2011)
33. Rettkowski, J., Goehringer, D.: RAR-NoC: a reconfigurable and adaptive routable network-on-chip for FPGA-based multiprocessor systems. In: Proceedings of the International Conference on Reconfigurable Computing and FPGAs (ReConFig) (2014)
34. Goehringer, D., Huebner, M., Schatz, V., Becker, J.: Runtime adaptive multi-processor system-on-chip: RAMPSoC. In: Proceedings of the IEEE International Symposium Parallel and Distributed Processing (IPDPS), pp. 1–7 (2008)
35. Goehringer, D., Werner, S., Huebner, M., Becker, J.: RAMPSoCVM: runtime support and hardware virtualization for a runtime adaptive MPSoC. In: Proceedings of the International Conference on Field Programmable Logic and Applications (FPL) (2011)
36. Goehringer, D., Meder, L., Werner, S., Oey, O., Becker, J., Huebner, M.: Adaptive multi-client network-on-chip memory core: hardware architecture, software abstraction layer and application exploration. Hindawi Int. J. Reconfig. Comput., **2012**(298561) pp. 1–14 (2012)
37. Oey, O.,Werner, S., Goehringer, D., Stuckert, A., Becker, J., Huebner, M.: Virtualization of heterogeneous and adaptive multi-core /multi-board systems. In: Proceedings of the Conference on Design and Architectures for Signal and Image Processing (DASIP) (2012)
38. Goehringer, D., Huebner, M., Benz, M., Becker, J.: A Design methodology for application partitioning and architecture development of reconfigurable multiprocessor systems-on-chip. In: Proceedings of the 18th International IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM) (2010)
39. MathWorks: Real-Time Workshop Embedded Coder. Available at: www.mathworks.de
40. GCC: the GNU Compiler Collection. Available at: gcc.gnu.org
41. Kernighan, B.W., Lin, S.: An efficient heuristic procedure for partitioning graphs. Bell Syst. Tech. J. **49**(12), 291–307 (1970)
42. Fiduccia, C.M., Mattheyses, R.M.: A linear-time heuristic for improving network partitions. In: Proceedings of the 19th Design Automation conference (DAC), pp. 175–181 (1982)

Chapter 6
Ninesilica: A Homogeneous MPSoC Approach for SDR Platforms

Roberto Airoldi, Fabio Garzia, Tapani Ahonen, and Jari Nurmi

6.1 Introduction

Embedded run-time Reconfigurable/Reprogrammable hardware is today largely recognized as a necessity for the design of future generations of System-on-Chips (SoCs) in order to face the continuous evolution of algorithms and protocols. In particular, Multi-Processor Systems-on-Chip (MPSoC) are widely considered an effective solution to deal with the increasing complexity of algorithms, their rapid evolution due to standard updates and increase in performance requirements [14]. MPSoCs offer high flexibility and high computational power. Moreover, they offer programmers a known and standardized programming environment that does not break the familiar Von-Neumann paradigm. Inter-core networking issues and parallel thread dependencies are handled using standards and APIs that are closely related to computer networks and thus well established both in industry and literature. On the other hand, the flexibility of MPSoCs comes at the price of more complex design parameters: MPSoCs are intrinsically more redundant than application-specific or application-oriented designs and very often also of most existing reconfigurable hardware solutions. General purpose processor cores feature GOPS/mm^2 and GOPS/mW signatures around 1 or 2 orders of magnitude larger than application specific circuitry. As an example, Table 6.1 summarizes GOPS, GOPS/mm^2, and GOPS/W for state-of-the-art signal processing platforms.

While the area issue is somehow not critical in terms of silicon real estate due to the resources made available by technology scaling, distributing a design over a large silicon area will make it more subject to Process, Voltage, and Temperature

R. Airoldi • F. Garzia • T. Ahonen • J. Nurmi (✉)
Department of Electronics and Communications Engineering, Tampere University of Technology, Tampere, Finland
e-mail: Roberto.Airoldi@nsn.com; Fabio.Garzia@iis.fraunhofer.de; ukaata@gmail.com; Jari.Nurmi@tut.fi

W. Hussain et al. (eds.), *Computing Platforms for Software-Defined Radio*,
DOI 10.1007/978-3-319-49679-5_6

Table 6.1 GOPS performance of State-of-the-art signal processing platforms

Device	Techno	GOPS	GOPS/mm^2	GOPS/W
Morpheus [12]	CMOS 90	60	0.6	20
ARM 9	CMOS 90	0.35	0.15	1
ST ASIC	CMOS 90	NA	10	1000
TU iVisual [6]	CMOS 180	77	1.16	205
KAIST vect. Proc. [9]	CMOS 130	125	3.4	214
Philips Xetal II	CMOS 90	107	1.44	170
Cell [10]	CMOS 90	200	0.9	0.5
UCB Bee2	FPGA	NA	NA	2.2
Virtex-4 SX55	FPGA	50	NA	3.5
Xilinx Virtex-II 8000	FPGA	450	NA	3.5

(PVT) variability issues leading to lower design yields. Even more critical is the increase in power consumption: power is not only a critical design parameter in itself. The presence of strong current densities in the die might lead to significant design issues such as thermal dissipation, static and dynamic IR drop, electro-migration and electro-magnetic interferences that in turn may cause hard and soft errors that can jeopardize the functionality of the whole design.

When considering SDR platforms, the majority of the proposed approach is based on heterogeneous solutions [13, 15]. Indeed heterogeneous solutions offer today the best compromise in terms of performance, area, and power consumption. On the other hand, such architectures introduce complications in the application development. In fact, a heterogeneous set of components requires a heterogeneous set of tools, based on different programming languages and interfaces. Therefore, from a programmability and scalability point of views, homogeneous architecture might be more appealing. Furthermore problems related to task scheduling, resources contention, and hardware scalability have a higher impact when trying to realize heterogeneous platform composed of an increasing number of nodes. Therefore, it is interesting from a research point of view to explore homogeneous MPSoC solutions as a feasible way to mitigate such issues.

In this book chapter, the authors propose a homogeneous MPSoC implementing kernels related to wireless communications systems. Focus of the research work is the analysis of implementation efficiency, scalability of the algorithms with the number of processing nodes and portability of the solutions. Key point of the research will also be the impact that power saving techniques have on the system performance when applied to MPSoC.

The rest of the paper is organized as follows. Section 6.2 introduces Ninesilica architecture and discusses its main features. Section 6.3 briefly describes the study cases analyzed and their implementation on Ninesilica. Section 6.4 summarizes the obtained results in terms of achieved speed-ups, absolute performance, energy consumption, and parallelization efficiency. Finally, Sect. 6.5 discusses conclusions and future work directions.

6.2 Ninesilica Architecture

Ninesilica architecture provides a generic platform composed of an arbitrary number of Processing Nodes (PNs) arranged in a mesh topology. The interconnection between PNs is supported by a hierarchical Network-on-Chip [1]. The template allows the partition of the platform in clusters, which are composed of an arbitrary number of nodes, obtaining a hierarchical type of architecture. Ninesilica is a homogeneous cluster composed of nine nodes arranged in a 3×3 mesh. The cluster can be utilized as standalone block or as a building block to realize clustered many-core architectures.

A schematic view of one Ninesilica cluster and its internal node structure are given in Fig. 6.1. Each node is composed of 16 KBytes data memory (DMEM), 16 KBytes of instruction memory (IMEM) besides COFFEE RISC core and Network Interface (NI). Hardware Description Language (HDL) of COFFEE RISC core is freely available from the COFFEE RISC project website [7]. The PE utilized in for this research work is not the main focal point. In fact, the authors are interested in the system properties that are independent from the type of PE utilized; the obtained results could be applied to any similar architecture based on different type of PEs, affecting only the absolute performance, but for example, scalability, parallelization efficiency, and relative power consumption would not be influenced by the PE choice.

6.2.1 Network-on-Chip

The NoC is the backbone of the template, allowing the composition of different architectures based on a different number of cluster and/or nodes. The hierarchical NoC provides two levels of communication inside a cluster. The lower level takes care of the inter-node communication while the higher level takes care of the

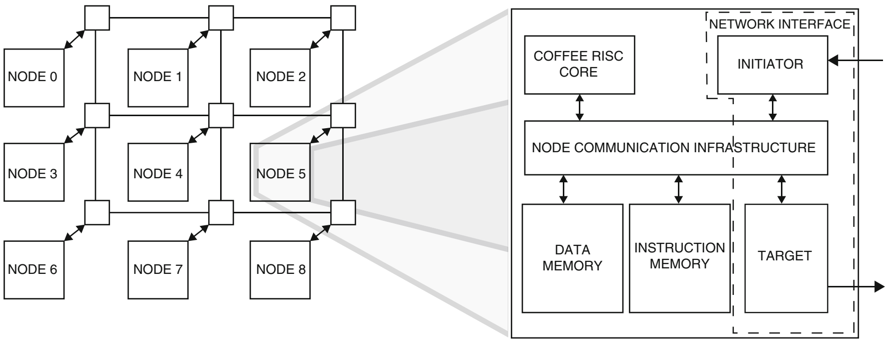


Fig. 6.1 Schematic view of Ninesilica cluster and its node structure

communication at cluster level. The NI of each node acts as a bridge between the two communication levels. The NoC supports single-cast, multi-cast (symmetric and non), and broadcast communication. The multicast and broadcasting is accomplished via the repetition of the packet from a node to another. It was proved that the utilization of the broadcasting technique could improve the system performance of a 30 % factor [2].

In clustered architectures, a third level of hierarchy connects the different clusters together. The interconnection routes can be adapted at run-time to different traffic conditions. Each NI contains a look-up-table with 16 possible routing paths for the communication over the NoC. Routing paths can be redefined at run-time according to the application requirements. The clustering is a feasible way to efficiently implement many-core systems, while keeping the communication overhead limited. This is true in particular when most of the data exchanges occur between nodes belonging to the same cluster, while only few synchronization messages and data exchanges will take place between clusters.

6.2.2 Power Management

The template provides power management support based on clock gating controlled via software. Each node in the architecture has three power domains: one for the memories, one for the network interface, and one for the PE. Therefore it is possible to define different sleep states for each node. Light sleep mode is utilized when a PE has no processing to be done and it is waiting for a new chunk of data. In such a scenario DMEM and NI need to be active to allow the download of the data. However, the PE can stay in a sleep mode, since there is no processing to be performed. Sleep mode is utilized when processing in a node is completed. In this case there are two possibilities. Either the node is utilized as repeater for multicast communications or the node is completely idle. In the first case, both PE and memories can be kept in sleep mode, however the NI needs to be clocked to allow the re-routing of communications. In the second case, the node is completely switched off, allowing a higher energy saving.

6.2.3 I/O Management

An SDR application is characterized by data-streaming. In Ninesilica cluster, the only node connected to the input ports is the central node. The central node acts as cluster controller and it takes care of data and tasks scheduling, power management, and synchronization. The central position in the cluster offers the most uniform latency in the data distribution and a balanced workload on the NoC. When considering clustered many-core architecture, each central node of a cluster is equipped with I/O, allowing each cluster to work independently on the received data stream. This way it is possible to reduce the communication between clusters to control-flow data, keeping a low communication overhead.

Table 6.2 Stratix IV FPGA synthesis results of Ninesilica and 4 Ninesilica clusters architecture

Entity	ALUT	Logic regs	DSP blocks	% Utilization
COFFEE RISC	7054	4941	16	1.6
NI	381	411	0	0.1
Central node	7617	5308	16	1.7
Processing node	7360	5167	144	1.7
Ninesilica cluster	71,679	50,897	0	17
NoC switch (average)	2155	1489	0	0.5
Whole NoC	99,026	69,223	0	23.4
4-cluster architecture	352,970	247,099	576	83.5

6.2.4 *Hardware Implementation*

Ninesilica was fully implemented on an Altera Stratix-IV FPGA device (EP4GX530). In particular, we implemented a standalone Ninesilica and a 4-cluster architecture. The synthesis was carried out using Quartus II version 9.1 SP1 design flow. Table 6.2 collects synthesis results. The four Ninesilica clusters utilize 83.5 % of the available resources. Moreover, analyzing the breakdown of the system, it is possible to notice that at node level, most of the resources are consumed by the COFFEE RISC while the network interface occupies less than 0.1 %. Operating frequency, setting a slow mode synthesis is 115.0 MHz. The maximum working frequency in the fast FPGA mode is 180.0 MHz.

6.3 Cases Studies

To analyze the performance of Ninesilica and the 4-cluster architecture performing SDR applications, we have considered two significant communication techniques of state-of-the-art radio standards, and their respective kernels. The first one is WCDMA [8]. WCDMA is widely used for state-of-the-art cellular communications. Therefore its implementation is significant to the scope of this research work. The second communication technique analyzed is OFDM [13]. In the past few years OFDM has been gaining a lot of interest as a feasible way to obtain high data rates. OFDM is widely utilized in most of today's broadband communications standard, such as: IEEE 802.11a/g/n (WiFi), IEEE 802.16 (WiMax), and 3GPP-LTE (4G). OFDM has also been considered as a physical layer communication technique for highly innovative communications systems, as for example cognitive radios [11]. WCDMA modulation, demodulation and timing synchronization are based on the computation of correlations (e.g., in rake receivers) and for OFDM the most demanding kernel computationally is the modulation/demodulation block, which is based on the computation of IFFT/FFT. According to the communication standard, the FFT size can span from 64-point (i.e., IEEE 802.11a/g) up to 2048-point (i.e., 3GPP-LTE).

6.3.1 WCDMA

One of the most significant kernels in WCDMA is the computation of correlations. Correlations are utilized in the demodulation (rake receivers) and in the time-synchronization steps. The time-synchronization steps involve different functionalities from the multi-path detection to the cell synchronization. In this section, we will briefly analyze how a correlation is mapped on the Ninesilica cluster and see a few detail of the cell search algorithm implementation. The computation of correlations for the timing synchronization can be formulated as

$$r_n = \frac{1}{256} \sum_{i=0}^{255} R_{n+i} \times C_i \tag{6.1}$$

where r_n is the nth correlation point; R_{n+i} is the $(n + i)$th input of the data-stream; C_i is the ith sample of the coefficients. The parallelization can be done in two different ways. In the first case, the accumulation can be split into N independent accumulations, which are finally added together. In such a case, considering N computational nodes, the formula can be rewritten as

$$r_n = \frac{1}{256} \sum_{j=0}^{N-1} \sum_{i=0}^{255/N} R_{n+i\times j} \times C_{i\times j} \tag{6.2}$$

where each internal accumulation is carried out by a single computational node. The second option is to divide the computation on different set of nodes where nodes perform the computation of following correlation points of the sliding window. In this case, the communication overhead can be kept low since the data-stream is broadcasted to the nodes and they operate only on their assigned window.

The cell search algorithm is composed of three following steps [4]. In the first step in slot synchronization, the receiver gets aligned with the transmitted signal. This step is accomplished by correlating the received signal with a known sequence, which is transmitted at the beginning of each slot. The detection of high peaks in the correlation output identifies the boundary of a slot. In the second step of frame synchronization, the slots are actually classified and recognized. This allows the selection of the possible scrambling code utilized by the transmitting base-station. In this phase the incoming data-stream is correlated to 16 possible sequences to identify the frame structure and hence the code-group of the transmitting cell. Finally in the last step of scrambling code identification, the possible scrambling codes (which are 8) are tested against the received stream. The scrambling code which offers the highest correlation pattern is then identified as the one transmitted

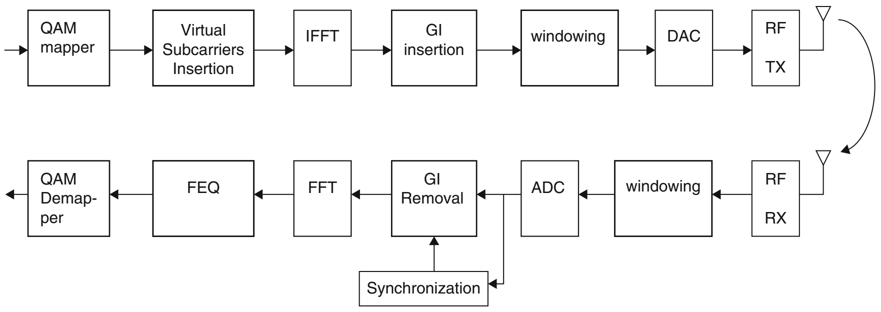


Fig. 6.2 Block diagram of a generic OFDM transceiver

by the base-station. Hence knowing the scrambling code utilized and the timing synchronization from the previous two steps, it is possible to start the demodulation of the incoming signal.

6.3.2 OFDM

A block level view of transmitter and receiver scheme of a generic OFDM is given in Fig. 6.2. If we consider the computational complexity, modulator/demodulator, which are based on the computation of IFFT/FFT, are the most critical blocks. Therefore how the FFTs are implemented is quite important. Moreover, in a flexible radio context, the implementation of the FFT cannot be fixed in size and computation time. In fact, different standards utilize different numbers of subcarriers (FFT size) and require different symbol processing time. Hence, the FFTs computation cannot be anymore done by a monolithic hardwired block, but it requires a higher level of flexibility, which can allow changing the computational block structure according to the OFDM system currently running on the platform.

In such a scenario, the implementation of FFTs on programmable parallel architectures gains a lot of interest as a feasible way to obtain high flexibility as well as high performance. However, when moving from a single DSP core to multi-core platforms there are new issues that arise. Communication overhead, which is a natural bottleneck of parallel architecture, might affect algorithm's performance lightly or heavily according to the algorithm partitioning adopted. In fact, slightly different parallelization strategies might lead into different results both in terms of performance and parallelization efficiency. Another important choice in the implementation of the FFT block is the algorithm utilized. In literature, it is possible to find many variants of algorithms leading to a reduced computation complexity. However, the parallelization scheme introduced might dilute the benefit introduced by the algorithm reduction. The parallelization technique and the algorithm partitioning on the multi-core platform can significantly affect the final algorithm performance.

6.4 Analysis of Results

The kernels introduced in the previous section have been implemented and profiled via cycle accurate simulation on:

1. a single core architecture (utilized as reference architecture) based on the same PE of Ninesilica;
2. Ninesilica cluster;
3. and the 4-cluster architecture.

The experimental work shows the achieved speed-ups, parallelization-efficiency, and relative energy saving when clock gating techniques are applied.

For the WCDMA technique, we implemented a single correlation point and the cell search algorithm. For the OFDM system, we implemented a receiver for IEEE 802.11a/g. Moreover, we analyzed in detail the FFT kernel and its parallel implementation, implementing different algorithms and trying to find out which algorithms give the best performance and in which situations.

6.4.1 WCDMA Cell Search

Cell search for WCDMA can be divided into three steps: slot synchronization, frame synchronization, and scrambling code identification. Each step of the cell search requires the computation of correlations on a sliding window with a different set of coefficients according to the algorithm. Table 6.3 collects the profiling for the computation of a single correlation point as well as the each single step of the cell search algorithm, on a single core architecture and Ninesilica. Table 6.3 also reports the relative energy saving that can be achieved when applying clock gating to the system. It is worth mentioning that the introduction of clock gating does not affect the profiling performance, even if software controlled, but it only avoids the energy consumption in situations where cores are on hold.

Table 6.3 Performance of Ninesilica cluster computing the WCDMA cell search algorithm

	# clock cycles	# clock cycles		Energy savings
Task	Single COFFEE	NineSilica	Speed-up	Via clock gating
Correlation point	12,381	2546	5×	–
Slot synch. (fixed part)	52,890,764	7,147,387	7.5×	18 %
Frame synch.	3,750,593	471,458	8×	46 %
Scrambling code ID.	149,973	56,203	2.7×	40 %
Cell search	57,410,380	7,802,348	7.3×	33 %

Table 6.4 Performance comparison between Ninesilica and 4-cluster architecture

Task	# clock cycles 4-cluster architecture	# clock cycles NineSilica	Speed-up
Correlation point	688	2546	3.7×
Slot synch. (fixed part)	1,906,649	7,147,387	3.7×
Frame synch.	276,428	471,458	1.7×
Scrambling code ID.	56,203	56,203	1×
Cell search	2,366,580	7,802,348	3.3×

Scaling the system from a single Ninesilica cluster to a 4-cluster architecture requires the redefinition of the implemented step. After a detailed study of the application it was possible to notice that for some steps of the algorithm a further parallelization would have led into a lower speed-up and performance due to a higher impact of the communication overhead. For such reason, the peak detection of the slot synchronization and the scrambling code identification steps have been kept on a single cluster while multi-path detection and frame synchronization have been further distributed on the 4 clusters. Table 6.4 summarizes the performance results and achieved speed-ups for the cell search implementation on Ninesilica and the 4-cluster architecture.

6.4.2 FFT Parallelization and OFDM Receiver

The implementation of parallel FFT is a hot topic in the research community and many research works can be found that consider different aspects of the parallel implementation. In [5], the authors focused their attention on how to reduce the communication overhead, utilizing an NoC as communication infra-structure. Bahn et al. [5] also presents the study of different algorithm solutions for the data exchange between the PE which led to different performance levels. Our interest was to understand the behavior of different FFT algorithms on parallel architectures. We have considered three FFT algorithms for implementation: radix-2, radix-4, and radix-8. For the parallel implementation on Ninesilica, we studied the algorithms data flows and we implemented the partitioning in the way that could lead into the lowest number of data exchanges between nodes, keeping the overhead of communication as low as possible [3]. Radix-2 algorithm performs three data exchange steps. Figure 6.3a shows the data exchange actors for each one of the three phases. From the figure it is possible to notice that each data exchange phases require the communication and synchronization between pair of nodes with different pair for each exchange phase. Radix-4 algorithm reduces the number of stages required to perform the FFT and at the same times reduces the number of exchange phases required. In fact, for a radix-4 algorithm only two data exchange phases are

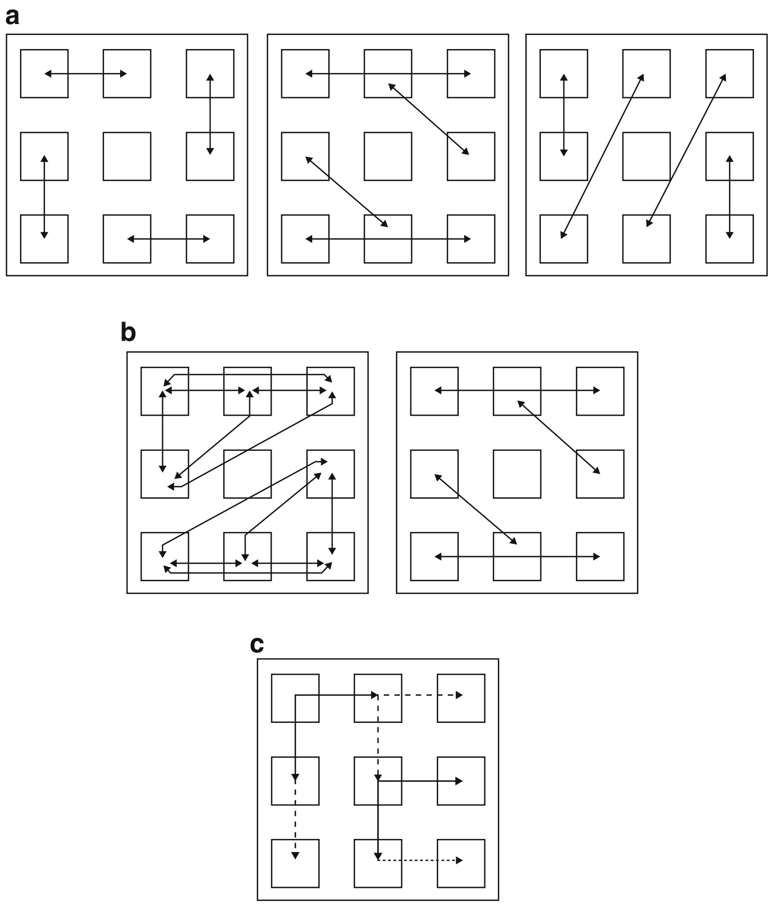


Fig. 6.3 Actors for data exchange phases for radix-N algorithm: (**a**) radix-2; (**b**) radix-4; (**c**) Radix-8

needed (see Fig. 6.3b). However in this case the first data exchange requires the data exchange and synchronization not between pairs of nodes but group of 4 nodes at the time. Therefore, a more complex data exchange and synchronization are required. Finally, radix-8 algorithm requires only a single data exchange phase. This single data exchange phase needs each node to communicate and synchronize with every other node in the architecture. Such operation can be performed taking advantage of the NoC capability of generating broadcast messages. An example of a broadcast message is shown in Fig. 6.3c, where the top left corner node is generating a message that via repetition reaches every node in the system. This leads in an even higher overhead of synchronization and data exchange which potentially could nullify the advantages of having a single data exchange phase.

Table 6.5 FFT algorithms profiling on NineSilica

FFT size	# clock cycles	# clock cycles	
Algorithm	COFFEE	NineSilica	Speed-up
64-point			
Radix-2	22,214	3773	5.9×
Radix-4	10,937	3388	3.2×
Radix-8	10,282	3880	2.6×
2048-point			
Radix-2	1,066,595	154,805	6.9×
Radix-4	656,536	129,803	5×
Radix-8	639,514	221,107	2.9×

Table 6.5 summarizes the profiling results for the implementation of small (64-point) and large (2048-point) FFTs for the three radix algorithms. Analyzing the absolute performance it is possible to notice that radix-4 algorithm shows the best performance in both cases. However, when analyzing the parallelization efficiency, radix-2 algorithms give the best results. The algorithm simplifications introduced by radix-8, which on a single core leads to good performance, are completely hidden by the overhead of communication required for the data exchange, making such approach not suitable for parallel implementations.

Once that the best suitable implementation for the FFT block was identified we implemented a receiver for the IEEE 802.11a/g which bases its physical layer on the OFDM technique. In particular, it utilizes a 64-point FFT for the demodulation. Beside the actual modulation the receiver has to perform tasks such as timing synchronization, channel equalization, and symbol de-mapping. We defined two macro-tasks. The first task takes care of packed detection and timing synchronization; the second one takes care of the demodulation, equalization, and symbol demapping. Figure 6.4 shows the partitioning of the macro-tasks on the single Ninesilica cluster. Timing synchronization is based on the evaluation of correlations sequences. The data is managed by the central node which is also responsible for the evaluation of the results of the correlated data. For the second macro-step the demodulation is performed by utilizing a radix-4 FFT algorithms. While the computational nodes are demodulating the symbol, the central node is performing the evaluation of the coefficient for the channel equalization, and then delivering the new set of data to be demodulated. Such pipelined approach is able to hide the overhead of communication under the computation of the previous symbol.

6.5 Conclusions

In this chapter, we presented Silicon café template and its first instances: Ninesilica cluster and 4-cluster architecture. The utilization of homogeneous approach is not very convenient when compared to heterogeneous solutions. Heterogeneous

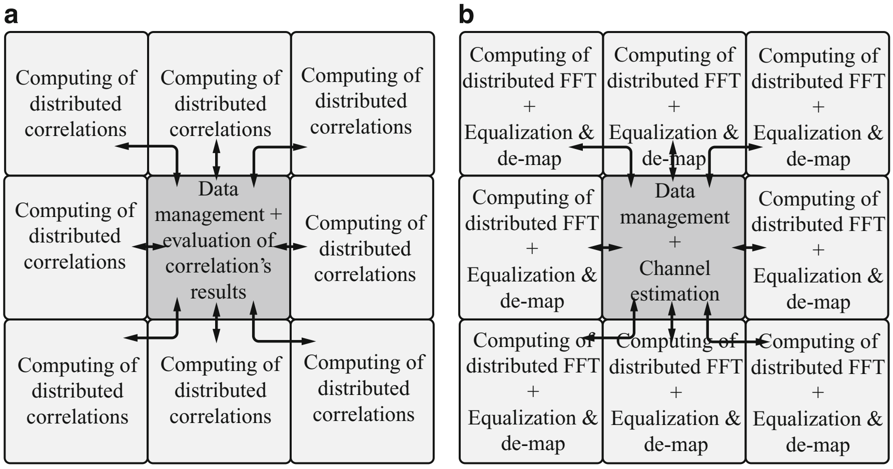

a
Computing of distributed correlations
Data management + evaluation of correlation's results
b
Computing of distributed FFT + Equalization & de-map
Data management + Channel estimation

Fig. 6.4 Macro-steps for the implementation of an OFDM receiver on Ninesilica cluster: (**a**) packet detection and timing synchronization; (**b**) demodulation, channel equalization and symbol demapping

solutions offer the best performance in terms of power, computation efficiency and processing power. However, the number of processing elements inside an MPSoC is continuously growing and heterogeneous solutions might become too complex to design. Therefore the exploration of homogeneous platforms is needed. In this chapter, we showed that homogeneous solutions offer a high scalability in terms of hardware implementation as well as software partitioning. Speed-ups close to Amdahl's limit have been achieved and high energy saving are possible via the utilization of clock gating controlled via software. The possibility to realize architecture composed of clusters would also allow the implementation of hierarchical solutions and architecture composed by a heterogeneous set of homogeneous clusters.

References

1. Ahonen, T., Nurmi, J.: Hierarchically heterogeneous network-on-chip. In: Proceedings of International Conference on "Computer as a Tool", EUROCON, 9–12 September 2007, pp. 2580–2586
2. Airoldi, R., Garzia, F., Ahonen, T., Milojevic, D., Nurmi, J.: Implementation of W-CDMA cell search on a FPGA based multi-processor system-on-chip with power management. In: Proceedings of the IX International Symposium on Systems, Architectures, MOdeling and Simulation (SAMOS IX), pp. 88–97. Lecture Notes in Computer Science. Springer, Berlin (2009)
3. Airoldi, R., Garzia, F., Nurmi, J.: FFT algorithms evaluation on a homogeneous multi-processor system-on-chip. In: Proceedings of the 2010 International Conference on Parallel Processing (ICPP) Workshops, 13–16 September 2010, San Diego, pp. 58–64

4. Bahl, S.K.: Cell searching in WCDMA. IEEE Potentials **22**(2), 16–19 (2003)
5. Bahn, J.H., Yang, J., Bagherzadeh, N.: Parallel FFT algorithms on network-on-chip. In: Proceedings of Fifth International Conference on Information Technology: New Generations ITNG, pp. 1087–1093 (2008)
6. Cheng, C.-C., Lin, C.-H., Li, C.-T., Chen, L.-G.: ivisual: An intelligent visual sensor SoC with 2790 fps CMOS image sensor and 205 GOPS/W vision processor. IEEE J. Solid State Circuits **44**(1), 127–135 (2009)
7. COFFEE Core Project Website, consulted 20.03.2016. http://coffee.tut.fi
8. Dahlman, E., Beming, P., Knutsson, J., Ovesjo, F., Persson, M., Roobol, C.: WCDMA-the radio interface for future mobile multimedia communications. IEEE Trans. Veh. Technol. **47**(4), 1105–1118 (1998)
9. Lee, S., Oh, J., Park, J., Kwon, J., Kim, M., Yoo, H.-J.: A 345 mW heterogeneous many-core processor with an intelligent inference engine for robust object recognition. **99**, 1 (2010). Early Access
10. Pham, D., Asano, S., Bolliger, M., Day, M.N., Hofstee, H.P., Johns, C., Kahle, J., Kameyama, A., Keaty, J., Masubuchi, Y., Riley, M., Shippy, D., Stasiak, D., Suzuoki, M., Wang, M., Warnock, J., Weitzel, S., Wendel, D., Yamazaki, T., Yazawa, K.: The design and implementation of a first-generation cell processor. In: 2005 IEEE International Proceedings of Digest of Technical Papers Solid-State Circuits Conference ISSCC, pp. 184–592 (2005)
11. Rajbanshi, R., Wyglinski, A.M., Minden, G.J.: An efficient implementation of NC-OFDM transceivers for cognitive radios. In: 1st International Conference on Cognitive Radio Oriented Wireless Networks and Communications, 8–10 June 2006, pp. 1–5
12. Rossi, D., Campi, F., Spolzino, S., Pucillo, S., Guerrieri, R.: A heterogeneous digital signal processor for dynamically reconfigurable computing. IEEE J. Solid State Circuits **45**(8), 1615–1626 (2010)
13. Wang, X.: OFDM and its application to 4G. In: Proceedings of International Conference on Wireless and Optical Communications 14th Annual WOCC 2005, 22–23 April 2005, p. 69
14. Wolf, W.: Multiprocessor system-on-chip technology. IEEE Signal Process. Mag. **26**, 50–54 (2009)
15. Zhang, Q., Kokkeler, A.B.J., Smit, G.J.M.: Cognitive radio design on an MPSoC reconfigurable platform. In: Proceedings of 2nd International Conference on Cognitive Radio Oriented Wireless Networks and Communications CrownCom 2007, pp. 187–191 (2007)

Part II
Software-Based Radio Cognition and Implementation Tools

Chapter 7
Application of the Scalable Communications Core as an SDR Baseband

Anthony Chun and Jeffrey D. Hoffman

7.1 Introduction

Today's consumers are increasingly reliant upon their wireless computing devices (smart phones, tablets, wearables, and ultrabooks) for computation and communication, anywhere and anytime. Users expect constant connectivity for voice, data, and multimedia access that has led to an explosion in the number of standards that a wireless device needs to support: a multitude of cellular standards, mobile broadband such as WiMAX (Worldwide Interoperability for Microwave Access) IEEE 802.16 or 3GPP Long Term Evolution (LTE), wireless local area network (WiFi), personal area network (Bluetooth), satellite navigation such as the Global Positioning System (GPS) or Galileo or GLONASS, various broadcast digital TV standards, and wireless display.

For the radio developer, there are a number of aspects to provide a superior wireless device for the consumer:

1. Support for all of the target radio standards with excellent quality of service.
2. Seamless management of the target radio standards based upon the user's profile.
3. Implementation of the radio that maximizes the user experience, including small form factor, low cost, and long battery life.

In order to meet the requirement to support numerous radio standards requires that the device incorporates multiple antenna, front end, baseband, and Media

A. Chun (✉)
Internet of Things Group, Intel Corporation, 2200 Mission College Blvd., 95054 Santa Clara, CA, USA
e-mail: anthony.l.chun@intel.com

J.D. Hoffman
Internet of Things Group, Intel Corporation, 2111 NE 25th Ave., 97124 Hillsboro, OR, USA
e-mail: jeffhoffmanor@gmail.com

W. Hussain et al. (eds.), *Computing Platforms for Software-Defined Radio*,
DOI 10.1007/978-3-319-49679-5_7

Access Control (MAC) components. Because instantiating a radio chain for each standard can quickly increase the size of the radio, flexible radio solutions that implement multiple standards have become the goal of radio developers. This has led to the development of Software Defined Radio (SDR) technology that is comprised of flexible components where each component can be reconfigured or reprogrammed as necessary.

For a programmable radio, selecting and managing the wireless configurations within the consumer's device is becoming increasingly complex. For example, the consumer may maintain a profile indicating which wireless services he or she is subscribed to based upon physical location, time of day, etc.

We can envision a set of programmable wireless radio technologies to meet the consumers' needs and provide a seamless user experience:

1. *Flexible Radio*: the radio supports multiple standards although not all are available simultaneously. The change in the radio configuration is initiated by the user and the rate of change of the radio's configuration is relatively slow, such as when the user arrives in a different location.
2. *Adaptive Radio*: the radio adapts automatically to maximize the user experience based on sensing of the radio environment, for example, by increasing throughput or minimizing latency by selecting the appropriate Modulation Coding Scheme if the radio environment is favorable. The change in the radio configuration is initiated by the radio and the rate of change of the radio configuration is relatively fast, such as when the radio detects an improvement in the Signal-to-Noise ratio or the availability or lack of availability of a particular wireless standard. An example might be a handoff between the cellular and WiFi radios when the user enters his/her home.
3. *Cognitive Radio*: the radio adapts autonomously to maximize the user experience based on the radio environment and the user's preferences. The change in the radio configuration is initiated by the radio using a cognition engine and the rate of change of the radio personality is relatively fast, such as when the radio detects nearby unused spectrum in order to maximize the available data rate and minimize the user's costs.

There are subtle differences in the above definitions: the required rate of change of the radio configuration can be slower in a Flexible Radio and faster in an Adaptive or Cognitive Radio. On the other hand, the Cognitive Radio requires more sophisticated control and sensing capability than either the Adaptive or Flexible Radios.

In order to meet the consumers' demand for a wireless device that has a small form factor, is inexpensive and is energy-efficient with a long battery life, the radio architecture must be highly optimized. One of the challenges in the introduction of SDR technology in consumer wireless devices (vs. in base stations) is meeting the power and area requirements with flexible components that incur a higher cost in area and power when compared to the corresponding fixed versions.

A critical component of the SDR is the digital baseband in the physical layer (PHY), which executes computationally intensive algorithms but also must be flexible in order to accommodate different algorithms for modulation/demodulation, equalization, encoding/decoding, etc. The *Scalable Communications Core* (SCC) is an example of an efficient and flexible wireless baseband that was developed by Intel Labs in Intel Corporation [4, 5, 11]. The SCC architecture is a multi-core SoC consisting of flexible coarse-grained *Processing Elements* (PE's) connected via a *Network-on-Chip* (NoC).

The architecture achieves flexibility via the following features:

- The PEs implement common baseband signal processing kernels such as demodulation, forward error correction decoding, and descrambling that were defined after studying numerous wireless standards. Depending upon the kernel, each PE is either programmable (complete flexibility) or configurable (limited flexibility).
- The programmable PEs are programmed in a high level language via a tool chain that includes a compiler, debugger, simulator, and profiler.
- The configurable PEs are configured by setting appropriate control registers.
- The NoC enables flexible data routing between the right set of PEs for each different standard.

The architecture is also area- and energy-efficient:

- Each PE design is tailored to its target kernel with the appropriate arithmetic types, operators, and memory which is kept local to improve energy efficiency.
- The flexibility of each PE is constrained to be sufficient to meet the target requirements—another way to improves both energy and area efficiency.
- The PEs are multi-threaded so that the architecture can simultaneously process multiple standards using shared resources, which improve area efficiency.
- Data-driven control, in which configuration parameters are sent with data, enables relatively fast reconfiguration of the PEs for supporting a different standard with minimal intervention of a microcontroller, which improves energy efficiency.

To demonstrate the technology, we implemented a test chip based upon the SCC architecture, programmed it for a variety of standards and measured the area and power consumption of the implementation. When it was originally designed, SCC was intended to demonstrate a Flexible Radio architecture by showing how multiple standards could be implemented on a programmable baseband; however, we believe that SCC architecture with the integration of a Cognitive Engine can be potentially applied to Adaptive and Cognitive Radio applications. In any case the architectural and programming lessons that we learned from this research are worth sharing with other SDR developers.

This chapter is organized as follows: in Sect. 7.2, we begin by describing the SDR baseband requirements followed by an analysis of the key computational kernels in Sect. 7.4. In Sect. 7.5, we discuss the SCC architecture, kernels, and the NoC interconnect. In Sect. 7.6, we describe the implementation of various wireless protocols and Sect. 7.7 summarizes the measurements of our silicon prototype. Finally, we conclude with the lessons that we learned in Sect. 7.8.

7.2 SDR Requirements

The ETSI standard describes the capability requirements for SDR [9]. These requirements and their impact on the baseband architecture are summarized below:

1. Multiple Standards: The SDR device operates on multiple bands using multiple bandwidths ranging from 200 kHz to 500 MHz and covers existing cellular and non-cellular radios and new radio technologies. *Impact: the radio must be programmable for a range of current standards and be extensible via hardware computational resources and software tools to support future standards.*
2. Incremental Configuration: The SDR device loads a new radio while the current set of radios is already running. *Impact: the entire radio cannot be shut down when a new radio configuration is added and resource allocation must be dynamic.*
3. Simultaneous Operation: The SDR device executes a number of radio standard simultaneously using coexistence rules to mitigate interference. *Impact: the radio must have a global controller to coordinate the different operational radios.*
4. Resource Sharing: The SDR device enables sharing of computational and memory resources among multiple radio applications. *Impact: the architecture must enable hardware sharing to maximize efficiency and support context switching between different radio applications.*

7.3 Supported Standards

As indicated in the previous section, the consumer SDR should support cellular and non-cellular standards. Some of the desired applications include cellular, Wireless Wide Area Network (WWAN) for mobile broadband, Wireless Personal Area Network (WPAN) for communications between the device and a peripheral, Wireless Local Area Network (WLAN) for local access, Global Navigation Satellite System (GNSS) to support Location-Based services, Digital Video Broadcast for High Definition video, and Wireless Display for projecting video. Some examples of the radios that an SDR should support are shown in Table 7.1.

It is apparent from Table 7.1 that the flexible baseband needs to support a wide range of data rates from 50 bits/s to 4 Gbits/s, modulations ranging from spread BPSK to OFDM with 8K subcarriers, and potential requirement for state-of-the-art 4×4 MIMO processing.

Table 7.1 Consumer SDR basebands should support a variety of wireless standards

Application	Example standard	User data rates	Notes
Cellular	3GPP Release 8 LTE [1]	324.6 Mbits/s (peak downstream rate)	4×4 MIMO, OFDMA with 4096 carriers, SC-FDMA
Digital video broadcast	DVB-T, ETSI EN 300 744 [8]	Up to 31.7 Mbits/s	OFDM with 8K subcarriers
Global navigation satellite service	GPS [10]	50 bits/s	BPSK direct sequence spread spectrum
Wireless local area network	IEEE 802.11a/n [12]	Up to 600 Mbits/s	4×4 MIMO, OFDM
Wireless wide area network	WiMAX IEEE 802.16e[a] [13]	40 Mbits/s	OFDMA with 2048 subcarriers
Wireless personal area network	Bluetooth v3.0+HS [3]	Up to 24 Mbits/s	Frequency hopping GFSK and Differential 8PSK
Wireless display	IEEE 802.15.3c-2009 [14]	Up to 5 Gbits/s	OFDM with 512 subcarriers

[a] WiMAX was still being considered as a WWAN standard in 2008 when this work was done

7.4 Baseband Algorithms

7.4.1 Kernels

We developed the SCC architecture by first examining numerous standards and studying baseband algorithms for many wireless standards, including some that listed in Table 7.1 [5]. We identified baseband kernels that are common to numerous standards, as shown in Table 7.2.

For our architecture, we arbitrarily defined the digital baseband as the physical layer operations between the Digital Front End (DFE), which includes filtering, resampling and gain control operations, and the Media Access Control (MAC), which is part of the data link layer and includes addressing and access control mechanisms.

An example of a generic OFDM baseband is shown in Fig. 7.1. The key baseband algorithms for the receiver include the following:

- Channel Estimation: identification of the frequency response of the RF channel characteristics such as fading and multipath based on pilot sequences or carriers.
- Channel Equalization: mitigation of the RF channel characteristics based on either time or frequency-domain techniques.
- Multiple-Input and Multiple-Output (MIMO) detection: processing of spatial diversity based on Minimum Mean Squared Error or Maximum Likelihood Detection (or suboptimal variations of these) algorithms.
- Demodulation: correction of carrier and timing offsets and mapping of channel symbols to either soft or hard decisions.

Table 7.2 Wireless standards share common baseband kernels

Algorithm	WiFi	WiMAX	3G/LTE	DVB-H	UWB
FIR/IIR	X	X	X	X	X
Correlation	X	X	X	X	X
Spreading		X	X		
FFT	X	X	X	X	X
Channel estimation	X	X	X	X	X
QAM mapping	X	X	X	X	X
Interleaving	X	X	X	X	X
Convolutional coding	X	X	X	X	X
Turbo coding		X	X		
Reed–Solomon coding		X		X	X
Randomization	X	X	X	X	X
CRC	X	X	X		X

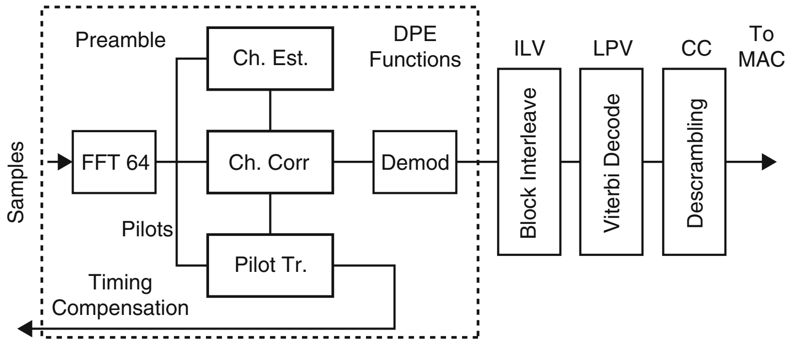


Fig. 7.1 Block diagram of an example OFDM receiver baseband

- Despreading: correlation of spread spectrum code sequences and recovery of spread data.
- Block or convolutional deinterleaving: mitigation of burst bit errors.
- Forward Error Correction (FEC): decoding of convolutional, Turbo, Low Density Parity Check (LDPC), and Reed–Solomon codes to mitigate channel bit errors (Note: only the Viterbi decoder is shown in the figure).
- Descrambling: randomization of data patterns to improve spectral characteristics.

On the transmitter side, the baseband also includes the corresponding inverse operations of scrambling, FEC encoding, interleaving, spreading, insertion of pilots, and modulation.

We have arbitrarily assigned the filtering, resampling, and prefix removal operations for Orthogonal Frequency Division Multiplexing (OFDM)-based protocols to the Digital Front End that is implemented in a dedicated block in our architecture.

7.4.2 Kernel Analysis

After studying numerous wireless standards and analyzing the required baseband algorithms, we observed many characteristics of the baseband kernels that are summarized in Table 7.3.

Table 7.3 Observations of digital baseband kernels

Observation	Description	Implication
Regular data flow	Data flow is regular and unidirectional. Example: in the receiver pipeline, data flows from the antenna to the front end, analog-to-digital converter (ADC), demodulator, deinterleaver, FEC decoder, descrambler, and then to the MAC.	Group operations into kernels where data movement and data types are localized.
Constrained kernel types	The kernel types for communications algorithms are very well defined and constrained. Example: for OFDM-based standards, FFTs and inverse FFTs can be grouped into an optimized FFT butterfly data path.	Define kernels to implement specific arithmetic operations.
Local data access patterns	Within each kernel, the data are localized and the memory access patterns are regular. Examples: fixed data strides used for each stage of the FFT or Viterbi Algorithm Add-Compare-Select (ACS) butterflies.	Keep memory local to each kernel, which saves on bus power, reduces number of ports on each memory and power consumption at the expense of overall increased area. Simplify the design of address generator units.
Amenable to parallelism	Computationally intensive algorithms such as FFT and Viterbi decoding can be implemented to take advantage of parallelism.	Implement data-level parallelism for many of the kernels.
Stream processing	For each kernel, samples stream through the data path continuously and configuration of the data path remains fixed for a large number of consecutive samples.	Reduce the number of instructions that are read and decoded from code memory, which reduces the number of memory touches thereby reducing power consumption.
Limited data types	Data types within each kernel are constrained. Examples: the demodulator processes complex 8 or 10-bit samples while the descrambler processes hard decision bits.	Tailor the data path of each kernel for a specific data type, which improves area and energy efficiency.
Limited global control	Global control of the baseband is needed relatively infrequently. Example: control is needed for initial acquisition of a particular protocol or when a base station handoff occurs.	Reduce the utilization of the global controller by decentralizing control operations within a kernel.

7.4.3 Kernel Flexibility

After defining the key baseband kernels, the next question is determining how flexible the kernels need to be. The kernels need to be flexible enough to support differences between standards. For example, the equalization and demodulation is different for OFDM-based protocols such as WiFi, WiMAX, LTE, and DVB-H as compared to single carrier modulations such as GPS, Bluetooth, and the Advanced Television Systems Committee (ATSC) broadcast digital TV standard. Even within the set of OFDM-based protocols, the FFT sizes and locations of pilots may be different; the turbo codes used in LTE and WiMAX are also different.

Flexibility is also essential to enabling handset vendors who use the same hardware baseband to differentiate themselves by providing superior performance with their own unique software. For example, a vendor could use a more powerful equalizer algorithm to provide better quality of service compared to a competitor. Some of the kernels need to be more flexible than others. For example, the algorithms for channel estimation and equalization can be optimized by the developer to provide better performance. On the other hand, there is a limited set of convolutional codes, turbo codes, and Reed–Solomon codes that are used in most standards so that these kernels do not need complete flexibility.

It is apparent that a mixture of flexible and configurable kernels may be sufficient to address most of the baseband processing requirements and that total flexibility (and the resulting power and area costs) is not required. However, there is the possibility of finding an unforeseen algorithm where the limited flexibility results in an unsuccessful implementation of a protocol, which we will describe later.

7.4.4 Hardware Accelerators

While it is certainly feasible to implement a flexible baseband on general purpose multi-core processors, DSPs, or FPGAs, we believe that the energy and area costs would be too high for a small battery-operated handheld device. This observation led us to focus on baseband implementations based on hardware accelerators. The type and number of different accelerators has an impact on the hardware and software complexity.

Homogeneous accelerators define a single type of accelerator for all kernels and use a single programming model but are less energy- and area-efficient if the kernels are very different. Heterogeneous accelerators that are optimized for different kernels may be more efficient but more difficult to program without a unified software development tool.

Fine-grained accelerators, in which each accelerator performs a portion of the computation within a kernel, may offer greater flexibility for composing an arbitrary algorithm but may also have higher costs in terms of interconnect area and programming complexity. Coarse-grained accelerators, in which each accelerator implements an entire kernel, are less flexible for implementing algorithms that deviate from the kernel definition and are not scalable but may be more efficient in terms of interconnect area and have a simpler programming model.

7.5 Architecture of the Scalable Communications Core

The previous section described aspects of baseband algorithms and kernels that led to our definition of the architecture of the Scalable Communications Core. This section discusses the SCC architecture, its components, and the features that enable it to meet the competing demands of flexibility and efficiency.

7.5.1 SCC Requirements

The requirements for the SCC research project were much more modest than the broader SDR requirements discussed in Sect. 7.2. Our initial goal was to support Wireless LAN (WiFi), Wireless broadband (WiMAX), and broadcast digital TV on a programmable baseband with simultaneous operation of either WiFi or WiMAX and digital TV, enabling the consumer to surf the Internet while watching broadcast TV. We had no explicit requirements for resource sharing or configuration time. However, the architecture that we developed inherently addresses these requirements.

7.5.2 Overall SCC Architecture

Figure 7.2 illustrates the implementation of SCC architecture in our silicon test chip. Based upon the algorithm analysis described in the previous section, we defined an architecture consisting of a heterogeneous set of coarse-grained processing elements, a packet-based two-dimensional 3×3 NoC data interconnect, inband, data-driven control flow, and out-of-band configuration using an off-the-shelf microcontroller.

7.5.3 Processing Elements

The SCC architecture consists of heterogeneous coarse-grained accelerators, each targeted for a different kernel. We selected heterogeneous accelerators because they can more efficiently implement the diverse set of baseband kernels, although

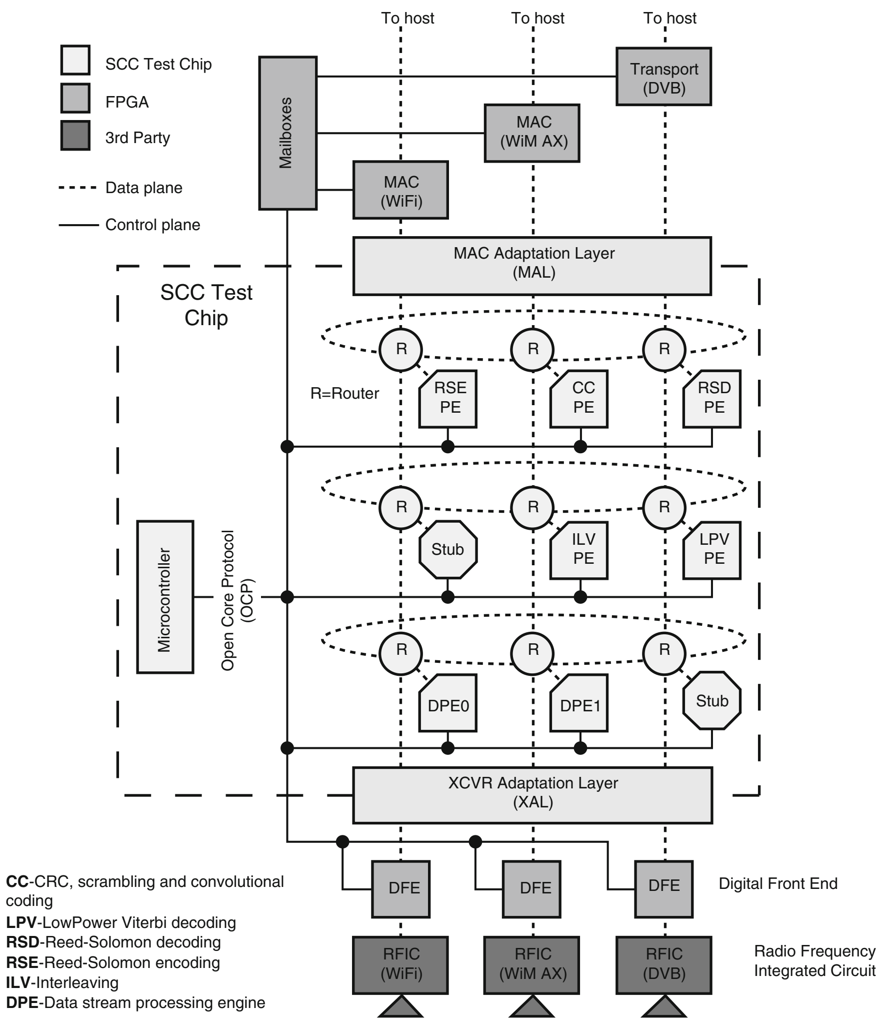


Fig. 7.2 Block diagram of the Onega baseband test chip prototype platform showing the 3×3 Network-on-Chip, seven PEs, the microcontroller and off-chip interfaces to the MAC (MAC Adaptation Layer), and Digital Front End (XCVR Adaptation Layer)

this placed a greater burden on the developers to program them. We decided on coarse granularity because of the constrained flexibility that is required for the target kernels.

The PEs currently defined in SCC are summarized in Table 7.3. Each PE is optimized for a different kernel to improve area and energy efficiency and is configurable for a range of operations.

The most flexible PE is the Data stream Processing Element (DPE), which is a programmable stream accelerator that covers a wide range of fixed point mathematical operations. It consists of a flexible Very Long Instruction Word (VLIW) data path engine and a closely coupled programmable microcontroller. The DPE performs computationally intensive operations such as the FFT/IFFT, channel estimation, equalization, despreading, and soft-bit mapping.

Two instances of the DPE are included in the SCC prototype that was taped out, one with more memory than the other. For example, the larger DPE was used to perform larger FFTs that are required for WiMAX processing. As a result of processing the larger FFT size, the throughput of the larger DPE is less that of the smaller DPE. The DPE is very efficient for streaming workloads since the data path configuration is fixed for the duration of a particular stream of data so that the VLIW instructions are issued infrequently. The other PEs in the SCC architecture include an Interleaver (ILV), Low Power Viterbi Decoder (LPV), Reed–Solomon Encoder (RSE) and Decoder (RSD), and a CRC/Convolutional Coder (CC). A High Speed Viterbi Decoder (that simultaneously decodes multiple threads) and a Turbo Decoder were designed but not included in the silicon prototype because they did not meet the tapeout schedule (Table 7.4).

7.5.4 Resource Sharing

To take advantage of the commonality between the kernels used by different standards, most of the PEs were designed to support multiple threads so that hardware resources could be shared to improve the area efficiency. Some of the key features that enabled resource sharing include defining control and configuration formats for each thread, supporting context switching in the data path, providing enough data path resources to meet the real-time throughput requirements for all of the target threads, and clocking the data path faster than real-time.

Table 7.4 Summary of SCC processing elements showing latency, throughput, and area in a 65 nm process and clock rate of 233 MHz

Name	Label	Function	Throughput (MS/s)	Area (mm^2)
Data stream processing engine	DPE0	FFT, channel estimate, phase-freq. estimate, QAM map	160	2.13
Data stream processing engine	DPE1	FFT, etc.	111	4.69
Interleaver	ILV	Puncturing/depuncturing, block, and convolutional interleaving and deinterleaving	932	1.57
Low power Viterbi	LPV	Viterbi decoding (single thread)	58.2	0.21
Reed–Solomon decode	RSD	Reed–Solomon decoding	76.1	0.26
Reed–Solomon encode	RSE	Reed–Solomon encoding	123	0.09
CRC and convolutional encode	CC	CRC, scrambling/descrambling, convolutional encoding	932	0.21
ARC controller	ARC	System control and configuration	N/a	0.73

An example of resource sharing is the Interleaver PE. This PE includes a single data path that is shared among multiple baseband inputs that are designated as "streams." The data path can implement a different interleaving algorithm for each radio standard or "function" that is programmed via microcode. For example, the 802.16e WiMAX interleaving algorithm is defined as a function. Overlapping bursts in a WiMAX frame are contextually independent streams that use the same interleaving function.

The PE code memory contains a block of microcode for each of the available radio standards. The Interleaver PE receives a packet consisting of a header and data. The data are stored in available local memory. The Stream ID in the packet header is used to look up the context for that stream of data, where the context could indicate how much of the block of data for that interleaver has been received. The Function ID is used to look up the microcode that is specific to the interleaving algorithm for that radio standard; this is the microcode that is executed by the data path on the data that are stored in local memory. Finally, the Stream and Function IDs are used to look up the header that is applied to the output data packet. The output header is prepended to the output data and the output packet is sent to the next PE in the chain.

7.5.5 NoC Interconnect

To connect the PEs, we initially considered interconnect options such as cross-bars and busses. We eventually selected a packet-based NoC arranged in a two-dimensional 3×3 mesh because it was more energy-efficient than the other options [2]. Each PE is connected to a very simple router that consists of a crossbar switch that supports nearest neighbor connections between PEs, as shown in Fig. 7.3. Data are encapsulated within packets, where the packet size is a function of the PE requirements and the packet headers contain routing and control information. Packet routing between the processing elements is dictated by the protocol lineup. Packets can be routed past unused PEs.

The advantage of the NoC over other interconnect options is that it enables flexible routing between any set of PEs with low area and energy costs because the number of physical multiplexers is kept small. Finally, the NoC supports a modular and extensible architecture in which new PEs to be inserted into future instantiations as necessary; for example, a Low Density Parity Check (LDPC) decoder block could be inserted into the NoC to support a future protocol that requires it.

The NoC was designed to support the traffic corresponding to the simultaneous operation of WiFi/WiMAX and digital TV. For the worst case throughput for concurrent operation of IEEE 802.11n and DVB-T, the minimum bandwidth was designed to be 4.612 Gbits/s [2].

The potential disadvantage of the NoC is the nondeterministic delay that is a function of the current network traffic. To mitigate the potential latency issues, the length of physical packets was kept relatively short. As will be seen in Sect. 7.7, key latency issues for real-time operation of 802.11a were met using this approach.

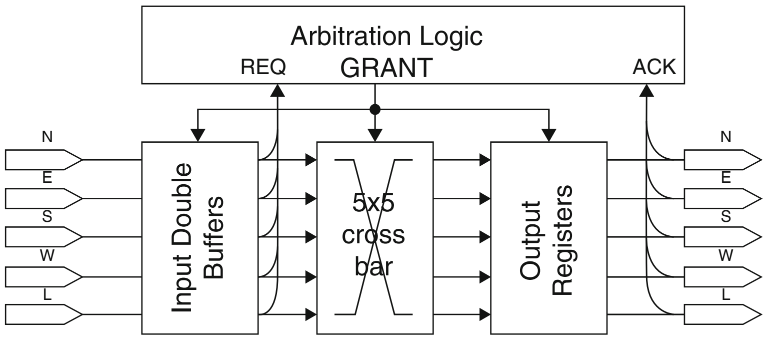


Fig. 7.3 Block diagram of the router that is used in the SCC Network-on-Chip

7.5.6 High Level Control

Wireless protocols require both infrequent high level control and frequent low level control. High level control is required for global changes in protocol processing such as base station acquisition or handoff as well as for coordination of multiple radios to minimize interference. These operations are implemented using an off-the-shelf microcontroller. However, microcontrollers have the potential to increase the overall energy consumption and decrease battery life, especially if they are constantly on. In the SCC architecture, the microcontroller is normally in sleep mode and only wakes up to change the lineup of PEs in the processing flow, significantly reducing the energy consumption of the architecture. Most of the low level PE configuration is implemented as data-driven control using the packet headers.

7.5.7 Low Level Data-Driven Control

In our architecture, low level control (for example, the current modulation type) is implemented via the NoC packet headers that are attached to each packet of data. The headers contain routing information and tags, where the routing information points to the next PE in the processing chain and the tag points to a configuration for the data path that is to be applied to that data.

Data-driven control is depicted in Fig. 7.4. Each packet header includes the coordinates of the destination PE (dX, dY), the Stream ID (SID) that identifies the data stream for supporting multiple threads, and the Function ID (FID) that defines the operation that the data path will perform. The SID is used for context switching of the data path and the FID is used to look up the data path configuration. The combination of the SID and FID is used to look up the destination of the next PE in the processing chain.

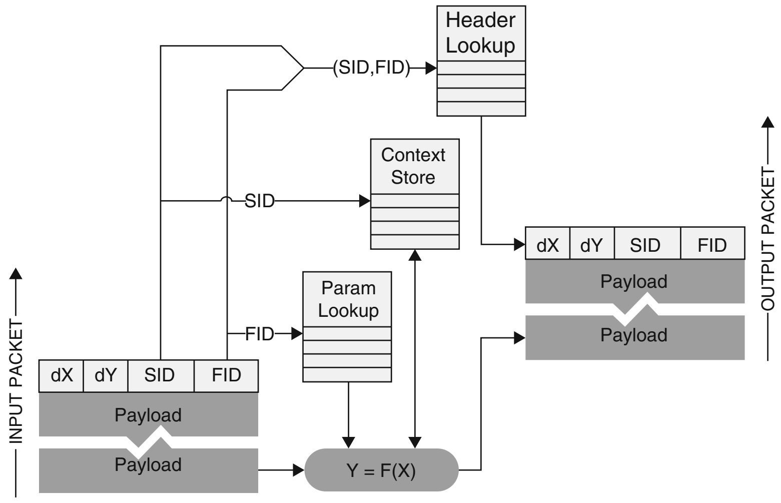


Fig. 7.4 Functional depiction of data-driven control in the SCC architecture. Each packet header includes the destination coordinates (d*X*, d*Y*), a Stream ID (SID) used for distinguishing multiple threads, and a Function ID (FID) that defines the operation that is to be performed

The value of Data-Driven Control is that it enables a new protocol to be dynamically added to the baseband while it is running. Here is an example. The configurations for Protocols A, B, and C have been pre-generated and are stored in the corresponding configuration memories of each PE in the processing chain. Suppose Protocols A and B are currently running on the baseband. When the higher-level control decides to initiate processing of Protocol C, the data corresponding to Protocol C are prepended with the headers corresponding to this protocol. As the packets flow through the baseband, each PE reads the packet header and implements the processing for Protocol C. The reconfiguration of the baseband is performed with minimal higher-level intervention and without reconfiguring the entire baseband. This saves power since the microcontroller can continue to sleep and the baseband can continue uninterrupted.

7.5.8 Programming Technology

Because of the complexity of the baseband algorithms, having capable programming technology is key to enabling developers to efficiently program the architecture for a new protocol. However, one of the challenges of a heterogeneous accelerator architecture is providing a uniform programming environment for all of the PEs since the range of programmability varies with the PE. For example, programming the PEs ranges from setting registers for the code polynomials in the LPV, RSE, and RSD to writing microcode for an interleaver implementation in the ILV to writing C code for the FFT in the DPE.

When we started this project, we had a goal of enabling developers to program SCC as if it were one monolithic DSP; unfortunately, because of schedule constraints we were able to develop a complete programming environment only for the DPE. For the DPE, which is the most flexible of the PEs, the tools included a simulator and an integrated development environment that coordinates programming, compiling, debugging, and configuring the DPE [16]. The DPE is programmed via two languages: C code for its internal microcontroller and the *Data stream Programming Language* (DPL) for the data path functions. DPL is a stream language tailored to the DPE architecture that separately specifies the data path configuration and data flow. The DPE programming environment was essential to enabling us to implement complex demodulation algorithms.

Our programming technology did not include an early architecture exploration tool that would have enabled us to quickly simulate the required algorithms and identify gaps in the architecture. As we will describe in the next section, our Bluetooth implementation did not meet real-time requirements because our instruction set did not include some key DPE instructions. As new protocols are added to the architecture, an architecture exploration tool is essential; we have begun developing tools to support this goal [21].

7.6 Protocol Implementations on SCC

This section discusses the implementation of a several wireless protocols on the SCC architecture. We developed two types of protocol implementations: those that were instantiated and validated on the prototype test chip and those that were coded and executed in the software simulation environment. We measured performance and power consumption for WiFi 802.11a/n and a limited feature set of WiMAX 802.16e on the test silicon.

To challenge the flexibility of the architecture and utility of the programming technology, we mapped a limited feature set of Digital Video Broadcast-Handheld (DVB-H), Bluetooth, and GPS protocols after silicon design was completed. The performance for these applications was not optimal because of limitations uncovered in the silicon test chip: the Bluetooth implementation, while functional, did not meet real-time throughput requirements.

The protocol implementation results are summarized here and were described in detail in [5].

7.6.1 WiFi

We implemented and validated all data rates and key operational modes such as receive and transmit for IEEE 802.11a and a subset of 802.11n including 2×2 MIMO on the test chip.

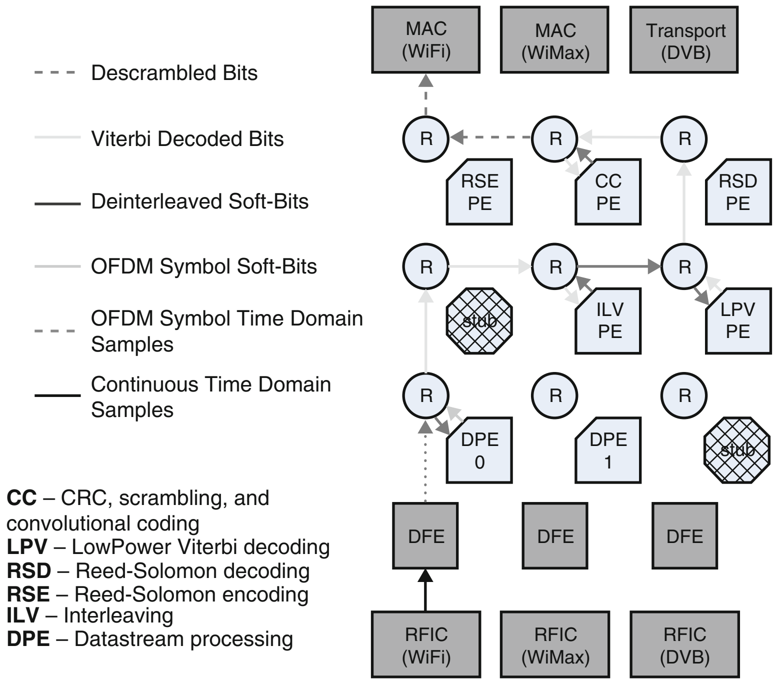

Descrambled Bits
Viterbi Decoded Bits
Deinterleaved Soft-Bits
OFDM Symbol Soft-Bits
OFDM Symbol Time Domain Samples
Continuous Time Domain Samples
CC – CRC, scrambling, and convolutional coding
LPV – LowPower Viterbi decoding
RSD – Reed-Solomon decoding
RSE – Reed-Solomon encoding
ILV – Interleaving
DPE – Datastream processing
MAC (WiFi)
MAC (WiMax)
Transport (DVB)
R
RSE PE
CC PE
RSD PE
stub
ILV PE
LPV PE
DPE 0
DPE 1
DFE
RFIC (WiFi)
RFIC (WiMax)
RFIC (DVB)

Fig. 7.5 Processing flow for WiFi 802.11a Rx showing data routing through the NoC. The stubs correspond to the High Speed Viterbi Decoder and the Turbo Decoder that were not ready for the tapeout. Routing for WiMAX Rx is similar except that the larger DPE is used

The mapping of the computationally intensive 802.11a Rx (receiver) functions and dataflow to SCC is shown in Fig. 7.5 and is described below:

- The DPE receives OFDM symbols in packets from the Digital Front End via the Transceiver Adaptation Layer (XAL) (implemented in an FPGA on the test board that contains the SCC prototype chip) and performs an FFT, channel estimation using the 802.11a long training symbol, channel correction, soft-bit mapping, pilot tracking, and symbol timing correction.
- The ILV deinterleaves soft bits from the DPE.
- The LPV performs Viterbi decoding of soft bits from the ILV.
- The CC descrambles the decoded hard decisions from the LPV.
- The descrambled bits are sent off chip to the MAC via the MAC Adaptation Layer (MAL).

The 802.11a Tx functions and data flow mapping are similar although in the opposite direction.

Two important results that we measured on the test chip are the latency and Bit Error Rate (BER). As discussed in [4], we met the critical 802.11a/n Short Inter-Frame Spacing (SIFS) maximum latency of 16 μs. The measured maximum

SIFS time on the test silicon was 15.955 μs is important because it shows that the nondeterministic, packet-based NoC interconnect can meet stringent real-time requirements.

The AWGN-only BER performance of the initial 802.11a was measured in the lab for a range of data rates and is shown in Fig. 7.6 and suffers from performance degradations of about 5 dB from ideal.

7.6.2 WiMAX

We validated a subset of the WiMAX standard on the test chip. Since WiMAX is an OFDM-based protocol like WiFi, the implementation mapping was similar to that of Fig. 7.5 except that the DPE with more memory was used in order to process the larger WiMAX FFT. The measured BER performance with AWGN showed about 5 dB of degradation with respect to ideal.

7.6.3 DVB-H

We successfully implemented the receiver for the OFDM-based DVB-H TV standard on the software simulator using a mapping that was similar to that of WiMAX. This implementation supported the 2K FFT and QPSK modulation and validated all functions from cyclic prefix removal to descrambling.

7.6.4 Bluetooth

The Bluetooth standard, which uses a single carrier modulation that is quite dissimilar from the OFDM-based protocols that we had previously implemented, stressed the flexibility of the architecture. Although Bluetooth was not in our list of required protocols that were considered in our Silicon prototype, we nevertheless attempted to implement a basic data rate (1 Mbit/s) Bluetooth transmitter and receiver on the DPE software simulator in order to show the versatility of the architecture. We met with mixed success: while the transmitter would operate in real-time, the receiver implementation only supported a data rate of 0.2 Mbits/s well below the goal of 1 Mbits/s. After the test chip had been taped out, we identified a couple of key instructions that the DPE data path did not support that would have enabled it to meet the target throughput. In this case, having early architecture exploration tools to help us identify missing instructions across the full set of required or potential protocols would have been invaluable.

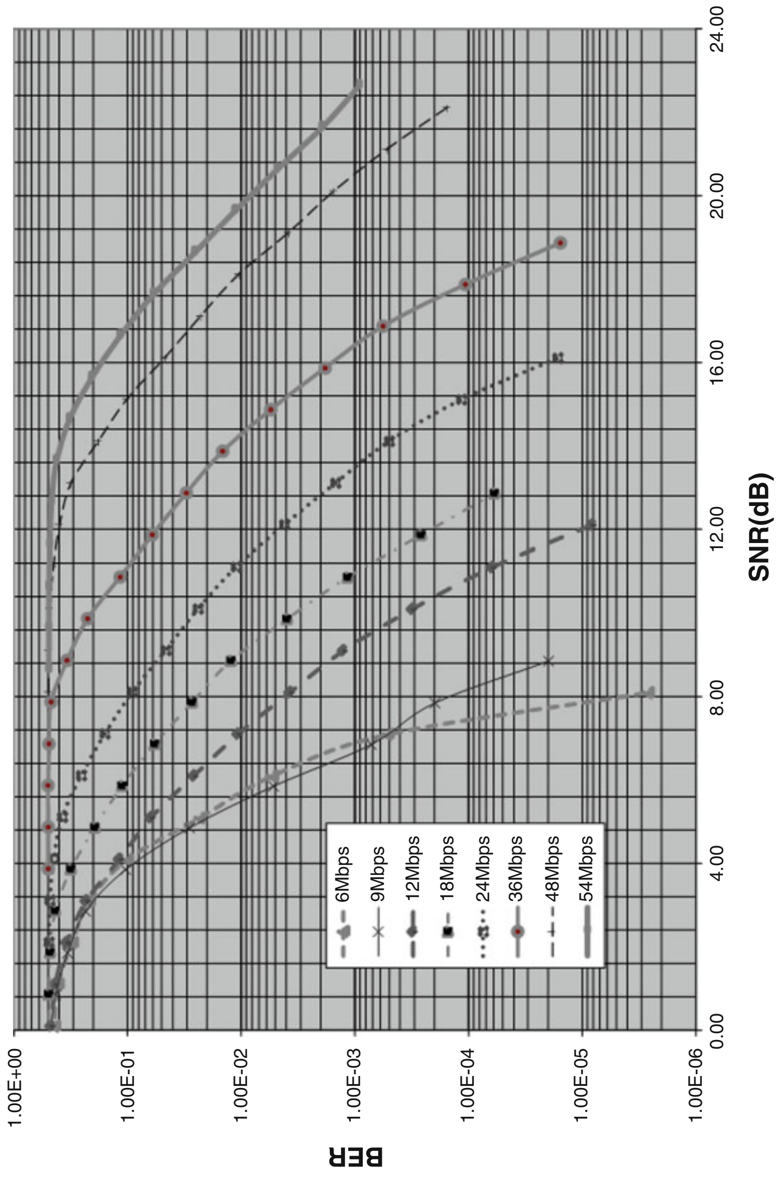


Fig. 7.6 Measured bit error rate performance of various modes of IEEE 802.11a with AWGN on the SCC test chip

7.6.5 Global Positioning System

We implemented the GPS baseband because it is prevalent on most wireless devices. It is also quite different from our initial set of protocols as it uses direct sequence spread spectrum. We implemented two critical parts of a GPS baseband: the acquisition of the Coarse/Acquisition (C/A) spread spectrum code and the despreader code tracking loop. The code acquisition block performed cross-correlation between the received samples and the candidate Gold code in the frequency domain while the despreader tracking loop was implemented as a Costas loop. Both algorithms were coded and run on the DPE simulator and met the real-time requirements for GPS.

7.7 Silicon Prototype

In order to measure the power consumption and area of the architecture, we taped out the *Onega* (named for a lake in Russia) test chip at the end of 2007 in a 65 nm CMOS process. Figure 7.7 is a photo of the die showing the size and placement of the PEs. We completed post-silicon validation in early 2009. The total area is

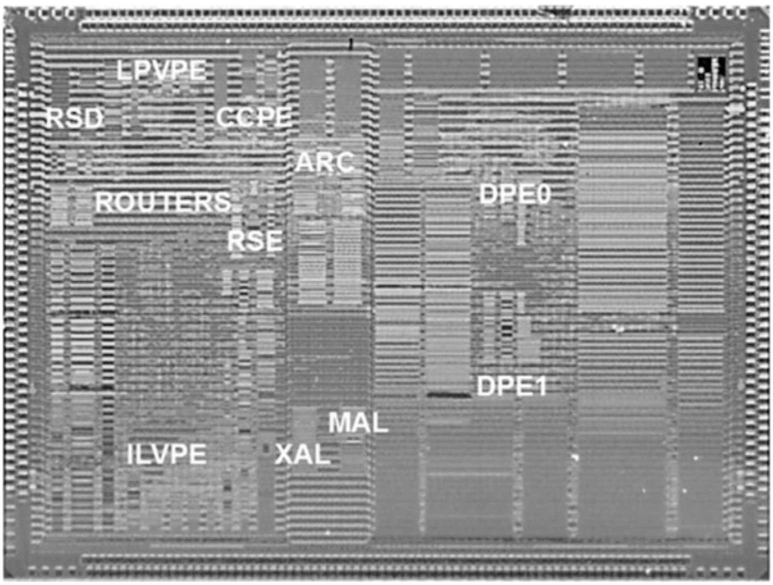


Fig. 7.7 Die photo of the Onega test chip showing the various accelerators (DPE0, DPE1, ILVPE, LPVPE, RSD, RSE, and CCPE), microcontroller (ARC), interconnect (routers) and off-chip interfaces to the MAC (MAC Adaptation Layer or MAL), and Digital Front End (Transceiver Adaptation Layer or XAL)

Table 7.5 Measured power and area of the SCC prototype and comparable architectures, where the comparison results are scaled to a 65 nm process technology

Device	Area-normalized to 65 nm (mm^2)	Rx power normalized to 65 nm process (mW)	Tx power normalized to 65 nm (mW)	Rx power efficiency (nJ/bit)	Tx power efficiency (nJ/bit)	Notes
SCC Core	21	146	104	2.70	1.93	802.11a 54Mbps
SCC DPE only		71	52	1.31	0.96	802.11a 54Mbps
SCC Core	21	188	145	3.25	2.51	802.11n 2×2MIMO 57.78Mbps
SCC DPE only		90	72	1.56	1.25	802.11n 2×2MIMO 57.78Mbps
FAUST[15]	22.08	321.10	270.40	3.21	2.70	OFDM SISO@100Mbps
SIMT [18]	3.33	29.58	N/A	0.93	N/A	Rx only DVB 31.76Mbps
Atheros [17]	7.56	157.17	141.96	0.73	0.66	802.11n 2×2MIMO 216Mbps baseband and MAC
MAGALI [6]	29.6	219	60.0 (estimated)	20.28	5.56	3GPP-LTE Rx MIMO 2×2, Tx MIMO 2×2

24.35 mm^2 with a core area of 20.75 mm^2. The 10.2 mm^2 consumed by the logic includes two DPEs: one optimized for 802.11 OFDM symbol sizes, while the second has larger memories to support the larger OFDM symbols of WiMAX and DVB-H. Table 7.5 summarizes our measured power consumption and energy efficiency in joules per bit for 802.11a and 802.11n for the entire core and for the DPE alone.

7.7.1 *Comparison to other Flexible basebands*

As discussed in [7], flexible baseband architectures can be divided into either Reconfigurable Arrays (either Fine or Coarse-grained) or VLIW architectures. Using this taxonomy, the overall SCC architecture is classified as hybrid that combines a Coarse Grain Reconfigurable Array (like FAUST [15]) with a VLIW processing element: namely, the DPE.

Table 7.5 also compares our results with other flexible basebands, including SIMT [18], IMEC SDR [19], FAUST [15], and MAGALI [6], where the results were normalized to a 65 nm process. In addition, we compared the cost of multi-protocol flexibility of SCC with a single protocol Atheros ASIC [17] that includes

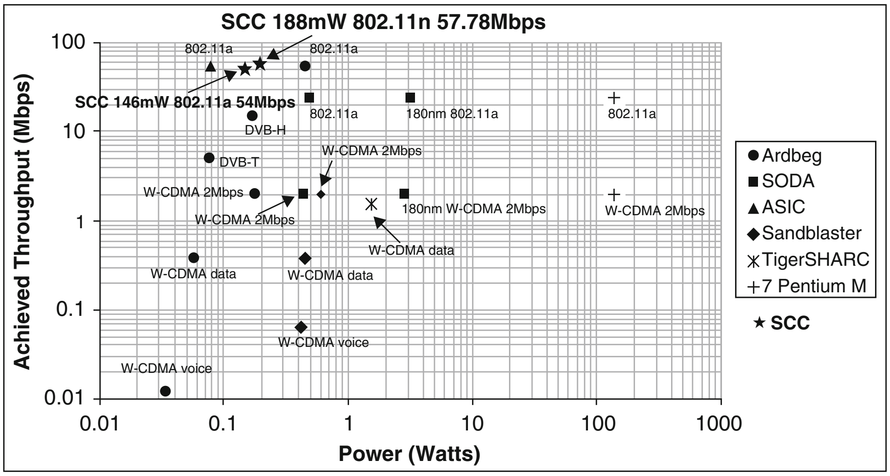


Fig. 7.8 Comparison of SCC power consumption and throughput with examples from [20]. These results are not normalized for improvements in process technology. The ideal implementation would be in the *upper left corner* of the plot

the entire radio. The energy efficiency of SCC compares favorably with other published flexible baseband solutions for OFDM, DVB, and LTE. As shown in Fig. 7.8, we also compared the measured SCC throughput vs. power consumption results for 802.11a and 802.11n with examples of flexible basebands [20]. These results, which were not normalized for different process technologies, indicate that SCC is very competitive with these examples and is only surpassed by an ASIC for energy efficiency at broadband data rates.

7.8 Lessons Learned

The SCC architecture meets the requirements for an SDR baseband, including support for Multiple Standards, Resource Sharing, Incremental Reconfiguration, and Simultaneous Operation. In addition, the measured power and area results are competitive with comparable flexible baseband architectures. In particular, the lessons we learned from this project are worth sharing with the SDR community:

- Architecture Efficiency: The combination of coarse-grained, heterogeneous accelerators, localized memory, and distributed control provides an energy efficiency that was competitive with similar flexible architectures.
- Lightweight NoC Interconnect: A very lightweight packet-based NoC interconnect is power- and area-efficient, will meet the critical throughput and latency requirements with very low area and power penalty, and will support future extensibility of the architecture.

- Targeted Flexibility: By implementing a set of very diverse wireless protocols that met real-time requirements, we showed that a coarse-grained architecture can be made sufficiently flexible and efficient.
- Resource Sharing: Sharing hardware resources in a multi-threaded design enables the architecture to achieve competitive area efficiency.
- Efficient Control: The combination of High Level and Data-Driven Control supports both infrequent global coordination of multiple protocols and frequent low level PE configuration to support dynamic changes to the protocol lineup.
- Programming Technology: High level programming technology was essential to enabling us to implement five very different protocols efficiently. Having an architecture exploration tool would have ensured successful implementation of potential future protocols and is a requirement for the future.

Software Defined Radio technology is essential to meeting consumer demand for ubiquitous wireless access using a multitude of standards via Flexible, Adaptive, and Cognitive Radios. This chapter has described aspects of the Scalable Communications Core architecture that can be applied to these future devices.

Acknowledgements A work as comprehensive as this would not be possible without the dedicated Scalable Communications Core team at Intel Labs at Intel Corporation, especially Mariano Aguirre, David Arditti, Anton Bobkov, David Bormann, Jorge Carballido, Frank Carroll, Jenny Chang, Inching Chen, Aliaksei Chapyzhenka, Vladislav Chernyshev, Luis Cuellar, Saul Cuellar, Carlos Flores, Kapil Gulati, Vladimir Ivanov, Alexey Kostyakov, Ana Ledezma, Jane Lin, Kyle McCanta, Veronica Mikheeva, Sergey Mironov, Brando Perez Esparza, Carlos Ornelas, Vladimir Pavlov, Vladimir Pudovkin, Victor Rivera Alvarez, Mario Rubio, Edgar Borrayo Sandoval, Oleg Semenov, Kirk Skeba, Denise Souza, Tatiana Stounina, Ernest Tsui, Arturo Veloz, and Sergey Yakoushkin.

We dedicate this work to the memory of K. Soumyanath.

References

1. 3rd Generation Partnership Project. Technical Specification Group Radio Access Network; Evolved Universal Terrestrial Radio Access (E-UTRA); Physical Channels and Modulation. (Release 8) 3GPP TS 36.211 V8.1.0 (2007–2011)
2. Arditti Ilitzky, D., et al.: Architecture of the scalable communications core's network on chip. IEEE Micro **27**(5), 62–74 (2007)
3. Bluetooth Special Interest Group. Bluetooth Specification Version 2.0 + HS, 21 April 2009
4. Chun, A.: Key lessons from the scalable communications core: a reconfigurable wireless baseband. IEEE Commun. Mag. **48**(12), 101–109 (2010)
5. Chun, A., et al.: Overview of the scalable communications core: a reconfigurable wireless baseband in 65nm CMOS. In: IEEE Computer Society Annual Symposium on VLSI (ISVLSI '09), pp. 1–6 (2009)
6. Clermidy, F., et al.: A 477mW NoC-based digital baseband for MIMO 4G SDR. ISSCC Digest of Technical Papers, pp. 278–279, Feb 2010)
7. Dejonghe, A., et al., Green reconfigurable radio systems. IEEE Signal Process. Mag. **24**, 20–101 (2007)
8. ETSI. Digital Video Broadcasting (DVB); framing structure, channel coding and modulation for digital terrestrial television. EN 300 744 v1.5.1, Nov (2004)

9. ETSI TR 102 680 V1.1.1 (2009–03) Technical Report Reconfigurable Radio Systems (RRS); SDR Reference Architecture for Mobile Device (2003)
10. Global Positioning System Standard Positioning Service Signal Specification, 2nd edn. 2 June 1995
11. Hoffman, J., et al.: Overview of the scalable communications core. In: IEEE Computer Society Annual Symposium on VLSI (ISVLSI '07), pp. 3–8 (2007)
12. IEEE. Part 11: wireless LAN medium access control (MAC) and physical layer (PHY) specifications. 802.11a-1999 (1999)
13. IEEE. IEEE Standard for Local and metropolitan area networks Part 16: air inter-face for fixed and mobile broadband wireless access systems amendment 2: physical and medium access control layers for combined fixed and mobile operation in licensed bands. IEEE Std 802.16e$^{\mathrm{TM}}$-2005, 28 Feb 2006
14. IEEE. Wireless medium access control (MAC) and physical layer (PHY) specification for high rate wireless personal area networks (WPANs). IEEE Std 802.15.3c-2009, 12 Oct 2009
15. Lattard, D., et al.: A telecom baseband circuit based on an asynchronous network-on-chip. ISSCC Digest of Technical Papers, pp. 258–259, Feb (2007)
16. Mironov, S., et al.: DPL: domain specific programming language and tools. In: XI International Symposium on Problems of Redundancy in Information and Control Systems (2007)
17. Nathawad, L., et al.: A dual-band CMOS MIMO radio SoC for IEEE 802.11n Wireless LAN. ISSCC Digest of Technical Papers, pp. 358–359, Feb 2008
18. Nilsson, A., et al.: An 11mm^2 70mW fully-programmable baseband processor for mobile WiMAX and DVB-T/H in 0.12μm CMOS. ISSCC Digest of Technical Papers, pp. 266–267, Feb 2008
19. Van der Perre, L., et al.: Architectures and circuits for software defined radios: scaling and scalability for low cost and low energy. ISSCC Digest of Technical Papers, pp. 568–569, Feb 2007
20. Woh, M., et al.: From SODA to scotch: the evolution of a wireless baseband processor. In: International Symposium on Microarchitecture, MICRO-41, pp. 152–163 (2008)
21. Yakoushkin, S., et al.: MoonRidge: an automated design framework for workload-optimized power-efficient reconfigurable accelerators. In: 1st Workshop on SoC Architecture, Accelerators and Workloads (SAW-1), 10 Jan 2010

Chapter 8
HW/SW Co-design Toolset for Customization of Exposed Datapath Processors

Pekka Jääskeläinen, Timo Viitanen, Jarmo Takala, and Heikki Berg

8.1 Introduction

It is nowadays common to integrate multiple different computing devices in a single chip, each device serving a different application domain or accelerating a specific part of an application. While specialized fixed function hardware accelerators have been shown to bring performance and efficiency gains, designing and implementing new designs has remained a high-cost exercise. In addition, programmable cores are often favored to fixed-function accelerators for their post-manufacture flexibility (e.g., on-the-field bug fixes or algorithm updates) and their ability to reuse computer hardware in case multiple algorithms can be efficiently mapped to the same accelerator.

Customized processors provide a middle ground between fixed function accelerators and generic programmable cores. They bring benefits of hardware tailoring to programmable designs, while adding new advantages such as reduced implementation verification effort. The hardware of customized processor is optimized for executing a predefined set of applications, while allowing the very same design being used to run other, close enough routines by switching the executed software in the instruction memory.

The processor hardware tailoring is dictated by the use case. In case a processor is customized to execute a single application efficiently, but still support software-based functionality updates, terms *Application-Specific Instruction-set Processor (ASIP)* or *Application-Specific Processor (ASP)* can be used to describe the

P. Jääskeläinen (✉) • T. Viitanen • J. Takala
Tampere University of Technology, Tampere, Finland
e-mail: Pekka.Jaaskelainen@tut.fi; Timo.Viitanen@tut.fi; Jarmo.Takala@tut.fi

H. Berg
Nokia Technologies, Tampere, Finland
e-mail: heikki.berg@nokia.com

end result. A customized processor can also be optimized for classes of applications such as *Software Defined Radio (SDR)*, in which case the term *Domain-Specific Processor (DSP)* is more suitable.

In any case, the processor customization process is demanding due to the abundance of different parameters in the design space to choose from, and very error-prone, which results in high non-recurring engineering costs. Moreover, as the design process of customized processors is usually iterative in nature, porting the required software program codes to new processor variations needs either manual assembly language program rewrites or updating the compiler so it can generate assembly code for each new processor variant. One approach to simplifying the processor customization process is to compose the processor from a set of component libraries and other verified building blocks, thereby reducing the required verification effort. The software porting problem can be alleviated by automatically adapting software development kits.

In the rest of this chapter, we describe *TTA-Based Co-Design Environment (TCE)*, an open source processor design and programming toolset based on a processor template that efficiently supports *Instruction-Level Parallelism (ILP)*. TCE enables rapid design of high-level language programmable cores ranging from tiny scalar microcontrollers to wide VLIW-style machines with a resource oriented design methodology that emphasizes the reuse of predesigned and preverified components.

8.2 Exposed Datapath Processor Template

To implement a design and programming toolset for customized processors, the *design space* of the supported customized processor alternatives needs to be limited. This is done by defining a *processor template* that describes the set of customization parameters within which the processors can vary. The parameters drive the retargeting of the software development toolchain, most importantly the compiler and the simulator. Here we have selected exposed datapath as one of the characteristics of the architectural template. The exposed datapath means that some additional processor datapath details are visible to the programmer or compiler such that the programmer can directly control the additional details. Such architectures have been discussed earlier, e.g., MOVE [4], MOVE-Pro [6], FlexCore [13], STA [6], and ELM [5].

The main focus of the TCE toolset is on energy-efficient data-oriented computing scenarios. Therefore, compiler-controlled static structures are favored over hardware-controlled dynamic structures whenever feasible. This can be seen in the choice of the basis for the processor template of TCE which exploits *transport triggering* paradigm in form of *Transport Triggered Architectures (TTA)* [2, 11].

TTA processors have a programmer-exposed *interconnection (IC)* network. A program is defined as a set of *move instructions* between the ports of the datapath components, including function units and register files. The operations

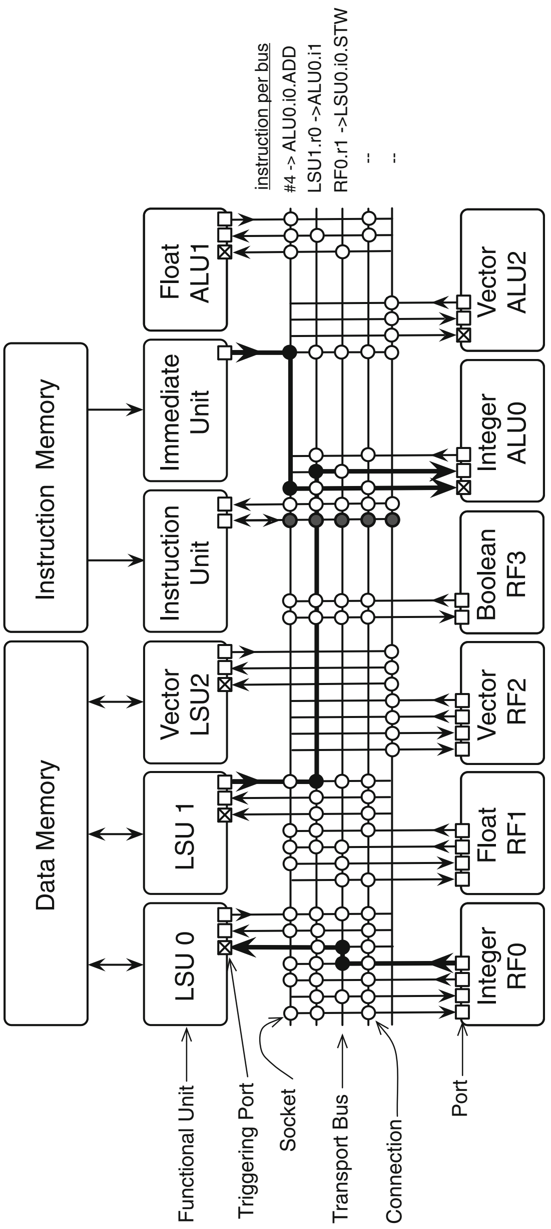


Fig. 8.1 Example of a TTA processor instantiated from the template. In TTA, data transports between components are explicitly programmed. The example instruction defines *move instructions* for three buses out of five, performing an integer summation of a value loaded from a data memory with a constant while simultaneously storing a previously computed value to memory

are triggered as a side effect of transporting data to designated function unit ports. Figure 8.1 presents an example TTA processor with five transport buses, i.e., five data transports can be executed simultaneously. In this processor, each instruction contains five *move slots*, where each slot specifies the data transport carried out in each bus. The figure illustrates execution of an instruction with three parallel moves:

```
#4 → ALU0.i0.ADD;
LSU1.r0 → ALU0.i1;
RF0.r1 → LSU0.i0.STW
```

On the first transport bus, an immediate value is moved to input port 0 of the function unit ALU0. The immediate value is actually obtained from the *immediate unit*, which has only one output port. The move carries also information about the operation to be executed; opcode ADD is transported to function unit along with the operand. The second bus transports an operand from load-store unit LSU1 to the input port 1 of the ALU0. The actual load from memory has been specified by one of the previous moves and this move simply transports the result of the memory access. The third bus is used to transport a value from output port 1 in register file RF0 to the input port 0 of the load-store unit LSU0. The third move contains opcode indicating that the transported word is to be stored to memory. The actual store address has been defined by another move to port 1 of the LSU0. The remaining two buses are not used in the example instruction, thus they can be considered executing an NOP.

Figure 8.1 shows that the instructions control the operation of each transport bus through the instruction unit. The connection to each bus convoys control information, e.g., the source and destination of the transport move, possible opcode for the operation to be executed, etc. The function units are connected to the transport buses with the aid of *sockets*. The interconnection network in our architectural template consists of buses and sockets. The concept of socket is illustrated in Fig. 8.2; each port of a function unit has a socket, which defines the connections to the buses. When the control information in a bus indicates that the port is the destination for the current move instruction, data from the bus is passed to the port. In similar fashion, data from the source port is forwarded to the bus.

In TTAs, operation execution is a side result of operand transport; when operands are available in the function unit, the operation is triggered. In this sense, the execution model reminds static data flow machines. The architecture template used here defines that that one of the input ports is a *trigger port* and a move to this port triggers the operation. Figure 8.3 illustrates the concept; the trigger port is indicated by a cross in the input port. A move to this port will latch data from the bus to trigger register and the operation execution starts with operands from trigger port and other

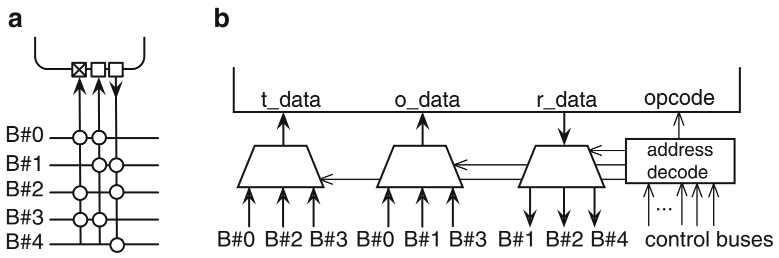


Fig. 8.2 Principal socket interface for function units: (**a**) high abstraction level representation and (**b**) structure. The result register in the output is optional

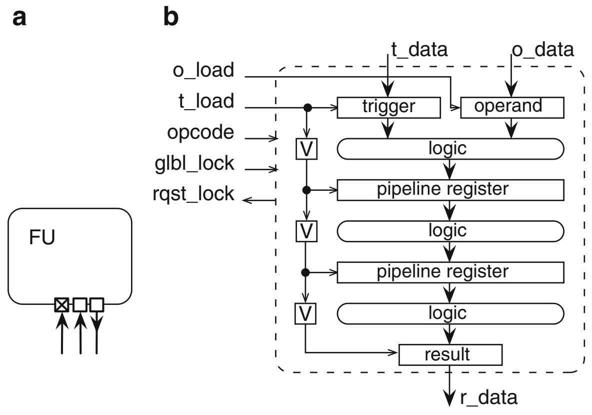


Fig. 8.3 Function units in transport triggered architecture: (**a**) high-level abstraction representation and (**b**) principal block diagram

operand registers; the function unit in Fig. 8.3 expects two operands, thus there is one trigger register and one operand register. The operand to the operand register can be moved by an earlier instruction. The operand can also be moved in the same instruction as the trigger port moves if there are enough buses available.

Function units can be pipelined independently, and there are several methods to pipeline the function units in TTAs. In TCE, we use semi-virtual time latching [3] where the pipeline is controlled with valid bits depicted in Fig. 8.3b. The pipeline starts an operation whenever there is a move to the trigger port, i.e., the *o_load* signal is active. The pipeline operation is controlled by valid bits, which imply that a single pipeline stage is active only once for one trigger move. The result can be read from the result register three instructions after the trigger move. However, the pipeline operates only on the cycles when instructions are issued; if an external or internal event has caused processor to be locked (*glbl_lock* signal is active), the pipeline is inactive. The architectural template requires each operation in function units to have a deterministic latency such that the result read for the operation can be scheduled properly. If the function unit faces an unexpected longer latency operation, e.g., a memory refresh cycle or a function unit has iterative operation of which latency depends on the inputs, the unit can request the processor to be interlocked (by activating the *rqst_lock* signal) until the on-going operation is completed.

SDR applications can often utilize a lot of computational parallelism in its various granularities, but at the same time their usage environment places limits to the power consumption due to thermal design and battery constraints. Therefore, a popular processor architecture choice for SDR applications is a static multi-issue architecture with a simple control hardware. The traditional VLIW architecture fulfills these requirements by providing multiple parallel function units to support ILP while not imposing the hardware complexity from out of order execution support. The VLIW function units in SDR designs can also provide SIMD operations to exploit data level parallelism in the algorithm at hand, which together with ILP can provide very high operation per watt performance in SDR designs.

The use of the TTA paradigm to provide a more scalable alternative for traditional VLIW architectures has been studied extensively in [3]. One of the main findings is that TTAs can support more ILP with simpler low-power register files than "traditional" VLIWs because of the capability to route results directly between function unit port registers, without accessing the larger general purpose register files. In addition, the general ability to control the timing of the operand and result data transports alleviates the parallel access requirements of the register files. The simpler register files can remove the register file from the critical path and bring savings in required chip area and power consumption.

The TTA instruction format resembles those of horizontal microcode programmed architectures which are notorious for their poor instruction density. However, our example designs and experiments indicate that the additional instruction bits due to the exposed datapath control are negligible compared to the savings if the workload is data-oriented and the interconnection network is carefully optimized [8, 14].

The exposed datapath template also opens unique non-apparent opportunities. For example, due to the explicit result transfers, the function units are independently executing isolated modular components in the datapath. In the point of view of processor design methodology, the modularity allows point-and-click style tailoring of the datapath resources from existing processor component databases. It also means the function units can have latencies and pipeline lengths from a single cycle to no practical upper bound because the hazard detection hardware does not need to account for the structural hazards resulting from concurrent completion of operation results. Likewise, there is no practical limit to the number of outputs produced by operations. For example, there have been experiments where long latency fixed function accelerators with tens of result words have been integrated to the processor datapath in order to reduce accelerator communication and synchronization costs.

Using the TTA as a template in TCE means that it allows designing completely new TTA processors from the scratch by allowing the user to define the sets of basic TTA components to include. The customizable parameters are summarized in Table 8.1. An interesting customizable aspect in TTA processors is the interconnection network. As it is visible to the programmer, it enables carefully customizing the connectivity based on the application's need as will be discussed later in this chapter. This can help in achieving implementation goals such as high clock frequency or low power consumption. Another useful feature is the support for multiple disjoint address spaces: one can add one or more private address spaces for local memories inside a core that can be accessed using address space type qualifier attributes in the input C code.

Table 8.1 Configuration parameters in TCE's processor template

Register files

- Number of register files

For each register file:

- Number of registers
- WIDTH
- Number of read/write ports

Function units

- Number of function units

For each function unit:

- Operation set implemented by FU
- Number of input and output ports
- Width of the ports
- Resource sharing / pipelining
- Accessed address space (in case of a load-store unit)

Operation set For each operation:

- Number of operands
- Number of results
- Data type of the results and operands
- Operation state data

Instruction encoding

- Number of instruction formats

For each instruction format:

- Immediate (constant) support

Address spaces

- Number of address spaces

For each address space:

- Size
- Address range
- The numerical id (referred to from program code)

Top level

- Endianness

8.3 Processor Description Formats

TCE uses several file formats and component libraries for setting the processor template parameters for new designs. The roles of the different files and libraries are designed to enhance processor design and verification effort reuse. These XML-based file formats are edited using graphical user interfaces without the toolset user seeing the file contents and if preferred can be edited also using a text editor.

8.3.1 Architecture Description

An *Architecture Description File (ADF)* stores the programmer-visible architectural aspects of a designed processor, such as the number and type of register files and function units, operation latencies and resource sharing of the operations inside function units, and the connections between the processor components [1].

The function unit architectures described in an ADF file can refer to one or more *operation* descriptions. An operation in TCE is a behavioral level abstraction separated from the function unit. Operation descriptions are stored in *Operation Set Abstraction Layer (OSAL)* databases. The granularity of an OSAL operation can range from basic operations to complex multioutput operations with internal state.

Operation descriptions in OSAL contain C++ or DAG-based models to simulate the behavior of the operation in the processor simulator and static metadata to drive the high-level language compiler, such as the number of operands and results, the semantics of the operations, whether the operation reads or modifies memory, or has other effects to the program state. The automatically retargeting compiler needs these properties together with the architecture data from the ADF.

OSAL and ADF enable processor design space exploration at the architectural level. This allows fast exploration cycles with enough of modeling accuracy to drive the selection of the components.

The separation of the concepts of a function unit (with programmer visible latencies) and the behavioral operation descriptions is a key feature which enables trial-and-error cycles with the operation set design without needing to describe the operation candidates as detailed *Register Transfer Level (RTL)* descriptions.

8.3.2 Architecture Implementation

ADF and OSAL are not sufficient alone to implement the processor in hardware. For example, each programmer-identical function unit or register file can be implemented in multiple ways, emphasizing different goals such as small area or high clock frequency.

Hardware implementation choices are defined using an *Implementation Definition File (IDF)*. IDFs connect the architecture components described in ADF files to component implementations stored in *Hardware Databases (HDB)*. HDBs contain RTL level VHDL or Verilog implementations of register files and function units, along with their metadata. The component implementations can be easily reused in future architectures, reducing the validation effort of new customized processors.

8.4 HW/SW Co-design Flow

Processor customization in TCE is usually conducted as a hardware–software co-design process. The processor design is iterated by varying the processor template parameters defined using the TCE file formats while adapting the software to better exploit the features, such as by calling custom operation intrinsics or enhancing parallelization opportunities from the program side.

The co-design process is supported by a set of tools, which are illustrated in Fig. 8.4. Initially, the designer has a set of requirements and goals placed to the end result. It is usual to have a real time requirement as the primary requirement, given as a maximum run time for the programs of interest. A secondary goal can be to optimize the processor for low power consumption or minimal chip area. In some cases, there can be a strict area and/or power budget which cannot be exceeded, and the goal is to design a processor that is as fast as possible and still goes below the budget.

The iterative customization process starts with an initial predesigned architecture, which can be a minimal machine shipped with TCE that contains just enough resources for the compiler to provide C programming language support for the design, or a previous architecture that is close to the placed targets, but needs further customization to fully meet them.

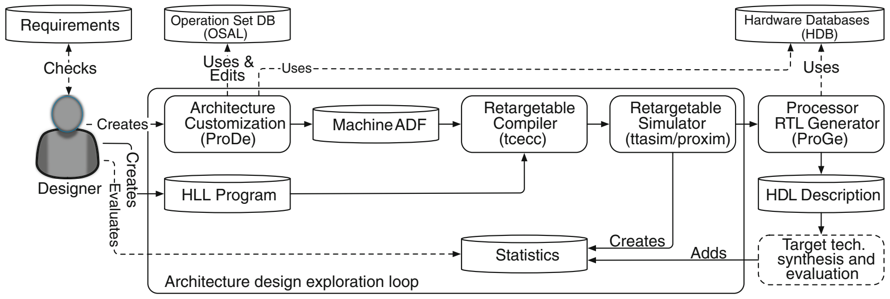

Requirements
Operation Set DB (OSAL)
Hardware Databases (HDB)
Checks
Uses & Edits
Uses
Uses
Designer
Creates
Architecture Customization (ProDe)
Machine ADF
Retargetable Compiler (tcecc)
Retargetable Simulator (ttasim/proxim)
Processor RTL Generator (ProGe)
Creates
Evaluates
HLL Program
HDL Description
Statistics
Creates
Adds
Target tech. synthesis and evaluation
Architecture design exploration loop

Fig. 8.4 TCE processor customization flow

8.4.1 Architecture Design Space Exploration

The designer can add, modify, and remove architecture components using a graphical user interface tool called *Processor Designer (ProDe)*. Each iteration of the processor can be evaluated by compiling the software of interest to the architecture using TCE's retargetable high-level language compiler and simulating the resulting parallel assembly code using an architecture simulator.

The simulator statistics give the runtime of the program, the utilization of different datapath components and indicating bottlenecks in the design. The processor simulator provides a compiled simulation engine for fast evaluation cycles, and a more accurate interpretive engine for software debugging which supports common software debugging features such as breakpoints.

The accuracy of the TCE processor simulator is at the instruction-set level, but gives enough information to drive the architecture design process due to the static TTA-based processor template. Because the instruction-set controls the fine details of the processor as data transfers, the simulation is closer to a more detailed RTL simulation, especially regarding the metrics of interest.

The architecture exploration cycle enables low effort evaluation of different custom operations. For a completely new processor operation, the designer describes the operation simulation behavior in C/C++ to OSAL, estimates its latency in instruction cycles when implemented in hardware, and adds the operation to one of the function units in the architecture. This way it is possible to see the effects of the custom hardware to the cycle count, before deciding whether to include it in the design or not. As the OSAL behavior description is C/C++, usually the designer can just copy-paste the piece of program code that should be replaced by the custom operation call to get it modeled.

8.4.2 Processor Hardware Generation

When a design point that fulfills the requirements is found or more accurate statistics of a design point is needed, the designer uses a tool called *Processor Generator (ProGe)* which produces a synthesizable RTL implementation of the processor. Thanks to the modular TTA template, the RTL generation is straightforward and reliable; ProGe collects component implementations from a set of HDBs, produces the interconnection network connecting them, and then generates an instruction decoder that expands the instructions to datapath control signals.

The designer has to implement only the new function units in VHDL or Verilog and add them to an HDB. Most of the more generic components in the new processor designs are found readily available in previously accumulated HDBs and only the special function units that implement application-specific custom operations need to be manually implemented. To assist in the implementation process, a tool is available to automatically validate the function unit implementation against its architecture simulation model.

The generated RTL is fed to a standard third party synthesis and simulation flow. This step generates more detailed statistics of the processor at hand, which can again drive further iterations in the architectural exploration process. The detailed statistics include timing details to verify if the target clock frequency is achieved, chip area to ensure the design is not too large and for low power designs, the power consumption when executing the applications of interest.

The detailed implementation level statistics map trivially back to the design actions at the architecture level. For example, the chip area can be reduced by removing architecture components. Similarly, adding more pipeline stages to complex function units, or reducing the connectivity of the interconnection network helps increasing the maximum clock frequency.

8.5 Automated Architecture Design

We think that the best processor designs should involve some human designer effort instead of attempting to completely automatize the process. After all, when describing a new architecture, what is actually being designed is a programmer–hardware interface, a user interface of a kind. This applies especially to designs that will be implemented with expensive ASIC processes, or are expected to be at least partially assembly programmed. Such designs are better to be iterated at least partially manually in order to produce designs that are more understandable and logical for programmers. We have found *semiautomated* design space exploration a good compromise in this matter; instead of aiming towards a fully automated processor design process, the designer uses assisting tools for iterating different *aspects* of the design at hand automatically.

One aspect of TTA designs that calls for automated architecture design support is the interconnection network; connectivity between components in larger TTA designs is hard to manage manually due to the huge space of options. Therefore, usually in the earlier phases of TCE design process the connectivity aspect is simply ignored and a fully connected interconnection matrix is used in order to evaluate the compute resource sufficiency. Only after the application of interest is saturated with enough register file and function unit resources, the connectivity is pruned.

A redundant fully connected interconnection network can spoil TTA designs that are usually otherwise very streamlined and energy efficient. In addition to increasing the critical path, worsening the power consumption and expanding the required chip area, the excessive connectivity results in bloated instruction words due to the large number of sources and destinations to encode. Loose instruction words reflect in larger instruction memory requirements and energy waste in instruction fetch and decode.

TCE provides multiple automated *design space exploration tools* for iterating the interconnection network. One of them, called *Bus Merger* [14], starts by creating a connectivity matrix similar to VLIW designs with full FU bypass network connectivity (see Fig. 8.5). This network is then gradually reduced by merging

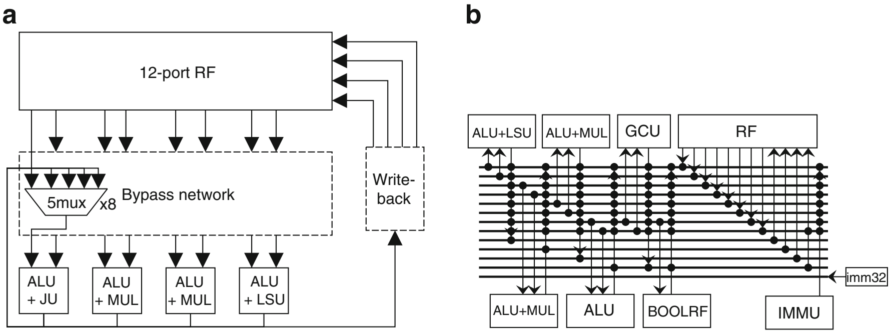


Fig. 8.5 (**a**) An example 4-issue VLIW datapath with a fully connected FU bypass network, and (**b**) the corresponding TTA architecture with equal connectivity

buses (producing a bus with a union of the connections) that are rarely used at the same time for data transports, on each iteration evaluating the program to the new architecture using the compiler and the simulator. Finally, the original overly complex VLIW-like register files are simplified, thanks to the reduced need for RF ports.

The greedy Bus Merger finds rather good application-specific ICs quickly for VLIW-style designs, but might not be efficient for smaller architectures with a small number of equally highly utilized buses. Another option provided by TCE for automated connectivity optimization is *Connection Sweeper*. As the name suggests, it works in a brute force fashion, sweeping the interconnection network in a trial-and-error fashion, removing connections and testing the effect to the cycle count, stopping when the performance degrades more than a given threshold. It first tries to remove RF connections as they often reside in the critical path.

Other automated design assisting exploration tools in TCE include *Simple IC Optimizer*, a connectivity optimizer that simply removes all unused connections, a useful tool for FPGA softcore designs; *Machine Minimizer*, a brute force tool that tries to remove components until the cycle count increases above a given threshold (*Machine Grower* is its counterpart) and *Cluster Generator*, that helps to produce clustered-VLIW style TTAs. The automated exploration tools can be called from the command line, enabling a design methodology where the machine is partially constructed from a starting point architecture using scripts or *makefiles*.

8.6 Programming Support

Optimizing the code manually for the designed processor using its assembly language might be a last step taken after the processor design has been frozen and the programmer wants to squeeze off the last loose execution cycles in the application of interest. During the design process, however, the use of assembly is not feasible due to the changing target; whenever the architecture is changed, the affected parts of the assembly code would need to be rewritten. This would make the fast iterative evaluation of different architecture parameters practically impossible, or at least heavily restrict the practical number of evaluated architectures.

Other motivations for using high-level programming languages or intermediate formats for application description is to hide the actual instruction-set (which is very low level in TTAs) from the programmer. This avoids constraining new designs with legacy backwards compatibility issues and the design process is simplified because instruction encoding specifics are pushed to the background.

In our experience, the benefits from using the assembly language in comparison to relying on an HLL compiler often stem from the inability of the compiler to extract adequate parallelism from the program description to utilize all the parallel processor resources, or from the inability to use complex custom operations automatically to accelerate the program. TCE attempts to alleviate this by taking advantage of, in addition to the usual C/C++ languages, the parallel OpenCL standard. The explicit and implicit parallelism of OpenCL C helps in the utilization issue while the automatically generated OpenCL vendor extensions of the TCE compiler can be used to execute custom operations manually from the source code [7].

Due to making the high-level programming of the designed processors a priority, a key tool in TCE is its retargetable software compiler, tcecc. The compiler uses the LLVM [10] project as a backbone and *pocl* [9] to provide OpenCL support. The frontend supports the ISO C99 standard with a few exceptions, most of the C++98 language constructs, and a subset of OpenCL C.

The main compilation phases of tcecc are shown in Fig. 8.6. Initially, LLVM's Clang frontend converts the source code into the LLVM's internal representation. After the frontend has compiled the source code to LLVM bytecode, the utility software libraries such as libc are linked in, producing a fully linked self-contained bytecode program. Then standard LLVM IR optimization passes are applied to the bytecode-level program, and due to it being fully linked, the whole-program optimizations can be applied aggressively. The optimized bytecode is then passed to the TCE retargetable code generation.

To make custom operation evaluation easier, the automatically retargeting back-end has the facilities to exploit custom operations automatically. Custom operations can be described in OSAL as data-flow graphs of more primitive operations, which the LLVM instruction selector automatically attempts to detect and replace in the program code.

Complex custom operations, such as those that implement long chains of primitive operations, or those that produce multiple results, often cannot be automatically found from intermediate codes produced from high-level language programs. For this, tcecc produces intrinsics that can be used manually by the programmer from the source code.

For example, this C code snippet calls a custom operation called "ADDSUB" that computes both the sum and the difference of its operands in parallel and produces them as two separate results:

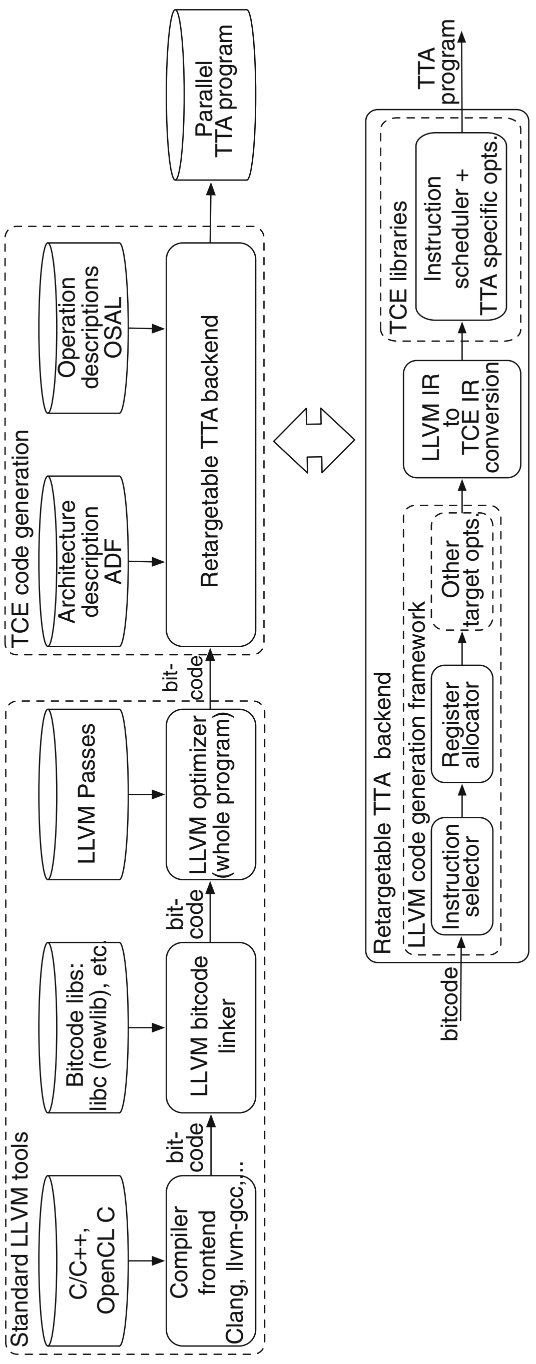


Fig. 8.6 tcecc compiler internals

```
int a, b, sum, diff;
...
_TCE_ADDSUB(a, b, sum, diff);
```

The actual processor described in ADF must then include at least one function unit that supports the operation, one of which is triggered by code generated by the compiler. If the programmer wants to restrict the operation to be triggered in a specific function unit (making the source code more target specific in the process), another intrinsics version can be used:

```
_TCEFU_ADDSUB("ALU1", a, b, sum, diff);
```

This instructs the compiler to generate code to call operation ADDSUB in a function unit referred to as ALU1 in the targeted ADF.

8.7 Layered Verification

Using a component library based processor design with automated RTL generation is helpful in reducing implementation and verification effort. As the processors can be composed of preverified function unit and register file implementations, the effort required for new designs should reduce gradually as the databases are augmented with new validated components. However, additional verification is still usually preferred to gain trust on the produced processor implementation.

The approach TCE takes to verifying the designs is shown in Fig. 8.7. It uses a layered top-down approach where each level in the implementation abstraction hierarchy is compared against the previous one. At the first level, the designer who uses a portable high-level language program as an application description can implement and verify the software functionality using a desktop environment and the work station's CPU. Printouts to standard output can be used to produce the initial "golden reference" data.

Each layer of implementation abstraction can be compared to the golden reference data by including a standard output instruction in the processor architecture, used to produce the output at the different levels. In RTL simulation, the function unit implementation uses the VHDL or Verilog file printout APIs, and at the FPGA prototyping stage it may write to a JTAG-UART console. Further verification can be done by comparing bus traces which contain values in each processor transport bus at each instruction cycle.

Finally, the core is integrated to a system level model. TCE provides facilities to help producing project files and other integration files to different FPGA flows. System level simulation is supported via bindings for SystemC. The bindings allow plugging TTA instruction accurate core simulation models to larger SystemC models to enable cycle accurate performance modeling and functional verification simulations. The following is an example of how to integrate a TTA core simulation model to a system simulation by connecting its clock and global lock signals:

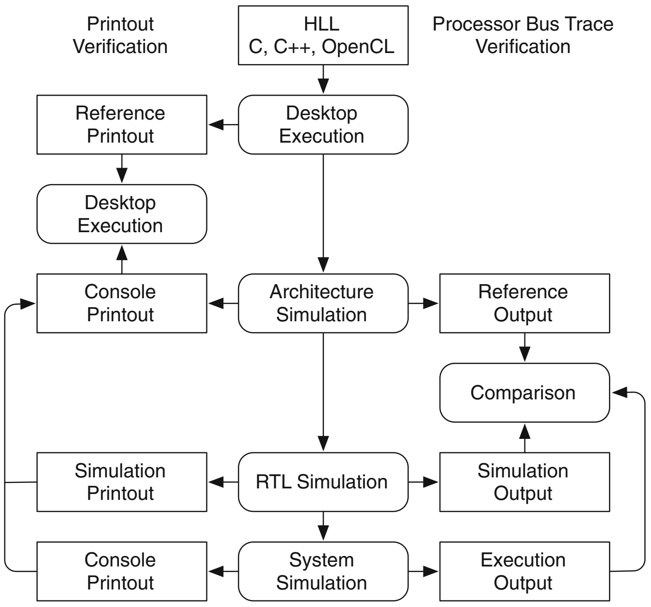


Fig. 8.7 Verification flow

```
...
#include <tce_systemc.hh>
...
int sc_main(int argc, char* argv[]) {
    ...
    TTACore tta("tta_core_name", "processor.adf", "program.tpef");
    tta.clock(clk.signal());
    tta.global_lock(glock);
    ...
}
```

The FPGA flow can produce the same standard output printouts using a development board with a JTAG interface or similar. When using the system level simulation, verification data is produced similarly to the architecture simulator.

Post-fabrication verification is performed using suites of test programs that stress the different aspects of the processor design. At this point, additional verification and debugging can be done by using an automatically generated debug interface which can be used to output verification data and single step the processor.

8.8 Conclusions

In this chapter, we described TCE, a processor customization toolset based on the exposed datapath transport triggered architecture. TCE provides tool support for iterative processor customization starting from high-level programming languages, producing synthesizable RTL implementations of the processors along with instruction parallel binary code.

TCE is available as a liberally licensed open source project and can be downloaded via the project web page [12]. The toolset is mature and it has been tested with various design cases over the past decades, and is evolving with new developments focusing on the compiler code generation quality and energy saving features, both in the produced hardware and those that can be achieved with the compiler.

References

1. Cilio, A., Schot, H., Janssen, J., Jääskeläinen, P.: Architecture definition file: processor architecture definition file format for a new TTA design framework (2014). http://tce.cs.tut.fi/specs/ADF.pdf
2. Corporaal, H.: Transport triggered architectures: design and evaluation. Ph.D. thesis, TU Delft (1995)
3. Corporaal, H.: Microprocessor Architectures: From VLIW to TTA. Wiley, Chichester (1997)
4. Corporaal, H., Mulder, H.: MOVE: a framework for high-performance processor design. In: Proceedings of ACM/IEEE Conference on Supercomputing, pp. 692–701 (1991). doi:http://doi.acm.org/10.1145/125826.126159
5. Dally, W., Balfour, J., Black-Shaffer, D., Chen, J., Harting, R., Parikh, V., Park, J., Sheffield, D.: Efficient embedded computing. Computer **41**, 27–32 (2008). doi:http://doi.ieeecomputersociety.org/10.1109/MC.2008.224
6. He, Y., She, D., Mesman, B., Corporaal, H.: MOVE-Pro: A low power and high code density TTA architecture. In: Proceedings of International Conference on Embedded Computer Systems: Architectures, MOdeling and Simulation (SAMOS), pp. 294–301 (2011). doi:10.1109/SAMOS.2011.6045474
7. Jääskeläinen, P., de La Lama, C., Huerta, P., Takala, J.: OpenCL-based design methodology for application-specific processors. Trans. HiPEAC **5** (2011). Available online
8. Jääskeläinen, P., Kultala, H., Viitanen, T., Takala, J.: Code density and energy efficiency of exposed datapath architectures. J. Signal Process. Syst. 1–16 (2014). doi:10.1007/s11265-014-0924-x. http://dx.doi.org/10.1007/s11265-014-0924-x
9. Jääskeläinen, P., de La Lama, C.S., Schnetter, E., Raiskila, K., Takala, J., Berg, H.: pocl: a performance-portable OpenCL implementation. Int. J. Parallel Prog. 1–34 (2014). doi:10.1007/s10766-014-0320-y. http://dx.doi.org/10.1007/s10766-014-0320-y
10. Lattner, C., Adve, V.: LLVM: a compilation framework for lifelong program analysis & transformation. In: Proceedings of the International Symposium on Code Generation Optimization, pp. 75–87 (2004)
11. Lipovski, G.: The architecture of a simple, effective control processor. In: 2nd Euromicro Symposium on Microprocessing and Microprogramming, pp. 7–19 (1976)
12. TCE: TTA-based co-design environment (2015). http://tce.cs.tut.fi

13. Thuresson, M., Själander, M., Björk, M., Svensson, L., Larsson-Edefors, P., Stenström, P.: FlexCore: Utilizing exposed datapath control for efficient computing. In: Proceedings of International Conference on Embedded Computer Systems: Architectures, MOdeling and Simulation (SAMOS), pp. 18–25 (2007). doi:10.1109/ICSAMOS.2007.4285729
14. Viitanen, T., Kultala, H., Jääskeläinen, P., Takala, J.: Heuristics for greedy transport triggered architecture interconnect exploration. In: Proceedings of the 2014 International Conference on Compilers, Architecture and Synthesis for Embedded Systems, CASES '14, pp. 2:1–2:7. ACM, New York, NY (2014). doi:10.1145/2656106.2656123. http://doi.acm.org/10.1145/2656106.2656123

Chapter 9
FPGA-Based Cognitive Radio Platform with Reconfigurable Front-End and Antenna

Aitor Arriola, Pedro Manuel Rodríguez, Raúl Torrego, Félix Casado, Zaloa Fernández, Mikel Mendicute, Eñaut Muxika, Juan Ignacio Sancho, and Iñaki Val

9.1 Introduction

In the last few decades there has been an increasing interest towards replacing wired communications by their wireless counterparts, mainly due to the various advantages the latter offer, such as cost reduction, ease of maintenance, and the possibility of reaching areas where conventional wiring is too difficult or costly [1–3]. This is especially important in industrial environments (e.g. factories, airplanes, or trains), where the advantages presented by wireless solutions have a large economical impact.

However, industrial environments present significant challenges for wireless communications [4, 5]: signal shadowing, multipath fading, and the presence of interferences are major issues that need to be considered carefully. In this context, Software Defined Radios (SDR) and Cognitive Radios (CR) can help to overcome these limitations, and appear as promising opportunities for the successful deployment of wireless communication in industrial environments [6, 7].

In this chapter a Field Programmable Gate Array (FPGA)-based SDR platform with cognitive features is presented. This platform relies on a hardware that is reconfigurable at all levels (i.e. baseband processing, Radio-Frequency (RF) front-

A. Arriola (✉) • P.M. Rodríguez • R. Torrego • F. Casado • Z. Fernández • I. Val
ICT Department, IK4-IKERLAN, Arrasate-Mondragón, Spain
e-mail: aarriola@ikerlan.es

M. Mendicute • E. Muxika
Electronics and Computing Department, University of Mondragón, Arrasate-Mondragón, Spain
e-mail: mmendikute@mondragon.edu

J.I. Sancho
Department of Electronics and Communications, CEIT and Tecnun - University of Navarra, San Sebastián, Spain
e-mail: isancho@ceit.es

W. Hussain et al. (eds.), *Computing Platforms for Software-Defined Radio*,
DOI 10.1007/978-3-319-49679-5_9

end, and antenna), allowing the operation in different frequency bands; this is an interesting feature in order to fight undesired effects such as multipath fading or interferences. Based on this hardware, an FGPA-based RF-to-Ethernet bridge has been implemented, which is able to switch between the Industrial, Scientific, and Medical (ISM) bands of 868 MHz and 2.45 GHz; a reconfigurable antenna has also been integrated in this platform, which is optimized for direct placement on metallic surfaces. The rationale behind these selections has been the following one:

- *FPGA-based*: there are several implementation possibilities for SDRs, from classical Digital Signal Processors (DSPs) or general-purpose processors, to dedicated multi-core architectures [8]. Among these, the use of FPGAs and dynamic partial reconfiguration fits perfectly with the flexibility and high-performance computing demanded by SDRs.
- *RF-to-Ethernet Bridge*: an RF extension to Ethernet has been chosen as an application example, as Ethernet and its industrial variations such as Industrial Ethernet or Ethernet for Control Automation Technology (EtherCAT) are some of the most used standards for wired industrial-control applications [9].
- *868 MHz and 2.45 GHz ISM bands*: these are the frequency bands where most industrial wireless communications are focused nowadays [4]. The solutions in each of these bands have some particular features: the ISM band of 868 MHz is mainly operated by proprietary protocols and narrowband communication systems (i.e. systems with high sensitivity and high interference rejection), which take advantage of the smaller path-loss in this band; on the other hand, the systems operating in the ISM band of 2.45 GHz benefit from a huge market of low-cost Radio-Frequency Integrated Circuits (RF-ICs) and from the existence of several standard protocol stacks tailored for industrial environments (e.g. Wireless Highway Addressable Remote Transducer Protocol (WirelessHART) [10], International Society of Automation 100.11a (ISA100.11a) [11], and Wireless networks for Industrial Automation—Process Automation (WIA-PA)). The main drawback of the 2.45 GHz band is that it is very crowded as a consequence of its popularity, what results in high interference from other users. Therefore, an RF platform which can be reconfigured between these two bands increases the availability of the wireless link.
- *Antenna for metallic environments*: in industrial environments, antennas are prone to have metallic objects in their vicinity. As shown in [12], monopole-like antennas exhibit significant variations in their input impedance and radiation efficiency when placed near metallic objects. Hence, the reconfigurable antenna designed for this platform has been based on microstrip topology, which shows a great robustness against any object placed under the antenna [13].

The rest of the chapter is organized as follows: Sect. 9.2 introduces the concepts of SDR and Cognitive Radio, and their particularities for FPGA implementation; Sect. 9.3 describes the cognitive platform, including design and implementation details of its main building blocks. Finally, measurement results and concluding remarks are presented in Sects. 9.4 and 9.5, respectively.

9.2 FPGA-Based Architectures for Cognitive and Software Defined Radio

The term “Software Radio” was first used by Joseph Mitola to refer to reconfigurable or reprogrammable radios that can support different functionalities at different times by changing their configuration in software [14]. This definition has, however, endured different interpretations since then, leading the term *Software Defined Radios* to be used for a wide spectrum of implementations, ranging from pure software implementations on general-purpose processors to FPGA-based implementations. Regarding reconfiguration capabilities, SDR covers minimal changes, where single parameters like filter coefficients or amplifier gains are updated, as well as coarse reconfigurations where a whole new communication standard is applied [i.e. reconfiguration from WiFi to WiMAX (Worldwide Interoperability for Microwave Access)].

Moreover, Software Defined Radios are the base for other novel architectures for communication systems, like Adaptive Radios, Cognitive Radios, or Intelligent Radios. Figure 9.1 shows the evolution between these architectures. *Adaptive Radios* monitor their own performance and are able to change some of their characteristics using SDR, in order to improve their performance without the user’s intervention. *Cognitive Radios* behave in a similar way but, apart from monitoring their own performance, they are also aware of the characteristics of the RF environment and change their parameters accordingly. One of the most typical characteristics measured by Cognitive Radios is the availability of the RF spectrum: using spectrum-sensing algorithms, Cognitive Radios can detect which wireless channels are available and change their transmission frequency to them in order to improve the quality of the communication link. This process is also known as Dynamic Spectrum Access (DSA). Finally, *Intelligent Radios* are an evolution of Cognitive Radios with an ability to learn; this means that Intelligent Radios are not only aware of their RF environment, but also of their previous decisions.

Regarding the implementation of SDR and CR on FPGAs, dynamic partial reconfiguration appears as a very convenient technique, as it enables an FPGA to

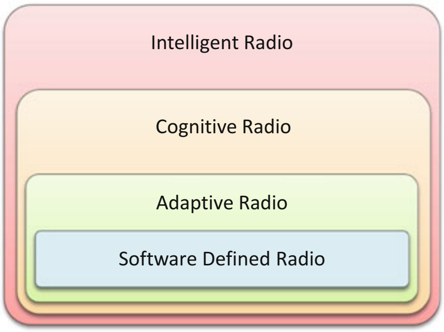


Fig. 9.1 Evolution of radio communication technologies

change the functionality of part of the chip while the rest of it continues working uninterrupted [15]. This characteristic comes from the use of Static Random Access Memory (SRAM) technology for the configuration memory on the FPGA, what allows runtime download of new configuration data. One of the key parameters to consider in partial reconfiguration is the reconfiguration time; during this time the reconfigurable area is not functional, so it must be carefully considered.

On the other hand, the use of rapid prototyping tools is highly advisable for the design and implementation of SDR signal-processing functions on FPGAs [16]. These tools enable a model-based design, therefore allowing the designer to program the required algorithms in a graphical way, generating synthesizable code automatically, as well as launching functional simulations in the early design steps leading to huge time savings. For the platform described in this chapter, *Xilinx's System Generator for DSP* tool has been used [17].

9.3 Platform Description

The cognitive SDR platform consists of two FPGA-based nodes, being each of them able to carry out both the Ethernet-to-RF and RF-to-Ethernet conversion. The platform is reconfigurable at the three main parts that make up each of the nodes: a Virtex 6 FPGA in charge of baseband processing, an RF front-end, and an antenna.

The system works in a half duplex mode; this means that each time one of the nodes acts as an Ethernet-to-RF bridge, while the other one acts as an RF-to-Ethernet bridge. In order to achieve the reliability needed in industrial communications scenarios, the system implements a DSA algorithm; taking advantage of the capability of the system for transmitting in two ISM bands (868 MHz and 2.45 GHz), the DSA algorithm selects the most suitable frequency band in order to avoid interference and, as a consequence, the platform is fully reconfigured. The platform also allows small in-band frequency changes (<10 MHz) in the transmission frequency by reconfiguring the digital oscillator that generates the Intermediate Frequency (IF). This reconfiguration of the oscillator is carried out using dynamic partial reconfiguration of the FPGA. A block diagram of the whole platform is detailed in Fig. 9.2.

The signal-processing tasks of the platform have been hardware-implemented on an FPGA device, thanks to the high performance and flexibility offered. *System Generator for DSP*, Xilinx's rapid prototyping tool, has been used for designing, implementing, and testing the baseband data-processing algorithms. The transmitter and the receiver parts use a MicroBlaze processor in charge of controlling different parts of the architecture: FPGA dynamic partial reconfiguration, programming of the front-end, and control of the antenna.

With regard to dynamic partial reconfiguration, in this case it has been used to reconfigure the local oscillators of the channel-selection downconverter, and produce small in-band frequency changes. However, it could also be used to change any other parameter in the baseband signal-processing tasks, e.g., modulation, data

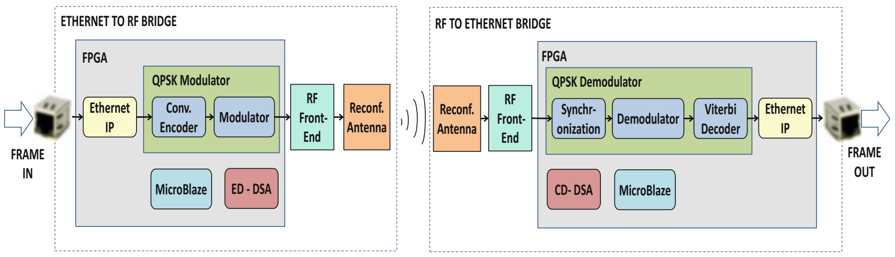


Fig. 9.2 Block diagram of the Ethernet-to-RF and RF-to-Ethernet bridges ©Springer [18]

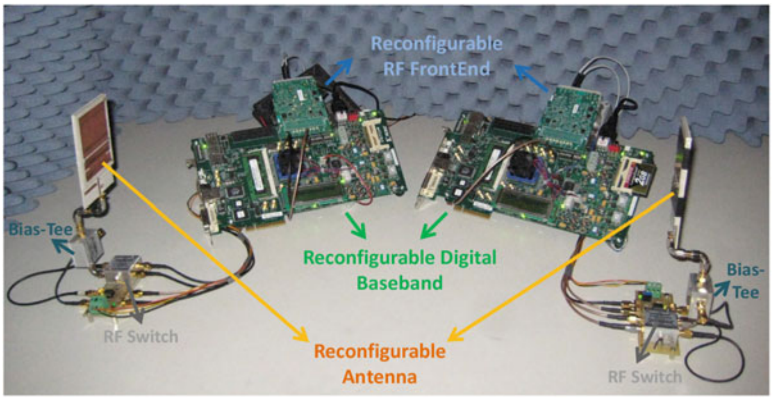


Fig. 9.3 Prototypes of Ethernet-to-RF and RF-to-Ethernet bridges ©Springer [19]

rate, etc. The MicroBlaze processor has access to both the Internal Configuration Access Port (ICAP) of the FPGA and the external Flash memory where the different partial bitstreams are stored. When an in-band frequency change is required, the MicroBlaze processor will read the corresponding partial bitstream from the Flash memory and download it to the configuration memory of the FPGA through the ICAP port.

A prototype of the platform is shown in Fig. 9.3., which depicts the FPGA boards in charge of the baseband processing, the reconfigurable RF boards, and the reconfigurable antennas with their associated circuitry.

The following sections will describe in detail the building blocks shown in Fig. 9.3.

9.3.1 Ethernet-to-RF FPGA

The FPGA acting as Ethernet-to-RF bridge implements three main functions: Ethernet frame-processing, Quadrature Phase-Shift Keying (QPSK) modulation, and Energy-Detection (ED)-based DSA.

9.3.1.1 Ethernet Frame-Processing IP Core

The Ethernet frame-processing Intellectual Property (IP) core is in charge of receiving the frames from the Ethernet physical layer (PHY) using a Media Independent Interface (MII), reconditioning them, and routing them to the QPSK modulator. It implements an Ethernet Medium Access Control (MAC) layer, two dual-port RAMs (implemented using FPGA Block-RAMs (BRAMs) for improved speed), and a control using a Finite State Machine (FSM). A high-level hardware diagram of this IP core is shown in Fig. 9.4.

The Ethernet MAC block manages the communication between the PHY and the user logic. It receives the frames from an MII interface and translates them into a byte-to-byte format. Due to the different data rates of the incoming information and the modulator, it is not possible to connect them directly; the dual-port RAMs enable this connection by allowing asynchronous and independent read/write operations in each of their ports. The control of these operations is carried out by the FSM.

It should be noted that only the Ethernet-to-modulator transition has been mentioned, but the Ethernet frame-processing IP core is bidirectional (therefore, a single Ethernet-processing IP core has been implemented per node). In the node acting as RF-to-Ethernet bridge, this block is in charge of properly encapsulating the demodulated data into a frame and delivering it to the Ethernet PHY.

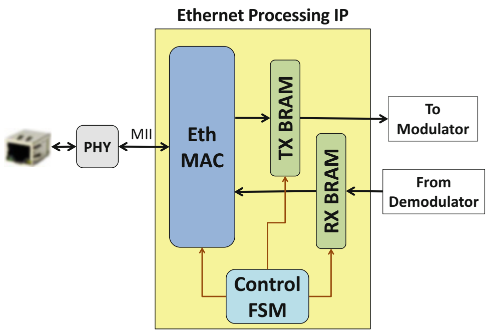


Fig. 9.4 High-level hardware diagram of the Ethernet-processing IP core ©Springer [18]

9.3.1.2 QPSK Modulator

As shown in Fig. 9.2, the QPSK modulator is in charge of two main functions: data encoding and modulation. Data encoding increases the reliability of the communication using a convolutional encoder with rate ½. The modulator receives the data stream from the encoder and adds a 256-symbol preamble for start-of-frame detection, frequency, and phase-recovery in the receiver. A 32-symbol Start Frame Delimiter (SFD) is added as well. On the other hand, as Ethernet frames may have different lengths, this value is measured and included in two bytes (8 QPSK symbols) before the payload. The complete frame to be transmitted over the air is represented in Fig. 9.5.

Afterwards, data is mapped onto the QPSK constellation and the signal is oversampled by a factor of 4. The reason for this is that at the other end of the wireless link, the receiver oversamples the signal by the same factor of 4 in order to improve the symbol timing estimation [20]. Therefore, the complete modem should maintain the same oversampling factor for transmission and reception, taking into account that they use a common clock frequency.

Finally, the signal is filtered with a raised-cosine filter (*roll-off factor* = 0.35) in order to optimize the spectrum efficiency prior to being sent to the reconfigurable front-end.

9.3.1.3 Energy Detection for Spectrum Sensing

In parallel with the frame-processing IP core and the QPSK modulator, a two-stage ED-based DSA algorithm is running [21]. This algorithm looks for available frequencies within the spectrum based on energy presence and selects the transmission frequency accordingly. Hence, the goal of the spectrum-sensing algorithm is to decide between two hypotheses: H_0 for an available channel and H_1 for a channel occupied by another signal:

$$\begin{aligned} H_0 &: y[n] = w[n] \\ H_1 &: y[n] = s[n] + w[n] \end{aligned} \tag{9.1}$$

where $y[n]$ is the received signal, $w[n]$ is white Gaussian noise, and $s[n]$ is the external signal that may occupy the channel. The decision is taken according to a test statistic, which in the case of the ED detector corresponds to the energy of the received signal.

The ED is based on two stages, which are shown in Fig. 9.6. In the first stage the detector analyzes the energy of the received signal:

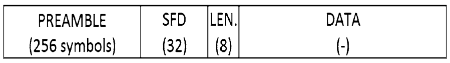


Fig. 9.5 QPSK modulator frame ©Springer [18]

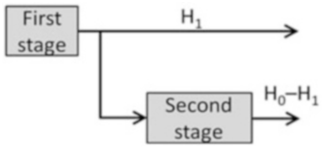


Fig. 9.6 Two-stage ED scheme ©Springer [18]

$$\sum |y[n]|^2 \gtrless \lambda_{\mathrm{ED}} \tag{9.2}$$

If the energy is higher than the threshold (λ_{ED}), the detector decides that the sensing is under the hypothesis H_1, executing the second stage otherwise. The second stage is an ED with two thresholds; these thresholds define three regions: the two hypotheses H_0 and H_1, and an uncertainty zone. The Ethernet-to-RF bridge will transmit in the assessed channel as long as it is available (i.e. the outcome of the detector is H_0); if the analysis results in hypothesis H_1 or the uncertainty zone, the Ethernet-to-RF will continue looking for an available channel. On the other hand, it must be noted that the performance of an ED is reduced as noise level grows; therefore, an estimator of noise power has also been implemented using a recursive averaging algorithm [20].

9.3.2 RF-to-Ethernet FPGA

Similarly, the FPGA that carries out the inverse operation, i.e., the RF-to-Ethernet conversion, also implements three functions: DSA based on a Cyclostationary Detector (CD), QPSK demodulation, and Ethernet frame processing.

9.3.2.1 Cyclostationary Detection for Spectrum Sensing

The RF-to-Ethernet bridge has no information about the frequency channel that the Ethernet-to-RF bridge is using. For this reason, the DSA technique must distinguish between the desired signal and possible interference sources. To achieve this purpose, a CD-based DSA technique has been implemented, which allows the RF-to-Ethernet bridge to tune to the correct frequency channel. This kind of detector has been chosen instead of a matched filter-based detector, because it does not require any synchronization; it basically analyzes the cyclostationary features of the signal to distinguish the desired transmission.

Modulated signals are generally cyclostationary (i.e. their autocorrelation is periodic). This is due to cyclic prefix, synchronization, preambles, modulation, etc. [22]. Hence, the autocorrelation of a cyclostationary signal s can be represented as a function of its Discrete Fourier Transform (DFT):

$$R_s(n, n+\tau) = \sum_{\alpha=0}^{\infty} R_s^{\alpha}(\tau) e^{j2\pi\alpha n} \tag{9.3}$$

where R_s^{α}, called *Cyclic Autocorrelation*, are the coefficients of the DFT, defined as:

$$R_s^{\alpha} = \frac{1}{N} \sum_{n=0}^{N-1} s(n) s^*(n-\tau) e^{-j\frac{2\pi\alpha n}{N}} \tag{9.4}$$

where α is the cyclic frequency. Considering x as the received signal, R_x^{α} will be zero unless α is equal to a period of the autocorrelation function of s. The test statistic in a CD is the value of R_x^{α}:

$$R_x^{\alpha} \gtrless \lambda_{\mathrm{CD}} \tag{9.5}$$

The value of α has been fixed to a period of the autocorrelation function of the desired signal s. The value of the cyclic autocorrelation for this α allows distinguishing between the QPSK signal transmitted by the Ethernet-to-RF bridge and a possible interferer.

Regarding implementation, Shift Register Look-Up Tables (SRL) and a COordinate Rotation DIgital Computer (CORDIC) block are in charge of calculating the cyclic autocorrelation [23]. The SRLs are in charge of delaying the signal to compute the autocorrelation; then, the CORDIC block is used to generate the complex exponential.

9.3.2.2 QPSK Demodulator

Once the system is configured at the correct frequency channel, baseband Inphase and Quadrature (IQ) data coming from the RF front-end is processed by the QPSK demodulator (see Fig. 9.2). A detailed view of the signal paths inside the demodulator is shown in Fig. 9.7:

1. The system detects a transmitted message by means of a preamble detector, which triggers the feed-forward timing-recovery algorithm based on Maximum Likelihood estimation [20].
2. A Farrow-structure interpolator filter is used to get optimal symbol values.
3. A data-aided coarse frequency estimator is used to compensate the carrier frequency drift [24]. Joint phase and SFD estimation [25] are done first detecting where the frame starts, and then the phase offset is estimated based on the preamble training data.
4. A closed loop PLL circuit estimates and corrects the residual frequency offset using the received data, by means of decision-directed criteria.
5. Once the data symbols are correctly aligned, the payload data is decoded using the Viterbi algorithm and routed to the Ethernet frame-processing IP core.

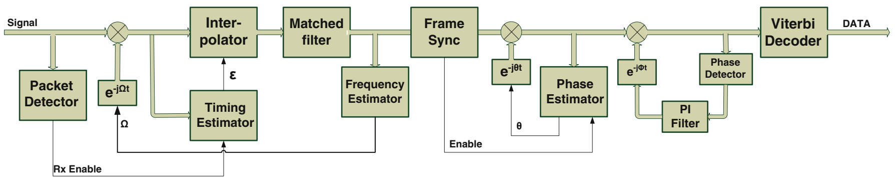


Fig. 9.7 Signal paths inside the demodulator

The demodulator also includes an Automatic Gain Controller (AGC). This block uses the preamble to compute the power of the incoming signal, and sets the analog amplifiers of the RF front-end accordingly, so that the received signal at the Analog-to-Digital Converters (ADCs) reaches 90 % of full scale. Once a frame has been properly demodulated, the AGC sets back the amplifiers to their maximum gain in order to obtain the highest possible sensitivity. The dynamic range of the receiver is specified from −95 to −30 dBm.

9.3.2.3 Ethernet Frame-Processing IP Core

In the node configured as RF-to-Ethernet bridge, the Ethernet frame-processing IP core is in charge of managing the communications between the demodulator and the Ethernet PHY. This IP core encapsulates the received data into an Ethernet frame, adjusts the data rates with the double-port RAMs, and delivers the frame to the Ethernet PHY via a MAC IP core. As mentioned earlier, a single Ethernet-processing IP core is present in each node, which is able to manage a bidirectional communication.

9.3.3 RF Front-End and Antenna

A *Nutaq Radio420S* module (depicted in Fig. 9.3) has been used as reconfigurable RF front-end [26]. This module operates between 300 MHz and 3 GHz with a bandwidth between 1.5 and 28 MHz. Its transmitted power can be adjusted between −21.5 and 10 dBm, and its sensitivity ranges between −90 dBm (0.3–1.5 GHz) and −103 dBm (1.5–3 GHz). It should be noted that this module has separate RF transmission and reception ports; hence, in order to use a single antenna, a Mini-Circuits ZYSW-2-50DR RF-switch has been used (see Fig. 9.8). This switch has a wideband operation (Direct Current (DC) to 5 GHz) and is controlled with a Transistor–Transistor Logic (TTL) signal coming from the main FPGA.

Regarding the antenna, a frequency-reconfigurable topology has been used which operates in the two selected ISM bands. Reconfigurable antennas are an optimal solution to cover different bands as, depending on their reconfiguration state (usually controlled by switching elements), they can cover the desired band and reject the

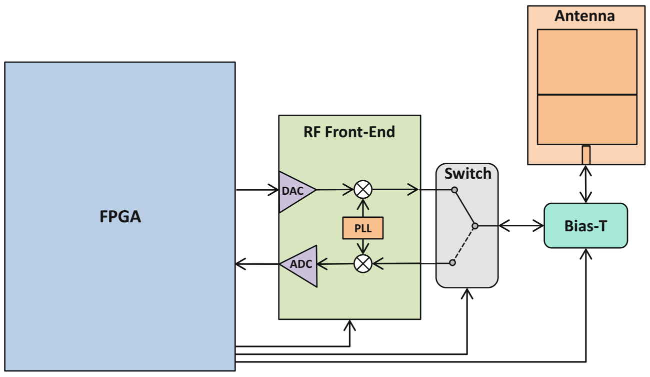


Fig. 9.8 RF front-end connections ©Springer [19]

undesired one [27]. There are other antenna options to cover different bands, such as Ultra-Wideband (UWB) antennas [28], which cover a very large spectrum, or multiband antennas [29] (e.g. those used in smartphones), which cover different bands simultaneously. However, when using UWB or multiband antennas, all bands are covered at the same time; this means that if any interference is detected, even though the whole system changes its operating frequency, the interference is still received and the front-end has to be able to filter it. Reconfigurable antennas, on the contrary, provide this extra filtering within the antenna structure itself.

In Fig. 9.9, the topology of the proposed reconfigurable antenna is shown. An electromagnetically fed microstrip antenna has been selected. The microstrip topology shows great robustness against objects placed under the antenna, such as metal structures, what makes it suitable for industrial environments. On the other hand, using an electromagnetic feed allows the antenna designer to use thicker substrates for the microstrip antenna, what improves significantly the bandwidth and efficiency of the antenna. However, with this kind of feed there is no physical connection between the feeding line and the radiating element, what adds extra complexity for providing the DC supply to the RF switches placed on the radiator.

For the present antenna two PIN (P-type, Intrinsic, N-type) diodes (BAP55LX model by NXP [30]) have been used as switching elements. These elements provide high switching speed and low actuation voltage, what makes them suitable for the integration in high-data rate and low-power reconfigurable platforms. RF Micro-Electro-Mechanical Systems (MEMS) can also be used as switching elements in antennas [31], but they have lower switching speeds and require higher actuation voltages.

The PIN diodes are placed on the antenna and are fed by DC lines. It should be noted that both the RF signal and the DC power are provided by the FPGA board, and combined with a Bias-T circuit (Mini-Circuits ZFBT-4R2G-FT), as depicted in Fig. 9.8.

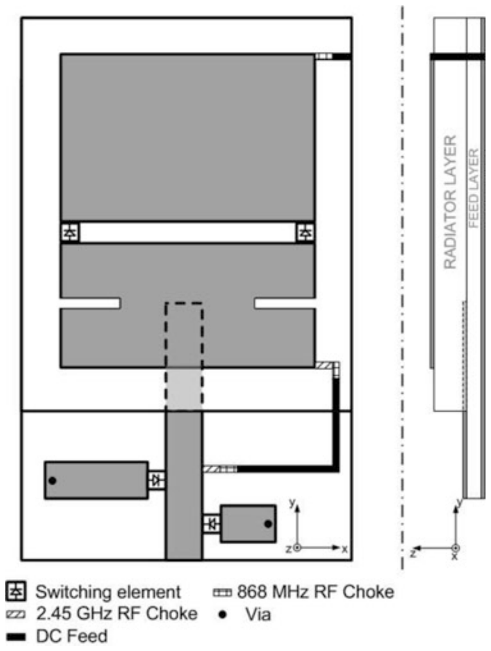


Fig. 9.9 Topology of proposed reconfigurable antenna and matching network ©Springer [19]

Table 9.1 States of PIN diodes

	ISM 868 MHz	ISM 2.45 GHz
PIN diodes on antenna	ON	OFF
PIN diodes on RMN	OFF	ON

The operation of the PIN diodes placed on the radiator element is as follows: in order to work at the 2.45 GHz ISM band, the diodes are put in OFF state (ideally behaving as an open) and the resulting antenna is a microstrip patch antenna with a parasitic element at the other side of the diodes; to operate at the 868 MHz ISM band, the diodes are put in ON state (ideally behaving as a short) and the resulting antenna is the original patch plus an extension which shifts the frequency downwards. On the other hand, in order to ensure a proper matching in both ISM bands, a Reconfigurable Matching Network (RMN) has been designed and added to the antenna. This RMN (shown in Fig. 9.9) is a microstrip stub-based network, and its reconfiguration is also done through PIN Diodes. Depending on the frequency of operation, the FPGA provides a DC voltage to the PIN diodes that control the antenna and the RMN. Table 9.1 shows the state of the PIN diodes depending on the selected ISM band.

9.4 Measurements

This section details the measurements from the RF-to-Ethernet bridge. It includes time, system performance, and antenna measurements.

9.4.1 Time Measurements

System latency and frequency-reconfiguration delay have been measured in order to quantify the time-domain performance of the RF-to-Ethernet bridge. System latency includes the end-to-end latency of a digital frame (once the platform is configured on a specific frequency), while the frequency-reconfiguration delay accounts for the time needed to change the operating frequency of the platform. The obtained results for both parameters are presented in the following subsections.

9.4.1.1 Latency

Table 9.2 shows the latency introduced by the different blocks of the system, as well as the overall latency. These measurements have been done with *ChipScope*, Xilinx's logic analyzer. It should be noted that air propagation delay, although negligible, should also be added to the measured times in order to obtain an exact latency. In this case the boards were a few meters apart, so the air propagation delay is several orders of magnitude below the latency of the different blocks of the system.

It can be observed that due to the double-port BRAMs needed to adapt the data rate in the Ethernet-to-FPGA and FPGA-to-Ethernet transitions, the latency is also dependent on frame length. Besides, these transitions are the ones that introduce the longest latencies in the system. Due to the system topology and working frequency of the selected FPGA, it is not possible to remove these transitions and their associated latencies.

Table 9.2 Latency measurements

	System latency (μs)								
Frame length (bytes)	Eth MAC RX	Eth BRAM RX	Mod.	Front-end TX	Front-end RX	Demod.	Eth BRAM TX	Eth MAC TX	Total
32	1.52	5.20	24.00	0.10	0.50	67.00	42.80	0.70	141.82
128	1.52	11.80	24.00	0.10	0.50	67.00	163.00	0.70	268.62
512	1.52	42.50	24.00	0.10	0.50	67.00	643.00	0.70	779.32
1024	1.52	83.80	24.00	0.10	0.50	67.00	1280.00	0.70	1457.62
1356	1.52	110.00	24.00	0.10	0.50	67.00	1700.00	0.70	1903.82

Table 9.3 Frequency-reconfiguration delays

		Front-end	
FPGA	Antenna	Programming and calibration	Physical change
94 μs	53 μs	1.83 ms	12 ns

9.4.1.2 Frequency-Reconfiguration Delays

When performing frequency reconfiguration, two different cases can be distinguished: the in-band channel change (i.e. within the ISM band) and the complete band change, i.e., from 868.0 MHz to 2.45 GHz, or vice versa. The first case requires only the partial reconfiguration of the FPGA, while the RF front-end and the antenna remain unaltered. The second case, on the other hand, requires reconfiguration at all levels of the platform, i.e., FPGA, front-end, and antenna.

Frequency-reconfiguration delays have been measured and they are reported in Table 9.3. The measurement represents the time from the moment in which the MicroBlaze processor generates the reconfiguration order to the moment in which the physical frequency change happens.

A few comments on these results:

1. In order to reconfigure the antenna it is necessary to change its DC polarization. This change is achieved through a General-Purpose Input/Output (GPIO) from the MicroBlaze processor and glue logic. The presented antenna reconfiguration time measures the delay of the complete process.
2. In order to reconfigure the front-end, the MicroBlaze processor first recalculates several configuration parameters, e.g., Phase-Locked Loop (PLL) parameters, amplifier gains, etc., using the new frequency as input parameter; then it programs these parameters into the front-end through a Serial Peripheral Interface (SPI) interface, and finally commands the front-end for calibrations. When all this process is completed, the physical frequency change takes place.
3. In order to carry out a transmission-frequency change it is necessary to reconfigure both the RF front-end and the antenna. These two operations are performed in parallel. Therefore, the worst case will indicate the time during which the system is not available.

9.4.2 System Performance

Several parameters that account for the performance of the RF-to-Ethernet bridge are detailed in this section, such as resource utilization within the FPGA, effective data rate, and Bit/Packet Error Rate curves.

Table 9.4 FPGA resource utilization summary

	SLICES	LUT	Flip-Flop	BRAM	DSP48
Ethernet IP	394	866	790	2	0
QPSK modulator	554	250	1197	4	168
QPSK demodulator	9684	30,582	16,691	14	259
Energy detector	303	503	425	1	40
Cyclostationary detector	10,325	26,125	30,650	0	525
MicroBlaze	11,321	24,551	22,800	78	19
Total	32,581	82,877	72,553	99	1011

9.4.2.1 FPGA Resource Utilization

Table 9.4 shows the resource utilization within the FPGA. Assuming the number of SLICES as the most representative value of the size of the design, each of the nodes in the Ethernet-to-RF bridge fits correctly into the XC6VLX240T Virtex 6 FPGA, occupying 32,581 SLICES out of the 37,680 available SLICES. It can be observed that the most resource-hungry blocks of the system are the QPSK demodulator, the Cyclostationary Detector, and the MicroBlaze processor. The first two blocks are related to the receiver, which has inherently complex signal-processing algorithms. Regarding the MicroBlaze processor, in the present application it is only used to manage the dynamic partial reconfiguration process, control the reconfigurable front-end, and execute some minor control tasks; however, it has enough processing power to implement more complex functions in the future that justify its resource utilization, e.g., a MAC layer for multi-point wireless networks.

9.4.2.2 Effective Data Rate

The baseband data-processing algorithms implemented in the FPGA and the IQ ADC/DAC converters in the adjustable RF front-end are both clocked at 25.6 MHz. An oversampling rate of 4 has been used in the system, what leads to a 12.8 Mbps raw data rate in the wireless link. However, the inclusion of a ½ rate convolutional encoder to increase the reliability of the communication reduces the effective data rate of the system by half. Besides, the 256-symbol preamble and 32-symbol SFD included in each of the frames also degrade this effective data rate. Moreover, taking into account that the length of these two sections of the frame is fixed, the degradation will be more significant for small frames. Figure 9.10 shows the effective data rate of the system versus frame length. It can be observed that the effective data rate suffers some degradation from the raw 12.8 Mbps. However, considering that typical data-update periods in industrial wireless sensor and actuator networks are between 10 and 500 ms [32], and considering also that these applications transmit small amounts of data, the achieved data rate is suitable for this type of industrial wireless networks.

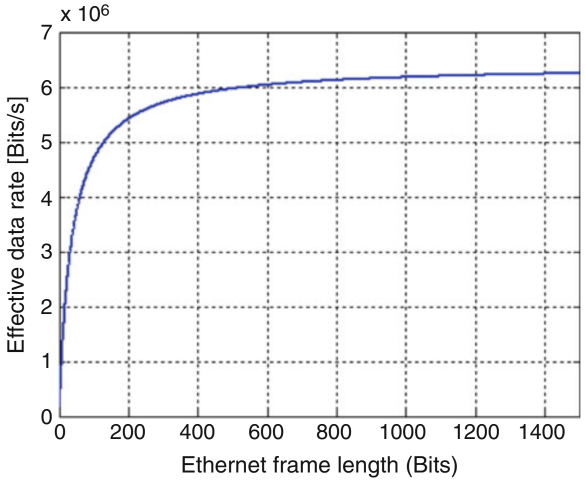


Fig. 9.10 Effective data rate vs. Ethernet frame length ©Springer [18]

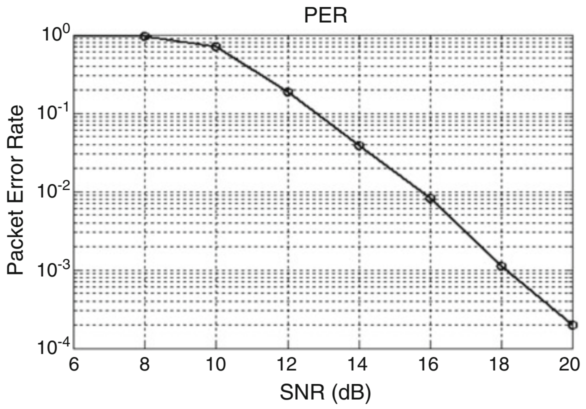


Fig. 9.11 AWGN packet error rate ©Springer [18]

9.4.2.3 Bit Error Rate/Packet Error Rate Performance Curves

In order to evaluate the performance of the receiver, Packet Error Rate (PER) and Bit Error Rate (BER) values have been measured over a range of Signal-to-Noise Ratios (SNR). For this purpose, an RF channel emulator has been used which tracks the transmitted power and adds noise to the signal based on its bandwidth. 32-Byte data frames have been used in each test [33].

The resulting PER performance for an Additive White Gaussian Noise (AWGN) channel and several SNR values is shown in Fig. 9.11 and BER values are depicted in Fig. 9.12. A PER value of 0.01 (1 %) is obtained for an SNR of 16 dB. This result indicates that the platform is suitable as a proof-of-concept for reconfigurable systems, but its baseband performance can be further optimized.

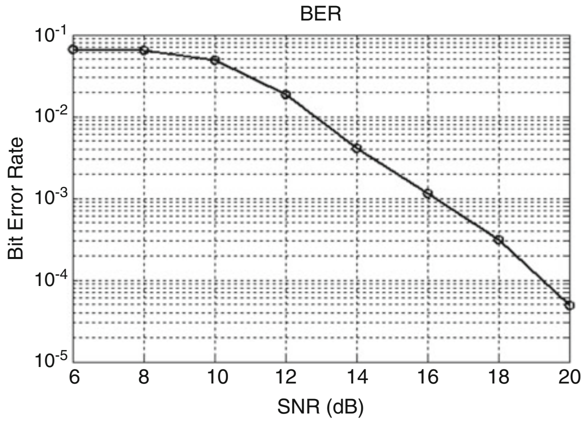


Fig. 9.12 AWGN bit error rate ©Springer [18]

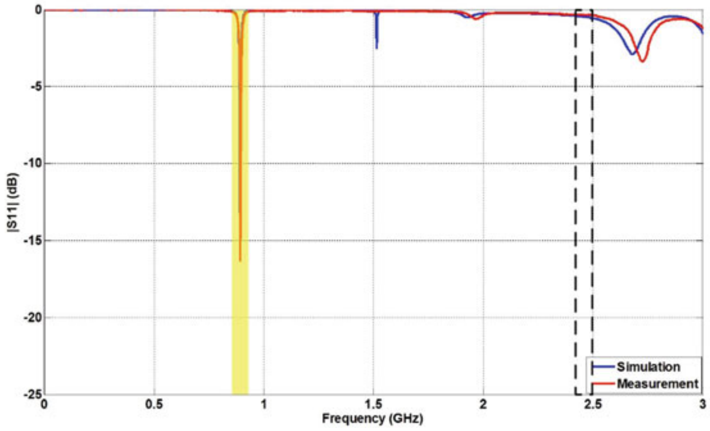


Fig. 9.13 Simulated and measured return loss for reconfigurable antenna at ISM 868 MHz state ©Springer [19]

9.4.3 Antenna

In Figs. 9.13 and 9.14 the simulated and measured return losses of the reconfigurable antenna are shown for the two frequency states, i.e., 868.0 MHz and 2.45 GHz. In each case the operational band is colored and the non-operational band is marked with a dashed line. The simulations have been carried out with *CST Microwave Studio* electromagnetic solver [34].

It can be observed that in the working bands, ISM 868 MHz in Fig. 9.13 and ISM 2.45 GHz in Fig. 9.14, the antenna shows low return-loss values, i.e., good impedance match. This means that in these bands the vast majority of the transmitted/received power is delivered to the antenna, while in the non-working

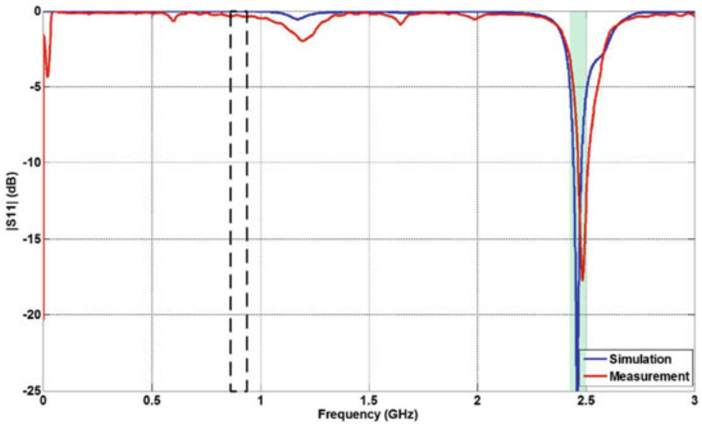


Fig. 9.14 Simulated and measured return loss for reconfigurable antenna at ISM 2.45 GHz state ©Springer [19]

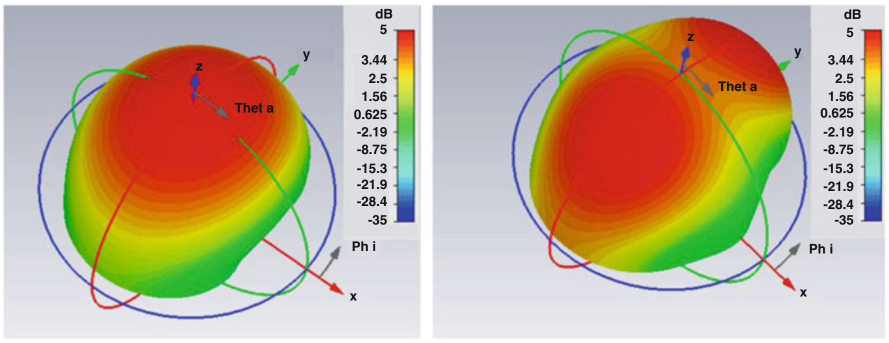


Fig. 9.15 Simulated radiation patterns of reconfigurable antenna: 868 MHz state (*left*); 2.45 GHz state (*right*) ©Springer [19]

bands, ISM 2.45 GHz in Fig. 9.13 and ISM 868 MHz in Fig. 9.14, the antenna is mismatched and, as a consequence, most of the power is reflected towards the RF front-end.

In Fig. 9.15 the simulated radiation patterns are depicted. In order to take a deeper look, the radiation patterns for both bands have also been measured in an anechoic chamber. The measurement setup is shown in Fig. 9.16. Given the fact that the electrical size of the antenna is small at 868 MHz, a sleeve balun has been used in order to suppress the outer currents on the RF cable.

In Figs. 9.17 and 9.18, simulated and measured radiation patterns at 868 MHz and 2.45 GHz are shown. An excellent agreement has been obtained between simulations and measurements. Comparing the radiation patterns in both bands, it can be concluded that they are of the same kind, i.e., in both cases the radiation pattern is broadside (maximum gain normal to the plane of the antenna). On the

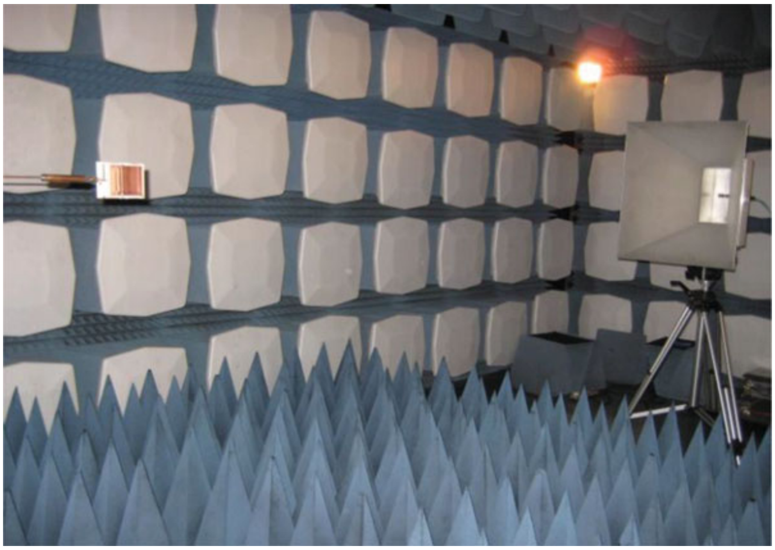


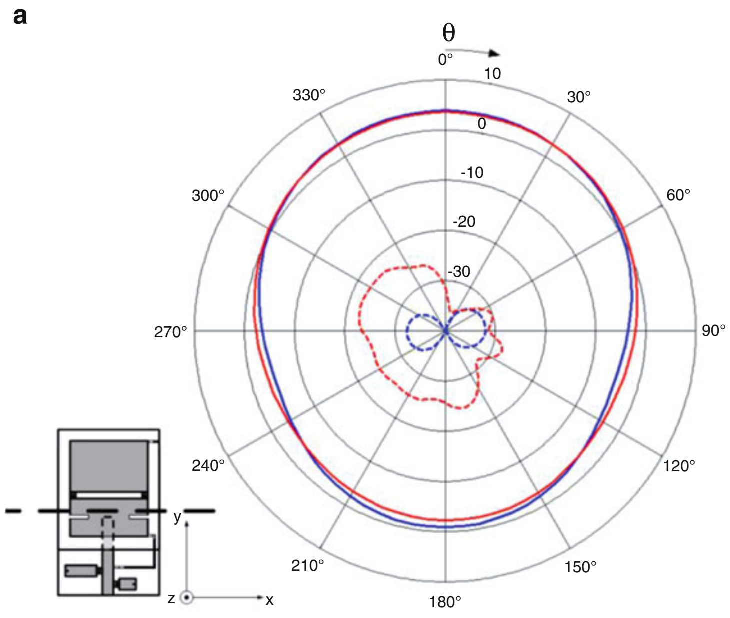
Fig. 9.16 Radiation pattern measurement setup ©Springer [19]

other hand, at 868 MHz it was expected to obtain a typical patch antenna radiation pattern, with a strong contribution from the co-polar component and a negligible cross-polar component. The reason for the higher cross-polar component observed in the measurement is the diffraction in the cable used in the measurement setup. On the other hand, at 2.45 GHz the contribution of an orthogonal higher-order mode increases both the simulated and measured cross-polar patterns (see Fig. 9.18a). This is an orthogonal mode that resonates at 2.6 GHz.

9.5 Conclusions

In this chapter, an FPGA-based SDR platform with cognitive features has been presented. This platform is based on a fully reconfigurable hardware and operates as a 12.8 Mbps RF-to-Ethernet bridge in the ISM bands of 868 MHz and 2.45 GHz. The reconfiguration between these two bands is a powerful tool for fighting undesired effects found in industrial environments, such as signal shadowing or multipath fading. Moreover, in order to ensure the communication in an interference-free frequency, the platform includes a Dynamic Spectrum Access algorithm.

The data-processing algorithms of the platform have been implemented in an FPGA using Xilinx's *System Generator for DSP* rapid prototyping tool. A MicroBlaze processor has also been included to control the dynamic partial reconfiguration of the FPGA for small in-band frequency changes. In order to achieve full-band reconfiguration, a commercial RF front-end and a custom reconfigurable antenna



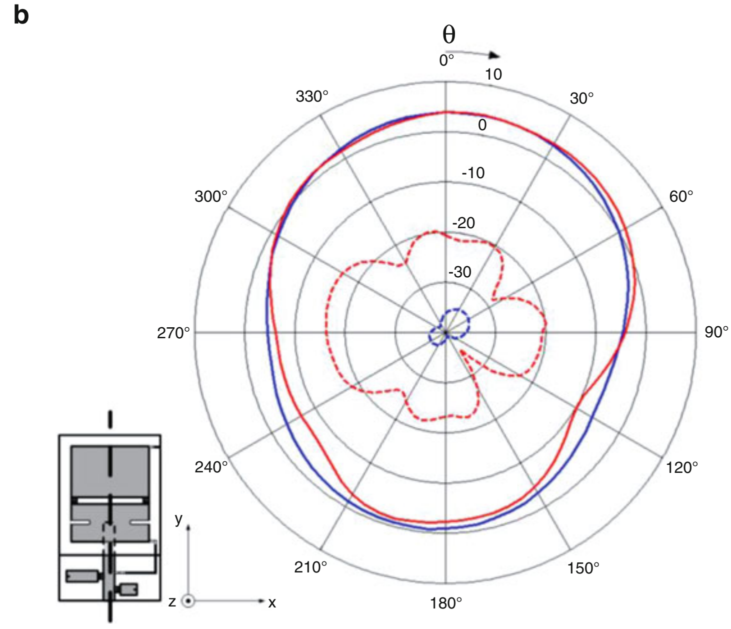
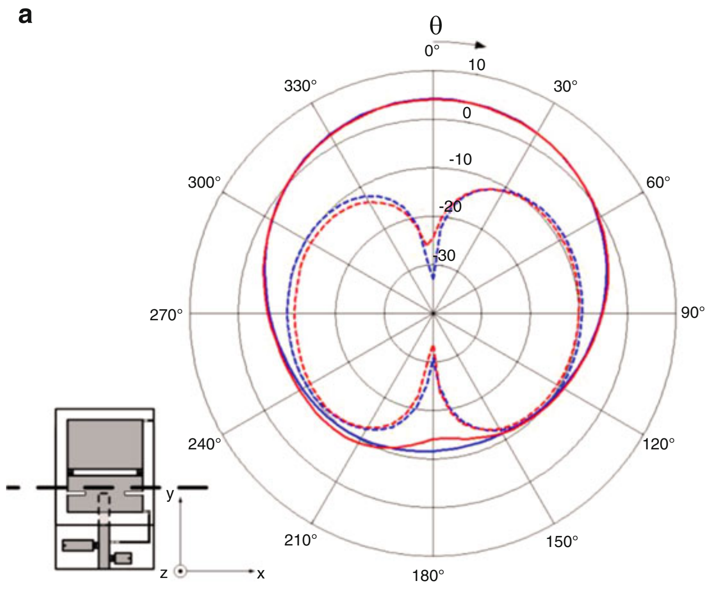


Fig. 9.17 Simulated (*blue*) and measured (*red*) co-polar (*solid line*) and cross-polar (*dashed line*) antenna gain (dBi) at 868.0 MHz © Springer [19]. (**a**) Plane $\phi = 0°$ (XZ). (**b**) Plane $\phi = 90°$ (YZ)



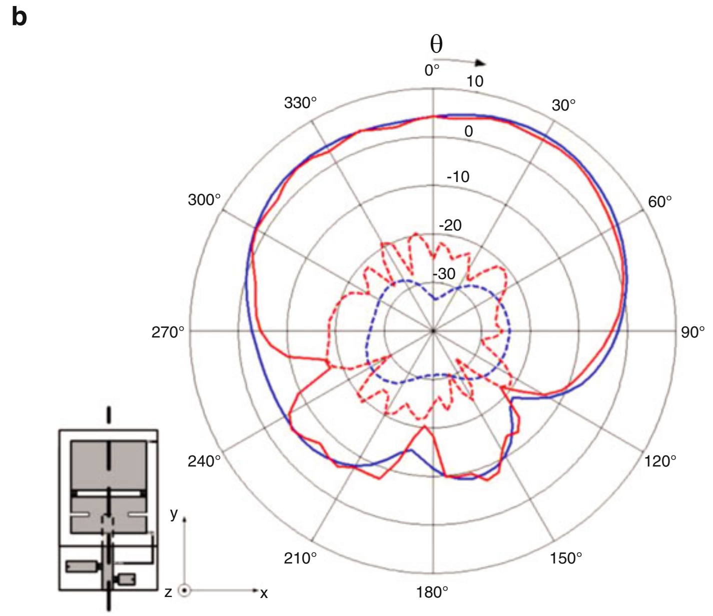


Fig. 9.18 Simulated (*blue*) and measured (*red*) co-polar (*solid line*) and cross-polar (*dashed line*) antenna gain (dBi) at 2.45 GHz © Springer [19]. (**a**) Plane $\phi = 0°$ (XZ). (**b**) Plane $\phi = 90°$ (YZ)

have been integrated. The antenna includes PIN diodes for frequency reconfiguration, and its design has been optimized for direct placement on metallic surfaces, which is a useful feature for industrial environments. Design and implementation details of all the previous blocks have been presented, along with the measurement results.

This platform sets the framework for implementing and testing future mechanisms, such as time-critical cognitive MAC layers with the aim of improving the reliability, robustness, and timeliness of wireless communications in industrial environments under hard real-time requirements [35].

Acknowledgements This work has been partly supported by EFITRANS (ETORTEK) and CIIRCOS (PC2013-68) projects of the Basque Government (Spain), and by COWITRACC (TEC2014-59490-C2-2-P) project of the Spanish Ministry of Economy and Competitiveness.

References

1. Gungor, V.C., Hancke, G.P.: Industrial wireless sensor networks: challenges, design principles, and technical approaches. In: IEEE Trans. Ind. Electron. **56**(10), 4258–4265 (2009)
2. Bocca, M., Eriksson, L.M., Mahmood, A., Jantti, R., Kullaa, J.: A synchronized wireless sensor network for experimental modal analysis in structural health monitoring. Comput. Aided Civil Infrastruct. Eng. **26**(7), 483–499 (2011)
3. Kaur, N., Monga, S.: Comparison of wired and wireless networks: a review. Int. J. Adv. Eng. Technol. **V**(II), 34–35 (2014)
4. Stenumgaard, P., Chilo, J., Ferrer-Coll, P., Angskog, P.: Challenges and conditions for wireless machine-to-machine communications in industrial environments. IEEE Commun. Mag. **51**(6), 187–192 (2013)
5. Varela, M.S., Sanchez, M.G.: RMS delay and coherence bandwidth measurements in indoor radio channels in the UHF band. IEEE Trans. Veh. Technol. **50**(2), 515–525 (2001)
6. Raut, R.D., Kulat, K.D.: SDR design for cognitive radio. In: 4th International Conference on Modeling, Simulation and Applied Optimization (ICMSAO), pp. 1–8 (2011)
7. Kaiser, T., Perez-Guirao, M.D., Wilzeck, A.: Cognitive radio & networks in the perspective of industrial wireless communications. In: Second International Workshop on Cognitive Radio and Advanced Spectrum Management. CogART, pp. 24–29 (2009)
8. Zong, W., Arslan, T.: A low power reconfigurable heterogeneous architecture for a mobile SDR system. In: International Conference on ICECE Technology, pp. 313–316 (2008)
9. EtherCAT: Industrial Ethernet technologies: overview (2014). http://www.ethercat.at/download/documents/Industrial_Ethernet_Technologies.pdf
10. HART: WirelessHART technology overview (2014). http://en.hartcomm.org/hcp/tech/wihart/wireless_overview.html
11. ISA: ISA-100.11a-2011 wireless systems for industrial automation: Process control and related applications. https://www.isa.org/store/products/product-detail/?productId=118261
12. Casado, F., Arriola, A., Arruti, E., Parron, J., Ortego, I., Sancho, I.: 2.45 GHz printed IFA on metallic environments: Clearance distance and retuning considerations. In: 6th European Conference on Antennas and Propagation (EUCAP), pp. 921–924 (2012)
13. Guocheng, L., Mrad, N., Xiao, G., Zhenzhong, L., Dayan, B.: Metallic environmental effect on RF-based energy transmission. IEEE Antennas Wirel. Propag. Lett. **11**, 925–928 (2012)
14. Mitola, J.: Software radios-survey, critical evaluation and future directions. In: National Telesystems Conference. NTC-92, pp. 13/15–13/23 (1992)

15. Torrego, R., Val, I., Muxika, E.: OQPSK cognitive modulator fully-FPGA-implemented via dynamic partial reconfiguration and rapid prototyping tools. Wireless Innovation Forum Conference on Technologies and Software Defined Radio (SDR-WInnComm) (2011)
16. Haessig, D., Hwang, J., Gallagher, S., Uhm, M.: A case study of Xilinx System Generator design flow for rapid development of SDR waveforms. In: SDR Forum Technical Conference (2005)
17. Xilinx: System Generator for DSP (2016). http://www.xilinx.com/products/design-tools/vivado/integration/sysgen.html
18. Rodriguez, P.M., Torrego, R., Casado, F., Fernandez, Z., Mendicute, M., Arriola, A., Val, I.: Dynamic spectrum access integrated in a wideband cognitive RF-ethernet bridge for industrial control applications. J. Signal Process. Syst. **83**(1), 19–28 (2016)
19. Casado, F., Torrego, R., Rodriguez, P., Arriola, A., Val, I.: Reconfigurable antenna and dynamic spectrum access algorithm: integration in a cognitive radio platform for reliable communications. J. Signal Process. Syst. **78**(3), 267–274 (2014)
20. Vo, N.D., Le-Ngoc, T.: Maximum likelihood (ML) symbol timing recovery (STR) techniques for reconfigurable PAM and QAM modems. Wirel. Person. Commun. **41**(3), 379–391 (2007)
21. Bagwari, A., Tomar, G.S.: Cooperative spectrum sensing in multiple energy detectors based cognitive radio networks using adaptive double-threshold scheme. Int. J. Electron. **101**(11), 1546–1558 (2014)
22. Adlard, J.: Frequency shift filtering for cyclostationary signals, Ph.D. dissertation (2000)
23. Kosunen, M., Turunen, V., Kokkinen, K., Ryynanen, J.: Survey and analysis of cyclostationary signal detector implementations on FPGA. IEEE J. Emerging Sel. Top. Circuits Syst. **3**(4), 541–551 (2013)
24. Luise, M., Reggiannini, R.: Carrier frequency recovery in all-digital modems for burst-mode transmissions. IEEE Trans. Commun. **43**(2/3/4), 1169–1178 (1995)
25. Meyr, H., Moeneclaey, M., Fechtel, S.: Digital Communication Receivers: Synchronization, Channel Estimation, and Signal Processing. Wiley Inc., New York (1997)
26. Nutaq: Nutaq Radio420X (2015). http://nutaq.com/en/products/radio420x
27. Jong-Hyuk, L., Zhe-Jun, J., Chang-Wook, S., Tae-Yeoul, Y.: Simultaneous frequency and isolation reconfigurable MIMO PIFA using PIN diodes. IEEE Trans. Antennas Propag. **60**(12), 5939–5946 (2012)
28. Azim, R., Islam, M.T., Misran, N.: Compact tapered-shape slot antenna for UWB applications. IEEE Antennas Wirel. Propag. Lett. **10**, 1190–1193 (2011)
29. Nashaat, D.M., Elsadek, H.A., Ghali, H.: Single feed compact Quad-Band PIFA antenna for wireless communication applications. IEEE Trans. Antennas Propag. **53**(8), 2631–2635 (2005)
30. NXP: BAP55LX silicon pin diode (2016). http://www.nxp.com/products/rf/diodes/pin_diodes/BAP55LX.html
31. Ilkyu, K., Rahmat-Samii, Y.: RF MEMS switchable slot patch antenna integrated with bias network. IEEE Trans. Antennas Propag. **59**(12), 4811–4815 (2011)
32. Akerberg, J., Gidlund, M., Bjorkman, M.: Future research challenges in wireless sensor and actuator networks targeting industrial automation. In: 9th IEEE International Conference on Industrial Informatics (INDIN), pp. 410–415 (2011)
33. Val, I., Casado, F., Rodriguez, P.M., Arriola, A.: FPGA-based wideband channel emulator for evaluation of wireless sensor networks in industrial environments. In: IEEE International Conference on Emerging Technology and Factory Automation (ETFA), pp. 1–7 (2014)
34. CST: Microwave Studio (2016). http://www.cst.com
35. Rodriguez, P.M., Val, I., Lizeaga, A., Mendicute, M.: Evaluation of cognitive radio for mission-critical and time-critical WSAN in industrial environments under interference. In: IEEE World Conference on Factory Communication Systems (WFCS) (2015)

Chapter 10
Synchronization in NC-OFDM-Based Cognitive Radio Platforms

Farid Shamani, Tapani Ahonen, and Jari Nurmi

10.1 Introduction

With advancements in wireless technology, applications demand higher data rates. On the other hand, spectrum scarcity is becoming a major problem due to the increment of the spectrum users [1]. As Fig. 10.1 illustrates, a real measurement taken in downtown Berkeley shows that licensed users (also known as primary users) occupied most of the prime spectrum while do not utilize it efficiently. The Dynamic Spectrum Access (DSA) is a promising solution to cope with spectrum scarcity problem. Conceptually, DSA exploits the white spaces between frequencies occupied by several licensed users. Figure 10.2 depicts how DSA enables the secondary usage of the white spaces within a licensed spectrum without interfering primary user's activities. DSA approach also improves spectrum utilization significantly [6].

With the advent of the Software Defined Radio (SDR) in 1991, transceiver could carry out the entire baseband processing in software. Indeed, the SDR is a platform in which, at least, a portion of the implementation is held in software [24]. A more intelligent and advanced version of the SDR known as Cognitive Radio (CR) which was proposed by Joseph Mitola in 1999 [13]. The CR is a radio platform which is always aware of its environment and can rapidly change its operating parameters by considering new characteristics of the spectrum. The user is not even notified when the CR is changing its corresponding parameters. The CR is an alternative solution to mitigate spectrum scarcity problem by reusing the portion of the spectrum

F. Shamani (✉)
Tampere University of Technology, Korkeakoulunkatu 1, 33720, Tampere, Finland
e-mail: farid.shamani@tut.fi

T. Ahonen • J. Nurmi
Department of Electronics and Communications Engineering, Tampere University of Technology, Tampere, Finland
e-mail: ukaata@gmail.com; jari.nurmi@tut.fi

W. Hussain et al. (eds.), *Computing Platforms for Software-Defined Radio*,
DOI 10.1007/978-3-319-49679-5_10

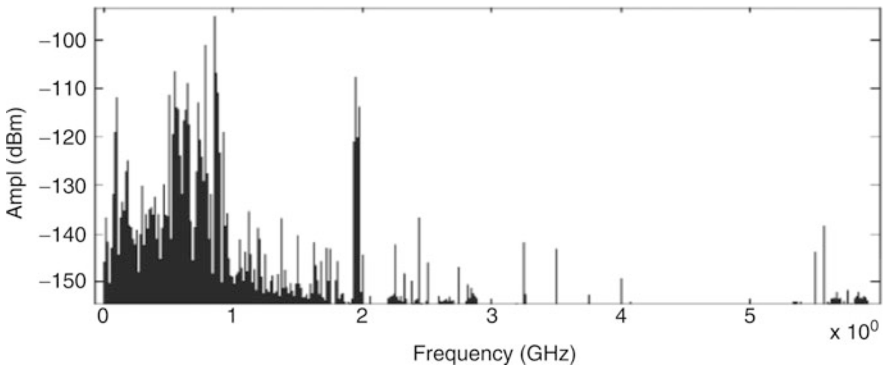


Fig. 10.1 Spectrum utilization snapshot at downtown Berkeley [25]

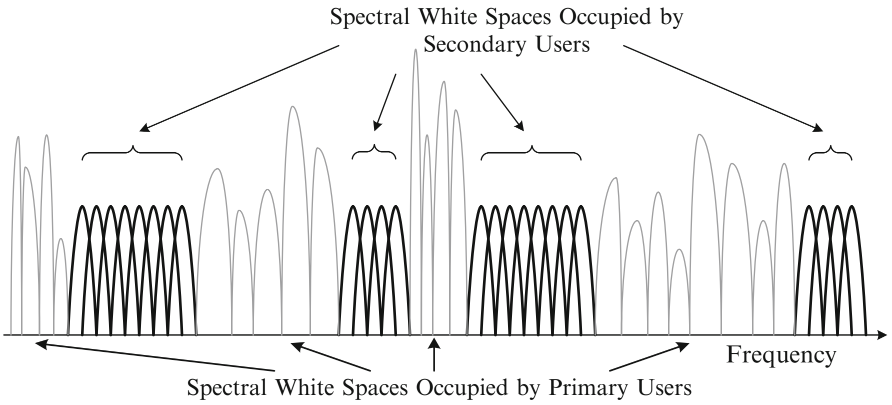

Spectral White Spaces Occupied by Secondary Users
Frequency
Spectral White Spaces Occupied by Primary Users

Fig. 10.2 Spectrum utilization using DSA technology [24, p. 151]

assigned to the licensed user for secondary users without disrupting the operations of any nearby primary user [7]. At first glance, the above-mentioned approaches look simple. However, a simple alteration in frequency-domain arises several challenges in the transceiver of which the synchronization is one of the prominent ones. Briefly, the receiver has no idea that the transmitter sends the data in which frequency band.

10.2 Synchronization Issues in NC-OFDM Systems

Synchronization is an essential issue in all digital communication systems in order to extract the data out of the received signal. Therefore, one of the main challenges for digital design engineers is to establish a reliable, robust, and accurate synchronization. Synchronization is a process in which the receiver detects the existence of an incoming packet and then distinguishes the boundaries of the received packet.

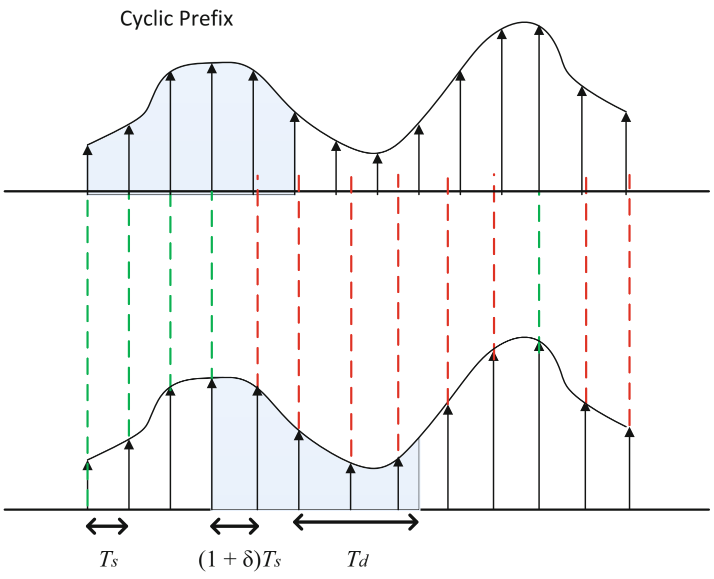


Fig. 10.3 The effect of a bad synchronizer in an ideal channel [20]

Synchronization errors yield in either time, frequency, or both. Conceptually, the receiver must run the sampling process on the incoming wave stream in certain time intervals. Any alteration in sampling time misleads the receiver to properly extract the data. Figure 10.3 illustrates the picture when the receiver is not able to execute the sampling process on time. As it can be seen, sampling at a period of $(1 + \delta)T_s$ causes a δ shift in time which leads to an incorrect sampling.

10.2.1 OFDM Versus NC-OFDM

NC-OFDM is an extension of the conventional OFDM modulation where unused subcarriers are deactivated to eliminate interference made by the secondary user with adjacent licensed users. Therefore, understanding the architecture of an OFDM transceiver is strongly required before stepping forward to NC-OFDM systems. As Fig. 10.4 shows, a high-speed data stream $X(n)$ is demultiplexed into N_u parallel ones $x^{(k)}(n), k = 0, \ldots, N_u$. This procedure is usually performed by using a serial-to-parallel converter to form a set of data subcarriers. Then each subcarrier is individually modulated using either QAM or PSK modulation technique to produce $y^{(k)}(n), k = 0, \ldots, N_u$ [24]. Typically, as long as the receiver knows the modulation pattern, each subcarrier is modulated with the same constellation [3]. Thereafter, the baseband OFDM waveform $s^{(\ell)}(n), \ell = 0, \ldots, N$ is constructed as an N-input IDFT unit with $N \geq N_u$ defined as Eq. (10.1). The remaining $N - N_u$ unused inputs

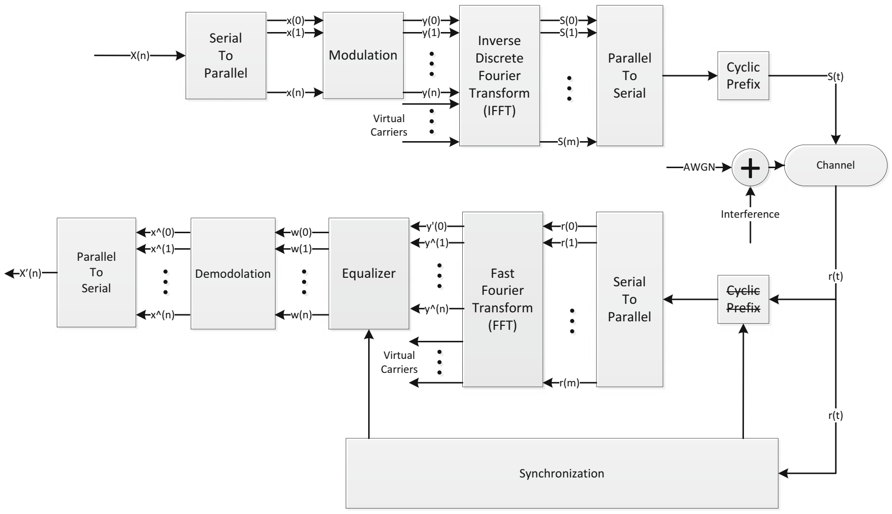


Fig. 10.4 OFDM transceiver architecture [19]

(Also known as VR) are set to zero. VCs are employed as guard bands to cope with the Inter-Symbol Interference (ISI) problem. ISI is imposed to the system due to the interferences caused by transmission power of the adjacent subcarriers. Typically, IDFT is implemented using an IFFT function. Eventually, the transmitter inserts a CP before converting all the subcarriers to the composite signal $s(n)$ [15].

$$s^{(\ell)}(n) = \frac{1}{N}\sum_{k=0}^{N-1} y^{(k)}(n)e^{j2\pi k\ell/N} \tag{10.1}$$

The receiver performs the inverse procedure to the received signal. The first step is to remove the CP from the received signal $r(n)$. Next, considering Eq. (10.2), a serial-to-parallel demultiplexer converts the serial stream to parallel ones. Thereafter, the information is extracted by performing DFT function on the received parallel waveforms $r^{(\ell)}(n)$. DFT is performed using an FFT function to produce $\hat{y}^{(k)}(n)$. Next step is to compensate channel distortion imposed to the subcarriers using an equalizer. The equalized subcarriers $\omega(n)$ are demodulated and, subsequently, the serial steam $x'(n)$ is fetched by employing a parallel-to-serial converter on parallel streams $x^{(k)}(n)$ [15].

$$\hat{y}^{(\ell)}(n) = \sum_{\ell=0}^{N-1} r^{(\ell)}(n)e^{(-j2\pi k\ell/N)} \tag{10.2}$$

Figure 10.5 presents the architecture of an NC-OFDM transceiver. By taking into consideration both Figs. 10.4 and 10.5, the fundamental difference in both architectures is the synchronizer. In contrast to the OFDM where the synchronization was

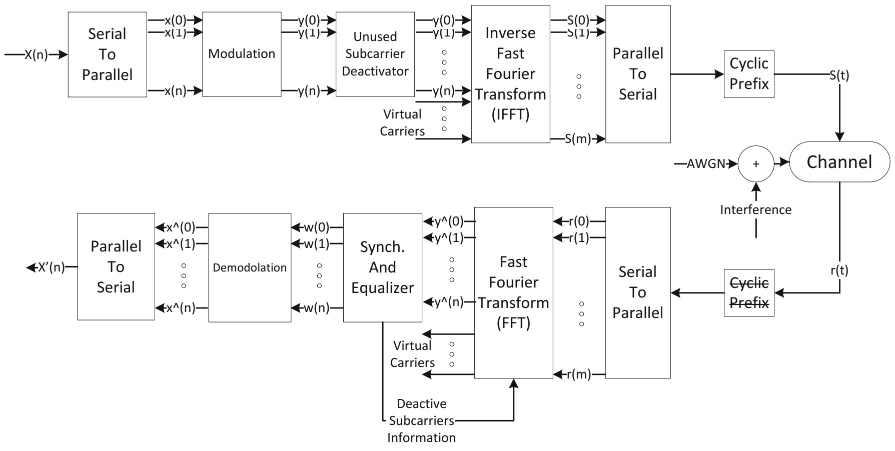


Fig. 10.5 NC-OFDM transceiver architecture [19]

done prior to the Fast Fourier Transform (FFT) (synchronization in time-domain), an NC-OFDM receiver performs the synchronization following the FFT (synchronization in frequency-domain). The main reason which enforces the receiver to perform synchronization in frequency-domain is a simple change in time-domain representation of the signal. This is mainly due to the time-domain stochastic alteration of the preamble, pilot, and data carriers of the secondary user based on primary user activities. In addition, all secondary users must reduce their transmission power to avoid interfering with the licensed user due to the sidelobe leakages caused by the OFDM nature.

10.2.2 Synchronization Methods

As previously discussed, contrary to OFDM, in NC-OFDM systems a precise location for signal carriers are not guaranteed. Moreover, the location of the licensed user is changed across the spectrum over the time. Therefore, an NC-OFDM receiver has no prior information about the subcarriers employed by the secondary transmitter. This is one of the key challenges emerged from synchronization point of view. Some researches proposed Out-Of-Band (OOB) architecture to inform the receiver about the new characteristics of the spectrum [11, 17, 23, 26, 27]. In this methodology, a special-purpose pre-dedicated channel is established between the transmitter and receiver to reveal information with regard to the active subchannels using by the secondary user as well as inactive ones. However, OOB methodology seems not to be a suitable solution since the dedicated channel may not be available in some practical situations [16].

Conceptually, in contrast to OOB mechanism, the prerequisite synchronization information of a secondary transmitter is embedded to the incoming packet itself. However, the secondary receiver still has the problem to detect a secondary transmission. Fortunately, the Federal Communication Commision (FCC) provides secondary users willing to transmit over the licensed spectrum with transmission information of the primary user, including the structure of the signal, power, location of the primary user, etc. [21]. Having substantial information about the primary user's activities enables the secondary receiver to filter out subcarriers occupied by the primary user. Following discusses few proposed method with regard to in-band communication in NC-OFDM systems.

In [16], the existence of the primary user as well as the corresponding interferences is discussed. Authors explain an A Posterior Probability (APP) algorithm to discover active subchannels. A threshold-based Hard Decision-based Detection (HDD) scheme is used to detect NC-OFDM symbols as soon as an active subchannel is detected. However, the HDD operates quite confident for the subset of subchannels far away from primary user band while the performance is drastically degraded for subchannels closed to the primary user band. In order to mitigate the performance degradation to some extent, a Soft Decision-based Detection (SDD) is performed to increase accuracy for (noisy) subchannels adjacent to the primary user. However, the existence of the primary user still causes severe synchronization problems. Eventually, the performance of the proposed method is heavily dependent on the primary user activities, power, etc.

In [10], authors claim that the system code rate of the proposed algorithm in [16] is 1/4 when only half of the subcarriers are active. Hence, in order to improve the system code rate, they proposed a Low-Density Parity-Check (LDPC) after performing the APP algorithm. The authors emphasized more on how to create the LDPC code while a perfect spectrum synchronization is taken into assumption. Since the obtained results are based on simulations, in hardware perspective the proposed method requires additional hardware element to decode the received LDPC. In addition, employing more hardware element might increase the energy consumption as well as the silicon area.

In [22], a fractional bandwidth model has been proposed in which a different preamble pattern is designed. The information of the active subbands generated by a Pseudo-Noise (PN) sequence is transmitted over the channel. Therefore, the receiver identifies and detects the active subbands in the frequency-domain. In time-domain, the preamble is represented with two identical halves whose sign bits are inverted. However, the proposed method is based on considering only contiguous subbands (OFDM-based CR system) which is not practical in NC-OFDM systems. Moreover, in reality the NC-OFDM secondary transmitter always keeps its transmission power lower than the licensed user to mitigate interfering with the primary band due to the sidelobe leakages [9].

Saha et al. [6, 18] proposed a blind synchronization method where the receiver is capable of regenerating time-domain preamble of the frequency-domain representation of the same preamble. Furthermore, they considered the idea of employing multiplier-less correlator instead of performing a full 16-bit Multiplier-Accumulate

(MAC) operation in frequency-domain. Furthermore, the primary user's activities are taken into account and, consequently, subchannels occupied by the primary user are filtered using a binary mask. As a result, the existence of the primary user is eliminated right after performing the FFT. Henceforward, the rest of the synchronization process is similar to the one of OFDM.

In OFDM systems, synchronization can be performed either in time-domain, frequency-domain, or both. Since synchronization in frequency-domain requires many cycles to compute, an OFDM receiver prefers to perform synchronization in time-domain [4]. Furthermore, each packet starts with the so-called *preamble*. The preamble is repetitive sequence known for both transmitter and receiver. Due to the repetitive nature of the preamble, time-domain representation of the preamble has a good autocorrelation property. Nevertheless, any change in time-domain representation of the preamble will cause the NC-OFDM receiver to fail detecting a packet. On the other hand, a secondary transmitter has to alter its transmitting frequencies over the time (in the frequency-domain) depending on the licensed user's activities. A simple change in frequency-domain results in a complete change of the time-domain waveform of the preamble. This is the reason why an NC-OFDM receiver has no idea about the time-domain representation of the preamble. Therefore, synchronization based on preamble detection should be performed in frequency-domain in NC-OFDM-based systems. In addition, NC-OFDM receivers are compromised with two main challenges, as well. First, only a portion of the subcarriers are available (the rest are occupied by the licensed user). Second, they should face a low SNR region due to the low transmission power of the secondary user (to mitigate interferences with the licensed user) [9]. The above-mentioned obstacles disable conventional synchronization methods which were feasible in OFDM systems. Figure 10.6 depicts the architecture of an NC-OFDM receiver. By taking into account the block diagram, synchronization is performed in several steps explained in the following [18].

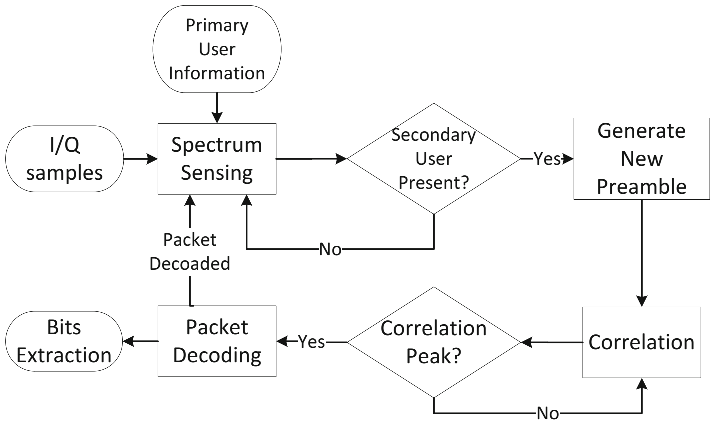


Fig. 10.6 NC-OFDM synchronization scheme [20]

10.2.3 Active Subcarrier Detection

The first step is to filter those subcarriers occupied by the licensed user. Since characteristics of the licensed user are known for the secondary receiver [21], it can be simply done by excluding primary user's subcarriers when collecting all the subcarriers (after the FFT). The hardware circuit of such a filter can be a simple XOR gate. Henceforth, a set of collected subcarriers implies the entire spectrum where the secondary user might have been active. At this stage, all subcarriers are employed for spectrum sensing in a pipeline manner using one of the spectrum sensing methods. The corresponding spectrum sensing results of all subcarriers are reported to a decision maker unit. The decision maker unit compares computed results with an approximate threshold. The threshold can be approximately calculated as follows: when there is no incoming signal or during the inter-packet duration, the output of the FFT is the magnitude of the noise level. The decision maker unit is able to dynamically approximate a threshold by subtracting the noise level from the FFT output. The last step in this phase is to stop spectrum sensing as soon as the decision maker unit detects the results that have risen over the threshold. Note that a copy of the incoming packet is also buffered to extract time-domain samples in the next step.

10.2.4 Preamble Regeneration in Time-Domain

The time-domain representation of the frequency-domain coefficients are generated by employing a low-cost IFFT unit. Once the time-domain preamble is generated, the coefficients of the correlator are re-initialized by new samples. Henceforward, the synchronization mechanism is quite similar to OFDM. The similarity of the new coefficients with the buffered version of the incoming packet (which stored in previous step) is computed to detect the boundary of the signal. As soon as a peak is found, the correlation is inferred to be successful. The last step is to call decoding unit to extract the data out of the received bits.

10.2.5 Failure in Correlation

When the correlation computations do not yield appropriate results, for whatever reason, the spectrum sensing mechanism is started to obtain a new set of subcarriers which is occupied by the secondary transmitter. There are several reasons which might desynchronize the receiver with the transmitter. A low secondary signal SNR is one of the main reasons. This happens due to the fact that a secondary user near to a primary user should minimize its transmission power as low as possible. A high power primary user has the potential to overwhelm any adjacent secondary transmitter. Furthermore, a secondary transmitter might have been using a white space spectrum which belonged to a turned-off primary. When the primary user shows an activity on the white space band, the secondary user should immediately disrupt its transmission. In such circumstances, the receiver must be intelligent and agile enough to start sensing the channel from the beginning.

10.3 Proposed Synchronization Dataflow

In this section, we propose a reconfigurable platform for sensing the spectrum as well as the preamble detection/generation. The term "reconfigurability" mainly refers to the correlator architecture to perform autocorrelation and cross-correlation on demand. In addition, instead of using wide-input multicorrelator (e.g., 4096-tap), we employ several smaller ones in parallel. Figure 10.7 illustrates the block diagram of the proposed receiver architecture. However in order to keep the figure clean, only one multicorrelator is shown in the figure. In this design, a *controller unit* monitors the behavior of the whole system. At the instantiation, multicorrelators are set up by the controller to perform spectrum sensing. Spectrum sensing is done by executing the autocorrelation function of the incoming signal with a delayed version of itself. Each multicorrelator is assigned to compute a particular portion of the spectrum. Computation results obtained by all the multicorrelators are compared to an approximate threshold. As soon as the computation results of a multicorrelator are exceeded over the threshold, the decision maker unit reports the event to the controller. Next, the controller issues the command to *preamble regenerator unit* to generate time-domain representation of the frequency-domain

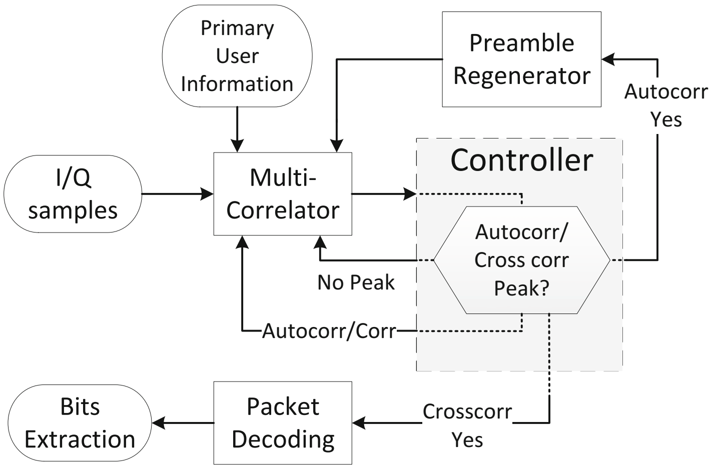


Fig. 10.7 Proposed NC-OFDM synchronization scheme [20]

preamble. In the meantime, the controller sets up the corresponding multicorrelator to compute cross-correlation function while dismisses the other multicorrelators. Once the time-domain version of the preamble is generated, the coefficients of the active multicorrelator are set by the time-domain coefficient of the generated preamble while the buffered version of the In-Phase/Quadrature (I/Q) signals is fed as the input signal. When the multicorrelator is re-initiated, the maximum similarity between the noisy buffered signal and the clean version of the time-domain preamble is computed. Again, the obtained result is compared to a second threshold by the decision maker unit. Once the second peak is detected, the controller infers that the packet is detected and it is ready to be handed to the decoder unit. When the decoder unit extracts the data bits, it reports to the controller to make the proper action.

10.4 FPGA Implementation of the Multicorrelator

Figure 10.8 presents an overview of the proposed NC-OFDM synchronizer. The core content of the multicorrelator is based on a *Memory Block* to store predefined preambles as well as the regenerated preambles, a *MAC* Operator for computing complex Sum-of-Products (SoPs), a *Decision Maker* unit to compare obtained results with a threshold, followed by a *Controller* to monitor the entire system; each block is described in the following in detail.

10.4.1 Memory Block

In this design, the considered memory block is inferred as an SRAM initialized by frequency-domain representation of the preamble. The SRAM is used to store predefined preamble values in frequency-domain representation as well as the regenerated preambles. The SRAM data are mainly used as a sequence of coefficients for cross-correlation purposes. The SRAM has a 32-bit data width output port where the first half (16 least significant bits) includes *Real* part of the preamble and, consecutively, the second half (16 most significant bits) contains the *Imaginary* part. One coefficient is inserted to the FIR structure unit per clock cycle.

10.4.2 Decision Maker Unit

The main objective of the *Decision Maker* unit is to compare results computed by the MAC operator with a threshold. The decision maker employs different threshold for autocorrelation and cross-correlation purposes. When the multicorrelator is set to perform autocorrelation function, the decision maker estimates the noise level

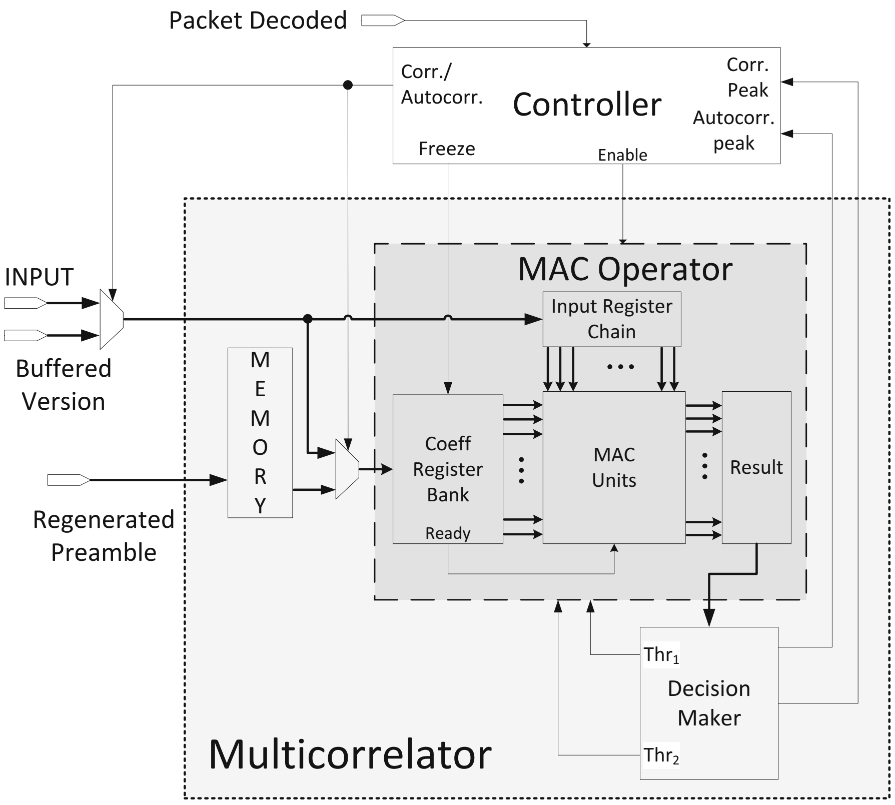


Fig. 10.8 The infrastructure of the proposed NC-OFDM synchronizer

during the silent time and, based on that, an approximate threshold is set. When there is an incoming packet, the noise level is subtracted from the FFT output and the final 16-bit result is compared to the approximate threshold. When the result exceeds the threshold, an autocorrelation peak is found, meaning that the transmitter is sending a packet. In cross-correlation mode, the decision maker unit makes the decision based on a comparison between the results given by the MAC operator with a 2-step threshold level. The main idea of employing 2-step threshold level is explained in [2] in detail. Figure 10.9 shows how the decision maker unit decides whether a peak is found. There are two threshold level: a preliminary threshold Thr_1 alongside an original threshold Thr_2. When the first half of the incoming signal exceeds Thr_1, the decision maker unit issues a command to the MAC operator to compute the rest of the calculation. This is generally made to minimize unnecessary calculations of the MAC operations. Indeed, MAC operations are the most power-hungry operation in this design. If the first half of the MAC operations meets the Thr_1, the second half of the MAC operations are executed. The packet is detected if the MAC operation result exceeds the Thr_2. Otherwise, the decision maker unit infers that the packet

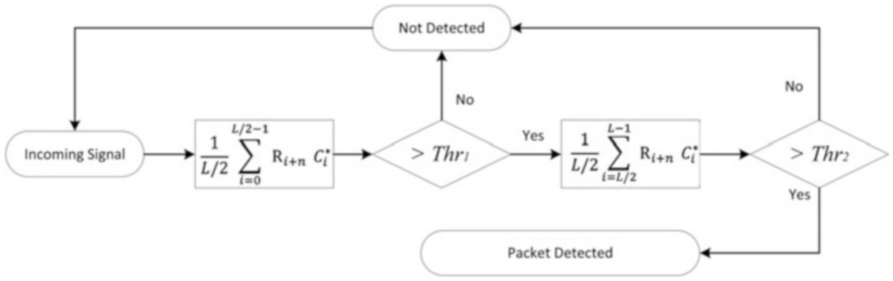

Not Detected
No
No
Incoming Signal
$\frac{1}{L/2}\sum_{i=0}^{L/2-1} R_{i+n} C_i^*$
$> Thr_1$
Yes
$\frac{1}{L/2}\sum_{i=L/2}^{L-1} R_{i+n} C_i^*$
$> Thr_2$
Yes
Packet Detected

Fig. 10.9 2-step threshold detection flowchart [20].

was not detected and, consequently, inform the controller to discard undergoing procedures. The decision maker unit informs the controller with two separate signals each of which indicates a particular peak that is found either in autocorrelation or cross-correlation. Choosing a proper value for both Thr_1 and Thr_2 has an impressive impact on correct packet detection even in low SNR regions. A high-value threshold will improve the algorithm robustness, however, it might result in missing frames with lower energy. On the opposite side, a low value might mislead the decision to detect the noise in lower SNR regions. For instance, a good approximation for Thr_1 value could be 45 % of the original Thr_2 in 2-step algorithm [2].

10.4.3 Controller Block

The controller block is the most responsible block in the design. The main task of this block is to make proper action by continuously monitoring the behavior of the system. When the receiver is waiting for a packet, the synchronizer continuously performs autocorrelation function between the incoming signals with a delayed version of itself. Therefore, the controller sets the MAC operator to perform autocorrelation function. In order to perform the autocorrelation function, the controller routes the incoming signal with the delay length of D to the coefficient register bank through a multiplexer. Whenever the coefficient register bank is appropriately loaded, the controller freezes the register bank to avoid register overloading. Henceforward, the controller waits on the decision maker unit to respond. Once the autocorrelation peak is confirmed by the decision maker, the controller prepares the environment for the multicorrelator to perform cross-correlation function between a buffered version of the packet and regenerated time-domain preambles. The cross-correlation function is executed in the same way as the autocorrelation function. This time, the coefficient register bank is loaded by the regenerated preambles which are stored in the memory. Only one coefficient can be loaded to the register bank per clock cycle. Based on the MAC operator architecture, the multicorrelator performs cross-correlation function at the same time the coefficients are being loaded. The functionality of the MAC operator is explained in next subsection. When the coefficients of the multicorrelator are fed by proper values, the controller freezes the register bank and waits for any prompt from decision maker unit. Once the second peak is confirmed, the controller disables the MAC operator and hands the packet to the decoding unit to extract the data. When the data is extracted, the decoding unit informs the controller to restart the synchronization process.

10.4.4 MAC Operator Unit

Due to the nature of the NC-OFDM, the highest order of the tap, e.g. 4096-tap, is required to perform synchronization over the entire spectrum. Therefore, an NC-OFDM receiver has a massive power dissipation along with a huge silicon usage [12]. The MAC operator unit is quite similar to Finite Impulse Response (FIR) filters from infrastructure point of view. As Eq. (10.3) presents, an FIR filter of order N calculates $y(n)$ as the SoP of coefficients $h(k)$ with the input $x(n-k)$. The hardware implementation of the MAC operator unit also calculates $y(n)$ as the SoP of complex conjugate coefficients $c(k)$ with incoming signal $x(n-k)$ on a window whose length is N [8]. The Conjugation is simply done by inverting the sign bit of the imaginary part of the coefficients. As previously discussed, both coefficient and incoming signal are 32 bits long where we stipulated that the first half is the Real part and the remaining 16 bits are the Imaginary part. It is also worth taking into consideration that the products are complex are complex multiplications which have the potential to double resource usage resource usage from the hardware point of view. Furthermore, we considered fixed-width truncation method where only 16 most significant bits are taken into consideration as the final result of the MAC operation. The MAC operator unit is the most intensive block of the multicorrelator due to the massive workload of complex computations. The infrastructure of the MAC operator is described in the following subsections.

$$y(n) = c(n) * x(n) = \sum_{k=0}^{N-1} c(k)x(n-k) \tag{10.3}$$

10.4.4.1 MAC-Based Operator

In general, the MAC operator is composed of three fundamental functional units aptly named adders, multipliers, and delay elements. An N-tap MAC operator is consisted of N delay elements, N multipliers, as well as $N-1$ adders. Furthermore, different architectures are obtained by different arrangements of these functional units. Technically, there are two well-known architectures for implementing MAC operator known as Direct Form (DF) and Transposed Direct Form (TDF), each of which is capable of calculating $y(n)$. We also experience different structures namely

Parallel Direct Form (PDF) as well as Pipelined-Parallel Direct Form (PPDF). Figure 10.10 pictures the above-mentioned architectures with their pros and cons. Figure 10.10a presents the DF architecture where the delay elements are located through the input path. A large N-tap DF introduces a long critical path due to the nature of its architecture. If we assume that each addition takes T_a and each multiplication takes T_m unit of time and number of taps are equal to 256 (as it is in this implementation), the critical path of the DF architecture is equal to $T_{\text{crit}} = T_m + 255T_a$. Therefore, it is not a wise choice for large MAC operator designs at all.

Figure 10.10b shows the TDF architecture which is the optimized version of the DF form. In this architecture, since the delay elements are located between the adders, the critical path is bounded by $T_{\text{crit}} = T_m + T_a$ irrespective to the number of taps. In the TDF version, the inverse version of the coefficient chain $C_{(k)}$, e.i. $(C_k, C_{k-1}, \ldots, C_0)$, is multiplied with the input signal $X(n)$. Thus, this architecture does not require a register chain in its input path (see Fig. 10.8). However, the TDF version has the potential to introduce a long interconnection on its input path.

Figure 10.10c illustrates the PDF form where at the first glance it looks similar to the DF version. The idea behind the PDF architecture is to cope with long interconnection input as well as the critical path to some extent. Despite the DF form, PDF distributes the adders in several parallel levels (hierarchy levels). The number of required levels is proportional to the $\lceil \log_2^N \rceil$ where N is the number of taps. In this case study, since $N = 256$, 8 levels of adders are required. Consequently, the critical path of a 256-tap MAC operator is equal to $T_{\text{crit}} = T_m + \left(\lceil \log_2^N \rceil \times T_a\right) = T_m + 8T_a$. The hardware implementation procedure of this architecture is more complex than the DF and TDF forms.

Figure 10.10d depicts the pipelined version of the PDF form where a register is inserted between each two adders. Although the inserted registers maintain the critical path at its minimum level ($T_{\text{crit}} = T_m + T_a$) while preventing long interconnections, a variety of register elements are introduced to the system. Moreover, the PPDF architecture has the same complexity as the PDF form.

10.4.4.2 Multiplier-less-Based Operator

A study in [5, 14] shows that the M-L-based multicorrelator design offers comparable synchronization performance. In this approach, a sign bit correlation between coefficients and input signal is considered instead of performing 16-bit multiplication. The sign bit correlation can be implemented in hardware by a simple XOR gate. The M-L form reduces a substantial number of registers as well as the DSP blocks. Although the M-L-based multicorrelator takes only 1-bit into account, simulation results obtained by the ModelSim software show that the performance of the MAC operator is at a satisfactory level in our case study. Furthermore, Saha et al. [18] acknowledged that the sign-based correlator has satisfactory performance even in low SNR regions, as well. Based on these evidences, we can claim that the

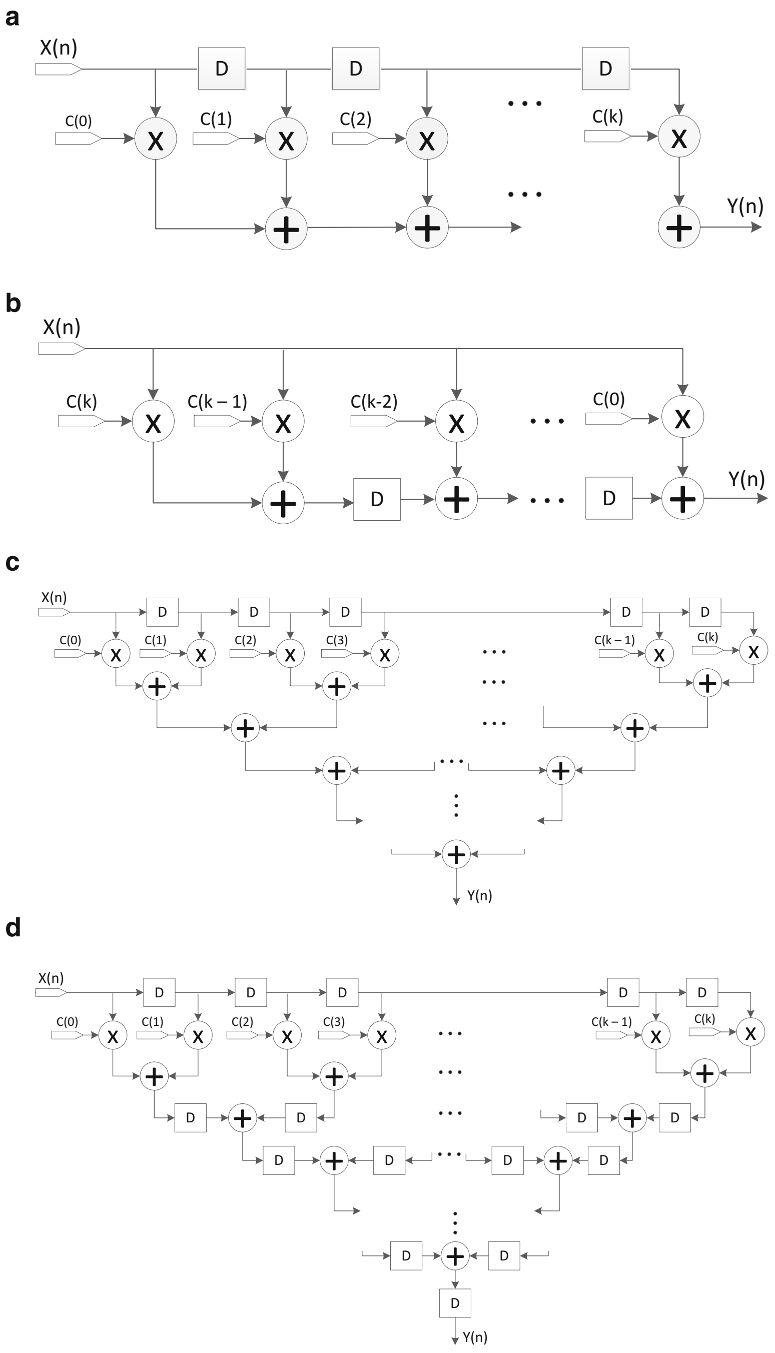


Fig. 10.10 Different architectures for MAC operator unit [20]. (**a**) Direct Form (DF). (**b**) Transposed Direct Form (TDF). (**c**) Parallel Direct Form (PDF). (**d**) Pipelined-Parallel Direct Form (PPDF)

M-L-based multicorrelator is the best choice in our case study. More discussion with regard to the synthesis results of the multicorrelator with different architectures are addressed in Sect. 10.5.

10.5 Synthesis Results

This section discusses with respect to the proposed multicorrelator configured with different architectures. We used VHDL description language to implement the multicorrelator, ModelSim software for simulation, and verification progresses, Quartus-II environment to synthesize the design and generate the netlist. We also took into account the Altera family FPGA targeting Stratix-V speed grade 2 series to prototype the design. Table 10.1 shows synthesis results in terms of logic utilization, DSP blocks, total registers, and maximum frequency. The results show that the M-L-based multicorrelator achieved the best results in most cases whereas the DF can be considered as the worst design approach. As previously explained, the PPDF architecture consumes more resources due to the use of more register elements. The M-L approach drastically reduces logic utilization due to mitigating substantial DSP blocks, registers, etc. In this case study, the M-L-based multicorrelator preserved more than 75 % logic elements as well as 78 % reduction in total registers in comparison with MAC-based architectures. All MAC-based architectures use 512 DSP blocks (32 % of the total) because of the multiplication purposes whereas the M-L-based design does not require any DSP block. The maximum frequency is approximately similar in all architectures, excluding the DF architecture. The M-L-based design achieved the highest frequency between all configurations. Power dissipation analysis is reported based on the results given by Quartus-II PowerPlay Power Analyzer Tool. The analyzer directly reads the SAIF generated by the ModelSim for the multicorrelator running at 50 MHz. The static probability and toggle rate are calculated based on the generated VCD file for detecting a packet

Table 10.1 Synchronizer synthesis results

	TDF	DF	PDF	PPDF	ML
Logic utilization (ALMs)	12,483	12,490	7504	14,611	1876
Total DSP blocks	512	512	512	512	0
Total registers	16,378	16,883	8873	28,037	2886
Max. freq. (MHz)	238	68	88	240	257

Table 10.2 Thermal power dissipation (mW)

	TDF	DF	PDF	PPDF	ML
Total	1436.82	3371.45	1286.90	1138.75	937.15
Dynamic	344.15	200.60	265.90	160.10	9.40
Static	1070.05	1062.70	987.10	955.80	951.50
I/O	22.60	2108.15	33.90	22.85	26.20

Table 10.3 Thermal power dissipation by hierarchy (mW)

	TDF	DF	PDF	PPDF	ML
Memory block	0.78	0.81	1.21	0.80	0.79
Threshold block	0.01	0.01	0.01	0.01	0.01
Controller block	0.02	0.02	0.02	0.02	0.02
FIR filter block	230.20	165.55	192.40	128.80	4.20

Table 10.4 Cell power consumption (mW)

	TDF	DF	PDF	PPDF	ML
DSP block	216.50	130.4	157.75	95.2	0
Combinational cell	76.60	10.80	51.80	1.3	1.85
Register cell	2.10	30.30	10.65	32.65	2.45
I/O	25.1	2090.10	15.80	4.25	7.45

after the boot-up. Table 10.2 summarized the estimated results. The DF architecture has the most power dissipation rather than the other implementations due to the long critical path through its I/O. The dynamic thermal power is noticeably decreased in M-L-based approach. It is generally due to the fact that the M-L architecture mitigates massive number of complex multiplications. Furthermore, the M-L-based design saves more than 94 % of the dynamic power consumption in comparison with the PPDF form. The PPDF architecture is the lowest power-hungry architecture in MAC-based designs.

Dynamic power dissipation by hierarchy is reported in Table 10.3 for different architectures. The MAC operator unit is the most power-hungry block in MAC-based architecture while the controller block as well as the decision maker are the least power consumer, respectively. The power dissipation was reduced due to employing a sign bit correlation instead of 16-bit multiply accumulate operations. Table 10.4 depicts the power dissipation in each cell. The results show that the M-L-based multicorrelator achieves a better result in comparison with MAC-based architectures. The reason why the DF has a massive power dissipation can be explained due to the long critical path which exist in the I/O.

10.6 Conclusion

This chapter discussed about fundamental principles with regard to synchronization issues in Non-Contiguous Orthogonal Frequency Division Multiplexing (NC-OFDM) systems. The architecture of an OFDM transceiver as well as an NC-OFDM transceiver were explored. In contrast to the OFDM system where the synchronization is based on time-domain preamble, an NC-OFDM receiver should regenerate the time-domain representation of the preamble in frequency-domain. Following by, the proposed synchronization data flow as well as the state of the art in synchronizer explained. The main idea was to perform spectrum sensing along with packet detection by a single multicorrelator. The second idea was to segment the supreme spectrum into several subchannels and employ several multicorrelator (instead of searching a wide spectrum). The core content of the synchronizer was based on MAC operations which were quite similar to FIR filters. The MAC operator unit implemented with different structures including Direct Form, Transposed Direct Form, Parallel Direct Form, Pipelined-Parallel Direct Form, Multiplier-Less architectures. The synthesis results inferred that the Multiplier-Less-based multicorrelator had better performance in terms of maximum frequency, silicon area, dynamic power consumption, etc., compared to other configurations. On the other hand, the worst case architecture was Direct Form in all aspects in our case study while there was a trade-off for remaining architectures. As experiences showed, the simplest MAC-based architecture in terms of implementation belonged to Transposed Direct Form and Direct Form, whereas the most complex ones were Pipelined-Parallel Direct Form and Pipelined Direct Form, respectively.

References

1. Acharya, J., Viswanathan, H., Venkatesan, S.: Timing acquisition for non-contiguous OFDM-based dynamic spectrum access. In: 3rd IEEE Symposium on New Frontiers in Dynamic Spectrum Access Networks (DySPAN), Chicago, IL, pp. 1–10 (2008)
2. Airoldi, R., Nurmi, J.: Design of a matched filter for timing synchronization. In: Conference on Design and Architectures for Signal and Image Processing (DASIP), pp. 247–251, 8–10 October 2013
3. Bobrowksi, K.M.: Practical implementation consideration for spectrally agile waveforms in cognitive radio. Ph.D. thesis, Worcester Polytechnic Institute (2009)
4. Chiueh, T., Tsai, P., Lai, I.: Baseband Receiver Design for Wireless MIMO-OFDM Communications, 2nd edn., p. 346. Wiley-IEEE Press, Singapore (2012)
5. Diaz, I., Wilhelmsson, L., Rodrigues, J., Lofgren, J., Olsson, T., Owall, V.: A sign-bit autocorrelation architecture for fractional frequency offset estimation in OFDM. In: Proceedings of IEEE International Symposium on Circuits and Systems (ISCAS), pp. 3765–3768, 30 May–02 June 2010
6. Dutta, A., Saha, D., Grunwald, D., Sicker, D.: Practical implementation of blind synchronization in NC-OFDM based cognitive radio networks. In: Proceedings of the 2010 ACM Workshop on Cognitive Radio Networks, pp. 1–6 (2010)
7. Feng, S., Zheng, H., Haiguang, W.; Jinnan, L., Zhang, P.: Preamble design for non-contiguous spectrum usage in cognitive radio networks. In: IEEE Wireless Communications and Networking Conference (WCNC), pp. 1–6, April 2009
8. Ghosh, D., Sharma, D., Aziz, A.: A novel low area and high performance programmable FIR filter design using dynamic random access memory. In: 48th Midwest Symposium on Circuits and Systems, vol. 2, pp. 1477–1480, 7–10 August 2005
9. Huang, B., Wang, J., Tang, W., Li, S.: An effective synchronization scheme for NC-OFDM systems in cognitive radio context. In: IEEE International Conference on Wireless Information Technology and Systems (ICWITS), pp. 1–4, 28 August–03 September 2010
10. Li, L., Qu, D., Jiang, T., Ding, J.: Design of LDPC codes for non-contiguous OFDM-based communication systems. In: IEEE International Conference on Communications (ICC), pp. 4712–4716, 10–15 June 2012

11. Liu, J., Feng, S., Wang, H.: Comb-type pilot aided channel estimation in non-contiguous OFDM systems for cognitive radio. In: 5th International Conference on Wireless Communications, Networking and Mobile Computing (WiCom), pp. 1–4, 24–26 September 2009
12. Mahesh, R., Vinod, A.P.: New reconfigurable architectures for implementing FIR filters with low complexity. IEEE Trans. Comput. Aided Des. Integr. Circuits Syst. **29**(2), 275–288 (2010)
13. Mitola, J.: Cognitive radio for flexible mobile multimedia communications. In: IEEE International Workshop on Mobile Multimedia Communications (MoMuC '99), pp. 3–10 (1999)
14. Pham, T.H., Fahmy, S.A., McLoughlin, I.V.: Low-power correlation for IEEE 802.16 OFDM synchronization on FPGA. IEEE Trans. Very Large Scale Integr. (VLSI) Syst. **21**(8), 1549–1553 (2013)
15. Pun, M., Morelli, M., Jay Kuo, C.-C.: Multi-Carrier Techniques for Broadband Wireless Communications. A Signal Processing Perspective, vol. 3, p. 257. Imperial College Press, London
16. Qu, D., Ding, J., Jiang, T., Sun, X.J.: Detection of non-contiguous OFDM symbols for cognitive radio systems without out-of-band spectrum synchronization. IEEE Trans. Wirel. Commun. **10**(2), 693–701 (2011)
17. Rajbanshi, R., Wyglinski, A.M., Minden, G.J.: An efficient implementation of NC-OFDM transceivers for cognitive radios. In: 1st International Conference on Cognitive Radio Oriented Wireless Networks and Communications, pp. 1–5, 8–10 June 2006
18. Saha, D., Dutta, A., Grunwald, D., Sicker, D.: Blind synchronization for NC-OFDM - when "Channels" are conventions, not mandates. In: IEEE Symposium on New Frontiers in Dynamic Spectrum Access Networks (DySPAN), pp. 552–563, 3–6 May 2011
19. Shamani, F.: Design of a flexible timing synchronization scheme for cognitive radio applications. M.Sc. thesis, Tampere University of Technology, p. 75 (2013)
20. Shamani, F., Airoldi, R.; Ahonen, T.; Nurmi, J.: FPGA implementation of a flexible synchronizer for cognitive radio applications. In: Conference on Design and Architectures for Signal and Image Processing (DASIP), Madrid, pp. 1–8 (2014)
21. White Space Database Administrators Guide, [WWW]: Federal Communications Commission (2013). Available at http://www.fcc.gov/encyclopedia/white-space-database-administrators-guide. Accessed on 29 October 2013
22. Won, J.Y., Kang, H.G., Kim, Y.H., Song, I., Song, M.S.: Fractional bandwidth mode detection and synchronization for OFDM-based cognitive radio systems. In: IEEE Vehicular Technology Conference (VTC), pp. 1599–1603, 11–14 May 2008
23. Wyglinski, A.M.: Effects of bit allocation on non-contiguous multicarrier-based cognitive radio transceivers. In: 64th IEEE Vehicular Technology Conference, pp. 1–5, 25–28 September 2006
24. Wyglinski, A.M., Nekoyee, M., Houauthor, T.: Cognitive Radio Communications and Networks, p. 714 . Academic, Oxford (2010)
25. Xing, Y., Kushwaha, H., Subbalakshmi, K.P., Chandramouli, R.: Codes and games for dynamic spectrum access. In: Arsalan, H. (ed.), Cognitive Radio, Software Defined Radio, and Adaptive Wireless Systems, chap. 6, p. 163. Springer, Dordrecht (2007)
26. Zhou, X., Qiu, R.: An adaptive synchronization algorithm for non-contiguous OFDM cognitive radio systems. In: International Communication Conference on Wireless Mobile and Computing (CCWMC), pp. 102–106, 14–16 November 2011
27. Zhou, Y., Pagadarai, S., Wyglinski, A.M.: Feasibility of NC-OFDM transmission in dynamic spectrum access networks. In: IEEE Military Communications Conference (MILCOM), pp. 1–5, 18–21 October 2009

Chapter 11
Towards Adaptive Cryptography and Security with Software Defined Platforms

Antti Hakkala, Jouni Isoaho, and Seppo Virtanen

11.1 Introduction

Security and privacy are essential components of modern communication systems, of which the Internet is arguably the primary medium for most of us. The Internet was originally designed on the base assumption that all users will comply with standardized protocols and rules, but development in the last two decades unfortunately shows that this assumption can be challenged on several different levels.

First, the rise of cybercrime in the first decade of the twenty-first century has provided evidence that there are adversaries willing to compromise any and all assumptions of security, and they do it for profit. Cybercrime in itself has grown to become a multi-billion business, and their methods range from social engineering to capitalizing on existing vulnerabilities in software and hardware [38]. In the battle against cybercrime the main focus should be in building security in the systems and maintaining the ability to adapt to discovered vulnerabilities, thus swaying the balance in the continuous security arms race to the defenders' benefit.

Second, we can recognize another even more pressing driver for advances in security and privacy: the advent of global network surveillance. Performed by the most powerful intelligence organizations in the world, global network surveillance has emerged as one of the main threats against security and privacy on

A. Hakkala (✉) • S. Virtanen
Communication Systems Laboratory, Department of Information Technology,
University of Turku, Turku, Finland
e-mail: Antti.Hakkala@utu.fi; seppo.virtanen@utu.fi

J. Isoaho
Department of Applied Physics, Laboratory of Electronics and Information Technology,
University of Turku, Turku, Finland
e-mail: jouni.isoaho@utu.fi

W. Hussain et al. (eds.), *Computing Platforms for Software-Defined Radio*,
DOI 10.1007/978-3-319-49679-5_11

the Internet [2, 47]. Ordinary criminals target Internet users—whether businesses or individuals—stealing their data and credit card details by leveraging existing vulnerabilities in software and hardware. Intelligence agencies have taken a more disruptive approach. By undermining security protocols and standards these organizations have succeeded at making Internet communication less secure. When a vulnerability is detected in software, it is patched with a software update. If the flaw is in a hardware implementation of a standardized algorithm, and that particular algorithm in itself is found to be compromised, the flaw cannot be fixed without changing the hardware itself. It is hard to convince consumers to upgrade their hardware because of such a security flaw. It is in most cases also not cost effective in the consumer device segment. In more specialized or mission critical hardware, however, replacement can be worth serious consideration. This can be true even in cases where consumer devices would be left in a vulnerable state through their lifetime, e.g., WLAN access points supporting only outdated encryption options such as WEP.

Traditionally different types of communication have required specific terminal equipment and infrastructure, all specific to a single application. Placing phone calls typically required a phone and either a wired or wireless telephone connection. Browsing websites and reading email required a computer and a network connection. Watching television required a television set, and so on. Nowadays the situation is significantly more complex. While we do have handheld battery-powered devices capable of handling all of these communication scenarios, the pressure for security in communications has not yet reached the point where all communications regardless of type and platform are encrypted by default. This would place even more requirements on multipurpose devices such as smartphones, but the current generation devices are already more than capable of handling this. One significant caveat is whether better security and privacy should come at the cost of end user experience. In a consumer product this could be considered unacceptable. The emergence of this trend can already be seen from certain key stakeholder reactions to mass Internet surveillance [21, 42, 49]. While it can even be considered idealistic in the current state of events, we hope to see a demand for security and privacy that will result in security and encryption permeating all tiers of hardware devices, even those with insufficient resources to implement these features.

Future communication systems need to support a variety of signal processing algorithms, radio interfaces, and communication and security protocols in the same device, with the capability to adapt to changing situations. We already observed the present trend towards provision of high processing speed for a wide variety of applications by handheld battery-powered devices in [6]. We can see this trend realized in current smartphones and software ecosystems. With the advent of Internet of Things (IoT) and Wireless Sensor Network (WSN) systems, smaller devices with less computational capacity than current handheld devices will have to fulfill the same requirements for secure communication as present handheld devices. We see that a new trend driven by data security and privacy concerns will require more security-related functionality from all devices, regardless of their position in the communication chain—whether they are measuring, processing, or just transmitting data. For example, sensors in healthcare IoT must use strong methods to secure medically sensitive data, and military sensor network applications must adapt to changing environments and corresponding security requirements.

The field of embedded security is under intense research. Some challenges, as identified in [27], are, e.g., securing the data content and data transportation within the device, data exchange with the recipient device at the other end of the wireless link, and data transportation between end-to-end applications. These are vital capabilities and indispensable built-in features for any communication device that aims to serve as a credible solution for current and future communication applications. Software Defined Radio (SDR) and Open Wireless Architecture (OWA) have previously been established as potential design paradigms to address the needs for higher system integration and multiple communication interface support.

The objective of this chapter is to specify a design methodology and process as a framework for enabling resource-constrained devices both to implement necessary security features and to preserve processing power to the actual target application. We refer to this as the Software Defined Secure Communications (SDSC) framework. In this chapter we examine the augmentation of the SDR and OWA concepts with built-in security support, and present our approach to building security in the framework. We identify and analyze the requirements set by modern security algorithms and applications to future communication systems, and further specify a design methodology which leverages SDR principles, adaptive cryptography as well as software and reconfigurable hardware for such systems. As a result of the analysis, we discuss the potential benefits of this approach in the future, where IoT and sensor networks are ubiquitous. Essentially we see our method as a flexible, programmable, and security oriented design framework for security enabled SDR applications that can be used to create proprietary hardware and software designs, which can later be adapted to several different functions with software. We also see benefits of our method in shortened time-to-market and potential cost savings in increased volume and customization options.

In our earlier studies we have introduced the programmable and parameterizable TACO (Tools for Application-specific hardware/software codesign) hardware platform [51, 52] for protocol processing (PP) applications, and proposed a digital signal processing (DSP) extension to it [41], making it possible to run PP and DSP tasks in parallel on the same programmable processor with optimized execution units. SDR capabilities were introduced to the architecture in [1].

The chapter continues with an overview of the TACO architecture and concept in Sect. 11.2, where we also discuss the PP and DSP features of the architecture. In Sect. 11.3 we give an overview of our SDR research on the architecture. In Sect. 11.4 we provide the necessary background on cryptography and its relevance to security protocols. We describe our approach to accelerating cryptographic protocols by identifying and accelerating their underlying primitive operations, and discuss the use of Transport Triggered Architecture (TTA) based processors in cryptography. In Sect. 11.5 we present our design methodology and flow for the SDSC platform and discuss cryptography reconfiguration approaches and motivations for different target applications. A preliminary case study and a simulation experiment on incorporating RSA support onto our platform are presented, with results that are encouraging but limited in scope. In Sect. 11.6 we discuss future work in the form of potential target applications and implementation scenarios for our approach, and begin to outline the challenges we face in the future of communication systems. Finally, in Sect. 11.7 we provide our concluding remarks.

11.2 TACO Platform for Application Domain Specific Programmable Acceleration

The TACO hardware platform is based on the Transport Triggered Architecture paradigm [9, 50] and is aimed at developing application domain specific programmable processors. A processor customized for a specific application does not only outperform a general purpose core, but is also expected to consume less power and circuit area. Application-specific hardware can reduce the computational load associated with complicated operations that are cumbersome when executed on traditional processors. A customized processor can meet the desired throughput requirements at a lower clock speed, thus reducing energy expense. Unlike a general purpose execution unit, a highly tailored processor will not implement any extra functionality, except for the subset needed for the desired application domain. Because of the TTA based architecture, TACO protocol processors are programmable and configurable, providing dedicated hardware to deal efficiently with application-specific tasks. As seen in Fig. 11.1, a TACO processor consists of a set of functional units (FUs) connected by an interconnection network of one or more buses. There may be multiple FU instances of the same type in a TACO processor, allowing parallel execution of respective operations. This resource redundancy, depending on circumstances, may serve either to speed up the execution of a particular single application or to provide real multitasking for a set of simultaneously run applications. The interconnection network is composed of one or more buses, supported by an interconnection network controller. The interconnection network is presented in more detail in Fig. 11.2, where the individual connections between FUs and buses are shown. The number of buses directly limits instruction level parallelism, as it determines the maximum number of simultaneous data moves. As TACO is a modular architecture, FUs are designed separately and are provided standard interfaces to the interconnection network. This also makes it easy to automate the design process to a high degree. Each FU implements a set of functions, and the final configuration is defined by the choice of appropriate FUs to the target application. FUs can be modified internally as long as they provide the same interface to the sockets connecting them to the interconnection network. This makes module re-usability straightforward and simple, and it is an integral part of the TACO hardware platform and design methodology.

The performance of the architecture can be scaled up by adding extra FUs, buses, or by increasing the capacity of the data transports and storage. Increasing the number of existing FUs provides more performance for parallel and pipelinable tasks, while adding FUs with new functions in turn provides support for new

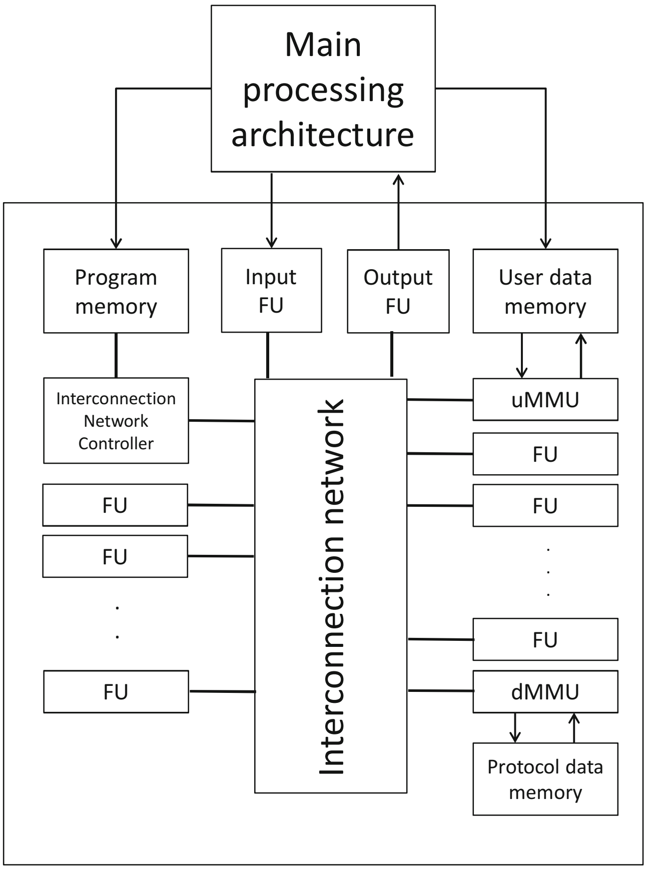


Fig. 11.1 General TACO processor architecture

application types. Performance of TTA processors is at least equal to that of their conventional or DSP counterparts (i.e., processors of different types but with functionally and quantitatively equivalent capacity). According to [24, 25], this performance is also achieved at far lower cost, which encourages our further development of optimized functionality on TACO.

TACO is programmed by only one type of instruction, *move*, which specifies independent data transports on each of the defined buses. In contrast to traditional processors, data transports trigger operations, and not vice versa. The operation of the processor occurs as a side-effect of the transports between functional units, transparently to the control unit and its instructions. The architecture allows for one processor cycle to fit several bus transports. The interconnection network controller implements the data transports on the buses. The controller uses a program memory, from which it fetches instructions, splits them into sub-instructions, and dispatches each sub-instruction onto the corresponding bus. Each move sub-instruction has

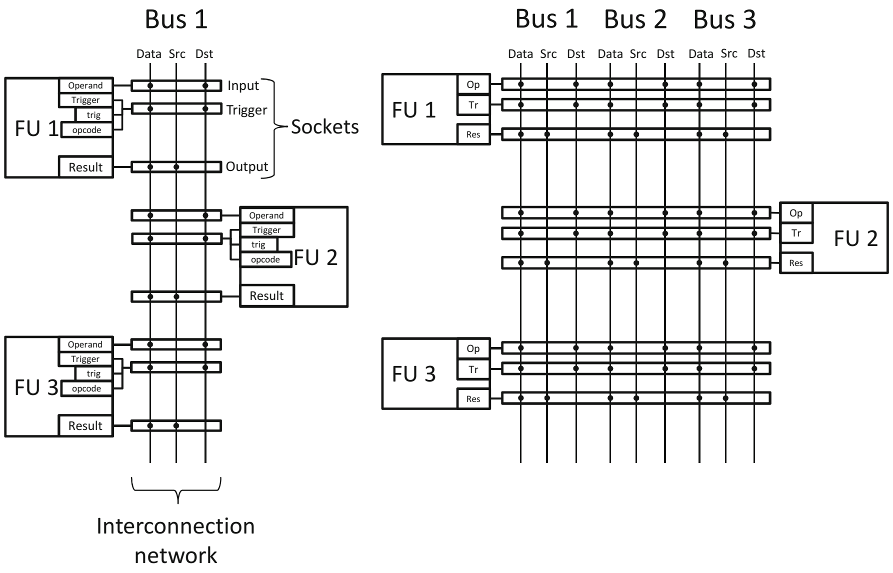


Fig. 11.2 TACO processor interconnection network. Each interconnection bus consists of a data bus, a source address bus, and a destination address bus. *Left:* Three FUs and one bus: only one data transport per clock cycle. *Right:* Three FUs and three buses: three data transports per clock cycle are possible, so each functional unit can be triggered to function on each clock cycle

a source and a destination field, and the data is carried accordingly from one FU register to another on the respective bus. When adding new FUs to a design, the instruction format does not change as long as the existing FUs are addressable by the length of source and destination addresses, and unless more parallelism is required. The excellent pipelining capacity of TTA, together with the emphasis on moving data, makes it a good platform for data-intensive applications. It is exceptionally well suited to protocol processing and for all communication-related tasks in general.

11.2.1 Protocol Processing Application Domain

The first application domain we have addressed in the development of TACO was protocol processing (PP). The following discussion on building the TACO platform for the PP application domain is based on our earlier research in [50–52].

In order to develop customized devices for the PP application domain we carefully studied commonly used protocols and protocol processing applications. Special emphasis was placed on finding functionality that varies very little from one protocol to another or remains virtually the same across a set of protocols. Studies by Jantsch et al. [29] and Virtanen et al. [50, 51] identified characteristic functionality that, if parameterized, is generalizable to several protocols instead of just one specific protocol. These studies identified, for example, the need for bit pattern matching and masking, substring replacement in bit strings, Boolean comparisons, counters, and checksum calculations. They are also functionality that can be parameterized. One finding was also that protocol processing can be implemented using only unsigned arithmetic (i.e., there is no need for managing negative values), making hardware implementation considerably simpler. The emphasis in these studies was on analyzing protocols that can be regarded as layer 1–3 protocols (physical, data link, and network layer) in the OSI reference model. In these layers the protocols are not end-to-end protocols, but require intermediate stations (e.g., repeaters, bridges, switches, routers, etc.) between the source and destination terminals. Thus, these protocols present a clear need for application-specific hardware systems in addition to application software, whereas the end-to-end protocols in OSI layers 4 and above are often completely implemented as software running on networked workstations. However, the higher layer end-to-end protocols can also potentially take advantage of application-specific processors, particularly in mobile devices and low-power sensor networks. In terms of the work presented in this chapter, we argue that we cannot concentrate solely on the three lowest OSI layers, as security-related protocols reside on the upper layer of the OSI model. Especially important are protocols providing end-to-end encryption, an essential part of information security.

11.2.2 TACO Extension to Digital Signal Processing Domain

In [41] we explored possibilities for augmenting our hardware platform to support DSP operations, in addition to the pre-existing support for protocol processing. As our first step towards bridging the gap between the protocol processing and DSP domains, we enhanced the TACO hardware platform to support finite impulse response (FIR) filtering. To enable efficient filtering we had to implement the multiply-and-accumulate (MAC) operation typical to many DSP applications. The approach we chose required remarkably few modifications to our hardware platform. The process of designing and implementing a new FU for the multiply-and-accumulate (MAC) operation demonstrated the flexibility of our hardware platform in terms of adding support for new functionality even across the boundaries of supported application domains. As our initial starting point in the design of the DSP domain MAC FU, we used an existing protocol processing FU, namely the Internet check sum calculation unit presented in [51].

Fine-tuning of the TACO modules allowed us to enhance our protocol processor architecture with support for some DSP applications without loss of performance in either domain, and at a very low cost. We determined that the MAC FU consumes about 5–10 % of the total chip area of a typical TACO processor implementation. With a single MAC FU hardware implementation we were able to specify multiple system implementation schemes with different optimization factors for the target application. The conducted analysis suggests that the obtained enhancement of TACO platform allows efficient parallel execution of application software requiring both protocol processing and DSP operations, and this was also demonstrated by experimental results. This work laid foundation for further studies on parallel execution of operations from DSP, protocol processing, and other domains on our hardware platform. With these encouraging results we proceed to the software defined radio domain.

11.3 TACO as a Platform for Software Defined Radio

In [1] we took the cross-domain customization of TACO further and explored both methodological and technical aspects of adding support for Software Defined Radio domain to the architecture. Our goal was to deliver a novel software defined approach for designing and implementing common baseband processing tasks, with focus on exploring the algorithmic and architectural design spaces of 3G and 4G systems. Specifically, we identified computational and geometric structures shared by diverse coding schemes, services and hardware platforms related to SDR domain, and integrated physical layer support for them in extensible hardware. While the primary goal was to enhance the TACO platform with the features of a programmable SDR enabled processor, in this process the methodology of extending the TACO platform to support new application domains was also realized. This process is done in two phases. First, the initial design is done using standard design principles and methods for the original target application. After this the design is extended to support a new set of requirements by identifying target components in the design and modifying them to fulfill the requirements from the new application domain.

The approach incorporates control structures, component abstractions and parameterization, and architectural optimization into a system design process. We found the proposed platform and design methodology to be very suitable for SDR domain applications, as they have strict requirements on power consumption, chip size, processing speed and scalability for other existing and future applications and standards. Since the TACO platform has the scalability and modularity required by SDR applications, the design framework allows the designer to focus on the internals of the required hardware components due to the well-defined structural entities of the hardware platform. A key benefit of our approach is the support for making FUs definable in software. Once a new radio application has been mapped and partitioned to existing FUs, the designer typically needs to make only minor changes or additions to the internal implementation of existing units. In many cases, modifying the external interfaces of the functional units is not necessary when adding support for novel functionality. These characteristics result in a shortened turnaround time and less design effort in the process of upgrading the platform to support new (radio) applications. On the architectural abstraction level, the designer typically does not have to modify the platform implementation, except to choose

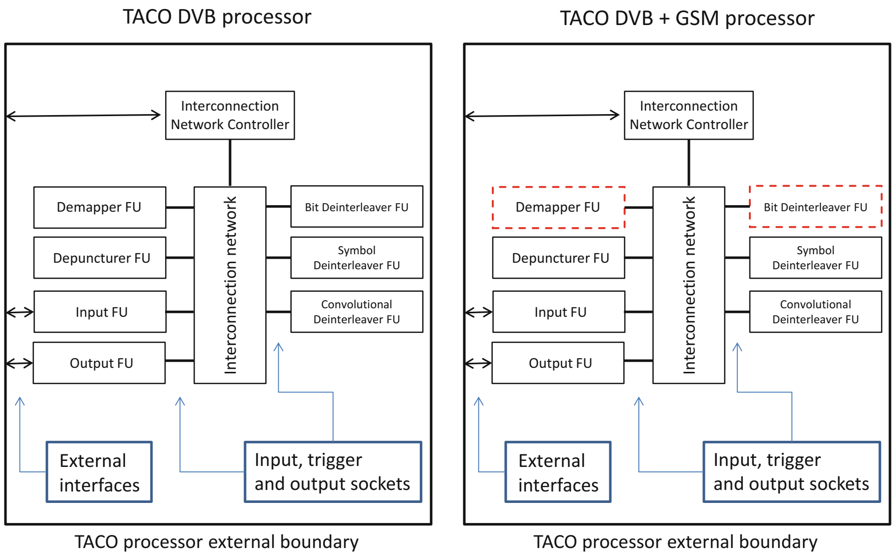


Fig. 11.3 TACO DVB + GSM processor

the utilized level of execution parallelism and of data transport capacity. Parallel execution of operations can be realized by adding several FUs of the same type to a design, and data transport capacity is affected by the number of buses and the interconnection network.

In the case study presented in [1], we designed a TACO platform for Digital Video Broadcasting (DVB) and then modified the system to additionally support GSM. The platform implementation required 4000 lines of VHDL code. As expected, we found that extending the platform from supporting only DVB to also support GSM did not considerably increase the amount of code, as most additions were needed into specific processes inside the FU implementations. Only two FUs needed to be enhanced for GSM support: demapper and bit deinterleaver. These FUs are marked with a dashed line in Fig. 11.3. We determined in our study that the cost of extending one functional unit to support additional, software definable functionality from another standard was an area increase of only about 10–15 % and a power increase of 23–26 %, showing that these existing units already supported most of the new functionality. At the processor level, the GSM extension produced only 2.4 % of total processor area and 2.7 % of overall power consumption. The extension required 4 days of hardware design and verification, 2 days for each FU that needed modifications.

11.4 Towards Software Defined Secure Communication: Multi-Domain Integration for Secure Network Applications

In this section we will extend the TACO platform and design methodology towards a comprehensive communications solution. The *Software Defined Secure Communication* (SDSC) platform will provide support for all aspects of communication, and we start outlining the platform by examining necessary building blocks for secure communication systems, examining the characteristic functionality found in security algorithms, and discussing on how to adapt to a changing security environment.

11.4.1 Cryptography Fundamentals

Cryptography is a core element of security. All secure protocols and communication standards include methods for guaranteeing data confidentiality, integrity, and authenticity. They are the essential cornerstones of information security, so we will have to extend the TACO platform to handle these vital areas. Confidentiality is provided by encryption, while integrity and authenticity are provided by error correction, cryptographic hash functions, Message Authentication Codes (MAC), and digital signatures. The choice of a cryptographic solution for a target application is not trivial, as there are many choices, constraints, and metrics for the designer to consider. The communication cipher suite for an IoT healthcare solution will differ significantly from a smartphone due to differences

Cryptographic ciphers can be broadly divided into three groups: private key, public key, and cryptographic hash algorithms [46]. Private key algorithms use the same key for encryption and decryption, while public key algorithms have two keys: public key for encryption and private key for decryption. Hash algorithms compress an arbitrary length input to a fixed length output, and attempt to provide unique outputs for each input. All three types of algorithms are important for secure communication protocols. Public key algorithms are used for initiating a secure communication channel over an insecure medium. Keys for a private key algorithm are then exchanged over this secure channel, and that private key algorithm is used for bulk data encryption. The integrity of communications can be verified using secure hash functions, which can provide proof that a message has not been altered in transition. Cryptographic hash functions are also often used to derive session keys from master encryption keys, using key derivation schemes such as PBKDF2,[1] and bcrypt [43].

[1]Defined in IETF RFC 2898, https://tools.ietf.org/html/rfc2898

11.4.1.1 Accelerating Cryptography

There are several different approaches to accelerating cryptography algorithms [34]. Because of large operands and complicated operations that are ill-suited for standard computer architectures, software cryptography implementations tend to be slow when compared to hardware. Thus the natural approach is to build cipher-specific hardware. Dedicated hardware solutions are orders of magnitude faster than software implementations, but they have some significant drawbacks. Hardware implementation of an algorithm cannot be changed later, should the need arise, thus locking us to that algorithm implementation. Reconfigurable hardware such as Field Programmable Gate Arrays (FPGA) has been used to solve this issue.

In [17] we have surveyed the methods for accelerating different cryptographic algorithms. Exponentiation operations are very common in cryptography, and combined with modular arithmetic, they provide the backbone of ciphers such as RSA [44]. Since these operations can be regarded as bottlenecks, accelerating them will boost the performance. Known efficient multiplication algorithms and different number representations such as Montgomery representation [37] for modular arithmetic can be leveraged for significant gains. The Chinese Remainder Theorem (CRT) can be applied specifically to RSA, providing up to four times faster operation [19]. Finite field arithmetic is crucial to Elliptic Curve Cryptography (ECC) [32, 36]. ECC operates in finite prime or extension fields. In prime fields, operands are reduced modulo a prime, and in extension fields the reduction is done modulo an irreducible polynomial. A representation of the arithmetic hierarchy for ECC is presented in [31], where the relationship between the arithmetic operations and underlying primitives is shown. Elliptic curve operations are all based on modular arithmetic, and this makes modular arithmetic an essential underlying operation for ECC as well.

Similarly, finite field arithmetic is also essential for AES [39], in which calculations are done in the binary extension field $GF(2^8)$. In some cases leveraging different coordinate systems for representing field elements can eliminate the need for computationally costly operations. As an example of reuse of cryptographic primitives, AES calculations can be accelerated with an Elliptic Curve instruction set designed to accelerate calculations in finite fields of characteristic 2^n [48]. Another example is the use of the Intel AES instruction set [5] to accelerate AES-based SHA-3 hash function candidates, an approach which leverages existing hardware on processors to accelerate the execution of structurally similar algorithms. A summary of essential operations in various cryptography algorithms is presented in Table 11.1.

11.4.2 Previous Work on TTA and Cryptography

As far as we are aware, there is very little literature on the use of transport triggered architecture based systems in the cryptography domain. TTA has mostly been

Table 11.1 Summary of essential features and associated methods in cryptography

Base operation	Techniques and methods
Fast exponentiation	Fast multiplication algorithms, windowing methods (see, e.g., [15])
Fast modular arithmetic	Montgomery [37] and Barrett [3] modular reduction, and derivatives [20, 22, 53], Itoh-Tsujii inversion [28], repeated squarings [30], Extended Euclidean Algorithm, Chinese Remainder Theorem
Fast point scalar multiplication	Various coordinate systems, fast multiplication algorithms (see, e.g., [18])
Fast substitution, rotation, and permutation	Operation-specific Instruction Set Extensions [4], hardware implementation

used for protocol processing, and so far the available literature clearly reflects this. Hardware implementations of IWEP, RC4, and 3DES all based on TTA are analyzed in [24]. The paper aims to explore Instruction Level Parallelism (ILP) for the previously mentioned algorithms, finding that parallelism can be exploited in some limited scope. The suitability of TTA for efficient AES encryption is studied in [26], where AES on different platforms, including TTA based processor, for wireless sensor networks was examined in a survey study. A coprocessor for elliptic curve cryptography acceleration based on TTA is presented in [35]. Here, the presented system is targeted at the Chinese standard ECC algorithm SM2. Their implementation was capable of scalar multiplication in 3 ms at 80 MHz for a 192 bit elliptic curve. A TTA hardware implementation of RSA based on the Chinese Remainder Theorem is presented in [16], where the presented implementation is capable of decrypting RSA at the rate of 106 kbps at 100 MHz. A TTA based hardware solution for RSA key expansion is presented in [23], where the presented hardware is capable of generating three 1024 bit RSA keys per second at 100 MHz, when implemented in VHDL and synthesized in a 0.18 μm process.

Our approach with the TACO platform had been primarily directed towards protocol processing, but both the advantages of the TTA based platform and the requirements of future networks support extending the platform to the cryptography domain. The development of communication systems towards IoT and sensor networks also put pressure on providing significant cryptography resources with flexible cipher suites and broad communication protocol support on hardware.

11.4.3 The Pyramid Model of Security

Security design can be generalized as a pyramid with different layers representing abstraction levels of security systems [45]. At the highest abstraction level is the security protocol architecture, which contains the protocols used for security

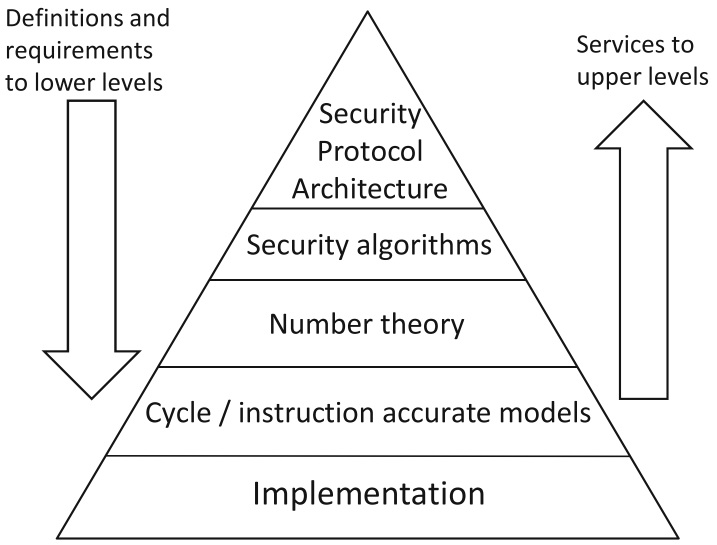

Definitions and requirements to lower levels
Services to upper levels
Security Protocol Architecture
Security algorithms
Number theory
Cycle / instruction accurate models
Implementation

Fig. 11.4 Pyramid model of security. Adapted from [45]

purposes. The next abstraction level contains the actual algorithms such as RSA, DSA, and SHA. The third abstraction level contains the "building blocks" of algorithms, derived from number theory. The fourth abstraction level contains cycle accurate platform independent representations of these algorithms, and the fifth level contains the physical implementation of the algorithms. If approached from a completely different point of view, the security pyramid model can also be seen as a service architecture. It incorporates a distinct customer–service relationship between the layers, where the upper layers issue directives and requirements for the layers below, whereas the latter provide service to the layers above, according to the received requirements [17]. The model is illustrated in Fig. 11.4.

When we consider different protocols and algorithms for our SDSC platform and design methodology, we are mostly concerned with the first, second, and third layer of the security pyramid. These layers constitute the manual part of the design flow, while the fourth layer is handled by the TACO design toolkit. The mapping of the pyramid model to the TACO design flow is examined later in Sect. 11.5.

11.4.4 Adaptive Cryptography for Dynamic Security Dimensions

Security requirements for future communication systems will continue to evolve, and when applications and use cases develop, security features and capabilities must evolve and adapt with them. One source of motivation for strengthening security features is the widespread surveillance of communication networks, affecting everyone on the Internet. This affects a significant part of the world's population and businesses.

The security landscape for IoT is challenging. When all devices can be—and many are—used to transfer sensitive information and all of it is subject to potential eavesdropping by malicious parties, securing these devices is a difficult but necessary task. IoT also brings an additional layer of complexity by making the number of communicating devices grow significantly in the future. Another strong driver are sensor networks, which require robust and adaptive security features as well.

Our chosen approach to this problem is to first recognize the underlying primitive operations for cryptography algorithms and then selecting a representative set of primitives to implement in hardware. In this way we can provide hardware acceleration to the computationally most challenging operations, and also provide flexibility with regard to chosen algorithms and cipher suites, as they are implemented on a higher abstraction level.

Adaptive cryptography has been defined as a cryptographic library from where different implementations of algorithms can be selected to adapt to changes in requirements [10, 11]. The target application in the referred papers was IPSec, for which an FPGA implementation of a large cipher suite was made. We consider this approach to be inadequate in the face of current developments regarding network surveillance and the efforts to undermine cryptography standards by intelligence agencies. With this in mind, to respond to these challenges we will redefine adaptive cryptography as

> adapting to changing cryptography requirements on the fly, without replacing physical hardware, even when the changes warrant using a *completely new* algorithm.

We suggest an approach where the important underlying operations for cryptography are implemented in hardware, and the higher abstraction level is implemented in software. In this manner we will take a performance hit when compared to pure hardware implementations, but we see that this performance hit is acceptable with the gain of better flexibility and adaptability to unforeseen situations.

With adaptive cryptography techniques, we can create systems that are more versatile and less subject to catastrophic failure in case of a serious threat to the underlying security infrastructure and algorithms. We will discuss one particular promising implementation approach in Sect. 11.6 where we discuss the concept of security dimensions.

In Fig. 11.5 we show the landscape of various encryption application areas and targets. Adaptive cryptography targets the application area where resources are constrained but flexibility is required. This application area will see more and more interest in the future, based on the direction of societal development and how communication systems are used. The advent of big data has made nearly everything that is capable of collecting data a viable data source, and this data is communicated through the Internet. At the same time, important societal functions are also being transformed by the Internet, leading to more and more aspects of our lives being

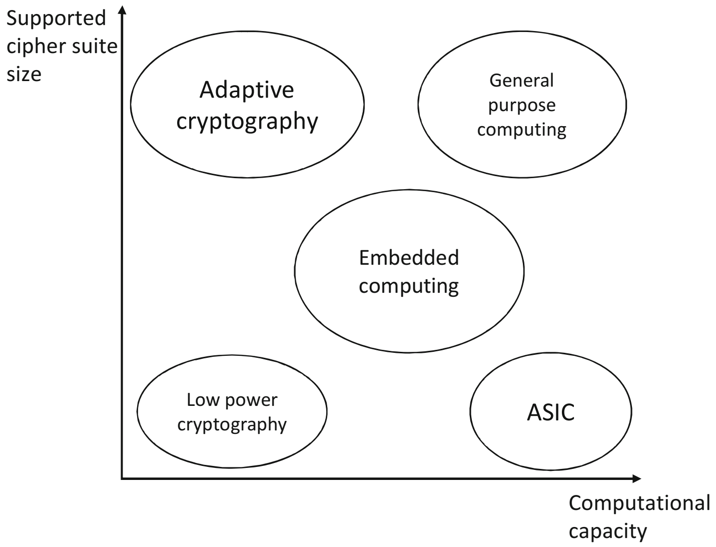


Fig. 11.5 Landscape of different encryption application areas

online. This in turn leads to developments that are not as positive, such as online crime, cyberterrorism, and mass Internet surveillance—all are phenomena that have surfaced strongly in the 2010s. The TACO SDSC platform aims to provide a starting point for building systems capable of answering to these difficult challenges.

11.5 Design Methodology and Flow

In this section we define the design methodology and flow for the SDSC platform. The methodology is built on and extended from the SDR platform methodology proposed in [1]. We see the SDSC concept as a viable platform for next-generation communication devices. Parameterizability, programmability, and run-time reconfigurability are all properties that make designing new hardware and software for emerging applications easier. This also provides better product lifetime, as product platforms are more extensible. Modularity allows reusing blocks both at design phase and at run-time, helps optimizing resource utilization and it is crucial for the design flow itself. This is especially important in energy- and space-constrained environments with high performance demands.

11.5.1 Target Application and Domain Specific Requirement Analysis

Several design stage issues must be taken into account. To identify potential points of application domain specific optimization, we need to identify computationally heaviest portions of the code and find potential sets of functionality that are common throughout the application domain. These are priority targets for acceleration and thus are subject to being broken down into primitives at platform level. The analysis of the target application starts at the pseudo code level, where we start to identify recurring and parallel operations.

With looped operations we must verify that the operations are truly independent from each other. In this case it is possible to accelerate functions by adding additional FUs to the design and calculating each iteration of the operation on an individual FU in parallel. This is demonstrated in Fig. 11.6, where a loop of 3 iterations on a single FU is replaced by three FUs in parallel. For some of our intended applications in the SDSC domain, such as cryptography, repeated operations are often dependent of each other. As an example, it would be intuitive to have a design with several FUs which implement one round of a symmetric cipher such as AES, thus increasing encryption and decryption speed. But because the input to a round is dependent on the output of the previous round in most cipher modes, this approach cannot be taken. One exception to this is the Electronic Code book (ECB) mode, where the rounds are independent, but it is seldom used in any real-world applications because it lacks semantic security [14]. This in combination with the findings in [24] suggests that while parallelism does exist,

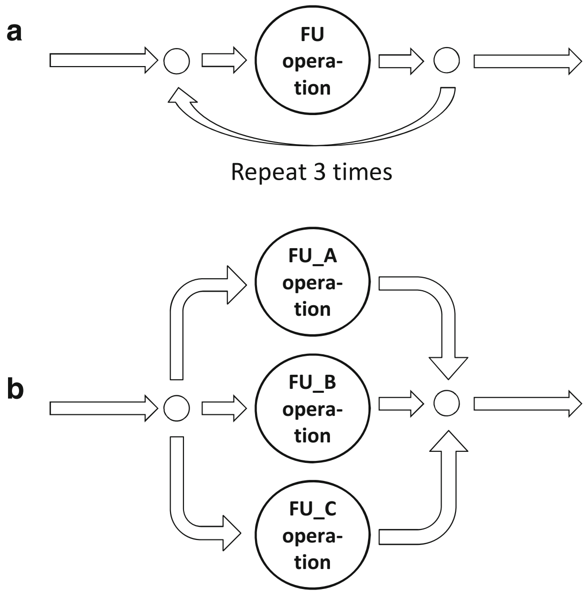


Fig. 11.6 (**a**) Repeated operation with single FU. (**b**) Parallel implementation using several FUs

it is not common enough in cryptography applications. Acceleration of primitives and different arithmetic techniques thus are often better options for optimizing cryptography performance.

Reuse of previously designed hardware instances, whether whole or partial, is a key design goal in this design methodology. This approach provides significant advantage in design time. Using existing design instances that have similar functionality as templates is a practical approach that provides the designer with a baseline design with low initial cost, sparing a lot of time and effort. After template initialization, all necessary functional units are then added to the design. This is based on a careful analysis of the target application and what are the essential operations that need to be implemented on the hardware level.

11.5.2 Design Methodology and Flow

The proposed SDSC design methodology is described in Fig. 11.7. The first step is analysis of the target application, and the result of this analysis is a list of functionality requirements for that application. This makes the methodology a top-down approach. When these key functions have been identified, the necessary algorithms required for implementing them must be chosen. The application domain of the target application will heavily influence this process. It also provides a chance to group similar functions into hardware units and augmenting already existing template components, thus promoting reuse of existing components. If no such functionality is available in the platform's library of existing hardware templates, a new hardware template must be designed for this purpose.

Next, a hardware platform instance for sequential processing is specified. In this instance, only one interconnection bus and one FU of each required FU type are included. This gives us an initial setup that satisfies functional processing requirements of the target application, but processing speed requirements are not typically met at this stage. This process is shown in Fig. 11.8, where the system-level model is mapped to VHDL library components. This sequential instance, along with associated sequential program code, is then simulated, testing for correct operation. Once the functionality is verified, the next stage in the methodology is designer-driven design space exploration. Here, parallel processing capacity is added to the model to meet the processing speed requirements of the target application. The designer typically specifies 2–3 additional hardware instances, varying the number of data transport buses (data transportation parallelism) and the number of FUs of the same type in the hardware instance (FU level parallelism). Both methods increase processing performance. The target application and its computational capacity requirement greatly affects the set of feasible hardware configuration instances. This process is not automatic. Optimization factors derived from power consumption and chip size requirements, as well as current processing performance requirements and even future extensibility are all manually defined by the designer. When we are satisfied with the obtained model we test the optimized

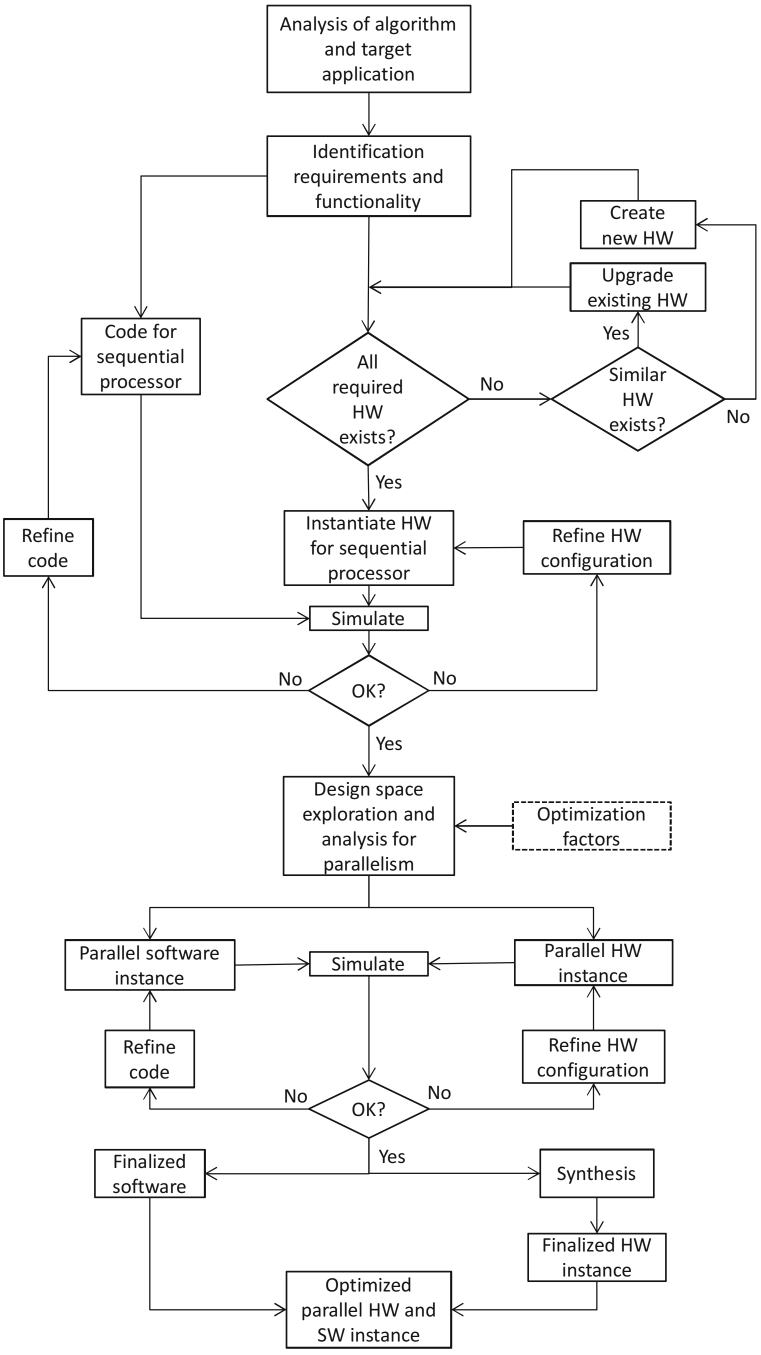
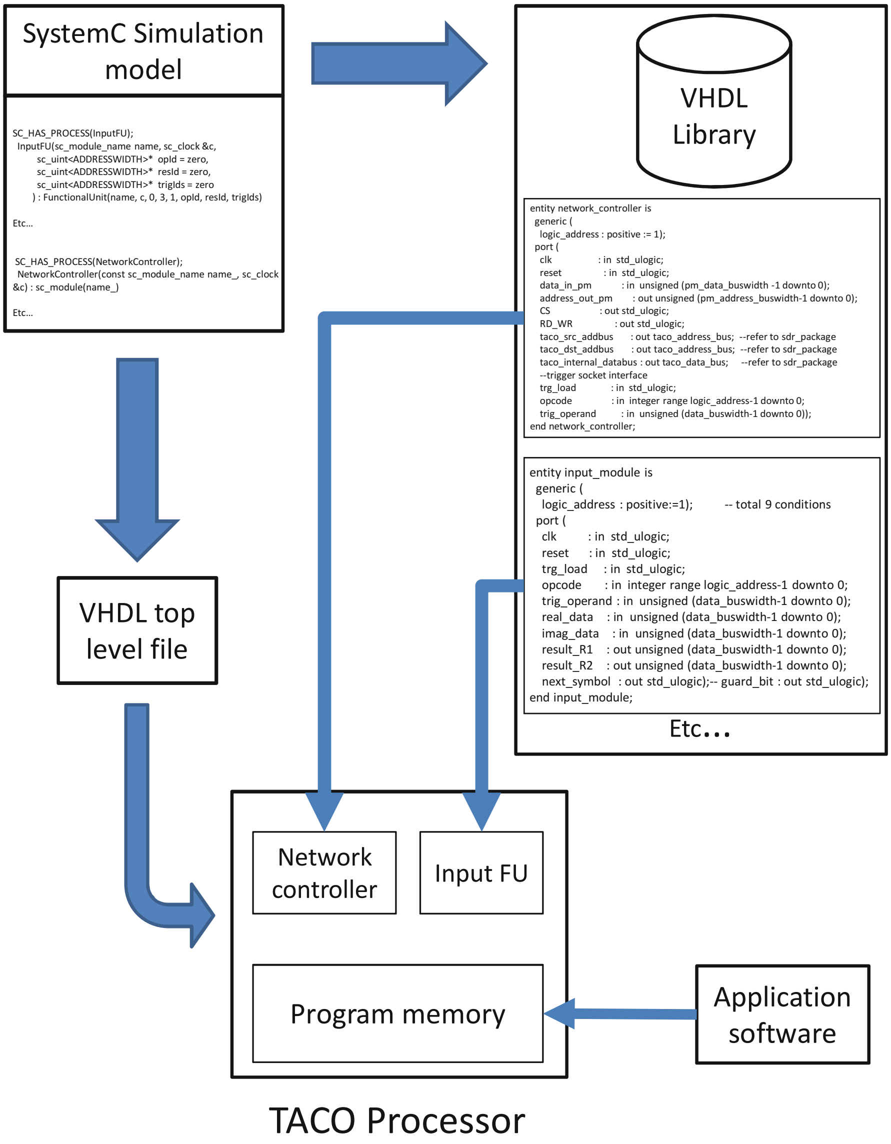


Fig. 11.7 Design flow for hardware–software codesign in TACO SDSC



```
SC_HAS_PROCESS(InputFU);
 InputFU(sc_module_name name, sc_clock &c,
        sc_uint<ADDRESSWIDTH>*  opId = zero,
        sc_uint<ADDRESSWIDTH>*  resId = zero,
        sc_uint<ADDRESSWIDTH>*  trigIds = zero
      ) : FunctionalUnit(name, c, 0, 3, 1, opId, resId, trigIds)

Etc...


 SC_HAS_PROCESS(NetworkController);
 NetworkController(const sc_module_name name_, sc_clock
&c) : sc_module(name_)

Etc...
```

```
entity network_controller is
 generic (
  logic_address : positive := 1);
 port (
  clk                : in  std_ulogic;
  reset              : in  std_ulogic;
  data_in_pm         : in  unsigned (pm_data_buswidth -1 downto 0);
  address_out_pm     : out unsigned (pm_address_buswidth-1 downto 0);
  CS                 : out std_ulogic;
  RD_WR              : out std_ulogic;
  taco_src_addbus    : out taco_address_bus;  --refer to sdr_package
  taco_dst_addbus    : out taco_address_bus;  --refer to sdr_package
  taco_internal_databus : out taco_data_bus;  --refer to sdr_package
  --trigger socket interface
  trg_load           : in  std_ulogic;
  opcode             : in  integer range logic_address-1 downto 0;
  trig_operand       : in  unsigned (data_buswidth-1 downto 0));
end network_controller;
```

```
entity input_module is
 generic (
  logic_address : positive:=1);         -- total 9 conditions
 port (
  clk         : in  std_ulogic;
  reset       : in  std_ulogic;
  trg_load    : in  std_ulogic;
  opcode      : in  integer range logic_address-1 downto 0;
  trig_operand : in  unsigned (data_buswidth-1 downto 0);
  real_data   : in  unsigned (data_buswidth-1 downto 0);
  imag_data   : in  unsigned (data_buswidth-1 downto 0);
  result_R1   : out unsigned (data_buswidth-1 downto 0);
  result_R2   : out unsigned (data_buswidth-1 downto 0);
  next_symbol  : out std_ulogic);-- guard_bit : out std_ulogic);
end input_module;
```

Fig. 11.8 Process flow for mapping of system-level designs to VHDL hardware library components and their relation to the final platform instance, complemented by the co-designed application software

version for functional correctness and processing speed, and estimate its hardware characteristics (power consumption and chip size) based on the data gathered from the VHDL component library. When all tests are approved, the hardware is synthesized and the software is finalized for that hardware instance.

11.5.3 Cryptography Reconfiguration Approaches

In the case of a fixed hardware implementation, adversaries immediately know what algorithms are used and what are available for use in different applications. With essential operations implemented in hardware and encryption on the algorithm level implemented in software with hardware acceleration, the software designer has a lot of control on the actual security features of the system. Reacting to vulnerabilities is similarly significantly more flexible, as we can first approach the problem as a software problem, even in cases when hardware level changes are normally warranted.

An effective way for an attacker to approach a hardware cryptography implementation is to try to map its behavior and to find potential side channel attacks [33], which allow the attacker to extract information about the encrypted data from the operation of the physical device itself. If a hardware implementation of an algorithm is found to be vulnerable to a side channel attack, mitigating this threat can be next to impossible. With adaptive cryptography, responding to side channel attacks and timing attacks changes from impossible to plausible due to the possibility of reprogramming the software and still using the same hardware acceleration blocks as before.

If we are using a reconfigurable hardware platform such as FPGA, it expands the options available for cryptography reconfiguration. In addition to redesigning the software, we can also change hardware configuration without actually replacing the hardware. In such use cases where the target application changes with a non-negligible probability, this can be a valid approach. Also the cost of reconfigurable hardware can be prohibitive in many applications, where the unit cost of a computation platform must be pushed as low as possible.

The design flow for reconfiguring cryptography functions and the relation of this process to the TACO design flow is presented in Fig. 11.9. In the case of a change in the security environment that warrants algorithm level changes to the device,

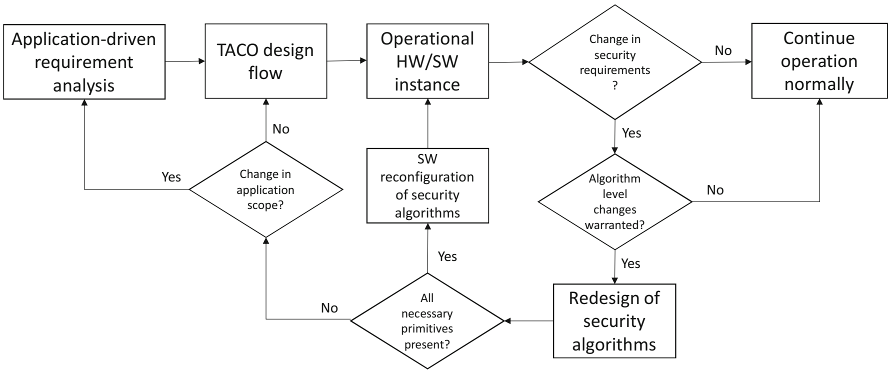

Application-driven requirement analysis
TACO design flow
Operational HW/SW instance
Change in security requirements ?
No
Continue operation normally
Yes
No
Yes
Change in application scope?
SW reconfiguration of security algorithms
Algorithm level changes warranted?
No
Yes
Yes
No
All necessary primitives present?
Redesign of security algorithms

Fig. 11.9 Design flow for reconfiguring security functions

we first enter the software segment of the flow, where the security algorithms are re-evaluated and redesigned as necessary. After this we examine the suitability of the current hardware platform. On one hand, if all necessary primitives are present, no hardware reconfiguration is necessary and we can resume operation with new, redesigned software. On the other hand, if hardware implementations of necessary primitives are not present, we enter the hardware segment of the flow. Here we will first assess whether the initial application requirements have changed and need to be re-evaluated, or if we can just add the necessary hardware instances and go through design space exploration on the existing design. Regardless, we will re-iterate the TACO design flow to get a new design suitable for the changed situation. If we already have hardware reconfiguration capability, reacting to changes in application scope and required cryptography primitives is significantly more streamlined, as we can adapt the existing hardware to a new configuration that we get as the result of a new iteration of the TACO design flow without replacing hardware.

11.5.3.1 Product Customization on the Market

A potential benefit from using the TACO SDSC design flow is the opportunity for swift design of hardware implementations that can be mass produced at large volume and later customized to the specific configuration required by the target application on the software layer. In this approach a single hardware platform capable of supporting several functions and cryptography cipher sets is designed, and the actual product differentiation is done with software. Another significant benefit is that when primitive operations are implemented on hardware and their relationships are defined on the software layer, we can fix possible bugs in implementation by rewriting the software. This makes it possible to fix a bug in production with a software update that would require product recall if implemented fully in hardware.

Reconfigurable hardware naturally makes this process less vital, but the target applications for this approach would probably be those where it is not cost effective or feasible to use reconfigurable hardware. Regardless of the type of the hardware platform, this level of flexibility in customization after manufacture allows for larger manufacturing volumes and cost savings, while at the same time shortening the time-to-market for a product.

11.5.4 Case Study

To test the feasibility of a TACO SDSC platform, the RSA algorithm [44] was chosen as the initial algorithm for implementation. The RSA algorithm is familiar and well known, and it is suitable for this purpose as such. This case study and its results are preliminary, and while far from complete or compelling, still provide us with encouraging initial results for further optimization and improvement of the TACO SDSC hardware platform and its design methodology.

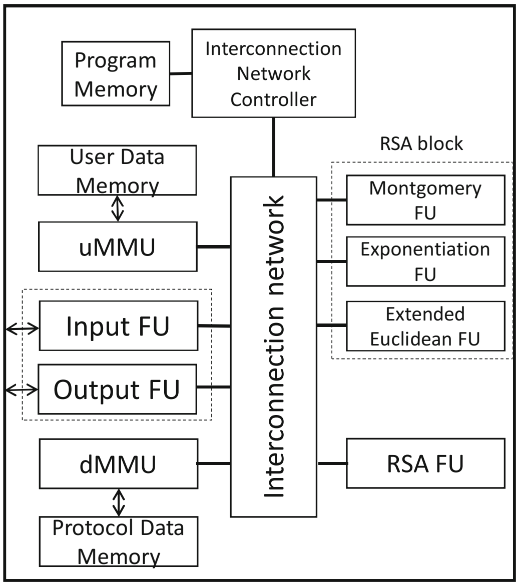


Fig. 11.10 TACO processor with both RSA implementations

11.5.4.1 Architecture

The RSA algorithm was implemented on a TACO instance both as a single FU capable of computing a whole round of RSA encryption and as separate FUs that implement the necessary operations. The simulations were performed in a Linux environment on SystemC 2.3.0. The conceptual model of the test instance is presented in Fig. 11.10. The dedicated RSA FU contains an implementation of the RSA cipher, which reads in the operands, performs RSA encryption on the input, and stores the result in a designated memory location. The separate FUs are programmed to do the same task using individual FUs for Montgomery modular reduction [37], exponentiation, and Extended Euclidean Algorithm for finding modular inverses. These individual operations are used in several types of cryptography algorithms, and as such are a compelling choice for a cryptography enhanced processor instance.

11.5.4.2 Results

The simulations on the processor architecture instance show that when compared with the single Functional Unit, the three separate Functional Units were capable of performing the same RSA encryption operation with a 31 % overhead on performance time. A performance decrease of roughly one-third can be considered significant on a hardware platform, but given that this performance loss comes with the flexibility to change the encryption algorithm by reprogramming, and that these results are preliminary in nature and the code is far from optimized, these results are encouraging for continued improvement on the TACO SDSC design methodology. The directions we plan to explore will be discussed next in Sect. 11.6, where we expand some of our ideas on further research on the SDSC platform and methodology.

11.6 Application and Implementation Scenarios

In this final section we will briefly discuss some interesting and potentially noteworthy applications and implementation scenarios.

11.6.1 Dynamic Security Dimensions and Adaptive Cryptography

In [40] we have explored the concept of dynamic security dimensions and their effects on sensor network security. The concept is based on the various target applications that sensor networks have, and that security requirements can and do change on the fly. We aim to identify all relevant security dimensions, and model them in a manner which facilitates the construction of an adaptive holistic security system for low-power and resource-constrained devices. We identify sensor networks and IoT devices as targets that would benefit greatly from this approach.

To summarize, we first model the security landscape of sensor networks as individual "dimensions" that represent distinct, measurable characteristics. The key dimensions that we have identified are energy (d_{e}), processing capacity (d_{proc}), memory capacity (d_{me}), data coherence (d_{co}), data lifetime (d_{li}), location security level (d_{loc}), and mobility (d_{mo}). These dimensions, or dimension vectors, are modified by corresponding scalar weight factors (w), so that we can model the changes in environment, and by grouping related dimensions together we aim to reduce the complexity of modeling the security environment. We define the final set of compound dimensions as the 4-tuple

$$(\alpha, \beta, \gamma, \delta) = \Bigg(d_{\mathrm{e}} w_{\mathrm{e}}, \frac{d_{\mathrm{me}} w_{\mathrm{me}} + d_{\mathrm{proc}} w_{\mathrm{proc}}}{w_{\mathrm{me}} + w_{\mathrm{proc}}}, \frac{d_{\mathrm{mo}} w_{\mathrm{mo}} + d_{\mathrm{loc}} w_{\mathrm{loc}}}{w_{\mathrm{mo}} + w_{\mathrm{loc}}}, \frac{d_{\mathrm{co}} w_{\mathrm{co}} + d_{\mathrm{li}} w_{\mathrm{li}}}{w_{\mathrm{co}} + w_{\mathrm{li}}}\Bigg). \tag{11.1}$$

Energy is the only dimension left uncompounded, given its importance in resource-constrained systems. These dimensions can be used to model the security environment with extreme granularity by assigning a mapping between different real-world scenarios and the four-dimensional security landscape. For example, a certain region of the security space where energy consumption is not an issue, and the coherence and lifetime of data are high, using the strongest available encryption algorithm supported by the current configuration is warranted. Another region could have high limits for energy consumption and low data lifetime and coherence, so the security environment would allow data to be transmitted even without any confidentiality measures in place. For some situations even integrity preserving features can be turned off to conserve energy.

Changes in dimensions can force changes in operating parameters, and this can warrant changing operational security features in the system. These changes can happen quickly or gradually. Even the slow kind of change can be dramatic, if, for example, a certain set of cryptographic ciphers is compromised. While unlikely, this not impossible, as cryptanalysis on existing systems is always ongoing and intelligence organizations and other equivalent governmental organizations have capabilities that can force changes in this field [8]. The ability to adapt even older hardware to a new situation can be a preferable approach, compared to having to replace all hardware with new designs, having to wait for them to be designed and verified and manufactured, and so on.

The main motivation of this approach is to design a dynamically adaptive sensor network security framework, capable of assessing available resources and changing application requirements in order to provide optimal protection with minimal overhead and cost. The framework observes the network continuously using an intrusion detection/prevention system and also uses location related information and feedback from a trust management module responsible for mapping different trust scenarios and conditions on the sensor network and its surroundings. With this information, the end system based on this framework takes action to achieve optimal security-energy point. This translates into maximum security under a given resource and context. The security framework will be developed based on a platform vision to enable cost effective and application-independent realization. The scalability feature of the framework is also an important aspect since the number of nodes in low-power wireless networks varies from a few to many nodes based on the application requirement.

We see that incorporating adaptive cryptography as a concept is vital to the evolution of such a framework, and the TACO SDSC platform will give us a robust and efficient HW-SW platform for development and eventual implementation.

11.6.2 Building Security in the Design

The current version of the TACO SDSC methodology does not yet take into account some aspects of secure processor design as outlined in [7] and [12], such as secure handling of cryptographic keys. These security issues have been addressed in designs such as SAFES [13], and similar care should be taken when designing cryptography functionality for TACO SDSC processors. This is a clear improvement target for the design process in order to incorporate security and safety into the core of the methodology.

In a broader scope, the concept of building security and trust into systems that we use every day is a vital aspect of future communication systems. The TACO SDSC platform and methodology make designing communication systems that can adapt to changing environments and requirements significantly easier. For a single hardware instance, we can support several different software implementations of the same functionality. By doing the changes only on the software level, we provide an additional layer of resistance to attacks by malicious parties. If a hardware instance is deemed to be vulnerable to an attack, by reconfiguring the system on the software level we can extend our mitigation capabilities substantially.

11.7 Conclusion

When we consider the requirements of future communication systems, the need for flexible encryption in all communication layers is evident. With the advent of IoT, ubiquitous computing, and a data-intensive information society, devices and sensors with strict requirements for both efficient communications and robust security will be in demand. The SDR concept can be leveraged and enhanced to answer these requirements.

In this chapter we have presented a design methodology and process for a Software Defined Secure Communications framework. As a platform, SDSC can provide secure, flexible, and lightweight communication to sensor networks and IoT devices without significantly compromising performance, and the presented framework provides the methods and processes for exploring this further. Additional benefits of this design approach include potential cost savings in larger production volumes and shorter time-to-market for products when we can design devices, that are highly customizable with software and implement only key primitive operations in hardware. We see that our design model coupled with the concept of adaptive cryptography could be leveraged towards developing flexible and yet secure application-specific devices for future communication networks.

Acknowledgements The authors would like to thank Alexey Boyko and Pramesh Khanal for their contribution in the evolution of this manuscript.

References

1. Anwar, M.I., Virtanen, S., Isoaho, J.: A software defined approach for common baseband processing. J. Syst. Archit. **54**, 769–786 (2008)
2. Ball, J., Borger, J., Greenwald, G.: Revealed: how US and UK spy agencies defeat internet privacy and security. http://www.theguardian.com/world/2013/sep/05/nsa-gchq-encryption-codes-security. 6 Sept 2013
3. Barrett, P.: Implementing the Rivest Shamir and Adleman public key encryption algorithm on a standard digital signal processor. In: Advances in Cryptology, pp. 311–323. Springer, New York (1987)

4. Bartolini, S., Giorgi, R., Martinelli, E.: Instruction set extensions for cryptographic applications. In: Koç, Ç.K. (ed.) Cryptographic Engineering, pp. 191–233. Springer, New York (2009)
5. Benadjila, R., Billet, O., Gueron, S., Robshaw, M.J.: The intel AES instructions set and the SHA-3 candidates. In: ASIACRYPT 2009: Proceedings of the 15th International Conference on the Theory and Application of Cryptology and Information Security, pp. 162–178. Springer, New York (2009)
6. Björkqvist, J., Virtanen, S.: Convergence of hardware and software in platforms for radio technologies. IEEE Commun. Mag. **44**, 52–57 (2006)
7. Bossuet, L., Grand, M., Gaspar, L., Fischer, V., Gogniat, G.: Architectures of flexible symmetric key crypto engines – a survey: from hardware coprocessor to multi-crypto-processor system on chip. ACM Comput. Surv. (CSUR) **45**(4), 41 (2013)
8. Checkoway, S., Fredrikson, M., Niederhagen, R., Everspaugh, A., Green, M., Lange, T., Ristenpart, T., Bernstein, D.J., Maskiewicz, J., Shacham, H.: On the Practical Exploitability of Dual EC in TLS Implementations, in USENIX Security (2014)
9. Corporaal, H.: Microprocessor Architectures from VLIW to TTA. Wiley, Chichester (1998)
10. Dandalis, A., Prasanna, V.K.: An adaptive cryptographic engine for internet protocol security architectures. In: ACM Transactions On Design Automation Of Electronic Systems, vol. 9, pp. 333–353. Association for Computing Machinery, New York (2004). ISSN: 1084-4309, July 2004
11. Dandalis, A., Prasanna, V.K., Rolim, J.D.P.: An adaptive cryptographic engine for IPSec architectures. In: 2000 IEEE Symposium on Field-Programmable Custom Computing Machines, IEEE, pp. 132–141 (2000)
12. Gaspar, L., Fischer, V., Bernard, F., Bossuet, L., Cotret, P.: HCrypt: a novel concept of crypto-processor with secured key management. In: 2010 International Conference on Reconfigurable Computing and FPGAs (ReConFig), IEEE, pp. 280–285 (2010)
13. Gogniat, G., Wolf, T., Burleson, W., Diguet, J.-P., Bossuet, L., Vaslin, R.: Reconfigurable hardware for high-security/high-performance embedded systems: the SAFES perspective. IEEE Trans. Very Large Scale Integr. VLSI Syst. **16**(2), 144–155 (2008)
14. Goldwasser, S., Micali, S.: Probabilistic encryption. J. Comput. Syst. Sci. **28**(2), 270–299 (1984). Elsevier
15. Gordon, D.M.: A survey of fast exponentiation methods. J. Algoritm. **27**(1), 129–146 (1998). Elsevier
16. Guo, W., Liu, Y., Bai, S., Wei, J., Sun, D.: Hardware architecture for RSA cryptography based on residue number system. In: Transactions of Tianjin University, vol. 18, pp. 237–242. Springer, Heidelberg (2012)
17. Hakkala, A., Virtanen, S.: Accelerating cryptographic protocols: a review of theory and technologies. In: Proceedings of CTRQ 2011: The Fourth International Conference on Communication Theory, Reliability, and Quality of Service (2011)
18. Hankerson, D., Menezes, A.J., Vanstone, S.: Guide to Elliptic Curve Cryptography. Springer Science and Business Media, Heidelberg (2006)
19. Hansen, K., Larsen, T., Olsen, K.: On the efficiency of fast RSA variants in modern mobile phones. Int. J. Comput. Sci. Inf. Secur. **6**(3), 136–140 (2009)
20. Hasenplaugh, W., Gaubatz, G., Gopal, V.: Fast modular reduction. In: 18th IEEE Symposium on Computer Arithmetic (ARITH'07), pp. 1063–6889. IEEE Computer Society, Los Alamitos, CA (2007)
21. Hern, A.: Apple defies FBI and offers encryption by default on new operating system. In: The Guardian. Online at http://www.theguardian.com/technology/2014/oct/17/apple-defies-fbi-encryption-mac-osx Last accessed 17th Oct 2014
22. Hong, S.M., Oh, S.Y., Yoon, H.: New modular multiplication algorithms for fast modular exponentiation. In: Advances in Cryptology - EUROCRYPT'96, pp. 166–177. Springer, Berlin (1996)
23. Hu, J., Guo, W., Wei, J., Chang, Y., Sun, D.: A novel architecture for fast RSA key generation based on RNS. In: 2011 Fourth International Symposium on Parallel Architectures, Algorithms and Programming (PAAP), IEEE, pp. 345–349 (2011)

24. Hämäläinen, P., Hännikäinen, M., Hämäläinen, T.D., Corporaal, H., Saarinen, J.: Implementation of encryption algorithms on transport triggered architectures. In: Proceedings of IEEE International Symposium on Circuits and Systems, vol. 4, pp. 726–729 (2001)
25. Hämäläinen, P., Heikkinen, J., Hännikäinen, M., Hämäläinen, T.D.: Design of transport triggered architecture processors for wireless encryption. In: Proceedings of 8th Euromicro Conference on Digital System Design (2005)
26. Hämäläinen, P., Hännikäinen, M., Hämäläinen, T.D.: Review of hardware architectures for advanced encryption standard implementations considering wireless sensor networks. In: Embedded Computer Systems: Architectures, Modeling, and Simulation, pp. 443–453. Springer, New York (2007)
27. Isoaho, J., Virtanen, S., Plosila, J.: Current challenges in embedded communication systems. Int. J. Embed. Real-Time Commun. Syst. **11**, 1–21 (2010)
28. Itoh, T., Tsujii, S.: A fast algorithm for computing multiplicative inverses in GF(2^m) using normal bases. In: Information and Computation, vol. 78(3), pp. 171–177. Elsevier, Amsterdam (1988)
29. Jantsch, A., Öberg, J., Hemani, A.: Is there a niche for a general protocol processor core? In: Proceedings of the 16th IEEE Norchip Conference, pp. 93–100 (1998)
30. Järvinen, K.U.: On repeated squarings in binary fields. In: Proceedings of the 16th International Workshop on Selected Areas in Cryptography, SAC 2009. Lecture Notes in Computer Science, vol. 5867, pp. 331–349. Springer, New York (2009)
31. Khalil-Hani, M., Hau, Y.W.: SystemC HW/SW co-design methodology applied to the design of an elliptic curve crypto system on chip. International Conference on Microelectronics, ICM 2008, pp. 147–150 (IEEE, 2008)
32. Koblitz, N.: Elliptic curve cryptosystems. Math. Comput. **48**, 203–209 (1987)
33. Kocher, P.C: Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. Advances in Cryptology - CRYPTO'96, pp. 104–113. Springer, Berlin (1996)
34. Kocher, P., Lee, R., McGraw, G., Raghunathan, A.: Security as a new dimension in embedded system design. In: Proceedings of the 41st Annual Design Automation Conference (2004)
35. Liu, Y., Guo, W., Tan, Y., Wei, J., Sun, D.: An efficient scheme for implementation of SM2 digital signature over GF (p). Contemporary Research on E-business Technology and Strategy, pp. 250–258. Springer, New York (2012)
36. Miller, V.: Use of elliptic curves in cryptography. In: Proceedings of the Advances in Cryptology - CRYPTO'85. Springer, New York (1986)
37. Montgomery, P.L.: Modular multiplication without trial division. Math. Comput. **44**(170), 519–521 (1985)
38. Moore, T., Clayton, R., Anderson, R.: The economics of online crime. J. Econ. Perspect. **23**(3), 3–20 (2009). American Economic Association, 2015.05.26
39. National Institute of Standards and Technology (NIST): Advanced encryption standard. In: Federal Information Processing Standards Publication 197, 26 Nov 2001
40. Nigussie, E., Hakkala, A., Virtanen, S., Isoaho, J.: Energy-aware adaptive security management for wireless sensor networks. In: IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks, IEEE (2014)
41. Paakkulainen, J., Virtanen, S., Isoaho, J.: Tuning a protocol processor architecture towards DSP operations. Lect. Notes Comput. Sci. **3553**, 132–141 (2005)
42. Poulsen, K.: Apple's iPhone encryption is a godsend, even if cops hate it. Wired. Online at http://www.wired.com/2014/10/golden-key/ 10th Aug 2014. Last accessed 28 May 2015
43. Provos, N., Mazieres, D.: A future-adaptable password scheme. In: FREENIX Track USENIX Annual Technical Conference, pp. 81–91 (1999)
44. Rivest, R.L., Shamir, A., Adleman, L.: A method for obtaining digital signatures and public-key crypto systems. Commun. ACM **21**(2), 126 (1978)
45. Schaumont, P., Verbauwhede, I.: A reconfiguration hierarchy for elliptic curve cryptography. ASILOMAR Conference on Signals, Systems and Computers, IEEE, vol. 1, pp. 449–453 (2001)

46. Schneier, B.: Applied Cryptography: Protocols, Algorithms and Source Code in C, 2nd edn. Wiley, New York (1996)
47. Schneier, B.: Data and Goliath. W. W. Norton and Co., New York (2015)
48. Tillich, S., Großschädl, J.: Accelerating AES using instruction set extensions for elliptic curve cryptography. Proceedings of Computational Science and Its Applications ICCSA 2005, pp. 665–675 (2005)
49. Timberg, C.: Google encrypts data amid backlash against NSA spying. The Washington Post. Online at http://www.washingtonpost.com/business/technology/google-encrypts-data-amid-backlash-against-nsa-spying/2013/09/06/9acc3c20-1722-11e3-a2ec-b47e45e6f8ef_story.html. 6th Sept 2013. Last accessed 28 May 2015
50. Truscan, D., Virtanen, S., Lilius, J.: Protocol processor design issues. In: Nurmi, J. (ed.) Processor Design: System-on-Chip Computing for ASICs and FPGAs. Chapter 12, Springer, Dordrecht, pp. 257–285 (2007)
51. Virtanen, S.: A framework for rapid design and evaluation of protocol processors, University of Turku, 2004
52. Virtanen, S., Nurmi, T., Paakkulainen, J., Lilius, J.: A system-level framework for designing and evaluating protocol processor architectures. Int. J. Embed. Syst. **1**(1–2), 78–90 (2006)
53. Wu, C.-L.: An efficient common-multiplicand-multiplication method to the Montgomery algorithm for speeding up exponentiation. Inform. Sci. **179**(4), 410–421 (2009)

Chapter 12
The Future of Software-Defined Radio: Recommendations

Waqar Hussain, Jouni Isoaho, and Jari Nurmi

An efficient Software-Defined Radio solution comes when all the aspects of system design are collectively addressed under application specifications and constraints. It includes all—the efforts to design wideband antennas, powerful software to process huge bandwidth of information, optimizations at hardware to maximize performance, and nevertheless to mention compilers and operating systems. It is important that every engineer or a scientist working on a particular block of SDR should have a bare-minimum understanding of the entire design stack. There is a need to have clear vision about the targets to be achieved, trade-offs to be made, and a unified approach so that all the objectives are measurable to enable a qualitative and quantitative analysis.

The contributions as chapters in the books provide conclusions and based on those we can indicate recommendations for future designs. Computationally intensive parallel processing tasks are suitable for homogeneous/heterogeneous multicore architectures. The cores in such platform can be chosen based on application scenarios, for example a general-purpose task is more suitable for a RISC processor to handle or signal processing task to a DSP. Special logic can be implemented on FPGAs and rASIPs can be used to allow maximum flexibility. The cores can communicate with each other over a network-on-chip or on an efficient bus architecture. If the objective is to provide maximum instruction-level parallelism, then multiple VLIW cores can be integrated together over

W. Hussain (✉) • J. Nurmi
Department of Electronics and Communications Engineering,
Tampere University of Technology, Tampere, Finland
e-mail: waqar.hussain@tut.fi; jari.nurmi@tut.fi

J. Isoaho
Department of Applied Physics, Laboratory of Electronics and Information Technology,
University of Turku, Turku, Finland
e-mail: jisoaho@utu.fi

W. Hussain et al. (eds.), *Computing Platforms for Software-Defined Radio*,
DOI 10.1007/978-3-319-49679-5_12

a high-performance communication infrastructure. In such cases, LLVM-based C-compiler StreamIt-based compilers are optimal choices. Reconfigurable MPSoCs is also a good solution for SDRs as they are highly programmable and require a minimum knowledge of the underlying hardware by the SDR programmers. Open source libraries such as GNU radio will attract more programmers towards the SDR development. From a scalable communications core, we can expect that it yields several advantages including architectural efficiency, efficient interconnects, targeted flexibility and control as well as programmability. Tool development is also very important for consistent SDR evolution. We need tools that can do rapid prototyping, provide early synthesis results and near-accurate performance estimates. The tools should contain support for multiple compilers of different programming languages.

In the end, we can emphasize there is a need to continuously invent new applications to keep a high demand for SDR technology. All those interesting features that this books highlights come at a cost, i.e., power and area/resources. An SDR world—present day Internet-of-Things can become an environmental hazard due to the unprecedented energy requirements. It is urgently required that special emphasis should be given to the global view and idea of SDR design paradigm from all fields of technology.

Bibliography

1. Barker, C.W., Barker, E.B.: Recommendation for the Triple Data Encryption Algorithm (TDEA) Block Cipher, SP 800–67 Rev. 1. National Institute of Standards and Technology, Gaithersburg (2012)
2. Benini, L., Macii, A., Poncino, M.: A recursive algorithm for low-power memory partitioning. In: Proceedings of the 2000 International Symposium on Low Power Electronics and Design, IEEE, pp. 78–83 (2000)
3. Brodersen, R.W.: Low Power Digital CMOS Design. Springer, New York (1995)
4. Brown, B., Aaron, M.: The politics of nature. In: Smith, J. (ed.) The Rise of Modern Genomics, 3rd edn. Wiley, New York (2001)
5. Chun, A. et al.: Application of the Intel Reconfigurable Communications Architecture to 802.11a, 3G and 4G Standards. In: Frontiers of Mobile and Wireless Communication, 31 May–2 June 2004
6. Corporaal, H., Hoogerbrugge, J.: Cosynthesis with the MOVE framework. In: Symposium on Modelling, Analysis, and Simulation, Citeseer, pp. 184–189 (1996)
7. Eberli, S., et al.: An IEEE 802.11a baseband receiver implementation on an application specific processor. In: 50th Midwest Symposium on Circuits and Systems, pp. 1324–1327 (2007)
8. Ekpanyapong, M., Minz, J.R., Watewai, T., Lee, H.-H.S., Lim, S.K.: Profile-guided microarchitectural floor planning for deep submicron processor design. IEEE Trans. Comput. Aided Des. Integr. Circuits Syst. **25**(7), 1289–1300 (2006)
9. ElGamal, T.: A public key cryptosystem and a signature scheme based on discrete logarithms. Advances in Cryptology, pp. 10–18. Springer, New York (1985)
10. Federal Information Processing Standards Publication 186-2: Digital Signature Standard, National Institute of Standards and Technology (NIST), 2000
11. Fürer, M.: Faster integer multiplication. SIAM J. Comput. **39**(3), 979–1005 (2009)
12. Gentry, C.: Fully homomorphic encryption using ideal lattices. In: STOC, vol. 9, pp. 169–178 (2009)
13. Gruian, F.: System-level design methods for low-energy architectures containing variable voltage processors. In: Power-Aware Computer Systems, pp. 1–12. Springer, Berlin (2001)
14. Guzma, V., Jääskeläinen, P., Kellomäki, P., Takala, J.: Impact of software bypassing on instruction level parallelism and register file traffic. In: Embedded Computer Systems: Architectures, Modeling, and Simulation, pp. 23–32. Springer, New York (2008)
15. Herbert, S., Marculescu, D.: Analysis of dynamic voltage/frequency scaling in chip-multiprocessors. In: 2007 ACM/IEEE International Symposium on Low Power Electronics and Design (ISLPED), IEEE, pp. 38–43 (2007)

W. Hussain et al. (eds.), *Computing Platforms for Software-Defined Radio*,
DOI 10.1007/978-3-319-49679-5

16. Hung, W.-L., Xie, Y., Vijaykrishnan, N., Addo-Quaye, C., Theocharides, T., Irwin, M.J.: Thermal-aware floorplanning using genetic algorithms. In: Sixth International Symposium on Quality of Electronic Design, 2005, IEEE, pp. 634–639 (2005)
17. Hwang, T., Yang, C., Wu, G., Li, S., Li, G.Y.: OFDM and its wireless applications: a survey. IEEE Trans. Veh. Technol. **58**(4), 1673–1694 (2009)
18. Jejurikar, R., Gupta, R.: Dynamic voltage scaling for systemwide energy minimization in real-time embedded systems. In: Proceedings of the 2004 International Symposium on Low Power Electronics and Design, 2004, IEEE, pp. 78–81 (2004)
19. Karatsuba, A., Ofman, Y.: Multiplication of multidigit numbers on automata. Sov. Phys. Dokl. **7**, 595 (1963)
20. Kellomäki, P., Guzma, V., Takala, J.: Safe pre-pass software bypassing for transport triggered processors. Acta Technica Napocensis: Electronica-Telecomunicatii. **49**(3), 5–10 (2008)
21. Lee, R.: Precision architecture. Computer **22**(1), 78–91 (1989). IEEE Computer Society Press
22. Lee, R., Mahon, M., Morris, D.: Pathlength reduction features in the PA-RISC architecture In: Compcon Spring'92. Thirty-Seventh IEEE Computer Society International Conference, Digest of Papers, IEEE, pp. 129–135 (1992)
23. McMahan, L., Lee, R.: Pathlengths of SPEC benchmarks for PA-RISC, MIPS, and SPARC. In: Compcon Spring'93, Digest of Papers, IEEE, pp. 481–490 (1993)
24. Melander, J.: Design of SIC FFT architectures. Department of Electrical Engineers, Linköping University, 1997
25. Melpignano, D., Benini, L., Flamand, E., Jego, B., Lepley, T., Haugou, G., Clermidy, F., Dutoit, D.: Platform 2012, a many-core computing accelerator for embedded SoCs: performance evaluation of visual analytics applications. In: Proceedings of the 49th Annual Design Automation Conference (DAC '12), pp. 1137–1142. ACM, New York (2012)
26. Nutaq, It's all SDR-related: Understanding adaptive radio, cognitive radio, and intelligent radio. http://www.nutaq.com/its-all-sdr-related-understanding-adaptive-radio-cognitive-radio-and-intelligent-radio/. Accessed 20.12.2016
27. Reyes, A.C., Castillo, A.K.V., Morales-Sandoval, M., Diaz-Perez, A.: A performance comparison of elliptic curve scalar multiplication algorithms on smartphones. In: 2013 International Conference on Electronics, Communications and Computing (CONIELECOMP), pp. 114–119, March 2013
28. Schneier, B., Kelsey, J., Whiting, D., Wagner, D., Hall, C., Ferguson, N.: The Twofish Encryption Algorithm: A 128-bit Block Cipher. Wiley, New York (1999). ISBN: 0-471-35381-7
29. Schönhage, A.D. Dr, Strassen, V.: Schnelle multiplikation grosser zahlen. In: Computing, vol. 7(3–4), pp. 281–292. Springer, Berlin (1971)
30. Tabak, D., Lipovski, G.J.: MOVE architecture in digital controllers. IEEE Trans. Comput. **29**(2), 180–190 (1980)
31. Toom, A.L.: The complexity of a scheme of functional elements realizing the multiplication of integers. Sov. Math. Dokl. **3**(4), 714–716 (1963)
32. Wang, Z., Arslan, T.: A low power reconfigurable heterogeneous architecture for a mobile SDR system. In: IEEE International Symposium on Circuits and Systems, 2009, ISCAS 2009, pp. 2025–2028, 24–27 May 2009
33. Wong, K.L., Wu, C.H., Su, S.W.: Ultrawide-band square planar metal-plate monopole antenna with a trident-shaped feeding strip. IEEE Trans. Antennas Propag. **53**(4), 1262–1269 (2005)