Evaluations of Hardware Platforms, Methods and Tools for Low-Volume Software-Defined Signal Processing Applications

Bertrand Rousseau

Thesis submitted in fulfillment
of the requirements for the degree of
Docteur en Sciences de l'Ingénieur

Dissertation committee:
Pr. Jean-Didier Legat (UCL, ICTEAM/ELEN) - Advisor
Pr. Jean-Jacques Quisquater (UCL, ICTEAM/ELEN)
Pr. Benoît Macq (UCL, ICTEAM/ELEN)
Dr. Hans Vandierendonck (Ghent University, Dept. ELIS)
Dr. Michaël Hübner (Karlsrühe Institute of Technology, ITIV)
Pr. Michel Verleysen (UCL, ICTEAM/ELEN) - President

February 2011

CONTENTS

Acknowledgments vii

Abstract ix

Introduction xi

I.1 Problem statement xii
I.2 Thesis contributions xiv
I.3 Outline xvi

Author's publication list xix

1 Software-defined signal processing applications and platforms 1

1.1 Introduction 1
1.2 Advanced signal processing applications 2
1.3 Software-defined signal processing applications 2
1.4 Low-volume embedded applications 3
1.5 Implementation requirements for low-volume embedded SDA 5
1.6 Semiconductor technology 5
1.7 Summary of hardware platform requirements for low-volume embedded SDA 7
1.8 Existing platforms for SDA 8
1.8.1 Issues with conventional heterogeneous platforms 8
1.8.2 Multi-core processors 9
1.8.3 SIMD processors 10
1.8.4 Coarse-grain reconfigurable platforms 11
1.8.5 FPGAs 12
1.8.6 Homogeneous many-core platforms 12
1.9 Summary of existing platforms 14

2 SDA platform comparisons 19

2.1 Introduction 19
2.2 Platform comparisons 20
2.3 Platform model characterization 21
2.4 Comparison issues 22
2.5 Technology and voltage adaptations 22
2.6 Application benchmarks 24
2.7 Power efficiency 28
2.8 Computational load 31
2.9 Dynamic function change 33
2.10 Development efforts 34
2.11 Genericity 36
2.12 Summary 38

3 Analysis of DPR impact on design and development 43

3.1 Introduction 43
3.2 Dynamic partial reconfiguration of FPGA 44
3.2.1 Configuration chain 44
3.2.2 Bitstream composition 45
3.2.3 FPGA configuration interfaces 46
3.2.4 Reconfiguration process 47
3.2.5 Evolution of DPR in the latest components 48
3.3 Related work 48
3.4 DPR benefits 49
3.5 Advantages for SDA and low-volume applications 51
3.6 DPR impacts on FPGA design 52
3.6.1 Design partition 53
3.6.2 Bus macros 53
3.6.3 Bitstream storage 54
3.6.4 Additional hardware required for reconfiguration 55
3.7 Impacts on the development flow 56
3.8 Hardware virtualization management 57
3.9 Summary 58

4 A minimal flow for FPGA dynamic reconfiguration 61

4.1 Introduction 61
4.2 Certification of safety-critical systems 62
4.3 Related work 63

4.4 Certification of FPGA designs 63
4.4.1 Certification of conventional FPGA designs 63
4.4.2 DPR certification issues 64
4.5 A minimal flow for DPR 65
4.5.1 Definition 65
4.5.2 Flow implementation 67
4.5.3 Experimental results 69
4.5.4 Limitations 70
4.6 Certification of DPR-enabled designs 70
4.7 Summary 71

5 Evaluation of DPR impacts on FPGA design power consumption 75

5.1 Introduction 75
5.2 Related work 76
5.3 Experimental setup 77
5.4 Evaluation of power reduction opportunities 79
5.5 Performance comparison with a silicon processor core 81
5.6 Evaluation of the reconfiguration power consumption 82
5.7 Impact of the reconfiguration cost 83
5.8 Memory system power consumption 84
5.9 Chip size reduction 86
5.10 Summary 87

6 Methodology for architectural-level comparison of HMCP processor cores 91

6.1 Introduction 91
6.2 Related work 93
6.3 Methodology for architectural comparison 93
6.4 Architecture template 95
6.5 Microarchitecture optimizations 97
6.6 Designed processor cores 97
6.7 Processor core implementations 99
6.8 Methodology validation 101
6.9 Summary 103

7 Processor architecture impact on homogeneous many-core platforms 107

7.1 Introduction 107
7.2 Evaluation methodology 108

7.3 HMCP power consumption 109
7.4 Evaluation of core communication overhead 109
7.5 Processor core comparisons 111
7.6 Voltage and frequency scaling 113
7.7 HMCP power, area and throughput 115
7.7.1 Power and throughput 115
7.7.2 Power and area 117
7.8 Core architecture impact on the platform 118
7.8.1 Impact on area 118
7.8.2 Power efficiency 120
7.8.3 Impact of area on power efficiency 120
7.9 Impact of the communication network 122
7.9.1 Evaluation methodology 122
7.9.2 Impact on platform area 124
7.9.3 Evaluation of network power consumption 125
7.9.4 Impact on the platform power 126
7.10 Summary 128

8 Conclusion 131

8.1 Context of the work 132
8.2 Summary of the contributions 133
8.3 Conclusions 136
8.4 Future work 137
8.4.1 Comparison of platform models for SDAs 137
8.4.2 FPGAs with DPR 138
8.4.3 HMCPs 138

References 141

ACKNOWLEDGMENTS

First, I would like to thank my advisor Prof. Jean-Didier Legat for giving me the opportunity to make this Ph.D., and for his guidance during this work.

I would also like to thank the members of my examination committee: Prof. Benoît Macq and Prof. Jean-Jacques Quisquater for the constructive reviews they made, and Prof. Michel Verleysen for chairing the defenses. I am further grateful to Dr. Michaël Hübner for his advice on FPGA dynamic partial reconfiguration and Dr. Hans Vandierendonck for his advice and remarks that contributed greatly to improve the quality of my Ph.D. dissertation.

I would also like to thank all the people I worked with during my Ph.D. Especially, I would like to thank Philippe Manet, whose advice and visions carried me all these years, helped me to get through the many challenges I encountered during this research, and allowed me to go far beyond where I would have gone by myself.

I would also like to thank all the people for the collaborations I had with them, especially for this thesis: Hans, Philippe, Thibault Delavallée, Igor Loiselle and Angelo Kuti Lusala. In particular, I would like to thank Igor and Thibault for allowing me to use the results presented in Figure 3.9 and Figure 5.4 of this dissertation, which they produced and that we published collaboratively. I would also like to thank them both, together with Philippe, for their careful proofreading of this text.

I would also like to thank all the people at the DICE microelectronics laboratory at UCL for their hospitality.

Finally, I would like to thank my family and friends for their kind support during those years.

B. R.

ABSTRACT

Embedded signal processing applications evolve towards advanced applications supporting many standards and providing advanced functionalities. In those applications, the control and software parts are increasing. Consequently, their implementations shift progressively from full hardware implementations to partial or full software implementations. In this thesis, those software implementations are regrouped under the name of software-defined signal processing applications, or SDAs.

Among embedded electronics, many applications have low volumes of production. This is for instance the case of professional electronics, product preseries, application prototypes or high-end consumer products. For those applications, the non-recurring engineering costs (NREs) related to application development and platform realization are a major issue. Indeed, due to their limited production volumes, those applications cannot easily amortize high NREs.

Conventional hardware platforms for signal processing applications like heterogeneous platforms cannot meet all the requirements of SDAs and low-volume applications. This situation has motivated the development of new hardware platform models like multi-core processors, SIMD processors, coarse-grain reconfigurable platforms, fine-grain reconfigurable platforms like FPGAs, and homogeneous many-core platforms (HMCP).

In this thesis, those five hardware platform models are evaluated and compared. The evaluations allow to identify the advantages and disadvantages of each platform with respect to low-volume SDA requirements. Two platform models are further studied: FPGAs with dynamic partial reconfiguration and HMCPs. For FPGAs with DPR, this work studies the impact of DPR on design, development and power consumption. The benefits of DPR for SDAs are identified and evaluated, and some tools and methods are proposed to solve the issues introduced by this technique regarding development and certification. For HMCPs, a precise evaluation of the impact of the processor core architecture on the platform performance is realized. In order to enable this evaluation, a processor core implementation methodology is proposed that allows to perform precise comparisons at the architectural level.

INTRODUCTION

I.1 PROBLEM STATEMENT

Embedded systems are dedicated to execute specific functions, generally under real-time constraints. They are used in many different application fields like telecommunications, automotive, aerospace, military, industrial control, etc. Embedded systems must work with a very limited power budget since they are generally battery-powered or have to deal with cooling issues. The platforms used to implement those embedded systems must therefore be highly power efficient.

Among applications implemented in embedded systems, signal processing applications evolve towards applications supporting a high number of standards and providing advanced functionalities. For instance, mobile communication systems must support many standards (e.g.: GSM, UMTS, LTE, 4G) and have to support advanced functionalities like web browsing or video editing, etc. Other domains evolve in the same way. In video applications for instance, smart cameras are capable of analyzing the content of a video stream in real time [1]. This evolution introduces a significant part of control and software in those applications. Moreover, those applications work with high data rates, and use elaborated signal processing operations. For instance, 4G mobile standards foresee data rates up to 1Gb/s. Consequently, those applications demand very high computation loads.

Simultaneously with the evolution of signal processing applications, signal processing platforms have progressively shifted from platforms supporting only the function of a single standard to advanced platforms providing access to several standards as a set of services. In those systems, standard implementations have progressively shifted from complete hardware implementations to partial or full software implementations. In this thesis, those software implementations are regrouped under the name of software-defined signal processing applications, or SDA in short. In the telecommunication field, a popular SDA is the software-defined radio [2]. In SDAs, applications are defined by the execution of a program and composed dynamically by executing a set of services. This approach provides a high flexibility that fully enables multi-standard applications, and eases the development of advanced functionalities. SDA is therefore a key technology for advanced signal processing applications.

Among embedded applications, many of them are characterized by low production volumes. This is for instance the case of professional applications, product preseries, application prototypes, or high-end consumer products. Because of their limited production volumes, those applications cannot amortize easily high non-recurring engineering costs (NREs) caused by the development of a new application and the platform realization. For low-volume applications, NREs are therefore a major issue.

Platforms used to implement low-volume SDAs must therefore meet the requirements of embedded systems, advanced signal processing applications, SDAs and low-volume applications. Those requirements are: (i) the platform must provide a high power efficiency, (ii) it must support high computational load, (iii)

it must support the high flexibility required by SDAs. Finally, it must also keep NREs at a relatively low level by (iv) reducing the application development efforts and (v) reducing the costs related to the hardware platform development.

Semiconductor technology evolution conflicts with those requirements. Indeed, technology evolution is characterized by regular transistor downscalings allowing to fit more transistors in a same area. Until deep submicron nodes, this scaling was accompanied by significant dynamic power reductions and operating frequency increases. However, after these nodes, limitations have appeared that strongly limit those improvements [3]. Moreover, production processes have become significantly more expensive at each new node. Consequently, in deep submicron nodes, each new node provides more transistors, but very limited improvements in power and frequency, and high increases in production costs.

Because of those limitations, technology scaling alone cannot meet the requirements of low-volume SDAs. The platforms used for those applications must therefore find other solutions to provide better power efficiencies and support high computational loads despite the limited gains in power and frequency. A solution for this issue is to exploit high parallelism, new architectures with better power efficiencies, and aggressive optimizations. Moreover, platforms produced in the latest technology nodes are more expensive and cause therefore higher NREs. In order to reduce those NREs, platforms must be highly generic, so they can be used in many applications and amortized on larger volumes.

Conventional solutions for signal processing applications are single-core GPP/DSP and heterogeneous platforms associating a GPP/DSP with application-specific processors and accelerators [4]. However, the functionalities of those platforms are limited by the available accelerators. Moreover, those approaches are highly dedicated to specific applications, and therefore lack genericity. They cannot therefore meet the requirements of SDA. This situation has motivated the development of new platforms like multi-core processors, SIMD processors, coarse-grain reconfigurable platforms, fine-grain reconfigurable platforms like FPGAs, and homogeneous many-core platforms (HMCP).

Two of those platforms are interesting candidates for SDAs in low-volume applications with a high diversity of standards:

- **FGPAs** are highly generic platforms. They can exploit a very high parallelism. Several FPGAs support dynamic partial reconfiguration which allows to dynamically change the functional blocks instantiated in a device. This technique increases the flexibility of FPGA designs and provides the flexibility required by SDAs.

- **HMCPs** are platforms composed of many identical processor cores. Typical HMCPs use up to several hundreds of cores. Those platforms have a very high programmability thanks to good support from the tools. They can exploit a very high parallelism thanks to their numerous cores. They are also very generic since they can be used for many different applications.

However, in FPGA, DPR usage complicates the development, which is a disadvantage for low-volume applications, particularly for applications requiring strict validations or even certifications like professional electronics. Reconfigurations also consume energy. Precise characterization of this energy is therefore required to correctly evaluate the interest of this approach for SDAs.

Regarding HMCPs, the power efficiency of those platforms is limited by the overhead caused by the use of fully programmable processor cores. This overhead strongly depends on the platform core architecture. In order to improve HMCP power efficiency, the impact of core architecture on the platform must therefore be evaluated.

This thesis studies hardware platforms for low-volume embedded SDAs. In particular, two platforms are evaluated: FPGA with dynamic partial reconfiguration, and homogeneous many-core platforms (HMCP). Regarding DPR, this work focuses on the impact on development and power of this technique. Regarding HMCPs, this work focuses on the core architecture impact on the platform power efficiency.

I.2 THESIS CONTRIBUTIONS

The main contribution of this thesis is **a set of analyzes, methodologies, and solutions regarding the use of FPGAs with DPR and HMCPs to implement low-volume software-defined signal processing applications**.

The detailed contributions are:

1. **A comparison of existing SDA platform models with respect to SDA requirements and low-volume applications**:

   Several platform models have been proposed to implement SDAs. In this thesis, five main models are considered: multi-core processors, fine-grain and coarse-grain reconfigurable platforms, SIMD processors and homogeneous many-core platforms. This contribution introduces and describes those models. Evaluations of the models are realized by using criterions directly derived from SDA and low-volume application requirements.

2. **An analysis of the impact of dynamic partial reconfiguration on design and development:**

   DPR allows to modify dynamically the configuration of an FPGA. It allows to provide virtually more hardware over time than physically present. This hardware virtualization provides an increased flexibility

to those platforms that is interesting for SDAs. However, the use of dynamically changing hardware has repercussions at every level of a design, from the bit-level to the operating system. Using DPR has therefore a strong impact on design and development of signal processing applications. This contribution analyzes the benefits of using DPR to implement SDAs in FPGAs. It also analyzes the impact on the design by evaluating the additional complexity introduced by DPR in the design and the development flow. Missing elements of the flow and the tools are identified.

3. **A minimal flow and a set of guidelines to use DPR in applications that require certification**:

   Some signal processing applications, like safety-critical applications, require strict validation and certification to guarantee that the designed product will work as specified. DPR introduces additional complexities in the design and development of applications, and the DPR tools from the vendors are not certified. This makes the certification of a design using DPR extremely difficult. This contribution provides a simple flow that allows to perform DPR, this flow is based on documentation and only uses simple operations, which makes its certification easier. Guidelines are also provided to reduce the impact of the additional issues introduced by DPR for certification.

4. **An analysis of DPR impact on the power consumption of a design:**

   DPR enables hardware virtualization by allowing to manage dynamically the functional blocks instantiated in an FPGA. This technique provides power improvements by instantiating dedicated functional blocks for each needed function. In this contribution, power consumption reduction opportunities of an SDA are evaluated. Measures are realized on a design implementing signal processing applications with DPR. DPR also introduces new sources of power consumption which cause an overhead compared to a conventional FPGA design. In order to evaluate this overhead, measures have been performed to evaluate the power consumed by the reconfiguration process.

5. **A methodology to compare precisely HMCP processor cores at architectural level:**

   HMCP power efficiency strongly depends on the architecture of its processor cores. In order to evaluate the impact of an architecture on the platform power efficiency, platforms must be compared using different core architectures. A methodology is proposed to realize implementations of processor cores that strongly reduce the interferences of their

implementation specificities. Using those cores allows therefore to perform comparisons at the architecture level. Three optimized RISC processor implementations are realized: a single-issue RISC core, a 3-issue and a 5-issue VLIW.

6. **An evaluation of the impact of core architecture on the power efficiency of HMCPs:**

   A comparison of three HMCPs is realized using the three implemented processor cores. Relations are identified that allow to estimate the platform power, frequency and area using the performance obtained for a single core. Using this approach, the comparison allows to compare HMCP at the architectural level. The results provide insights on the core architecture impact on the platform performance.

I.3 OUTLINE

The text is organized as follows:

Chapter 1 introduces SDA and low-volume embedded applications. Hardware platform requirements are identified for low-volume embedded SDAs. Existing platform model candidates for SDA are presented.

Chapter 2 presents a comparison of the existing SDA platforms for the requirements of SDAs and low-volume applications.

Chapter 3 introduces dynamic partial reconfiguration of FPGA. The benefits of DPR for SDAs are discussed and the impact of this technique on design and development is evaluated.
The content of this chapter has been published in [JP1, BC1, CP2, CP5].

Chapter 4 presents a minimal flow capable of realizing DPR on FPGAs. The interests of this flow for applications requiring strict certifications like safety-critical are discussed. Guidelines are also introduced to further ease DPR-enabled design certification.
The content of this chapter has been published in [CP2].

Chapter 5 presents an evaluation of DPR impact on FPGA design power consumption. Power consumption reduction opportunities are evaluated with measures performed on a real application using DPR. The additional power consumption sources introduced by DPR are also measured and discussed.
The content of this chapter has been published in [BC1, CP1, CP5].

Chapter 6 presents a methodology for processor core implementations that enables precise comparisons at the architectural level.
The content of this chapter has been published in [CP7].

Chapter 7 presents an analysis of processor core architecture impact on the power efficiency of an homogeneous many-core platform.

Finally, the conclusion presents a summary of the thesis main contributions and results.

AUTHOR'S PUBLICATION LIST

Journal paper

JP1. Philippe Manet, Daniel Maufroid, Leonardo Tosi, Gregory Gailliard, Olivier Mulertt, Marco Di Ciano, Jean-Didier Legat,1 Denis Aulagnier, Christian Gamrat, Raffaele Liberati, Vincenzo La Barba, Pol Cuvelier, Bertrand Rousseau, Paul Gelineau, "An Evaluation of Dynamic Partial Reconfiguration for Signal and Image Processing in Professional Electronics Applications", in EURASIP Journal on Embedded Systems, vol. 2008, Article ID 367860, 11 pages, 2008.

Book chapter

BC1. B. Rousseau, P. Manet, T. Delavalléee, I. Loiselle, and J.-D. Legat, Dynamically Reconfigurable Architectures for Software-Defined Radio in Professional Electronic Applications. Springer, (to be published).

Conference papers with peer review

CP1. Philippe Manet, Bertrand Rousseau, Vital Kuti Lusala Tsumbu Mbi, Jean-Didier Legat, "Opportunité d'utiliser la reconfiguration dynamique pour optimiser la consommation d'applications implémentées en FPGA", in Faible Tension Faible Consommation (FTFC), Paris, France, pp. 87-91 , May 2007.

CP2. Bertrand Rousseau, Philippe Manet, Denis Galerin, Diego Merkenbreack, Jean-Didier Legat, Fabienne Dedeken, Yves Gabriel "Enabling certification for dynamic partial reconfiguration using a minimal flow", in Design Automation and Test in Europe (DATE), Nice, France, pp.983-988, April 2007.

CP3. Vital Kuti Lusala Tsumbu Mbi, Philippe Manet, Bertrand Rousseau, Jean-Didier Legat, "NOC implementation in FPGA using torus topology", in International Conference on Field Programmable Logic and Applications (FPL), Amsterdam, Netherlands, August 2007.

CP4. Vital Kuti Lusala Tsumbu Mbi, Philippe Manet, Bertrand Rousseau, Jean-Didier Legat, "Mise en oeuvre d'un NoC avec QoS à l'aide des circuits virtuels", in Faible Tension Faible Consommation (FTFC), Louvain-la-Neuve, Belgium, pp. 141-144 , May 2008.

CP5. Thibault Delavallée, Bertrand Rousseau, Philippe Manet, Hans Vandierendonck, Jean-Didier Legat, "Modeling the impact of bitstreams transfer in dynamically reconfigurable platforms", in Faible Tension Faible Consommation (FTFC), Louvain-la-Neuve, Belgium, pp. 145-149 , May 2008

CP6. Igor Loiselle, Bertrand Rousseau, Philippe Manet, Hans Vandierendonck, Jean-Didier Legat, "Virtualization analysis of a reconfigurable region for adaptive DSP and multimedia applications", in Faible Tension Faible Consommation (FTFC), Louvain-la-Neuve, Belgium, pp. 157-161 , May 2008.

CP7. B. Rousseau, P. Manet, I. Loiselle, J.-D. Legat, and H. Vandierendonck, "A methodology for precise comparisons of processor core architectures for homogeneous many-core dsp platforms," in Proceedings of Design and Architectures for Signal and Image Processing conference, pp 76-83, Edinburgh, October 2010.

Workshops, abstracts and posters

WP1. Thibault Delavallée, Igor Loiselle, Philippe Manet, Bertrand Rousseau, Jean-Didier Legat, "Automatic HDL code generation for system-level design", in Architectures and Compilers for Embedded Systems (ACES), Edegem, Belgium, pp.27-30 , October 2006.

WP2. Philippe Manet, Bertrand Rousseau, Jean-Didier Legat, "Using Dynamic Reconfiguration of FPGA for telecommunication applications", in Architectures and Compilers for Embedded Systems (ACES), Edegem, Belgium, pp.38-41 , October 2006.

WP3. Bertrand Rousseau, Philippe Manet, Jean-Didier Legat, "A coarse grain reconfigurable architecture for sensor networks", in Architectures and Compilers for Embedded Systems (ACES), Edegem, Belgium, pp.35-37, October 2006.

WP4. Bertrand Rousseau, Philippe Manet, Vital Kuti Lusala Tsumbu Mbi, "Opportunity for power consumption optimizations using dynamic partial reconfiguration on FPGA", in Architectures and Compilers for Embedded Systems (ACES), Edegem, Belgium, pp.21-24, September 2007.

WP5. Vital Kuti Lusala Tsumbu Mbi, Bertrand Rousseau, Philippe Manet, "Building multiprocessor system with torus NoC in FPGA", in Architectures and Compilers for Embedded Systems (ACES), Edegem, Belgium, pp.65-68, September 2007.

CHAPTER 1

SOFTWARE-DEFINED SIGNAL PROCESSING APPLICATIONS AND PLATFORMS

1.1 INTRODUCTION

This chapter presents software-defined signal processing applications and identifies hardware platform requirements for the implementation of an SDA in low-volume applications. It highlights several aspects that are: advanced signal processing applications, software-defined applications, low-volume applications, and technology limitations. They all bring constraints for the hardware platform implementation.

This chapter also introduces existing hardware platform models for SDA. In this work, five main platform models are considered: multi-core platforms, coarse-grain reconfigurable platforms, fine-grain reconfigurable platforms (through FPGAs), SIMD and homogeneous many-core platforms. Each model is briefly described, and existing platforms are presented.

This chapter is organized as follows: in the first section, the evolution of signal processing applications towards advanced signal processing applications is explained. In the next section, the shift of signal processing application implementations to software-defined applications is presented. The benefits of this approach are motivated, and their specific constraints are detailed. In the following section, low-volume embedded applications are presented. After, the limitations introduced by technology scaling in the latest technology nodes and the conflicts with SDA and low-volume applications are discussed. Finally, the five

major requirements for SDA are identified and existing platform models for the implementation of SDA are described.

1.2 ADVANCED SIGNAL PROCESSING APPLICATIONS

Signal processing applications evolve towards applications supporting a very high standard count and providing advanced functionalities. For instance, in the telecommunication field, mobile terminals must support several standard generations (e.g. GSM, UMTS, LTE, 4G), as defined by the roadmaps of this field. In the same way, in the military domain, next generations of radio systems must support several modulations, and even be capable of loading dynamically new modulations [6]. This evolution is not limited to telecommunication applications. In video applications like HDTV set-top boxes, a high number of video decoding standards must be supported, while providing advanced services to customers.

The data rates of those applications can be very high. For instance, 4G future mobile communication standards foresee data rates up to 1Gb/s. Those data rates allow to support applications like videoconference. In the same way, video applications work with very high definitions.

In order to provide access to those high data rates and definitions, signal processing operations in those applications have become more elaborated. For instance, the H.264 video standard introduces improvements like variable precision down to quarter pixel for motion estimation [7]. In communication applications, future applications foresee auto-adaptive standards, and even cognitive radios [2].

Applications also provide advanced functionalities which become similar to those provided by desktop PCs. This is for instance the case in surveillance applications, where smart cameras can record and process video streams in real-time in order to analyze their contents [1]. Those systems perform complex operations on the video streams to provide functionalities like motion recognition, object or person identification, etc. Those advanced functionalities, together with more elaborated algorithms, introduce increased control and software parts in those applications.

1.3 SOFTWARE-DEFINED SIGNAL PROCESSING APPLICATIONS

Simultaneously with the evolution of signal processing applications, signal processing platforms have progressively shifted from platforms supporting only the functions required by a single standard to advanced platforms providing access to several standards as a set of services. For instance, set-top box platforms provide applications the possibility to decode many different video streams [8]. In those platforms, standard implementations have evolved from a complete definition with hardware to a partial or full software implementation. In this thesis, those software implementations are regrouped under the name software-defined

signal processing applications, or SDA. In the telecommunication field, a popular advanced signal processing application is the software-defined radio [2]. This application aims to virtualize a complete modulation chain by using software.

With the SDA approach, a signal processing application is defined by a program execution. This approach allows to define flexibly signal processing operations. This feature fully enables multi-standard solutions. The software approach also eases the implementation of advanced functionalities introduced by desktop PC-like applications or even highly adaptive systems. SDA is therefore a key technology for the support of advanced signal processing applications.

1.4 LOW-VOLUME EMBEDDED APPLICATIONS

Embedded systems are dedicated to the execution of specific functions, generally under real-time constraints. They are used in many different application fields, like telecommunications, consumer electronics, aerospace, military, industrial control, etc.

The embedded application market is mainly led by consumer electronics. In these applications, there is a limited number of products (GSM, GPS, netbook, etc.) and large user groups, which allows to produce in very high volumes. For instance, GSMs are produced in tens of millions of units. In this market, signal processing applications are based on standards defined by standardization organizations, like the IEEE. This approach allows product developers to share those standard definitions and to agree on a precise set of standards. Together with the large market size, it also allows to create a viable IP ecosystem. The product platform can therefore be composed by assembling IPs on a chip, like a System-on-Chip for instance. The high production volumes allow to amortize the high costs of those platforms dedicated to a small set of products.

Other applications are characterized by lower volumes. It is the case for instance of professional applications, product preseries, application prototypes, or even high-end consumer products. Preseries are products allowing to test the commercial viability of a product on the market. Prototypes are R&D applications in the industry.

In low-volume applications, the field of professional electronics represents a very large set of applications from many domains: military, avionics, industrial control, aerospace, oil drilling, power plants, etc. This application group is composed of a very high number of different products, and therefore designs. Indeed, the needs of a given product strongly depend on its operating environment. For instance, a radar system has not the same requirements if it is used in a control tower, or in an airplane. Those products are therefore dedicated to the specific needs of a small user group. To meet their needs, those groups define and work with their own standards. Standard definitions are generally kept private for confidentiality reasons, or simply because their specificities do not justify sharing them. This high standard diversity and small user group limit IP availability.

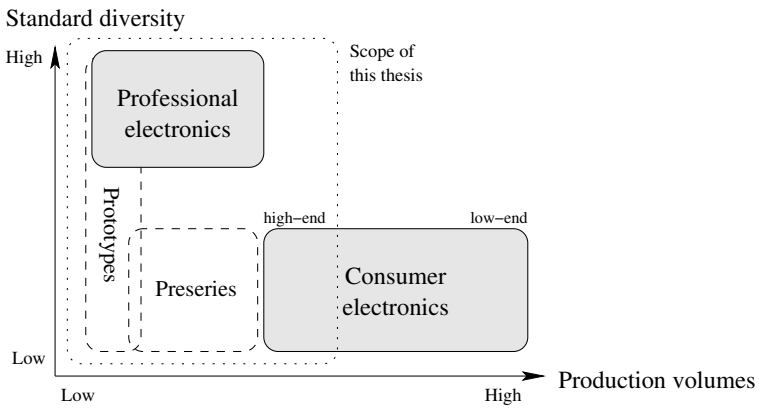

Fig. 1.1. Production volumes and standard diversity for several application groups.

For applications supported by an embedded system, the total cost is composed of non-recurring engineering costs (NREs) and production costs. In low-volume applications, most platforms are used for several applications. NREs of a given application are therefore composed of NREs required to port a new application to a platform, and NREs required to realize the platform. Due to their limited volumes, low-volume applications cannot amortize high NREs. The cost per unit, however, can be very high.

Low-volume applications, and more particularly professional electronics, have a very high product diversity and a low standardization. There is therefore no large IP provider market for those applications, unlike in consumer electronics. This incurs large development efforts and therefore higher NREs. Moreover, some applications like professional applications must provide a high level of quality. Those require strict validations, or even certifications which make developments even more complex. This situation stresses even more the NRE issue.

Figure 1.1 shows the previously presented applications depending on their production volumes and their standard diversities. Consumer applications have very high volumes for low-end products, and reduced volumes for high-end products. Those applications have a relatively low standard diversity. Product preseries have a low standard diversity and low-volumes. Prototypes are very-low-volume applications. They represent a large range of standard diversity since they are used in many different fields. Finally, professional electronic applications have low-volume, but a very high standard diversity.

The set of low-volume to mid-volume applications defines this thesis scope. In those applications, NREs are a major issue. When considering the platforms used to implement SDAs for those applications, it is therefore crucial to keep the costs related the application development and the costs related to the platform relatively low.

1.5 IMPLEMENTATION REQUIREMENTS FOR LOW-VOLUME EMBEDDED SDA

SDAs dynamically define the system functionality by executing a program. The platform must therefore allows modifications of its functionality with the flexibility of software. Moreover, in order to support the high data rates of advanced signal processing applications and their elaborated operations, the platform must also support a very high workload.

Power consumption is also a major issue for the implementation of those applications on embedded systems. Indeed, those systems are powered by batteries, or must face cooling issues which are difficult to manage in integrated environments. They must therefore work with a very limited power budget [9].

The section 1.4 has shown that NREs are an important issue for low-volume embedded applications. It is therefore crucial that the platform allows to reduce the development efforts and the platform costs.

Those constraints define several requirements for the implementation of low-volume embedded SDAs:

- The platform must support dynamic changes of its functionality with the flexibility of software.
- It must support a high workload in order to face the high data rates and elaborated operations of advanced signal processing applications.
- It must have a low power consumption in order to respect the limited power budget of embedded systems.
- It must limit NREs related to the development efforts and the platform.

1.6 SEMICONDUCTOR TECHNOLOGY

The semiconductor industry follows roadmaps defined by the ITRS which states that the available transistor count in a chip should double about every two years [10, 3]. This objective is reached by transistor scaling. Until deep submicron nodes, technology scaling was accompanied by significant reductions in dynamic power and increases in switching speed. After these nodes, technology scaling has started to provide less benefits. In particular, leakage currents have risen significantly. In order to reduce those currents, the threshold voltage needs to be high. Using this solution has however limited the speed increases [11] and prevented the diminution of the supply voltage, limiting therefore the reductions in dynamic power consumption [12]. Technology scaling in deep submicron technologies provides therefore more transistors on a same die area, but limited gains in speed and power reductions. The provided power reduction is not sufficient to allow to fully use the $2\times$ more available transistors in the same power budget.

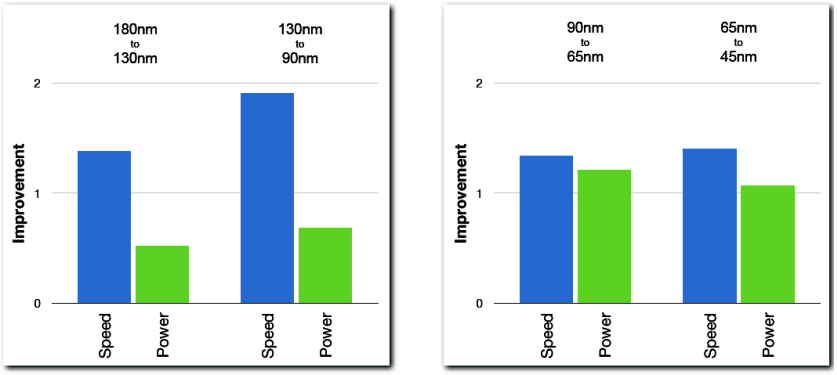

Fig. 1.2. Improvements in power and speed from 180nm to 45nm technology nodes. Picture is courtesy of ARM, ltd [3]

Furthermore, the production process has become more complex at each node and consequently far more expensive.

Figure 1.2 shows successive technology improvements from 180nm to 45nm nodes. The figure shows a speed improvement together with power reduction when going from 180nm to 130nm, and 130nm to 90nm. However, the same figure shows speed improvements with an increase in power consumption when going from 90nm to 65nm, and 65nm to 45nm. This shows that the performance need to be traded off with the power consumption.

Therefore, in deep submicron technology, each new node provides:

- 2× more transistors.
- Limited increases in operating frequencies.
- Very limited improvements in power consumptions.
- Higher production costs.

Technology limitations conflict with the requirements of SDA and low-volume applications. Indeed, SDA platforms must support a high level of programmability, which incurs a power consumption overhead caused by software execution. Since embedded systems must provide a very high power efficiency, this trend conflicts with the limited power reductions of the latest technology nodes. Moreover, the high workloads of SDAs are also conflicting with the very limited improvements in operating frequencies. Finally, the successive cost increases of the technology nodes make the platforms significantly more expensive to produce. This causes huge NREs when producing a platform in the latest technologies.

Consequently, due to those conflicts, technology scaling cannot alone meet the requirements of SDAs and low-volume embedded systems. The platforms used to implement those applications must therefore exploit other approaches:

- In order to support a high workload without increases in operating frequencies, the platform must exploit a high level of parallelism.
- Power consumption reduction must be obtained by exploiting new architectures with a better power efficiency, or by performing aggressive optimizations on the designs.
- In order to reduce the NREs, the platforms must be highly generic. Indeed, generic platforms can be used in a higher number of applications, and therefore amortized on a higher volume.

1.7 SUMMARY OF HARDWARE PLATFORM REQUIREMENTS FOR LOW-VOLUME EMBEDDED SDA

In section 1.2, it has been shown that advanced signal processing applications introduce more elaborated operations, advanced functionalities, and higher data rates. In section 1.3, it has been shown that signal processing applications evolve towards software-defined applications in which the platform functionality is defined by the execution of a program. Section 1.4 has presented the NRE issues of low-volume applications and platforms. Finally, section 1.6 has introduced the technology limitations caused by technology scaling.

Those observations allow to identify five critical requirements that a platform used to implement low-volume embedded SDA must respect:

1. The platform must support dynamic changes of its functionality with the flexibility of software in order to fully support the SDA approach.
2. It must support a high computational load in order to face the high data rates and elaborated operations of advanced signal processing applications.
3. It must have a high power efficiency in order to respect the limited power budget of embedded systems. Moreover, this power efficiency must be obtained by exploiting architectural techniques or aggressive optimizations on the design in order to solve technology limitations.
4. In order to reduce the NREs related to the application development, the platform must be easily programmed to reduce development efforts.
5. In order to reduce the NREs related to the platform itself, it must be highly generic. It allows to use the platform in many applications, and therefore amortize the costs on a larger volume.

In the following section, existing SDA platforms are presented. For each platform model, a short description is given. At the end of the chapter, a summary of the presented platforms is made. In the next chapter, the platforms will be compared using the five requirements that have been identified here.

1.8 EXISTING PLATFORMS FOR SDA

1.8.1 Issues with conventional heterogeneous platforms

At present, platforms for signal processing are mainly represented by two approaches: single-core GPP/DSP, and heterogeneous platforms associating a GPP/DSP with application-specific processors and accelerators [4] like illustrated in Figure 1.3. This last solution can take the form of highly dedicated platforms with many custom IPs like Systems-on-Chips [13].

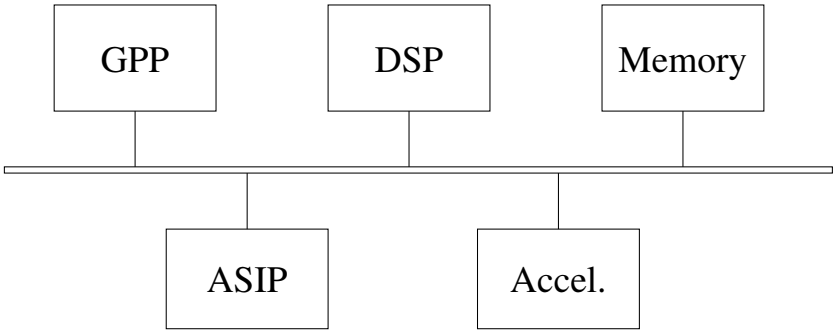

Fig. 1.3. Example of heterogeneous platform for signal processing.

Those platforms show several issues in regard to SDA requirements. Single-core platforms can only exploit a limited parallelism, and generally work at high frequencies. As such, they provide limited computational throughputs and have low power efficiencies. GPPs/DSPs associated with accelerators allow to get better performance. In those platforms a processor acts as a controller which offloads kernels with high computational loads to dedicated optimized functional blocks. The flexibility of this approach is limited since the available functions are limited by the accelerators present in the platform. Moreover, using custom IPs limits the genericity of the solution.

The limitations of those platforms have motivated researches to find new platforms capable of meeting the requirements of SDA. In this work, five main platform models are considered: multi-core processors, SIMD processors, fine-grain reconfigurable platforms, coarse-grain reconfigurable platforms, and homogeneous many-core platforms. In the next sections, each of those five models are presented. Heterogeneous platforms are not considered in this work since they act as a refinement of those main platform models. Indeed, the processor cores used in heterogeneous platforms are instances of those main models. Moreover, since those platforms are dedicated to specific application fields, they lack genericity for low-volume applications.

1.8.2 Multi-core processors

Multi-core DSPs are composed of few cores (e.g. 2-4 cores) with private L1 caches. Signal processing applications are implemented on those platforms by dividing them in several threads and distributing their execution on the available cores. To support high computational loads, multi-core platforms use complex processor cores with high IPCs, like for instance VLIW cores with more than 4 execution issues, which operate at high frequencies.

Inter-core communications can be done in several ways. Those platforms implement complex memory systems with an address space shared between all the cores, together with synchronization and cache coherence mechanisms implemented in hardware. The most frequent communication mean consists therefore in using shared caches that are managed by the memory system. Other means include the use of register-mapped FIFOs or DMA transfers to local scratchpads. Several means can also be combined. Figure 1.4 represents a typical multi-core platform with 4 cores. Each core has local L1 memories for instruction and data and a L2 memory cache is shared between all the cores. Typical L2 memories provide more than 1MB whereas L1 memories only provide several KB (e.g. 8-32KB). The cores are interconnected by a shared bus that provides access to external memories, IOs, and peripherals.

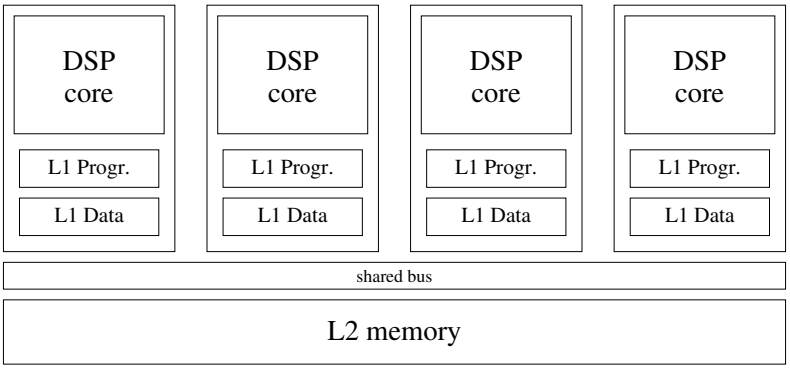

Fig. 1.4. Typical configuration of a multi-core DSP architecture. The core have each their local L1 memories and share a large L2 memory. A shared bus provides access to external memories, IOs and peripherals.

Examples of multi-core platforms are the TCI6487 [14] and SB3011[15]. The TCI6487 is a multi-core platform developed by Texas Instruments. It contains 3 high-performance C64x DSPs connected together by a shared bus. Each C64x core is an 8-way VLIW with 2 clusters that can run at 1GHz. The C64x DSP is a popular DSP architecture for signal processing. This platform offers an easy migration path for signal processing developers to high-throughput signal processing applications like SDAs. The SB3011 is a commercial multi-core DSP that

is developed by Sandbridge. It contains 4 cores that can run at 600MHz. The cores are accompanied by four 4-way SIMD ALUs. Each core supports multi-threading and is capable of executing up to 8 threads simultaneously. SB3011 is claimed to have a fully optimized layout that consumes 2.7× less power than a synthesized version.

1.8.3 SIMD processors

SIMD processors execute instructions that operate on data vectors. Those vectors are generally large, their typical lengths are about 8 to 16 16-bit words. Figure 1.5 represents a typical configuration for a SIMD processor architecture. Performing operations on vectors allows those platforms to exploit the data-parallelism available in DSP algorithms. As some part of a program are inherently sequential, SIMD processors also have a synchronized scalar pipeline dedicated to the execution of those sequential parts. In the vector pipeline, a special stage is required to perform shuffle operations that allows to move data inside a vector.

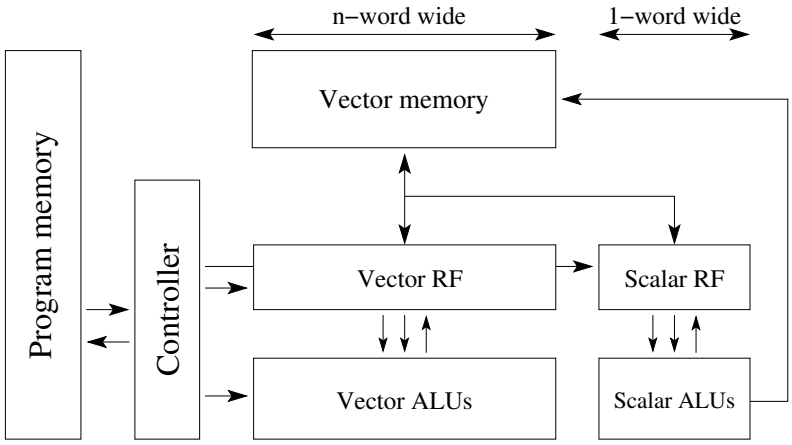

Fig. 1.5. Typical SIMD processor architecture with 1 vector pipeline and 1 scalar pipeline.

Examples of SIMD processors are EVP16 [16] and SODA/AnySP [17, 18]. The EVP16 is a commercial platform developed by NXP. It uses 16-word 16-bit vectors. The core runs at 300MHz in 90nm at a supply voltage of 1V. The EVP16 uses a VLIW architectural model in order to be able to dispatch instructions to its 5 vector and 4 scalar pipelines at each cycle. SODA and AnySP are two SIMD research platforms developed by the University of Michigan. SODA is composed of 4 SIMD processing elements with 32-word vector pipelines, one AGU, and one scalar pipeline. Each processing element runs at a frequency of 400MHz in 180nm technology at 1.8V. AnySP is an improved SODA processor: the vector pipeline is doubled in size and cut in eight 8-word pipelines.

1.8.4 Coarse-grain reconfigurable platforms

Coarse-grain reconfigurable platforms are capable of modifying dynamically the configuration of their datapaths at word-level. Word-level reconfigurability allows to tune a datapath to fit the specific operations performed in a signal processing kernel and to speed up its execution. The reconfigurable datapaths are grouped together and form a programmable accelerator for signal processing. Figure 1.6 represents a typical coarse-grain reconfigurable platform: a processor, which handle control-oriented parts of the code, is coupled to the coarse-grain reconfigurable accelerator. The accelerator can be loosely coupled (e.g. through a bus) or tightly coupled to the processor. (e.g. by using a shared register file)

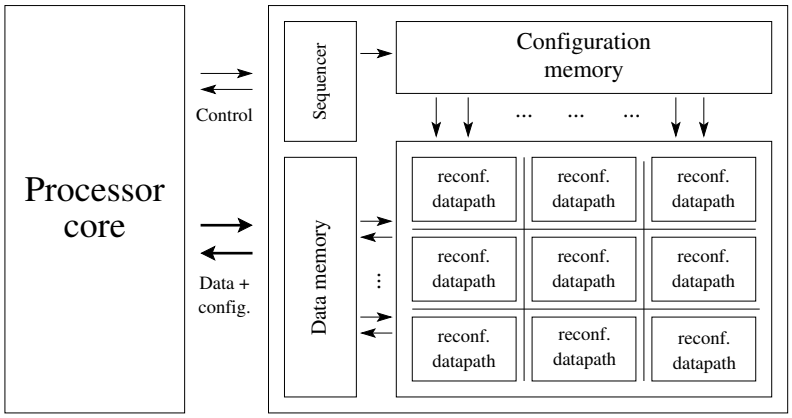

Fig. 1.6. Typical coarse-grain reconfigurable platform.

During kernel execution the datapaths in the accelerator are reconfigured dynamically by configuration words stored in dedicated memories. A sequencer is responsible of fetching the successive configuration words from the memories. When the processor has to execute a kernel on the accelerator, it loads configurations and data in the accelerator, and then starts the configuration sequencer.

Examples of coarse-grain reconfigurable platforms are ADRES [19], the Montium tile processor [20] and the XPP-III processor [21]. ADRES is a research platform developed by IMEC. It is a coarse-grain reconfigurable array composed of 16 tiles with 64-bit reconfigurable datapaths. Each tile has a local register file, and can communicate with other tiles. The array of reconfigurable tiles is tightly coupled to a 3-way VLIW processor. ADRES runs at 400MHz in 90nm at 1V.

The Montium tile processor is a coarse-grain reconfigurable processor composed of five ALUs, each associated with two local memories. Montium is a former research platform of the university of Twente that is now commercialized by Recore Systems. The Montium tiles are designed to be used on heterogeneous multiprocessor systems-on-chips where they act as programmable accelerators.

Each tile contains a communication and configuration unit (CCU) that fetches and receives data, and receives configurations to program the tile.

The XPP-III is a coarse-grain reconfigurable processor developed by PACT. XPP-III is composed of an array of dataflow processing elements coupled with one or several processors. The processor cores allow to handle control-oriented code. They use the array to accelerate the execution of signal processing kernels. Flow graphs are mapped on the array by routing together multiple processing elements. Control signals allow to control the data stream between the tiles. Multiple flow graphs can coexist simultaneously in the array, and they can be reconfigured independently from each other, which provides the possibility to map dynamically new graphs during execution.

1.8.5 FPGAs

FPGAs are fine-grain reconfigurable platforms that can be reconfigured at the bit level. FPGAs are composed of configurable look-up tables, registers and multiplexers grouped together in clusters. Communications between clusters are realized by configurable routing resources. Recent FPGAs offer a large amount of reconfigurable resources. For instance, current high-end FPGA components like the Virtex 6 LX-760 offer about 760.000 configurable logic cells [22]. They also provide optimized hard macros like on-chip memory banks and high-speed serial IOs. Some FPGAs also provide processor cores as hard macros.

FPGAs are typically configured at boot-time, and keep their configurations during the whole duration of the application execution. However, several FPGAs support dynamic partial reconfiguration (DPR) of their resources. DPR allows to modify a part of the logic without interrupting the non-reconfigured logic. It enables dynamic instantiations of functional blocks in the device resources during runtime. Dynamic partial reconfiguration of FPGAs is described more thoroughly in Chapter 3.

Xilinx [23] and Altera [24] are the main suppliers of FPGAs with a large amount of reconfigurable resources. Xilinx FPGAs were the first to support dynamic partial reconfiguration. Altera has announced support for DPR in its product roadmap [25].

1.8.6 Homogeneous many-core platforms

Homogeneous many-core platforms (HMCP), illustrated in Figure 1.7, are composed of numerous identical cores. HMCPs can reach several hundreds of cores. Whereas multi-core platforms only exploit around 8 processor cores or less, HMCPs reach high execution parallelism by dividing a streaming application in small independent actors, and mapping them on many separate cores. Combined together, the high number of available cores allows to reach a high computational throughput. Consequently, HMCPs do not require high-IPC processor cores like in multi-core platforms. Their cores are typically simple RISC cores with a few

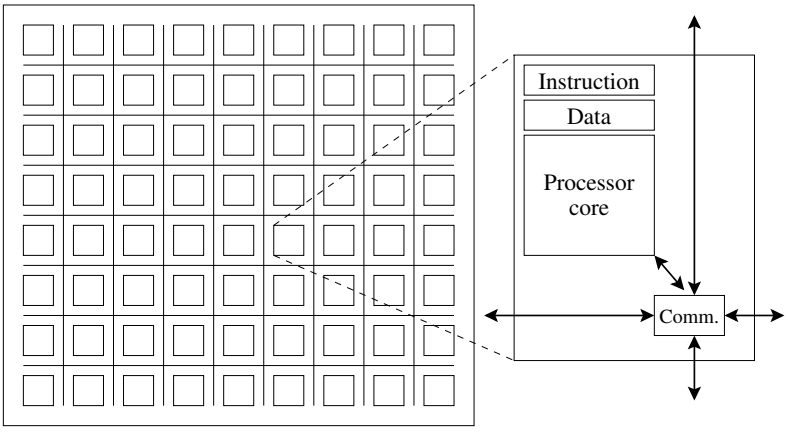

Fig. 1.7. Typical many-core platform architecture. The platform contains a high number of identical cores. The cores have each their local memories for data and instructions. They communicate together using a scalable communication structure.

execution issues, ranging from a single-issue core to a 3-way VLIW core. Those cores also operate at moderate frequencies, i.e. 300-600MHz.

HMCP communication mechanisms are simpler than in multi-core platforms. They exploit highly-scalable communication networks whereas multi-core platforms typically use shared buses, possibly organized in a hierarchical fashion [26]. Additionally, those platforms generally do not implement any cache coherence mechanism. The cores have local memories and communicate data by explicitly sending it to other cores using DMA transfers or packets.

Examples of HMCPs are AsAP [27], PC102 [28], Ambric [29], and Tile64 [30]. AsAP is a research platform from UC Davis. It is a many-core platform composed of 164 identical cores and 3 accelerators for telecommunication and video kernels. AsAP is a globally synchronous, locally asynchronous (GALS) platform: each core has its own local oscillator and voltage controller. At 1.2V, the cores run at 594MHz. Each core has small local memories to store instructions and data. They communicate with each other using FIFOs. PC102 is a commercial many-core platform developed by picoChip. The PC102 is composed of 308 3-way VLIW cores running at 300MHz each. Communications between cores are realized by using a configurable circuit-oriented grid of inter-core links. The Ambric Am2045 is a commercial platform composed of 336 single-issue cores running at 300MHz. Each core has 32-bit ALUs capable of dual 16-bit operations. The cores communicate with each other using communication means similar to FIFOs. Tile64 is a commercial many-core platform developed by Tilera. The Tile64 platform is the commercial version of the MIT RAW processor [31]. The platform is composed of 64 cores. The Tile64 targets control-intensive applications like deep packet inspection or use for cloud computing. The cores communicate between them using an elaborated network-on-chip composed of several layers with differenti-

ated traffics. Tile64 also supports cache coherence mechanisms between the L2 caches of each core. Each core runs at 700MHz in 90nm technology at 1V.

1.9 SUMMARY OF EXISTING PLATFORMS

Table 1.1 summarizes the platforms that have been presented in this section. One can make the following observations:

- In multi-core platforms, the TCI6487 multi-core platform runs at a high frequency, i.e. 1GHz). This is typical for the high-performance cores it uses.
- HMCPs have bigger areas. This is explained by their very high issue counts compared to other platforms. Among those platforms, the Tile64 platform runs at a particularly high frequency. Indeed, this platform also targets high-speed applications like deep packet inspection or cloud computing.
- The coarse-grain reconfigurable platforms, except ADRES, run at low frequencies. Montium and XPP-III platforms run at respectively 100MHz and 150MHz. Montium is a small tile composed of only 5 ALUs. Montium are targeted to be used as accelerators in heterogeneous SoCs [32].

In Figure 1.8, the platforms are illustrated in function of their power and throughput. In this figure, the throughput values correspond to the maximal theoretical throughput estimated by $T_{p,max} = F_{max} \times N_{issues}$, where F_{max} is the platform maximal operating frequency and N_{issues} is the platform issue count. The throughput is estimated for 16-bit instructions since this is the most used data width in DSP applications. Diagonal lines are the points with the same power efficiency.

As the figure shows, SDA platforms differ significantly among them by their throughput values. This difference between the higher and lower values represents more than 2 orders of magnitude on the whole platform set.

Together with the platforms presented in this chapter, additional platforms have been added in the figure. Those platforms are illustrated with white squares. Two of those additional platforms are the $DSP1$, $VLIW3$. Those two platforms are single-core RISC platforms implemented in this thesis. They have respectively 1 and 3 issues, their composition is detailed in Chapter 6. As they are single-core platforms, they provide a very low throughput compared to SDA platforms.

The other platforms are the Cell processor developed by IBM, the G80 GPGPU from NVIDIA, and the Core 2 Duo GPP from Intel. When comparing SDA platforms to those platforms, it is worth noting that those platforms consume significantly more power due to their use in server or desktop computing.

	Type	From	Size	Freq. [MHz]	Voltage [V]	Tech. [nm]	Area [mm^2]	Sources
TCI6487	Multi-core	Texas Instr.	3×8-way VLIW	1000	1.10	65	40.8	[14, 33]
SB3011	Multi-core	Sandbridge	4×8-thread cores	600	0.90	90	-	[15]
AsAP	HMCP	UC Davis	164×1-issue core	594	0.95	65	32.8	[27, 34]
PC102	HMCP	picoChip	308×3-way VLIWs	300	1.20	130	160*	[28, 35]
Ambric	HMCP	Nethra	336×1-issue core	300	1.20	130	145*	[36, 29, 37]
Tile64	HMCP	Tilera	64×3-way VLIWs	700	1.00	90	433*	[30, 38]
EVP16	SIMD	NXP	5 vector + 4 scalar	300	1.20	90	6.4	[16, 39]
SODA	SIMD	U. Michigan	4×32-way + 4 scalar	400	1.80	180	3.5	[17, 40]
AnySP	SIMD	U. Michigan	32×8-way + 32 scalar	300	0.90	65	13.14	[18]
ADRES	Coarse-grain	IMEC	3-way VLIW + 16 tiles	400	1.00	90	5.79	[19]
Montium	Coarse-grain	Recore Systems	5 ALUs	100	1.20	130	1.8	[41, 20, 42]
XPP-III	Coarse-grain	PACT	2 cores + 42 PEs	150	1.00	90	-	[43, 21, 44]
EP3C40	FPGA	Altera	40.000 LE	500	1.20	65	-	[24, 45]

Table 1.1. Summary of SDA platforms main characteristics. Numbers with an asterisk (*) are estimations based on published data.

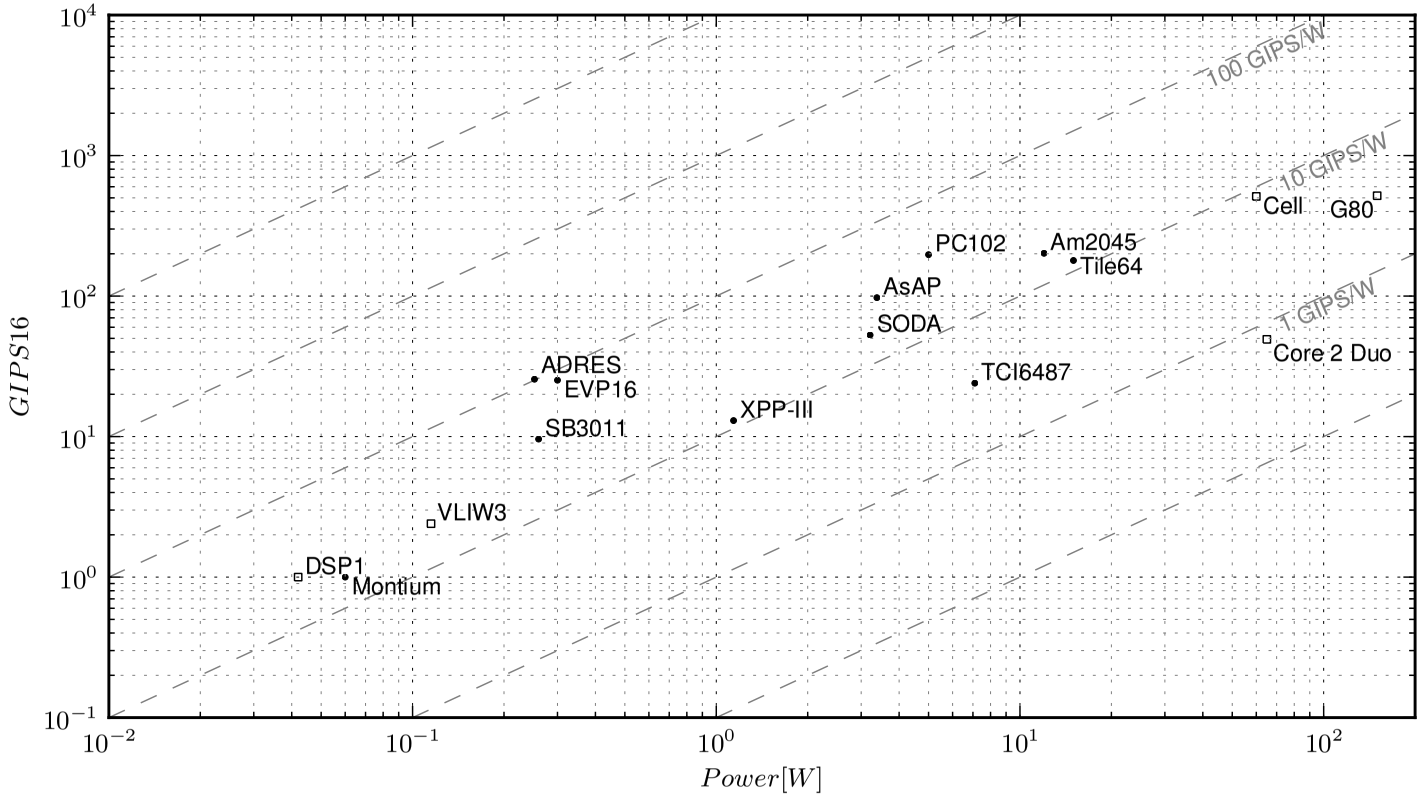

Fig. 1.8. Maximal theoretical computational throughput in billion instructions per second for 16-bit instructions versus power of the presented SDA platforms. Platforms with white squares have been added for comparison purpose. Those are: Cell, NVIDIA G80 GPGPU, Intel Core 2 Duo GPP, $DSP1$ and $VLIW3$. The $DSP1$ and $VLIW3$ are two RISC processor cores developed in this thesis. They are presented in chapter 6.

CHAPTER 2

SDA PLATFORM COMPARISONS

2.1 INTRODUCTION

In the previous chapter, five platform models for SDAs have been identified. For each model, existing platforms have been presented. Those platforms are listed by model in Table 2.1.

Multi-core	**SIMD**	**Coarse-grain**	**Fine-grain**	**Many-core**
TCI6487	EVP16	ADRES	FPGA	AsAP
SB3011	SODA/AnySP	Montium		Am2045
		XPP-III		PC102
				Tile64

Table 2.1. Presented platforms listed by platform models.

In this chapter, the five platform models are evaluated for the requirements of low-volume SDAs. Model evaluations are realized by using criterions directly derived from SDA and low-volume application requirements. Using those evaluations, the five models are compared and their use in low-volume SDAs is discussed.

The text is organized as follows: the first section describes the comparison methodology used in this chapter. The second section discusses the issue of plat-

form model characterization. The next section identifies issues to solve in order to perform the comparison. Section 2.5 describes how voltage and scaling conversions have been applied to compare the platforms in the same reference technology and voltage. Section 2.6 describes the application benchmarks. The following sections present the evaluation of the platform model for all the identified criterions. Finally, a summary compares and discusses the results.

2.2 PLATFORM COMPARISONS

This chapter proposes an evaluation of platform models for SDA. Many of those platforms are not available or require significant development efforts in order to evaluate a complete application. Therefore, implementing a real SDA on all platforms in order to compare them is not possible in the scope of this work. For this reason, it is proposed here to compare the platforms using a set of five criterions that are directly derived from the requirements of SDA hardware platforms and low-volume applications. For each criterion, evaluations are realized for all platform models.

Those criterions are:

1. dynamic function change
2. power efficiency
3. computational load
4. development efforts
5. genericity

Regarding the computational load criterion, platform models can be scaled up or down depending on the throughput required by the application. As SDAs require high throughputs, hardware platforms for SDAs must be scaled up to support high computational loads. However, larger platforms use more silicon area, which increases the production costs. The area efficiency of a platform is therefore important. Moreover, platform models do not all increase their computational loads with the same efficiency. Indeed, platforms with poor scalability will provide less improvements than others when increasing their areas. Consequently, in order to characterize the computational load of several platform models, it is proposed here to take these aspects into account by characterizing their area efficiencies and scalabilities.

Criterions 2 & 3 demand therefore to estimate the power, area and computational load of the platform models. Evaluations of the power and computational load of a platform model can be realized by using tools that estimate the platform performance. For SDA platforms however, such tools are not available and

precise estimations can only be realized by using real platform implementations or precise simulation models like annotated netlists. Many of those platform implementations or models are unfortunately not available. In this chapter, power and computational load values are therefore estimated on the basis of published numbers. In the evaluations, the computational load is expressed in billion instructions per second for 16-bit integer instructions (GIPS16) since most DSP algorithms operate on this type of data.

2.3 PLATFORM MODEL CHARACTERIZATION

In the previous chapter, several DSP platform models have been presented. Each of those models have their own properties that do not depend on applications. For each model, existing platforms have been cited as examples. Those platforms are specific implementations of platform models. They are generally specialized for a particular application field (e.g. video, telecommunication) in order to provide good performance on those applications. All existing platforms possess therefore characteristics that are inherited from their platform models and characteristics that are due to their implementation specificities, which depend on the targeted application type.

In this chapter, the objective is to precisely characterize the presented platform models rather than their specific implementations. Indeed, evaluating a platform model allows to identify the advantages and disadvantages of each model instance. Knowing this, it becomes possible to select the platform model that fits the best the specific needs of an application. In particular, in this work, the platform model characterization is used to identify the best platform models to implement a low-volume SDA. In order to do this, the platform models are evaluated using the five criterions derived from low-volume SDA requirements, as presented in the previous section.

The evaluation of a platform model can be partly realized on basis of the model fundamental properties. Three of the five studied criterions can be evaluated so: dynamic function change, genericity, and development efforts. In this work, those three criterions are therefore evaluated using qualitative discussions based on platform model properties, which are independent from benchmarks. However, some aspects related to platform performance, like power efficiency and area efficiency, can only be evaluated using results obtained on platform implementations. Platform model characterizations can then be made by observing the performance of several instances of the same model, in order to identify common features that can be attributed to the model. However, those implementations are generally specialized for a specific application type. In order to perform evaluations correctly, it is therefore necessary to respect their targeted application field while benchmarking them. This approach allows to obtain representative results.

In this chapter, platform performance evaluations are realized using two benchmarks: the first is a general benchmark evaluating the typical performance

of the presented existing platforms, and the second is a more specific benchmark evaluating the performance of those platforms on a given application. This application is a telecommunication application implementing an OFDM modulation. All the studied platforms target this particular application, and many of them publish performance results for it. The second benchmark allows therefore to evaluate correctly the performance of the platform implementations. By combining the results obtained with the two benchmarks on all the instances of a platform model, one can therefore draw conclusions on the platform model itself.

2.4 COMPARISON ISSUES

Published numbers for each platform are given for specific implementations. Those implementations are realized with different technologies and voltages. In order to realize the comparisons, it is therefore necessary to convert all the used numbers in the same technology. For that reason, all numbers used for the evaluations in this chapter have been converted in the same reference technology. The next section explains how the conversions have been realized.

The computational load of each platform depends on the efficiency of the platform models and on the applications. Indeed, each platform is capable of executing a certain number of instructions per cycle. This number varies depending on the application. In order to realize precise performance comparisons, it is therefore necessary to define benchmarks allowing to take the platform efficiency for given applications into account.

For some models, the concept of instruction is vague or nonexistent. For instance, coarse-grain architectures execute operations in their ALUs. Those operations do not directly correspond to instructions. Another platform causing similar problems is the FPGA. In this platform, there are no instructions at all. In order to enable the evaluation of those platforms, specific benchmarks must also be proposed.

2.5 TECHNOLOGY AND VOLTAGE ADAPTATIONS

The evaluated SDA platforms are made in different technologies and operate at different supply voltages. This prevents the comparison of their performance independently from the variations introduced by the technology. To enable the comparisons, the frequency, power and area values of each platform must be converted to an identical technology and an identical voltage. In this work, the reference technology and voltage are 65nm and 1V. In order to realize this conversion, correction ratios have been used to obtain estimated power, frequencies and area values.

The conversion is realized in two separate steps: first, the technology conversion, and then the voltage conversion. Regarding the technology conversion, correction ratios have been estimated on the basis of frequency and power values

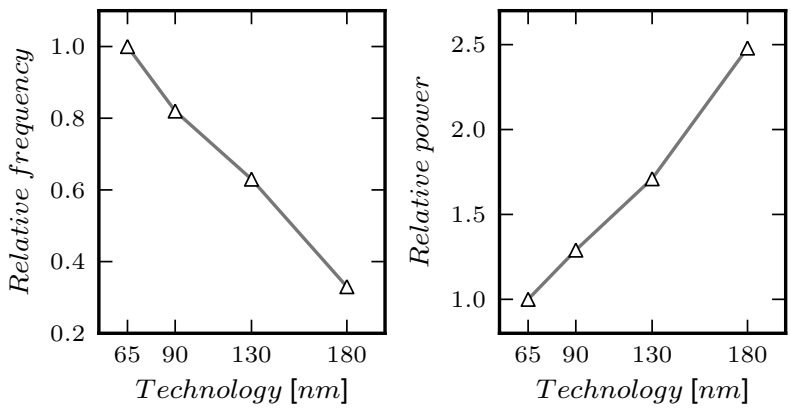

Fig. 2.1. Relative impact of technology scaling on power and frequency.

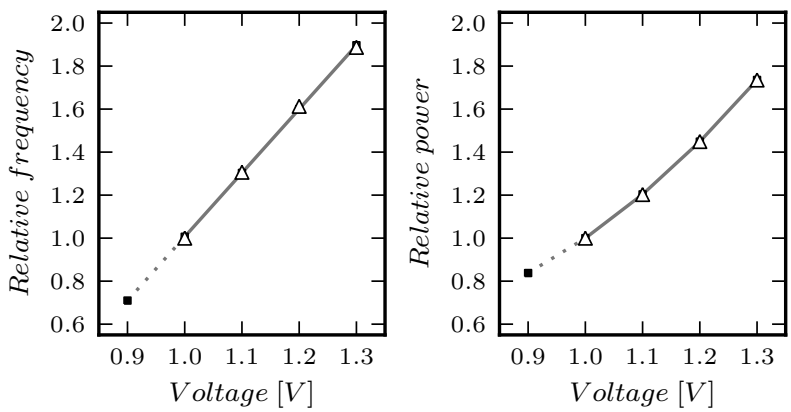

Fig. 2.2. Relative impact of voltage scaling on power and frequency. Values at 0.9V are obtained using polynomial extrapolation.

of a complete processor core realized in several technology nodes of a mainstream bulk commercial process [46]. Figure 2.1 presents the ratios that have been used to convert frequency and power values from 180nm to 65nm. Interconnect delays in integrated circuit wires have increased significantly in deep submicron nodes and have become an important limiting factor for circuit performance. This limits the gains in operating frequencies of those circuits when scaling down the technology [47]. In this work, as the technology conversion factors are based on a complete design including its interconnects, the estimations take this effect into account.

Regarding area, conversions have been realized by multiplying the areas by $1/s^2$ where s is the ratio between the origin technology size and the target technology size. To apply the voltage conversion, correction ratios have been estimated on the basis of the power and frequency values of a design at different supply voltages in a 65nm low-power SVT technology from STMicroelectronics. The ratios are illustrated in Figure 2.2. More details about these estimations are given in chapter 7.

Using correction factors, the power, frequency and area values of the compared SDA platforms have been converted in the reference technology and voltage supply. Figure 2.3 presents the updated platform throughput and power values. As a general effect, the conversions improve the performance of the platforms due to the significant power reductions introduced by both technology and voltage scaling. Frequency values tend to stay similar which is a consequence of the power frequency improvements in deep-submicron nodes.

2.6 APPLICATION BENCHMARKS

The evaluation of the platform computation load is done by estimating the platform throughput, which is the number of instructions per second a platform can execute. To be accurate, the estimation of the computational throughput must take into account the platform efficiency regarding the issue slot utilization. Indeed, in parallel platforms, all issues are not necessarily used at each cycle. In order to do this, the platform throughput must be evaluated by the following formula:

$$T_p = F_{max} \times IPC \tag{2.1}$$

$Fmax$ is the highest operating frequency of the platform. IPC represents the number of instructions that the platform can execute at each cycle. For a given platform, the IPC strongly depends on the application. As the IPC is application-dependent, different benchmarks will lead to different IPC values. In order to characterize the performance of a platform, one generally uses a mean IPC value which is estimated on a set of representative benchmarks. However, not many benchmark results are available due to the limited availability of the studied platforms.

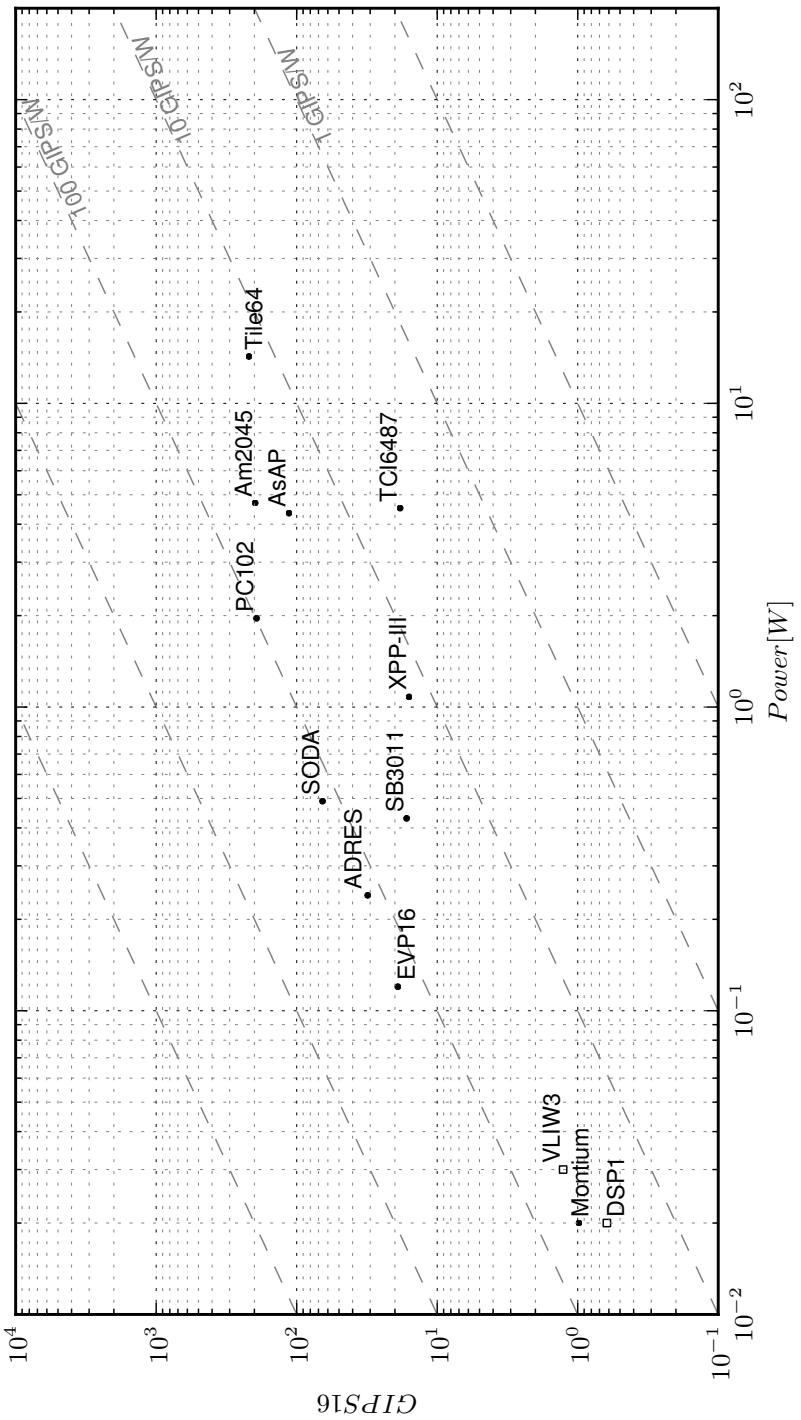

Fig. 2.3. Maximal theoretical computational throughput in billion instructions per second for 16-bit instructions versus power of the presented SDA platforms. Values are all converted to 65nm technology and a supply voltage of 1V. $DSP1$ and $VLIW3$ platforms have been added for comparison purpose. They are two RISC processor cores developed in this thesis. Those cores are presented in Chapter 6.

In this chapter, platform computation load evaluations are based on throughput values representing the platform throughput values for a typical application:

$$T_{p,typ} = F_{max} \times IPC_{typ} \tag{2.2}$$

IPC_{typ} represents the number of instructions that a platform is likely to execute at each cycle for a typical application. This IPC value is lower than the maximal theoretical IPC value attainable by a platform since it will never reach its peak value on a real application. In this work, in order to evaluate $T_{p,typ}$, IPC_{typ} values for each platform are estimated on basis of published numbers. IPC_{typ} values are based either on published typical IPC values, or are computed on basis of published benchmark performance when available. For platforms where no such data are available, IPC_{typ} values are estimated by using typical IPC values of similar architectures whenever possible.

Table 2.2 summarizes the estimated IPC_{typ} values for each platform. The IPC values used for the $T_{p,typ}$ benchmark are estimated for 16-bit instructions. However, in the table, 32-bit IPC values are also given whenever available. The $DSP1$, $VLIW3$ and $VLIW5$ are three 32-bit processor cores developed in this thesis. They are representative of DSP monocore solutions. Their IPC_{typ} values are precisely computed as the average IPC value on 6 DSP kernel benchmarks. Description of the cores and detailed benchmark results are presented in Chapter 6. The cores have a SIMD ALU capable of executing either one 32-bit operation, two 16-bit operations or four 8-bit operations in parallel. Since the cores operate on 32-bit data words, the conversion of the core IPCs to 16-bit IPCs has been performed by precisely counting instructions of an equivalent 16-bit code.

TILE64 IPC_{typ} for 16-bit instructions is given in [30]. The IPC_{typ} value of PC102 cores is given in [28]. Am2045 uses 336 single-issue 32-bit cores, and the core ALUs have some SIMD capability, like the ALUs of the $DSP1$ platform. The IPC_{typ} platform estimations are therefore based on the $DSP1$ 16-bit IPC values. AsAP uses 16-bit single issue RISC cores and can perform at most one 16-bit operation per cycle. Therefore, for this platform, the 32-bit IPC_{typ} value of the $DSP1$ has been used to perform the estimation. EVP16 IPC_{typ} is estimated on basis of benchmark results published in [39]. SODA IPC_{typ} value is estimated based on typical SIMD pipeline utilization data given in [48]. ADRES IPC_{typ} value is estimated on basis of its 64-bit IPC value and the percentage of SIMD operations given in [19]. For the SB3011 platform, not enough data were available to evaluate the IPC_{typ}. The Montium platform is a 16-bit 5-issue core, in order to estimate its IPC_{typ}, the IPC_{typ} of the $VLIW5$ has been used. For XPP-III, the typical number of instructions executed per second and frequency are given in [44], which allows in turn to compute the corresponding IPC_{typ}. As the EP3C40 is an FPGA, there are no instructions, and hence no IPC_{typ} values. Finally, the TCI6487 IPC_{typ} value is given in [33].

Estimating an accurate value for IPC_{typ} requires to evaluate it on a sufficiently big set of benchmarks to be truly representative of the platform performance. However, performance numbers for SDA platforms are generally given

Platform	Width	32-bit IPC_{typ}	16-bit IPC_{typ}
DSP1	32	0.96	1.51
VLIW3	32	2.55	4.01
VLIW5	32	4.11	6.46
TILE64	32	144	192
PC102	16	-	616
Am2045	32	322.56*	507.36*
AsAP	16	-	157.44*
EVP16	16	-	28.07
SODA	16	-	42.40
ADRES	64	-	23.92
SB3011	32	N/A	N/A
Montium	16	-	5.96*
XPP-III	16	-	86.67
EP3C40	N/A	N/A	N/A
TCI6487	16	-	6

Table 2.2. IPC_{typ} values of the compared platforms for 32-bit and 16-bit instructions. Values are estimated on basis of published IPC numbers or benchmark performance numbers. Asterisks ("*") represent platforms where IPC_{typ} values of similar architectures have been used for the estimation.

for a very limited number of benchmarks. If the IPC values vary significantly between applications, the typical IPC value, IPC_{typ}, may be misleading. Indeed, some SDA platforms exploit specific features to reach high parallelism. Those features introduce strong variations in their results depending on the applications. For instance, the performance of a platform could be good for telecommunication applications, but very low for video applications since those applications use a different set of operations (ex: 8-bit operations on pixels). Estimations of the throughputs based on $T_{p,typ}$ have therefore a limited accuracy since it will potentially estimate very different performance depending on the application domain.

In order to address this issue, additional estimations are made by using a reference application. The chosen application is an OFDM symbol modulation for the 802.11a wireless telecommunication standard. This reference application is a software-defined radio application and is currently a common target of existing SDA platforms. This application is widely used by SDA platforms to evaluate their performance, therefore some numbers are available.

Another issue related to throughput estimation is due to the fact that the concept of instruction is vague or even nonexistent in many platforms, especially for coarse-grain and FPGA platforms. In order to solve this issue, executed instruction count values from a reference platform are used for all estimations of the 802.11a benchmark. Those values are obtained with the single-issue RISC

kernel	million instr./s
Symbol mapping	0.17
FFT	1.58
FIR	3.48
Interpolator	4.02
Total	**9.25**

Table 2.3. Instruction per second values used to evaluate computational throughputs of SDA platforms on the 802.11a benchmark. The values are obtained with the $DSP1$ platform for a 802.11a modulation at 54Mb/s.

core that has been designed in this thesis, the $DSP1$. Values for the 802.11a kernels in this platform are presented in Table 2.3.

2.7 POWER EFFICIENCY

SDA platforms require high computational throughputs. Higher throughputs cause higher power consumptions. In order to characterize the energy consumption of a platform for SDA, it is therefore required to consider both the platform throughput and its power consumption. In order to do so, the platforms are compared by their power efficiencies.

In Figure 2.4, the power efficiencies of existing SDA platforms are illustrated for the $T_{p,typ}$ throughput values, and for the 802.11a application with the $DSP1$ MIPS values. Results are given only for platforms that provide enough data to perform the estimation. The $DSP1$ and $VLIW3$ have been included in the comparison to provide reference points.

The results for the $T_{p,typ}$ benchmark show that some platforms, namely EVP16, SODA, ADRES and PC102, obtain significantly higher power efficiencies than others. Those platforms have very high execution issue count. The accuracy of those results depend significantly on the actual utilization of those execution issues. For instance, SIMD platform efficiencies depend heavily on the utilization of the vector issues, and many-core platform efficiencies depend on the number of cores in use. The $T_{p,typ}$ estimations suppose that the vector issues are fully exploited, and that all cores are in execution, which is a situation that is difficult or impossible to obtain in reality. Since the power figures used for the estimations correspond to the actual power consumption of the platforms, the computed power efficiencies overestimate the real performance.

The results for the 802.11a allow to compare the throughputs of the platforms while avoiding to make hypotheses on execution issue utilizations. It also allows to include the FPGA in the comparisons. The results obtained with this metric show that the platforms provide similar power efficiencies with the exception of the FPGA, ADRES and AsAP.

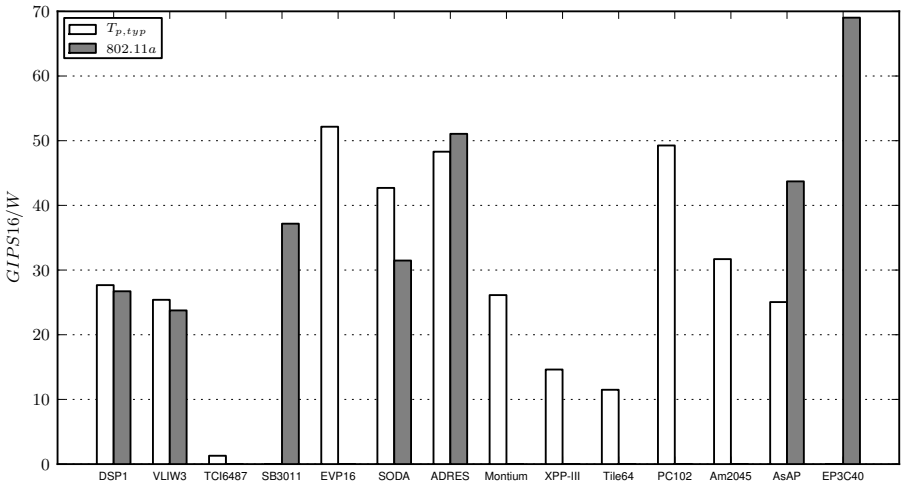

Fig. 2.4. Power efficiencies of the presented SDA platforms, along with the $DSP1$ and $VLIW3$ cores. The throughput values correspond the $T_{p,typ}$ throughputs and the throughputs obtained for the 802.11a application with the reference MIPS values. No bars are present when the data was insufficient to realize the estimation.

The power efficiency of the FPGA platform is very high due to the high throughput that it can reach by using dedicated hardware functional blocks to perform the modulation without incurring the overhead of instruction execution that processors have.

The ADRES platform also provides good results, compared to the $DSP1$ platform, it is almost two times more power efficient. According to the authors [19], this can be explained by the fact that ADRES can perform most of the modulation by instantiating a configuration in its accelerator, and process most of the workload with the same configuration. This situation puts the platform in a steady state regarding configurations, which suppresses the dynamic power consumption of the configuration memories and control wires. This reduces considerably the access count to instruction and data memories, which are the major power consumption sources in the $DSP1$ core, as illustrated in Table 2.4.

AsAP has also high performance on the 802.11a evaluation. This can be explained by the fact that AsAP core clocks are halted whenever the executed thread wait for inputs or outputs. This considerably reduces the power since many cores in their 802.11a implementation spend most of the time waiting.

In Figure 2.4, results show that the TCI6487 multi-core platform perform rather badly, mainly due to its high power consumption. On the contrary, the SB3011 performs well. However, the high performance can be explained by the fact that it is actually a hybrid platform combining SIMD operations, high multi-threading and aggressive optimizations of the design. Indeed, the synthesized

	Power [mW]	%
Core	2.69	33%
Data mem.	1.75	21%
Instruction mem.	3.75	46%
Total	**8.19**	100%

Table 2.4. Power consumption values for the core and memories of the $DSP1$ core. The power consumptions of the memories represent about 67% of the total power.

version of this platform consumes about 2.7 times more power than the optimized version [15]. This indicates that a significant part of the performance actually do not come from its platform model.

Table 2.5 summarizes the evaluation of the power efficiencies of the studied SDA platform models:

	Multi-core	**SIMD**	**Coarse-grain**	**FPGA + DPR**	**Many-core**
Power efficiency	+	++(+)	++(+)	+++	++

Table 2.5. Power efficiency evaluations of the SDA platform models. "+" in parentheses correspond to functionalities that improve the capability of dynamic function change, but that are not present in all instances of the model.

Multi-core platforms use complex memory systems. Those functionalities consume a significant share of the platform power [26]. Moreover, they use complex cores working at high frequencies that provide very good performance on sequential parts of the programs but are less power efficient than simpler cores with lower throughputs like the cores in an HMCP. For instance, the TCI6487 platform uses three C64x DSP cores working at 1GHz. Their high power consumptions lead to low power efficiencies.

Homogeneous many-core platform performance are better than multi-core performance, but their power efficiencies are a bit lower than the others. FPGAs provide very good power efficiencies due to the high throughput that is provided by using highly optimized datapaths instantiated in its configurable logic. SIMD and coarse-grain platforms can give good performance, but results show that those are highly dependent on the ability to use efficiently all the available issues.

It is worth noting that other platforms than SIMD processors also use SIMD operations. For instance, multi-core, many-core and other platforms like CGRAs have ALUs implementing SIMD operations. Those SIMD operations are performed on short vectors of 2 or 4 words encoded in the native data width of the

processor (e.g. 2x16-bit or 4x8-bit in a 32-bit processor). In those platforms, the prime interest of SIMD operations is to maximize the exploitation of the data bus width providing operands to the ALUs. This approach allows to increase the platform throughput by performing several operations in parallel using a single instruction. Those operations can also improve the platform energy efficiency but they do not necessarily render it more efficient than a platform without SIMD blocks. For instance, 16-bit platforms such as PC102 or AsAP have no SIMD blocks, but they have nevertheless power efficiencies that are similar or better than platforms with SIMD ALUs (e.g. Am20145, DSP1). Other techniques (e.g. reconfigurable computing) can also contribute significantly to improve the energy efficiency of a platform.

2.8 COMPUTATIONAL LOAD

In this section, the platform computational load of each platform is characterized by the platform area efficiency and scalability. The scalability of the SDA platform models has been evaluated, the first row of Table 2.6 summarizes the evaluations:

	Multi-core	**SIMD**	**Coarse-grain**	**FPGA + DPR**	**Many-core**
Scalability	++	+	++	++++	++++
Area efficiency	+++	+++(+)	+++	N/A	+++
Summary	++	++(+)	+++	+++	+++

Table 2.6. Evaluation of scalabilities and area efficiencies of the studied SDA platform models. The last row provides an overall evaluation of the throughput of the platform models.

Multi-core platforms have a limited scalability due to their memory systems which do not scale well since it quickly limits the throughput of the cores when their number increase.

On the contrary, homogeneous many-core platforms use scalable communication means, and most of them have simple memory systems with private address spaces. Tile64 is however an exception since it provides a global address space with support for cache coherence. This provides HMCPs with an excellent scalability.

SIMD processors are limited when sizing up their issues, since vector width cannot be increased efficiently above a certain width. Above these values, most algorithms do not fit the issue width anymore, which leads to poor issue and memory utilization. Working on large vectors also make unaligned memory accesses less efficient. Indeed, in order to execute instructions on unaligned data, SIMD processors must recompose a vector from data stored in several vectors in

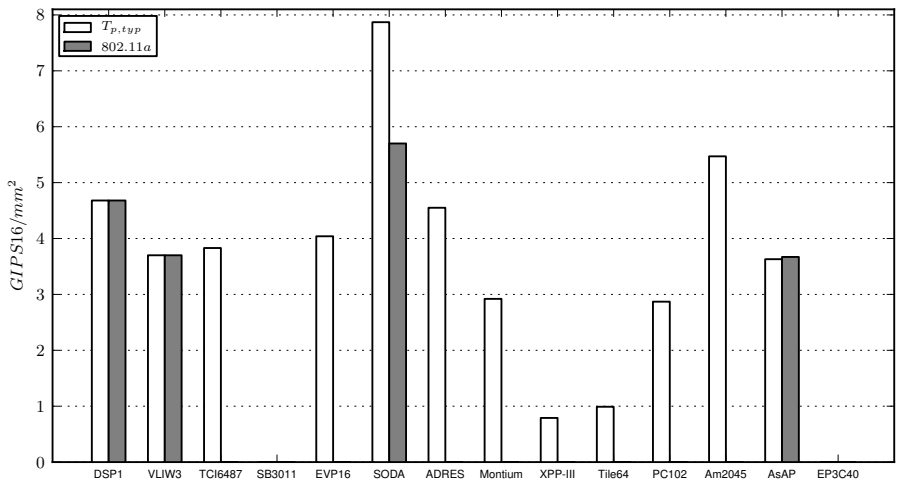

Fig. 2.5. Area efficiencies of the studied SDA platforms, along with the $DSP1$ and $VLIW3$. Areas are evaluated without the data memories.

memory. Those realignment operations require dedicated hardware, like a shuffle network, and take several execution cycles to perform. This introduces overheads in area, power, and latency. Those overheads strongly reduce the processor performance [49, 50]. Most SIMD processors limit therefore their vector widths to 8 or 16 words.

Coarse-grain reconfigurable platforms can increase their issues by increasing their number of ALUs. ALU scalability is however limited since delays in inter-ALU links rise significantly when many ALUs are used.

FPGAs have a good scalability since they provide uniform reconfigurable logic. More logic can easily be added by using bigger devices, and throughput can be increased by instantiating more functional blocks to process data in parallel.

Figure 2.5 shows the area efficiencies of the SDA platforms, along with the $DSP1$ and $VLIW3$ platforms. Data memory areas have been subtracted from the platform area values since data memory sizes depend on the application. Results show that most platforms provide a similar area efficiency. Note that those differences would be even more attenuated if the area of an L2 memory were added to the evaluations since those memories represent a significant part of a platform area. The results show that $DSP1$ and $VLIW3$ platforms have area efficiencies that are competitive with high-throughput platforms. This can be explained by the high utilization of their issues. Indeed, as shown in Chapter 6, the IPC of those platforms are respectively 0.96 and 2.51.

The evaluations are summarized in the second row of Table 2.6. Values estimated using $T_{p,typ}$ show that SIMD platforms provide better results since they require less area for instructions. However, those values suppose that their vector

issues are fully used, which is not the case for many algorithms. The significantly lower value of the SODA platform obtained for the 802.11a evaluation confirms this. Tile64 provides a lower area efficiency. This platform targets more general-purpose workloads. Consequently, it uses a complex communication network to maintain cache coherence and a high throughput between the cores that causes an overhead in area.

2.9 DYNAMIC FUNCTION CHANGE

SDAs aim to virtualize signal processing applications. This allows to change dynamically the definition of the application to modify the functionality of the hardware and therefore to build multi-standard applications without interrupting platform services. Consequently, hardware platforms for SDA must be able to change dynamically their functions. Moreover, to support different standards, it must support a large amount of functions.

SDA platforms meet those requirements by executing software. By changing the program or by just calling a function, the system functionality can also be changed. On some platforms, changing the program is as simple as jumping to another section of the code. The platform memory system then automatically fetches the required instructions and places it in the instruction caches. Other platforms require to explicitly perform code segment placement in instruction memories. This can be tricky since the programmer has to handle it manually. Depending on how program changes are managed, a platform will be able to change its functionality more or less easily.

Table 2.7 presents a qualitative evaluation of the ability of each platform to dynamically change its functionality:

	Multi-core	**SIMD**	**Coarse-grain**	**FPGA + DPR**	**Many-core**
Dyn. change	++++	++(++)	++	++	++(++)

Table 2.7. Qualitative evaluation of dynamic function change capabilities of the different SDA platform models.

Multi-core platforms perform well since the memory systems of those platforms can automatically fetch the required instructions from the main memory and place it in the core instruction cache. However, those memory systems can become complex and power-hungry when the number of cores increases.

In HMCP, to provide a good scalability, most platforms do not implement sophisticated memory systems. In those platforms, instructions must be explicitly placed in each core instruction memory by using DMA transfers or sending configuration packets. Homogeneous many-core platforms have therefore a less

efficient support for dynamic function changes. Tile64 is an exception however: it has a complex memory system since it targets more general-purpose workloads.

Modifying the program of coarse-grain reconfigurable platforms requires to change both instructions and configurations. As there is no global program counter for configuration vectors, those must be explicitly loaded in the configuration memories.

Regarding SIMD processors, some of them can support automatic code fetch through cache mechanisms, this is for instance the case of EVP16. However, in order to further reduce power consumption, other platforms like AnySP require to load explicitly the code segment in each processing element memory.

FPGAs are particular platforms. In order to execute software, they must be coupled to a processor. The processor can then handle the application, and use functional blocks instantiated in the FPGA logic to execute signal processing kernels with high computational throughput. In standard FPGA designs, the device is configured at boot-time and cannot be changed as long as it is in service. In that case, only the instantiated functions are available during runtime. However, dynamic partial reconfiguration allows to instantiate dynamically new functionalities in the FPGA. The configuration data of the functional blocks are stored in external memories. Since those memories are large, many configurations can be stored. This allows FPGAs to support dynamically changing applications like SDAs. However, reconfigurations must be explicitly handled by the programmer. Those aspects are discussed in more details in the next chapter.

2.10 DEVELOPMENT EFFORTS

Low-volume applications cannot afford high NRE costs. Development efforts must therefore be reduced to limit the application development costs. Those efforts depend on three aspects:

1. Toolchain: the toolchain provides tools and flows for application development. They allow the automation of a part of the development tasks. Depending on its maturity, the flow can ease the development of a solution by defining a precise sequence of steps that allows to build a working application. Tools provide automation of those steps and hide unnecessary complexity to reduce development efforts and avoid human errors. They also provide verification means that allows efficient debugging.

2. Developer qualification: depending on the programming model and the maturity of the tools, the qualifications required from the developer are different. For instance, using platforms with more experimental execution models, like reconfigurable platforms, will require higher qualifications. However, platforms with mature flows like FPGAs require only usual developer qualifications.

3. Development time: this is the time needed to port an application to a platform.

Table 2.8 presents a summary of the qualitative evaluation of those three aspects for the SDA platform models presented in section 1.8:

	Multi-core	**SIMD**	**Coarse-grain**	**FPGA + DPR**	**Many-core**
Ease of development	++++	+	+	++	+++
Toolchain maturity	++++	+	+	+++(+)	+++
Development speed	++++	+	+	++	+++
Summary	++++	+	+	+++	+++

Table 2.8. Qualitative evaluation of development efforts for the different SDA platform models. Ease of development depends on developer qualification: complex developments require higher qualifications. Toolchain maturity qualifies the toolchain completeness and efficiency.

Multi-core platforms have very good toolchains. Moreover, those platforms use a standard execution model which does not require high qualifications to develop. One can therefore quickly develop an application with those platforms.

Developing on many-core platforms introduces additional complexities compared to single or multi-core approaches. On many-core platforms, the developer must manage explicitly the successive core configurations and the mapping of the threads on the numerous available cores. This makes application development on those platforms harder than for multi-core platforms.

SIMD processors require to vectorize algorithms. Automatic compilation of algorithms explicitly expressed as vectors provides good results. However, autovectorization of C code is still a challenge. Indeed, complex code transformations and data access management are required to enable automatic vectorization [51, 52]. Current autovectorizing compilers still provide poor results [53, 39]. Therefore, it leaves this task to the programmer. High qualifications are needed since efficient vectorization of DSP kernels is a complex task. Moreover, there is no guarantee on the performance.

Multi-core, many-core and other platforms like CGRAs can also execute SIMD operations in their ALUs. However, this SIMD parallelism is different from the parallelism exploited by SIMD processors. Indeed, in those platforms, the SIMD ALUs exploit very short vectors, with 2 or 4 words. This is for instance the case of the TILE64, Am2045, SB3011 and ADRES platforms, and the processor cores developed in this thesis. This kind of parallelism often corresponds to the natural data encoding in memory, like for instance complex numbers encoded as 2x16 bits or a pixel line encoded as 4x8 bits. Kernels algorithms require no or few transformations to be expressed in these representations. The required efforts to vectorize them is therefore very limited compared to SIMD processors.

Coarse-grain reconfigurable platforms have only experimental toolchains and limited support for automatic compilation. CGRA mixes spatial and sequential execution. Automatic compilation tools must identify and map the various computation graphs on the configurable ALUs and routing resources, and schedule the successive configurations. Automating those tasks is complex and current solutions rely on generic problem solving algorithms like simulated annealing [54, 55]. Those generic solutions are used on small kernels and compilations take a long time to perform [54]. Generic solvers also provide limited capabilities to model precisely architectural specificities and therefore exploit them in the solutions. This limits the quality of the results [56]. Consequently, to obtain high performance, the developer must perform by hand the mapping and scheduling.

FPGAs require hardware development, which is more complex than software. However, FPGA toolchains are mature and many tools are available to automate development and verification. Moreover, many tools offer the possibility to automatically synthesize efficient FPGA designs from high level application descriptions or even C code. Those tools and flows considerably ease development on FPGA. Nevertheless, the use of DPR further complicates the development of a design. This particular issue is addressed in the next chapter of this thesis.

2.11 GENERICITY

To reduce the NRE costs related to the production of the hardware platform, low-volume applications can use generic platforms. Those platforms must be used for many different applications or even application types (e.g. video, telecommunication) and therefore used in higher volumes, which allows to further amortize their development costs. To be generic, a platform must support many different functionalities and, furthermore, it needs to support them efficiently.

In order to support efficiently an application, a processor must have specific instructions for this application. For instance, in order to support video compression, a processor must support efficiently instructions such as the sum of absolute difference on pixels. Processors with an instruction set providing specific instructions for several applications are therefore more generic. However, in order to support many instructions, it is necessary to implement the related datapaths, which increases the ALU areas. In some platform models, having large ALUs is an issue since it introduces an area overhead. Those models must therefore use simpler ALUs, which limits their genericity.

Table 2.9 summarizes the genericity evaluations for each SDA platform model:

	Multi-core	**SIMD**	**Coarse-grain**	**FPGA + DPR**	**Many-core**
Genericity	+++	++	++	++++	++(+)

Table 2.9. Qualitative evaluations of genericity for the SDA platform models.

	Issue count	ALU area [μm^2]	ALU area per issue [μm^2]	$\frac{ALU\ area}{ALU\ area_{DSP1}}$
$DSP1$	1	13251	13251	100%
$VLIW3$	3	22229	7409	56%
$VLIW5$	5	41109	8221	62%
$SIMD8$	8	106008	13251	100%
$CGRA16$	16	212016	13251	100%

Table 2.10. Comparison of total ALU area and area per issue for five core architectures

Multi-core platforms use big cores with a high number of issues. In those platforms, the register file represents a large part of the platform area, and can represent up to two thirds of the total area in some configurations [57]. Their ALUs can therefore be relatively large without incurring a significant overhead. Issues can also have different ALUs, each supporting different specific instructions. For instance, in the C64x DSP, each cluster has 4 different ALUs. This approach allows the core to support an extended instruction set by distributing the support for specific instructions in the numerous available ALUs.

Homogeneous many-core platforms use simpler cores than in multi-core platforms. Those cores are typically RISC cores with a low issue count (e.g. 1-3 issues). In those cores, using complex ALUs causes therefore a significant area overhead. Moreover, the limited issue count does not allow to use many different ALUs. Those platforms have therefore a lower genericity than multi-core platforms.

In SIMD processors, the datapaths required to execute each supported instruction must be present in every execution issue. Moreover, ALUs are replicated many times in order to execute instructions on wide data vectors of more than 8 words. Consequently, supporting many instructions requires to replicate a large ALU many times. This introduces a trade-off between the processor genericity and the total area of its ALUs. This is not the case in the cores typically used in multi-core and manycore platforms, e.g. VLIW cores, where only a single datapath must be added to provide a specific functionality to the whole core. In the same way, coarse-grain reconfigurable platforms are composed of several reconfigurable ALUs. In order to obtain a high level of parallelism, those ALUs are replicated many times. For instance, in the ADRES platforms, there are 16 identical ALUs. The datapaths required to execute the supported instructions must also be replicated in every ALU. This introduces the same trade-off between genericity and the total ALU area in the chip as in SIMD processors.

Table 2.10 provides an illustration of this trade-off. The table shows the total ALU areas and ALU areas per issue for five processor configurations built using the same ALU datapaths. Those ALUs correspond to the ALUs used in the three processors developed in this work, their composition is detailed in Chapter 6. They all work on 32-bit words, which is the data width that has been consid-

ered for the three implemented processors. The processor ALUs are dimensioned to provide optimal IPCs on the 802.11a application. In Table 2.10, $SIMD8$ corresponds to an 8-way SIMD processor, and $CGRA16$ corresponds to a coarse-grain reconfigurable processor with 16x32-bit ALUs. In order to provide the same level of functionality as the $DSP1$, $VLIW3$ and $VLIW5$ processors, an instance of each type of ALU datapath must be present in every issue of the $SIMD8$ and $CGRA16$ processors. Therefore, the $SIMD8$ and the $CGRA16$ have the same ALU area per issue as the $DSP1$, which also requires one instance of each ALU datapath due to its single issue. However, in comparison with the $VLIW3$ and the $VLIW5$, the $SIMD8$ and $CGRA16$ processors have bigger ALU areas per issue. Indeed, the ALU areas per issue of the $VLIW3$ and $VLIW5$ represents respectively 56% and 62% of the ALU areas per issue of those two processors.

Each application type necessitates specific shuffle patterns. For instance, in telecommunication applications, performing an FFT requires to support the butterfly pattern to be implemented efficiently. Implementing generic shuffle stages, like a crossbar, cause however a large overhead in area and power. SIMD processors have therefore shuffle stages with a limited genericity. SIMD processors have also a limited support for algorithms that do not support efficient vectorizations, like for instance errorcorrection codes which work at the bit level, e.g. Viterbi or turbo codes. In coarse-grain reconfigurable processors, generic interconnection topologies also incur a significant overhead in area and power [58]. Those overheads limit the supported instructions and functionalities, and therefore the genericity of those platforms.

FPGAs can be configured at bit-level, and provide a large amount of configurable resources. The fine level of configurability allows to create optimized implementations for each required functionality. This approach provides therefore efficient implementations for many functions. Consequently FPGAs have a very high level of genericity.

2.12 SUMMARY

Platforms for SDAs targeting low-volume applications with a high diversity of standards must meet several requirements. The platforms must be able to change their functions dynamically to support the virtualization that SDAs necessitate. They must also support very high computational throughputs and be highly power-efficient. As production volumes are low, they must limit the non-recurring engineering costs related to the application and platform development costs. In order to do so, platforms must reduce the development efforts to implement an application and be highly generic.

The platform models presented in the previous chapter have been characterized for the five requirements of low-volume SDAs: dynamic function change, development efforts, genericity, power efficiency and throughput. Three of those criterions do not depend on a specific application field or benchmark: development efforts, dynamic function change and genericity. Their evaluations have

therefore been based directly on the platform model properties. However, in order to characterize the power efficiencies and throughputs of the platform models, performance estimations must be realized on platform implementations. In this work, those estimations are based on published results due to the low availability of the studied platforms. Two benchmarks have been used to perform those estimations: the first benchmark evaluates the platform typical performance and the second evaluates the platform performance on a telecommunication SDA. All the platform implementations evaluated in this work target this SDA. The results obtained with the benchmarks provide therefore representative results of the performance attainable by those platforms. Their analysis allows to characterize the performance of each model.

In the literature, cross-platform model comparisons are rare, or are limited to simple surveys of existing platforms [59, 60]. In this work, platform model evaluations are based on model properties which do not depend on any specific application type or benchmark. The evaluation minimizes therefore the typical interferences occurring in those comparisons, especially interferences due to technology. Using these evaluations, a platform model comparison grid can be built that allows to clearly identify the advantages and disadvantages of each platform model compared to each other. Using the grid, one can choose appropriately the platform model to use in order to implement an SDA for low-volume applications. Table 2.11 illustrates the comparison grid which summarizes the evaluations performed in this chapter:

	Multi-core	**SIMD**	**Coarse-grain**	**FPGA + DPR**	**Many-core**
Dyn. change	++++	++(++)	++	++	++(++)
Development efforts	++++	+	+	+++	+++
Genericity	+++	++	++	++++	++(+)
Power efficiency	+	++(+)	++(+)	+++	++
Throughput	++	++(+)	+++	+++	+++

Table 2.11. Summary of evaluation of SDA platform models for the five requirements of SDA for low-volume applications with a high diversity of standards.

The results show that multi-core, SIMD and coarse-grain platforms show important issues for low-volume SDAs:

- Multi-core platforms have a very limited power efficiency due to high power consumption of their cores caused by high frequencies and complex memory systems with cache coherence. Moreover, they provide only a limited throughput since they can use only a limited number of cores, i.e. 2-8 cores, due to the limited scalability of the memory system.
- SIMD processors require significant higher development efforts than for multi-core, many-core and FPGA platforms since algorithms must be vec-

torized by hand in order to provide a high power efficiency and throughput. Vectorization is a complex task which demands high qualifications and results are not guaranteed. Automatic vectorization of C code is still a challenge. Those processor models also lack genericity since not all kernels can be vectorized, and some elements of their datapaths cannot be made highly generic without incurring a degradation in efficiency.

- Coarse-grain platforms demand development efforts similar to SIMD processor efforts. Those platforms have a more complex execution model, which mixes spatial and sequential execution of instructions, and they only have limited support for automatic compilation. Indeed, compilation for those platforms require to identify and map the computation graphs and schedule the reconfigurations. Researches are still ongoing to find solutions to automate these tasks. Currently, those complex tasks must therefore be realized partially or totally by hand. Moreover, the architecture of coarse-grain platform interconnections and their ALUs cannot be made highly generic without reducing their efficiency.

FPGAs with DPR and homogeneous many-core platforms are however interesting candidates for low-volume SDAs. Indeed, FPGAs have a good power efficiency, and are highly generic platforms. With DPR, they can provide the flexibility required by SDAs. However, DPR can complicate the development of an application.

Homogeneous many-core platforms are also very interesting platforms for SDAs. They provide a good genericity, high throughputs and require less development efforts compared to SIMD processors, FPGAs, and coarse-grain platforms. Their power efficiencies are good, but slightly lower than those last platforms. In order to implement SDAs on those platforms, their power efficiencies must therefore be studied. HMCP power efficiency strongly depends on its core.

CHAPTER 3

ANALYSIS OF DPR IMPACT ON DESIGN AND DEVELOPMENT

3.1 INTRODUCTION

As it has been shown in Chapter 2, FPGAs are very interesting platforms for low-volume software-defined signal processing applications. Standard FPGA designs are configured at boot-time and keep their configurations for the whole duration of the application execution. The available functionalities are therefore limited to the functions instantiated in the device. This approach strongly limits the capability of the FPGA to provide many different functionalities.

Several FPGAs support dynamic partial reconfiguration (DPR) of their resources. This technique allows to reconfigure the device logic without interrupting the component. DPR has been shown to offer interesting opportunities for FPGA designs [61, 62, 63, 64, 65, 66] and has potential applications in many domains, including software-defined radio [67], evolvable hardware [68], or even wearable electronics [69]. By using DPR, FPGAs have the possibility to dynamically instantiate functional blocks in the reconfigurable logic. The configuration data for those functional blocks are stored in external memories. Since those memories can be very large, many different functional blocks can be stored and made available to the application. Consequently, DPR gives to FPGAs the level of dynamic functionality change required by SDAs.

Applications with low volumes of production cannot amortize high NRE costs. It is therefore critical to reduce the development efforts when developing such

applications. In order to use DPR to build SDAs for low-volume applications, it is therefore important to evaluate the required development efforts. For this reason, this chapter analyzes of the impact of DPR on the design and development of FPGA designs. The benefits of DPR for SDA are also presented. Additional hardware needs and design rules of DPR are identified and discussed. Missing elements of the development flows are also identified.

In order to manage dynamic reconfigurations in the low-level layers of the design, it is proposed to handle reconfigurations locally by associating reconfigurable regions of the FPGA with small microcontrollers. The advantage of this solution is discussed and an evaluation is performed to measure the hardware overhead that it would occur on a design.

The chapter is organized as follows: the first section introduces dynamic partial reconfiguration. The main concepts of DPR and the reconfiguration process are explained. The evolution of DPR in FPGA components is also presented. Section 3.4 and 3.5 discuss DPR benefits and the advantages for SDAs. Section 3.6 and 3.7 present respectively the analysis of the impact of DPR on the design of an application and the flow. In the last section, the possibility of managing DPR locally by using small microcontrollers is discussed and the feasibility of this solution is evaluated by estimating the hardware overhead of this approach.

3.2 DYNAMIC PARTIAL RECONFIGURATION OF FPGA

This section introduces the main concepts related to dynamic partial reconfiguration of FPGAs. Most of the concepts presented here are related to Virtex FPGAs from Xilinx since those large devices are the most used components supporting DPR. In the next section, the elements operating during FPGA configuration are presented. The following sections then describe in more details two of those elements: the configuration bitstream, which holds the configuration data of the device, and the configuration interfaces, which provide control on the FPGA configuration. The complete partial reconfiguration process is then explained. In the last section, the evolution of DPR in the latest FPGAs is discussed.

3.2.1 Configuration chain

FPGA reconfigurable logic elements are configured by the data stored in a configuration memory. The content of this memory defines the content of the lookup tables and memory blocks, and configures routing resources in the device. Other components, like DSP blocks, high speed IOs controllers, or digital clock managers are also configured with the content of this memory. The configuration memory is modified through a configuration chain, which is illustrated in Figure 3.1.

The first element of this chain is the configuration bitstream. This bitstream is a binary file composed of configuration frames, which are the smallest amount

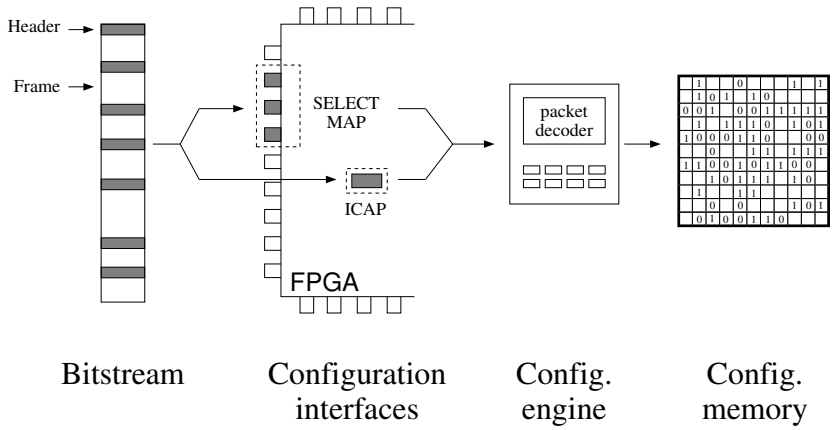

Fig. 3.1. Components of the FPGA configuration chain for a Virtex FPGA.

of configuration data that can be written in the FPGA configuration memory. During configuration of the FPGA, a bitstream is written to a configuration port using its specific protocol. These configuration ports are external or internal. The bitstream written in the configuration port is then interpreted by a reconfiguration engine which has access to the configuration memory and rewrites it. There is therefore no direct access to the configuration memory from the user logic.

3.2.2 Bitstream composition

The bitstream is the configuration file for the FPGA. It consists in the data to be written to one of the configuration interfaces of the device. A bitstream is a sequence of packets, each packet containing a header and data. The headers specify where the data will be written and the number of datawords present in the packet. Data can be used to program the configuration engine or can be configuration data that will be written into the configuration memory of the FPGA. Bitstream structure is simple to understand, and is documented [70, 71].

Configuration data are composed of frames. A frame is the smallest configurable part of the FPGA. In V4 and V5 FPGAs, frames are composed of 41×32 bits and represent a tile of the logic contained in the device that has the height of a row of the FPGA, and the width of a fraction of a FPGA column. Each frame in the device has a unique address. This frame address is composed of several concatenated fields: minor address, column, row, location, and type. Knowing the address of a frame, it is possible to find its position in the FPGA, and vice-versa. Configuring any region of a FPGA is actually equivalent to write the configuration data for all the frames that compose this region.

A typical packet writing a frame is presented in fig 3.2. The header is composed of an address writing instruction, the address of the frame, then a frame

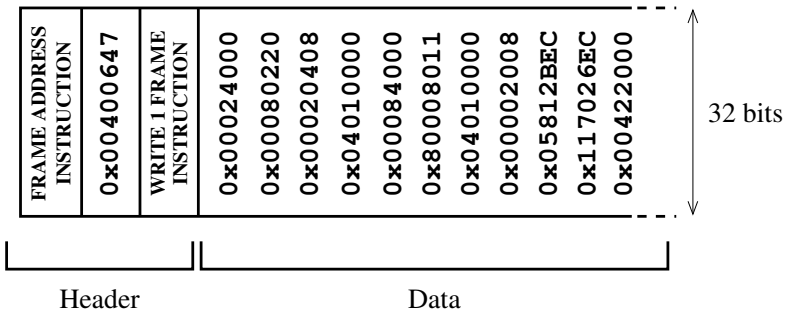

Fig. 3.2. Bitstream packet writing one frame in configuration memory

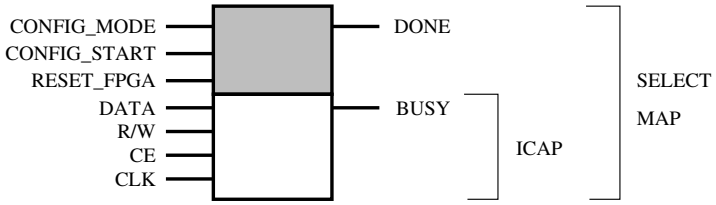

Fig. 3.3. Select Map and ICAP ports

writing instruction. The data of the packet are the frame bits. Such packets can be followed by packets telling the configuration engine to copy this frame at other places of the configuration memory. This technique, called "Multiple Frame Writing" is used to reduce the size of the bitstream. Packets can contain several frames. In this case, the configuration engine automatically increments the frame address and writes the next frame to the next address.

Typically, a bitstream that configures completely a FPGA is composed of three parts: the first part is composed of several packets that initiate the device, the second part is one long packet containing all the frames, and finally the last part is composed of several packets that close the configuration process and start the FPGA.

3.2.3 FPGA configuration interfaces

Access to configuration memory can only be performed using the configuration interfaces of the device. In the latest Virtex FPGAs, there are multiple interfaces for configuration: several external interfaces (Select Map, JTAG and serial), and one internal interface. This internal port is called the ICAP (Internal Configuration Access Port). Figure 3.3 shows the pins of the Select Map and ICAP ports. The figure shows that the ICAP is actually a simplified version of the Select Map port. Both ports are basically composed of a data port, a clock signal, and

status/control signals. The data width of these ports is configurable and support datawidths of 8 or 32 bits.

To perform FPGA configuration, a bitstream is written in the reconfiguration port. The configuration ports also allow configuration readback, which enables to read to the internal configuration memory. The ICAP is accessible from the user-logic plane of the FPGA, which enables self-reconfiguration, i.e. configuration of the device from its own logic. There are two ICAPs in the Virtex 4 and Virtex 5 FPGAs.

3.2.4 Reconfiguration process

Partial reconfiguration is realized by using a configuration bitstream which includes write commands and configuration data for only a subset of the device logic. In DPR terminology, those bitstreams are called partial bitstreams. The partial reconfiguration process is illustrated in Figure 3.4. To perform partial reconfiguration, a partial bitstream is written to a configuration port. The configuration engine then interprets the bitstream and rewrites the configuration data of the specified resources only. If the reconfiguration is successful, the reconfigured logic is activated at the end the bitstream write. During reconfiguration, reconfigured logic resources are halted, but the unmodified logic continues to work without interruption.

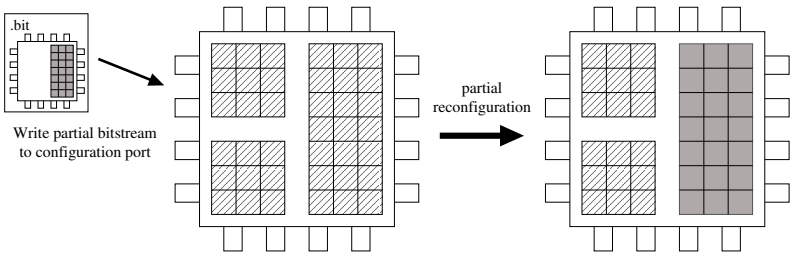

Fig. 3.4. Illustration of the FPGA partial reconfiguration process. A partial bitstream is written in a configuration port, and modifies the configuration of a part of the user logic.

In latest Virtex FPGAs, the ICAP port can work at a maximum theoretical speed of 3.2Gb/s when configured for a 32-bit wide data input and clocked at 100MHz. The bitstreams containing the configuration of an entire FPGA can be large, some of them can go up to several tens of megabits in the latest generation of FPGAs. Typical partial bitstreams represent therefore about several hundreds of kilobytes, which means that the reconfiguration process can take up to several milliseconds to complete for large partial reconfigurations.

3.2.5 Evolution of DPR in the latest components

At present, the main supplier of FPGAs capable of DPR with a large amount of reconfigurable logic is Xilinx with its Virtex serie (in versions Virtex II and above) [72]. Other FPGA models are known to support dynamic reconfiguration as well. For instance, the Atmel AT40K can also perform DPR [73], but has far less resources. Altera has announced in its roadmap that future components will also support DPR [25].

DPR was introduced in the Virtex family with the Virtex II component. Its layout was organized in columns, defining an entire column of the component as the smallest configuration granularity. Therefore, the DPR had to cope with severe hardware constraints that lessened its use. Since the Virtex 4 component, several improvements facilitate the use of DPR, making it a viable solution for some specific applications like SDAs. The improvements for DPR have been made on the layout architecture, the clock tree and the ICAP. Virtex 4 components and the next generations are still organized in columns but the partial reconfiguration granularity is reduced here to only a part of a column. Therefore, the reconfigured regions of the FPGA can be almost any height and width.

In those devices, the width of the ICAP port has also been extended to 32 bits and its speed has been increased up to 100 MHz. This significantly speeds up the reconfiguration time, which is a main concern when using DPR since the hardware in the reconfigured regions cannot be used during the reconfiguration process.

3.3 RELATED WORK

Early works on dynamic partial reconfiguration of FPGAs have been realized on the XC6200 component from Xilinx. This device is the first commercial FPGA specifically designed for dynamic reconfigurable computing. In [74], McKay and Singh evaluate the benefits of specializing a circuit instantiated in a XC6200 FPGA using dynamic reconfiguration. In [75], power reduction opportunities enabled by reconfigurations are evaluated for a motion estimation kernel. In [76], issues related to virtual hardware management are studied.

Subsequent works study DPR usage on more recent components like Virtex FPGAs from Xilinx. In [77], Blodget et al. propose a software and hardware architecture allowing to dynamically reconfigure an FPGA under the control of an embedded processor. In [78], Horta et al. present a platform that uses dynamic hardware plugin modules. Using those hardware plugins, the platform can modify dynamically the implemented functions in a network router. This work also proposes specific interfaces to communicate with the reconfigurable modules and a tool to produce and manage configuration data. In [79] and [80], Ulmman et al. present a complete run-time reconfigurable system that enables on-demand reconfigurations in an automotive application. These contributions

propose an architecture supporting dynamically reconfigurable modules and a task management system that can prioritize reconfiguration requests.

Other works study more specific issues related to DPR exploitation. They propose design flows and methodologies enabling the integration of DPR in application development [81, 82, 83], operating systems allowing to handle task executions using reconfigurable hardware resources [84, 85, 86], solutions for configuration scheduling [87, 82], techniques for module physical placement in the component logic [88, 89], and tools for dynamic configuration data adaptations enabling module relocation in the device at runtime [90, 91, 92].

Other contributions study the benefits provided by DPR at the application level. They analyze opportunities for updates after deployment [93] or SEU mitigation using DPR [94]. Among those works, several of them study DPR interests for specific applications, like evolving hardware where DPR provides the possibility to modify and evolve device configuration [68], wearable computing nodes where DPR provides increased power efficiency and adaptability [69], or software-defined radio where DPR enables dynamic modification of the waveform modulation [95, 67]. This thesis has the particularity to focus on low-volume applications and their specificities, like strict validation. It studies the issues related the industrial usage of DPR. More specifically, this chapter identifies issues related to design and development of applications using DPR.

3.4 DPR BENEFITS

DPR allows to reconfigure configurable resources for a new use without interrupting the device. Thanks to this capability, all the functional blocks are not required at all times in the device, only a smaller set of those must be present. Consequently, this functionality can be exploited to reduce the size of the device. Using a smaller device has two major advantages: it allows to reduce the cost of the device and the static power consumption.

Figure 3.5 illustrates how DPR allows to reduce the device size. The upper graph shows the tasks performed by an application and their concurrency. In a telecommunication application, tasks A and B can be pre-filtering operations for instance, and tasks C & D can demodulations. In the presented scenario, only two tasks are performed simultaneously at all times. On the lower part of the figure, two FPGA implementations of this application are presented. On the left side, a conventional static FPGA design is showed. In order to provide the required functionalities, all the needed functional blocks must be instantiated in parallel in the device. This design requires therefore sufficient area to place them all in the FPGA. On the right, an FPGA design capable of performing DPR is showed. In this design, only two reconfigurable regions are needed to provide the same functionality as the static design. In the figure, those reconfigurable regions are named R1 and R2. The area needed for the two additional functional blocks can therefore be reclaimed, which allows to save area and use a smaller

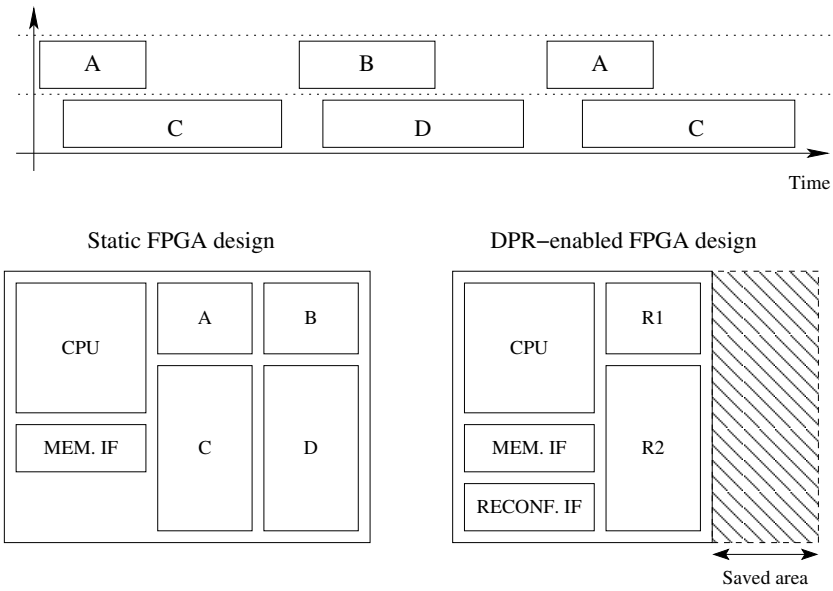

Fig. 3.5. Comparison between a conventional static FPGA design and a DPR-enabled design for the same usage scenario. The application only requires two functional blocks at the same time. The DPR-enabled design can reallocate its resources dynamically. The component size can therefore be reduced since only two functional blocks are strictly required at all times.

FPGA. However, an additional hardware block is required to interact with the reconfiguration port.

DPR can also be used to reduce the dynamic power consumption of a design. Indeed, in a conventional design, some functional blocks must cover many different parameter values or data sets. To face all possible combinations, a generic solution is used. However, many of those generic implementations are less efficient than specific ones. With DPR, specific functional blocks with better power efficiencies can be instantiated dynamically in the design depending on the actual application need. In the same way, the computational load that a design can handle can also be adapted by dynamically loading more specific blocks providing higher throughputs, or by instantiating additional functional blocks in the component to increase parallelism.

Additionally to those advantages in component size, power and throughput, DPR also brings new functionalities at system-level compared to conventional static FPGA designs. In [96], Manet et al. have identified many of those by evaluating the use of DPR in real industrial and military applications. Those advantages are the following:

- Survivability: by allowing reconfiguration of the system in a degraded but safe mode when a part of it is damaged. It is necessary for applications running in harsh environments where environmental conditions can exceed the normal operating range.
- Mission change: DPR enables reconfiguration of an application for an entire mission without interrupting services. This is crucial for systems where the real-time issue is not critical, but the interruption issue is. DPR provides an easy and safe way to strongly modify an entire system.
- Environment change: during operation, the application can be developed specifically for several environments and switch dynamically. For instance, this feature can be used to switch an image processing system from day to night vision.
- Adaptive algorithm change: DPR allows to adapt dynamically an algorithm depending on the external conditions. Compared to environmental changes, this aspect has a lower granularity, since it is performed at algorithm level.
- On-line system test: a system in harsh environment can be damaged. DPR can be used here to temporarily instantiate a testing module to evaluate system functionality level.
- Hardware virtualization: DPR allows to have more hardware available than physically present in the FPGA. It allows to manage a set of hardware modules as a component library. This is a key advantage for SDA.

3.5 ADVANTAGES FOR SDA AND LOW-VOLUME APPLICATIONS

In the previous chapter, the comparison of hardware platform models for SDA has shown that fine-grain reconfigurable platforms like FPGAs are very interesting candidates for SDA. Indeed, those platforms are highly generic, support high computational throughputs and have good power efficiencies. Conventional static FPGA designs cannot change their functionalities during use, which prevents the implementation of dynamically defined applications such as SDAs. However, using DPR, the functionalities instantiated in a design can be changed dynamically. The dynamic partial reconfiguration of FPGA enables therefore SDA implementations on FPGAs, and allows to leverage the advantages of those platforms.

As presented in the previous section, DPR provides many advantages and enables new system-level functionalities. By using DPR, SDAs benefit of all those advantages. However, a particularly interesting functionality for SDAs is the hardware virtualization capability. Indeed, with DPR, definitions of functional blocks can be stored as partial bitstreams in external memories. Those definitions can be loaded dynamically in the FPGA without interrupting the application by performing DPR. Since the external memories storing the bitstreams can be very

large, many functional blocks can be stored. For instance, 1GB of external flash memory can store more than 2000 bitstreams of 500KB, which represents the configuration of a Microblaze soft-core processor. Large libraries of functional blocks can therefore be created. New functionalities can also be implemented in the platform after deployment by adding more partial bitstreams in the external memory, even during runtime. Since SDAs must support a very high number of different functionalities, this capability is a very important benefit. Moreover, the possibility to upgrade a working design with new functionalities is also a very compelling advantage for low-volume applications such as professional electronics, where designs are often kept in service for several decades.

Hardware virtualization also allows to use hardware blocks with the flexibility of software. Indeed, if a block is not available in the device, a reconfiguration can be performed to instantiate it dynamically. This reconfiguration can be done automatically, for instance after a function call in a processor acting as the application controller. Since hardware implementations provide high throughputs and power efficiencies, this approach also allows to improve the design performance. This aspect of hardware virtualization is therefore another major benefit for SDAs since it meets their requirements of dynamic function change capability, high throughputs and power efficiency.

Thanks to these advantages, FPGAs capable of DPR have become very interesting platforms to develop SDAs. Several FPGA platforms capable of DPR have been developed to build SDA, mainly for the software-defined radio. It is the case for instance of a platform from ISR technologies [97], which uses a Virtex-II FPGA [98]. Another platform, from Thales, is also leveraging DPR. This solution has been shown to be able to reach the requirements of software-defined radio, and to benefit from the system-level advantages of DPR [99]. Software-defined radio is also foreseen by Xilinx as the main target application for DPR [100].

3.6 DPR IMPACTS ON FPGA DESIGN

In order to use DPR, usual FPGA design techniques must be adapted. DPR introduces new design rules, new interfaces and protocols, and new constraints. This makes FPGA design for DPR more complex than traditional FPGA designs. In order to use DPR, a design must comply to five main requirements:

1. The FPGA logic must be partitioned in reconfigurable and non-reconfigurable regions.
2. Specific communication structures must be used to communicate between a reconfigurable region and other regions.
3. The memory system must be carefully designed in order to respect real-time constraints.
4. A configuration port controller is required to control the configuration interface.

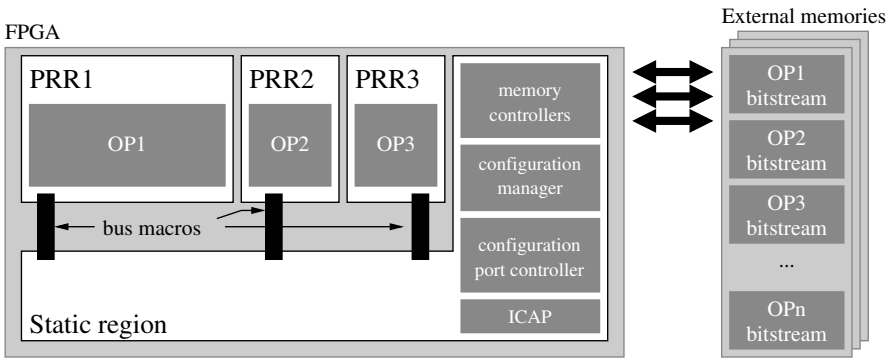

Fig. 3.6. A DPR-enabled design with a static region, three partially reconfigurable regions, and additional hardware for access to partial bitstreams in memories, management of reconfigurations and control of the internal configuration port.

5. A configuration manager is needed to manage reconfigurations.

Figure 3.6 illustrates a complete DPR-enabled FPGA design, containing all the required functional blocks and the specific structures that such a design requires. The next sections explain each of those requirements and their impacts on the design in details.

3.6.1 Design partition

A DPR-enabled design must be partitioned in reconfigurable regions and non-reconfigurable regions or static regions. A design with multiple reconfigurable regions must provide a separate partition for each region that can be reconfigured separately. Such a partition is illustrated in Figure 3.6. In this figure, the resources are partitioned into a static region and three reconfigurable regions. As illustrated, those regions can have different sizes and shapes.

Each partition must be assigned to clearly defined and non-overlapping physical sections of the device. This physical separation prevents reconfiguring other regions of the design during the reconfiguration of a specific reconfigurable region. Physical locations and boundaries must also follow device-specific physical placement rules. They can be restricted for instance by the component logic resource organization or its clock distribution structure. To realize this specific physical placement, constraints must be provided to place and route tools to guarantee that resources from a region will be placed strictly inside their physical locations.

3.6.2 Bus macros

At present, place & route tools for FPGA provide no possibility to impose a specific routing. This is an issue for DPR since multiple functional block con-

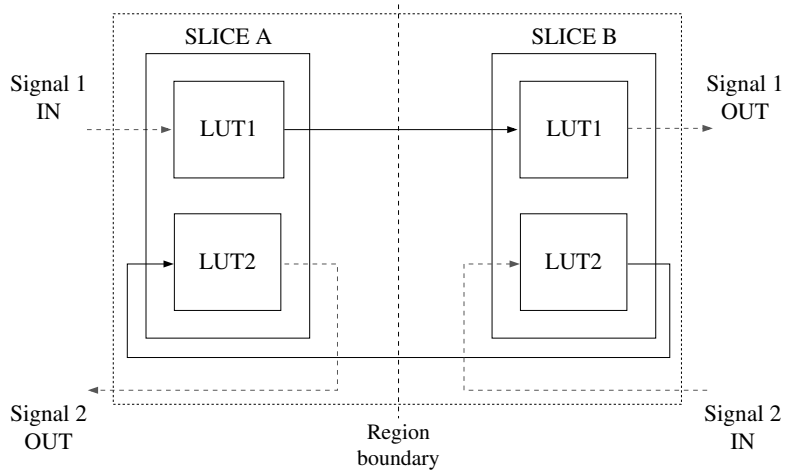

Fig. 3.7. A bus macro made of made of 2 slices and 2 interslice communication links implementing a bidirectional 1-bit connection.

figurations may not use the same routes to communicate with other regions and therefore render the design unusable after reconfiguration. To prevent this, specific communication structures called bus macros must be used to implement communication from and to a reconfigurable region. Those bus macros are pre-routed structures implementing a communication link using the FPGA reconfigurable logic. Bus macros must be instantiated in the design for each communication link between a reconfigurable region and another region. In Figure 3.6, the bus macros are represented by the black rectangle in between the reconfigurable regions and the static region.

Since those structures are pre-routed, using the same bus macro for each configuration of a reconfigurable region assures that routing will stay the same after reconfiguration. The simplest bus macro is composed of two pairs of interconnected LUTs. This bus macro configuration is illustrated in Figure 3.7. This bus macro implements a 1-bit bidirectional communication link. Bus macros depend on the communication link width, and on their positions in the device. A new bus macro must be generated for each different configuration of a link.

3.6.3 Bitstream storage

Typical partial bitstreams represent several hundreds of kilobytes. For instance, the partial bitstream of a Microblaze soft-core CPU represents about 500KB. For applications like SDAs, many partial bitstreams are needed and large memories are therefore required to store them. As FPGA internal memories are volatile and limited in size, external memories must be used for bitstream storage. However, those larger memories are generally slower and have higher latencies.

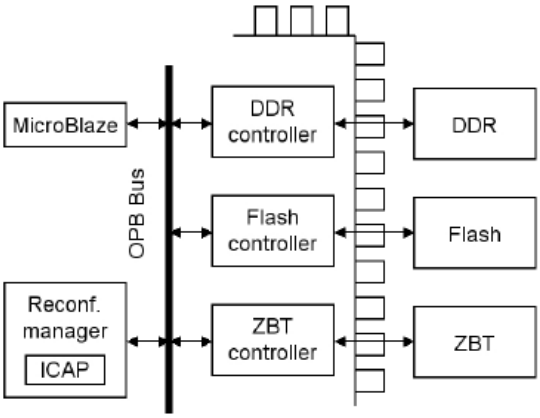

Fig. 3.8. Memory system for fast bitstream transfer using a DDR memory, a ZBT memory, and a dedicated bus to the controller.

For designs under strict real-time constraints, memory access speed and latencies are very important characteristics. To meet real-time requirements, one must use a dedicated memory bus with high-speed memories like zero bus turnaround (ZBT) memories which allow to reach high data transfer rates. However, since those memories are relatively small, they can only store few bitstreams. Indeed, typical ZBT RAM memories contain not much more than 1MB, which represents only 2 or 3 partial bitstreams.

In DPR-enabled designs, a memory system must therefore be carefully designed for partial bitstreams. For real-time applications, it must be implemented so that the bitstream transfer speed and latencies allow to meet the constraints. An example of a memory system is illustrated in Figure 3.8. In this example, several types of memories are combined to provide high transfer rates from the memories to the reconfiguration manager. Nonvolatile memories are used to store bitstreams in a persistent way, and other memories are used to provide higher throughputs.

3.6.4 Additional hardware required for reconfiguration

To manage the configuration interface and the reconfigurations, DPR-enabled designs require a configuration port controller and a configuration manager. The configuration port controller receives the bitstream and writes it to the configuration port using the required protocol. To speed up reconfigurations, it must allow bitstream transfers from a memory to the configuration port at full speed, by supporting DMA transfers for instance. Some configuration controllers are also able to perform configuration readback, which can be useful for verification. Since the reconfiguration hardware is vendor-specific, there are no standard re-

configuration port controller and no standard protocol. Consequently, a specific reconfiguration controller must be developed for each version of the port. This is an important issue since different versions of a same module must be maintained. The different versions also have different performance. For instance, the Virtex-II ICAP is only capable of 8-bit transfer at 66MHz whereas the latest ICAP is capable of 32-bit transfers at 100MHz.

A dynamic hardware manager is needed to manage reconfigurations. When a reconfiguration is needed, it has to initiate the bitstream transfers to the configuration port controller. A reconfiguration can take several milliseconds and is therefore a relatively slow process. In case of real-time constraints, the reconfigurations must be properly scheduled in order to hide the reconfiguration latencies and meet the constraints. The manager has also to manage the functional block allocations to the available reconfigurable regions without causing too much fragmentation in the device logic. As the manager can be complex, it can be implemented in software.

3.7 IMPACTS ON THE DEVELOPMENT FLOW

FPGA development flow is composed of several steps progressing in a top-down approach, from the specifications to the actual implementation. For each step, mature tools are available that provide efficient solutions. Regarding DPR, the existing development flows from the vendors [101, 102] do not provide a support that is as complete as for static FPGA designs. Indeed, those flows miss several important elements.

In particular, no behavioral models are available to simulate designs using DPR. Models of the configuration interfaces (e.g. ICAP), bus macros and the configuration process are however required in order to perform functional simulations, validate the respect of real-time constraints and debug the design. Without those models, the modules can only be tested and simulated separately. The design can then only be verified when all the modules are integrated in the platform. If a problem occurs, there is no way to reproduce the issue by simulation. Modeling tools such as SystemC [103, 104], can be used to build a custom high-level model. Some works have studied the possibility of using it [105, 106, 107, 108]. Other than models, some debugging tools are also missing. For instance, on-chip logic analyzer like ChipScope from Xilinx do not work in reconfigurable regions. This reduces considerably the possibility to observe and isolate errors in designs using DPR.

These missing elements are a major issue that can make development with DPR very difficult. The lack of tools for functional validation is particularly important for professional applications like safety-critical applications that require strict validation at each level of the work [109]. Those particular aspects are discussed in more details in the next chapter.

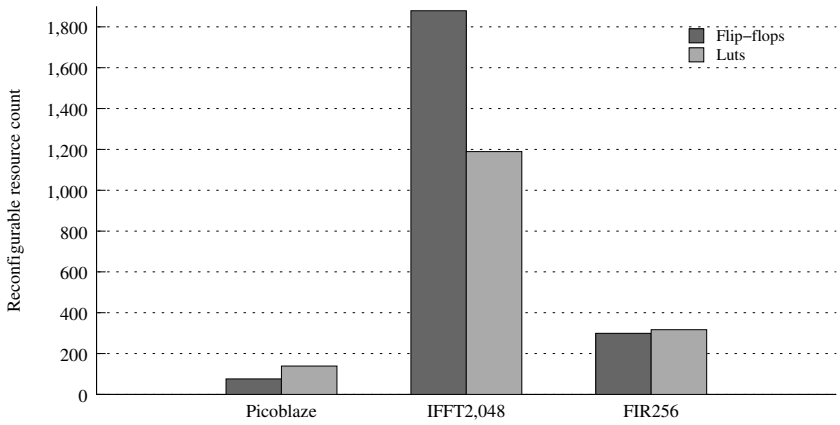

Fig. 3.9. Comparison of reconfigurable resources needed for the implementation of a Picoblaze microcontroller and hardware OFDM functional blocks in an FPGA.

There is at present no standards for DPR. DPR technologies are therefore largely vendor- and device-dependent. However, developing debugging, verification or accurate simulation tools require to know and model precisely the timing and behavior of the FPGA reconfiguration hardware, as well as the configuration memory. Currently, details about those hardware blocks are kept secret, and no models are provided by the vendors. Consequently, accurate tools enabling in-depth design debugging and verification are currently not available. Their development by third parties is difficult due to confidentiality issues.

3.8 HARDWARE VIRTUALIZATION MANAGEMENT

DPR provides true hardware virtualization: it allows to store a set of hardware functions in memory, as partial bitstreams. This mechanism allows to use hardware functional blocks just like functions in a software library. To enable hardware virtualization, a dynamic hardware manager is required to monitor the available hardware functions instantiated in the reconfigurable regions and to handle the requests of the application for dedicated hardware functions. For instance, if a specific function is not available when the application needs it, the manager must schedule a reconfiguration.

The transversal nature of DPR makes this task difficult. A possible solution to implement transparent virtualized hardware management is to use a programmable controller in front of each reconfigurable region to handle locally the low-level layers of the virtualization. This controller checks locally the status of its reconfigurable region and sends a request to the scheduler for a reconfiguration when needed.

Figure 3.9 shows a comparison of the hardware complexity of a microcontroller and two hardware functional blocks used in OFDM modulation [110]. The microcontroller is a picoblaze, which is a small 8-bit microcontroller from Xilinx optimized for use in FPGAs [111]. The compared functional blocks are generated with the CoreGen tool from Xilinx [112], and therefore correspond to optimized implementations of those blocks. The resource count for the microcontroller is very small when compared to the size of the blocks that would be typically implemented in the reconfigurable region. Using a microcontroller to locally handle virtualization cause therefore only a limited resource overhead. As such, it is an interesting solution to provide an intermediate level of virtualization.

3.9 SUMMARY

This chapter has introduced dynamic partial reconfiguration of FPGAs. In the previous chapter, it has been shown that FPGAs are very interesting platforms for SDAs. Combined with dynamic partial reconfiguration, those platforms provide sufficient flexibility to enable the implementation of SDAs, and therefore leverage the advantages of FPGAs.

Several works have already studied the benefits provided by DPR in specific applications. However, this thesis has the particularity to focus on low-volume SDAs and their specificities. Therefore, in this chapter, the benefits of DPR for those applications are identified. It is shown that, besides the general benefits provided by DPR, the hardware virtualization capability provides particularly compelling advantages for SDA implementations on FPGA.

The impacts introduced by DPR on the design and development are also analyzed. It is shown that DPR requires design partitions, specific communication structures, a carefully dimensioned memory system and new functional blocks like a configuration port controller and a configuration manager. DPR has therefore an impact at every level of a design, from the bit-level to the operating system. The tools for development of DPR-enabled designs must reflect this and provide adequate solutions. Development flows, however, lack many elements. There are at present no models for DPR that allows to simulate the behavior or the timing of the DPR. There are also no tools allowing to debug DPR-enabled designs. Consequently, it is impossible for a developer to verify properly a design before integration. This complicates the design of applications with DPR. Those aspects are particularly important for some low-volume applications like safety-critical applications.

To handle reconfigurations in the low-level layers of an application, a solution is proposed that suggests to associate a small microcontroller to each reconfigurable region. In order to evaluate the feasibility of this approach, the resources required to implement such a configuration controller are evaluated and compared to the resources of typical functional blocks instantiated in the reconfigurable regions. The results show that the overhead caused by this approach would be very limited.

CHAPTER 4

A MINIMAL FLOW FOR FPGA DYNAMIC RECONFIGURATION

4.1 INTRODUCTION

Some low-volume applications like professional electronics have to be strictly validated. Among those, safety-critical applications like avionics are the applications that require the highest level of validation since those systems could affect the life of human beings. Those applications must be certified in order to guarantee that they will work exactly as specified. Examples of such certifications are the ED80/DO254 for hardware and ED12B/DO178B for software [113, 114]. Those two certifications are used in civil aviation and are amongst the most restrictive standards.

Certification imposes the use of a validation methodology during the whole development process. This methodology allows to prove that at each step of the development, from the initial specifications to integration, the application works exactly as expected. This continuous validation renders the development of safety-critical applications very difficult. Moreover, to enable certification of the application, the development flows and the tools must also be certified in order to prove that they work as specified.

FPGAs introduce specific issues for certification since their behavior is defined by the content of a configuration memory. However, using very safe design principles, those devices can be used in safety-critical applications such as avionics [115]. The use of DPR in those designs further complicates their certification.

Indeed, DPR introduces additional complexities in the design and development of signal processing applications. Furthermore, the tools and flows provided by the vendors to create the partial bitstreams are proprietary and not certified. This makes the certification of safety-critical using DPR extremely difficult.

In this chapter, a new DPR flow is presented. This flow implements the minimal steps needed to create partial bitstreams and perform DPR on Xilinx FPGAs. It is based on the parsing of complete bitstreams and do not necessitate any specific DPR tool from the vendor. This solution reduces the complexity of tools certification, and make a step towards enabling the certification of DPR in safety-critical applications. Moreover, some guidelines are proposed to provide safe design principles for DPR. Using those guidelines, the complexity of design certification is also reduced. In order to validate the flow, a design is implemented that represents a real software-defined radio application where the modulation can be changed dynamically by using DPR.

This chapter is organized as follows: the first section introduces in details safety-critical systems and the requirements introduced by certification. In the next section, related works on certification issues are introduced. In section 4.4, the issues related to FPGA design certification are discussed. and some solutions are presented. Additional issues introduced by DPR are also identified and described. Section 4.5 introduces the proposed minimal flow and the developed tools. Experimental results validating the functionality are presented and flow limitations are discussed. Finally, Section 4.6 presents some guidelines that, used along with the minimal flow presented here, allow to further reduce the complexity of DPR-enabled design certification.

4.2 CERTIFICATION OF SAFETY-CRITICAL SYSTEMS

During development of safety-critical applications the most important issue is to guarantee that the end product will fully comply to initial system specifications. To ensure this, standards have been defined that provide rules and requirements for all steps of the product life cycle. In the civil aeronautical domain, those standards are the DO-178B for software certification and DO-154 for hardware certification [116, 117]. Those standards define several levels of criticality, ranging from level A for highly critical systems to level D for low criticality systems. For each level, the standards define the processes to be followed and the controls that must be performed [118, 119]. It also defines the level of independence required to execute those controls in order to guarantee sufficient impartiality in the reviews.

In order for certification to be accepted by authorities, every requirements of the system must be translated into component requirements and functionalities. The totality of these requirements must then be validated. Every step of the design flow as well as the tools must be validated and the traceability must be maintained through the entire process. This method allows to prove that the product will work as expected at each step and each level of the design flow.

A solution to ease certification of a complex system consists in decomposing the design in sub-modules that are easier to validate. The requirements of the whole system must then be translated for each subsystem. The frontiers of those must strictly correspond to their implemented functions and each subsystem must embody a same level of criticality. Interfaces of the subsystems must be clearly defined and validated as well.

4.3 RELATED WORK

There are not many works studying issues related to design certification. Indeed, this is a complex domain, and most works require an industrial partnership. Several works have already studied the usage of programmable hardware components such as FPGAs in safety-critical systems [120, 121, 122]. Those works highlight the current lack of methodologies and qualified tools allowing to efficiently develop safety-critical applications with FPGAs. Some of them study the use of formal methods that allow to prove the functionality of a design. Those works present or compare existing methodologies, tools and programming languages that implement those concepts [123, 118].

Other works introduce methodologies allowing to analyze precisely the safety of a given system [124, 125]. Those methodologies allow to express all the failures occurring in a complex system in a unified formalism, and to analyze the possible propagation of faults from a given module to the rest of the system. Such approaches enable the identification of weaknesses in a design, and provide useful feedback on where to focus the development.

Those works show that the issue of design certification is not a purely technical problem. Indeed, dealing with certification issues introduces major challenges on design flows, design verification methodologies and tools helping for design certification.

All existing works relate to FPGA devices in their conventional static usage. However, none of them currently considers the use of dynamic partial reconfiguration. In this thesis, the certification issue is therefore studied for FPGA designs using DPR in safety-critical applications. In particular, this work studies the design flow and design verification methodology issues. It also provides a flow for partial bitstream creation, which is an essential step in DPR flows, and design guidelines. Combined together, those contributions allow to significantly ease the certification of DPR designs.

4.4 CERTIFICATION OF FPGA DESIGNS

4.4.1 Certification of conventional FPGA designs

Certification is difficult to obtain for FPGA designs since they have specific complexities that complicate certification for safety-critical applications:

- They are programmable, they use a configuration memory and they require the use of a configuration process.
- They allow to integrate several functions in one device. This allows to reach better performance but makes the validation of the complete system harder.
- FPGA IPs and third-party code allow to have a competitive development process, but their certifications are an issue.
- The integrity of configuration memory must be ensured since it can be altered, for instance by soft-errors (SEU) which are frequent in avionics.
- FPGA development flows have specific additional steps like bitstream generation and loading. Those steps bring additional potential failures and must be certified.

Static FPGA designs are already used in avionics. Certification of FPGA devices is partly obtained by their use in many designs, which provides an extensive functionality coverage. Device issues are therefore regularly reported to the vendor and corrected. However, solutions must still be found to monitor and validate the design.

Several solutions exist to ease FPGA design certifications, one of them is modular designs. In such designs, modules are placed and routed and validated separately. Inter-module communications can be realized by using external IOs in order to further guarantee independence between modules, and avoid failure contagion. To ensure configuration memory integrity, the system can be continuously monitored, for instance with a checksum mechanism regularly performed on the content of configuration memory via readback [126].

4.4.2 DPR certification issues

The use of DPR in a design implies to comply to a set of additional design rules. It also implies additional steps and processes during development and system use. Those additions cause other issues to arise when certifying DPR-enabled designs:

1. **certification of the DPR flow:** the tools used to perform the operations related to DPR during development must be certified to prove that they perform as they are defined.
2. **partial bitstream validation:** the partial bitstreams generated by the tools must be valid. They need to be checked to ensure that they respect correct bitstream definitions.
3. **configuration port controller and configuration interface:** the configuration port controller introduces additional hardware in the design. It

must be validated to work properly. Additionally, the configuration must be certified to work properly. This is mostly an issue for the internal port since it is seldom used.

4. **transient states during reconfiguration:** some FPGA devices do not provide guarantee of glitchless reconfigurations. The design must ensure that the reconfigured regions are properly isolated during reconfigurations to avoid errors.

5. **several functional modes for a single module:** partial reconfiguration allows to change the functionality of a module. This makes module isolation harder since a part of a component cannot be identified as a single module anymore. Interfaces and monitoring systems must be adapted.

6. **more complex designs, particularly when complex scheduling is used:** dynamic modification capability of DPR-enabled designs allows them to have many different working configurations. Those configurations must be properly identified and certified to work properly.

4.5 A MINIMAL FLOW FOR DPR

4.5.1 Definition

The proposed flow is illustrated in Figure 4.1. The prerequisite for this flow is to have a DPR-enabled design, as explained in the previous chapter, with static and reconfigurable regions and bus macros between regions. The flow allows to create partial bitstreams for each functional block used by the application. In order to do so, it extracts configuration data from static snapshots of the design.

The flow is composed of 5 main steps:

1. **Static synthesis:** the first step consists in creating a static snapshot of the application. This step is realized by performing a static synthesis of the complete design with all reconfigurable regions configured with one of their functional blocks. The usual static flow is used here. In the following steps, a single reconfigurable region configuration is extracted from the generated bitstream to create one partial bitstream.

2. **Bitstream parsing:** during this step, the bitstream created in the previous step is read and every frame it contains is extracted. A set with all the frames is created. As the bitstream organization is known, it is possible to associate each extracted frame with its position in the FPGA.

3. **Frame selection:** as every frame composing the reconfigurable region is identified, it is possible to find all of them in the frames extracted in the previous step. Therefore, in this step, every frame from this region is selected from the complete set of frames.

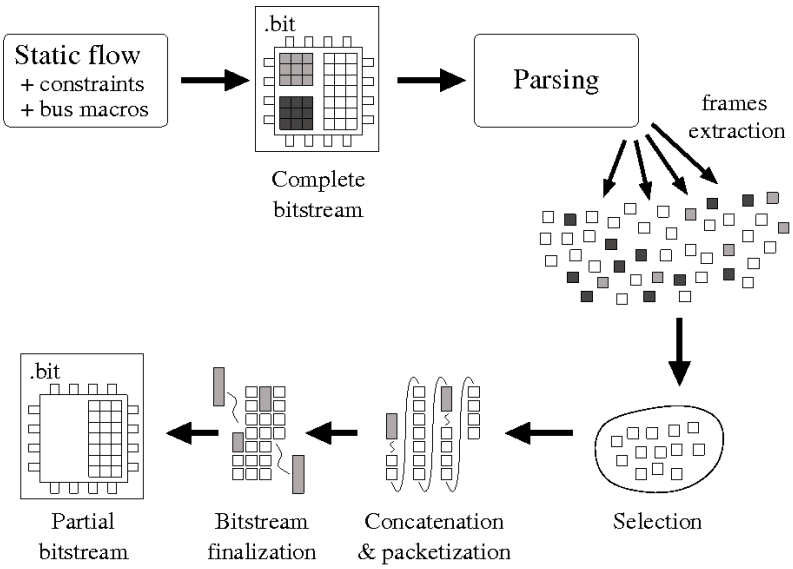

Fig. 4.1. Steps of the proposed minimal flow for DPR. A complete bitstream is created with the static flow. This bitstream contains a configuration of the reconfigurable region to extract. The bitstream is parsed, configuration frames are identified and the frames of the reconfigurable region are extracted. A partial bitstream is eventually composed with those frames.

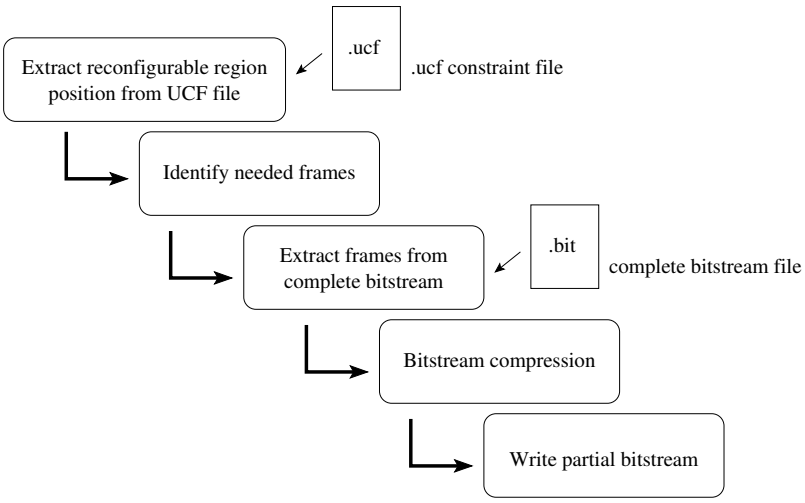

Fig. 4.2. Main steps performed by the developed tools implementing the proposed flow.

4. **Concatenation and packetization:** once all the frames from the reconfigurable region are selected, they are placed one after the other, in their address order. For each sequence of frames having consecutive addresses, a frame writing header starting at the first address of the sequence is added.

5. **Bitstream finalization:** in this step several packets are added to the ordered frames to compose a valid bitstream. These are the initialization header and the bitstream tail. They contain instructions that start and close the configuration process.

The tools used to create the initial bitstream are the tools of the conventional static flow. The proposed flow adds scripts that realize bitstream parsing, frame manipulation and bitstream composition. Every step has a moderate complexity. Those steps are described in more details in the next section.

To create all the partial bitstreams needed in an application, complete static bitstreams must be synthesized to cover all possible reconfigurable region configurations. This can be done by repeating step 1 several times. The placement constraints and the bus macros guarantee that the extracted functional blocks are entirely contained in the reconfigurable regions, and that communications with other blocks always use the same routes.

When there are several reconfigurable regions in the design, a single complete bitstream created in step 1 can be used to create several partial bitstreams. Steps 2-5 must then be repeated once for each reconfigurable region to extract.

4.5.2 Flow implementation

Tools have been developed in order to implement the proposed flow. The tools perform the complete flow in five main steps, illustrated in Figure 4.2. Those steps are:

1. **Extract reconfigurable region position:** In order to partition the design, the Xilinx tools use a constraint file called the UCF file. In DPR-enabled designs, this file contains the placement constraints which assign all design modules to static and reconfigurable regions. Those regions are defined as rectangular areas in the component. The first step of the developed tools consists in reading the UCF file to obtain the position and extent of the reconfigurable region to extract. In order to do so, the placement constraints contained in the file are parsed in order to get the information. Using this information, the configurable resources contained in the reconfigurable region can be identified.

2. **Identify needed frames:** this second step consists in identifying the frames to extract in the complete bitstream. As presented in the previous chapter, each configuration has a unique address. To perform this step, a mapping must be established between the addresses of the configuration

frames in the bitstream and the resources they configure in the component. However, as the mapping is not complex, it can be easily obtained by analyzing bitstreams. Using this mapping and the list of resources identified in the previous step, a list of frames to extract is built.

3. **Extract frames from the complete bitstream:** using the list of frames obtained in the previous step, the file containing the complete bitstream is parsed and the needed frames are extracted. The bitstream structure is explained in a datasheet provided by the vendor [71]. As explained in the previous chapter, this structure is simple: it is composed of packets containing commands for the configuration engine. Packets with write commands contain several configuration frames arranged as a sequence of frames with consecutive addresses. Those packets have all a header which provides the top address of the frame sequence. The address of all the frames contained in the packets can therefore be precisely determined. Consequently, all frames in the bitstream can be easily identified, and extracted when required. After this step, all the required frames are placed into a list, together with their addresses in order to identify them.

4. **Bitstream compression:** this step is optional. It uses a special technique that Xilinx bitstreams use to reduce the size of bitstreams, called multiple frame writing [71]. With this technique, the configuration data contained in one packet can be written at several locations in the configuration memory. Frames with identical content necessitate therefore only one write packet, followed by several special commands to copy it at multiple locations. Currently, the developed tools are only able to detect and compress empty frames, which are the most redundant frames. After this step, the list of frames built in the previous step is reduced to the list of nonempty frames. The address of empty frames are kept in a separate list.

5. **Write partial bitstream:** using the list of extracted frames, a valid bitstream is recreated by composed and concatenating packets. For each sequence of consecutive frames to write, a new packet is created. For each of those packets, a header with a write command is placed in front of the frame sequence. This operation is illustrated in the "concatenation & packetization" step in Figure 4.1. If bitstream compression is enabled, multiple frame writing is used for empty frames, using the list of empty frame addresses created in the previous step. The last part of this step consists in finalizing the bitstreams by adding a special header and a tail to the concatenated packets. This operation is illustrated in the "bitstream finalization" step in Figure 4.1. Those additional packets contain commands directed to the configuration engine, they initialize and end the configuration process. They also contain various information about the targeted device and the bitstream size, to allow verification of the bitstream consistency.

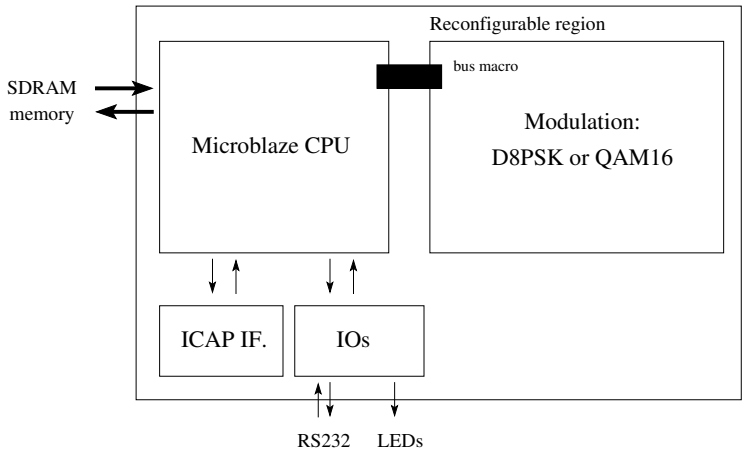

Fig. 4.3. Schematic of the DPR-enabled design used for experiments and verification of the proposed flow. The design contains a single reconfigurable region which performs a modulation. The modulation can be configured either in D8PSK mode or in QAM16 mode.

All the operations performed in these tools are based on the bitstream organization, as described in the datasheet provided by the vendor [71]. Only one step, i.e. frame identification, necessitate additional information, which is easily obtained by analyzing bitstreams since frame addressing is relatively straightforward. The tools have been implemented with several scripts written in python. The complete implementation of all the steps described in this section represents about 600 lines of code. All the operations performed in these steps have a low or moderate complexity since the bitstream structure is simple.

4.5.3 Experimental results

To validate the flow, a DPR-enabled design has been implemented. The design is illustrated in Figure 4.3, it implements a simple software-radio application where the functional blocks in charge of the modulation can be reconfigured. Two telecommunication functional blocks have been implemented for the reconfigurable region: a D8PSK modulator and a QAM16 modulator. The design is composed of a MicroBlaze soft-core CPU which controls the reconfiguration of the device. There is only one reconfigurable region which represents about a quarter of the device.

The design has been implemented in a Xilinx Virtex-4 LX 60 FPGA using the XC4VLX60 evaluation board from Avnet. All the modules of the design run at 80MHz. The critical path is situated in the SDRAM memory controller. The complete bitstreams used to create the partial bitstreams with the proposed flow have been created using the Xilinx ISE 7.1 and EDK 7.1 tools. Two partial

bitstreams were created: one for the D8PSK modulator, and one for the QAM16 modulator. For the experiments, the partial bitstreams are stored in the external SDRAM memory. To perform DPR, the Microblaze reads the bitstream in the SDRAM and writes them to the ICAP.

The system can successfully reconfigure a module into another using the partial bitstreams generated with the proposed flow. About 2800 frames are written during the reconfiguration. The sizes of the partial bitstreams are about 450KB each. In this design, the reconfiguration is performed using the MicroBlaze to write in the configuration port.

4.5.4 Limitations

The experiments show that the proposed flow can successfully realize partial dynamic reconfiguration of an FPGA design by using a simplified flow which is therefore easier to certify. However, it has some limitations:

- The flow does not allow reservation of resources inside modules. This prevents the use of long communication wires that passes through reconfigurable modules. This could be an issue for access to external IOs since those have fixed places in the device. Pin placement should therefore be planned carefully.
- The two columns located in the center cannot be used. These columns contains the FPGA clock tree configuration bits, which are not documented.
- The flow is strongly device-dependent: the positions of the columns are hardcoded and must be adapted for each different device.
- The granularity of reconfiguration is limited to a frame. There is no possibility to reconfigure a part of the logic configured by a frame. The configuration bits of a frame are not documented.

4.6 CERTIFICATION OF DPR-ENABLED DESIGNS

The proposed flow has been implemented as a script of about 600 lines of python code, which performs simple operations like file read and write, address computations, concatenations, etc. This moderate complexity allows to easily verify that the script works correctly, which eases its certification. Since other parts of the flow rely on the conventional static tools and processes, certification of those part are obtained through their frequent uses in conventional FPGA designs which provide a good bug coverage.

The flow alone is not sufficient to provide certification of DPR-enabled designs since the additional design techniques and functionalities that are required to implement DPR must be certified as well. However, under these additional guidelines, validation of the design can be facilitated:

- Communications are performed outside the FPGA, using external IOs. This behavior avoids the need for bus macros, and enables IO monitoring.
- Reconfigurations are performed using the external port, this avoids ICAP validation, which can be a problem since it is not commonly used.
- An external controller computes a checksum of the entire configuration memory. It proves that, after the reconfiguration process, (i) the integrity of the configuration memory of the fixed part is preserved and (ii) the reconfigurable region holds the expected function. This technique is already used for static designs.
- Use well-separated functionalities between the fixed and reconfigurable module. It is for example the case in the presented radio application where an entire waveform resides in the reconfigurable module. This approach allows to use the partitioning methodology explained in Section 4.4.1.
- The component can be stopped during reconfiguration, this avoids transient states. The reconfiguration is then no more dynamic but stays partial, which still allows to benefit from hardware virtualization.

The combination of these guidelines and the flow presented in the previous section provide solutions for the design flow and methodology issues introduced by the use of DPR in safety-critical systems. Together, they allow to ease significantly the certification of those designs. However, there remain several issues to consider. Work is indeed needed to provide formal verification methods capable of modeling DPR. Analysis methods are also needed to precisely characterize failures in DPR designs, and identify fault propagation. DPR integration in tools assisting the development of safety-critical applications is also a much needed feature.

4.7 SUMMARY

Some low-volume applications require strict validation or certification. In particular, safety-critical applications must be certified in order to guarantee that they will work exactly as specified. Certification imposes the use of a validation methodology during the whole development process. This continuous validation renders the development of safety-critical applications very difficult. Moreover, to enable certification of the application, the development flows and the tools must also be certified in order to prove that they work as expected. However, DPR introduces additional complexities in the design and development of signal processing applications and the tools and flow for DPR are not certified. This makes the certification of safety-critical applications using DPR extremely difficult.

Existing works on FPGA usage in safety-critical applications do not cover issues related to the use of DPR in FPGA designs. This chapter studies those

specific problems. The focus is on the tools, and more precisely, on the development flow for partial bitstream creation, which is a crucial part of DPR development. Design issues introduced by DPR requirements are also analyzed.

In this chapter, a new DPR flow is presented. This flow implements the minimal steps needed to create partial bitstreams and perform DPR on Xilinx FPGAs. It is based on the parsing of complete bitstreams and do not necessitate any specific DPR tool from the vendor. This solution reduces the complexity of tool certification, which eases the certification of DPR in safety-critical applications. Several guidelines are also proposed that provide safe design principles for DPR. Using those guidelines, the complexity of design certification is further reduced. The proposed flow has been validated on an application that use DPR.

The contributions in this work provide solutions for the design flow and methodology issues introduced by the use of DPR in safety-critical systems. However, future works are still required to solve remaining issues. Indeed, formal verification methods capable of modeling DPR are required, as well as failure analysis methods for DPR-enabled designs. DPR integration in tools assisting the development of safety-critical designs is also a much needed feature.

CHAPTER 5

EVALUATION OF DPR IMPACTS ON FPGA DESIGN POWER CONSUMPTION

5.1 INTRODUCTION

Static FPGA designs require to instantiate all the functions needed by the application in the device at boot-time. The set of available functions is therefore limited by the size of the component. To provide a large set of functions, such designs must either use large FPGAs, or use generic and therefore suboptimal hardware implementations, or even execute the function in software if no hardware implementation is available.

DPR allows to change the functional blocks instantiated in an FPGA without service interruption. Partial bitstreams of functional blocks can be stored in memory. Since those memories are large, many configurations can be stored to provide a large library of kernel hardware implementations. This library can be composed of many optimized implementations of signal processing kernels. Several implementations for the same functionality can even be stored in order to fit specific problem sizes or parameters. This approach provides a very high dynamic change capability to FPGA designs and enables true hardware virtualization.

Hardware virtualization allows to perform a signal processing kernel by instantiating dynamically the required functional block in the FPGA when needed. Since those functional blocks are optimized for the executed kernel and therefore highly efficient, hardware virtualization also provides new opportunities to improve the power efficiency of an FPGA design [127]. Indeed, using optimized

hardware implementations rather than a software implementation allows to reduce the energy required to perform a task.

DPR also introduces new sources of power consumption. To perform a partial reconfiguration, a bitstream must be read from memory, written in the reconfiguration port, and the configuration machinery must then partially overwrite the configuration memory. Energy is therefore consumed to perform each reconfiguration. Moreover, if the system must comply to strict real-time constraints, fast memories and a complex memory system are needed to increase the bitstream transfer speed. Those configurations have a higher power consumption, which increases the energy required to perform a reconfiguration.

This chapter presents an evaluation of the impacts of using DPR on the power consumption of an FPGA design. In order to evaluate the opportunities of power reduction enabled by the DPR, comparisons of energy consumptions are made between optimized hardware implementations of several functional modules and their software implementations. The comparisons are based on actual measures realized on a DPR-enabled design implemented on an FPGA development board. The sources of power consumption related to the reconfiguration are identified. These sources are evaluated by performing measures and estimations. The evaluation take into account the influence of real-time constraints on the power consumption.

The text is organized as follows: In the first section, the platform and the design used to perform the measures are presented. In the second section, works related to DPR power consumption evaluation are presented. In the next section, an evaluation of the gains provided by optimized implementations of functions dynamically instantiated in the FPGA is realized. The following section extends the evaluation by performing a comparison with a silicon processor developed in this thesis. Section 5.6 presents the evaluation of the different sources of power consumption related to partial reconfiguration. Section 5.7 discusses the power consumption reduction enabled by DPR in regard to the total reconfiguration energy cost. Section 5.8 analyzes the impact of real-time constraints on power consumption. Finally, the last section presents an evaluation of the potential chip size reduction provided by DPR for an SDR application based on the benchmarks used in this chapter.

5.2 RELATED WORK

The power consumption in DPR-enabled designs has been the subject of several works. Some of them have studied the opportunity to reduce power consumption by exploiting DPR. In [128] and [129], power reduction gains are obtained by using dynamic adaptations of the communication network and clock frequency management. Other works perform measures and estimations of the power consumption related to dynamic reconfigurations. In [130] and [131], measures of the dynamic power consumptions for complete and partial reconfigurations are realized for different FPGA components. In those evaluations, only the component

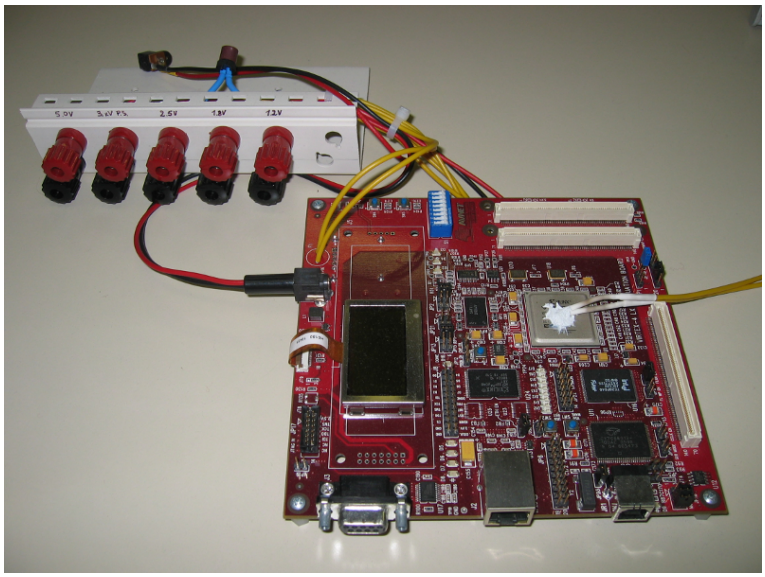

Fig. 5.1. Modified FPGA development board used for the current measures.

power consumption is taken into account. In this thesis, the power reduction opportunities provided by DPR are also evaluated. However, the study is focused on the gains related to the exploitation of hardware virtualization. Moreover, the evaluations consider the additional power consumption sources related to bitstream transfers from external memories.

5.3 EXPERIMENTAL SETUP

To evaluate DPR impacts on power, current measures have been realized on a test platform. The platform is based on an AVNET development board for the Virtex-4 LX60 FPGA from Xilinx. This FPGA model is implemented in 90nm technology, and its core voltage is 1.2V. The board, shown in Figure 5.1, has been modified to allow measurements of the currents passing through the voltage regulators. A single voltage regulator provides current to the internal logic of the FPGA. By measuring this current, precise measures of the FPGA internal power have been made. Since leakage currents are strongly temperature-dependent, a thermal sensor has been mounted on the component to monitor the device temperature. The measures of power consumption have all been performed at a steady temperature. With this setup, evaluations of current consumptions can be realized by performing differential measures on the whole design with and without the block to measure.

The layout of the FPGA design used for the experiments is presented in Figure 5.2. For the experiments, a simple partition scheme with regions covering the

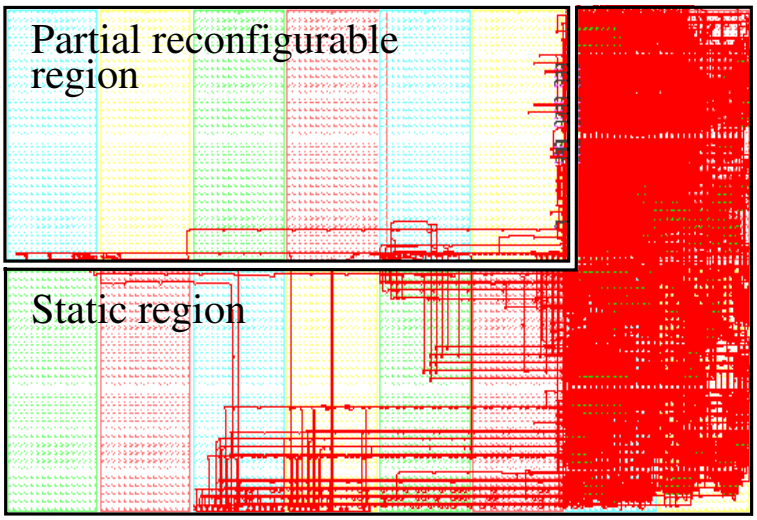

Fig. 5.2. FPGA test platform layout. The area is subdivided in two regions: a reconfigurable region where modules are instantiated and a static region with a Microblaze soft-core CPU which acts as a controller.

whole device length has been used in order to avoid placement conflicts and ease development. Indeed, physical placements of IOs in this board are not optimal and require to have a large static region covering most of the device. Moreover, using simple partitions eases the work of the FPGA synthesis tools.

The design is partitioned in a static region and a reconfigurable region. The reconfigurable region represents about 38% of the FPGA area. The static region contains a Microblaze soft-core CPU. The Microblaze is a single-issue RISC core provided by Xilinx. It is used to communicate with the system and to manage the reconfigurations. The CPU is connected to the memory controllers and the internal configuration port using an OPB bus [132]. For simplicity, the entire design uses a single clock at the frequency of 80MHz. The critical path is situated in the DDR memory controller.

Six different reconfigurable modules have been implemented as benchmarks:

- A 32-bit counter: this simple module is used to evaluate the impact of the underutilization of reconfigurable region resources.

- Two modules performing D8PSK and QAM16 modulations: those modules allow to realize the evaluation of a software-defined radio application.

- Two FFT modules: a 8-point pipelined FFT module and a 1024-point FFT module implemented which has been implemented with an area minimization constraint. FFT is a very popular operation in signal processing.

The different implementations allow to evaluate the power consumption of different implementations of the same function.

- Another Microblaze soft-core CPU: this module provides a complex functional block for the experiments, with a high logic resource usage. The Microblaze processor module uses about 33% of the resources of the reconfigurable region.

The benchmarks have all been implemented in hardware and in software, in order to enable comparison of both implementations. For the hardware implementations, the counter and the two modulators have been implemented by hand. The FFT modules have been automatically generated using the Xilinx hardware generation tools, CoreGen [112]. The Microblaze CPU is generated by Xilinx embedded platform tools, EDK. The software implementations have all been coded in C and compiled using the Xilinx embedded tools. Version 8.2 of the Xilinx tools have been used for hardware and software implementations.

All the benchmarks have been validated by following a top-down development methodology. With the exception of the Microblaze processor benchmark, all benchmarks have first been implemented in plain C, compiled, and run on a standard x86 machine to provide a reference implementation. Those implementations have been used to create a set of input and output test vectors. Afterwards, Microblaze software implementations and FPGA hardware implementations have been realized as explained. For each input test vector, the outputs of those implementations have been compared to the output of the reference implementation in order to validate them.

5.4 EVALUATION OF POWER REDUCTION OPPORTUNITIES

DPR enables hardware virtualization in FPGA designs. With hardware virtualization, hardware functional block configurations can be stored in an external memory. When the application needs to execute a specific kernel, the partial bitstream of the related functional block is fetched from the memory and is dynamically loaded in the design. Since external memories can be large, many hardware functional blocks can be stored. For instance, the partial bitstream of a complete Microblaze core represents about 500KB. 1GB of flash memory can therefore store more than 2000 of those. Consequently, hardware virtualization allows to create large libraries of hardware functional blocks, which provides a function diversity and a flexibility comparable to a software implementation.

By exploiting hardware virtualization, highly versatile applications like SDAs can execute their signal processing kernels by using optimized hardware implementations instead of software implementations. Using an optimized implementation allows to speed up the execution and to reduce the energy needed to perform the operation compared to a software execution.

In order to precisely evaluate the power consumption benefits of hardware virtualization, a comparison has been realized between the energy required to

perform a task using a software implementation and a hardware implementation dynamically instantiated in a reconfigurable region. The comparison is realized for the modules presented in the previous section. For each module, measures have been made of the task execution time and of the mean current consumption of the FPGA during the execution.

For the hardware implementations, task executions are controlled by the Microblaze CPU included in the design. The processor starts the tasks and then stay idle while waiting the end of the execution. Using hardware timers, the task execution time is precisely measured directly in the design. Current measures of the FPGA are performed during the task execution. As most of the task execution times are very short, current measures are hard or impossible to perform for a single task execution. To perform the measures, the tasks are therefore executed several times, back to back, in order to render the current consumption observable. The measured current correspond therefore to the mean current consumption of the whole FPGA while executing the kernel. The mean current consumption of the design has also been measured while all functional blocks and the processor were idle. By subtracting both values, the current consumed for the task execution can be computed.

For the software implementations, the tasks are executed directly on the Microblaze processor. The evaluation methodology is the same as for the hardware implementation: execution time is measured using hardware timers, and the current is evaluated by measuring the device mean current consumption during back to back executions and by subtracting the idle current consumption. By combining the supply voltage of the FPGA, the execution time and the mean current consumption, the energy consumed for each task execution can be computed. The results are shown in Table 5.1.

The results show that important gains can be obtained by using the optimized hardware implementation of the function. The results also present important variations in the gains between modules, which shows that each function implementation has its specific performance. This illustrates that different implementations from a set of functionally equivalent implementations lead to very different performance. There is therefore an opportunity to further reduce the power consumption by using highly specific implementations of a kernel providing the best performance for specific problem size, parameters and application constraints. This particular opportunity is not fully evaluated in this work. However, in [133], Noguera and Kennedy have shown that exploiting this technique can lead to significant power reductions. Using a Viterbi decoder as their benchmark, they show that a dynamic power reduction of about 75% can be obtained by switching to a decoder implementation with a lower error correction strength. In telecommunication applications, this reduction can be dynamically exploited if, for instance, the channel quality improves or if the required data rate is reduced.

	Power [mW]	Time [μs]	Time ratio	Energy [nJ]	Energy ratio
cptr32 sw	141.7	215×10^6	-	30.5×10^9	-
cptr32 hw	3.1	53.7×10^6	4.0	168.2×10^6	181.3
d8psk sw	45.3	1.6	-	72.8	-
d8psk hw	26.9	0.71	2.3	19.1	3.8
qam16 sw	53.6	1.6	-	85.7	-
qam16 hw	23	0.71	2.3	16.3	5.3
fft8 sw	64.4	13.6	-	876.4	-
fft8 hw	5.8	0.65	20.9	3.8	230.6
fft1024 sw	64.4	3×10^3	-	193.3×10^3	-
fft1024 hw	24.5	92.1	32.6	2.26×10^3	85.5

Table 5.1. Execution time and energy comparisons between the software implementation of a functional block and its hardware implementation.

5.5 PERFORMANCE COMPARISON WITH A SILICON PROCESSOR CORE

In the previous section, comparisons have been realized using a soft-core processor to execute the software benchmarks. This configuration corresponds to a complete implementation of the application in the FPGA, using a single component. This approach is the most interesting one to implement SDAs since it allows to reach a very strong integration level by reducing the number of required components. It allows to reduce the costs related to the platform and the energy consumption overhead caused by communications between discrete components.

However, it is interesting to compare the results with the performance of a silicon processor. Indeed, those implementations do not suffer from the consumption overhead of FPGA reconfigurable logic. Those processors provide therefore a better energy efficiency than soft-core processors. Moreover, some FPGA models also provide silicon processors on their dies. Using a silicon processor is therefore also possible when the platform is completely implemented in a single FPGA.

In order to evaluate the difference introduced by a silicon processor, a comparison is realized using the $DSP1$ processor developed in this thesis. The implementation and the composition of this core is detailed in Chapter 6. Performance comparisons are realized with the FFT1024 benchmark. However, the architectures of the $DSP1$ and the Microblaze processor cores are very different: they have different ALUs, they use different ISAs and microarchitectures. Those differences introduce interferences in the results. In order to realize a more precise performance comparison, additional works should therefore be realized, which could not be done in the scope of this thesis.

Table 5.2 presents comparison results. In order to make those results comparable, all numbers have been converted in the 65nm technology, at a 1V supply

	$DSP1$ $[J]$	FPGA_{SW} $[J]$	$\frac{DSP1}{\text{FPGA}_{SW}}$	FPGA_{HW} $[J]$	$\frac{DSP1}{\text{FPGA}_{HW}}$
FFT1024	6.5×10^{-6}	104.3×10^{-6}	0.06	1.22×10^{-6}	5.3

Table 5.2. Comparison of energy consumptions of the FFT1024 benchmark between a processor core implemented in silicon and the results obtained on FPGA, for the software and hardware executions. All results are converted to the 65nm technology at 1V.

voltage. The conversion has been performed using the ratios presented in Chapter 2. The results show that the $DSP1$ processor core consume around 16x less energy than the Microblaze soft-core processor. However, the FPGA hardware implementation of the benchmark is still more interesting than the $DSP1$ software implementation. Indeed, the $DSP1$ consumes about 5x more energy than this implementation. The results confirm therefore that the FPGA is a very interesting platform regarding energy consumption.

5.6 EVALUATION OF THE RECONFIGURATION POWER CONSUMPTION

Performing dynamic partial reconfiguration requires to fetch a partial bitstream from a memory and to send it to the configuration port controller which writes it in the configuration interface. The internal reconfiguration machinery then rewrites partially the content of the configuration memory. The execution of this process consumes power. Dynamic reconfigurations introduce therefore new power consumption sources. Three main sources of power consumption have been identified: bitstream transfer from external memories, bus transactions between the memories and the configuration port controller, and switching of the internal FPGA configuration hardware (i.e.: configuration engine, configuration memory).

In typical DPR, the bitstreams are stored in nonvolatile memories, like flash memories. Those memories allow to store a large amount of data in a permanent way, but they have a higher power consumption and a slower transfer speed than volatile memories. However, in applications with strong real-time constraints requiring fast reconfigurations, faster memories must be used, like DDR volatile memories. In order to evaluate the power consumption of the memory transfers, both types of memory must therefore be considered.

To evaluate the power consumption of the internal configuration hardware and bus transactions, current measures have been realized on the FPGA while performing partial reconfigurations. The largest bitstream among the six considered modules, which holds the configuration for a complete Microblaze processor module, represents about 500KB. The transfer of this bitstream through the reconfiguration interface represents an energy consumption of 0.2mJ.

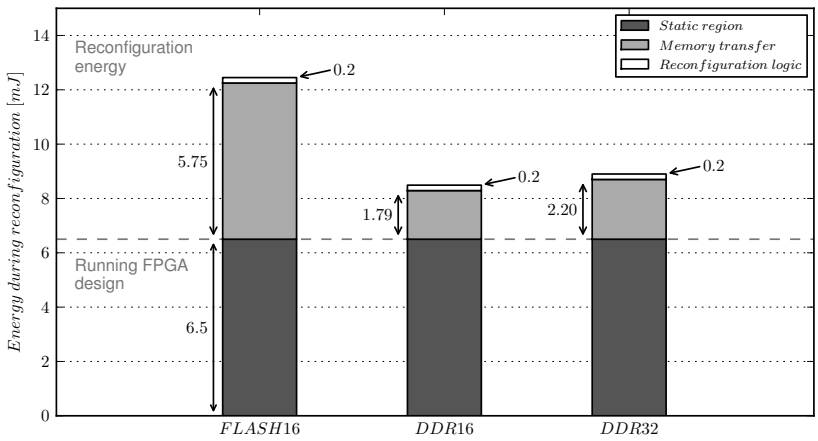

Fig. 5.3. Energy consumed by the FPGA and the memories during partial reconfiguration with a 500KB bitstream.

The power consumption caused by bitstream transfers from the memories have been estimated by using performance numbers from datasheets [134, 135, 136]. Three memories have been considered for the evaluation: a 16-bit nonvolatile flash memory and two DDR memories with data widths of respectively 16 and 32 bits. The first two models correspond to the memories present on the used development board. The estimations give an energy consumption of 11.5nJ/byte for the flash memory, 3.5nJ/byte for the 16-bit DDR and 4.4nJ/byte for the 32-bit DDR. Consequently, reading the 500KB bitstream costs 5.75mJ for the flash, 1.79mJ for the DDR16 and 2.20mJ for the DDR32.

Figure 5.3 shows the breakdown of the energy consumed to perform a reconfiguration with a 500KB bitstream for the three memory configurations. Since the FPGA static region continues to operate during reconfigurations, its energy consumption is also illustrated in the figure for comparison. This energy is however not associated to the reconfiguration process. The energy related to the reconfiguration is represented in the figure by the reconfiguration logic and memory transfer. The results show that the most important contribution is due to the reading of the bitstream in external memories. This energy is significantly more important than the energy consumed in the internal reconfiguration logic. Even for DDR SDRAM where it is lower, the memory transfer energy is about 8.95x higher than the internal logic energy.

5.7 IMPACT OF THE RECONFIGURATION COST

In the previous section, it has been shown that the energy consumption due to FPGA reconfigurations is related to bitstream transfers from the memories

	Bitstream size [KB]	**Reconf. energy [J]**	**HW exec. energy [J]**	$\frac{\textbf{Energy}_{reconf.}}{\textbf{Energy}_{exec.}}$
32-bit counter	144.3	5.75×10^{-4}	168.2×10^{-3}	3.42×10^{-3}
D8PSK	346.1	1.38×10^{-3}	19.1×10^{-9}	72.25×10^{3}
QAM16	324.6	1.30×10^{-3}	16.3×10^{-9}	79.75×10^{3}
FFT8	333.6	1.33×10^{-3}	3.8×10^{-9}	350×10^{3}
FFT1024	346.3	1.38×10^{-3}	2.26×10^{-6}	610.62
Microblaze	488.8	1.95×10^{-3}	-	-

Table 5.3. Comparison of task execution and reconfiguration energies for the considered benchmarks.

and to the FPGA internal configuration hardware. This consumption introduces a fixed energy cost to each reconfiguration. In order to gain from the power consumption reduction provided by the use of hardware implementations, it is therefore necessary to make the reconfiguration cost profitable.

Table 5.3 shows FPGA reconfiguration costs for the considered hardware functional blocks. For each benchmark, the table presents the associated partial bitstream size, the total energy cost for the reconfiguration, and the task execution energy. By computing the ratio between the reconfiguration cost and the task execution cost, it is possible to express the reconfiguration costs in terms of a task execution count. This ratio is given in the last column of the table. The results show that, except for the 32-bit counter, all benchmarks must be executed many times to consume as much energy as the reconfiguration requires. For instance, for the QAM16 kernel, the task must be executed about 80.000 times.

For comparison, in the 802.11a waveform at 54Mbit/s, a 64-point FFT is executed more than 220.000 times each second. Those results indicate therefore that dynamic partial reconfiguration is mostly interesting when configuration modifications are exploited for a significant time, e.g. when switching from a modulation to another for the time of a complete communication. On the contrary, this technique is less interesting when changing the architecture of the system with a finer time scale, like for instance in between two consecutive data frames.

5.8 MEMORY SYSTEM POWER CONSUMPTION

In section 5.6, it has been shown that memory transfers are the major contributor to reconfiguration power consumption. The energy consumed by these depends on the bitstream size, but also on the memory system and the application real-time constraints. For applications with strict real-time constraints, reconfiguration time must be kept sufficiently short to respect the constraints. To reduce reconfiguration time, the memory system must be designed to allow bitstream transfer to the configuration port controller at sufficient speed. For

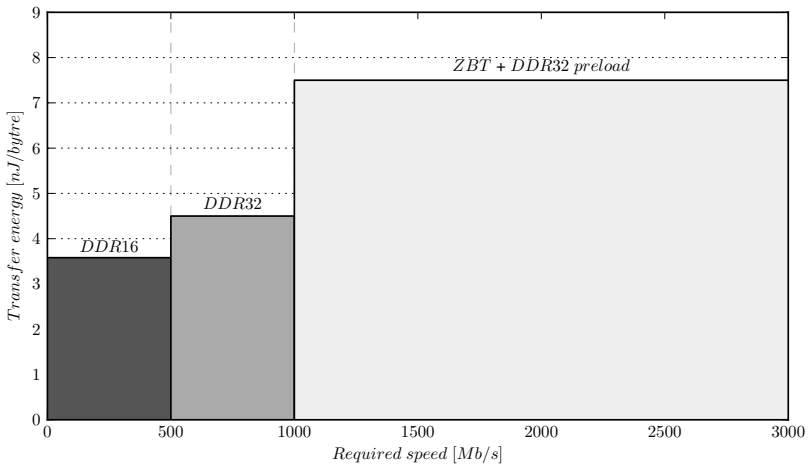

Fig. 5.4. Energy consumed by the memory system during bitstream transfers from memories to the FPGA depending on the required throughput.

instance, specific memories such as Zero-Bus Turnaround (ZBT) RAM could be used to reach high data transfer rates. However, since those memories are small, they require first to transfer the bitstream from a large but slower memory to them. Those additional transfers cause more power to be consumed for each reconfiguration. Consequently, depending on the required bandwidth between the memory and the configuration port controller, the energy consumed by memory transfers will vary.

An estimation of the memory system power consumption has been realized for a range of memory bandwidths going up to 3.0Gb/s, near the theoretical maximum bandwidth of the configuration port at 3.2Gb/s. The results are shown in Figure 5.4. For each bandwidth value, the figure presents the energy consumed by the transfers. For each range of throughputs, the illustrated value corresponds to the memory system configuration causing the lowest power consumption. The results show that for low throughputs, DDR memories can be used. When high throughputs are needed, fast memories like ZBT are required. Since faster memories like ZBT require more transfers to preload bitstreams, the complete operation consumes more energy. The fastest configuration considered here is obtained using a ZBT RAM preloaded with data from a 32-bit wide DDR SDRAM. With this configuration, the memory transfers consume up to 2.1x more than the slowest configuration. The memory system can therefore consume significantly more energy to perform reconfigurations when high speeds are needed.

Module	**Reconfigurable region usage [%]**	**LX-60 usage [%]**	**LX-60 usage with static region [%]**
D8PSK	7.8	2.92	27.92
QAM16	7.8	2.92	27.92
FFT8	9.75	3.66	28.66
FFT1024	14.34	5.38	30.38
Microblaze	32.95	12.38	37.38
All in parallel	-	27.26	51.73*

Table 5.4. FPGA resource usage of the benchmark modules. Number with an asterisk ("*") do not take the reconfiguration interface into account.

5.9 CHIP SIZE REDUCTION

With the exception of the 32-bit counter, the hardware modules considered in this chapter correspond to functional blocks used in a software-defined radio application where the platform is able to dynamically change the implemented communication standard. In the considered application, the platform is composed of a Microblaze processor core acting as the application controller, and the modulations are performed in hardware using functional blocks instantiated in the FPGA. The platform can perform a D8PSK modulation, a QAM16 modulation, or OFDM-based modulations using the FFT modules. Another Microblaze processor core can also be used to perform the modulation in software with full throughput when the modulation to perform cannot exploit the available hardware functional blocks.

To provide all those functionalities, a conventional static design would have to instantiate all functional blocks in parallel in the FPGA. Alternatively, DPR can be used to instantiate dynamically the required functional block in a reconfigurable region. As presented in Chapter 3, this approach allows to reduce the amount of required reconfigurable resources since not all functional blocks must then be instantiated in parallel.

In order to evaluate the potential area reduction enabled by DPR, the real resource usage of each module has been evaluated. Table 5.4 presents the results. They show that the largest module is the Microblaze processor core, which represents 12.38% of the complete device resources. Given that the static region only occupies a bit more than 25% of the whole device, it means that the actual required size is only 37.38% of the area of the LX-60 FPGA. The reconfiguration interface represents 0.5% of the device logic. Therefore, if all modules were instantiated in parallel in a static design, the total area consumption of the LX-60 device would be 51.73%. Using DPR allows therefore a 28% reduction of the required area compared to the conventional static implementation. The static design would therefore require a LX-40 FPGA whereas the DPR-enabled design

can fit in a smaller model, like the LX-25 for instance, while preserving the same level of functionality.

5.10 SUMMARY

DPR enables true hardware virtualization of FPGA. Libraries of optimized and specialized hardware implementations of functions can be created by storing partial bitstreams in external memories. Hardware virtualization provides new opportunities to reduce the power consumption of a signal processing application. When the application requires to execute a specific function, an optimized functional block can be dynamically instantiated in the design. Since those functional blocks are highly optimized, they provide a very high power efficiency.

In this chapter, the opportunity of reducing the power consumption of a signal processing application is evaluated. A comparison is made between the energy required to perform a kernel using a dynamically instantiated functional block and a software execution. The evaluation are realized by performing current measures on a development board. This methodology allows to have precise measures of the FPGA current during execution of the function. The results show that important gains can be obtained by exploiting optimized hardware functional blocks compared to a software execution. Moreover, the results also show that by using highly specialized implementations of kernels, the power consumption can be further reduced. A comparison is also realized with a software execution performed in a silicon processor. The evaluation confirms that the hardware implementations in the FPGA provide a better energy efficiency than the software execution.

DPR also introduces new sources of power consumption. The sources of power consumption are the transfers of the partial bitstreams from the memory, the configuration port controller, and the internal configuration machinery which has to rewrite a part of the configuration memory. An evaluation of the energy consumed by those sources has been performed using current measures and estimations. The results show that the memory transfers are the most important contributors to the reconfiguration process consumption.

The reconfiguration of the FPGA introduces a fixed energy cost for every reconfiguration. Evaluations performed in this chapter have shown that these costs represent many executions of the functional blocks. DPR is therefore more interesting in applications where the dynamically instantiated blocks are used for a long time, for instance when switching from a modulation to another for the time of a complete communication.

Since signal processing applications must comply to strict real-time constraints, fast memories and complex memory systems are needed to increase the bitstream transfer speed in order to perform reconfigurations sufficiently fast. Depending on the memory system configurations, the energy consumptions caused by partial bitstream transfers will vary. An estimation of the influence

of real-time constraints on the energy required to perform reconfiguration is realized. The results show that faster memory system can consume significantly more energy.

Finally, the opportunity to reduce the FPGA size by exploiting DPR is evaluated for a software-defined radio application based on the benchmarks considered in this chapter. Results show that significant area reductions can be obtained. Indeed, the DPR-enabled design requires 28% less resources than a static version where all functional blocks must be instantiated in parallel.

CHAPTER 6

METHODOLOGY FOR ARCHITECTURAL-LEVEL COMPARISON OF HMCP PROCESSOR CORES

6.1 INTRODUCTION

Homogeneous many-core platforms (HMCP) are used for DSP applications. Those platforms integrate up to several hundreds of processor cores [137, 138]. Those cores are typically RISC architectures, having single or multiple issues like VLIW. Thanks to their very high parallelism, HMCPs can reach very high throughputs. They also have very high programmability level, and good compilation support [139] compared to heterogeneous platforms or SIMD accelerators [140]. Those platforms are thus good candidates for SDA.

Due to the very limited power budget of SDA, HMCPs must have a very high power efficiency. This is a very strong design constraint, which has an influence on the choice of the processor core architecture. In order to choose this architecture, it is thus very important to be able to evaluate the impact of the core architecture on platform performance, including power.

On homogeneous many-core platforms, to get more performance, one can use more cores. However, adding more cores increases the total platform power and area, and this increase depends on the power and area of the used cores. Different cores lead to different platform configurations and performance. For instance, using simple cores provides a low IPC, but their low area and power consumption allow to put many of them on an HMCP within a given power and area budget. On the contrary, using more complex cores provides better IPC per

core, but also requires more area and power [141]. In this case, less cores can be used within the same budget. As those examples illustrate, there is a strong interaction between the performance of an HMCP and the IPC, power and area of its cores.

In order to choose the best architecture for the cores of an HMCP, besides the IPC, it is necessary to fully characterize the power and area of the candidates. To evaluate the IPC of a core, one can use a simulator, but to evaluate precisely the power and area, it is necessary to use real implementations, like an IP or even a chip. However, when comparing different processor architectures by using specific implementations, those differ on many aspects: ISA, technology, process flavor, hardware optimizations or compilation optimizations. Each of those aspects has an influence on the core performance. In order to isolate the impact of the core architecture on the platform performance, it is necessary to reduce the interferences introduced by a specific architecture implementation.

In order to realize precise performance comparisons at the architectural level, this chapter presents a methodology to compare processor core architectures. This methodology allows to remove the variations introduced by their specific implementations. It is based on the use of a common architecture template to build implementations of the compared cores. Together with those templates, specific optimizations are applied, when relevant. The relevance is evaluated for each core, by the real impact on its speed, area and power. The three core architectures have been implemented following this methodology. In order to validate the methodology, the performance of the three cores and the impacts of the applied optimizations are evaluated using benchmarks composed of DSP kernels from multimedia and telecommunication applications. Additionally to this validation, the performance of the three cores are also precisely compared.

This chapter is organized as follows: the next section presents existing many-core architectures and several works in the domain. The proposed methodology is described in the section 6.3. It discusses the criterions allowing to build comparable processor cores and realize a fair comparison of their performance. Section 6.4 and 6.5 describe the concept of architectural templates and discuss the need to apply specific optimizations on the compared cores. Three cores are implemented to validate the methodology. Their compositions and implementations are presented in the sections 6.6 and 6.7. Section 6.8 presents results validating the methodology. The optimizations applied to the compared cores are described, and the results illustrating their impacts on the core performance are discussed.

6.2 RELATED WORK

There are numerous existing HMCPs, some are also called massively parallel processor arrays (MPPA). However, there is no work that tries to justify which processor core architecture is the best for those platforms. PicoArray from picoChip [137] uses more than 300 3-issue VLIW cores with a 16-bit datapath. The Tile64 platform from Tilera, based on the RAW research platform [31], has 64 32-bit 3-issue VLIW cores. Their larger platform, the Tile-Gx, uses 100 cores. Ambric platforms [138] have 336 32-bit single-issue RISC DSP cores. Among research platforms, AsAP is also an HMCP, with 167 single-issue cores [34]. The many-core WPPA platform [142] has a configurable number of small VLIW cores.

Several works propose design space exploration frameworks for multi-core platforms. Those frameworks help to quickly identify several platform configurations which are potential solutions for a group of applications [143, 144]. They are generally based on fast simulators and performance models of processor architectures. This approach speeds up the exploration but reduces the precision of the results. The calibration of the models used in those design space exploration frameworks is performed with a limited set of real implementations. This work provides number to calibrate those models.

Other works propose architectural description languages (ADL) that enable high-level descriptions of a processor architecture. This approach allows to build application-specific instruction-set processors (ASIPs) for dedicated workloads. On the basis of this description, the associated tools can automatically generate a simulator, a toolchain and RTL code [145, 146, 147]. The use of ADLs allows to easily evaluate the performance of several architectural solutions, by describing the different architectures and simulating the applications using the generated tools. Nevertheless, the performance obtained with the automatic optimizations performed by tools used for ASIPs are limited, and does not allow to take full advantages of all the architectural features. Some work have already highlighted performance differences between optimized and automatically generated ASIP cores [148].

6.3 METHODOLOGY FOR ARCHITECTURAL COMPARISON

Precise processor comparisons are performed using implementations, by comparing chips, IPs, or results from datasheets. Those implementations have different ISAs, technology nodes, process flavors, hardware optimizations or software compilation optimizations. Those implementation specificities introduce interferences on the performance of compared cores. These interferences are caused by different sources of variations, they are listed in Table 6.1. Those variations make it very difficult to evaluate the influence of the core architecture on the performance of implementations. To build processor cores comparable at the architectural level, those variations must be removed.

Variation source	Causes
Physical	Technology, process flavor, development flow, supply voltage.
Microarchitecture	Core internal organization, function implementations, available functional units, ISA, memory blocks.
Software	Code scheduling, benchmarks.

Table 6.1. List of variations between processor implementations

When comparing processor architectures, it is important to make sure that the architecture comparisons are fair. To evaluate them correctly, each core must be able to fully exploit the benefits of its architecture. It is thus important to guarantee that the processor implementations and the code they execute are fair regarding the evaluations of architectural features.

In order to build comparable processors and to realize fair comparisons, the methodology proposed in this work consists in complying with the following criterions:

1. **the cores are implemented in the same technology**: it allows the cores to benefit from the same timing and power performance and to operate in the same conditions.

2. **they are implemented using the same development flow**: the designs must be synthesized, placed and routed with the same tools and the same constraints. Thanks to this, they take benefit of the same automatic optimizations.

3. **they use the same memories**: using the same memory blocks give them the same performance. Memory netlists are generated with the same memory compiler.

4. **they use the same ISA**: the ISA has an influence on the complexity of the decoding circuits, on kernel sizes, and on the instruction memory access count. The compared processors have the same DSP instructions to optimize kernel execution.

5. **they use a maximum of resources defined with the same code**: identical functional blocks must be defined with the same HDL code or the same placed and routed netlists (e.g.: ALUs, instruction decoder, memory ports).

6. **the code is optimized by hand:** it allows to take the most out each architecture instance, which allows in turn to evaluate their specific performance. Compilation by hand prevents the code to depend on specific compiler optimizations.

7. **they are well balanced**: for instance, the set of functional units of the cores must be chosen in order to maximize both IPC and resource usage. It allows a specific architecture to provide a representative amount of parallelism.

8. **specific optimizations are applied on the core when relevant:** it allows to ensure that no architecture implementations suffer from detrimental overheads which introduce biases in the evaluation of its performance.

Criterions 1-3 allow to remove the physical variations in the implementations. Criterions 4 & 5 allow to remove the variations of the microarchitecture. To comply to those latter criterions, the compared processor cores are implemented by using common architectural templates. Those architectural templates are explained in the next section. Criterions 6-8 allow to preserve the specificities of the architectures without bias. Optimizations are discussed in section 6.5. They also provide fair comparisons between the implementations. Criterion 6 allows to remove software variations.

6.4 ARCHITECTURE TEMPLATE

The microarchitecture of a core defines its internal organization and implementation. It has a strong impact on its performance. Some elements and characteristics of the microarchitecture can be common to different architectures, some others, on the contrary, are specific. When comparing architectures, it is very important to identify common microarchitectural characteristics and impose a comparable implementation between them. This approach allows to reduce the variations in microarchitecture implementations strictly to the architecture model differences. To illustrate this, Table 6.2 identifies common components and specificities in the microarchitectures of a family of N-issue RISC processors. This family of processor is the one compared in this work.

Common components	**Specificities**
Pipeline stage count & function	Execution issue count
Instruction fetcher	RF size
Instruction decoder	RF port count
Register file	Bypass input count
Bypass network	ALU count
Functional units	Memory port count

Table 6.2. List of common components and specificities in N-issue RISC processors

In order to build comparable implementations that reduce variations of the microarchitecture, a common architectural template is used to build the compared architectures. Using a template guarantees to get uniform implementations for the common functionalities between the different architectures and provides shared generic implementations for the functionalities of a core. It defines a common organization for all the implementations and their evolutions with the issue count increase.

The template defines:

- pipeline stage count and composition;
- the functional units available for all the compared architectures;
- the presence of specific functionalities, like the bypass network or the interlock signals preventing write hazards;
- the evolution of the different modules when the issue count increases (e.g.: the read/write port count of the register files, the number of bypass network inputs).

Figure 6.1 illustrates the concept of architectural templates for a family of N-issue RISC processors. In this template, multiple-issue cores correspond to VLIW processors. This template is the one used for the architectures compared in this chapter.

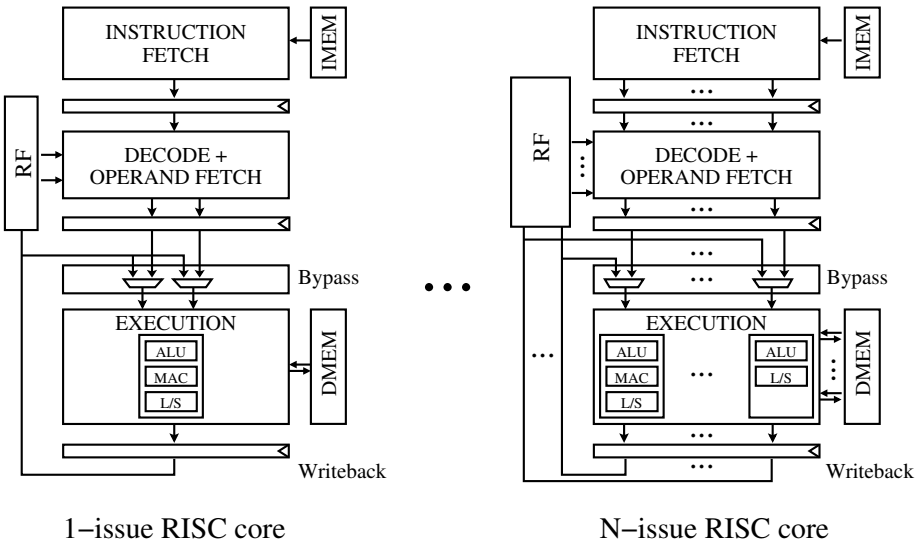

Fig. 6.1. Architectural template for 1-issue to N-issue RISC cores

6.5 MICROARCHITECTURE OPTIMIZATIONS

When the number of issues increases, the complexity and the timing of some functional modules degrades. It is notably the case of the following elements:

- **the register file** presents a significant overhead when the read/write port count increases;
- **the bypass network:** the data source selector of the bypass network, as well as their control circuits do not scale well with the increase in inputs. It causes additional delays which worsen the timing of the functional units.
- **interlock signals:** those signals require long control lines across the pipeline stages. They can introduce additional delays.

For most of those drawbacks, numerous contributions have already been proposed to mitigate their bottlenecks. In order to realize a fair comparison, it is therefore necessary to implement them.

The development tools can also introduce suboptimal implementations. Some automatic optimizations that are realized by the synthesis tools do not benefit to every functional modules. For instance, automatic clock gating can cause the insertion of too many clock gating cells. This can cause an overhead in area and a degradation of the timing in some circuits. Those degradations can eventually cause an increase in power consumption. To fix those overheads, some optimizations need to be realized manually.

To enable fair comparisons, it is thus necessary to realize specific optimizations that allow to reduce or remove the overheads caused by the scaling of the issue count and the tools. Consequently, the methodology proposed in this work suggests to identify the functional modules that can benefit from those specific optimizations, and apply them when they are relevant, depending on their impacts on the performance and the required precision. Automatic optimizations realized by the tools must be monitored, and be replaced by more efficient manual optimizations when necessary.

6.6 DESIGNED PROCESSOR CORES

Three processor cores have been implemented by following the methodology proposed in this work. Their implementations have been realized using the template presented in Figure 6.1. Each core uses an identical standard DSP instruction set. The first architecture is a scalar single-issue RISC processor, called $DSP1$. The two other implemented architectures are VLIW processors, with respectively 3 and 5 issues, called $VLIW3$ and $VLIW5$.

The chosen cores cover the range of candidate architectures for HMCPs. Indeed, those platforms reach execution parallelism by dividing a streaming application in small independent actors, and mapping them on many separate cores.

Combined together, the high number of available cores allows to reach a high computational throughput. Consequently, HMCPs do not use high-IPC processor cores like in multi-core platforms.

The three architectures use the same datapath components:

1. **ALU$_{INT}$:** 32-bit integer computation unit. This unit performs basic integer arithmetic operations, logical operations and comparisons. It supports bit-level operations, like byte swapping or bit rotations. Those operations are performed in one cycle.

2. **ALU$_{SIMD}$:** SIMD operation unit. It performs 2×16-bit and 4×8-bit operations like absolute differences, scalar products, etc. Those operations are performed in one or two cycles. This ALU improves the exploitation of the data bus width providing operands to the ALUs, which allows to increase the platform throughput.

3. **MAC:** multiplication-accumulation unit, which can perform a double 16-bit multiplication in two cycles, and a multiplication-accumulation in three cycles. This unit allows to implement efficiently filtering operations that are numerous in telecommunication applications.

The composition of the three processors, with the description of their units, the size of their register files, their bypass networks and their memory ports is summarized in Table 6.3. The set of functional units selected for each core instance has been chosen to balance them correctly, as explained in section 6.3. The number of memory ports and the selected ALUs allow to maximize the use of the available resources while also maximizing the IPC.

$DSP1$	$VLIW3$	$VLIW5$
1 issue	3 issues	5 issues
32×32-bit reg.	64×32-bit reg.	64×32-bit reg.
1×ALU$_{INT}$	3×ALU$_{INT}$	5×ALU$_{INT}$
1×ALU$_{SIMD}$	2×ALU$_{SIMD}$	3×ALU$_{SIMD}$
1×MAC	1×MAC	2×MAC
1 memory port	2 memory ports	2 memory ports
Simple bypass	7-input bypass	11-input bypass

Table 6.3. Composition of the implemented processor cores

The three processor cores are accompanied by instruction memories and scratchpads for their data. The $DSP1$ processor has a 4KB instruction memory, and a 8KB scratchpad for its data. The VLIW processors have larger instruction memories since their code sizes increase due to the use of unrolling and software pipelining techniques. Those memories have been precisely dimensioned following their code growths. The growth factors have been estimated by comparing

the size of the benchmark codes of the VLIW cores with the codes of the $DSP1$ core. For the $VLIW3$, this growth factor is $2\times$, and $2.7\times$ for the $VLIW5$. Both VLIW processors have the same total amount of data memory as the $DSP1$, distributed in two 4KB scratchpads. The cores have all the same amount of data memory since their size is dictated by the size of the data elements processed by the algorithms. They cannot be modified for the different architectures.

6.7 PROCESSOR CORE IMPLEMENTATIONS

The three processors have been coded using verilog HDL, then synthesized, placed and routed using digital standard cell libraries of a low power SVT CMOS 65nm technology from STMicroelectronics. The memory blocks have been obtained using a memory compiler from this technology. This process flavor has a high threshold voltage that allows to strongly reduce leakages. Because of this, the power consumption of the cores is dominated by dynamic power and there is no bias introduced by leakage power. Table 6.4 compares the dynamic and leakage power of the three implemented cores. The leakage power is two orders of magnitude lower than the dynamic power. However, the high threshold voltage also limits the transistor speed, which limits the operating frequency of the designs realized in this technology.

	$DSP1$	$VLIW3$	$VLIW5$
Dynamic power at 100MHz [W]	2.69e-3	8.72e-3	15.40e-3
Leakage power [W]	6.58e-6	7.51e-6	20.94e-6

Table 6.4. Dynamic and leakage power of the cores on the 802.11a benchmark.

Figures 6.2 and 6.3 respectively show the evolution of the netlist areas and consumed energies with respect to the frequency constraint imposed to the synthesis and physical implementation tools. The evaluated energy consumption corresponds to the execution of a benchmark performing the modulation of a frame of the 802.11a wireless telecommunication standard [149].

There is a strong degradation of the area and energies around 400 and 500MHz constraints. Those degradations are due to the oversizing of the circuits in order to reach the imposed constraints. The netlists that are retained for the rest of this work are highlighted in the figures with white markers. They correspond to configurations allowing to reach the highest frequencies without having strong interferences from the constraints on the area and energy consumption. The retained netlists for the $DSP1$, $VLIW3$ and $VLIW5$ are generated respectively with 500MHz, 400MHz, and 400MHz constraints. The $DSP1$ processor can thus operate at a frequency 25% higher than the VLIW. The simplicity of its circuit, mainly the register file and the bypass network, allows it to reach better timing performance.

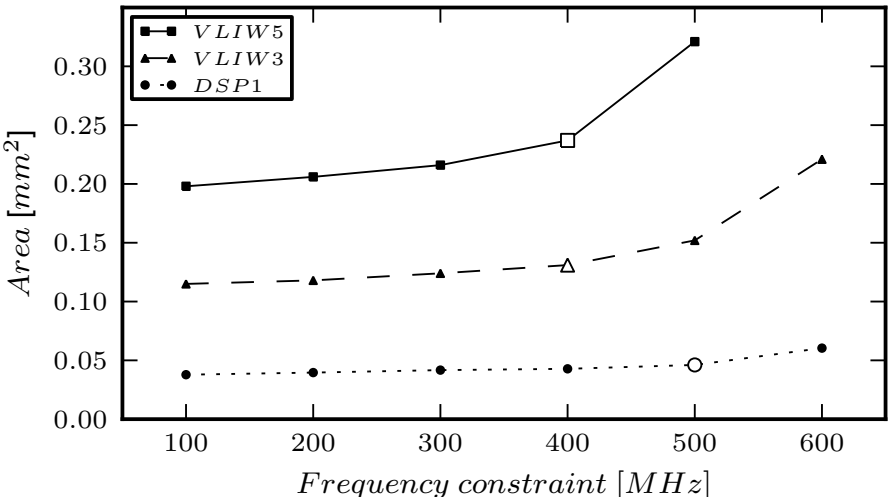

Fig. 6.2. Processor core areas for each frequency constraint applied during place and route. White markers represent the selected netlists for each architecture used in the following experiments.

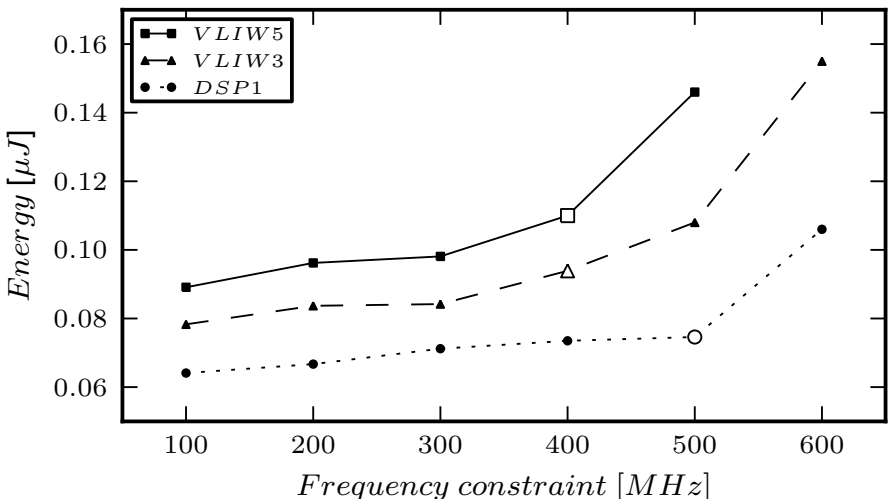

Fig. 6.3. Processor core energies on the 802.11a benchmark for each frequency constraint applied during place and route. White markers represent the selected netlists for each architecture used in the following experiments.

6.8 METHODOLOGY VALIDATION

The performance of the three processors have been evaluated on a set of 6 DSP benchmarks. Those benchmarks are kernels from telecommunication and image processing applications. They represent the most part of the workload for those applications. As explained in the section 6.3, benchmarks have been optimized by hand in order to maximize usage of all the resources available in the cores as well as the dedicated SIMD and DSP instructions. Performance results obtained for the three architectures are summarized in Table 6.5. Results show that the three cores reach very high IPCs compared with their issue widths. This shows that the proposed methodology allows to reach near-optimal IPC performance for each evaluated architectures.

	$DSP1$		$VLIW3$			$VLIW5$		
Name	Cycles	IPC	Cycles	Speedup	IPC	Cycles	Speedup	IPC
fir32	9543	0.99	3398	2.81	2.69	2249	4.24	4.30
fft64	5430	0.93	2133	2.55	2.58	1438	3.78	4.22
d8psk	13130	0.93	5403	2.43	2.44	3597	3.65	3.82
802.11a	27760	0.95	10766	2.58	2.55	7148	3.88	4.06
sad	346	1.00	120	2.88	2.88	76	4.55	4.59
dct	770	0.93	370	2.08	1.89	193	3.99	3.66
Mean	-	0.96	-	2.55	2.51	-	4.02	4.11

Table 6.5. Performance results on kernel benchmarks for the $DSP1$, $VLIW3$, and $VLIW5$ platforms.

The power consumptions of the cores have been evaluated by extracting switching activities from post-layout netlist simulations on the benchmarks. Figure 6.4 shows the breakdown of the dynamic power for the modules composing the three cores, and Table 6.6 shows the corresponding absolute values. It shows that most of the power is consumed in the execution stage, this confirms that the cores are correctly optimized since most of the energy is actually consumed to perform useful work.

Several specific optimizations have been applied on the three implemented cores in order to reduce their disadvantages, as suggested by the methodology. Clock gating has been carefully applied by hand on all designs. All those optimizations allow to reduce the total power consumption of the $VLIW5$ register file to only 27% of the core power. Bypass controls of the VLIW processors are precomputed during the decode stage in order to reduce the critical paths of the input source selection circuits. Several optimizations have been applied on register files in order to reduce their power consumptions [150, 151]. First, registers have been partitioned in several groups and data gating cells have been placed on write port data signals for each partition. This technique allows to reduce the

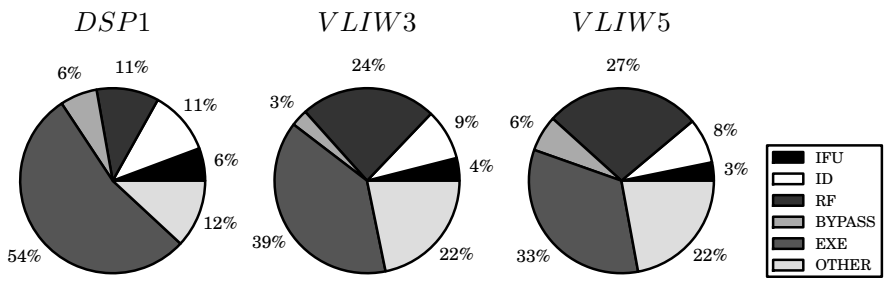

Fig. 6.4. Dynamic power breakdown for the three implemented cores. The "OTHER" category corresponds to top-level circuits, its power is dominated by the clock tree.

Power [mW]	$DSP1$	$VLIW3$	$VLIW5$
IFU	0.15	0.30	0.46
ID	0.28	0.70	1.15
RF	0.28	1.82	3.95
BYPASS	0.15	0.22	0.91
EXE	1.38	2.99	4.83
OTHER	0.31	1.69	3.21
Total	2.55	7.72	14.5

Table 6.6. Dynamic power consumption of the three implemented processor cores.

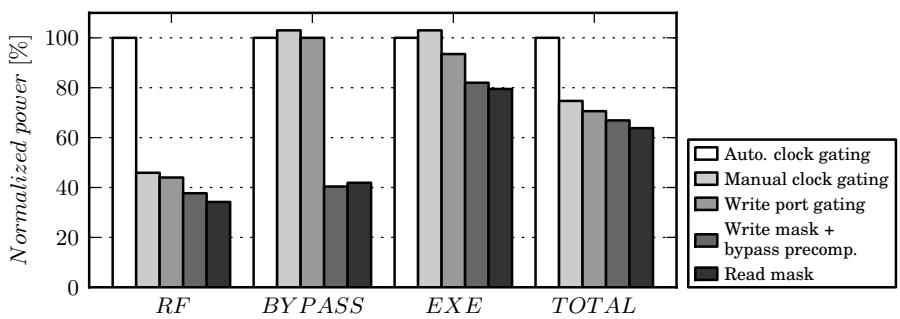

Fig. 6.5. Cumulative power consumption reduction in the $VLIW5$ processor due to architectural optimizations.

fanout of these signals. Second, unnecessary read and write operations in the register file are masked. The unnecessary read operations correspond to values that are provided by the bypass network to the datapaths, and unnecessary write operations correspond to register values that are replaced by new ones before they are read with the data produced in other pipeline stages.

In order to validate the interest of those optimizations, their impacts have been evaluated by measuring the power consumption reduction of different optimized circuits. Figure 6.5 illustrates the cumulative reductions for three circuits of the $VLIW5$ core, and the whole core. In the register files, the cumulated techniques allow to reduce the energy consumption by 3× compared to a naive implementation without aggressive optimizations and using automatic clock gating cell insertion. Bypass control signal precomputation allows to reduce significantly the power consumption of this stage, and of the execution (EXE) stage. This is explained by the improvements of the timing of the functional units, which allows the synthesis tools to use smaller logic gates consuming less power. After optimizations, the power consumption of the bypass stage and the execution stage is reduced by respectively 60% and 20%.

For the $VLIW3$ architecture, the same optimizations have been performed as for the $VLIW5$. Detailed results are however not available for this core since it has been modeled after the optimized $VLIW5$. The $DSP1$ core has also received the same optimizations, except for the bypass stage which was too small to benefit from them. For this core, the reduction on total dynamic power consumption is 8%, whereas it is 40% for the $VLIW5$. Optimizations are indeed more efficient on $VLIW$ processors since those architectures require larger register files and bypass stages which introduce inefficiencies. However, in the $DSP1$, the RF stage optimizations still reduce its dynamic power consumption by 22%, which is a significant power reduction.

The results show that significant power reductions can be obtained by performing manual optimizations on the designs. The optimizations are particularly effective on large cores like VLIW processors. It validates therefore the need to operate optimizations manually in order to realize a fair comparison between different architectures.

6.9 SUMMARY

This chapter presents a methodology that allows to realize precise comparisons of IPC, power and area for different processor architectures. Using this methodology, it is possible to analyze the impact of core architecture on the performance of an HMCP. The methodology is based on the use of a common architectural template, together with the application of specific optimizations when relevant for the core performance. A validation of the methodology is performed through the implementation of three RISC cores: single-issue RISC core, and two VLIW processors with 3 and 5 issues. The cores are implemented in low power SVT 65nm from STMicroelectronics. Their performance are evaluated on 6 DSP ker-

nels. Results show that the methodology allows to build cores that reach near-optimal IPC performance for each evaluated architecture. They also show that optimizations can significantly impact performance. Therefore, it confirms that specific optimizations are required to realize fair comparisons.

CHAPTER 7

PROCESSOR ARCHITECTURE IMPACT ON HOMOGENEOUS MANY-CORE PLATFORMS

7.1 INTRODUCTION

In the previous chapter, a methodology has been introduced to enable precise processor core comparisons at architectural level. Using this methodology, three RISC cores have been implemented: a single-issue RISC core, a 3-issue VLIW core, and a 5-issue VLIW core. These cores cover the range of candidate architectures for HMCPs targeting DSP applications.

In this chapter, those cores are used to evaluate the core architecture impact on the power efficiency of HMCPs. In order to evaluate this impact, power efficiency comparisons are performed for three HMCPs based on the implemented core architectures. The comparisons are realized on the kernels of a telecommunication software-defined application. For each architecture, the platform power, throughput and area values are estimated on basis of results obtained on a single core. In order to enable those estimations, relations between the cores and the platform power, throughput and area are detailed.

The chapter is organized as follows: the first section describes the evaluation methodology. The second section discusses the power of HMCPs. The total platform power is separated in the core power consumption and communication network power consumption. Section 7.4 discusses the evaluation of inter-core communication impacts. In the following section, the power, area and performance of the implemented cores are compared. This comparison allows to identify the

properties of each core. Section 7.6 explains how the voltage and frequency scaling ratios have been estimated. Section 7.7 introduces the relations between the platform cores and the platform power, area and throughput. In section 7.8, the impact of the core architecture is evaluated for the three considered platforms. Finally, the impact of the communication structure on the results is studied in section 7.9.

7.2 EVALUATION METHODOLOGY

In this chapter, an evaluation of the core architecture impact on HMCP power efficiency is realized. In order to perform this evaluation, power, area, and application throughput of three HMCPs are compared. Those three platforms use different core architectures. The cores used for the comparison correspond to the cores implemented in the previous chapter. The methodology used to implement them allows to realize precise comparisons at the architectural level by removing interferences due to specific implementations. The comparisons allow therefore to evaluate the architecture impact on the platforms.

To evaluate the platforms, power, throughput and area values of the platform are estimated on the basis of numbers obtained for the execution on a single core. In order to enable those estimations, relations linking the power, area, and throughput of an HMCP to its cores are identified. Those estimations also take into account voltage and frequency scalings.

Platform performance are evaluated on a telecommunication SDA. This application is an OFDM symbol modulation for the 802.11a wireless standard [149]. The application is composed of 4 kernels: a symbol mapping operation, a FIR filter, an FFT and an interpolator. In the 802.11a standard, the maximal throughput of a channel is 54Mb/s. However, in this work, throughputs up to 100Mb/s are considered to model high workloads.

The 802.11a benchmark is a typical telecommunication application targeted by most SDA platforms. It has the typical high data-level parallelism of DSP applications, and it is therefore highly representative of that kind of workload. Moreover, the processor cores show consistent results on the 802.11a benchmark and the mean core performance, which are also evaluated on kernels from other application domains than telecommunication. This is for instance the case of the *sad* and *dct* kernels which comes from video and image processing applications.

However, different application types use different operations, and have different bandwidth requirements and memory access patterns. Studying precisely the application impact requires therefore to compare the platform performance on several complete application implementations. To perform these comparisons, new kernels are required, and additional ALU operations have to be implemented. Moreover, those additional kernels have to be compiled by hand to avoid compiler interferences. Performing these tasks requires a significant work,

and it has not been done in this thesis. Those aspects can however be studied in future works.

7.3 HMCP POWER CONSUMPTION

In HMCP, cores communicate with each other through a dedicated communication network that can be buses, FIFOs, or a Network-on-Chip [152, 153]. Homogeneous many-core platform power is therefore composed of two main contributions: the computation power caused by application execution on the cores, and the communication network:

$$P_{tot} = P_{comp} + P_{comm} \tag{7.1}$$

In this equation, P_{comp} represents the computation power which is the power needed to execute the benchmark instructions, and P_{comm} is the power related to the communication network. P_{comp} depends on the core architecture, the executed instruction count, memory accesses, etc. P_{comm} depends on the communication network architecture, the network topology, the traffic distribution, etc.

In equation 7.1, P_{comp} depends on the work that the cores must perform. For each core, this work can be divided in two contributions:

$$WL_{core} = WL_{single\,core} + WL_{init\,comm} \tag{7.2}$$

In this equation, WL_{core} represents the complete workload of a core. $WL_{single\,core}$ is the task workload when executed on a single-core processor. $WL_{init\,comm}$ is the additional workload caused by the initiation of inter-core communications, e.g. DMA transfers, packet sendings, FIFO write. Due to the implementations of the communication network as an IO peripheral, instructions executed to initiate inter-core communications are similar to load/store instructions. Once the communication is initialized by the core, the remaining of the communication is performed by the communication network and its power consumption is modeled by P_{comm}.

As Equation 7.1 shows, contributions from the computation and inter-core communications to the total power consumption can be evaluated separately. By taking $WL_{init\,comm}$ into account, it is possible to make the two contributions independent.

7.4 EVALUATION OF CORE COMMUNICATION OVERHEAD

In order to model correctly core workloads in an HMCP, the overhead of communication initializations must be taken into account. In the benchmark used in this chapter, kernel communications are performed by accesses to data buffers

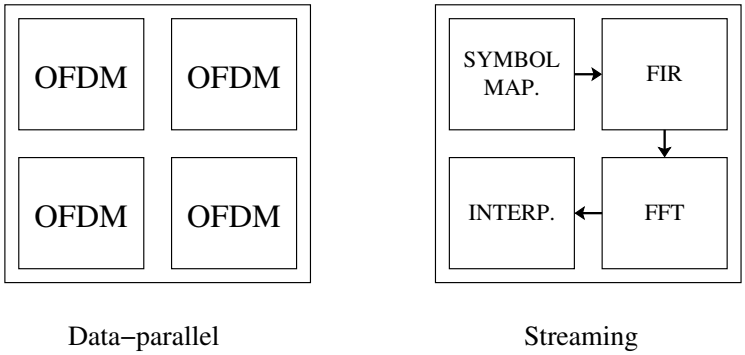

Fig. 7.1. Illustration of two different mappings of an OFDM symbol modulation in an HMCP

mapped in memories. Since those operations are equivalent to communication initialization operations, this approach allows to model the overhead due to communication initializations between cores, and therefore to estimate correctly $WL_{single\,core}$, and hence P_{comp}.

When mapping an application on an HMCP, two different kind of mappings can be considered. Those two approaches are illustrated in Figure 7.1. The first mapping exploits application-level data parallelism by performing the whole task on a single core, and replicating the task on several cores. In this case, all task kernels are executed successively in the same core. For instance, for an OFDM symbol modulation, this approach consists in mapping a whole symbol modulation on each core. The second approach consists in mapping each kernel on separated cores. The cores are then chained together to execute the application in streaming, each core feeding the next with the processed application data. If the parallelism is too low, kernels can also be divided.

In this chapter, platform performance are evaluated using numbers obtained for the execution of a benchmark on a single core. This kind of execution corresponds directly to an application mapping exploiting data parallelism. The evaluation can therefore precisely model this mapping. The results obtained with the particular benchmark used here can however also be used to estimate the performance of the streaming mapping. Indeed, the benchmarks use memory buffers to store intermediate results between kernels, and memory accesses have the same costs as communication initializations. This corresponds to the mapping of one kernel per core. When a kernel is divided to map its execution on several cores for load balancing issues, the communications stay low and cause little initialization overhead, at least for filtering and interpolation. For instance, if the 33-tap FIRs used in the interpolator are split in two 17-tap and 16-tap FIRs, the exchanged data is only the content of the accumulator. The approach followed in this work allows therefore to estimate precisely the HMCP computation power with a sufficient independence from the many-core mapping.

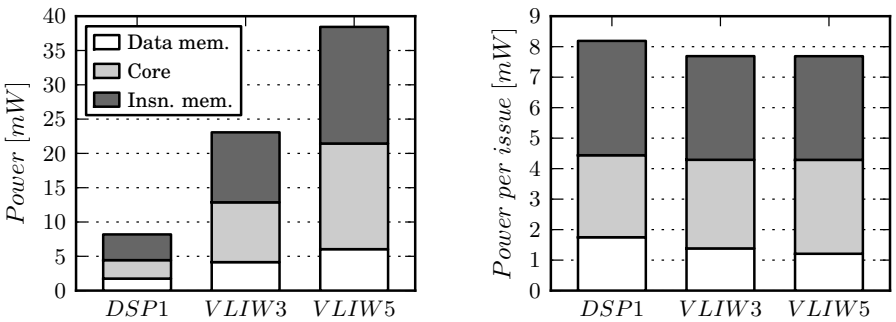

Fig. 7.2. Processor and memories mean powers (left) and mean powers per issue (right) for each architecture.

7.5 PROCESSOR CORE COMPARISONS

This section presents the performance of the three cores used in this chapter to evaluate the platform powers, areas and throughputs. The cores have been evaluated on 6 DSP benchmarks. Table 7.1 presents the consumed energy and the IPC obtained for the benchmark executions. The results show that the $VLIW3$ and $VLIW5$ processors can provide mean speedups of respectively 2.55 and 4.02 compared to the $DSP1$ processor.

Figure 7.2 shows the breakdown of the total mean power consumption of the three cores and their memories. Being the simplest core, $DSP1$ dissipates the less total power. Figure 7.2 also illustrates the mean power consumption normalized by the execution issue count of the cores. One can see that the normalized power is roughly identical between all implemented cores, and that the $DSP1$ is actually the core consuming the more power per issue. Note that this metric does not take the IPC into account.

The energy consumed by the three cores on the benchmarks is presented in Table 7.1. For most benchmarks, the $DSP1$ consumes the less energy. The only exception is for the *sad* benchmark where the very high speedup on the two other cores allow to compensate their higher total power. The $VLIW3$ and $VLIW5$ cores have mean energy consumptions which are respectively 9% and 20% higher than $DSP1$ energy consumption. The loss of power efficiency of the VLIW processors is explained by the higher complexity of some of their circuits like the bypass and the register file, for which each access consumes more power when the number of inputs is higher. Moreover, in those architectures, the code is filled with more NOP instructions, which also causes a power overhead. Those results indicate that using more complex cores introduces an overhead in energy consumption. However, even if the $DSP1$ has a better power efficiency, the $VLIW5$ core still provides a 4× speedup with an overhead of only 20% in energy.

	$DSP1$			$VLIW3$					$VLIW5$				
Name	Cycles	IPC	Energy [J]	Cycles	Speedup	IPC	Energy [J]	Δ_E [%]	Cycles	Speedup	IPC	Energy [J]	Δ_E [%]
fir32	9543	0.99	8.18e-7	3398	2.81	2.69	8.39e-7	3%	2249	4.24	4.30	8.94e-7	9%
fft64	5430	0.93	3.99e-7	2133	2.55	2.58	4.45e-7	12%	1438	3.78	4.22	5.49e-7	37%
d8psk	13130	0.93	1.08e-6	5403	2.43	2.44	1.23e-6	14%	3597	3.65	3.82	1.35e-6	25%
802.11a	27760	0.95	2.27e-6	10766	2.58	2.55	2.48e-6	9%	7148	3.88	4.06	2.75e-6	21%
sad	346	1.00	2.79e-8	120	2.88	2.88	2.54e-8	-9%	76	4.55	4.59	2.68e-8	-4%
dct	770	0.93	5.87e-8	370	2.08	1.89	7.50e-8	28%	193	3.99	3.66	7.76e-8	32%
Mean	-	0.96	-	-	2.55	2.51	-	9%	-	4.02	4.11	-	20%

Table 7.1. Performance results on kernel benchmarks for the $DSP1$, $VLIW3$, and $VLIW5$ platforms.

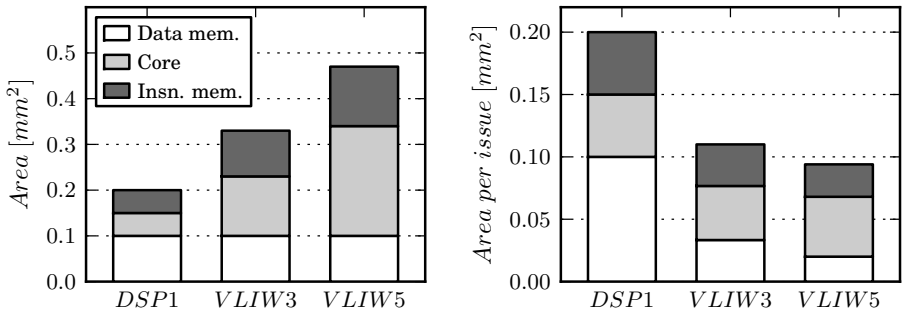

Fig. 7.3. Processor and memories areas (left) and areas normalized per issue count (right) for each architecture.

Figure 7.3 compares the areas of the processors with their memories. The total areas of the $VLIW3$ and $VLIW5$ processors with their memories are respectively 1.7× and 2.4× larger than the area of the $DSP1$ core with its memories. Figure 7.3 also shows the total areas divided by the number of issues. The $DSP1$ has a higher area per issue ratio since it must have all datapath units and data memories but has only one issue. Cores with several issues thus make a better use of their area. For instance, $VLIW5$ provides a 4× speedup for only 2.4×more area.

7.6 VOLTAGE AND FREQUENCY SCALING

In CMOS technology, the power of a circuit is expressed by the following relation:

$$P_{tot} = P_{dyn} + P_{stat} \tag{7.3}$$

$$= \frac{1}{2} \times \alpha \times C \times V^2 \times f + I_{leak} \times V \tag{7.4}$$

where α is the switching activity, C the total capacity of the design, V the voltage and f the operating frequency. As explained in section 6.7, the technology that is used in this work is a 65nm low power SVT technology. This technology has a static power that is several orders of magnitude smaller than the dynamic power. In these conditions:

$$P_{tot} \cong P_{dyn} = \frac{1}{2} \times \alpha \times C \times V^2 \times f \tag{7.5}$$

The frequency is proportional to the dynamic current of the transistors. This dynamic current is proportional to the voltage difference between V and the threshold voltage V_{th}. The frequency can thus be described like this:

$$f = k_f \times (V - V_{th}) \tag{7.6}$$

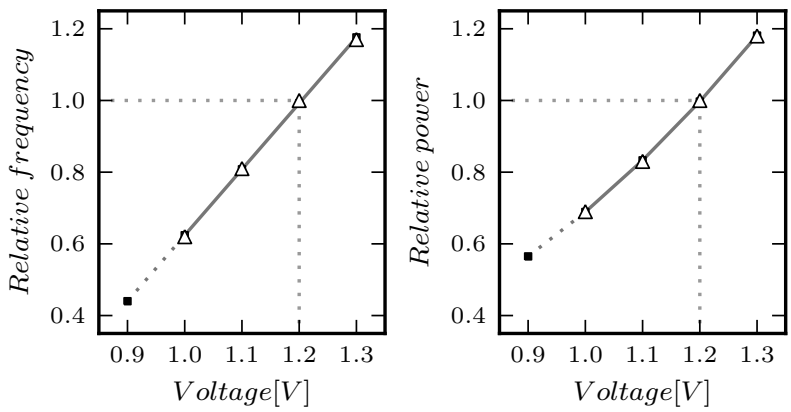

Fig. 7.4. Relative impact of voltage scaling on power and frequency. Results are obtained from measures on placed and routed designs. Values at 0.9V are obtained using polynomial extrapolation.

where k_f is a constant. It is thus possible to express the power of the circuit with the frequency only:

$$P_{tot} \cong \frac{1}{2} \times \alpha \times k_f \times C \times f \times (\frac{f}{k_f} + V_{th})^2 \tag{7.7}$$

This relation is thus a third-degree relation of the power depending on the frequency f. When V is large compared to V_{th}, this relation can be approximated by the following equation:

$$P_{tot} \cong \frac{1}{2} \times \frac{\alpha \times C}{k_f} \times f^3 \tag{7.8}$$

In this work, as V_{th} is situated around 0.4V, V is not significantly greater than V_{th}. This approximation has therefore not been used for the evaluation. It is however used in the following equations in order to simplify their understanding.

The digital cell libraries used in this work are characterized for four working points: 1.0V, 1.1V, 1.2V and 1.3V. Power and frequency evaluations have been performed on the designs at those points. Their relative evolutions with the voltage is shown in Figure 7.4. The reference voltage is 1.2V, which is the reference voltage of the technology. In Figure 7.4, one can see that the frequency is changing linearly with the voltage, and that the power is changing with the square of the voltage. Those measures comply with equations 7.6 and 7.5. The lowest voltage considered in this work is 0.9V. The scaling factor values at this voltage have been extrapolated on the basis of polynomials fitted on the measures, which is in agreement with Equation 7.7. Those extrapolations are illustrated with dashed lines in Figure 7.4.

7.7 HMCP POWER, AREA AND THROUGHPUT

This section presents the relations between the platform power, throughput and area and its cores. Using those relations, estimations of the platform power, throughput and area of an HMCP can be realized on basis of the performance of a single core.

7.7.1 Power and throughput

For an HMCP, the computation power depends on the number n of active cores:

$$P_{comp} = n \times P_{core} \tag{7.9}$$

Using the relation 7.8, it is possible to express the power of the cores depending on the frequency:

$$P_{comp} \cong n \times \frac{1}{2} \times \frac{\alpha \times C}{k_f} \times f^3 \tag{7.10}$$

The operating frequency is linked with the throughput by the following relation:

$$T_p = n \times f \times b_{cy} \tag{7.11}$$

where b_{cy} represents the number of bits encoded per execution cycle. Using the relations 7.10 and 7.11, it is then possible to express the power depending on the active core count and the throughput:

$$P_{comp} \cong k_p \times \frac{T_p^3}{n^2} \tag{7.12}$$

where k_p is a constant. This relation allows to express the computation power of an HMCP depending on the imposed throughput and the number of cores. It is illustrated in Figure 7.5 for the $DSP1$ platform.

Figure 7.6 illustrates the evolution of the power of an HMCP composed of $DSP1$ cores for a given throughput range. For each configuration of n active cores, illustrated by the dashed curves in the figure, the power is constant as long as the cores can work at the minimum voltage and frequency. When this configuration cannot sustain the throughput anymore, the frequency and the voltage must be increased. The power then raises with the cube of the throughput, according to the relation 7.12, and eventually reaches its maximum when the supply voltage is set to its maximal value at 1.3V.

The platform reaches its best power efficiency when the throughput constraint corresponds exactly to the computation capacity that the active cores can offer at the lowest voltage. Those points only depend on the number of active cores, and they are all placed on the same line. This line is illustrated in the Figure 7.6

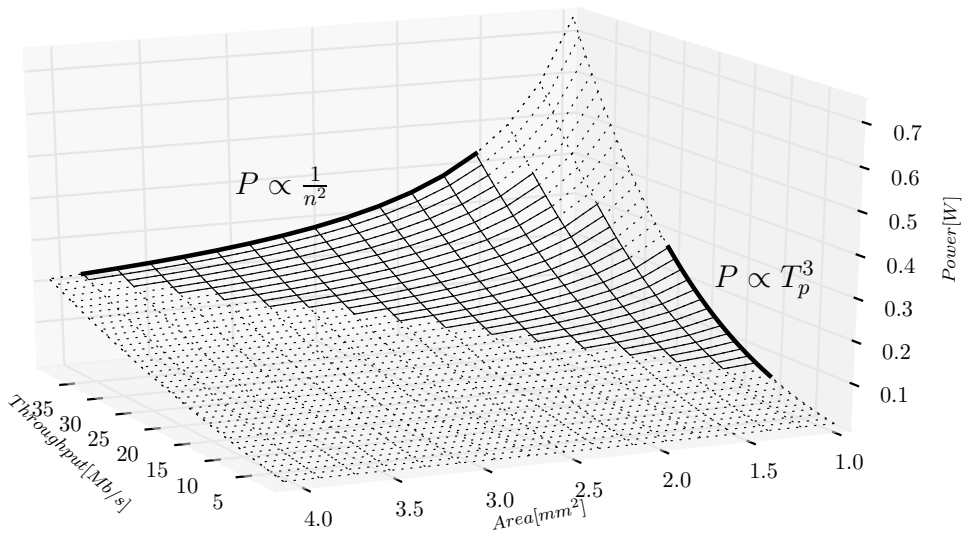

Fig. 7.5. $DSP1$ performance representation in the three axes: power, throughput, and area. The surface in plain black represents the reachable throughput values for a set of active cores on the available voltage range.

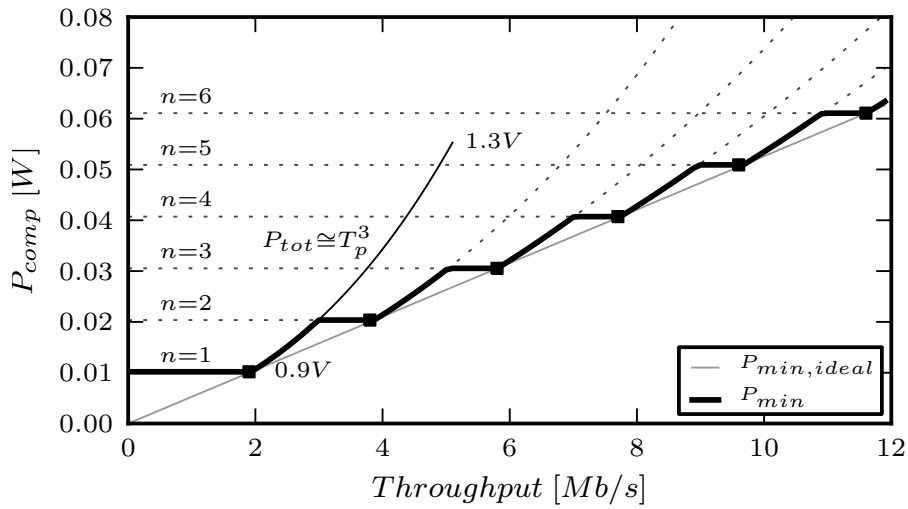

Fig. 7.6. Power vs. throughput for a platform based on the $DSP1$ for different platform configurations. Each configuration uses a number n of active cores

by the gray $P_{min,ideal}$ line. However, the active core count can only take integer values, and thus only some points of this line are actually reachable. Those points correspond to the black squares on the Figure 7.6. The minimal power of the platform is thus in reality a sawtooth-shaped line illustrated by the thick black line in Figure 7.6. This shape is the consequence of the discrete number of core of the platform.

This line is composed of plateaus and cubic growths. Plateaus correspond to throughput values where a configuration of cores work at the minimal voltage, and cubic growth correspond to throughput values where the voltage must be raised in order to keep up with the increased throughput. Those growths end up meeting the next plateau where the use of an additional core is more efficient than increasing the voltage. This new configuration then become the next minimal power plateau.

7.7.2 Power and area

The area of the platform is directly proportional to the active core count, and the area of a core A_{core}, comprising a processor core and its memories:

$$A_{platform} = n \times A_{core} \tag{7.13}$$

Using relation 7.12, it is possible to express the computation power of a platform depending on its area:

$$P_{comp} \cong k_p \times T_p^3 \times \left[\frac{A_{core}}{A_{platform}}\right]^2 \tag{7.14}$$

The relation 7.12 is illustrated on Figure 7.7 for different numbers of active cores and different throughput levels. For each throughput level, the power decreases when the area increases since the frequency and the voltage of the active cores is reduced each time an additional core is used. The voltage can vary between a maximal value at 1.3V and a minimal value at 0.9V. Every intermediate point is not reachable since only some area values are actually possible. Those points are illustrated by black squares on Figure 7.7. When the voltage reaches its lowest or highest value, the power of each core does not change anymore. The total power then only depends on the number of active cores, as shown in relation 7.9. The minimal and maximal power values are thus all placed on two lines, illustrated by the dashed lines on Figure 7.7.

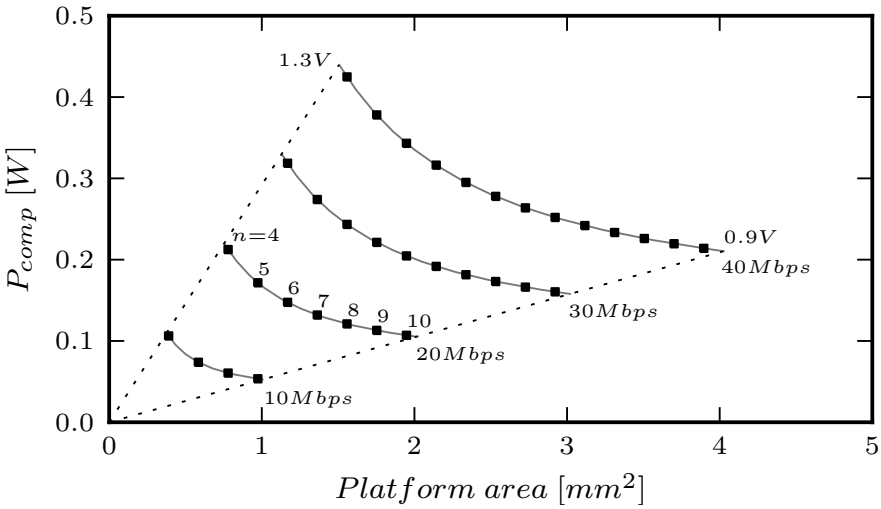

Fig. 7.7. Power vs. area for a $DSP1$ platform at several throughput levels. Black squares represent platform configurations with n active cores.

7.8 CORE ARCHITECTURE IMPACT ON THE PLATFORM

7.8.1 Impact on area

Depending on the ILP contained in the application and exploitable by the architecture, the number of cores required by a platform in order to reach a given throughput will be different. In this section, three platforms are considered, each platform is based on the cores presented in the previous chapter: the first platform uses $DSP1$ cores, the second $VLIW3$ cores, and the last uses $VLIW5$ cores. Figure 7.8 gives the required core count for each platform depending on the throughput at 1.2V. Because of its limited IPC, the number of cores required for the $DSP1$ platform is sensibly higher than for the others. For the platform based on $DSP1$, 2.9× (resp. 1.5×) more cores are required than $VLIW5$ (resp. $VLIW3$) cores for the same throughput. Working on a reduced number of cores has several advantages: in streaming mappings, it allows to reduce inter-core communications since more tasks can be performed inside a single core. It can also ease to port an application to the platform since an important part of the parallelism will then be exploited inside a core, which is automatically handled by the compiler when the instruction-level parallelism is available.

The total area of an HMCP depends on the core count and on the area of each core and memories. The low parallelism of simple architectures, like the $DSP1$, does not allow them to use their datapaths as efficiently as VLIW architectures since all functions must be present but only one instruction can be issued at each cycle. Moreover, data memory sizes are dictated by the application needs and its

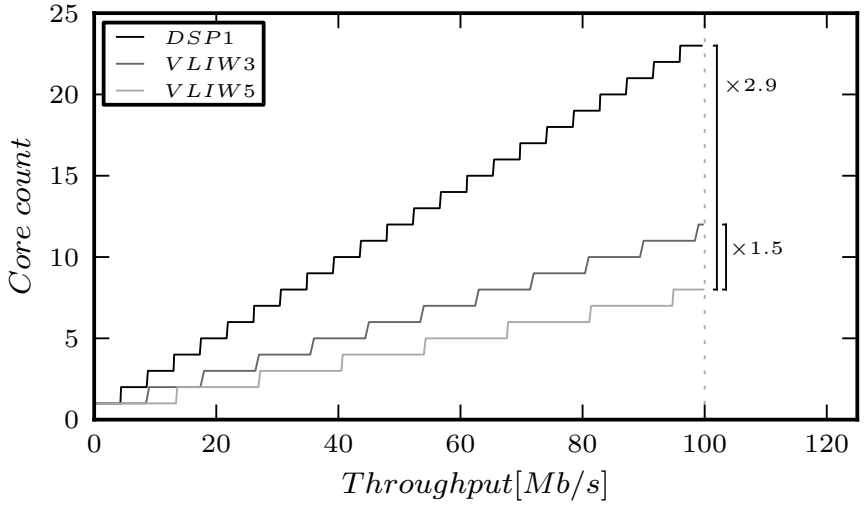

Fig. 7.8. Core count vs. throughput @1.2V. At 100Mb/s, the $DSP1$ platform needs 2.9× more cores than the $VLIW5$ platform, and the $VLIW3$ platform 1.5× more.

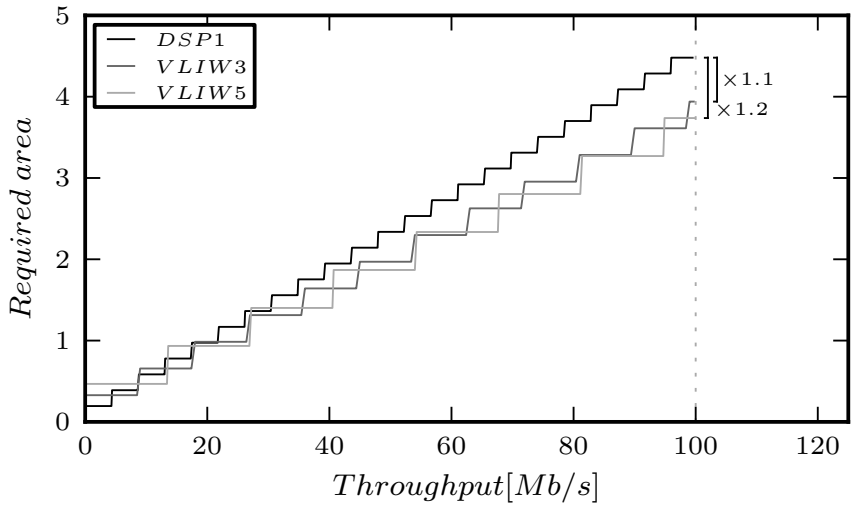

Fig. 7.9. Area vs. throughput @1.2V. At 100Mb/s, the $DSP1$ platform is 1.1× bigger than the $VLIW5$ platform, and the $VLIW3$ platform 1.2× bigger.

data structures. Each core will potentially have to process an entire data element. For instance, for an FFT, each core must be able to process a complete OFDM symbol. It is thus not possible to reduce those memories below a certain minimal size. Therefore, simpler core architectures exploit their area less efficiently than the others on the datapath level and on the data memory level. Consequently, platforms using those architectures also exploit less efficiently their area, which leads to a higher area consumption. Figure 7.9 shows the total areas of the three platforms depending on the application throughput at 1.2V. It is possible to see the consequence of the poor area utilization of $DSP1$: the platform based on $DSP1$ cores requires a higher area than platforms based on the VLIW cores for the same throughput. At 100Mb/s, the $DSP1$ platform requires respectively 20% and 10% more area than the $VLIW5$ and $VLIW3$ platforms.

7.8.2 Power efficiency

The power efficiency of the three platforms is compared using the minimal power they can reach. Figure 7.10 illustrates this minimal power for throughput constraints up to 100Mb/s. This is the same view as in Figure 7.6, for each architecture. $DSP1$ architecture allows to reach the lowest power for all throughput constraints. $VLIW5$ (resp. $VLIW3$) mean power consumption is 25% (resp. 10%) higher than $DSP1$ mean power consumption. The minimal power of an HMCP is reached when the cores work at their minimal voltage without any constraints on area. When the three platforms reach this state, core voltages do not change anymore. The power efficiencies of the platforms are then equal to the power efficiencies of the cores. With no area constraints, $DSP1$-based platforms can thus reach a minimal power that is lower than VLIW-based platforms, since the $DSP1$ has a higher power efficiency.

7.8.3 Impact of area on power efficiency

An HMCP can improve its power efficiency by exploiting more cores in parallel, and by reducing their voltages in order to put each core in its maximal power efficiency state, which is reached when the voltage takes its minimum value. This is a consequence of relation 7.5. This introduces a compromise between the platform area and its consumption. Figure 7.11 shows the consumption of the three platforms depending on their areas for a throughput constraint of 100Mb/s. These lines represent the same view as the one shown in Figure 7.7. For a same throughput, the platform power is reduced when the area is raising. At its maximal power efficiency, the $DSP1$ platform allows a reduction of 19% of the power compared to the $VLIW5$-based platform, but at the cost of a 25% bigger area. For the $VLIW3$-based platform, the gain is only 8% for an area increase of 18% (not shown in the graph). When the platform is constrained, for instance, for cost reasons, the powers of the different platforms are similar. On all the common range of area values, the power values of the three platforms

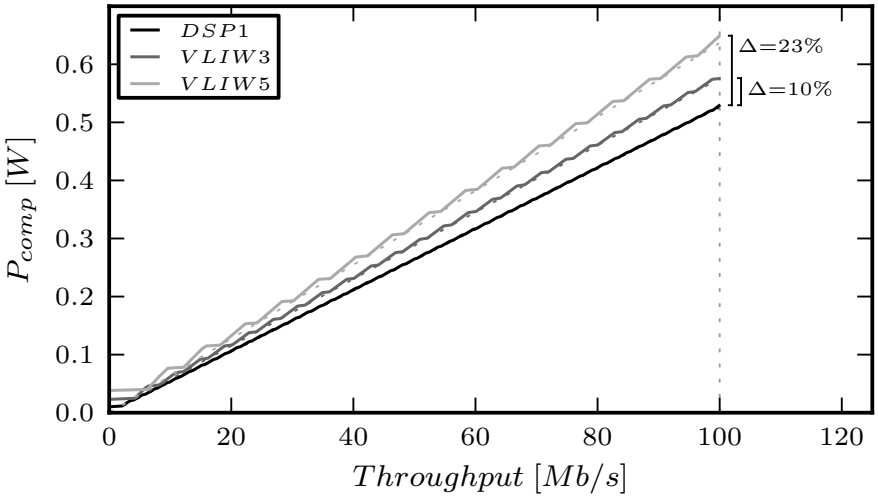

Fig. 7.10. Power vs. throughput for all platforms. At 100Mb/s, the $DSP1$ consumes the lowest power. The mean differences between power of platforms using different core architectures are 23% for the $VLIW5$, and 10% for the $VLIW3$.

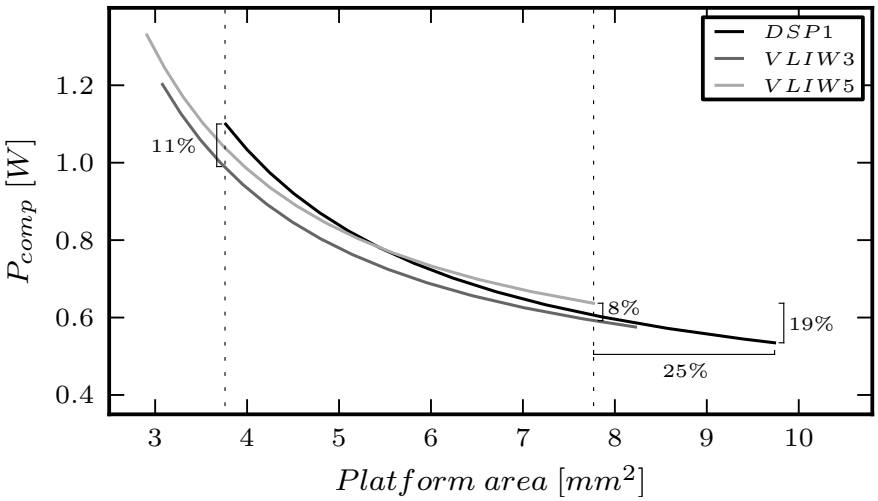

Fig. 7.11. Power vs. area for all platforms under a constraint of $100Mb/s$. Power differences between architectures stay below 11% on the common range of area values. The lowest power is obtained using $DSP1$ cores with a 25% larger area and a 19% lower power than when using $VLIW5$ cores.

differ from less than 11%. This difference is only 8% at the area corresponding to the $VLIW5$-based platform area when working at the lowest voltage. Moreover, on this same range, the $VLIW3$ platform has the best power efficiency. This is explained by the fact that the better power efficiency of $DSP1$ platforms, which is obtained when there is no area constraint, is degraded by its poor area utilization, as illustrated in Figure 7.3.

For the same area, the $DSP1$ platform will always need a higher voltage in order to reach the same throughput as the VLIW-based platforms. When a platform is constrained by its area, the lower power efficiency of VLIW platforms (as shown in Figure 7.10) can thus be mitigated by its good area utilization. Moreover, the power efficiencies of the cores are then very similar whatever the architecture used.

7.9 IMPACT OF THE COMMUNICATION NETWORK

7.9.1 Evaluation methodology

In order to provide efficient implementations to perform inter-core communications in the platform, HMCPs exploit highly-scalable communication networks. The power consumption due to communications in those networks depends on several parameters. The main parameters are the following:

- network topology: different topologies can be used. For instance, 2-D mesh or torus networks.
- routing algorithm: data routing can be realized statically or dynamically, in a deterministic way or not.
- network size: network sizes can vary, they can have different channel width, wire lengths, and routers with different complexities.
- application: the application determine the amount of data to transmit and the real-time constraints. It has also a strong influence on the mapping and the traffic distribution.

Network dimensions can be adapted to the cores used in the platform, since cores with different architectures generally have different physical dimensions. However, it is also possible to use a same network, with the same physical dimensions, in many-core platforms using different cores and having a different core count. Figure 7.12 illustrates the use of an identical network for different platform configurations with a 2-D mesh network, which is a popular topology in HMCPs. By using a single router for several cores, this configuration is possible. For the cores developed in this thesis, such a configuration can be obtained if the cores are grouped together with respectively 2, 3, and 5 cores per router for the $VLIW5$, $VLIW3$ and $DSP1$ architectures. In that case, the areas of each

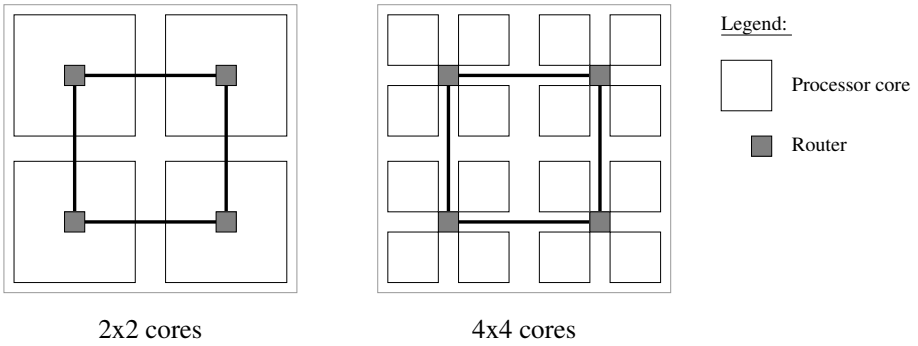

Fig. 7.12. Two homogeneous many-core platform configurations with the same area but different cores and a different core count. The two platforms use the same communication structure, with the same physical dimensions. In the second configuration, a router is connected to several cores.

core group are very similar, and it is possible to use a network with the same physical dimensions that fits the three platforms.

The precise study of all communication network parameters goes beyond the scope of this work. Consequently, in order to keep the analysis general, the study of the communication network impacts is done without considering a specific network architecture. Moreover, in order to compare the influence of the communication network in platforms using different core architectures, it is supposed that the network used in the platforms are the same, i.e. that they have the same architecture and the same physical dimensions. It is also supposed that the cores are distributed uniformly on each router, and that the networks use the same routing algorithm.

For the evaluation of the communication network impact on the platform, the application is the same as in the previous evaluations, with the same mapping on the cores. In this application, the data to modulate are sent from the IOs to the core, and the modulated data are sent from the cores to the IOs linked to a digital/analog converter (DAC). Figure 7.13 illustrates this situation: data enter from the left of the platform, and modulated data exit from the right to the DAC. In the considered application, the modulations, are mapped in data-parallel mode on the cores. In this configuration, kernel executions are entirely performed inside each core. Network traffic is therefore limited to input/output data.

In this work, it is supposed that the compared platforms use the same physical network. Moreover, the cores are distributed uniformly on each router, as well as kernel executions on the cores. The traffic between the IOs and the router is therefore the same for each core architecture. Consequently, for the same input data throughput, the network activity is the same in the three compared platforms.

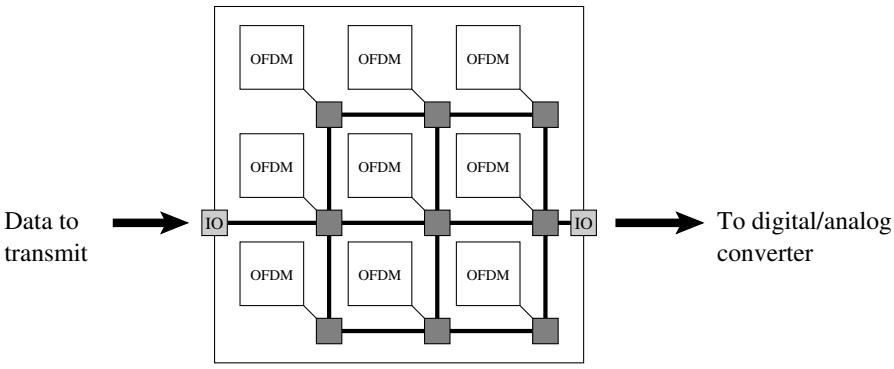

Fig. 7.13. Application configuration on the compared platforms. Each OFDM kernel is fully mapped on a single processor core.

The methodology presented here does not require to consider a particular network architecture. It allows therefore to realize a general study of the communication network impact on an HMCP. However, the hypotheses used to perform the analysis also limit the variability of the parameters taken into account. In order to go further in the study of the communication network impact, additional work must be performed. For instance, comparisons can be realized on different platform configurations with different networks. In the same way, different applications can also be evaluated in order to consider different traffics, mappings and throughputs.

7.9.2 Impact on platform area

In HMCPs, the communication structures have highly scalable architectures. In a general way, the area occupied by the network grows proportionally to the platform core count. The area also depends on the network architecture and its dimensions. For instance, the AsAP platform uses a simple 16-bit 2D mesh network which is circuit-oriented. In this platform, the network represents about 7% of the total area occupied by the cores [27]. Other platforms, like TILE64, use more complex networks, with higher operating frequencies and more elaborated routing algorithms [30].

For the evaluation realized in this work, it is supposed that the network used in the three platforms is the same. In that case, the network impact on the platform areas can be expressed as the addition of a constant value A_{comm} which represents the communication structure area:

$$A_{platform} = n \times A_{core} + A_{comm} \tag{7.15}$$

Consequently, considering the communication structure does not change the results obtained in the previous section, since the three platforms are always

compared on identical area values, and the impact on areas is the same on all platforms. In the following, the simplified equation 7.13 is therefore still used to perform the evaluation.

7.9.3 Evaluation of network power consumption

The mean communication power of a platform can also be computed using the following equation:

$$P_{comm} = \frac{E_{comm}}{T} \tag{7.16}$$

In equation 7.16, E_{comm} represents the consumed energy in the communication network for the transmission of a set of packets during a time T. In the considered applications, the packet set consists in all the input packets going to the cores, and all the packets going out to the DAC. T is then the time period on which all the input data must be modulated and sent out to reach the required throughput. For a given throughput, and with the considered hypotheses, the energy consumed by the network varies only with the supply voltage. Therefore, equation 7.16 can be rewritten to take this into account:

$$P_{comm} = \frac{k_{comm} \times V_{dd}^2}{T} \tag{7.17}$$

In equation 7.17, k_{comm} is a constant representing the impact of all other parameters, like network physical dimensions, its architecture, the traffic distribution, the transmitted data, etc. In order to perform the comparisons, it is supposed that the traffic and the networks do not vary between the platforms. Moreover, the platforms are compared for the same throughput. k_{comm} and the period T are therefore identical for the three platforms. The communication powers of two such platforms are therefore related by the following equation:

$$P_{comm,2} = P_{comm,1} \times \frac{V_{dd,2}^2}{V_{dd,1}^2} \tag{7.18}$$

In equation 7.18, symbols with different indices are related to different platforms. This last equation allows to compute the communication network power of a platform knowing the other.

In this work, in order to compare the communication network power of the three platforms, one of them is selected as a reference platform. The communication network power of the other platforms are then computed using equation 7.18. The communication network power of the reference platform is estimated differently. Since it has been chosen to study the communication network impact in a general way, without considering a specific network, this power is estimated as a percentage of the total platform power:

$$P_{comm} = \eta \times P_{platform} \tag{7.19}$$

In equation 7.19, η is the percentage of the total platform power due to the communication structure. In the AsAP platform, this power represents between 7 and 10% of the total platform power [27]. However, in this platform, communication are mostly local. In order to study a more general situation, a higher percentage, up to 30%, is considered in the evaluations.

7.9.4 Impact on the platform power

If the communication network is supplied with the same voltage in all the compared platforms, equation 4 shows that the network power consumption is the same for each platform. In that case, the platform total power consumptions evolve the same way as P_{comp}, as presented in Figure 7.11 in the previous section. Indeed, if the communication power is the same for each platform, it causes their powers to rise with the same amount and do not change their relative positions.

The situation is not the same if the communication network is supplied with the same voltage as for the cores. Indeed, for a given platform area, the platform supply voltages are different in each platform in that case. Figure 7.14 presents the evaluation of this power for the three platforms. In this work, the $DSP1$ platform is selected as the reference platform, and its communication power is computed using equation 7.19. The communication network power consumptions of the $VLIW3$ and $VLIW5$ platforms are then computed relatively using equation 7.18. For clarity purpose, $DSP1$ communication power is only illustrated for a single percentage, fixed to 15% of the total platform power consumption. The corresponding $VLIW3$ and $VLIW5$ power consumption values are also illustrated. The results show that the $DSP1$ platform presents a higher power consumption due to its higher supply voltage compared to the other platform for a given platform area. Indeed, for the same platform area, $DSP1$ cores must work at a higher operating frequency than VLIW cores to reach the same throughput.

Figure 7.15 presents the evolution of the total platform consumption for a range of platform area values. This figure is an update of the Figure 7.11 presented in the previous section, which now includes power from the communication network. In this figure, three percentages of the total $DSP1$ platform power are considered for the communication network: 0%, 15% and 30%. 0% percentage corresponds to $P_{platform} = P_{comp}$, it is illustrated in order to enable comparison with the previous results. The corresponding power consumptions of the $VLIW3$ and $VLIW5$ platforms are also illustrated. The results show that taking the communication power consumption into account degrades $DSP1$ platform performance compared to the other platforms. Indeed, when the power percentage due to the communication network increases, $DSP1$ platform power progressively becomes higher than the VLIW platform power consumptions. At

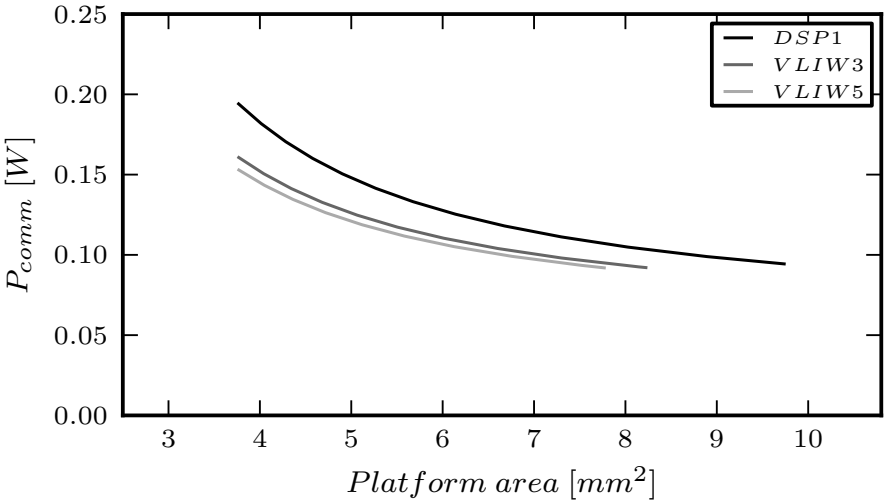

Fig. 7.14. Power consumed by the communication structure in the three compared platforms. The power of the $DSP1$ platform is used as the reference, and is estimated using equation 7.19 for a percentage of 15% of the total platform power. $VLIW3$ and $VLIW5$ platform power values are computed relatively using equation 7.18.

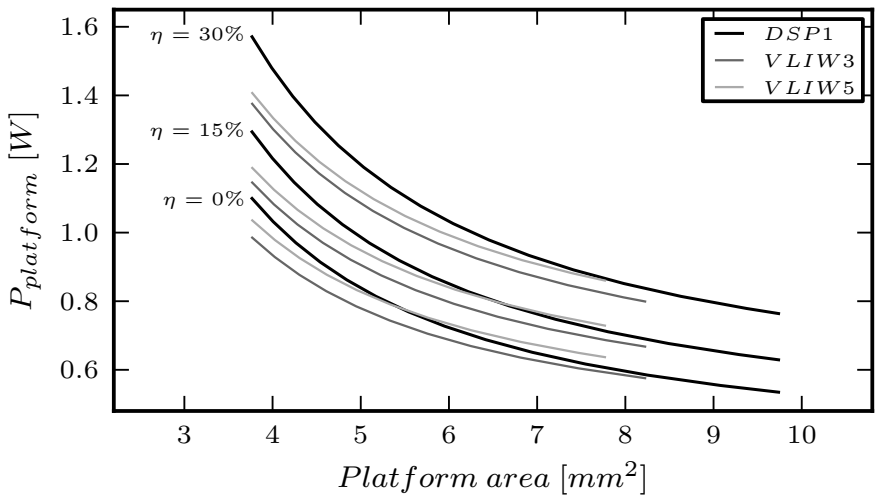

Fig. 7.15. Total power vs. area for all platforms under a constraint of 100 Mb/s. Communication power in the $DSP1$ platform is estimated as 0%, 15% and 30% of the total platform power. 0% corresponds to the power of the platform without any contribution from the communication structure.

30%, VLIW platforms even have platform power consumptions that are lower or equal to the $DSP1$ platform power on the whole considered range of area values. On the opposite, VLIW platform power consumptions become similar when the percentage of power due to the communication network increases. Consequently, the $DSP1$ platform suffers the most from the communication power due to its higher supply voltage for a given platform area.

7.10 SUMMARY

This chapter has evaluated the impact of core architectures on the power efficiency of homogeneous many-core platforms targeting DSP applications. To do so, a comparison of the performance of three HMCPs based on different processor architectures has been realized. To perform the comparison, both the contribution of the processor cores and the communication structure on the platform total power consumption have been evaluated.

The platform computation powers have been computed on the basis of their core performance. Since those performance have been evaluated on placed and routed cores, the evaluations provide precise results. The comparisons of the platform computation powers show that, without constraints on the platform area, the platform based on the single-issue core allows to reach the best power efficiency. However the required throughput is then obtained at the cost of a sensibly higher platform area. When the platform is constrained on its area, the power efficiencies of the three platforms become similar. On the common range of area values, the power consumptions of the three platforms differ from less than 11%. Those results can be explained by the fact that VLIW platforms provide better area utilization compared to the $DSP1$ platform which compensates its lower power efficiency.

The impact of the communication network has also been studied. The power consumption of the communication network has been compared for the three platforms. In order to realize the evaluation, the communication network power consumption has been modeled as a percentage of the total power consumption. This approach allows to prevent the analysis from depending on a specific network architecture. In order to compare the communication network impact on the three platforms, the study considers the use of the same network, with the same physical dimensions, on each platform with a uniform distribution of the cores on each network router. The application used for the evaluation and the kernel execution mapping on the cores is also the same. With this approach, the network activity is the same on each platform.

The analysis shows that if the network supply voltage is kept identical in the three platforms, then the results of the communication network power comparison are the same as for the computation power. Indeed, in that case, the communication network has the same impact on each platform. However, if the networks are supplied with the same voltages as the cores, the results show that the communications are unfavorable to $DSP1$ platforms. Indeed, for a given plat-

form area, VLIW platforms can operate with a lower supply voltage, and have therefore a lower communication network power consumption than $DSP1$ platforms. When the communication network power consumption represents 30% of the total platform power, VLIW platforms have even platform power consumptions that are lower or equal to the $DSP1$ platform power on the whole range of considered area values.

CHAPTER 8

CONCLUSION

8.1 CONTEXT OF THE WORK

Embedded signal processing applications evolve towards advanced applications supporting many standards and providing advanced functionalities. They also require high data rates, and use elaborated signal processing operations. Consequently, the control and software parts in those applications are increasing. Because of this evolution, implementations of signal processing applications shift progressively from full hardware implementations to partial or full software implementations. In this thesis, those software implementations are regrouped under the name software-defined signal processing applications, or SDAs.

Among embedded electronics, many applications have low volumes of production. This is for instance the case of professional electronics, product preseries, application prototypes or high-end consumer products. For those applications, the non-recurring engineering costs (NREs) related to application development and platform realization are a major issue. Indeed, due to their limited production volumes, those applications cannot easily amortize high NREs.

This thesis has studied hardware platforms for low-volume embedded SDAs. The requirements of advanced signal processing applications, SDAs, embedded systems, and the technology limitations introduced by technology scaling in deep-submicron nodes have been identified. Together, they define five main requirements for hardware platforms targeting low-volume SDAs:

1. The platform must support dynamic changes of its functionality with the flexibility of software in order to fully support the SDA approach.

2. It must support a high computational load in order to face the high data rates and elaborated operations of advanced signal processing applications.

3. It must have a high power efficiency in order to respect the limited power budget of embedded systems. Moreover, this power efficiency must be obtained by exploiting architectural techniques or aggressive optimizations on the design in order to solve technology limitations.

4. In order to reduce the NREs related to the application development, the platform must be easily programmed to reduce development efforts.

5. In order to reduce the NREs related to the platform itself, it must be highly generic. It allows to use the platform in many applications, and therefore amortize the costs on a larger volume.

Conventional solutions for signal processing applications like heterogeneous platforms lack flexibility and genericity. They cannot therefore meet those requirements. This situation has motivated the development of new platforms. The platform models considered in this work are multi-core processors, SIMD processors, coarse-grain reconfigurable platforms, fine-grain reconfigurable platforms like FPGAs, and homogeneous many-core platforms (HMCP). In this thesis, two

platforms are studied for low-volume SDAs: FPGAs with dynamic partial reconfiguration and HMCPs.

8.2 SUMMARY OF THE CONTRIBUTIONS

The main contribution of this thesis is **a set of analyzes, methodologies, and solutions regarding the use of FPGAs with DPR and HMCPs to implement low-volume software-defined signal processing applications.**

The detailed contributions are:

1. **A comparison of existing SDA platform models in regards to SDA requirements and low-volume applications:**

   Five main platforms have been introduced, and for all models, existing platforms are presented. A comparison of the existing SDA platform models has been realized. To perform this comparison, the platform models have been evaluated according to five criterions directly derived from low-volume SDA requirements. The evaluations have been realized on basis of global platform model properties, and representative benchmarks. The comparisons have provided insights on the advantages and disadvantages of the different platforms. The results show that FPGAs with DPR and HMCPs are the best candidates.

2. **An analysis of the impact of dynamic partial reconfiguration on design and development:**

   The advantages of using DPR for low-volume SDAs have been identified. The additional complexity introduced by DPR is evaluated for the design and development flow. The missing elements of the tools and flows are also identified. The analysis shows that DPR presents important advantages for SDAs: it provides the flexibility and high performance that these applications require. The results also show that DPR is a technique that has repercussions at every level of a system. Development flows lack critical elements like simulation tools that enable to evaluate and test a design before full integration and implementation. These missing elements are an important issue for applications which require a strict validation or certification, like it is the case for several professional applications.

   In order to exploit DPR, this contribution also proposes to handle dynamic reconfigurations of reconfigurable blocks using small local microcontrollers to manage them. Evaluations show that using this approach only incurs a small resource overhead.

3. **A minimal flow and a set of guidelines to use DPR in applications that require certification:**

   A minimal flow is proposed to realize dynamic partial reconfiguration. The flow is based on bitstream documentation and only use simple operations. This makes it more easily certifiable than vendor tools that are proprietary and not certified. DPR also introduces additional complexity in the design which makes certification of the design more complex. In order to ease certification, guidelines are proposed for the development of an application using a DPR-enabled design. Those guidelines contribute to facilitate DPR-enabled design certification.

4. **An analysis of DPR impact on the power consumption of a design:**

   DPR allows to improve power efficiency of a design by exploiting hardware virtualization. This technique allows to instantiate dynamically optimized hardware blocks in the FPGA in order to perform a specific operation. It provides the efficiency of hardware implementations while having the flexibility and diversity of software. The potential gains in power have been evaluated by realizing measures on an application. Those measures show that DPR allows to realize significant power gains by exploiting dedicated hardware rather than a software implementation of a function.

   The reconfiguration process energy consumption is also analyzed. The different sources of power consumption related to the reconfiguration are evaluated, while taking into account the real-time constraints. Measures are performed on a DPR-enabled design. They show that the impact of the reconfiguration on the power is mainly caused by the transfers from the external memories. This power is even more important when real-time constraints impose a high reconfiguration speed since faster and power hungry memories are required. The results also show that dynamic partial reconfiguration usage is not suitable for a high reconfiguration frequency. Indeed, the reconfiguration energy cost is several orders of magnitude higher than the energy cost of the operation execution in the reconfigured functional block. In order to be profitable, reconfigured blocks must therefore be used for a long period of time.

   The potential area reduction provided by hardware virtualization is also evaluated on an SDR application. Results show that hardware virtualization also enables the use of smaller components, allowing therefore to reduce both static power consumption and costs.

5. **A methodology to compare precisely HMCP processor cores at architectural level:**

   A methodology is proposed to realize precise comparisons of processor cores with different architectures. It uses an architectural template together with specific optimizations. The methodology allows to create processor implementations that strongly reduce the interferences on their performance caused by their implementation specificities. Three RISC processor cores have been implemented using this methodology: a single-issue core, and two VLIW cores with 3 and 5 issues respectively. The performance of the three cores and the impact of the applied optimizations have been evaluated. The results show that the methodology allows to build near-optimal implementations of the cores, and confirm that optimizations are required in order to obtain representative performance.

6. **An evaluation of the impact of core architecture on the power efficiency of HMCP:**

   A comparison of three HMCPs is realized using the three implemented processor cores. The impact on processor architecture on the platform power is analyzed. For the evaluations, both the computation and the communication network power of the platform are considered. To perform the evaluations, relations are identified that allow to compute the platform power, frequency and area using the performance results obtained on a single core. To estimate the impact of the communication network, it has been considered that the three platforms use the same network.

   Regarding the computation power, the comparisons show that the simplest cores allow to reach the best power efficiency, but cause a very high area overhead. When the three platforms are constrained to a specific area, their power efficiencies are similar. Indeed, more complex cores, like the VLIWs, consume more energy but have a better area efficiency and provide higher speedups.

   When the communication network is taken into account, the evaluations show that the results do not change if the network has the same supply voltage in all compared platforms. However, if the network supply voltage is the same as for the cores, the results show that using simpler cores is a disadvantage. Indeed, simpler cores must run at higher frequencies to reach the required throughput for the same area, and need therefore a higher supply voltage. This causes the power related to the communication network to be higher in those platforms. Consequently, using VLIW cores allows to work with lower supply voltages, which reduces the communication network impact on the platform power.

8.3 CONCLUSIONS

This thesis has realized a study of platforms for SDAs in low-volume applications. Three aspects have been considered:

1. The evaluation and the comparison of platform models for low-volume SDAs.

2. The study and evaluation of a first platform candidate for low-volume SDAs: FPGAs with dynamic partial reconfiguration.

3. The study and evaluation of a second candidate: homogeneous many-core platforms.

The first part of this work provides a compared evaluation of five platform models for low-volume SDAs. Existing works generally consider only a single solution, or limit themselves to surveys of possible solutions. Since they do not perform any comparison, they do not allow to identify the most adequate platform model that can be used to implement an SDA. In this work, a global approach has been followed which considers several possible solutions. Moreover, a comparison based on five criterions has been proposed. Each platform has been evaluated using the same basis of comparison. Those evaluations have therefore allowed to create a comparison grid that helps to clearly identify the advantages and disadvantages of each platform with respect to low-volume SDA requirements.

Regarding DPR, this thesis studies the issues related to the industrial usage of DPR. Indeed, most existing works study DPR without evaluating the impact of this technology. However, with SDAs, designs are becoming significantly more complex. The issues related to application implementations using complex designs has therefore become a major challenge. Development flows and tools are key elements in this problem. Moreover, this work studies low-volume SDAs where development issues are even more important, for instance because of certification requirements. This thesis has studied those aspects. It has notably showed the impact of DPR in the design and the development. Missing elements in the development flow have been identified. In particular, a development flow and design guidelines have been proposed to ease certification.

A study has also been performed on the impact of DPR on design power consumption. Indeed, those aspects are crucial nowadays. Contrary to existing works, the evaluations realized here have been done for the complete system, and also take into account additional power consumption sources, like external memories. This work allows therefore to identify the key elements that contribute to the power consumption of DPR-enabled designs. In particular, the study has showed the major participation of partial bitstream transfers from external memories during the reconfigurations.

Regarding HMCPs, this work has precisely compared the impact of the core architecture on the platform performance. Existing HMCPs have nowadays sev-

eral hundreds of cores, but future platforms will have thousands of them. However, comparing precisely two cores with different architectures is already a difficult task. Consequently, comparing two homogeneous many-core platforms is even more difficult. In this work, a methodology is proposed that allows to perform precise comparisons of processor core with different architectures. This methodology solves the comparison issue and enables HMCP comparisons with different architectures.

Using the proposed methodology, three HMCPs based on different processor architectures have been compared. Existing platform comparisons only compares the throughput, power and features of those platforms. However, in order to compare HMCPs, one must also considered the interaction of the core area with those aspects. In this work, the impact of the core area is taken into account precisely since evaluations are based on placed and routed cores. The comparisons also take the inter-core communications into account. No specific communication network architecture is considered for the evaluations, which allows to keep the study general. Comparison results have shown that, contrary to what is generally thought, using simple cores with a low power consumption is not the only interesting approach in HMCPs. Indeed, the results have shown that more complex cores, like VLIWs, have a better area efficiency, and can provide performance similar to platforms using simpler cores.

In conclusion, this thesis has allowed to precisely evaluate and distinguish platform models targeting SDAs and low-volume applications. In particular, the knowledge of FPGAs with DPR and HMCPs have been strongly improved. Regarding DPR-enabled designs, the issues related to its industrial usage and the potential gains provided by DPR have been precisely evaluated. Several solutions are also proposed to ease its use in this context. Regarding HMCPs, the impact of the core architecture on the platform has been evaluated. This evaluation provides a better understanding of this impact and allows to make better choices when selecting the processor core architecture used in homogeneous many-core platforms.

8.4 FUTURE WORK

8.4.1 Comparison of platform models for SDAs

The platform model comparison has been realized on basis of criterions for low-volume SDAs. Performance evaluations have been realized on existing platforms, using benchmarks that are representative of their target applications. The evaluations are therefore representative of the platform performance.

However, most platforms are designed and dimensioned for the software-defined radio application. Indeed, telecommunications currently represent the most important application domain for SDAs, and SDR is the subject of many

researches. The SDR is therefore the application targeted by most of the platforms for SDAs. Consequently, in this work, the evaluations reflect the performance of those platforms but the results are influenced by the dominance of the SDR applications. With the evolution of signal processing applications towards SDAs, those platforms will progressively target other application domains. In order to complete the analysis that has been performed in this work, it will be necessary to consider other application domains and identify their particularities.

8.4.2 FPGAs with DPR

After the study of DPR impact on design and development, it appears that important elements are missing in the development flows. In particular, it is necessary to build tools that allow precise simulation and debugging of designs before their complete integration. DPR usage has also repercussions at every abstraction level. Using dynamic hardware is indeed a recent approach, and there is still no standard way of exploiting it. Future works in this domain require therefore to propose models allowing efficient exploitation of dynamic hardware.

Regarding certification, there is currently no integrated methodology allowing to help the designer in its work when building safety-critical applications using DPR-enabled designs. In order to enable this, it is necessary to continue the work in this domain by proposing complete development flows and tools that take the specificities of DPR into account. For instance, formal methods can be used to ease the specification, the evaluation and the validation of DPR-enabled designs.

Finally, regarding the evaluation of DPR impact on power consumption, this work has identified the key elements contributing to the design power consumption. The study has evaluated each of those elements in a general way without considering their impact compared to application requirements. Those relative aspects are however important when evaluating the opportunity of using DPR in a real application. For instance, the impact of partial bitstream storage and their access will be totally different in a sensor node or in an airport radar. It is therefore relevant to make additional researches considering the relative impact of DPR on specific applications. An interesting approach would be to analyze existing applicative works on DPR.

8.4.3 HMCPs

HMCP evaluations have been realized on software-defined radio applications. The evaluations have allowed to compare the performance of several HMCPs based on different processor core architectures, and therefore study the impact of the core architecture on their performance.

However, different applications also require different platform dimensions. For instance, in an HMCP, the memory blocks or the communication network will not be the same depending on the application. The relative impacts of several

application types on HMCP are currently not well understood. Indeed, studying those aspects with sufficient precision requires to build platforms that are highly optimized for the targeted applications. However, building such optimized implementations is a complex task: each application type requires specific instructions, specific memory sizes, a communication network that can provide sufficient throughput for the application, etc. Moreover, porting new applications has become very difficult with the shift from single-core to multi-core platforms since many issues related to development flows are still a challenge.

This development complexity renders evaluations and comparisons of HMCPs on several SDAs very difficult to realize. Due to this complexity, only a limited range of parameter values can be considered in the evaluations. It also prevents to perform early design evaluations, which demands to evaluate application performance without having completely implemented it. To get a better understanding on the impact of applications on a platform, further works are therefore required. In particular, it would be interesting to model complete SDAs in order to evaluate their impact on complex multi-core solutions without having to perform a complete implementation first.

REFERENCES

1. K. Berkel, A. Abbo, S. Balakrishnan, R. Kleihorst, P. Meuwissen, and R. Nas, "Vector Processing as an Enabler for Ambient Intelligence," AmIware Hardware Technology Drivers of Ambient Intelligence, pp. 223–243, 2006.
2. J. Mitola III and G. Q. Maguire Jr, "Cognitive radio: making software radios more personal," IEEE personal communications, vol. 6, no. 4, pp. 13–18, 1999.
3. K. Flautner, "The wall ahead is made of rubber," in Design and Diagnostics of Electronic Circuits and Systems, 2008. DDECS 2008. 11th IEEE Workshop on. IEEE, 2008, p. 1.
4. W. Wolf and A. Jerraya, Multiprocessor Systems on Chips. Morgan Kaufmann, 2005.
5. B. Rousseau, P. Manet, I. Loiselle, J.-D. Legat, and H. Vandierendonck, "A methodology for precise comparisons of processor core architectures for homogeneous many-core DSP platforms," in Proceedings of the 2010 Conference on Design and Architectures for Signal and Image Processing, 2010, pp. 76–83.
6. J. P. E. O. J. JTRS, "Software communication architecture specifications v2.2.2," Tech. Rep. [Online]. Available: http://sca.jpeojtrs.mil
7. I. Richardson, H. 264 and MPEG-4 video compression: video coding for next-generation multimedia. John Wiley & Sons Inc, 2003.
8. Fujitsu, "Mb86h61 hd multi-standard decoder," Tech. Rep., 2010.
9. K. De Bosschere, W. Luk, X. Martorell, N. Navarro, M. O'boyle, D. Pnevmatikatos, A. Ramirez, P. Sainrat, A. Seznec, P. Stenstrom et al., "High-performance embedded architecture and compilation roadmap," Transactions on High-Performance Embedded Architectures and Compilers I, pp. 5–29, 2007.
10. G. Moore, "Lithography and the future of Moore's law," in Proceedings of SPIE, vol. 2438, 1995, p. 2.
11. P. Ross, "Why cpu frequency stalled," Spectrum, IEEE, vol. 45, no. 4, p. 72, 2008.
12. J. Noguera, R. Esser, K. Paulsson, M. Huebner, and J. Becker, "Towards Novel Approaches in Design Automation for FPGA Power Optimization," Integrated Circuit and System Design. Power and Timing Modeling, Optimization and Simulation, pp. 419–428, 2009.
13. P. Kollig, C. Osborne, and T. Henriksson, "Heterogeneous multi-core platform for consumer multimedia applications," in Design, Automation & Test in Europe Conference & Exhibition, 2009. DATE'09. IEEE, 2009, pp. 1254–1259.

14. D. TMS320TCI6487, "Platform," Texas Instruments Incorporated, 2006.
15. J. Glossner, D. Iancu, M. Moudgill, G. Nacer, S. Jinturkar, S. Stanley, and M. Schulte, "The sandbridge sb3011 platform," EURASIP Journal on Embedded Systems, vol. 2007, no. 1, p. 16, 2007.
16. K. Van Berkel, F. Heinle, P. P. E. Meuwissen, K. Moerman, and M. Weiss, "Vector processing as an enabler for software-defined radio in handheld devices," EURASIP Journal on Advances in Signal Processing, vol. 2005, no. 16, pp. 2613–2625, 1900.
17. Y. Lin, H. Lee, M. Woh, Y. Harel, S. Mahlke, T. Mudge, C. Chakrabarti, and K. Flautner, "Soda: A low-power architecture for software radio," pp. 89–101, 2006.
18. M. Woh, S. Seo, S. Mahlke, T. Mudge, C. Chakrabarti, and K. Flautner, "Anysp: anytime anywhere anyway signal processing," pp. 128–139, 2009.
19. B. Bougard, B. De Sutter, D. Verkest, L. Van der Perre, and R. Lauwereins, "A coarse-grained array accelerator for software-defined radio baseband processing," IEEE micro, vol. 28, no. 4, pp. 41–50, 2008.
20. "Montium reconfigurable digital signal processing tile processor (tp)," Tech. Rep., September 2007.
21. "Xpp-iii processor overview," Tech. Rep., 2006.
22. X. Inc., "Virtex-6 family overview," Tech. Rep., 2010.
23. "Xilinx inc." [Online]. Available: http://www.xilinx.com
24. "Altera inc." [Online]. Available: http://www.altera.com
25. A. Corporation, "Altera unveils innovations for 28-nm fpgas," 2010. [Online]. Available: http://www.altera.com/corporate/news_room/releases/2010/products/nr-innovating-at-28-nm.html
26. R. Kumar, V. Zyuban, and D. M. Tullsen, "Interconnections in multi-core architectures: Understanding mechanisms, overheads and scaling," SIGARCH Comput. Archit. News, vol. 33, pp. 408–419, May 2005. [Online]. Available: http://doi.acm.org/10.1145/1080695.1070004
27. A. T. Tran, D. N. Truong, and B. M. Baas, "A reconfigurable source-synchronous on-chip network for gals many-core platforms," IEEE transactions on computer-aided design of integrated circuits and systems, vol. 29, no. 6, p. 897, 2010.
28. "Pc102 datasheet," Tech. Rep., July 2006.
29. M. Butts, "Synchronization through communication in a massively parallel processor array," Micro, IEEE, vol. 27, no. 5, pp. 32–40, Sept.-Oct. 2007.
30. S. Bell, B. Edwards, J. Amann, R. Conlin, K. Joyce, V. Leung, J. MacKay, M. Reif, L. Bao, J. Brown et al., "Tile64 processor: A 64-core soc with mesh interconnect," pp. 88–598, 2008.
31. M. B. Taylor, J. Kim, J. Miller, D. Wentzlaff, F. Ghodrat, B. Greenwald, H. Hoffman, P. Johnson, J.-W. Lee, W. Lee, A. Ma, A. Saraf, M. Seneski, N. Shnidman, V. Strumpen, M. Frank, S. Amarasinghe, and A. Agarwal, "The raw microprocessor: a computational fabric for software circuits and general-purpose programs," Micro, IEEE, vol. 22, no. 2, pp. 25–35, Mar/Apr 2002.

32. G. K. Rauwerda, P. M. Heysters, and G. J. M. Smit, "An ofdm receiver implemented on the coarse-grain reconfigurable montium processor," 2004.

33. T. Instruments, "Tms320tci6487/88 power consumption summary," Tech. Rep., September 2009.

34. D. N. Truong, W. H. Cheng, T. Mohsenin, Z. Yu, A. T. Jacobson, G. Landge, M. J. Meeuwsen, A. T. Tran, Z. Xiao, E. W. Work, J. W. Webb, P. Mejia, and B. M. Baas, "A 167-processor computational platform in 65 nm cmos," IEEE Journal of Solid-State Circuits (JSSC), vol. 44, no. 4, pp. 1130–1144, apr 2009.

35. P. Clarke, "Picochip preps architectural refresh after august tape-out," Dec. 2008. [Online]. Available: http://www.eetimes.com/electronics-news/4191975/PicoChip-preps-architectural-refresh-after-August-tape-out

36. M. Butts, B. Budlong, P. Wasson, and E. White, "Reconfigurable work farms on a massively parallel processor array," in Field-Programmable Custom Computing Machines, 2008. FCCM '08. 16th International Symposium on, April 2008, pp. 206–215.

37. M. Butts, A. M. Jones, and P. Wasson, "A structural object programming model, architecture, chip and tools for reconfigurable computing," Field-Programmable Custom Computing Machines, Annual IEEE Symposium on, vol. 0, pp. 55–64, 2007.

38. D. Killebrew and A. Lee, "L2 to Off-Chip Memory Interconnects for CMPs."

39. H. Wirz, "Vector processor study for a 3.9g bb implementation, terminal side," Tech. Rep., may 2007.

40. Y. Lin, "Realizing Software Defined Radio–A Study in Designing Mobile Supercomputers," Ph.D. dissertation, Citeseer, 2008.

41. G. K. Rauwerda, P. M. Heysters, and G. J. M. Smit, "An ofdm receiver implemented on the coarse-grain reconfigurable montium processor," 2004.

42. P. M. Heysters, G. J. M. Smit, and E. Molenkamp, "Montium - balancing between energy-efficiency, flexibility and performance," in Proceedings of ERSA'03, Las Vegas, USA, June 2003, pp. 235–241. [Online]. Available: http://eprints.eemcs.utwente.nl/1514/

43. V. Baumgarte, G. Ehlers, F. May, A. Nückel, M. Vorbach, and M. Weinhardt, "Pact xpp - a self-reconfigurable data processing architecture," J. Supercomput., vol. 26, no. 2, pp. 167–184, 2003.

44. F. Campi, R. Konig, M. Dreschmann, M. Neukirchner, D. Picard, M. Juttner, E. Schuler, A. Deledda, D. Rossi, A. Pasini et al., "RTL-to-layout implementation of an embedded coarse grained architecture for dynamically reconfigurable computing in systems-on-chip," in System-on-Chip, 2009. SOC 2009. International Symposium on. IEEE, 2009, pp. 110–113.

45. A. Corporation, Cyclone III Device Data Sheet, Altera, January 2010.

46. N. Inc., "Coolflux baseband signal processor," Tech. Rep., March 2008.

47. J. Davis, R. Venkatesan, A. Kaloyeros, M. Beylansky, S. Souri, K. Banerjee, K. Saraswat, A. Rahman, R. Reif, and J. Meindl, "Interconnect limits on gigascale integration (GSI) in the 21st century," Proceedings of the IEEE, vol. 89, no. 3, pp. 305–324, 2002.

48. M. Woh, Y. Lin, S. Seo, S. Mahlke, T. Mudge, C. Chakrabarti, R. Bruce, D. Kershaw, A. Reid, M. Wilder et al., "From SODA to scotch: The evolution of a wireless baseband processor," in Microarchitecture, 2008. MICRO-41. 2008 41st IEEE/ACM International Symposium on. IEEE, 2009, pp. 152–163.

49. M. Alvarez, E. Salami, A. Ramirez, and M. Valero, "Performance impact of unaligned memory operations in SIMD extensions for video codec applications," in Performance Analysis of Systems & Software, 2007. ISPASS 2007. IEEE International Symposium on. IEEE, 2007, pp. 62–71.

50. A. Shahbahrami, B. Juurlink, and S. Vassiliadis, "Performance impact of misaligned accesses in SIMD extensions," in Proc. Workshop on Circuits, Systems and Signal Processing. Citeseer, 2006.

51. G. Ren, P. Wu, and D. Padua, "A preliminary study on the vectorization of multimedia applications for multimedia extensions," Languages and Compilers for Parallel Computing, pp. 420–435, 2004.

52. D. Naishlos, "Autovectorization in GCC," in Proceedings of the 2004 GCC Developers Summit. Citeseer, 2004, pp. 105–118.

53. J. Derby, "Cell programming tutorial," Tech. Rep., May 2006.

54. B. Mei, S. Vernalde, D. Verkest, H. De Man, and R. Lauwereins, "DRESC: A retargetable compiler for coarse-grained reconfigurable architectures," in Field-Programmable Technology, 2002.(FPT). Proceedings. 2002 IEEE International Conference on. IEEE, 2003, pp. 166–173.

55. R. Hartenstein, "Coarse grain reconfigurable architecture (embedded tutorial)," in Proceedings of the 2001 Asia and South Pacific Design Automation Conference. ACM, 2001, pp. 564–570.

56. B. Sutter, P. Raghavan, and A. Lambrechts, "Coarse-Grained Reconfigurable Array Architectures," Handbook of Signal Processing Systems, pp. 449–484, 2010.

57. A. Terechko, M. Garg, H. Corporaal, P. Res, and N. Eindhoven, "Evaluation of speed and area of clustered vliw processors," pp. 557–563, 2005.

58. F. Bouwens, M. Berekovic, B. De Sutter, and G. Gaydadjiev, "Architecture enhancements for the ADRES coarse-grained reconfigurable array," in Proceedings of the 3rd international conference on High performance embedded architectures and compilers. Springer-Verlag, 2008, pp. 66–81.

59. L. J. Karam, I. AlKamal, A. Gatherer, G. A. Frantz, D. V. Anderson, and B. L. Evans, "Trends in multicore dsp platforms," IEEE Signal Process. Mag, vol. 26, no. 6, pp. 38–49, 2009.

60. G. Blake, R. G. Deslinski, T. Mudge, R. Chandra, L. Dagum, D. Kohr, D. Maydan, J. McDonald, R. Menon, H. Kim et al., "A survey of multicore processors: A review of their common attributes," IEEE Signal Processing Magazine, vol. 26, pp. 80–89.

61. J. Becker, A. Donlin, and M. Huebner, "New tool support and architectures in adaptive reconfigurable computing," pp. 134–139, 2007.

62. M. Alderighi, F. Casini, S. D'Angelo, M. Mancini, S. Pastore, and G. R. Sechi, "Evaluation of single event upset mitigation schemes for sram based fpgas using the flipper fault injection platform."

63. H. Wang, J. P. Delahaye, P. Leray, and J. Palicot, "Managing dynamic reconfiguration on mimo decoder," p. 197, 2007.

64. P. Sedcole, B. Blodget, J. Anderson, P. Lysaghi, and T. Becker, "Modular partial reconfigurable in virtex fpgas," pp. 211–216, 2005.

65. K. Wu and J. Madsen, "Run-time dynamic reconfiguration: a reality check based on fpga architectures from xilinx," pp. 192–195, 2005.

66. T. Pionteck, C. Albrecht, and R. Koch, "A dynamically reconfigurable packet-switched network-on-chip," p. 137, 2006.

67. Xilinx, JTRS SDR Kit, August 2006. [Online]. Available: http://www.xilinx.com/dsp/defense/jtrs_sdr_lounge.htm

68. A. Upegui and E. Sanchez, "Evolving hardware by dynamically reconfiguring xilinx fpgas." in ICES: From Biology to Hardware, 6th International Conference, ICES 2005, Sitges, Spain, September 12-14, 2005, Proceedings, ser. Lecture Notes in Computer Science, J. M. Moreno, J. Madrenas, and J. Cosp, Eds., vol. 3637. Springer, 2005, pp. 56–65.

69. C. Plessl, R. Enzler, H. Walder, J. Beutel, M. Platzner, and L. Thiele, "Reconfigurable hardware in wearable computing nodes," in Proc. Int. Symp. on Wearable Computers (ISWC'02). IEEE, October 2002, pp. 215–222. [Online]. Available: citeseer.ist.psu.edu/plessl02reconfigurable.html

70. Xilinx, "Virtex-5 configuration guide," may 2006. [Online]. Available: http://www.xilinx.com

71. ——, Virtex-4 Configuration Guide, 2006. [Online]. Available: http://www.xilinx.com

72. X. Corporation, "Xilinx website." [Online]. Available: http://www.xilinx.com

73. "Atmel at40k data sheet," Tech. Rep. [Online]. Available: http://www.atmel.com

74. N. McKay and S. Singh, "Dynamic specialisation of XC6200 FPGAs by partial evaluation," Field-Programmable Logic and Applications From FPGAs to Computing Paradigm, pp. 298–307, 1998.

75. S. Park and W. Burleson, "Reconfiguration for power saving in real-time motion estimation," in Acoustics, Speech and Signal Processing, 1998. Proceedings of the 1998 IEEE International Conference on, vol. 5. IEEE, 2002, pp. 3037–3040.

76. G. Brebner, "A virtual hardware operating system for the Xilinx XC6200," Field-Programmable Logic Smart Applications, New Paradigms and Compilers, pp. 327–336, 1996.

77. B. Blodget, P. James-Roxby, E. Kelle, S. McMillan, and P. Sundararajan, "A self-reconfiguring platform," FPL, Lisbon, Portugal, 2003. [Online]. Available: citeseer.ist.psu.edu/article/blodget03selfreconfiguring.html

78. E. Horta, J. Lockwood, D. Taylor, and D. Parlour, "Dynamic hardware plugins in an FPGA with partial run-time reconfiguration," in Proceedings of the 39th annual Design Automation Conference. ACM, 2002, pp. 343–348.

79. M. Ullmann, M. Hubner, B. Grimm, and J. Becker, "An FPGA run-time system for dynamical on-demand reconfiguration," in Parallel and Distributed Processing Symposium, 2004. Proceedings. 18th International, 2004, p. 135.

80. ——, "On-demand FPGA run-time system for dynamical reconfiguration with adaptive priorities," Field Programmable Logic and Application, pp. 454–463, 2004.

81. A. Donato, F. Ferrandi, M. Redaelli, M. Santambrogio, and D. Sciuto, "Caronte: a complete methodology to implement partially dynamically self-reconfiguring embedded systems on modern fpga," in IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM 2005), 2005.

82. Y. Qu and V. teknillinen tutkimuskeskus teknillinen tutkimuskeskus, System-level design and configuration management for run-time reconfigurable devices. Citeseer, 2007.

83. P. Merino, M. Jacome, and J. Lopez, "A methodology for task based partitioning and scheduling of dynamically reconfigurable systems," in FPGAs for Custom Computing Machines, 1998. Proceedings. IEEE Symposium on. IEEE, 2002, pp. 324–325.

84. J. Mignolet, V. Nollet, P. Coene, D. Verkest, S. Vernalde, and R. Lauwereins, "Infrastructure for design and management of relocatable tasks in a heterogeneous reconfigurable system-on-chip," in Proceedings of the conference on Design, Automation and Test in Europe-Volume 1. IEEE Computer Society, 2003, p. 10986.

85. G. Wigley, D. Kearney, and M. Jasiunas, "ReConfigME: a detailed implementation of an operating system for reconfigurable computing," in Parallel and Distributed Processing Symposium, 2006. IPDPS 2006. 20th International. IEEE, 2006, p. 8.

86. M. Santambrogio, V. Rana, and D. Sciuto, "Operating system support for online partial dynamic reconfiguration management," in Field Programmable Logic and Applications, 2008. FPL 2008. International Conference on. IEEE, 2008, pp. 455–458.

87. F. Redaelli, M. Santambrogio, and D. Sciuto, "Task scheduling with configuration prefetching and anti-fragmentation techniques on dynamically reconfigurable systems," 2008.

88. H. Walder, C. Steiger, and M. Platzner, "Fast online task placement on FPGAs: free space partitioning and 2D-hashing," in Parallel and Distributed Processing Symposium, 2003. Proceedings. International. IEEE, 2003, p. 8.

89. K. Bazargan, R. Kastner, and M. Sarrafzadeh, "Fast template placement for reconfigurable computing systems," IEEE Design & Test of Computers, vol. 17, no. 1, pp. 68–83, 2000.

90. F. Ferrandi, M. Morandi, M. Novati, M. Santambrogio, and D. Sciuto, "Dynamic reconfiguration: Core relocation via partial bitstreams filtering with minimal overhead," in System-on-Chip, 2006. International Symposium on. IEEE, 2007, pp. 1–4.

91. H. Kalte, G. Lee, M. Porrmann, and U. Ruckert, "Replica: A bitstream manipulation filter for module relocation in partial reconfigurable systems," in Parallel and Distributed Processing Symposium, 2005. Proceedings. 19th IEEE International. IEEE, 2005, p. 151b.

92. E. Horta and J. Lockwood, "PARBIT: a tool to transform bitfiles to implement partial reconfiguration of field programmable gate arrays (FPGAs)," in Proc. of FPL. Citeseer, 2002.

93. R. J. Fong, S. J. Harper, and P. M. Athanas, "A versatile framework for fpga field updates: An application of partial self-reconfiguation," RSP, p. 117, 2003.

94. C. Bolchini, D. Quarta, and M. Santambrogio, "SEU mitigation for SRAM-based FPGAs through dynamic partial reconfiguration," in Proceedings of the 17th ACM Great Lakes symposium on VLSI. ACM, 2007, pp. 55–60.

95. J. P. Delahaye, J. Palicot, C. Moy, and P. Leray, "Partial reconfiguration of fpgas for dynamical reconfiguration of a software radio platform," Mobile and Wireless Communications Summit, 2007. 16th IST, pp. 1–5, 2007.

96. P. Manet, D. Maufroid, L. Tosi, G. Gailliard, O. Mulertt, M. Di Ciano, J. Legat, D. Aulagnier, C. Gamrat, R. Liberati et al., "An evaluation of dynamic partial reconfiguration for signal and image processing in professional electronics applications," EURASIP Journal on Embedded Systems, vol. 2008, pp. 1–11, 2008.

97. "Isr technologies, inc." [Online]. Available: http://www.isr-technologies.com/

98. M. Uhm and J. Belzile, "Meeting software defined radio cost and power targets: Making sdr feasible," Military Embedded Systems, pp. 7–8, 2005.

99. M. Sarlotte, B. Counil, P. Gelineau, R. Chau, and D. Maufroid, "Partial reconfiguration concept in a sca approach," pp. 76 019–0016.

100. C. Kao, "Benefits of partial reconfiguration," Xilinx Xcell Journal, vol. 2005, no. 55, pp. 65–67, 2005.

101. X. Inc., "Xilinx xapp290: Two flows for partial reconfiguration: Module based or difference based," Tech. Rep., 2004.

102. P. Lysaght, B. Blodget, J. Mason, J. Young, and B. Bridgford, "Invited paper: enhanced architectures, design methodologies and cad tools for dynamic reconfiguration of xilinx fpgas," pp. 1–6, 2006.

103. F. Ghenassia, Transaction-level modeling with Systemc: TLM concepts and applications for embedded systems. Springer Verlag, 2005.

104. Y. Qu, K. Tiensyrja, J. P. Soininen, V. T. T. Electron, and F. Oulu, "Systemc-based design methodology for reconfigurable system-on-chip," pp. 364–371.

105. P. A. Hartmann, A. Schallenberg, F. Oppenhaimer, and W. Nebel, "Ossr+ r: Simulation and synthesis of selfadaptive systems," Proceedings of Field Programmable Logic and Applications (FPL2006), pp. 177–182, 2006.

106. A. Herrholz, E. Oppenheimer, P. A. Hartmann, A. Schallenberg, W. Nebel, C. Grimm, M. Damm, J. Haase, E. Brame, F. Herrera et al., "The andres project: analysis and design of run-time reconfigurable, heterogeneous systems," pp. 396–401, 2007.

107. G. Gailliard, B. Mercier, M. Sarlotte, B. Candaele, and F. Verdier, "Towards a systemc tlm based methodology for platform design and ip reuse: application to software defined radio," pp. 131–138.

108. G. Gailliard, E. Nicollet, M. Sarlotte, and F. Verdier, "Transaction level modelling of sca compliant software defined radio waveforms and platforms pim/psm," p. 971, 2007.

109. B. Rousseau, P. Manet, D. Galerin, D. Merkenbreack, J. D. Legat, F. Dedeken, and Y. Gabriel, "Enabling certification for dynamic partial reconfiguration using a minimal flow," p. 988, 2007.

110. I. Loiselle, B. Rousseau, P. Manet, H. Vandierendonck, and J. D. Legat, "Virtualization analysis of a reconfigurable region for adaptive dsp and multimedia applications," 2008.

111. X. Inc., "Xilinx xapp627: Picoblaze 8-bit microcontroller for virtex-ii series devices," Tech. Rep., 2003.

112. ——, "Xilinx coregen: Core generator 10.1 user guide," Tech. Rep., 2008.

113. R. D.-B. ED-12B, "Software considerations in airborne systems and equipment certification," Tech. Rep., 1992.

114. R. D.-E. ED-80, "Design assurance guidance for airborne electronic hardware," Tech. Rep., 2000.

115. Hispano-Suiza, "Fadec3, a new generation of full authority digital engine controls," 2005.

116. RTCA DO-254. Design Assurance Guidance for Airborne Electronic Hardware., April 2000.

117. RTCA-EUROCAE DO-178B/ED-12B. Software Considerations in Airborne Systems and Equipment Certification, December 1992.

118. A. Hilton and J. G. Hall, "Developing critical systems with pld components," in FMICS '05: Proceedings of the 10th international workshop on Formal methods for industrial critical systems. New York, NY, USA: ACM Press, 2005, pp. 72–79.

119. J. Ashley., Practical Guide to Certification and RE-Certification of AavA Software Elements. Software for Programmable Logic Devices., July 2005.

120. A. Kornecki, B. Butka, and J. Zalewski, "Software Tools for Safety-Critical Systems According to DO-254," Computer, vol. 41, no. 12, pp. 112–115, 2008.

121. A. Kornecki and J. Zalewski, "Hardware certification for real-time safety-critical systems: State of the art," Annual Reviews in Control, 2010.

122. M. Lee and T. Dewey, "Accelerating DO-254 for ASIC/FPGA Designs," VME and Critical Systems, pp. 28–30, 2007.

123. K. Karlsson and H. Forsberg, "Emerging verification methods for complex hardware in avionics," in Digital Avionics Systems Conference, 2005. DASC 2005. The 24th, vol. 1. IEEE, 2005, pp. 6–61.

124. P. Conmy and I. Bate, "Component-Based Safety Analysis of FPGAs," Industrial Informatics, IEEE Transactions on, vol. 6, no. 2, pp. 195–205, 2010.

125. I. Bate and P. Conmy, "Certification of FPGAs-Current Issues and Possible Solutions," Safety-Critical Systems: Problems, Process and Practice, pp. 149–165, 2009.

126. X. Inc., "Xapp015: Using the xc4000 readback capability," Tech. Rep.

127. K. Paulsson, M. Hubner, and J. Becker, "Exploitation of dynamic and partial hardware reconfiguration for on-line power/performance optimization," pp. 699–700, 2008.

128. K. Paulsson, M. Hubner, and J. Becker, "On-line optimization of FPGA power-dissipation by exploiting run-time adaption of communication primitives," in Proceedings of the 19th annual symposium on Integrated circuits and systems design. ACM, 2006, pp. 173–178.

129. K. Paulsson, M. Hubner, S. Bayar, and J. Becker, "exploitation of run-time partial reconfiguration for dynamic power management in Xilinx Spartan III-based systems," ReCoSoc2007, Montpellier, France, 2007.

130. M. Lorenz, L. Mengibar, M. Valderas, and L. Entrena, "Power Consumption Reduction Through Dynamic Reconfiguration," Field Programmable Logic and Application, pp. 751–760, 2004.

131. J. Becker, M. Huebner, and M. Ullmann, "Power estimation and power measurement of Xilinx Virtex FPGAs: trade-offs and limitations," in Integrated Circuits and Systems Design, 2003. SBCCI 2003. Proceedings. 16th Symposium on. IEEE, 2003, pp. 283–288.

132. X. Inc., "Logicore ip on-chip peripheral bus v2.0 with opb arbiter," Tech. Rep., 2010.

133. J. Noguera and I. Kennedy, "Power reduction in network equipment through adaptive partial reconfiguration," in Field Programmable Logic and Applications, 2007. FPL 2007. International Conference on. IEEE, 2007, pp. 240–245.

134. Micron, "Calculating memory system power for ddr sdram," Tech. Rep., 2001.

135. ——, "Mt46v16m16fg-6 ddr sdram features datasheet," Tech. Rep.

136. ——, "Mt55l256v32pf zbt sram features datasheet," Tech. Rep.

137. R. Baines and D. Pulley, "The picoarray and reconfigurable baseband processing for wireless basestations," Software Defined Radio, 2004.

138. M. Butts, A. M. Jones, and P. Wasson, "A structural object programming model, architecture, chip and tools for reconfigurable computing," pp. 55–64, 2007.

139. A. Duller, D. Towner, G. Panesar, A. Gray, and W. Robbins, "picoarray technology: the tool's story," in Design, Automation and Test in Europe, 2005. Proceedings, March 2005, pp. 106–111.

140. K. van Berkel, P. Meuwissen, N. Engin, and S. Balakrishnan, "Cvp: A programmable co vector processor for 3g mobile baseband processing," 2003.

141. M. J. Flynn and P. Hung, "Microprocessor design issues: thoughts on the road ahead," IEEE Micro, vol. 25, no. 3, pp. 16–31, 2005.

142. D. Kissler, F. Hannig, A. Kupriyanov, and J. Teich, "A dynamically reconfigurable weakly programmable processor array architecture template," in International Workshop on Reconfigurable Communication Centric System-on-Chips (ReCoSoC), 2006, pp. 31–37.

143. A. Richard, A. Vander Biest, A. Bartzas, A. Papanikolaou, D. Soudris, D. Milojevic, and F. Robert, "A multi-criteria estimation tool for system-on-chip."

144. G. Palermo, C. Silvano, and V. Zaccaria, "An efficient design space exploration methodology for multiprocessor soc architectures based on response surface methods," pp. 150–157, 2008.

145. G. Goossens, D. Lanneer, W. Geurts, and J. Van Praet, "Design of asips in multiprocessor socs using the chess/checkers retargetable tool suite," pp. 1–4, 2006.

146. "Coware/lisatek." [Online]. Available: http://www.coware.com

147. A. Halambi, P. Grun, V. Ganesh, A. Khare, N. Dutt, and A. Nicolau, "Expression: A language for architecture exploration through compiler/simulator retargetability," pp. 31–45, 1999.

148. O. Schliebusch, A. Chattopadhyay, R. Leupers, G. Ascheid, H. Meyr, M. Steinert, G. Braun, and A. Nohl, "Rtl processor synthesis for architecture exploration and implementation," p. 30156, 2004.

149. "802.11a-1999 high-speed physical layer in the 5 ghz band," Tech. Rep., February 1999.

150. H. Takamura, K. Inoue, and V. Moshnyaga, "Register file energy reduction by operand data reuse," Integrated Circuit Design. Power and Timing Modeling, Optimization and Simulation, pp. 289–306.

151. M. Müller, S. Simon, H. Gryska, A. Wortmann, and S. Buch, "Low power synthesizable register files for processor and ip cores," Integration, the VLSI Journal, vol. 39, no. 2, pp. 131–155, 2006.

152. S. Pasricha and N. Dutt, On-Chip Communication Architectures: System on Chip Interconnect. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2008.

153. G. D. Micheli and L. Benini, Networks on Chips: Technology and Tools (Systems on Silicon). San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2006.