Quality-driven model-based design of multi-processor accelerators : an application to LDPC decoders

Quality-driven Model-based Design of Multi-processor Accelerators

An Application to LDPC Decoders

PROEFSCHRIFT

ter verkrijging van de graad van doctor
aan de Technische Universiteit Eindhoven, op gezag van de
rector magnificus, prof.dr.ir. C.J. van Duijn, voor een
commissie aangewezen door het College voor
Promoties in het openbaar te verdedigen
op dinsdag 29 mei 2012 om 16.00 uur

door

Yahya Jan

geboren te Charsadda, Pakistan

Dit proefschrift is goedgekeurd door de promotor:

prof.dr. J. Pineda de Gyvez

Copromotor:
dr. L. Jóźwiak

Quality-driven Model-based Design of Multi-processor Accelerators
An Application to LDPC Decoders
/ by Yahya Jan. - Eindhoven : Eindhoven University of Technology, 2012.
A catalogue record is available from the Eindhoven University of Technology Library
ISBN 978-90-386-3136-3
NUR 959
Keywords: embedded systems / electronic design automation /
highly-demanding applications / multi-processor accelerators /
massively parallel systems.

Quality-driven Model-based Design of Multi-processor Accelerators

An Application to LDPC Decoders

Committee:

prof.dr. J. Pineda de Gyvez (promotor, TU Eindhoven)
dr. L. Jóźwiak (co-promotor, TU Eindhoven)
prof.dr.ir. R.H.J.M. Otten (TU Eindhoven)
prof.dr. K. Bertels (TU Delft)
dr. M. Duranton (Institute CARNOT CEA LIST France)
dr.ir. T.J. Tjalkens (TU Eindhoven)

The work reported in this thesis is supported by HEC (Higher Education Commission Pakistan) in collaboration with NUFFIC (Netherlands organization for international cooperation in higher education) and Eindhoven University of Technology.

Cover Design: Edwin van Drunen
Printing: Printservice Eindhoven University of Technology

Abstract

The recent spectacular progress in nano-electronic technology has enabled the implementation of very complex multi-processor systems on single chips (MPSoCs). However in parallel, new highly-demanding complex embedded applications are emerging, in fields like communication and networking, multimedia, medical instrumentation, monitoring and control, military, etc., which impose stringent and continuously increasing functional and parametric demands. The high demands of these applications cannot be satisfied by systems implemented on general purpose processors (GPP). For these applications increasingly complex and highly optimized application-specific MPSoCs are required to perform real-time computations to extremely tight schedules, when satisfying high demands regarding the energy, area, cost and development efficiency. High-quality MPSoCs for these highly-demanding applications can only be constructed through adequate usage of efficient application-specific system architectures exploiting adequate concepts of computation, storage and communication, as well as, usage of efficient design methods and electronic design automation (EDA) tools for synthesizing the actual high-quality hardware platforms implementing the architectures.

Some of the representative examples of these highly-demanding applications include the based-band processing in wired/wireless communication (e.g. the upcoming 4G wireless systems), different kinds of encoding/decoding in communication, image processing and multimedia, 3D graphics, ultra-high-definition television (UHDTV), and encryption applications, etc. These applications require to perform complex computations with a very high throughput, while at the same time demanding low energy and low cost. The decoders of the low density parity check (LDPC) codes, adopted as an advance error-correcting scheme in the newest wired/wireless communication standards, like IEEE 802.11n, 802.16e/m, 802.15.3c, 802.3an, etc., for applications as digital TV broadcasting, mm-wave WPAN, etc., can serve as a representative example of such applications. These standards, for instance the IEEE 802.15.3c, specify as high as 5∼6 Gbps throughput for the upcoming wireless communication systems. For the realization of the throughput as high as several Gbps, massively parallel multi-processor accelera-

tors are indispensable.

These modern highly-demanding applications involve massive parallelism of various kinds (e.g. task, data and functional, etc) and complex interrelationships among the data and computing operations. Therefore, an adequate accelerator design for such highly-demanding applications requires a careful exploration and exploitation of various kinds of parallelism and the resolution of complex interrelationships between the data and computing operations. More precisely, the accelerator design for such kind of applications has to involve adequately the combined micro- and macro-architecture design for the processors, and the corresponding adequate memory and communication architectures design. Since the processor's micro-/macro-architecture and the memory and communication architectures are strongly interrelated and cannot be designed in separation, complex mutual tradeoffs have to be resolved among the processor parallelism at the two levels, and the corresponding memory and communication architectures, as well as, among the performance, power and area. For the design of hardware accelerators, high-level-synthesis (HLS) methods and tools are often used. However, the HLS methods and tools only support the micro-architecture synthesis of a single processing unit, while not taking into account the macro-architecture, memory and communication synthesis and not accounting for the relationships and tradeoffs among these design aspects, which is necessary in the design of hardware accelerators for highly-demanding applications.

To address the issues highlighted above and to resolve the mutual tradeoffs effectively and efficiently, a novel quality-driven model-based hardware multi-processor design methodology and the related design space exploration (DSE) approach is proposed in this thesis that jointly consider the processor, memory and communication architectures, and the possible mutual tradeoffs among them.

For the ultra-high throughput requirements that demand massively parallel hardware multi-processor architectures, the communication and memory have usually a dominating influence on all the most important design aspects such as delay, area and power. The additional performance gains expected from an increased parallelism will end up in diminished returns, when exploding the interconnect or memory complexity. Although some research results on the memory and communication architectures can be found in the literature, these results are for programmable on-chip multi-processor systems that utilize the time-shared communication resources, such as shared buses or Network on Chip (NoC). Such communication resources are however not adequate to deliver the data transfer bandwidth required for the massively parallel multi-processor accelerators. Therefore, in the research reported in this thesis, several promising generic scalable communication and memory architectures are proposed that satisfy the required data transfer bandwidth of the high-end multi-processor accelerators. The proposed generic hierarchical partitioned communication and memory architectures ensure the scalability when applied for massively parallel hardware multi-processor accelerators.

The proposed design methodology makes it possible to perform an effective

and efficient exploration and exploitation of the various tradeoffs between the processing parallelism at the micro- and macro-architecture level and the corresponding memory and communication architectures, as well as, among the performance, power and area, to arrive at high-quality accelerator architectures. Several novel scheduling, processing parallelism exploration, and the memory and communication architecture exploration strategies are incorporated into the proposed architecture design space exploration framework.

To analyze and evaluate the proposed design methodology and its related design space exploration framework, a series of extensive case studies are performed through implementing and applying the methodology for industrial-strength applications of the LDPC decoding for the latest communication system standards. These case studies involved extensive architecture synthesis experiments with the LDPC decoder designs for IEEE 802.15.3c LDPC codes. In particular, the results of the experiments clearly demonstrate that neither the fully-serial nor the fully-parallel micro-architectures are adequate to satisfy the ultra-high performance requirements. The extreme fully-serial and fully-parallel micro-architectures are also not appropriate from the viewpoint of the area and power consumption. To satisfy the ultra-high performance or ultra-low power requirements, the combined micro-/macro-architecture exploration is necessary which explores and exploits various partially-parallel architecture combinations.

The results of the experiments confirmed that without considering the processor's micro- and macro-architecture design, as well as, the communication and memory architecture design in combination, it is very difficult to arrive at an adequate high-quality accelerator. They confirmed that the proposed design methodology adequately supports the design of complex multi-processor hardware accelerators, while taking into account the numerous complex tradeoffs.

To our knowledge, despite a more than a decade of research on the hardware accelerators for the highly-demanding applications, no similar holistic quality-driven design approach has been proposed. In the proposed design method, all the design components are taken jointly as a single design task and the mutual tradeoffs among them, as well as, among different design objectives are considered. Finally, using the method, it is possible to implement various high-quality complex multi-processor hardware accelerators for the highly-demanding applications (e.g. LDPC decoders of practical importance for the newest upcoming wireless communication standards) very quickly.

Acknowledgments

There are many people who contributed directly or indirectly to the work reported in this thesis and I would like to express my gratitude to all of them.

First of all, I would like to thank Prof. Jose Pineda de Gyvez, my promoter. His detailed reviews and constructive remarks significantly improved the quality and overall organization of this thesis.

Next, I would like to thank my co-promoter and direct supervisor Dr. Lech Jóźwiak who provided me the opportunity to perform research in the electronic systems group. His guidance and consistent involvement in all phases of my PhD research project is truly remarkable. During my PhD, we had many and often long technical meetings that helped me a lot in my work. His way of thinking, perception of things and his quality concepts have much improved my theoretical knowledge and understanding of systems. He also helped me in resolving several financial and other issues.

I would also like to thank all the members of my PhD promotion committee for their time and feedback on my thesis. My special thanks go to Dr. Marc Duranton and Dr. Tjalling J. Tjalkins for their collaboration and support during the initial phase of my PhD project.

It was a great and pleasant experience for me to work in the electronic system group. It took me almost no time to settle down in the working environment of the group. I enjoyed my work and the company of my colleagues. I would like to thank them all. I would like to thank Prof. Ralph Otten for accepting me in the group, and Ahsan Shabbir for his help on many issues both technical and non-technical, when we were together in the electronic systems group. Special thanks to our secretaries Marja and Rian for their guidance regarding all administrative issues, filling out forms written in Dutch, and for their always happy and friendly attitude.

I would like to thank my family, my mother Sarhad Bibi, my brothers Daud Jan, Ayub Jan, Easa Khan, and my sister Nighat Ara. I believe that without their support and encouragement, it would be difficult to achieve this result. Many many thanks to Easa Khan for his support, company and visits to Netherlands, when he was studying in Italy. I regret for being away from my family on two important occasions. First, the wedding ceremony of my sister in 2008, and then another important event in my family, the wedding ceremony of my two elder brothers, Daud Jan and Ayub Jan in early 2012. It was very hard for them to accept my absence, especially for my mother, no words to express. Therefore, I dedicate this thesis to my family including the cute Hassan Ahmad, son of my sister.

I would like to thank my friends from Netherlands. Special thanks go to Edwin and Ilinca for helping me in many ways. We spent many weekends together, whether it was cooking a meal together, watching a movie, going on biking trips. I never felt alone in Eindhoven due to their company, God bless them. I would also like to thank Karen and her family having lived with them for a long time. Finally, I would like to thank all of my Pakistani friends living in Netherlands. Especially, I would like to thank all of my friends living in Delft for entertaining me many times during holiday visits to Delft. I am grateful for the exceptional reception. A fantastic time with all of them, *thanks to all.*

Yahya Jan

Contents

CHAPTER 1

Introduction

During the last decade, a big progress and remarkable innovations have been made in various embedded system fields, like telecommunications, multimedia, consumer electronics, medical instrumentation, avionics, navigation and transportation, etc. Many new modern ways of information processing, communication and presentation have emerged. For example, the communications systems and standards that were originally developed for voice services have evolved to support extensive video, data and multi-media services with a high availability, reliability, speed and quality-of-service (QoS), as shown in Figure 1.1. Similarly, in the entertainment domain, the video processing/broadcasting systems of the past have been replaced by the current High-Definition (HD) systems and are in a transition-phase to the ultra-high-definition systems (UHDTV) with a rich set of new and innovative services such as video-on-demand (VoD) and youtube, etc, for example. The future wireless video local area network (WVLAN) can also serve as a very good example of such an advanced application with a mix of substantial innovation in video and communication, as shown in Figure 1.2. In the WVLAN all the video sources such as HD disc player, set-top boxes, audio/video (A/V) receiver, etc, communicate with the coordinator (digital display system). This communication involves exchange of huge video contents at extremely high speed among the different video sources and the display system. Also, the different video sources such as HD disc player, set-top boxes, etc, require to process (encoding/decoding) huge amounts of video data at extremely high speeds. All these innovations were possible due to the tremendous advancements in the nano-electronic and information technologies during the last decade. However, what

still needs improvements are the adequate design methods and electronic design automation (EDA) tools to effectively and efficiently design such kinds of complex and highly-demanding systems. This thesis addresses this need.

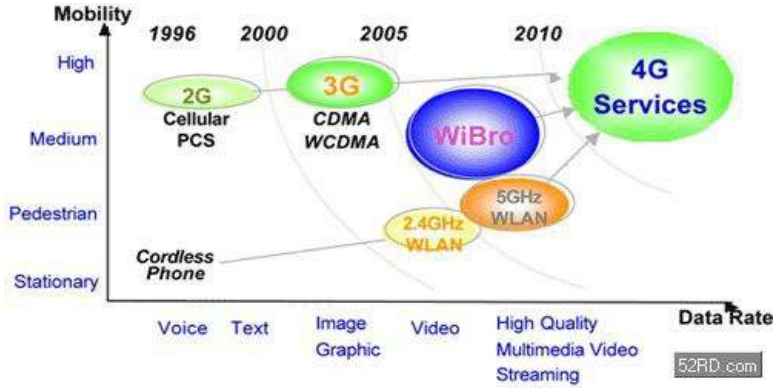

Figure 1.1: Example of innovation in communications systems

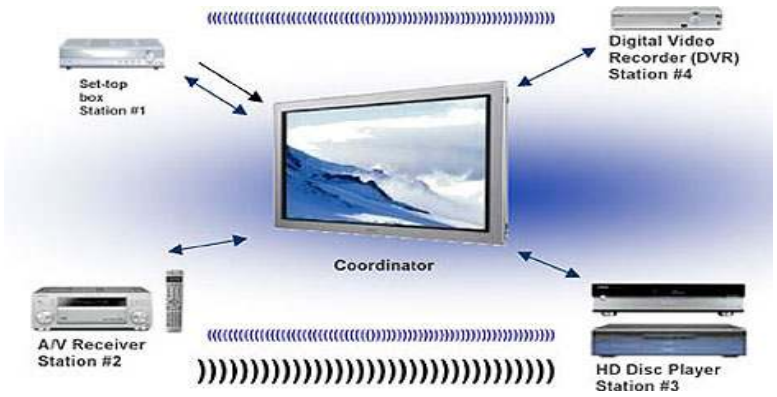

Figure 1.2: Example of wireless video LAN (WVLAN) for HD video applications (image: https://www.wirelesshd.org/)

The rest of this chapter is organized as follows. Section 1.1 discusses the highly-demanding applications. Section 1.2 briefly discusses the design, automation and implementation challenges in the design of hardware accelerators for the highly-demanding modern applications as those briefly considered above. Section 1.3 introduces the aim of the research reported in this thesis and lists the key contributions of this research. Section 1.4 presents the overall thesis organization.

1.1 Modern Highly-demanding Applications

The recent spectacular progress in nano-electronic technology has enabled the implementation of very complex multi-processor systems on single chips (MPSoCs) for high-performance (embedded) applications. On the other hand, however, new highly-demanding complex embedded applications are emerging, in fields like communication and networking, multimedia, medical instrumentation, monitoring and control, military, etc., which impose stringent and continuously increasing functional and parametric demands. The high demands of these applications cannot be satisfied by systems implemented on general purpose processors (GPP). For these highly-demanding applications increasingly complex and highly optimized application-specific MPSoCs are required. They have to perform real-time computations to extremely tight schedules, when satisfying high demands regarding the energy, area, cost and development efficiency, and often to be highly flexible to support different product versions, quickly changing standards, adaptability to changing operation conditions, design reuse and computational sharing, development and fabrication effort re-use, etc. High-quality MPSoCs for these highly-demanding applications can only be constructed through adequate usage of efficient application-specific system architectures and circuit implementations exploiting adequate concepts of computation, storage and communication, as well as, usage of efficient design methods and electronic design automation (EDA) tools for synthesizing the actual high-quality hardware platforms implementing the architectures, and for efficient mapping of applications onto the hardware platforms [1].

Some of the representative examples of these highly-demanding applications include the based-band processing in wired/wireless communication (e.g. the upcoming 4G wireless systems), different kinds of encoding/decoding in communication, image processing and multimedia, 3D graphics, ultra-high-definition television (UHDTV), and encryption applications, etc. These applications require to perform complex computations with a very high throughput, while at the same time demanding low energy and low cost. Moreover, they often require design-time and/or run-time (re-)configurability to support the new evolving and competing standards, and different data transmission rates. The decoders of the low density parity check (LDPC) codes [2], adopted as an advance error-correcting scheme in the newest wired/wireless communication standards, like IEEE 802.11n, 802.16e/m, 802.15.3c, 802.3an, etc., for applications as digital TV broadcasting, mm-wave WPAN [3], etc., can serve as a representative example of such applications. These standards specify ultra-high throughput figures in the range of Gbps [3]. Such ultra-high performance cannot be achieved using general purpose processors (GPPs), digital signal processors (DSPs) [4] or general purpose graphic processing units (GPGPUs) [5]. For example, an implementation of LDPC decoders on the famous Texas Instruments TMS320C64xx fixed point DSP processor running at 600 MHz delivers a throughput of only 5 Mbps

[4]. Similarly, implementations of LDPC decoders on the multi-core architectures, results in throughputs in the order of 1∼2 Mbps on the general-purpose x86 multi-cores, and ranging from 40 Mbps on the GPU to nearly 70 Mbps on the CELL Broadband Engine (CELL/B.E) as reported in [5]. For the realization of the throughput as high as several Gbps, while also remaining in the energy budget, massively parallel multi-processor accelerators are indispensable. Even for substantially lower performance levels the programmable processor based solutions are usually inadequate from the viewpoint of energy consumption.

Real-time high-end video processing (encoding/decoding) is another interesting and representative example of such demanding applications. To satisfy the real-time performance demands of many modern multimedia applications, like: video conferencing, video telephony, camcoders, surveillance, medical imaging, and especially HDTV and new emerging UHDTV in video broadcasting domain, ultra-high performance computational platforms are required. The problem is amplified by the quickly growing requirements of higher and higher quality, especially in the video broadcast domain, what results in a huge amount of data processing for the new standards of digital TV, like UHDTV that requires a resolution of (7680x4320)∼ 33Megapixel with a data rate of 24 Gbps [6]. Additionally, the latest standards video coding algorithms are much more complex due to the digital multimedia convergence and specifically access of multimedia through a variety of networks and different coding formats used by a single device, as well as, the slow vanishing of the old video coding standards (e.g. MPEG-2) and widespread adaptation of the new standards (e.g. H.264/AVC, VC1 etc). Often, the computational platforms for multimedia are also required to be (re-)configurable, to enable their adaptation to various domains, accessing networks, standards and work modes. Application-specific multi-processors are required to constitute the kernels of such (re-)configurable high-performance platforms. The Context-based Adaptive Binary Arithmetic Coding (CABAC), entropy encoder/decoder of H.264/AVC encoder/decoder can serve here as a one more example. Its purely software-based implementation on a general purpose computing engine results in an unsatisfactory performance even for a quite low quality and resolution (e.g. 30-40 cycles are required on average for a single bin decoding on DSP [7]). The situation is much worse for the High Definition (HD) video as the maximum bin rate requirement of HD (level 3.1 to 4.2) in H.264/AVC, averaged across a coded picture, ranges from 121 Mbins/s to 1.12 Gbins/s [8]. This makes the software solution inadequate to achieve the real-time performance for HD video as a multi-giga hertz general purpose processor would be required for HD encoding in real-time [9]. Moreover, the serial nature of CABAC paralyzes the other processes in video codec that could otherwise be performed in parallel, making CABAC a bottleneck in the overall codec performance. Consequently, to achieve the required performance, flexibility, low cost and low energy consumption, a sophisticated (re-)configurable hardware accelerator for CABAC is necessary.

Above, several examples are given of the high-end applications and their high-performance demands, as well as, the performances achieved for them on the

state-of-the-art programmable processors. There are many more such applications in various fields that require hardware acceleration to satisfy their high real-time performance demands. The orientation scores (OS) based image analysis in medical imaging field is another example of such demanding applications [10]. They involve massively parallel complex image orientation (filtering) and enhancement kernels for image analysis. The filters are the well-known convolution filters that extract some image features utilizing the neighboring pixel information. The unique feature of the filtering operation is the large filter kernel sizes involved in the computation of a single pixel. To adequately satisfy their performance demands, parallelism have to be exploited on a massive scale. For instance, the convolution filter (kernel size of 15×15) of the orientation scores algorithm requires to process pixels at a rate of 125 MPixel/s with 8-bits/Pixel results in the throughput of 1 Gbps. The convolution filter requires 225 multiplications and 224 additions per clock cycles to compute a single pixel of a single orientation for a kernel of size 15×15. The total number of orientations to investigate is from 12 to 32 and the kernel sizes in the range from 9 to 41. This requires a huge amount of computational resources to compute all the orientations in real-time. Other similar kind of operations are the various kinds of transforms such as fast fourier transform (FFT), discrete cosine transform (DCT), inverse discrete cosine transform (IDCT) and operations on matrix (multiplication, addition), etc, that are used extensively in various application fields, such as DCT/IDCT in H.264/AVC video encoder/decoder. Depending on the application, these operations are implemented with adequate parallelism in hardware at different performance points. Yet another application area where hardware accelerator is often required is the area of encryption applications.

The highly-demanding applications, as those briefly discussed above, may have different computation characteristics and requirements. The next section discusses the features, implementation challenges, and the kind of design approach needed to effectively and efficiently design high-performance hardware accelerators for such applications.

1.2 Main Challenges of Accelerators Design for Highly-demanding Applications

In the previous section, several application classes are discussed that require hardware acceleration, because their processing speed requirements are beyond the capabilities of today's programmable processors. In this section, the accelerator design and implementation challenges for the highly-demanding applications are briefly analyzed. As outlined in the previous section, the high physical and economic demands of many modern applications result in many different design issues and challenges that have to be adequately addressed during the accelerator design process.

Although all these applications require hardware acceleration to perform their computations with the required speed, they can be quite different regarding the character of computation and communication involved, and some other features.

For example, consider the transform and filtering based applications. All these applications have in common that they mainly involve functional parallelism and a simple local or regular communication. They either do not require (global) memory accesses, because they directly process the incoming stream of data, or they require relatively simple and regular, limited in space and time local memory accesses between which relatively large portions of computations are performed. The design of hardware accelerators for such kind of applications that do not involve complex communication or complex irregular memory accesses is usually limited to the micro-architecture RTL-level exploration and synthesis, and is reasonably supported by the existing methods of "application analysis and parallelization" (APP) and "high-level-synthesis" (HLS) [11–19], and the new emerging commercial HLS tools [20–26]. The HLS tools are often able to automatically produce a reasonable RTL-level representation of the required hardware from a high-level-language (HLL) design description, such as C/C++, SystemC, MATLAB, etc. The micro-architecture of such a processing unit is composed of interconnected register-transfer-level (RTL) data-path resources (such as, adders, multipliers, registers and multiplexers, etc) and a controller.

The limitation of HLS to only the micro-architecture exploration and synthesis of a single processing unit, as well as, some other limitations cause that it alone is inadequate for the design of high-performance complex multi-processor hardware accelerators for many modern applications, specifically applications involving complex relationships between the data and computing operations, as considered in this thesis (see Chapter 2 for details). The salient computation, storage and communication character of these applications in addition to the ultra-high performance, lead to many new challenges that have to be addressed, when designing hardware accelerators for these applications. The application features and related design challenges are briefly discussed below.

Many modern applications involve massive parallelism of various kinds, such as task, data and functional parallelism, as well as, complex interrelationships among the data and computing operations at the task level and complex inter-task data dependencies. To satisfy the ultra-high performance of these applications, parallelism has to be exploited on a massive scale. Therefore, to adequately serve these applications, the accelerator design for such applications require a careful exploration and exploitation of various kinds of parallelism and the resolution of complex interrelationships between the data and computing operations. In particular, the accelerator design for such kind of applications has to account for both the micro- and macro-architecture design for processors, and for the corresponding adequate memory and communication architectures design. Moreover, the processor's micro- and macro-architecture and the memory and communication architectures are strongly interrelated and cannot be designed in separation. Complex mutual tradeoffs have to be resolved among the processor parallelism at the

two levels, i.e. between the micro- and macro-architecture, and the corresponding memory and communication architectures, as well as, among the performance, power consumption and area. However, the existing HLS methods [11–19] and tools [20–26] as discussed earlier are limited in their scope to the synthesis of a single processing unit. They do not support the total complex design process of such high-end multi-processor accelerators that involves the combined exploration and synthesis of the adequate micro-/macro-architectures for processors, and the corresponding memory and communication architectures, as well as an adequate resolution of the mutual tradeoffs among these design aspects. New architecture design methods and supporting EDA tools are necessary to address the above challenges and to adequately support the design process of such high-end multi-processor accelerators.

The second major challenge is related to the bandwidth and scalability of the communication and memory architectures of the massively parallel hardware multi-processors that are necessary for the implementation of highly-demanding applications. For the massively parallel hardware multi-processors, the traditionally used flat communication architectures and multi-port memories do not scale well, and the memory and communication network influence on both the throughput and circuit area dominates the processors influence. The additional performance gains expected from an increased parallelism will end up in diminished returns, while exploding the communication or memory complexity. Although some research works related to the memory and communication architectures can be found in the literature [27–33] in the context of programmable on-chip multi-processor systems, the memory and communication architectures were proposed there for the much larger and much slower programmable processors (see Chapter 6 for details). They are not adequate for the small and ultra-fast hardware processors of the massively parallel multi-processor accelerators, due to a much too low bandwidth and scalability issues. These issues being of crucial importance for the massively parallel multi-processor accelerators are not adequately addressed by the state-of-the-art research [27–33] on the memory and communications architectures.

Moreover, similar applications can target different market segments imposing different requirements. For instance, let us consider the video decoding application. The same kind of video decoding can be used in a low power and area mobile device or a laptop, or a high-definition television (HDTV) from low through moderate to extremely high performance levels. Therefore, to quickly arrive at an adequate design quality, for a particular application with all its particular constraints, objectives and tradeoff profiles, an adequate multi-objective architecture design space exploration (DSE) is indispensable taking into account the various design objectives, constraints and tradeoffs. This is not supported by the current accelerator design methods [11–19] and tools [20–26], as they usually accept a few constraints, like only the clock speed constraint.

When this research was started, no adequate design methodology was in place for such kind of accelerators for the highly-demanding applications. Lack of an

adequate design methodology resulted in a large number of ad-hoc proposed solutions in the form of various particular ad-hoc point architectures for various problem instances with different throughput requirements. For instance, despite a more than a decade of research on the hardware accelerators for the LDPC decoding, only some partial proposals from a fragmented research on the processor, memory or communication part of the total accelerator design are available or ad-hoc implementations[1] for specific requirements.

The research work reported in this thesis provides solutions to the discussed above serious issues in the design of hardware multi-processor accelerators for highly-demanding applications. In the next section, the main aim of the research reported in this thesis is presented and the summary of its contributions.

1.3 The Aim and Key Contributions of this Thesis

The general aim of the research reported in this thesis was to analyze the issues and requirements of hardware accelerator design for modern highly-demanding applications, as well as, to propose, implement, analyze and evaluate an adequate semi-automatic design method addressing the issues and satisfying the requirements.

In the previous section, some of the main hardware accelerators design challenges for highly-demanding applications are briefly discussed. Here, the key contributions of the research reported in this thesis are listed. Addressing these design challenges requires contributions in both the design methods and EDA tools, as well as the architecture design. Additionally, some contributions are made towards a better understanding of design problems for highly-demanding applications. In the following, some of the major contributions are discussed in each sub-category.

1.3.1 Design Methods and EDA Tools

The contributions to this area are in the form of a new architecture design method and tool to address the design challenges of the accelerator design for highly-demanding applications that are not addressed by the previous research efforts. The contributions are as follows:

- analysis of several different highly-demanding modern applications and the design issues related to these applications; specifically, of the issues that can not be resolved using the traditional methods for design of hardware accelerator architectures
- formulation of requirements for an adequate hardware accelerator design method for the highly-demanding applications based on the results of the analysis

[1] 100∼150 research articles on LDPC decoder implementation up till now

- proposal of a novel multi-processor design method that adequately addresses the issues and satisfies the requirements, and is based on the quality-driven model-based system design approach proposed by Jóźwiak [1, 34–37]
- proposal of a novel multi-objective and multi-dimensional design space exploration (DSE) framework that performs the exploration and exploitation of various tradeoffs between the processing parallelism at the micro- and macro-architecture level, and the corresponding memory and communication architectures, as well as, among the performance, power consumption and area (PPA), to quickly arrive at high-quality accelerator architectures
- two novel operation scheduling techniques, the so-called tight scheduling (TS) and relaxed scheduling (RS), that are proposed to tradeoff the processors cost against the memory and communication structure costs based on whether the processors costs dominates or the memory and communication costs, respectively
- two novel performance and power optimization approaches based on the consideration of the micro-/macro-architecture level parallelism in combination; these approaches are much better than the traditional power and performance optimization methods for multi-processor systems
- application of the proposed multi-processor accelerator design method and implementation of the related DSE framework with all its core activities for the application field of LDPC decoders as a case study to evaluate the proposed design method
- semi-automatic design of numerous different LDPC decoder architectures and exploration of the tradeoffs among the various architectures for the newest communication system standards using the design method, and the related automatic DSE framework

1.3.2 Architecture Design

The second set of main contributions is the architecture scalability problem analysis and design of scalable memory and communication architectures for the massively parallel hardware multi-processor systems. The memory and communication architectures proposed in the past [27–33] for programmable on-chip multiprocessors systems are not adequate for the massively parallel hardware multiprocessor accelerators due to the data transfer bandwidth and scalability issues. Therefore, several novel application-specific memory and communication architectures are proposed in this thesis. The contributions to architecture design are as follows:

- analysis of the communication and memory architecture design and scalability issues for the massively parallel hardware multi-processors

- proposal of several novel generic hierarchical partitioned communication and memory architectures that ensure the scalability when applied to massively parallel hardware multi-processors, as well as, their application method to the memory and communication architecture design of massively parallel LDPC decoders
- design and implementation of novel generic check node processor (CNP) and variable node processor (VNP) micro-architectures for the LDPC decoders that span the full range of micro-architectures from the fully-serial, through the partially-parallel, to the fully-parallel
- a novel classification of the multi-processor architectures based on the processing parallelism to evaluate and compare the various architectural solutions

1.3.3 Better Understanding of Architecture Design Problems for Highly-demanding Applications

Using the proposed multi-processor accelerator design method and its application for a real-life application of LDPC decoding, a large series of architecture synthesis experiments were performed that not only confirmed adequacy of the design method, but also resulted in several findings of high importance for better understanding of architecture design problems for highly-demanding applications. The most important of them are listed below.

- demonstration of the numerous mutual complex tradeoffs between the micro- and macro-architecture levels and their influence on design parameters like, performance, power consumption and area, etc, to make adequate decisions on the number and type of processors
- demonstration that neither the fully-serial nor the fully-parallel micro-architectures are adequate to satisfy the ultra-high performance or low power requirements, but the partially-parallel architectures have to be explored and exploited
- demonstration that the ultra-high performance and low power requirements can only be satisfied on a reasonable cost through the combined micro-/macro-architecture, as well as, communication and memory architecture exploration
- demonstration that pipelining for performance enhancement is not the best approach in many cases, and may specifically not be suitable for the architectures exploiting massive parallelism at both architecture levels

To our knowledge, despite more than a decade of research on the hardware accelerator design, no similar holistic quality-driven design approach has been proposed.

In the design approach proposed in the scope of the research reported in this thesis, all the design components are considered in combination, their combined design forms a single design task, and the mutual interrelationships and tradeoffs among them and their various characteristics are analyzed and adequately resolved. Finally, using this method, it is possible to implement various complex multi-processor hardware accelerators for the highly-demanding applications (e.g. LDPC decoders of practical importance for the newest upcoming wireless communication standards) very effectively and efficiently, and particularly, very quickly. We hope that the original effort that resulted in this new design methodology for massively parallel multi-processor accelerators will act as a stepping stone on the way towards better well-founded design methods and EDA tools for the multi-processor system design.

1.4 Thesis Organization

The thesis is organized as follows. **Chapter 2** provides a detailed analysis of the issues and challenges in the design of hardware accelerators for highly-demanding applications. Based on the analysis, the requirements are formulated that have to be satisfied by an adequate accelerator design methodology. The main concepts of the proposed design approach are discussed. Moreover, this chapter includes a comprehensive overview of the existing research work performed in the scope of hardware accelerators. **Chapter 3** presents a novel quality-driven model-based multi-processor accelerator design methodology. The design flow and core steps of the proposed methodology are explained. Furthermore, a novel multi-objective and multi-dimensional design space exploration framework and its algorithms are discussed in detail. **Chapter 4** discusses the implementation of the proposed design methodology for the LDPC decoder applications used for the case studies. Through the case studies, the design methodology and its design space exploration framework are analyzed and evaluated. **Chapter 5** discusses the parallelism exploration and exploitation at the micro-/macro-architecture level and the mutual tradeoffs between the micro- and macro-architecture, when using as a representative example the design of LDPC decoders. Some interesting tradeoffs among performance, power consumption and area are highlighted that are the consequences of the joint consideration of the micro-/macro-architecture level parallelism. **Chapter 6** discusses the memory and communication issues involved in the design of massively parallel multi-processor accelerators for the high-end applications. Several novel communication and memory architectures are proposed that address the bandwidth and scalability issues of the massively parallel multi-processor accelerators. Finally, **Chapter 7** concludes this thesis and gives some directions for future works.

CHAPTER 2

Problem Analysis, Research Task Formulation and Related Research

This chapter focuses on the analysis of modern demanding applications from various application fields. The applications are analyzed from various perspectives including: the real-time performance requirements, the character of computation and communication involved, and some other features. The existing design methods and EDA tools, which are used to implement hardware accelerators are discussed. The issues that are involved in the architecture design of hardware accelerators for modern highly-demanding applications are discussed in detail. Afterwards, the LDPC decoding application is briefly introduced. It is used as a representative example to further illustrate the issues of these applications. Based on the analysis of the issues, the requirements are formulated that have to be addressed by an adequate accelerator design methodology for highly-demanding applications. The main concepts of our proposed design method are briefly outlined. The precise research task is formulated. The related research discussion provides an insight into the state-of-the-art methods and EDA tools for the design of hardware accelerators. The main limitations and capabilities of these design method and EDA tools are discussed. Finally, the existing hardware accelerator architectures are overviewed for LDPC decoders with a novel classification to get an insight into these architectures.

The rest of this chapter is organized as follows. Section 2.1 discusses the issues of hardware accelerators for highly-demanding applications. Section 2.2 considers the design issues of LDPC decoders. Section 2.3 discusses the requirements that

have to be addressed by an adequate accelerator design methodology for highly-demanding applications. The precise research task formulation and the main assumptions are presented in Section 2.4. Section 2.5 discusses the main concept of the proposed design approach. The related research work is discussed in Section 2.6. Finally, Section 2.7 concludes this chapter.

2.1 Issues of Hardware Accelerator Design for Highly-demanding Applications

In Chapter 1, the highly-demanding applications are introduced through several examples. A common feature of these applications is their ultra-high performance demands and also a low energy consumption requirement of many of them, especially, those being mobile. However, various highly-demanding applications may have different computational characteristics and specific design requirements. In the following, the features and the issues involved in the design of hardware accelerators for such applications are discussed.

Although hardware accelerator design is not a new problem, it is only partially solved, and there is much room for further extension and/or improvement of the existing methods and tools. One can construct a trivial hardware accelerator through a straightforward compilation of an algorithm described in a hardware description language, like Verilog or VHDL, or in a high-level language like C, C++, SystemC or MATLAB into hardware. However, in most cases the result of such a straightforward compilation will not be satisfactory for critical parts of demanding applications. In embedded computing, hardware acceleration has been intensively researched during the last decade, mainly for signal, video and image processing applications, for efficiently implementing in hardware transforms, filters and similar complex operations [11–19]. All these operations have in common that they mainly involve functional parallelism, and either do not require (global) memory accesses, because they directly process the incoming stream of data, or they require relatively simple and regular, limited in space and time local memory accesses between which relatively large portions of computations are performed. In consequence, the main problems of hardware accelerator design for this kind of applications are not related to memory or communication bottlenecks, but to an effective and efficient processing unit synthesis through an adequate parallelism exploitation of the basic register-transfer-level (RTL) operations needed for the implementation of the required computations, and adequate implementation of these basic operations in hardware. For this kind of applications, the basic concepts of an effective and efficient accelerator design can be summarized as follows [6, 38]:

- parallelism exploitation for execution of a particular computation instance due to availability of multiple application-specific operational resources working in parallel;

- parallelism exploitation for execution of several different computation instances at the same time due to pipelining;
- application-specific optimal synthesis of processing units, with tailored processing and data granularity.

More specifically, these concepts can be oriented towards the data parallelism, functional parallelism or their mixture. For data parallelism exploitation, multiple data instances of the same type are processed simultaneously provided the application allows for this and the corresponding resources are available. In case of functional parallelism, different operations are performed simultaneously on (possibly) different data instances. Also, the speculative execution can be used to exploit more parallelism. To design a high-quality hardware accelerator of this kind, it is necessary to perform a thorough analysis of the application's computation algorithms and exploit specific computational characteristics inherent to these algorithms. Different characteristics discovered and accounted for result in different approaches to the design of hardware accelerators of this kind, and therefore, in the past a number of different basic accelerator micro-architecture types were considered:

- straightforward datapath/controller architecture;
- parallel hardware architecture;
- pipeline hardware architecture;
- parallel-pipeline hardware architecture.

Summing up, for this kind of applications, the main problems of hardware accelerator design are limited to an effective and efficient computation unit design at the RTL-level (i.e. micro-architecture design) and circuit synthesis for the micro-architecture modules. Circuit synthesis can be performed automatically using one of many available EDA-tools. Currently, in many cases the micro-architecture design for this kind accelerators can also reasonably be supported by the methods of "application analysis and parallelization" (APP), "High-Level-Synthesis" (HLS) [11–19] and emerging commercial HLS tools [20–26]. Nevertheless, the RTL-level computation unit design is often not easy, because some of the modern demanding applications require resolution of complex data or control dependencies (e.g. CABAC decoding in the latest multi-domain video coding standard H.264/AVC [6]), what increases difficulty of an adequate pipeline construction. Further details on APP and HLS methods, and the corresponding state-of-the-art research and commercial tools can be found in Section 2.6 on the related research.

Many modern applications (e.g. various communication, multimedia, networking or encryption applications, etc) are of different kind, and involve serious design issues that are beyond the design capabilities of today's design methods and tools for hardware accelerators. These issues will be discussed in detail as follows.

Many modern highly-demanding applications involve sets of heterogeneous data-parallel tasks with complex inter-task data dependencies and complex interrelationships between the data and computing operations at the task level. Often the tasks iteratively operate on each other's data. One task consumes and produces data in one particular order, while another consumes and produces data in a different order. This all results in complex memory accesses and complex communication between the memories and processing elements in the related hardware multi-processor architectures. For applications of this kind, the main design problems are related to an adequate resolution of memory and communication bottlenecks and to decreasing the memory and communication hardware complexity, which has to be achieved through an adequate memory and communication structure design. Additionally, in the high performance multi-processor accelerators, parallelism has to be exploited on a massive scale. However, due to area, energy consumption and cost minimization requirements, partially-parallel architectures have to be usually used, which are more difficult to design than the fully-parallel ones.

Moreover, for this kind of applications, the memory and communication structure design, and micro-architecture design for computing units cannot be performed independently, because they substantially influence each other. For example, exploitation of more data parallelism in a computing unit micro-architecture usually requires getting the data in parallel for processing, i.e. having simultaneous access to memories in which the data reside (this results in e.g. vector, multi-bank or multi-port memories) and simultaneous transmission of the data (this results e.g. in multiple interconnects), or pre-fetching the data in parallel to other computations. This substantially increases the memory and communication hardware. From the above, it should be clear that for applications of this kind complex interrelationships exist between the computing unit design and corresponding memory and communication structure design, and complex tradeoffs have to be resolved between the accelerator effectiveness (e.g. computation speed or throughput) and efficiency (e.g. hardware complexity, power and energy consumption etc.).

Furthermore, many of the modern demanding applications involve algorithms with massive data parallelism or task-level functional parallelism (e.g. LDPC code decoders of the newest communication system standards like IEEE 802.11n, 802.16e/m, 802.15.3c, 802.3an, etc.). To adequately serve these applications, hardware accelerators with parallel multi-processor macro-architectures have to be considered, involving several identical or different concurrently working hardware processors, each operating on a (partly) different data sub-set. Each of these processors can also be more or less parallel. For this kind of accelerators, the accelerator's parallelism can be realized at two levels:

- macro-architecture level, where elements are elementary processors or accelerators and complex multi-processor or multi-accelerators are build of them, and

- micro-architecture level, where the internal architecture of a single processor or accelerator at the RTL-level can be parallel.

Moreover, there is a tradeoff between the amount of parallelism and resources at each of the two levels (e.g. similar performance can be achieved with less processors each being more parallel and better targeted to particular part of application, as with more processors each being less parallel and less application-specific). The two architecture levels are strongly interrelated and interwoven, also through their relationships with the memory and interconnection structures. In consequence, optimization of the performance/resources tradeoff required by a particular application can only be achieved through a careful construction of an adequate application-specific macro-/micro-architecture combination. The aim here is to find an adequate balance between the number of parallel hardware processors of various kinds, the intra-processor parallelism and complexity, the complexity and effectiveness of memory structures, and the complexity of the inter-processor and/or processor/memory communication, rather than to only optimize the processing units, or separately optimize the micro- or macro-architecture. To achieve this aim several promising micro-/macro-architecture combinations representing complete complex multi-processor accelerator architectures have to be considered, and finally, the best of them has to be selected for an actual realization (see Chapter 3 for details).

Many of the modern highly-demanding applications involve complex algorithms with multi-input multi-output (MIMO) operations and require highly optimized implementation through operation and micro-architecture level parallelism exploitation to satisfy the high-throughput requirements. The micro-architecture level synthesis should exploit among other the operation and data level parallelism, operation chaining, multi-cycle operations, structural and functional pipelining, etc. The traditional hardware design methodologies that incorporate the simple single (two or maximum three) input single output (SISO, MISO) RTL operations are not able to implement the MIMO operations effectively and efficiently. Example of the three input single output operation is the most widely used multiply-accumulate (MAC) operation, required in various applications. Further, these atomic operations when used for the realization of complex MIMO operations result in a large number of computation cycles, what makes it impossible to realize the ultra-high throughput for the high-end applications. Also, the MIMO operations can be implemented using various basic functional units (FUs) having different characteristics (such as fast or slow adders/multipliers, etc.) resulting in various cost/performance tradeoffs. When designing accelerators for highly-demanding applications, the possible micro-architecture tradeoffs have to be carefully considered (see Chapter 5 for details).

Additionally, for the high-end applications that require massively parallel multi-processor accelerators to satisfy the ultra-high throughput, the effective communication and memory architectures, and the compatibility of the processing, memory and communication subsystems play the decisive role. Although

some research results related to the memory and communication architectures can be found in the literature [27–33] in the context of programmable on-chip multi-processor systems, the memory and communication architectures were proposed there for the much larger and much slower programmable processors of on-chip multi-processors systems. They are not adequate for the small and ultra-fast hardware processors of the massively parallel multi-processor accelerators due to a much too low bandwidth and scalability issues. It will be demonstrated in this thesis that in the massively parallel multi-processor structures for many highly-demanding applications, the communication and memory architectures play a decisive role. The communication architectures can not be designed as a simple flat homogenous networks and the memory as a multi-port memory. Especially, the communication network among the processors or processors and memories has a dominating influence on all the most important design aspects such as delay, area and power consumption. The additional performance gains expected from an increased parallelism will end up in diminished returns, while exploding the interconnect complexity. Therefore, all the architectural, as well as, the data and computation mapping decisions regarding the memories and processors have to be made in the context of the communication architecture design to actually boost the performance. For the massively parallel hardware accelerators, the problem of how to keep up with the increasing processing parallelism, while ensuring the scalability of memory and communication is a very challenging design problem. To our knowledge, it has not been addressed satisfactorily till now (see Chapter 6 for details).

Furthermore, the back-to-back (cyclic) inter-task data dependencies are often so complex that make it extremely difficult or even impossible to overlap the processing of one set of tasks with another set of dependent tasks. The traditional schedulers[1] are not adequate for resolving the kinds of complex scheduling problems involving back-to-back data dependencies among the tasks with one task responsible for producing partial data not only for a single, but for many dependent tasks. This partial fulfillment of data paralyzes the data dependent processors to start their execution earlier until all the data is available, what results in a lower utilization of the processing resources. Even in the case, when an overlap schedule is found (what would result in reduction of a number of processors for a certain performance level), the influence of the scheduling decisions on the cost of memory and communication structures has to be taken into account. It is very probable for this kind of applications that the cost of memory and communication structures may surpass the cost savings due to the decreased processing resources. Therefore, the task scheduling freedom need to be carefully tradeoff against the memory and communication structures complexity, thereby reducing the overall cost to a larger degree and thus providing the opportunity to utilize even more processors (see chapters 3 and 5 for details).

Also, for various modern demanding applications the design objectives and

[1]Described in related work section of this chapter

constraints are quite different from those supported by the traditional[2] hardware accelerator design methods presented in literature that usually accept just a few constraints, like only the clock speed constraint. An actual multi-objective optimization and tradeoff exploration are necessary. For instance, let us consider the video decoding. The same kind of video decoding can be used in a low power and area mobile device or a laptop, or a high-definition television (HDTV) from low through moderate to extremely high performance levels. Therefore, to quickly arrive at an adequate design quality for a particular application with all its particular constraints, objectives and tradeoff profiles, an adequate architecture design space exploration is indispensable taking into account the various design objectives, constraints and tradeoffs, which is not supported by the current hardware design methods and tools even for the simple case of micro-architecture synthesis (see Chapter 3 for details).

Moreover, different applications adopt different algorithms and have different requirements in relation to the throughput, area and other parameters. For instance, the above-mentioned new communication system standards adopt different LDPC codes classes and decoding algorithms, and have different requirements regarding code rate, code length, throughput etc. As it is not often known in advance which of the proposed standards will actually be accepted, and due to profitability it is extremely important for the industry to have their equipment for a new standard ready before the standard is actually adopted, accelerators for applications involving new standards have often to be multi-standard, or easy adaptable for a particular standard after being designed and/or fabricated. Their adaptability can be achieved through the design-time adaptation and/or post-production (field-use) adaptation. In particular, the field-use adaptation can have a form of a run-time reconfiguration. An additional advantage of a run-time reconfigurable accelerator is its ability to dynamically adapt to changing operating conditions, e.g. it can adapt its structure and operation to different quality-of-service, energy usage and transmission speed requirements, noise levels and other environmental conditions. To implement the field-use (run-time) adaptation, the popular re-configurable FPGA technology could potentially be used. In FPGAs, the re-configuration resources and mechanisms are pre-implemented, and they guarantee a complete total reconfiguration ability. In consequence, when developing a re-configurable accelerator for an FPGA implementation, a trivial approach can be used of separately developing particular accelerators for different acceleration cases (e.g. for different LDPC codes and corresponding requirements) and totally (or partially when applicable) re-configuring the FPGA to implement each of the accelerators. However, due to the extremely high throughput and/or low energy consumption requirements of many modern demanding applications, re-configurable accelerators for such applications cannot usually be implemented using FPGAs. Due to the high overhead of their general reconfiguration resources, FPGAs are unable to deliver so high throughput and are not energy efficient.

[2]High-Level-Synthesis

Therefore, re-configurable accelerators for highly-demanding applications require a high-performance energy-efficient application-specific integrated circuit (ASIC) implementation. However, in the case of an ASIC implementation, the complexity and other features of the total computation, memory, communication and re-configuration resources supporting acceleration for all the required acceleration cases are of crucial importance, and therefore, the trivial development approach cannot be used. Therefore, the design goal should be to decide one globally optimal adaptable accelerator architecture that adequately satisfies the requirements for all the particular accelerator instances required, and not a set of the best individual accelerators for each particular acceleration case and their combined re-configurable implementation. Moreover, the re-configuration resources have to be accounted for. However, not the general total reconfiguration resources as in FPGAs should be aimed at, but limited efficient design-case-specific reconfiguration resources (see Chapter 3 for details).

Summing up, the massive parallelism to be exploited to achieve the ultra-high throughput required by the modern demanding applications, the complex interrelationships between the data and computing operations, and the combined massive parallelism exploitation at the two architecture levels (micro-/macro-architecture), make the design of an effective and efficient application-specific hardware accelerator a very challenging task. To effectively perform this task, the heterogeneous massive parallelism available in a given application has to be explored and exploited in an adequate manner to satisfactorily fulfill the design requirements through constructing an architecture that satisfies the required performance, power consumption and area tradeoffs. Proposing an adequate solution for this task is the main aim of the research reported in this thesis.

2.2 Accelerator Design for LDPC Decoders

In the previous section, the main issues involved in the accelerator design for modern highly-demanding applications are discussed. In this section, they will be illustrated and further explained for the accelerator design of LDPC decoding application.

Figure 2.1: LDPC encoding and decoding process

A systematic LDPC encoder encodes a message of k bits into a codeword of

length n with the message bits k followed by m parity checks, as shown in Figure 2.1. Each parity check is computed based on a sub-set of message bits. The codeword is transmitted through a communication channel to a decoder. The decoder checks the validity of the received codeword by re-computing the parity checks, using a parity check matrix (PCM) of size $m \times n$. For a codeword to be valid, it must satisfy the set of all m parity checks.

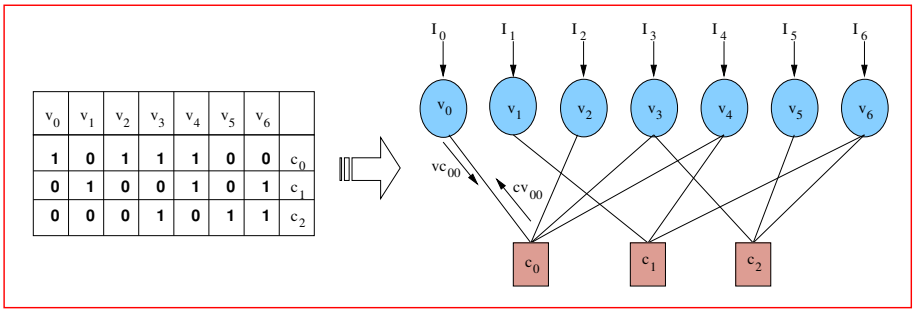

Figure 2.2: PCM for a (7,4) LDPC code and its corresponding Tanner graph, where $\{v_0.....v_6\}$ represents variable nodes, $\{c_0.....c_2\}$ represents check nodes and $\{I_0.....I_6\}$ represents the input intrinsic channel information

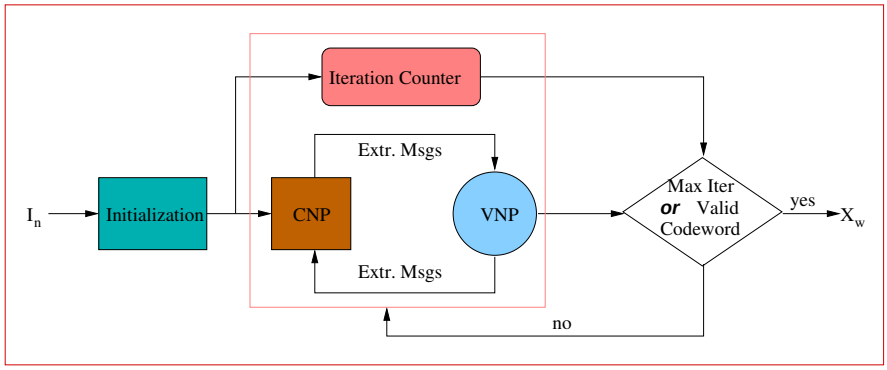

Figure 2.3: Decoding flow diagram representing the main steps of MPA algorithm

In Figure 2.2 an example PCM for a (7,4) LDPC code is given. "1" in a position $\text{PCM}_{i,j}$ of this matrix means that a particular bit participates in a parity check equation. For example, in the first row the bits at positions v_0, v_2, v_3, v_4 participate in the computation of the parity check c_0, that is, $c_0 = v_0 \oplus v_2 \oplus v_3 \oplus v_4$, where $\oplus$ represents the exclusive-OR operation. Each PCM can be

represented by its corresponding bipartite graph (Tanner graph). The Tanner graph corresponding to an (n, k) LDPC code consists of n variable nodes (VN) and $m = n - k$ check nodes (CN), connected with each other through edges, as shown in Figure 2.2. Each row in the PCM represents a parity check equation c_i, $0 \leq i \leq m-1$, and each column represents a coded bit v_j, $0 \leq j \leq n-1$. An edge exists between a CN i and VN j, if the corresponding value $\text{PCM}_{i,j}$ is non-zero in the PCM.

Usually, iterative Message Passing (MP) algorithms [39–42] are used for decoding of the LDPC codes. During the decoding, specific messages are exchanged among the nodes through the edges. The messages represent the log-likelihood ratios (LLRs) of the codeword bits based on the channel observations [39]. The algorithm starts with the so-called intrinsic LLRs of the received symbols based on the channel observations. Starting with the intrinsic LLR values, the algorithm iteratively updates the extrinsic LLR messages from the check nodes to variable nodes and from the variable nodes to check nodes and sends them among the VNs and CNs along the corresponding Tanner graph edges. If after several iterations the parity check equation is satisfied, the decoding stops, and the decoded codeword is created and considered to be a valid codeword. Otherwise, the algorithm further iterates until a given maximum number of iterations is reached. The main decoding steps of the MP algorithm are graphically represented in Figure 2.3. Since the Tanner graphs corresponding to practical LDPC codes of the newest communication system standards involve hundreds of variables and check nodes, and even more edges, LDPC decoding represents a massive computation and communication task. Moreover, the modern communication system standards require very high throughput in the range of Gbps and above, for applications like digital TV broadcasting, mmWave WPAN, etc. For the realization of the throughput as high as several Gbps, complex massively parallel multi-processor accelerators are necessary.

In many practical MP algorithms, the variable node computations are implemented as additions of the variable node inputs and the check node computations as *ln* or *tanh* function computation for each check node input and addition of the results of the *ln/tanh* computations[3]. In some simplified practical algorithms, the check nodes just compare their inputs to find the lowest and second lowest value. Since each node receives several inputs, the basic operations performed in nodes are the multi-input additions or multi-input comparisons. In the corresponding accelerator, the spectrum of possible implementations of each of these multi-input operations spans between the two extremes of a fully-serial slow processing in a simple two-input adder/comparator to a fully-parallel fast processing in a complex multi-input parallel adder/comparator. When the variable nodes perform their computations, the check nodes are waiting on the computation results and vice versa, but all nodes of a given kind, i.e. all the variable nodes or all the check nodes, may perform their computations in parallel. If all the nodes of a given

[3]Described in detail in Chapter 4

kind would actually perform their computations simultaneously, this would require a complex parallel access to the memories of all nodes of the opposite kind, and could only be realized with a very distributed memory structure and very complex and expensive interconnection structure. In contrary, performing the computations corresponding to different nodes fully-serially can requires just one memory access at a time and result in reasonably simple corresponding memory and interconnection structures.

Summing up, when considering the hardware acceleration for LDPC decoding, the possible micro-architectures span the full spectrum from a fully-serial to a fully-parallel, and the possible macro-architectures of the multi-accelerator structures span the full spectrum from a fully-serial [43] to a fully-parallel [44–51], with in between a large variety of partially-parallel architectures [52–83]. These architectures and their features are described in detail in Section 2.6. The large variety of possible partially-parallel architectures is due to the ability of (partial) parallelism exploitation at two levels: micro-architecture level (where the internal architecture of an elementary accelerator at the RTL-level can be parallel), and macro-architecture level (where complex multi-accelerators can be built of the elementary accelerators). At the macro-architecture level, the variable and check nodes and their respective computational processes are mapped to the corresponding variable node (VNP) and check node (CNP) processing units (PUs), respectively. At the micro-architecture level, both VNP and CNP computations can be realized through a (partially) parallel or serial computation process implemented in an elementary PU, in which (a number of) inputs of the VN or CN are processed simultaneously or one by one, respectively. The PUs micro-architecture has a huge impact on the accelerator's throughput, because these units constitute the computational kernels and determine the accelerator's operating frequency. Also, the mapping strategies of the variable and check nodes to their respective VNP and CNP elementary processors vary from one architectural choice to another and there are many various mapping possibilities for the partially-parallel architectures.

Also, complex tradeoffs are possible between the parallelism and resources at the micro-architecture level, and the parallelism and resources at the macro-architecture level. Moreover, changing the level of parallelism for computations in the micro- or macro-architecture of the LDPC accelerator requires a corresponding change of the memory and communication structure. Thus, the computation, memory and communication architectures are strictly interrelated and cannot be designed in separation. The large number of possible micro-/macro-architecture combinations and related node mappings leads to a large number of various tradeoff points in the LDPC accelerator design space representing various accelerator architectures with different characteristics.

To arrive at high-quality accelerator designs, the accelerator design space exploration (DSE) is necessary in which a substantial set of the most promising of these architectures will be constructed and analyzed, and the best of them will be selected for further analysis, refinement and actual implementation. To perform

the DSE, a new adequate design methodology is necessary.

2.3 Requirements of an Adequate Accelerator Design

In the last two sections, the issues of hardware accelerator design for highly-demanding applications are discussed. From the discussion, it should be clear that the existing HLS methods, specifically developed and limited to the RTL-level micro-architecture synthesis of processing units, are only able to partly support the internal architecture design for particular computation units, and are not sufficient to adequately support the total complex multi-processor hardware accelerator design process for the modern demanding applications. It should also be clear that a new more complex and sophisticated design methodology is needed for the modern demanding accelerator design than the existing HLS methods. This new methodology should adequately address many issues, including the following:

- micro-architecture synthesis of generic processors for each of the data-parallel task,
- macro-architecture synthesis of the multi-processor accelerator,
- memory and communication architectures synthesis, to resolve the memory and communication issues of the massively parallel multi-processor accelerators required for the ultra-high throughput applications,
- combined processor's micro-architecture and macro-architecture design, and the corresponding memory and communication architectures design, to resolve effectively and efficiently the complex mutual tradeoffs among them,
- tradeoffs exploitation between the micro- and macro-architecture for the processors, and the corresponding memory and communication architectures, as well as, among the performance, power consumption and area,
- multi-objective optimization and tradeoffs exploitation among the various optimization objectives (performance, power consumption, area, etc), and
- adaptable accelerator design accounting for the design-time or field-use adaptation.

From the previous sections of this chapter, it should be clear that only an appropriate accelerator architecture design space exploration (DSE) and tradeoff exploitation between various parts and features of possible architectures can guarantee an adequate accelerator design quality. Thus, the design process for demanding accelerators should rather be focused on the construction, analysis and evaluation of promising complete complex accelerator architectures, and using this for the

DSE by "what if" analysis, than on the fully automatic synthesis of individual computing units as offered by the today's HLS tools. At least partially, a different kind of design support is crucial here than that offered by the traditional HLS. HLS tools can and should be used in the scope of the demanding accelerator design, but for supporting the individual computing unit design tasks, and for so far as they are effective for these tasks.

2.4 Research Task Formulation

Problem Statement: The general aim of the research reported in this thesis was to ***analyze the issues and requirements of hardware accelerator design for the modern highly-demanding applications, as well as, to propose, implement, analyze and evaluate an adequate semi-automatic design method addressing the issues and satisfying the requirements***. In order to realize this aim, the following major research tasks had to be performed:

- to analyze several highly-demanding modern applications and the issues of accelerator design for these applications; specifically the issues that cannot be resolved using the traditional architecture design methodologies for hardware acceleration as HLS,
- to formulate the requirements for an adequate hardware accelerator design method for the highly-demanding applications,
- to propose an adequate hardware accelerator design methodology to adequately address the issues and satisfy the requirements,
- to propose an architecture design space exploration (DSE) framework for the semi-automatic design, exploration and evaluation of various architectures in order to find some optimal or near-optimal architectures, while taking into account the design constraints and objectives and the tradeoff preferences among the objectives,
- to analyze and evaluate the proposed design methodology and the related DSE framework through performing a series of experiments, when using some representative highly-demanding modern applications, and
- to arrive at some general conclusions based on the results of the research performed.

Assumptions: The following assumptions are made in relation to the above research task formulation and the context in which the research task is performed and the method is proposed:

- an application demands extremely high performance that is virtually impossible to be achieved using any programmable (multi-)processor based

implementation, and in consequence, requires application-specific hardware multi-processor accelerator to be used for exploiting the application parallelism to adequately satisfy the performance demands,

- an application involves a massive data parallelism and various tasks, and the application specification at the task level is available in a fully parallel form, with parallelism exposed for both of the architecture levels, i.e. the micro- and macro-architecture level,

- application re-partitioning, i.e. task merging, splitting, etc, after the initial application analysis and generic architecture platform design is not possible, i.e. the granularity of the parallel application specification has to be decided during or before the initial application analysis and generic platform design, and

- application involves complex interrelationships among the data and computing operations, as well as global and/or irregular information (data) flows.

2.5 The Proposed Design Approach

In order to address the issues and requirements of the hardware multi-processor accelerators for highly-demanding applications as formulated above, the proposed design approach is based on the *quality-driven model-based system-level design methodology* proposed by Jóźwiak [34]. According to the *quality-driven design paradigm*, design requirements represent a general model of the required quality that models the design problem at hand through the imposition of a number of constraints and objectives in relation to the acceptable or preferred problem solutions. It is thus an abstract model of a solution to the problem. Since it limits the space of the acceptable or preferred solutions to only a certain degree, it models many solutions concurrently. Each of the solutions fulfills all the hard constraints of the model, but different solutions can satisfy its objectives to various degrees. It is possible to distinguish three kinds of requirements: functional, structural, and parametric. Requirements of each of the three classes impose limitations on the structure of a required solution, but they do it in different ways. Structural requirements define the acceptable or preferred solution structures directly, by limiting them to a certain class or imposing a preference relation on them. Parametric requirements define the structures indirectly, by requiring the structures to have specific physical, economic or other properties (described by values of some parameters) that fulfill given constraints and satisfy stated objectives. Functional requirements also define the structures indirectly by requiring the structures to expose a certain externally observable behavior that realizes the required behavior.

The *quality-driven model-based system-level architecture synthesis* involves

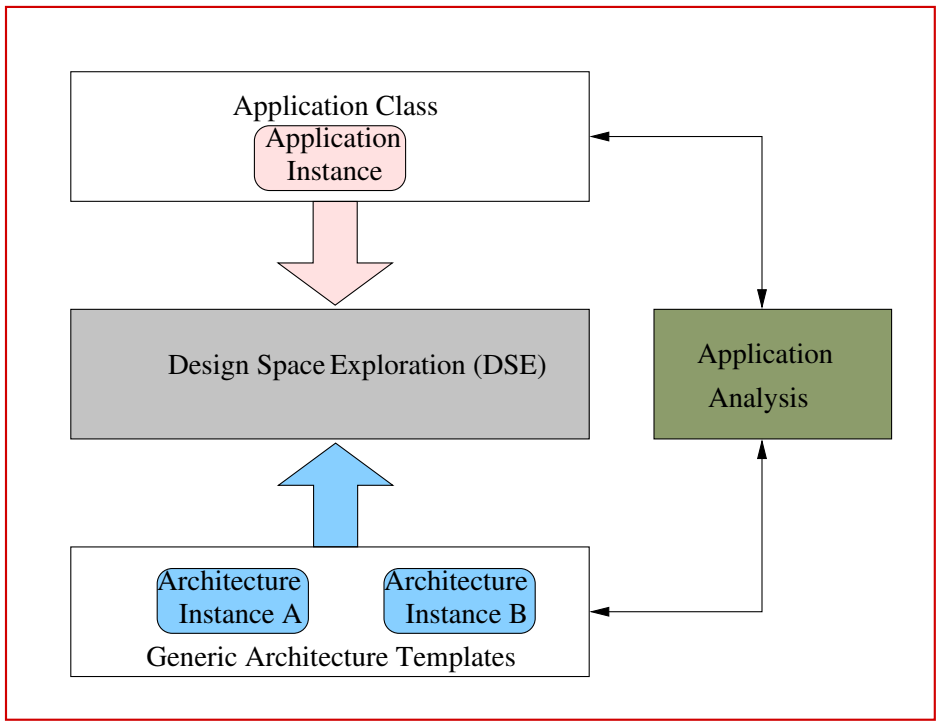

Figure 2.4: Quality-driven model-based system design

- an initial application (application class) analysis and generic architecture platform (template) design or selection for reuse, and
- the final model-based architecture exploration and synthesis when using the newly designed or selected for reuse platform or platforms

Based on the application class analysis, one or more promising generic system architectures are proposed that could satisfy the application requirements, and are used to design or reuse the generic architecture templates corresponding to these architectures, as shown in Figure 2.4. Then, for a particular application (application behavior model), and a particular instance of the generic architecture template and an abstract decision model, the design space exploration (DSE) is performed that aims at finding several promising architectures, and finally, the selection of the most promising architecture, as shown in Figure 2.4.

An approach similar to the one discussed above is followed for the hardware multi-processor accelerators design through the development and characterization of generic architecture templates for each application class, supported by an adequate DSE framework.

2.6 Related Research

In the first two sections of this chapter, the main issues of the hardware accelerators design for highly-demanding application are discussed. Also, the main limitation of the existing design methods and tools are described for the architecture exploration and synthesis of the hardware accelerators for highly-demanding applications. In this section, the existing methods and tools for the design of hardware accelerators and similar systems both from the academia and industry are discussed in more detail. The related research discussion will be made in relation to the following three parts:

i. system-level design,

ii. methods and tools for hardware accelerator design, and

iii. hardware accelerator designs for highly-demanding applications.

Macro-architecture synthesis of the multi-processor hardware accelerators is actually a specific case of the system-level design. One of the main problems in the system-level architecture synthesis is the problem of scheduling and mapping. Many scheduling and mapping methods have been proposed by the system-level design research, as for instance discussed in [84–89]. More information on macro-architecture exploration including scheduling and mapping can be found in [1]. However, in the research reported in this thesis, two new scheduling methods are proposed that better suit the hardware multi-processor accelerator design through enabling the tradeoff exploitation between the processors, and the corresponding memory and communication structures (see Chapter 3 for details). The very important and closely related to the focus of this thesis problem of adequate communication and memory architecture design unfortunately did not get enough attention in the system-level design. The related research devoted to this aspect is separately discussed in Section 6.1. Also, a large part of the existing research performed in the scope of the system-level design methods/frameworks is not directly applicable to the macro-architecture synthesis of the hardware multi-processor accelerators, as these methods/frameworks mainly cover the modeling aspects of the system-level design, as e.g. reported in [90–93].

In the whole scope of the hardware accelerator design, only the micro-architecture exploration and synthesis are reasonably supported by the methods of application analysis and parallelization (APP), and high-level-synthesis (HLS). Since the APP and HLS have attracted much attention and research effort in the past, the research work reported in thesis was not focused on them, but on many other not researched or much less researched aspects of hardware accelerator design. Nevertheless, below the methods and tools for APP and HLS are briefly discussed.

The methods and tools of application analysis can be used to analyze (profile) an application to discover its bottlenecks parts (called kernels) that can be then implemented as hardware accelerators. The methods of APP can be further

used to find some promising parallel versions of the applications or their parts (kernels). Finally, for a given behavior specification of a kernel and of the parametric constraints and objectives, HLS creates a corresponding RTL-level hardware structure that realizes the behavior required and satisfies the constraints and objectives. In general, this structure involves a data-path and control-path. The data-path represents a network of computation, memory and interconnection components, and the control-path one or more collaborating control automata. In virtually all cases, HLS only accounts for a simple memory in a form of registers and simple flat interconnect structure between the data-path functional units and registers. In case of the most popular synchronous systems, the control automaton is a finite state machine (FSM). Consequently, hardware accelerator synthesis using the APP and HLS methods and tools can produce a single (usually coarse-grain) RTL-level hardware accelerator or a network of collaborating (mixed-grain) RTL-level accelerators [94, 95], one accelerator for each application kernel. The macro-architecture design, as well as, memory and communication system design for the whole application or its more sophisticated kernels are here not considered at all. Moreover, the APP, as well as, the HLS methods and tools have their own limitations. The effectiveness of the HLS tools directly rely on the extent to which the application is actually parallelized (assuming sequential specification). Therefore, the existing APP methods and tools are explained first followed by the HLS.

2.6.1 Application Analysis and Parallelization

Many application specifications and reference specifications of standards are formulated in a sequential C, C++ or other code. Application analysis (profiling) of C code is required in order to reveal the execution behavior and some other important characteristics of an application. Very often the compiler front-ends and their intermediate representation (IR) are used in collaboration with application analysis and parallelization (restructuring) tools for the application analysis and application algorithm or code restructuring and optimization. Unlike the application profiling from the perspective of hardware/software co-design that identifies some frequently executed parts of the application (hot spots) to be implemented in hardware and other parts in software, the aim of the application analysis and restructuring from the point of view of hardware accelerators is to identify and make explicit the various kinds of parallelism of the application.

Analysis (Profiling) of the application's C code is required in order to reveal the execution behavior and some other important characteristics of an application. The execution behavior and additional characteristics involve several properties that are needed to be extracted using several different profiling techniques. These main properties are: the application code structure; dependency relations within the application; execution time behavior; memory access behavior; code coverage. The profiling results can be used for different purposes, mainly to: reveal how the application code works; find the bottlenecks in the application;

architecture-independent software optimizations; architecture-dependent software optimizations; architecture customization to a given application. Extraction of the profiling data can be accomplished with different profiling techniques which are mainly categorized as static and dynamic profiling. Source code is analyzed by parsing the code in order to expose the code structure. Code structure is usually represented with data-flow graph (DFG) and control-flow graph (CFG) or combination of both as control data flow graph (CDFG). Results from static profiling are totally data-independent and purely based on the analysis of code structure. On the other hand, dynamic profilers provides information on the dynamic behavior of an application is analyzed by executing the code with some input stimuli.

Further, the application profiling tools can be divided into the following three categories: source level, intermediate representation (IR) level and assembly level profilers. Source level profilers are the ones used in the initial phase in order to characterize the overall execution behavior of the application. GNU gprof [96], GNU gcov [97], VfAnalyst [98] are some examples of source level application profiling tools that make explicit the application bottlenecks and perform dependence analysis. They generate statistics about the CPU time spent in each function, and/or loop, and/or code line of the program. IR level or precision enhanced source code profiling intends to have more accurate profiling data. LANCE [99], microProfiler [100] and TotalProf [101] are some of the most popular tools that provide more accurate profiling information in comparison with source level profilers. For Instance, the microProfiler provides metrics such as the execution frequencies of different arithmetic and logical operations, frequently used C data types together with their dynamic min/max values, as well as, the bit width of arithmetic operands and constants [100]. It also aims to aid designers to take decisions about overall memory subsystem comprising of a number of different cache levels, scratch-pad memories and main memory by providing profiling information such as the total amount of memory requirements (e.g. static, run-time stack, heap) of an application, the most heavily accessed source level data objects, the most memory intensive portions of an application, etc. Assembly/instruction-set-simulator (ISS) based profilers provides the most accurate results. These profilers are specific to a particular microprocessor family. Examples of such profilers are ATOM [102], HALT [103], Spix [104], Tensilica Xtensa Xplorer [105] and Stretch Profiler [106], etc.

The application profiling tools as discussed above analyze the applications mainly for the purpose of performance optimization on a particular microprocessor family or hardware/software partitioning of an embedded application, or architecture and custom instruction synthesis of an application-specific instruction-set processor (ASIP). In the hardware accelerator context, the profiling results can be used to discover and characterize the bottlenecks of the application for a possible hardware acceleration. Also, the profiling tools provide analysis regarding the memory hierarchies, cache misses, branching, etc. Since hardware accelerators are usually targeted to those parts of an application that mainly involve deterministic

parallelism, most of this kind of analysis results are not useful. Moreover, in most cases the source and IR level profilers perform the performance estimation assuming execution on a single-issue RISC processor. The so computed performance estimates can be used to discover application bottlenecks, but cannot be used directly to resolve the bottlenecks through hardware acceleration, i.e. to decide how much parallelism and which way have to exploited to achieve the required performance. For this purpose adequate application parallelization (APP) and architecture exploration and synthesis methods and tools are needed.

Many compilers and APP tools are based on the Stanford University Intermediate Format (SUIF) compiler infrastructure [107], which was especially developed to support parallelism identification. The SUIF frontend supports compilation of C, C++ or FORTRAN into SUIF high-level machine-independent intermediate representation (IR), which is a data-flow representation. From this high-level IR, the Machine-SUIF, a back-end compiler, creates a medium-level IR and then a low-level machine representation [108]. Many compilers and other synthesis tools use SUIF for application analysis, as well as for machine-independent application restructuring and optimization, HW/SW co-synthesis and high-level hardware synthesis. For instance, Napa-C [109] and Nimble [110] compilers perform HW/SW partitioning and map the hardware kernels to FPGA hardware. Malleable Architecture Generator (MARGE) maps application blocks to collaborating hardware sub-systems implemented on FPGAs as functional units composed of a data-path and a controller [111]. CASH [112] transforms ANSI C to RTL Verilog constructing a distributed hardware architecture. Also, the RAW compiler [113], which converts C or Fortran to Verilog, uses SUIF to perform parallelization. DEFACTO [114] uses SUIF to get the data flow representation from C, perform selected loop transformations and data permutations on this representation and produce VHDL. ROCCC compiler [115] that generates RTL VHDL from C uses SUIF-2 and Machine-SUIF for the control and data flow graph analysis. It performs loop restructuring, storage optimization and pipelining, when using its own IR. MATCH compiler [116] uses SUIF and MATLAB M-File with compiler directives to transform architecture expressed in MATLAB into RTL VHDL for FPGA implementation.

Several other non-SUIF-based compiler front-ends are used for analyzing applications, including the well-known GNU GCC compiler [96, 97]. For instance, ASC [117] translates C++ programs into FPGA hardware, when making possible an iterative design space exploration and tradeoff's exploitation among the latency, throughput and area. Trimaran [118] and LANCE [99] environments use C front-ends and DFG IR representations for the application code analysis and parallelization.

Automatic parallelization (restructuring) consists of converting a sequential application code into a functionally equivalent more parallel version of the code that can be executed simultaneously using multiple hardware resources (e.g. several basic processors, accelerators or functional units). Automatic parallelization, in many cases mainly focuses on loop optimization, because very often loops ac-

count for a large majority of the computation effort and time.

Loop parallelization, also referred to as loop optimization, is performed as a combination of several different loop transformations. For instance, loop parallelization tries to split or unroll a loop so that its different iterations can be executed concurrently. However, before a loop can be actually parallelized, it has to be analyzed if the parallelization is safe and worth of effort. The first question is answered through data dependence analysis, i.e. determining whether the loop iterations can be executed independently of each other, and the second through estimation and comparison of the sequential and parallel execution times (also accounting for the communication overhead) and of the resources used in each case. Loop analysis is hard, mainly due to the difficulty of dependence analysis (e.g. pointers, recursion, indirect addressing, indirect calls, etc.), accesses to shared variables, I/O and other global resources, and the often unknown number of loop iterations.

Popular loop transformations include: loop splitting/strip mining (breaks a loop into multiple loops with the same bodies, but with different loop index subranges, with its special case being loop peeling that decides to perform that first loop iteration before entering the loop for the remaining iterations); loop fission/tiling (splits a loop into several loops over the same index range, but each having as its body only a part of the original loop's body); loop fusion (combines bodies of independent successive loops with the same number of iterations); loop-invariant code motion (places a loop-invariant code before the loop); loop permutation/interchange (exchanges inner loops with outer loops); loop unrolling (replicates the loop body several times to decrease the number iterations, i.e. the number of the loop condition tests and jumps - it is a kind of nested loop expansion enabling parallelism exploitation of the inner loops); loop un-switching (moves an if/else condition from inside of a loop to the outside through duplicating the loop's body and putting a copy of the body in both if and else clauses); loop reversal (reverses the order of the index variable assignment, and this way enables some dependency elimination); loop skewing (rearranges multi-dimensional array accesses of a nested loop in which each iteration of the inner loop depends on previous iterations, so that the dependencies are only between iterations of the outer loop); software pipelining (a specific out-of-order execution of loop iterations used to hide the different latencies of various functional units that shifts operations across the iteration borders through subdividing the loop operations into several stages and executing in a single iteration stage 1 from iteration i, stage 2 from iteration i-*1*, etc.). Recursive Variable Expansion (removes all the data dependencies from the loop and then the parallelism is only bounded by the amount of available resources [119, 120]). More information on loop parallelization can be found in [121–125].

For many applications, parallelism can also be substantially increased through control restructuring, i.e. combining of control nodes, so that the operations corresponding to these nodes can be executed in parallel. Transformation of a serial nested if-then-else structure into a parallel switch/case multi-branch structure can

serve here as an example. More generally, the control restructuring involves joining together frequently executed sequences of basic operation blocks. Since the average instruction-level parallelism per basic block is between 2 and 3.5 [126], basic block combining can further increase the instruction-level parallelism. Its examples include the trace scheduling [127], as well as super block [128, 129] and hyper-block formation [130, 131]. Combining of the frequently executed basic block sequences makes it possible to eliminate the control constraints associated with the alternative execution traces. For instance, trace scheduling finds complete traces from the start to the end of a given control data flow graph (CDFG) and combines the basic blocks of each such trace. Finalizing this section, it has to be acknowledged that some information for this section has been recalled from reference [1].

2.6.2 High-Level-Synthesis

The three main HLS tasks are: resource allocation, operation binding and operation scheduling. Resource allocation consists in deciding the quantity of hardware resources of each type. Operation binding (assignment) consists in the mapping of one or more operations on a given allocated resource instance. Binding of several mutually exclusive operations to one resource instance realizes resource sharing and facilitates hardware minimization. Scheduling determines the temporal ordering of operations through assigning each operation to a given time (control) step that corresponds to a state in the related control FSM. Two specific scheduling algorithms are ASAP (as soon as possible) and ALAP (as late as possible). The difference between the ALAP and ASAP time steps of a given operation represents the scheduling freedom of the operation. ASAP optimally solves the unconstrained scheduling problem and its run-time is proportional to the number of vertices and edges in the scheduled DFG. The constraint scheduling problem is NP-complete [132, 133]. Therefore, heuristic algorithms are usually used to solve it efficiently, as e.g. list scheduling [134], trace scheduling [135], force directed scheduling [136], percolation scheduling [137], etc. Scheduling can be performed before, after or together with binding. Since scheduling and binding are strictly interrelated, several methods have been proposed to deal with their interdependencies [138–140]. However, HLS still requires adequate extensions for the control-dominated and mixed applications, speculative execution and conditional resource sharing [141, 142], as well as, for the run-time reconfigurable systems.

Moreover, most of the HLS methods were developed for the former ASIC technologies. They made assumptions on costs of different kinds of hardware resources that do not hold for the modern nano-dimension technologies and FPGAs. For instance, the interconnection costs of secondary importance in traditional ASIC technologies are dominant in the modern nano-dimension technologies and FPGAs, simple arithmetic units (e.g. adder) traditionally assumed to be much more expensive than multiplexers or registers are of comparable cost in FPGAs,

etc. Also, power and energy minimization is very important for modern applications and implementation technologies with static (leakage) power becoming more and more important factor. Consequently, while traditional HLS methods were focused on the optimization of a certain tradeoff between the hardware speed and arithmetic resources needed, the HLS methods required for the modern re-configurable systems have to perform the multi-objective optimization of the overall architecture, while adequately addressing the growing interconnect and power/energy consumption importance. Below the recent research in the fields that require re-work or extensions are briefly reviewed.

Some of the HLS methods that account for interconnect and other resource minimization are discussed in [143–149]. Most of them are focused on bit-width aware scheduling and binding or multiplexer cost minimization during binding. However, one should also not forget the importance of accounting for the complexity of controllers necessary for the implementation of resource sharing, conditional execution and other concepts, their interrelations and interconnections with the controlled data-paths, as well as for the overall hardware architecture refinement and optimization at the RTL-level between the traditional output of HLS and input of the actual circuit synthesis. Adequate addressing of these issues can give substantial resource reduction effects [150]. For applications involving both data and control dependencies specific scheduling heuristics [151, 152], as well as, speculative execution [153] and conditional resource sharing [152] techniques have been proposed.

Traditional HLS tools work for a static hardware and require a substantial extension to support the run-time re-configuration, and particularly, to account for the temporal partitioning and re-configuration overhead. One of the first works on scheduling and temporal partitioning, while accounting for the re-configuration time is reported in [154]. In [154, 155] temporal partitioning, scheduling and clustering is discussed for DFG modeled behaviors, and in [156–158] for mixed behaviors. In [159] a tabular method is proposed for synthesis of re-configurable control-dominated systems. It should be stressed that algorithm parallelization techniques, pipelining, as well as scheduling and binding optimization of the loop bodies are the main techniques that have to be used in combination to create high-quality coarse-grain hardware accelerators. Consequently, these techniques have attracted much attention in recent research and development in the field of the coarse-grain accelerator synthesis [160–167].

In the past, there has been a substantial body of research and development in the coarse-grain acceleration, especially related to the loop extraction and their parallel hardware implementation. For instance, the Nimble compiler [168] extracts loops for hardware acceleration and performs hardware-oriented loop optimizations (e.g. loop unrolling, fusion, and pipelining) to generate multiple optimized versions of each loop. Subsequently, it decides which loops and which loop version will be actually implemented in hardware using a quality metric accounting for the execution times in hardware and software, and hardware re-configuration time. In a similar way, BRASS [169, 170] and DEFACTO [114]

select and process loops for their realization in hardware. Also, the innermost loop synthesis was investigated using some loop transformations techniques for re-configurable co-processors through pipeline vectorization in [171]. Loop tiling was considered for re-configurable accelerators in [172].

Hardware compilation from HLLs, and specifically from C sub-sets or dialects, has been intensively studied already since the early 1990s. For instance, PRISM [173] compiled from a C sub-set to an FPGA-like architecture. Deep C compiler [174] created synthesizable Verilog from C or FORTRAN. There are also many recent works in this field. SPARK [13] compiles C to synthesizable RTL VHDL. It creates data-path/controller architectures through performing some optimizations and scheduling of the application flow graphs, when exploiting the loop transformations and other code-restructuring techniques, as well as resource binding that minimizes interconnects. In the scope of the Cameron project [175], the SA-C (single assignment C) dialect of C has been developed to support hardware compilation for image processing applications for general re-configurable architectures, when using compiler directives to describe the hardware structure. SA-C is translated to behavioral VHDL using DFGs as IR, by exploiting similar techniques to SUIF (e.g. loop transformations, pipelining, data analysis, etc.) to create optimized DFGs. The DFG nodes representing arithmetic or logic operators are then implemented as combinational circuits, loops as pipelines controlled by FSM controllers, and data transfers as communication circuitry. This results in a heterogeneous mixed-grain accelerator architecture.

Recently, the exploitation of HLL compilers for application analysis and hardware compilation gets much more popularity in the industry. Synopsys provides hardware compilation from untimed sequential C algorithms with a HLS tool called *Synphony C Compiler*. Another Synopsys tool called the *Synphony Model Compiler* provides an automated method to synthesize electronic-system-level algorithmic representations from the Simulink/MATLAB model-based design environment. The tool allows the designers to explore the area/speed tradeoffs from a single model and to automatically create RTL-level hardware implementations. The Cadence *C-to-Silicon* HLS tool creates synthesizable RTL from untimed C/C++/SystemC algorithmic specifications. The Mentor's *Catapult C Synthesis* [20] HLS tool performs an automatic compilation from C/C++ into an RTL-level hardware netlist guided by the synthesis directives. It performs loop restructuring, variable and array mapping, and scheduling. Impulse CoDeveloper [22] includes a HLS tool called the *Impulse C compiler* that is based on the standard ANSI C. It provides an interactive parallel optimizer and Platform Support Packages for a wide range of FPGA-based platforms. It supports application profiling and HW/SW partitioning, parallelization and pipelining of critical C code sections mapped to hardware, and automatic generation of the corresponding FPGA hardware. The FORTE design system [176] provides a HLS tool called *Cynthesizer* that generates RTL from a high-level SystemC-TLM design description as input. MATLAB provides a hardware compilation tool HDL Coder [177] that compiles DSP applications specified in Simulink models and MATLAB code

to their corresponding VHDL code for FPGA implementation, when exploiting pre-designed IP hardware blocks. The Xilinx *AutoESL* [25] HLS tool enables C, C++ and SystemC specifications of the applications and generates synthesizable RTL targeted to various Xilinx FPGAs devices. In parallel to the above commercial tools, some HLS tools are also researched in academia such as *LegUp* [19] and *GAUT* [178], etc. The *LegUp* is an open source HLS tool based on the LLVM compiler infrastructure [179] that generates RTL Verilog for Altera FPGA devices. The *LegUp* tool employs the basic ASAP and ALAP scheduling techniques for operation scheduling based on the LLVM IR format. The *GAUT* is dedicated specifically to DSP applications. Starting from a pure C function, *GAUT* extracts parallelism before the resource allocation, assignment and scheduling. It generates as an output a synthesizable RTL level VHDL.

There are several reasons of the inadequacy of the traditional design approaches discussed in the previous two sections for the hardware accelerator design for highly-demanding applications. Some of the major reasons are the following. While the existing APP methods and tools may reasonably well make explicit the application bottlenecks and parallelism potential, exploitation of this potential to resolve the bottlenecks, while satisfying the application-specific constraints and objectives, is a very complicated task that is not well supported. It requires a combined exploitation of many possible parallelization operations at both micro- and macro-architecture level in such a way that the result satisfies the application constraints and optimizes the required tradeoffs among its objectives. It requires an actual (partial) architecture exploration and synthesis based from one side on the application analysis results, as delivered by the APP techniques, but from the other side on the estimation of the physical, economic, etc, parameters of the architectures under exploration and construction that are involved in the constraints and objectives. To our knowledge, no method has been proposed to perform such accelerator architecture exploration and synthesis. Therefore, a novel quality-driven model-based design method is proposed in the research reported in this thesis. Another major limitation is that the HLS tools only support the micro-architecture synthesis of a single processing unit, while not taking into account for the micro-architecture, memory and communication structures and tradeoffs among them. Even in case a parallel application structure is available, the complex task-level inter-/intra-/iteration-level loops dependencies between the data and computing operations require optimum design of complex memory and communication structures for the inter-/intra-loop data transfers, as well as, the complex scheduling problems on multiple resources (e.g. several basic processors, accelerators or functional units), that are far beyond the capabilities of the traditional HLS methods and tools.

Summing up, the existing application analysis and parallelization (APP), and the high-level-synthesis (HLS) tools offer only some limited partial capabilities for supporting the design of high-performance hardware accelerators for highly-demanding applications.

2.6.3 Overview of Hardware Accelerator Designs for Highly-demanding Applications

Here, several architecture designs proposed in the past for the LDPC decoding applications are discussed. They are further used in the evaluation of the proposed design methodology.

LDPC decoder design space is a complex multi-dimensional space. Many LDPC codes have been developed for various communication system standards with several code rates (R) and code lengths (L) in different application scenarios. From the algorithmic perspective, several different iterative decoding algorithms are available with varying error-correcting performances and complexities. Finally, numerous decoder architectures and their implementations are possible. Each of the above-mentioned aspects has a different impact on the accelerator design and generates various tradeoffs. For instance, more effective error-correcting algorithms require more complex computation processes, what results in a lower performance or in an increased accelerator hardware complexity and increased energy consumption. An often need to account for various LDPC standards with several PCMs, code rates or code lengths in one system, demands from the accelerator to be (re-)configurable. The ultra-high throughput demands and other stringent requirements decide to a high degree the accelerator architecture and implementation strategies. Having particular application requirements, all the above-mentioned design aspects have to be taken into account together to design an effective and efficient hardware accelerator for the LDPC decoding application.

The belief propagation (BP) algorithms, also known as message passing (MP) algorithms, are the most popular algorithms used for the LDPC code decoding. These algorithms are more precisely described in Chapter 4 of this thesis. They were proposed by Gallager [2], with a large algorithmic level optimizations proposed recently [40–42]. One of the most popular optimization is the layer belief propagation (LBP) algorithm, that serializes the BP algorithm in terms of its message updating sequence between the check and variable node updates, what results in convergence in half the number of iterations of BP algorithm [41]. In the original BP algorithm, all the variable nodes can be updated simultaneously, followed by the check node updates. However, in the LBP algorithm, together with each variable/check node update, all the connected check/variable nodes are also updated, before the start of next variable/check node update. This way all the recent updates, i.e. the most up-to-date information is used, which accelerates the convergence process. On the other hand, the benefit of the standard BP is their high parallelism in one or other kind of nodes, thereby the possibility of exploiting more parallelism, what is actually needed for the highly-demanding applications. On the hand, the LBP algorithms are sequential, but converge faster and require less number of iterations compared to the standard BP algorithms.

Several LDPC decoder implementation strategies are researched in the past. The implementation spectrum of LDPC codes spans between the two extremes of a fully-serial and a fully-parallel realizations, with in between a large number

of partially-parallel choices. The partial parallelism can be realized at two levels, i.e. the micro-architecture and the macro-architecture level. However, the past research only exploits full parallelism at each of the two levels [44–51], or fully-serial at both levels [43]. On the other hand, in case of partially-parallel macro-architectures, the partial parallelism is exploited only at the two extremes of micro-architecture level (fully-serial or fully-parallel micro-architecture) with just one ad-hoc selected macro-architecture partial-parallelism to achieve the required performance, without actually taking into account the mutual tradeoffs between the micro- and macro-architecture level, and even more importantly, without considering the complexity of the corresponding memory and communication structures [52–83]. As a result, the research efforts then concentrated on some ad-hoc approaches to optimize the various design parameters for these limited architecture choices, without actually exploring the architectures with all their complexity result in inferior designs. The above-mentioned architectures and their features are discussed in more detail as follows.

The advantage of the fully-parallel implementations is the very high throughput, but they are very complex and costly, and extremely inflexible regarding adaptation to various code lengths and code rates. The main problem of interconnects complexity for the fully-parallel LDPC decoders has been the focus of several research works in the past, with the main aim to minimize the interconnect complexities and delays [44–51], while satisfying the high-throughput requirements. The approaches proposed in the above-cited works try to solve the interconnect problems using the physical level interconnects optimizations, such as wire partitioning or an adequate floor-planning, etc, instead of architecture-level solutions. Moreover, the fully-parallel architectures can make sense only for the short-length LDPC codes. For the long codes, they are impossible to realize due to physical limitations. For example, the fully-parallel LDPC decoder design presented in [44] for the rate-1/2 LDPC codes and even of short length of 1024-bit lead to an average net length of 3 mm, the total die size of 52.5 mm^2, the power consumption of 690 mW and delivers a throughput of 1 Gbps. The architecture was targeted to CMOS 0.16 μm technology with five metal layers. However, the LDPC codes used in the digital video broadcasting standard (DVB-S2) are of 16200-bits or 64800-bits in length, what makes it impossible to realize in fully-parallel with a reasonable area and power consumption.

On the other hand, the fully-serial approach (both at the micro-/macro-architecture level) results in a low cost and a highly flexible architecture but with an extremely low throughput [43]. The partially-parallel approaches are most popular in research community mainly for the reasons of moderate level of throughput, area and flexibility [52–83]. Different micro-/macro-architecture parallelism levels result obviously in different architectures, which is presented in most of the state-of-art research works on LDPC decoders with a kind of ad-hoc complexity, performance and power comparisons. It is worth to be noted that these architectures only exploit the macro-architecture level parallelism and only consider the fully-serial or fully-parallel micro-architectures, with an ad-hoc

selected macro-architecture level parallelism to satisfy the performance.

The above-mentioned architectures are further sub-categorized based on their micro-architecture parallelism exploitation, i.e. either the fully-serial or fully-parallel micro-architectures, to get an insight into their features. The architectures proposed in [52–60, 62] exploit the fully-serial micro-architectures with some macro-architecture level parallelism to meet the performance demands. However, to meet the high performance demands with serial micro-architecture, a huge macro-parallelism is usually required. Even exploiting the full macro-architecture level parallelism, it is impossible to meet the high performance demands. Moreover, these architectures normally operate at high clock speeds due to their no micro-architecture parallelism exploitation, which also causes a high dynamic power consumption. This makes them specifically inappropriate for low power applications. Finally, more macro-parallelism means even more complex memory and communication structures.

On the other hand, the architectures presented in [61, 63–69] exploit the fully-parallel micro-architecture and an ad-hoc selected macro-architecture parallelism to meet the required performance demands. When the fully-parallel micro-architectures are realized for high-degree check or variable nodes (i.e. computing nodes with many inputs), the clock frequency drops tremendously due to a long critical path, which drastically reduces the performance. To improve the performance of such fully-parallel processing units, the processing units are aggressively pipelined (even in some cases multiple pipeline stages) to increase the clock speed, what in turn result in throughput improvement. Since for the high-end applications, hundreds of such processors are required, usage of the fully-parallel micro-architectures results in a huge accelerator cost, due to the extremely large number of pipelined registers. It is necessary to note that each processing element requires as many pipeline registers as its processing parallelism, and not a single register. Moreover, a further substantial increase of the cost results from the facts that for the fully-parallel micro-architecture pipelining is also required on the memory and communication side due to two reasons: the memory or communication structure delays may surpass the processors delays after the processor pipelining; and to balance or synchronize the pipeline.

Moreover, the memory and communication architecture design and related complexity can not be ignored for the high-end applications. Therefore, to decrease the communication complexity, the more regular Quasi-Cyclic (QC)-LDPC codes are adopted by the most communication standards, and the decoders are designed specifically for these codes. Below, the papers that account in a way for the memory and communication architecture for QC-LDPC codes are discussed.

Several papers on architectures for processing a single sub-matrix (serial micro-architectures for CNP) of a single macro-row (macro-architectures parallelism of CNP) and a single sub-matrix (serial micro-architecture for VNP) of a single macro-column (macro-architecture parallelism of VNP) are published. While processing a single sub-matrix in isolation only requires a simple local communication network (switch) and a simple memory structure [52, 56, 57, 62, 72], it does not

solve and does not even address the problem of an effective and efficient processing of the whole PCM matrix and related required communication architecture for this aim. Also, some architectures for processing only a single (or a part) macro-row and a single (or a part) macro-column in a fully-parallel micro-architecture were proposed, what required multiple local communication networks but no global network [58, 61, 82]. However, as demonstrated in Chapter 6 of this thesis, for the demanding accelerator cases, multiple such macro-rows and macro-columns have to be processed in parallel. This requires solution of a much more complex system of combined global and local communication problems, which results in hierarchical networks and complex memory architecture.

A direct comparison of the LDPC decoder architectures presented in literature is impossible due to the multi-dimensional and multi-objective design space of the LDPC code decoders. The various dimensions are well discussed earlier. These are: the various kinds of LDPC codes (code rates and code lengths) and their code-specific implementations; various LDPC code decoding algorithms with different complexities; different measures of number of iterations and bit precisions for different bit-error-rate (BER) and frame-error-rate (FER); various application requirements; different target technologies.

Due to the above issues, it is impossible to make a fair direct comparison of these various architecture. Additionally, the various processing parallelism exploited by different architectures further influences the picture. Nevertheless, some comparisons can be found in literature, and they are mostly based on comparing the performance, power consumption, area (PPA) and in some cases, the flexibility for different codes. Although authors of most architectures proposed for LDPC in the past perform some sort of comparison based on the above-mentioned parameters, these comparisons are not fair and of low value in most of the cases. For example, comparing the architectures for one kind of LDPC codes with one for another kind that are completely different in their features and requirements.

2.7 Conclusions

In this chapter, using examples of several modern highly-demanding applications, it was shown that today's programmable processors are not able to satisfy the performance targets of many modern highly-demanding applications. To satisfy their performance/power/cost demands, massively parallel hardware multi-processors are needed. Afterwards, the main issues involved in the design of hardware accelerators for highly-demanding applications were discussed. It was shown that those issues cannot be resolved using the traditional methodologies used for hardware accelerator design, specially, "application analysis and parallelization" (APP) and "high-level-synthesis" (HLS) methods, due to the APP and HLS limited scope and capabilities. The issues were further elaborated using as an example real-life application of LDPC decoding. Based on the analysis of the issues, the requirements were formulated that have to be satisfied by an adequate accelerator design

method. Subsequently, the main concepts of the proposed design method were discussed. This method will be discussed in detail in the next chapter. Finally, an overview of the related research was presented.

CHAPTER 3

Quality-driven Model-based Multi-processor Accelerator Design Methodology

In the previous chapter, the issues of accelerator design for highly-demanding applications were analyzed and the main requirements that have to be satisfied by an adequate accelerator design methodology were formulated. This chapter discusses our novel quality-driven model-based multi-processor accelerator design methodology, which addresses the issues and satisfies the requirements. The methodology is quality-driven and model-based. It is based on the quality-driven design paradigm. It exploits the concept of a generic architecture platform, modeled using generic architecture templates, and is supported by a novel multi-objective and multi-dimensional design space exploration (DSE) framework. The DSE framework explores the various tradeoffs among the micro- and macro-architecture for processors, and the corresponding memory and communication architectures, to construct high-quality multi-processor accelerators. The complete DSE flow is discussed in detail, starting from the design requirements to the final accelerator implementation, and the importance of each possible tradeoff at each exploration point is highlighted. The advantages of the proposed design method are discussed. LDPC decoder design is briefly introduced as a case study, and the design parameters that can be influenced and decided during the DSE are discussed.

This chapter is organized as follows. Section 3.1 discusses our proposed quality-driven model-based multi-processor accelerator design methodology and the related design space exploration framework. In Section 3.2, the design methodology is extended to the design of re-configurable hardware accelerators. Section

3.3 presents the proposed multi-objective and multi-dimensional design space exploration framework. Section 3.4 discusses the proposed architecture design space exploration and synthesis algorithms for various design issues. Section 3.5 discusses the advantages of the proposed design method. Section 3.6 discusses the design decision space and the search complexity of the proposed DSE approach. Section 3.7 briefly discusses the application of the design methodology to the design of LDPC decoders and the parameters that can be influenced and decided during the DSE. Section 3.8 concludes this chapter.

3.1 Quality-driven Model-based Multi-processor Accelerator Design Methodology

In this section, we propose and discuss an accelerator design methodology which addresses the issues of accelerator design for demanding applications and satisfies the requirements of an adequate accelerator design considered in the previous chapter. This methodology accounts for the micro-architecture synthesis of basic accelerators, as well as, for the macro-architecture, memory and communication synthesis of the multi-processor accelerators. It considers the micro-architecture and macro-architecture for the processors, and the corresponding memory and communication architectures synthesis, as one coherent complex task of the accelerator architecture synthesis, and not as several separate tasks. This allows for an adequate resolution of the strong interrelationships between the micro-architecture and macro-architecture for the processors, and the corresponding memory and communication architectures, as well as, among the performance, power consumption and area.

The methodology is ***quality-driven and model-based***. It is well-known that the design productivity and the design quality are major concerns in an (embedded) system development. In order to adequately address the above issues, quality-driven system design approach is proposed by Jóźwiak [1, 34–37], with a new definition of the quality. According to the quality-driven design paradigm, ***system design*** is actually about a ***definition of the required quality***, in the sense of a satisfactory answer to the following questions:

- What (new or modified) quality is required?
- How can it be achieved?

Consequently, quality-driven design methods and tools are necessary to ensure that a system will represent the actually required quality. Therefore, the design process for demanding hardware accelerators introduced and discussed here is a specific realization of the quality-driven design process proposed and discussed in [34]. In order to bring the quality-driven design into effect, quality has to be modeled, measured and compared. To enable it, the following generic quality definition has been proposed in [34]: ***Quality of a purposive systemic solution is***

its total effectiveness and efficiency in solving the problem the solution is required for. ***Effectiveness*** is the degree to which a solution attains its goals. ***Efficiency*** is the degree to which a solution uses resources in order to realize its aims. In turn, the effectiveness and efficiency can be expressed in terms of measurable parameters, and in this way quality can be modeled and measured. ***Effectiveness*** and ***Efficiency*** of a systematic solution together decide its grade of excellence. Their aggregation expresses quality. Design space exploration (DSE) with usage of well-structured quality models makes us possible to limit the scope of subjective design decision making and enlarge the scope of reasoning-based decision making with open and rational procedures which can be computerized. In particular, quality can be modeled in the form of ***multi-objective decision models***, being partial and abstract (i.e. reduced to the relevant and/or feasible concerns and precision levels) models of the required quality, expressed in the decision-theoretical terms. Multi-objective decision models, together with methods and tools for the estimation of the design parameters of these models related to the relevant design aspects and performances, enable application of the multi-objective decision methods for construction, improvement and selection of the most promising solutions.

A very important aspect of the quality-driven system design is design reuse, because it simultaneously enhances the system quality (due to the "maturity" of the reused designs) and the development efficiency (due to reuse of results of some development phases that are not necessary to be repeated). Therefore, our accelerator design methodology exploits a mixture of design reuse and synthesis. Generic system solutions, and especially ***generic system platforms*** for particular problem classes and ***generic architecture templates*** being their models, are among the major enablers of an adequate mixture of design reuse and synthesis. Since the generic templates are pre-designed based on the application class analysis, they can be reused to organize, direct and speedup the accelerator development process for each specific application of the class. Since they are generic, they and their parts can be adequately instantiated to (better) suit a particular application of a given class, but also some new application-specific modules may be added. The general form of a generic template constrains the solution search space to such a degree that the construction of particular solution instances for particular applications can be efficiently performed through an appropriate instantiation of the generic architecture template, and computation process scheduling and mapping on the instance of the template [34]. More general templates can adequately support larger application classes, which makes them better economically justified, as their non-recurring engineering (NRE) costs can be shared by more applications. On the other hand, more specific templates can be more effective and efficient in serving a particular application. The generic template based system approach to application-specific system development is thus well motivated both from the technological and economical viewpoint.

In order to explore the multi-objective and multi-dimensional design space of hardware multi-processor accelerators effectively and efficiently, the generic

model-based design approach is followed. According to Jóźwiak [34] well structured models of the required, proposed or delivered design quality can serve to conceptualize, denote, analyze and communicate the customer's and designer's ideas, to show that the requirements and designs are meaningful and correct, to guide the design process, to enable the explicit and well-organized design decision making with open and rational procedures, to enable design automation, etc. A general model of the required quality is formed by the system requirements. It is possible to distinguish three sorts of requirements: functional, structural, and parametric (involving some physical performance parameters, economic parameters, etc.). All the three sorts of requirements impose limits on the structure of a required solution, but they do so in different ways. The structural requirements define the acceptable or preferred solution structures directly, by limiting them to a certain class or imposing a preference relation on them. The parametric requirements define the structures indirectly, by requiring that the structure has such physical, economic or other properties (described by values of some parameters) as fulfill given constraints and satisfy stated objectives. The functional requirements also define the structures indirectly, by requiring the structure to expose a certain externally observable behavior that realizes the required behavior (fully - if the requirements are formulated as hard constraints, or to a certain degree - if they are formulated as objectives). The model of the required quality, represented by the design requirements, models the design problem at hand through the imposition of a number of requirements on the acceptable or preferred solutions, and so it can also be considered as an abstract model of a solution to the problem. Since such a model limits the space of acceptable or preferred solutions to a certain degree only, it models many solutions concurrently. Each of them fulfils all the hard constraints of the model, but different solutions can satisfy its objectives to various degrees.

For the reasons discussed above, the proposed accelerator design methodology adopts the quality-driven model-based design exploration and architecture synthesis approach proposed in [34], and exploits the concept of generic architecture templates. The quality of the accelerator required is modeled in the form of its design requirements involving the demanded accelerator behavior, and the structural and parametric constraints, objectives and tradeoffs to be satisfied by its design.

Accelerator architecture synthesis consists in the creation of an accelerator structure specification at the architecture level that supports the realization of the accelerator's behavior as specified by its behavioral requirements, and fulfills the structural and parametric requirements to a satisfactory degree. This structural specification defines:

- a set of architectural structural resources (i.e. computation, memory and communication resources),
- an exact composition of the architectural resources to form the architecture platform, and

- a corresponding mapping of the required computation processes on the so constructed architecture platform and a schedule of the computation processes.

To perform the accelerator architecture exploration and synthesis effectively and efficiently, the original accelerator requirements have to be analyzed and a partial (reduced to only certain architecture related concerns) and abstract (reduced to the necessary and/or possible precision level) architecture-level model of the requirements being adequate for the architecture design issue has to be constructed. The actual accelerator architecture exploration starts with such an *abstract model of the architecture design issue* composed of:

- an *abstract system behavior model* representing a system of computations that have to be realized;
- an *abstract accelerator hardware platform model* representing selected generic architecture templates; and
- an *abstract decision model* composed of a set of constraints, objectives and tradeoff preferences related to all accelerator characteristics important for the architecture synthesis issue.

The decision model defines how the hardware resources and the mapping and scheduling of the computational components onto the hardware resources are constrained and interrelated, and represents the designer's preferences and aspirations. Its constraints and preferences have to be fulfilled to a satisfactory degree by each acceptable architecture supporting the required computational processes.

Based on the analysis results of the so modeled required quality, the generic architecture templates of a multi-processor and its modules are instantiated and used to perform the DSE that aims at the construction of one or several most promising accelerator architectures supporting the required behavior and satisfying the demanded constraints and objectives. This is performed through analysis of various architectural choices and tradeoffs. The proposed approach considers the micro- and macro-architecture synthesis and optimization, as well as, the computing, memory and communication structures' synthesis as one coherent accelerator architecture synthesis and optimization task, and not as several separate tasks, as in the state-of-the-art methods. This allows for an adequate resolution of the strong interrelationships between the micro- and macro-architecture, and computation unit, memory and communication organization. It also supports an effective tradeoff exploitation between the micro- and macro-architecture, the memory and communication architecture, as well as, between the various aspects of accelerator's effectiveness and efficiency. According to our knowledge, the so formulated accelerator design problem is not yet explored in any of the previous works related to hardware accelerator design.

In more precise terms, the proposed *quality-driven model-based multi-processor architecture design method* involves the following core activities:

- *design of a pool of generic architecture platforms and their main modules, and platform modeling in the form of an abstract architecture template* (once for an application class)
- *abstract requirement modeling* (for each particular application),
- *generic architecture template and module instantiation* (for each particular application),
- *computation scheduling and mapping on the generic architecture template instance* (for each particular application and template instance)
- *architecture analysis, characterization, evaluation and selection* (for each constructed architecture),
- *architecture refinement and optimization* (processing, interfacing, and memories abstraction refinement and optimization - for the selected architectures only).

To perform the accelerator architecture exploration and synthesis effectively and efficiently, a pool of generic architecture templates corresponding to a given application class and their main resources (processors, memories and communication resources) are developed and modeled in advance. The generic architecture templates and units are pre-designed by analyzing various applications of this class, and particularly, analyzing the application's required behavior, and ranges of their structural and parametric demands. Each generic architecture template specifies several general aspects of the modeled architecture set, such as presence of certain modules types and the possibilities of the module's structural composition, and leaves other aspects (e.g. the number of modules of each type or their specific structural composition) to be derived through the DSE in which a template is adapted for a particular application. To prepare an adequate set of templates and models of their basic units, a significant analysis of the application class and possible corresponding conceptual accelerator designs is necessary. In fact, the generic templates represent generic conceptual accelerator designs which become actual designs after adequate further template instantiation, refinement and optimization for a particular application. The adaptation of a generic architecture template to a particular application with its particular set of behavioral and other requirements consists of the DSE through performing the most promising instantiations of the most promising generic templates and their resources to implement the required behavior, when satisfying the remaining application requirements. In result, several most promising architectures are designed and selected that match the requirements of the application under consideration to a satisfactory degree. The significant analysis and design effort required to design the architecture templates is however compensated due to enabling an effective and efficient DSE when using the templates.

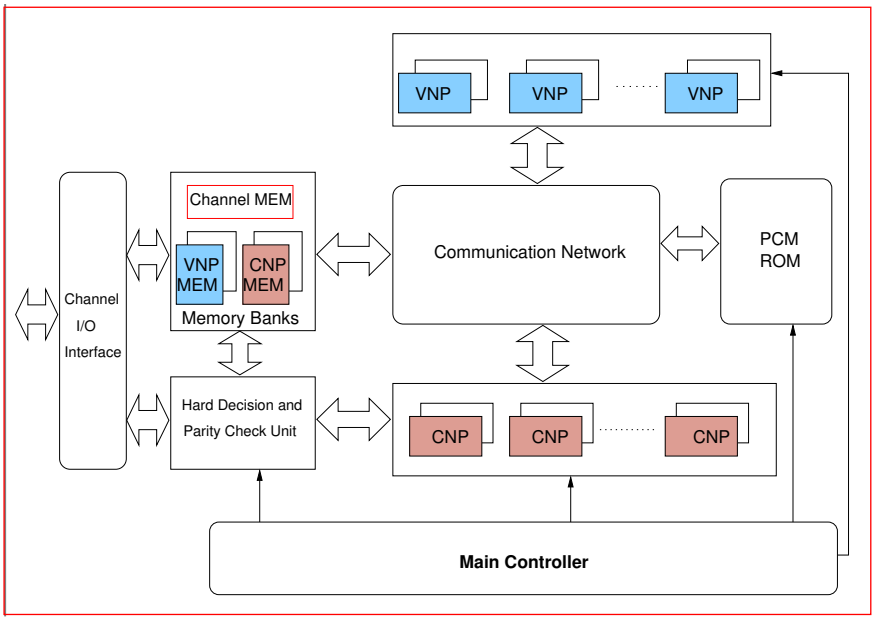

Figure 3.1: Example of a generic architecture template for LDPC decoding accelerators

Figure 3.1 shows an example of a generic architecture template for an LDPC decoding accelerator prescribing the presence and general structural organization of the architectural resources. It involves parameterized variable node processors (VNP) and check node processors (CNP), memories and a communication network for the communication among various processors and memories, ROM that can be configured to particular PCM, Hard Decision and Parity Check Unit, as well as, the Main Controller and Channel I/O Interface (see Section 4.3 of Chapter 4 for more details). Different instances of the generic architecture templates and their processing, memory and communication modules define different specific accelerators. Also, the original accelerator requirements, that may be very complex and include many details not relevant for architecture synthesis, have to be analyzed, and a much simpler abstract model of the behavioral and parametric requirements being adequate for the architecture design issue has to be constructed to enable an effective and efficient accelerator architecture exploration. The actual architecture exploration starts with such abstract model of the architecture design issue constructed in advance (see Figure 3.2).

For the development of the generic architecture templates and its modules existing system and circuit design methodologies and electronic design automation tools can be used. This involves the design and implementation of the generic template modules in hardware description languages (HDL) like VHDL or Ver-

ilog, which include optimized parameterizable processing units, communication and memory elements, and other generic architecture modules for an application class. It also involves the design of the top-level generic architecture templates. The generic HDL models can then be synthesized for different set of parameters, what enables estimation of their physical parameters, and will be used to evaluate the quality of various possible solutions. Moreover, the architecture models can be used for simulation (executable models) and the actual hardware prototyping or implementation as they are synthesizable models. The generic architecture templates are somewhat more difficult to develop than the instance templates due to their generality for a given application class. However, without such generic models it would be impossible to make any reasonably accurate design decisions on the architecture proposals. The carefully constructed and characterized generic architecture models make possible evaluation of complex architecture decisions with the necessary accuracy. Some practical examples of known generic architecture platforms and templates are numerous FPGA platforms (e.g. from Xilinx and Altera, etc) and (re-)configurable ASIP processors (e.g. from Tensillica or SiliconHive, etc).

To start the actual architecture exploration and synthesis process, the abstract behavioral and parametric requirements of a given application are analyzed to decide the most promising instantiations of the most promising generic templates and their resources (see Figure 3.2). Based on this analysis, the designer makes a proposal of one or more promising generic architecture template instances and their resource allocation that are expected to be adequate to realize the required accelerator behavior and satisfactorily fulfill the parametric and structural requirements. His decision is implemented through a corresponding instantiation of the generic architecture template and of its modules. Moreover, the network of computations represented by the accelerator behavior model is appropriately distributed over the structure of modules of each of the promising instances of the generic architecture templates and scheduled, when observing the parametric constraints, objectives and tradeoff preferences, to define one or more actual accelerator architectures that satisfy the specific (structural, physical, etc.) hard constraints and optimize the objectives of the quality model. Each actual accelerator architecture is defined through the selected template configuration (i.e. the selection and interrelationships of modules of a particular template), template module configuration, as well as, assignment of the required computations to the template modules and their schedule. The so constructed architecture is subsequently examined and analyzed to check to what degree the constraints, objectives and preferences are satisfied, this way, to provide feedback on the exploration result to the designer. In this architecture synthesis process, both the available accelerator resources, and the objectives, constraints and tradeoff preferences are imposed by the designer. On the other hand, the mapping and scheduling decisions determine the actual accelerator resource requests. To be acceptable, the resource requests must match in a satisfactory way the pool of the available resources, in the light of the objectives, constraints and tradeoff prefer-

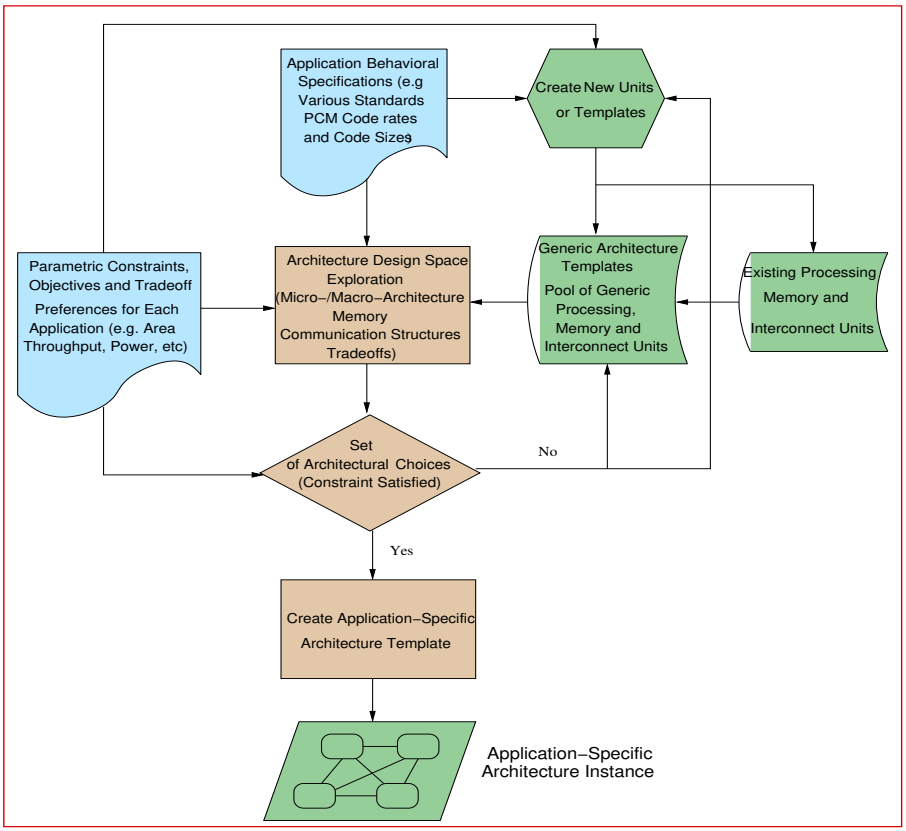

Figure 3.2: Architecture exploration framework of the proposed multi-processor accelerator design methodology

ences. If this is not the case, the designer may decide to propose new promising template instances (e.g. with more or more effective resources), create new more adequate units, modify templates or create new templates, or even modify the design requirements, and subsequently, to perform the next exploration cycle. If the requirements are satisfactorily fulfilled by one or more of the created architectures, some of the satisfactory architectures are further analyzed, refined and optimized, and finally, one of them is selected to be the actual application-specific architecture instance for the application considered (see Figure 3.2). This way, the pool of generic architecture templates and their corresponding parameterized processing, memory and interconnect resources available for a given class of applications is adapted to a particular application characterized by its particular set of behavioral and other requirements (see Figure 3.2).

During the DSE the major aspects of the multi-processor accelerator design,

being its micro-architecture, macro-architecture, memory architecture and communication structure, as well as, the tradeoffs among these aspects are considered and decided to optimize the accelerator quality expressed with its quality metrics (such as throughput, operation frequency, area, energy consumed, cost etc.) and information on the required tradeoffs. It is important to stress that these macro- and micro-architecture decisions are taken in combination, because both the macro- and micro-architecture decisions influence the throughput, area, and corresponding other important parameters, but they do it in different ways and to different degrees. For instance, by a limited area, one can use more elementary accelerators, but with less parallel processing and related hardware in each of them, or vice versa, and this can result in a different throughput and different values of other parameters for each of the alternatives. Therefore, during the DSE, several different promising combinations of the micro- and macro-architectures are constructed and analyzed.

Finally, to decide the most suitable architecture, the promising architectures constructed during the DSE are analyzed and characterized in relation to various metrics of interest and basic controllable system attributes affecting them (e.g. number of accelerator modules of each kind, clock frequency of each module, communication structures between modules, schedule and binding of the required behavior to the modules etc.), and the results of this analysis are compared to the design constraints and optimization objectives. This way, the designer receives feedback, composed of a set of constructed architectures and important characteristics of each of the architectures, showing to what degrees the particular design objectives and constraints are satisfied by each of the them. This feedback is used by the designer to control the further progress of the architecture exploration and synthesis process, and to decide the most suitable architecture. If all the constraints and objectives are met to a satisfactory degree by some of the constructed architectures, the most suitable of the architectures satisfying the requirements is selected, further analyzed, refined and optimized to represent the actual detailed design of the required accelerator. This way, the architecture DSE results in creation of an architectural structure that defines a specific composition of the computation, memory and interconnection resources at the macro- and micro-architecture level that supports the application's behavior required and satisfies its parametric constraints and objectives to a satisfactory degree.

3.2 Extension to (Re-)configurable Multi-processor Accelerator Design

The re-configurable accelerator DSE aims at deciding one globally optimal adaptable accelerator architecture that adequately satisfies the requirements for all the particular accelerator instances required for a given application (see Figure 3.3). It decides one globally optimal adaptable accelerator architecture for all the ac-

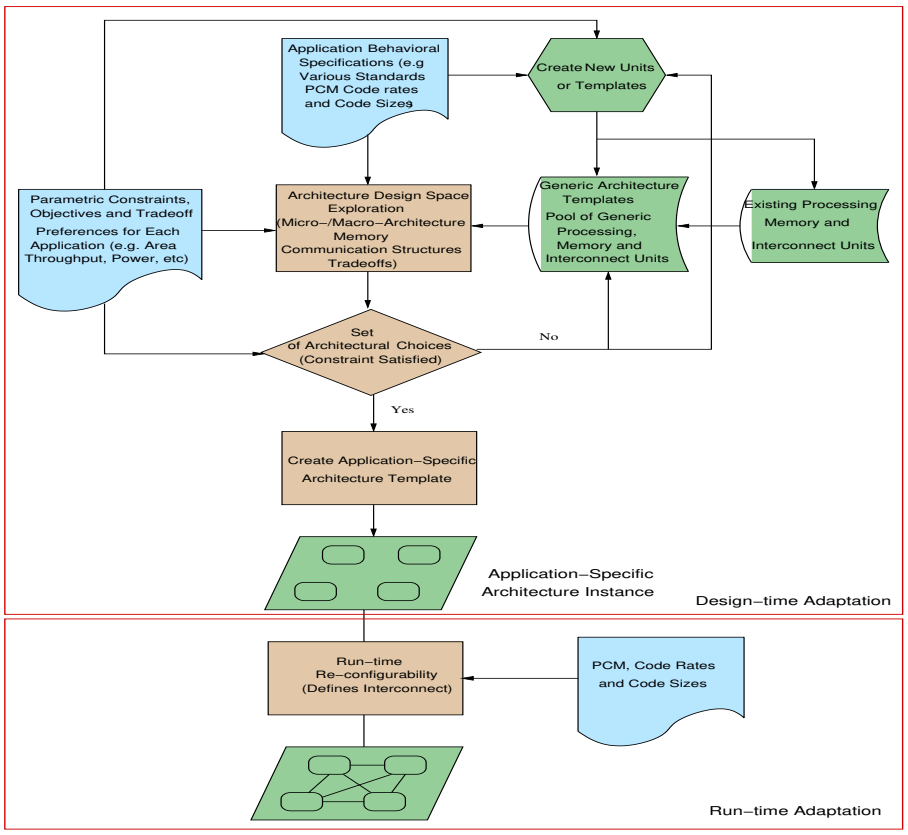

Figure 3.3: Architecture exploration framework of the proposed re-configurable multi-processor accelerator design methodology

celeration cases required, and not a set of the best individual accelerators for each particular acceleration case and their combined reconfigurable implementation. For instance, for the rate-7/8 672-bit LDPC code of IEEE 802.15.3c standard that has the highest check node degree (number of inputs) of 32 can be efficiently realized using serial micro-architecture, while the rate-1/2 672-bit code could be more efficiently realized in a parallel micro-architecture. Since the two architectures have not much in common, their joint reconfigurable implementation as an ASIC would involve extensive computation, communication and reconfiguration resources, and would not be efficient. In case of the LDPC decoding, the goal is to find one most promising reconfigurable accelerator architecture to adequately satisfy the combined requirements of all the codes for a particular application, and not a set of the best individual accelerators for each code separately and their combined reconfigurable implementation. Moreover, it also accounts for the

total re-configuration resources during the DSE. The actual DSE is performed as follows.

Using the DSE approach of the non-reconfigurable accelerators as discussed earlier, it first determines suitable accelerator architectures for the particular required acceleration cases that are most demanding in terms of throughput, area, etc, (i.e. suitable decoding accelerators for the particular most demanding code classes in case of LDPC). This results in finding a sub-set of suitable architectures for each of the most demanding acceleration cases that satisfy hard constraints. The architectures determine alternative sets of the necessary resources and configurations of the reconfigurable accelerator that are able to satisfy the requirements of the most demanding acceleration cases, i.e. the resources and their configuration that must be present in the reconfigurable acceleration system under construction. Subsequently, some sub-sets of promising architectures for the less demanding acceleration instances (codes) are explored and analyzed to determine similarity in their structure to the most demanding accelerator architectures. Exploitation of the architecture similarity for different acceleration cases required for a given application is of crucial importance for an adequate reconfigurable accelerator construction. It is necessary to efficiently re-use the processing, communication and memory resources for different required acceleration cases, and to reduce the re-configuration resources.

Finally, knowing the necessary resources and the architecture similarity for different required acceleration cases, the reconfigurable accelerator architecture construction adequately adapts the accelerator architecture of the most demanding case(s), or adapts and combines the accelerator architecture of the most demanding case(s) and several other individual accelerator architectures in such a way, as to realize all the required acceleration cases (decoders for all the required codes) and their particular requirements, but at the same time to minimize the total processing, memory, communication and reconfiguration resources, and the related power consumption, costs, etc.

As explained in Section 2.2 of Chapter 2, for ASIC-implemented re-configurable accelerators the total re-configuration when using very general re-configuration resources as in FPGAs is impractical. Therefore, to instantiate a particular accelerator, the aim is to (almost) only re-configure the interconnections between the basic processing units to form larger processing units, and between the processing units and memories, and to avoid the low-level reconfiguration inside of basic processing units (e.g. to form the basic processing units from gates). For the LDPC codes this is possible due to the strong similarities in the code structures, decoding algorithms and operations required for different code rates, code lengths and other requirements. Due to the re-configuration limited to the interconnect configuration at the level of basic functional units, the re-configuration resources remain limited, and the re-configuration overhead in the form of the configuration controller and configuration information remains low.

3.3 Multi-objective, Multi-dimensional Design Space Exploration

In the previous two sections, the design methodology and its design space exploration (DSE) framework for the multi-processor hardware accelerators are generally discussed. In this section, the core activities of the proposed multi-objective and multi-dimension DSE framework are more precisely discussed. The DSE framework is multi-dimensional in the sense that it jointly considers all the design dimensions: the processor, memory and communication sub-systems, as well as, the mutual complex tradeoffs among them. It is multi-objective in the sense that it targets various performance, area and power constraints/objectives and tradeoffs among them through controlling the basic parameters that influence them.

The multi-dimensional DSE enables to make adequate design decisions about all design elements relative to their significance and impact from the total solution viewpoint. For example, for the applications involving complex memory and communication issues, minimization of the processing resources for a certain performance level may influence the memory and communication cost significantly. Therefore, the DSE is performed considering the global optimization objectives and tradeoffs among all the design dimensions, considering their relative influences and significance. This cannot be achieved with the limited exploration capabilities offered by the today state-of-the-art architecture synthesis methods and tools (considering in most cases only the processing resources as being representative to considering the overall system resources). Moreover, in each of the design dimensions further explorations are possible to exploit other various complex tradeoffs inherent to the dimension. For instance, the performance/area/power is influenced in a different way and to a different degree by each micro-/macro-architecture decision regarding processors for a certain performance level. More importantly, while considering the total solution quality, adequate decisions have to be made regarding the communication and memory architecture for a particular processing micro-/macro-architecture, as well as, the mutual tradeoffs among them. Therefore, we account in the design exploration for all the above discussed and similar issues related to all design elements, and integrated the corresponding adequate mechanisms into the multi-objective multi-dimension DSE framework.

To adequately serve applications that demand ultra-high throughput, massively parallel hardware multi-processors are required. If these applications involve massive operations on data and complex irregular relationships between the data and computing operations on the data (as e.g. in the LDPC decoding), the communication and memory influence on the performance, area and energy consumption tends to dominate the processor influence. Therefore, in the design of high-performance accelerators for the data and communication intensive applications all the architecture and mapping decisions regarding processors should be made in favor of reducing the complexity of the communication and memory

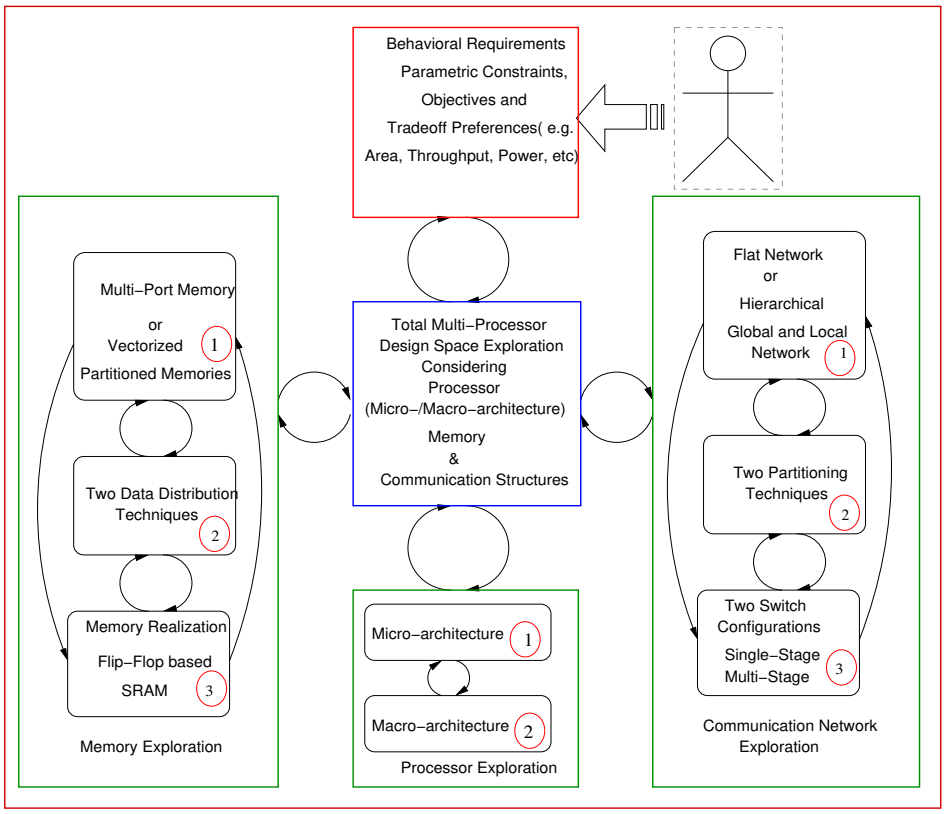

Figure 3.4: Multi-objective and Multi-dimension design space exploration framework

structures. For some less memory or communication intensive applications, the decision can be made in favor of computing hardware. These decisions among others are the scheduling and mapping decisions regarding the processors and the data allocation and mapping decisions regarding memories, etc. The multi-objective and multi-dimensional DSE is able to take adequate design decisions based on the actual influence and significance of each design element, and this way to control the global quality and tradeoffs in multiple dimensions.

The multi-objective and multi-dimensional DSE framework is represented in the form of a flow diagram in Figure 3.4. To start the actual architecture exploration and synthesis process, based on the initial requirement analysis and information on values of the design element characteristics, the designer makes a proposal of a certain number of particular instances of processors that would be sufficient to realize the required system behavior while satisfying the design's

parametric requirements. The processors have to handle the overall computation load. The number of processors and the processor type directly influence the performance. Therefore, the DSE seeds with a particular processing parallelism represented by the number and type of processors, that seems to be sufficient to realize the required performance. The parallelism can be realized through a number of promising micro-/macro-architecture combinations, as shown in Figure 3.4. Since a generic template-based design approach is followed, the generic processor templates make it possible to instantiate many different architecture instances with different micro-architecture level parallelism for a given application task (as e.g. in case of LDPC decoding, for the check node and variable node computations). Each micro-/macro-architecture combination influence the performance and other related attributes to different degrees. For instance, increase of the micro-parallelism for a specific task can reduce the number of clock cycles to execute the task and this way improve the performance. However, it is accompanied by an increase in resources, as well as, the increase of critical path delays (what results in the reduction of the processor clock speed), thereby negatively influencing the overall performance. Tradeoffs of this kind have to be considered during the micro-architecture exploration and synthesis, as shown in Figure 3.4. Moreover, the application or designer can impose the clock speed as a hard constraint, and in consequence, the micro-architectures that do not satisfy this constraint will be directly pruned away, reducing the search space. Also, the tradeoffs at the macro-architecture levels have to be considered together with the micro-architecture parallelism. In addition to the mutual tradeoffs between the micro-/macro-architecture levels, the processing architecture at both levels can also influence the related memory and communication structure in various ways and to different degrees. For instance, for a high micro- and low macro-parallelism, the local communication complexity may be high compared to the global, and vice versa. All these tradeoffs regarding processors and their significance will be explained in detail in Chapter 5.

An important exploration decision related to the processors is the task scheduling and task mapping on the processors (e.g. in case of LDPC decoding the mapping of Tanner graph nodes to check and variable node processors). Two kinds of schedules, namely the tight schedules (TS) and relaxed schedules (RS) are employed in the proposed DSE framework. The tight scheduling is performed through the exploitation of any possibility to maximize the processor utilization. This would result in the reduction of the total schedule length and this way enhances the performance. Alternatively, a reduction in the number of processors can be achieved for a fixed performance. However, it may increase the complexity of the communication and memory architectures as discussed earlier. Therefore, we refer to this type of schedule as the tight schedule, in which the processor utilization is maximized (equivalently minimize the schedule length for the allocated resources), and then must guarantee the necessary data transfer bandwidth at any cost of memory and communication. In the tight schedules, there is no possibility for the task rescheduling to processors. On the other hand, in the re-

laxed schedules, some rescheduling possibility of tasks to processors is preserved (while still meeting the performance constraints), thereby utilizing this scheduling freedom for the memory or communication costs reduction, in case the memory or communication costs dominates the processors cost. Through exploitation of the TS and RS schedules, the following two aims are realized. For the processing intensive applications, i.e. when the processors costs have the major influence on the overall system costs, the aim is to decrease the cost of the processing resources through applying the proposed TS scheduling approach. For the memory and communication intensive applications, i.e. if the memory or communication costs dominate the processors costs, the aim is to tradeoff the processors costs for the memory and communication costs through applying the relaxed scheduling.

All the information from the processor exploration (e.g. the number of processors and each processor type and the multiple scheduling decisions as described above) are passed to the total DSE that acts as a total system quality controller and decision maker, as shown in Figure 3.4.

The processor micro-/macro-architecture decisions and the associated initial scheduling and mapping decisions are passed through the total multi-processor DSE framework to the communication and memory architecture exploration and synthesis engine, as shown in Figure 3.4. The communication and memory architecture exploration and synthesis aims at finding an adequate communication and memory architecture to satisfy the required bandwidth among the memories and processors or processors and processors for the instantiated micro-/macro-architectures, while taking into account the processor scheduling and mapping decisions. The memory exploration engine resolves the memory access bottlenecks and ensures the required memory bandwidth in the presence of complex interrelationships among the data and computing operations through effective data allocation and mapping in multiple vector/multi-bank memories, supported by the communication exploration engine (see Figure 3.4). Specifically, we incorporated into the memory and communication architecture exploration framework, some novel memory and communication design strategies that ensure the memory and communication scalability for the massively parallel hardware multi-processors required for the ultra-high (multi-Gbps) performance applications. The particular strategies of the memory and communication exploration and the related tradeoffs will be explained in detail in Chapter 6 on the memory and communication architectures.

All decisions regarding memory and communication, as well as, re-scheduling decisions (if any) about processors are passed to the total design space exploration engine for each architecture instance. The total design space exploration has a built-in hardware estimator that computes the contribution of the individual design element to the total design area (cost), delay/throughput, power consumption, etc, to see their relative impact on the overall system quality. This way through the (partial) construction, analysis and evaluation of a number of promising architectures, an architecture that satisfies all the design constraints and objectives in the best possible way is actually constructed and selected.

All the architecture related information (e.g. the number and processor types, the memory and communication architectures) of the selected architecture are passed to the architecture template instantiation engine that creates the complete top-level RTL description of the selected architecture, that could further be used by the back-end synthesis and place and route tools for ASIC synthesis or for rapid prototyping using FPGAs, as shown in Figure 3.2. This way a high-quality hardware accelerator for a particular set of requirements can be very effectively and quickly created through the proposed novel quality-driven model-based multi-processor design methodology supported with a novel multi-objective and multi-dimensional DSE framework.

3.4 Architecture Design Space Exploration and Synthesis Algorithm

The proposed design space exploration (DSE) approach is not a "generate and check" approach in which the architectures are exhaustively or randomly generated and then checked to select the best of them, but it is a careful construction, analysis and selection approach in which only a limited set of the most promising architectures is stepwise constructed, analyzed and selected. The careful construction, analysis and selection of the (partial) solutions is a specific implementation of the quality-driven design decision-making process described in [35]. In this process, the design problem is decomposed into a number of interrelated issues. For each of the issues an issue quality model is extracted from the quality model of the total problem to guide the design decision making in the scope of this issue. The quality models involve the behavioral, structural and parametric parts. Using the issue quality models the set of issues is solved and the issue solutions are then composed to the complete problem solutions.

Unlike the traditional top-down design approach, in which the solutions (architectures) are generated and iteratively refined based on only (or mainly) the behavioral and parametric specifications, the proposed approach constructs solutions (architectures) heavily based on the structural specifications and related structural models, and not only on the behavioral and parametric specifications. The architecture construction is guided by the earlier developed quality models involving: behavioral models, structural (architecture) models and parametric models. The DSE framework operates on these models to perform the actual architecture synthesis. This way, our approach is a "meet in the middle" approach, which is neither a fully top-down nor a fully bottom-up approach, but a wise combination of these two approaches. Like in the bottom-up design approach, the structural models of the basic processing, memory and communication elements are constructed and characterized in advance (modeled in a hardware description languages like e.g. Verilog/VHDL). These basic elements are however generic (parameterized) and can be instantiated to suit different requirements.

Also, a generic top-level architecture template is constructed in advance. Then, the DSE is performed in the top-down manner to construct and explore different architectures, starting from the applications behavioral requirements and related parametric constraints and objectives, but only constructing some of the most promising architecture instances defined by the generic architecture template and its generic modules, and analyzing the architectures based on the known module characteristics.

Since this research does not focus on application parallelization that is reasonably covered by related research (see Section 2.6 of Chapter 2), it is assumed that the application specification at the task level is available in a fully parallel form, with parallelism exposed for both of the architecture levels, i.e. the micro- and macro-architecture level. The major exploration activities in the DSE framework involve the construction, analysis and the selection of various most promising combinations of application-specific processors micro- and macro-architecture instances, and the corresponding memory and communication architecture instances realizing the application's behavior in the light of the specific design constraints and objectives.

The quality of the required accelerator is represented by the following requirements: the behavioral accelerator specification in an adequate parallel form, the required accelerator throughput and frequency, as well as, the required tradeoff between the accelerator's area and power consumption, and the structural requirement to be constructed as one of the possible instances of the generic architecture template and its modules. In a large majority of practical cases, the throughput and clock speed are the hard constraints that must be satisfied, while the area, power and their mutual tradeoffs are considered as the design objectives that have to be optimized. In these cases, the effectiveness of an accelerator is represented/measured by the throughput (T_{cstr}) and frequency (F_{cstr}) constraints, while the area, power and their mutual tradeoffs reflect the efficiency of the accelerator. This way, the required accelerator quality is modeled, and this model of the required quality is then used to drive the overall design process that carefully step-wise constructs the most promising architectures. Any other formulation of the required quality is also possible, like for instance performance optimization for a given limit on resources. However, in most of the practical hardware accelerator design problem formulations, the throughput and clock speed are considered as the hard constraints, while the area, power, and the tradeoffs between them as optimization objectives. Therefore, this formulation of the required quality was used to perform the actual DSE in the design experiments.

Before further discussing the DSE and synthesis algorithm, the multi-objective decision making (MODM) problems, as well as, the related basic definitions and types of solutions (alternatives) are briefly introduced. This will help in understanding of the decision-making approach at each step of the DSE and synthesis algorithm.

MODM problems are usually formulated as:

$$\min_{x \in S} z(x) = \min_{x \in S} [z_1(x), z_2(x), ..., z_k(x)]^T \tag{3.1}$$

$$\textit{subject to: } g_j(x) \geq 0,\ j = 1,2,.....,m,\ \textit{and } x = (x_1,.....,x_n) \in S \tag{3.2}$$

where $x \in S$ is an n-dimensional vector of decision variables, which represents an alternative, and S is the decision space; $z_l(x) = l^{th}$ is *objective function,* $l = 1,2,...,k$; $g_j(x) = j^{th}$ *constraint function,* $j = 1,2,...,m$; and $z(x) = [z_1(x), z_2(x), ..., z_k(x)]$ is the vector of objectives to be optimized.

There are many different types of solutions (alternatives) distinguished in MODM problems. However, here the two types of solutions (alternatives) will be introduced that make the basis for the design decision-making in the DSE algorithm.

Definition 1. Feasible Solution: A ***feasible solution*** is a solution that meets or exceeds the decision maker's (DM) minimum expected level of achievement for all decision criteria. Criteria represent the rules of acceptability or standards of judgment for the alternatives. The solutions satisfying the constraint $g_j(x) \geq 0$ constitute a feasible decision space $S' \subset S$.

Definition 2. Non-dominated (Efficient) Solution: A feasible solution (alternative) $x^{(1)}$ ***dominates*** another feasible solution (alternative) $x^{(2)}$, if $x^{(1)}$ is at least as good as $x^{(2)}$ with respect to all criteria and is better than $x^{(2)}$ with respect to at least one criterion. A ***non-dominated solution*** is a feasible solution that is not dominated by any other feasible solution. That is, for a non-dominated solution an increase in the value of any one criterion is not possible without some decrease in the value of at least one other criterion.

Definition 3. Ideal Solution: An ideal solution is the vector of individual optima obtained by optimizing each objective function separately ignoring all other objectives in the feasible region. That is a point $z^\circ = (z_1^\circ, z_2^\circ, ...z_k^\circ)$ is an ideal vector, if it minimizes each objective function z_i in $z(x)$, i.e. $z_i^\circ = min(z_i(x)), x \in S, i \in [1, ..., k]$. In most practical cases, the ideal objective vector corresponds to a non-existent solution.

Definition 4. Dominance Relation: The dominance relation "alternative $x^{(1)}$ dominates $x^{(2)}$" ($x^{(1)} \succeq x^{(2)}$) is defined as

$$x^{(1)} \succeq x^{(2)} \leftrightarrow \forall i \in [1, ..., k] : z_i^{(1)} \leq z_i^{(2)} \wedge \exists i \in [1, ..., k] : z_i^{(1)} < z_i^{(2)} \tag{3.3}$$

Definition 5. Pareto Optimality: A solution $x^* \in S$ is Pareto-optimal if for every $x \in S$, $z(x)$ does not dominate $z(x^*)$, i.e. $z(x) \not\succeq z(x^*)$.

In the DSE algorithm, the decision making on a particular alternative for a certain design issue is based only on the hard constraints and dominance relation, and tradeoffs regarding the optimization objectives. In most cases, the hard constraints are on the throughput and the clock speed, and the optimization objectives are related to the area and power consumption, and tradeoffs between

them. Violation of any of the hard constraints would rule out a potential solution. The dominance relations are devised for each of the design issues. It will be explained latter how the hard constraints and dominance relations are used in pruning the design space towards the Pareto-optimal (non-dominated) partial solutions at each step in the DSE algorithm for each particular issue. The architecture decisions are evaluated based on the values of the parameters of the pre-designed architecture models for various partial solutions at each step of the design. The parameters set consist of $\{F_{max}, CC, A, PW@F_{max}\}$ for a given computation task, where F_{max} represents the maximum clock speed of the processor, CC represents the number of clock cycles required for a given computation task, A represents the (resources) area, PW represents the total power consumption at F_{max}. Some parameters may not be applicable for certain design issues, for instance CC in case of memories. First, the design issues that are considered during the DSE are described, followed by their resolution method based on the hard constraints (constraint satisfaction) and dominance relation.

The architecture design space exploration and synthesis is composed of the three main stages and is performed as follows. Each of the three main stages corresponds to one of the main design issues (sub-problems) that have to be solved to result in a complete accelerator architecture solution:

1. decision of the processing engine architecture, and basically, decision on the processing parallelism, which decomposes further into:
   i. decision on the micro-architecture of each particular processor, and
   ii. decision on the multi-processor accelerator macro-architecture;
2. decision of the memory and communication architecture, and involves:
   i. coarse communication architecture design (decision on hierarchy and partitioning)
   ii. memory and communication refinement and final synthesis
3. final selection and actual composition of the complete accelerator architecture.

The design decision (search) space can be represented in the form of a search tree, as shown in Figure 3.5. It represents the various kinds of possible architectural decisions at each level (depth) of the tree. It also shows the number of possible decisions at each particular tree level. The tree grows in breadth at each step of the design as the number of combinations grows. For instance, for each of the m possible micro-architecture decisions at *level-1* of the search tree, there are n possible macro-architectures at *level-2* of the search tree, as shown in Figure 3.5, which together results in $m.n$ possible micro-/macro-architecture combinations. Moreover, for a particular micro- and macro-architecture combination, there are several compatible memory and communication architectures

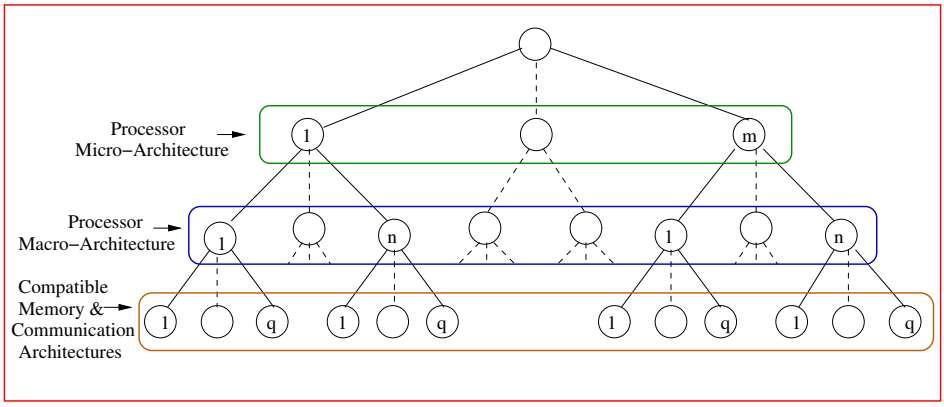

Figure 3.5: Design space exploration search tree representing particular architectural decisions at each level of the tree

possible, which are represented by the leaf nodes of the search tree. In Figure 3.5, they are shown as q possible decisions for each of the processor's micro- and macro-architecture decision. One complete path from the root node to the leaf represents a complete architecture composed of various decisions at each level of the tree. In the proposed approach, this search tree remain implicit, in the sense that it will never be fully constructed or traversed. When examining this tree, the proposed approach resembles the breadth-first-search (BFS), but only in relation to the search organization. First, a sub-set of the most promising micro-architectures is decided, which represents some of the decisions at the same level of the search tree, i.e. *level-1*. The key property of this search is that it proceeds by constructing and testing each node that is reachable from a parent node (partial architectures) before it actually expands any of the children. Therefore, the search tree is examined in a BFS way, contrary to a depth-first-search (DFS). In case of DFS, it would mean to construct and evaluate each complete architecture, because in DFS, the tree is examined to the leaf node before another path is considered. The proposed DSE approach is a parallel constructive search technique. It is parallel in the sense that several architecture points (decisions) are evaluated and selected in parallel for each design issue, when observing the design constraints and optimization objectives.

In general, given a branching factor b and a tree with depth d, the asymptotic time and space complexity is of the order $O(b^d)$ for the BFS approach. Equivalently, $O(|V| + |E|)$, where $|V|$ represents the cardinality of the set of nodes (vertices) and $|E|$ represents the cardinality of the set of edges in the search tree (graph). However, unlike the BFS that exhaustively enumerates all the nodes, in the proposed approach all the nodes at any level (depth) of the tree are not searched and evaluated. The proposed DSE algorithm selectively (rather than

exhaustively) search the space for the most promising architectural decisions (non-dominated partial architectures). These decisions are based on the hard constraints, dominance and implication relations, what results in a well-informed search. This very much reduces the sub-sets of the actually constructed (partial) architectures, and in consequence, the search time of the proposed DSE algorithm. In the following, it is described how the DSE algorithm on one hand ensures the optimality of the solutions, while on the other hand, much reduces the complexity of the search compared to the exhaustive approach for which the complexity is of the order $O(m.n.q)$.

The actual DSE algorithm organizes the design decision (state) space for the design decision-making at two abstraction levels. The top-level breaks down the decision space into a number of distinct issues in a tree like structure, as shown in Figure 3.5. The lower-level represents the actions (operators) that are performed at different stages (corresponds to particular nodes in the search tree at a particular level) during the search for the solutions. Moreover, each stage can itself represent a complete search problem implemented with a feasible search strategy relevant to the issue. It may also be just a state (node) in which some action (operators) can be used to evaluate (directly or indirectly) a node (state) towards the goal state (required solution). In the proposed DSE approach, the actions (operators) represent various evaluation functions that check each decision at each stage of the design. For instance, an evaluator function during micro-architecture decision stage is to check whether a given micro-architecture satisfy the clock speed constraint or not.

Since the throughput and clock speed are the hard constraints, and their satisfaction mainly depends on the processing parallelism, and in turn the required processing parallelism decides to a high degree the memory and communication architecture. Therefore, the architecture exploration starts with the decision of processing parallelism, and specifically, the selection of the micro-architecture (P_{mic}) parallelism for each of the task. The other reasons for initializing the search with the micro-architecture decisions are the the fewest legal values for P_{mic} compared to P_{mac}. This decision variable ordering is called "minimum-remaining-values (MRV) heuristic". It also helps in pruning the search tree by eliminating a large number of branches at the second step (usually more macro-architectures than micro-architectures). Each of the micro-architectures for the same task has different execution time expressed in clock cycles, and different clock frequency, area and power.

As the clock speed is a hard constraint, only the micro-architectures (P_{mic}) that satisfy the clock speed constraint are further considered, and those that do not satisfy the clock speed constraint are pruned away (see Steps 1-3 of Algorithm 1). This decision is possible to be made, due to the availability of the generic micro-architecture models for the processors and the related parameter estimates (clock frequency, execution time in clock cycles, area and power consumption). Lack of such quality models in the case of the traditional top-down design approach results in unpredictable and often lower accelerator quality, as well as,

expensive design re-iterations, which negatively impacts the design efficiency in terms of design-time and cost.

For a given computation task, the search of the micro-architectures is performed in the order of increasing micro-architecture parallelism. The processor's micro-architecture and frequency are related and satisfy the following relation,

$$x^{(1)}(i) < F_{cstr} \Longrightarrow x^{(2)}(i') < F_{cstr}, subject\ \ to\ (i \neq i') \wedge (i' > i) \tag{3.4}$$

That is, if an alternative $x^{(1)}$ of micro-architecture parallelism i is non-feasible (considering the clock speed constraint), then all the alternatives $x^{(2)}$ of higher micro-parallelism ($i' > i$) are also non-feasible. The above implication relation is valid for various micro-architectures under the assumption of no pipelining of micro-architectures. This rule allows the DSE framework to implicitly inference the non-feasibility of solutions and eliminate all the non-feasible solutions after explicitly finding only the first non-feasible solution. Elimination of a particular micro-architecture (P_{mic}) drastically reduces the search space, as it also eliminates the related macro-architectures (P_{mac}) that otherwise might be in the set of candidate architectures (S_{Arch}). The selected micro-architectures (P_{mic}) that satisfy the clock speed constraints are preserved and this information about the micro-architectures (P_{mic}) is added to the list of newly created set of candidate architectures (S_{Arch}) (Step 2). This type of exploration is performed for each different type of computing task, if an application involves different kinds of tasks that require different processors for their execution (Step 3). This way due to the clock speed constraint (F_{cstr}) consideration a large sub-set of micro- and macro-architecture combinations is eliminated from further consideration. A limited set of candidate architectures (S_{Arch}) is then passed to the next stage of exploration.

In the second step of first stage, the exploration of the required number of processors of each kind, i.e. the macro-architecture exploration is performed for each of the selected micro-architectures (P_{mic}). As the throughput is the other hard constraint (T_{cstr}) that must be satisfied, the macro-architectures (P_{mac}) are constructed that minimally satisfy the throughput constraint (T_{cstr}) with each of the micro-architectures (P_{mic}) selected in the previous step. These macro-architectures correspond to architectures involving minimum macro-parallelism (minimum resources) necessary to satisfy the throughput constraints when exploiting a given micro-architecture. This way, the hard constraint on the throughput usually eliminates a large number of non-feasible micro-/macro-architecture combinations from further consideration. However, for each of the micro- and macro-architecture combinations that minimally satisfy the T_{cstr}, all other solutions with higher macro-parallelism for each of the micro-architectures considered are also feasible solutions (a large design space due to large number of possible macro-architectures). To find the set of non-dominated macro-architectures in this large space in an acceptably short time, the dominance relation is used. The dominance relation ensures that all the dominated solutions (macro-architectures) are implicitly considered and eliminated from the set of candidate architectures

without the actual construction. The dominance consideration is based on the following two observations:

Algorithm 1 : Design space exploration (DSE) and synthesis algorithm for multi-processor accelerators

Input $\rightarrow$ *Behavioral (Task) specification, Throughput* (T_{cstr}), *Frequency* (F_{cstr})
Input $\rightarrow$ *set Aspiration Point (AP) for Area* (A_{obj}) *and Power* (P_{obj})
Output $\leftarrow$ $S_{Arch} = \{x_1, ..., x_n\}$ // set of architectures {1....n} that minimally satisfy the hard constraints of clock speed and throughput, and optimize the tradeoffs between area and power, where x = {processors, memory, communication}
Initialize $S_{Arch} = \emptyset$
Step 1: for each micro-architecture i $\in P_{mic}$ for a given task, check if i satisfy F_{cstr}, if yes go to step 2, else go to step 3 {implication relation holds, no need to check micro-architectures with higher parallelism} (search is performed in increasing P_{mic} order)
Step 2: create a new architecture instance x and update x with micro-architecture decision i and add to S_{Arch}=$\{S \cup x\}$, go to step 1
Step 3: repeat steps 1 and 2 for all other tasks
Step 4: for each $x \in S_{Arch}$ with micro-architecture i selected in step 2
Step 5: search for the macro-architecture $j \in P_{mac}$ that minimally satisfy T_{cstr}, if yes go to step 6, else go to step 7 {Note: T_{cstr} is computed based on the actual scheduling and mapping or analytically if trivial}
Step 6: update x with macro-architecture decision j, go to step 4 {[not to step 5] as further increase in P_{mac} with the same P_{mic} result in a feasible but not efficient solution both locally and globally}
Step 7: delete architecture instance x from S_{Arch}=$\{S \setminus x\}$, as it does not satisfy T_{cstr} with maximum P_{mac}, go to step 4 if all candidate x are not explored for P_{mac}, else go to step 8
Step 8: for each $x \in S_{Arch}$ compute the required memory and data transfer bandwidth, go to step 9 {Note: aggregate bandwidth = $P_{mic} \times P_{mac} \times b$: to match the processing bandwidth with the data transfer bandwidth}
Step 9: realize memory bandwidth using various memories architectures
Step 10: realize data transfer bandwidth using various communication architectures
Step 11: Among all the possible memory and communication architectures for the given micro-/macro-architecture combination that satisfies T_{cstr}, choose the one that satisfy T_{cstr} with minimum area or minimum power and go to step 12, if none of them satisfies the T_{cstr} then go to step 13
Step 12: update x with the selected memory and communication architecture decision, if all candidate x are not explored, go to step 8, else go to 14
Step 13: delete architecture instance x from S_{Arch}=$\{S \setminus x\}$, if all x are not explored, go to step 8, else go to 14
Step 14: for each $x \in S_{Arch}$, synthesize the architecture based on the decision vector values of processor's micro-/macro-architecture, memory and communication architecture

1. After finding a macro-architecture for a particular micro-architecture that minimally satisfies the throughput constraint, any further increase in the macro-architecture parallelism (number of processors) for the same micro-architecture will result in a feasible, but a non-efficient dominated solution, because both the area and power consumption increase due to the increase in the macro-architecture parallelism (more processors) for the same micro-architecture. It should be clear that due to the hard constraint on frequency

any tradeoff between power consumption and area through increasing parallelism and lowering frequency is not possible.

Mathematically, this dominance relation under the hard constraint of clock speed is represented as follows,

$$x^{(1)}(i,j) \succeq x^{(2)}(i',j'), \textit{subject to } (i = i') \wedge (j' > j) \tag{3.5}$$

where i, i' represents the micro-parallelism and j, j' the macro-parallelism of the alternatives $x^{(1)}$ and $x^{(2)}$, respectively. This way, we have to explicitly consider only the guaranteed non-dominated solutions being a particular micro-architecture combination with a macro-architecture that minimally satisfies T_{cstr}.

2. It is guaranteed that the macro-architecture (number of processors) for each micro-architecture which minimally satisfies the T_{cstr} is a non-dominated solution based on the dominance relation as discussed above. However, a macro-architecture for a particular micro-architecture that minimally satisfies T_{cstr} is non-dominated only locally (in relation to the given micro-architecture) and not globally (i.e. across the different micro- and macro-architecture combinations). In order to ensure that it is also non-dominated globally, the following observation is exploited about different micro- and macro-architecture combinations that minimally satisfy the T_{cstr} and F_{cstr}. A micro- and macro-architecture combination that minimally satisfies the T_{cstr} and F_{cstr} with a higher micro-architecture parallelism results in a lower power consumption than the one with lower micro-architecture parallelism, and the other way round for area. That is, the power cannot be improved without causing a degradation in area and the other way round for area (condition for Pareto-optimality). This observation (relation) ensures that all the different micro- and macro-architectures that minimally satisfy the T_{cstr} and F_{cstr} are also globally non-dominated solutions. This dominance relation for alternatives that are locally non-dominated under the condition that the relation in equation 3.4 is satisfied can be represented using the following equations:

$$\nexists x^{(2)}(i',j') \succeq x^{(1)}(i,j), \textit{subject to } (i' > i) \wedge (j' <= j) \tag{3.6}$$

$$\nexists x^{(2)}(i',j') \succeq x^{(1)}(i,j), \textit{subject to } (i' < i) \wedge (j' >= j) \tag{3.7}$$

That is, there exist no alternative $x^{(2)}$ that dominates $x^{(1)}$. $x^{(1)}$ represents a non-dominated alternative, and i, i' represents the micro-parallelism and j, j' the macro-parallelism of the alternatives $x^{(1)}$ and $x^{(2)}$, respectively.

This way, the hard constraint of throughput and clock speed and the dominance relations as discussed above for the micro-/macro-architecture combinations guarantee the Pareto-optimality of the micro-/macro-architecture combinations, and

at the same time enable a quick implicitly exhaustive search of the solution space, while explicitly considering only a small sub-set of all possible solutions.

The throughput estimation for a particular micro-/macro-architecture combination is performed either based on the scheduling of tasks on the processors, and this way computing the schedule length in clock cycles (CC), or analytically in case the scheduling problem is trivial. This way, the delivered throughput (T_{del}) measured in Mbps, which is one of the design quality metrics, can be determined using the following expression,

$$T_{del} = \beta.(F_{cstr}/CC) \tag{3.8}$$

where β is a constant factor that depends on the application under consideration. For instance, in case of the LDPC decoding, β represents the ratio of the frame size of a given LDPC code and the required number of decoding iterations. There are two possible cases during the macro-architecture (P_{mac}) exploration for a particular throughput constraint (T_{cstr}) when considered in combination with each of the micro-architectures (P_{mic}) with different parallelism (selected in the previous step). Either the throughput is achieved with some macro-architecture parallelism (P_{mac}) and micro-architecture parallelism P_{mic} (Steps 5-6). If this is the case, increasing the macro-architecture parallelism after finding the first feasible solution would result in a dominated solution (as discussed above). Therefore, the macro-architectures beyond this point with that particular micro-architecture (P_{mic}) are not further considered. This way, all the other solutions beyond this macro-architecture parallelism are not constructed, but observe that they were anyway considered implicitly. The second case is that it turns out to be impossible to realize the throughput (T_{cstr}) for a particular micro-architecture P_{mic} even exploiting the maximum (full) macro-architecture level parallelism. In this case, the micro-architecture (P_{mic}) and all the related macro-architectures (P_{mac}) will be pruned away from the set of candidate architectures (S_{Arch}) (Step 7). This way, a limited set of non-dominated solutions (Pareto-optimal solutions S_{Arch}), i.e. combinations of processor micro- and macro-architectures will be constructed.

In the second stage, the memory and communication architectures are decided for each of the earlier constructed candidate partial architectures (S_{Arch}) representing particular micro- and macro-architecture combinations (P_{mic}, P_{mac}) (Step 8). Each of the processor's micro- and macro-architecture combination required a compatible memory and communication architecture. For a memory architecture M_{arch} to be compatible with the processors, the number of input and output ports of the memories must be the same as the number of input and output ports of processors. Therefore, the storage and data transfer bandwidth per clock cycle must match the processing bandwidth, i.e.

$$bandwidth/cc = P_{mic}.P_{mac}.b \tag{3.9}$$

where b represents the bit width of data.

To ensure the storage and data transfer bandwidth required by processors on a low cost and satisfactory delays, different memory and communication architectures are considered during the DSE. The memory architectures include the simple multi-port vectorized memories and the complex multiple single/dual-port vectorized or multi-bank memories (Step 9). The communication architectures include the simple homogeneous flat architectures and the complex heterogeneous hierarchical partitioned communication architectures (Step 10). Moreover, the memories can be realized as SRAMs or register-based memories and the communication architectures using single-stage and multi-stage switches and shifters.

Although the top-level generic architecture template includes information about different types of memories required for different tasks and the data flows between the different tasks. The required storage and data transfer bandwidth depend on the micro- and macro-architecture processing parallelism required to satisfy the throughput constraint (T_{cstr}). Therefore, during the DSE different memory and communication architectures have to be considered. It should be noted that the actual data mapping to memories and the communication between the memories and processors are resolved statically during the mapping of tasks to processors and the corresponding data to memories. The physical RD/WR addresses for various memories and the routing addresses for various switches and shifters of the communication network are computed off-line. The off-line computed addresses are then used at run-time to control the RD/WR of data from/to memories and the communication (routing) among various processors and memories. More information on the memory and communication architectures can be found in Chapter 6 of this thesis.

The search, evaluation and selection of the memory and communication architectures is based on the area and power consumption of each particular memory and communication architecture and their impact on the throughput and the clock speed constraints (Step 11). The selection is made according to the following criteria. For a particular micro- and macro-architecture combination, only those memory and communication architectures are considered that satisfy the throughput and clock speed constraints. Therefore, the T_{del} and F_{del} is recomputed for each of the candidate architecture in S_{Arch} while taking into account the memory and communication structures delays. From among these architectures, only those are further considered, which cause the lowest increase in the area or power consumption. Those complete architectures $Arch_{inst}$ that satisfy the throughput and clock constraints are selected to the final candidate set of architectures (S_{Arch}) (Step 12), otherwise they are pruned away from the set of candidate architectures (Step 13).

This way, the DSE is accomplished for a particular set of requirements. Since the number of Pareto-optimal candidate processors's micro-/macro-architecture combinations obtained in the previous step is usually small, performing the memory and communication architectures design step exhaustively for each micro-/macro-architecture combination does not substantially impact the overall DSE efficiency. The experiments demonstrated that for a low processing parallelism,

the flat communication architecture performs the best, while for the moderate to high parallelism levels the hierarchical partitioned architecture performs the best both from the point of view of performance and area (see Chapter 6 for details). These observations are used to speed-up the memory and communication architectures exploration, when exploiting the information on the processing parallelism.

Above the basic operation of the method and DSE framework is explained, which is based on the exploitation of hard constraints and dominance relations. However, with the proposed method the tradeoffs among the different optimization objectives can also be exploited by application of the tradeoff resolution methods to the partial architectures S_{Arch} obtained in a given step. This can be done, for instance, using the distance-based multi-objective decision-making method called the ideal point or aspiration point method, i.e. by finding the distance between the ideal/aspired (solution) value of a criterion (power or area) and the actually achieved value of that criterion (power or area) in an alternative [180–183].

The tradeoff resolution can be for instance needed if for a given application no particular clock frequency is required. In this case, the architecture parallelism (area) can be traded against the power consumption. Lower power consumption can be obtained by increasing the processing parallelism above the required to minimally satisfy the throughput constraint and decreasing the clock frequency. In this case, the simple usage of the dominance relation to limit the set of the explicitly considered partial solutions, as described above is not possible, but just the tradeoff resolution helps to guarantee the high solution quality, while at the same time limiting the search space.

For the case, when the clock speed is not a hard constraint, all the processor's micro-architectures have to be considered (selected) during the micro-architecture exploration for each of the tasks. The following two assumptions are made in relation to this case. First, each of the processors operates at its maximal clock speed (F_{max}) determined by the processor's critical path delay. This is necessary to compute the corresponding throughput. Second, the designer/customer has to explicitly specify his aspirations regarding the required quality i.e. specify the area and power values as optimization and tradeoff goals (the solution is aimed to be as close as possible to the ideal/aspiration point of all the optimization objectives in case it is impossible to the solution with over parameters values in the actual ideal point). As the throughput is the only hard constraint that must be satisfied, the macro-architecture that minimally satisfies T_{cstr} with each of the micro-architecture combinations at their corresponding maximum clock speed are selected for further consideration. This way, a set of Pareto-optimal solutions is actually considered like in the previous basic case with a hard constraint on clock frequency, but only under the maximum clock speed assumption.

There are again two possible cases: either the macro-architecture with a particular micro-architecture satisfies T_{cstr} or do not satisfy. However, in this case, for each micro-/macro-architecture combination that satisfies T_{cstr} power can be

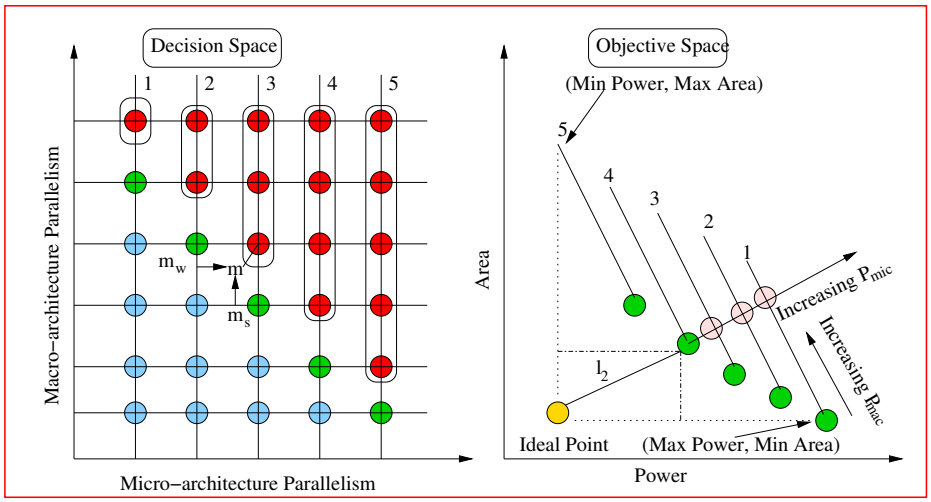

Figure 3.6: Ideal point method for tradeoff resolution among area and power

traded against area by further increasing the macro-architecture parallelism for each particular micro-architecture parallelism and reducing the clock speed or the other way round. This would result in a very large search space due to a large number of alternatives in the former (increasing macro-architecture parallelism) compared to the latter case (increasing micro-architecture parallelism). Therefore, to reduce the search space and to obtain the required power and area tradeoffs close to the ideal/aspired values, the search in the DSE algorithm is based on the following experimental observations.

After finding a micro-/macro-architecture combination that minimally satisfies the T_{cstr}, increasing the macro-architecture parallelism beyond that point to trade power against area will result in higher area increase and lower power reduction than by increasing the micro-architecture parallelism only (this is confirmed by experiments performed). For instance, consider the point m in Figure 3.6 (the decision space), approaching this point in the direction of increasing macro-architecture parallelism, i.e. from point m_s, cause more area and lower reduction in power than when this point is approached from point m_w just only by increasing the micro-architecture parallelism (assuming that both points are at Pareto-optimal front based on the maximal clock speed). However, the problem is actually to which of the micro-/macro-architecture combinations to select to direct the search for the required tradeoffs among power and area and at the same time reduce the search space. For this purpose, the ideal/aspiration point distance-based tradeoff resolution method is used [180].

The distance $l_p(x)$ of any point $z(x)$ in the objective space from the ideal point

z° can be calculated as,

$$l_p(x) = min\left(\sum_{i=1}^{k} w_i[z_i(x) - z_i^\circ]^p\right)^{1/p}, 1 \leq p \leq \infty, subject\,to\, x \in S \tag{3.10}$$

where k represents the number of criterion, w_i is their weights, z_i° the ideal point value of criterion i and $x \in S$ represents an alternative. The p can take any value between 1 and ∞. If the value of p is set at 1, the rectangular distance is computed. If $p = 2$, the straight-line (Euclidean distance) is computed. When $p = \infty$, the problem reduces to the problem of minimizing the largest deviation $[z_i(x) - z_i^\circ]$.

$$l_p(x) = min \max_{i\in[1,k]}\left(w_i[z_i(x) - z_i^\circ]\right), subject\,to\, x \in S \tag{3.11}$$

In our problem formulation, the value of $p = \{1, 2\}$ for the reason of two objectives (area and power). First, the distance is computed of each alternative (solution) selected based on the maximal frequency from the ideal solution z°. This is possible in the proposed architecture synthesis method due to the availability of parametric quality models of the architecture elements (processors). Among those alternatives, the one with the minimum distance $min(l_p(x))$ from the ideal point is chosen. Moreover, the search is carried out in the direction of increasing the micro-architecture parallelism, as this deliver the best tradeoff between the area and power consumption, as discussed earlier. This way, the optimality of the final architecture(s) in respect to the required quality is guaranteed in relation to (with precision of) the architecture models used and their characterization parameters.

There are also other formulations possible. For instance, if the objective is the minimization of power, one have to choose the fully-parallel micro-architecture and increasing the macro-architecture parallelism until the minimum power is obtained but at the cost of maximum area. In other case, if the objective is to maximize power reduction with minimum increase in area, then one choose the fully-serial micro-architecture and increase the macro-architecture parallelism, as shown in Figure 3.6.

After the final selection of one (or more) of the constructed most promising architectures, all the decisions regarding the processor micro- and macro-architectures for each processor type and the memory and communications architectures for all the selected architectures S_{Arch} are passed to an automatic architecture instantiation engine. The architecture instantiation engine generates the corresponding RTL-level structure for each of selected architecture S_{Arch}. The RTL-level accelerator structures can then be used for further analysis, rapid prototyping, refinement and final selection of the architecture.

3.5 Advantages of the Proposed Design Method

The major advantages of the proposed multi-processor accelerator design method include the following: higher quality-of-results (QoR) and much higher development efficiency comparing to design without using the method.

The high QoR is ensured by the DSE that efficiently searches the large space of possible accelerator structures to construct the accelerator, which maximizes the actually required quality, represented by the accelerator's behavioral, structural (generic architecture templates) and parametric (performance, power consumption and area) requirements. In its search, the DSE method constructs, analyzes and evaluates the architectures, when using the actual floorplanning data of the generic architecture templates, as well as, the actual values of the generic modules physical parameters (throughput, frequency, area, power) obtained through their physical characterization. Usage of the actual parameter values, instead of some rough estimated values, as could be computed from some abstract models, makes the architecture analysis and estimation of the DSE method very precise (the estimated values are very close to the actual values). This makes possible a very precise decision-making during the DSE, and in consequence, construction and selection of an architecture that maximizes or almost maximizes the actually required quality.

The second major advantage is the very low designer effort and very short time required for the construction of a particular accelerator for a particular application and its requirements. For instance, using the method, one designer can construct a complete high-quality accelerator for LDPC decoding in a few hours, while the effort required for a single LDPC decoder design without using such a method and DSE tool represents a work of a small design team in approximately six months. This way the design method has a development efficiency of several orders of magnitude higher in comparison to the construction of a single accelerator for a particular application case in a traditional way.

This huge design efficiency and result quality increase is on the cost of a prior development of the generic architecture template, its generic modules for processing, memory and communication, and the DSE framework. However, this only requires a comparable amount of effort and time as required for developing a single accelerator for such an application (as discussed above for the LDPC decoding). The architecture templates were designed and implemented by one designer within 5 months (including the application analysis and building an automatic template characterization engine), and required another 2 months for further template optimizations. The DSE framework was designed and implemented by one designer within 5 months. From this, it follows that a comparable amount of effort and time (please note individual versus small team) is needed for building an architecture synthesis framework and its generic templates, as for a single accelerator design. However, the initial effort of building the templates and DSE framework is immediately compensated by the high development efficiency and

quality-of-results (QoR) delivered by the method when constructing accelerators for different application cases.

It is worth to note that building such an architecture synthesis framework in a short time requires a solid knowledge and reasonable experience from multiple domains, including: application domain, architecture design, physical design and electronic design automation.

3.6 Design Decision Space and Search Complexity

The size of the design decision space in the proposed multi-processor accelerator architecture synthesis process can be mathematically expressed as:

$$\prod_{i}^{N}(m_i.n_i).\prod_{q}^{Q}q_{j,k} \tag{3.12}$$

where m_i and n_i represent all the possible micro- and macro-architecture alternatives for a given data-parallel task i, respectively. N represents the total number of different types of tasks for a given application. Since for each micro-architecture m of a given task i, there are n possible macro-architectures, the number of possible micro- and macro-architecture combinations will be the product of m_i and n_i. This is true for all the N tasks of a given application. Thus, the size of the search space rapidly increases with increase in the number of tasks N, as represented by the expression 3.12. For instance, for the rate-7/8 672-bit IEEE 802.15.3c LDPC code, 32 micro-architectures and 84 macro-architectures are possible for check node computation task. Similarly, 4 micro-architectures and 672 macro-architectures are possible for the variable node computation task. This in total results in a large number of possible micro- and macro-architecture combinations for both tasks 7225344 ($= 32 \times 84 \times 4 \times 672$).

Each of the processor's micro- and macro-architecture combinations requires a compatible memory and communication architecture $q_{j,k}$. In $q_{j,k}$, the subscript j represents a memory architecture (M_{arch}) and k a communication architecture (C_{arch}). Q represents the total number of different types of memory and communication architectures.

Several compatible memory architectures are considered in the proposed DSE approach. For a memory architecture M_{arch} to be compatible with the processors, the number of input and output ports of the memories must be the same as the number of input and output ports of processors. The M_{arch} involve several different memories as defined in the top-level generic architecture template for a particular application. The memory bandwidth for each kind of memory must match the bandwidth required for the processors. Two types of memory organizations are considered in the proposed approach: multi-port memories and vectorized single/dual port memories. Moreover, the proposed approach considers two types of memory implementations, Flip-Flop-based memories and SRAM

memories. Thus, the search space for the memory architectures involve two kinds of memory organizations (multi-port or vectorized single/dual port) and two possible implementations for each kind of memory. This has to be decided for each of the memory involved in the top-level generic architecture template. This is sufficient only to satisfy the bandwidth compatibility relation of the various memories with their corresponding processors.

Moreover, another important aspect of the M_{arch} design is to distribute, assign and map the data in the partitioned single/dual port vectorized memories. It should be noted that all such possible memory architectures have to be considered for each of the processor's micro- and macro-architecture combination (see Chapter 6 for details).

Similarly, the communication architectures C_{arch} realize the different communication segments among the various memories and processors as defined in the top-level generic architecture template. Three types of communication architectures C_{arch} are considered to realize the required data transfer bandwidth between different memories and processors. They are flat, hierarchical and hierarchical-partitioned communication networks. Each of the communication network (for each segment) can be realized using various kind of switches and shifters. Thus, in total three types of communication architectures (C_{arch}) with various combinations of switches and shifters are searched for each of the communication segment in the proposed DSE approach. It should be noted that the search for the communication network have to be performed for each of the processors' micro- and macro-architecture combination. C_{arch} must be compatible from one side to the memories and on the other side to the processors (see Chapter 6 for details).

To keep the search complexity lower and at the same time guarantee that non-efficient solution will be pruned away, the search approach in the proposed DSE algorithm is based on the dominance, implication and compatibility relations defined on the parameters of the pre-designed architecture models for various partial solutions at each architecture construction step. The physical parameter set consist of $\{F_{max}, CC, A, PW@F_{max}\}$ for a given computation task, where F_{max} represents the maximum clock speed of the processor, CC represents the number of clock cycles required for a given computations task, A represents the (resources) area, PW represents the total power consumption at F_{max}. The design constraints (T_{cstr} and F_{cstr}) are successively used at each design step together with the dominance, implication and compatibility relations to limit not only the search space, but also to prune away the dominated solutions.

In the following, it is discussed how the proposed DSE algorithm reduces the search complexity by explicitly evaluating a limited number of solutions rather than exhaustively enumerating all possible solutions at each exploration step.

The DSE algorithm first decides on the micro-architecture (P_{mic}) parallelism for each task based on the F_{cstr}. In the worst case, all the micro-architectures for all the tasks may satisfy the F_{cstr}, while in the best case, only a single one or a small number of micro-architectures. It would correspond to the evaluation of all m possible micro-architectures in the worst case, while only a single one

in the best case at *level-1* of the search tree, as shown in Figure 3.5. Rather than evaluating all the micro-architectures exhaustively, the search for the micro-architectures is performed based on the implication relation that limit the number of micro-architectures, which are evaluated explicitly. This way, the DSE algorithm searches a limited set of micro-architectures m_s among all the possible micro-architectures m_t. The proposed DSE approach not only reduces the search complexity of the current stage but also reduces the complexity of the successive stages. For instance, if a micro-architecture that violates F_{cstr}, it means that all the P_{mac} combinations for this P_{mic} would also violate the F_{cstr}. Therefore, all the branches of the tree which violate the F_{cstr} at *level-1* are not considered for search during the P_{mac} step. Thus, the design decision space reduces for the macro-architecture decision stage from $m_t.n_t$ to $m_s.n_t$. Equivalently, $(m_t - m_s).n_t$ solutions are pruned away prior to the macro-architecture exploration stage.

In the macro-architecture selection stage, the reduced decision space of size $m_s.n_t$ from the prior step is searched. This corresponds to all the n branches of the tree at *level-2* for each of the branch selected at *level-1*, as shown in Figure 3.5. The search in the proposed DSE algorithm is limited by the throughput constraint (T_{cstr}) together with the dominance relation, as discussed earlier in Section 3.4 of this chapter. In the worst case, all the n_t macro-architectures (P_{mac}) for a particular micro-architecture (P_{mic}) among the m_s would not satisfy the T_{cstr}, while in the best case a single one or a small number of macro-architectures. This would correspond to the evaluation of all n_t possible macro-architectures in the worst case, while a single one in the best case at *level-2* of the search tree, for each of the P_{mic} selected in the previous step, as shown in Figure 3.5. As the dominance relation eliminates all the dominated solutions, only $m_s.n_s$ are explicitly evaluated compared to the exhaustive approach in which all the $m_s.n_t$ micro- and macro-architectures combination have to be evaluated. Here, n_s represents the set of macro-architectures that are evaluated explicitly. This way, it reduces the search complexity of the macro-architecture exploration stage. Equivalently, $m_s.(n_t - n_s)$ are eliminated without the explicit evaluation based on the dominance relation. It also reduces the search complexity of the next stage, i.e. the memory and communication architecture exploration stage, by eliminating a large number of micro-/macro-architectures combinations that otherwise have to be considered.

For each of the selected micro-/macro-architecture combination in the reduced decision space of $m_s.n_s$, the corresponding compatible memory and communication architectures are explored. Since at this step the number of micro- and macro-architectures combinations to be further considered is usually very small, employing this step even exhaustively for each of the micro-/macro-architecture combination does not impact much the overall search efficiency. Nevertheless, some strategies are applied in the proposed DSE approach that reduce the complexity of the search during the memory and communication architecture exploration stage. For instance, for lower micro-/macro-architecture combinations, the fully-flat communication network performs better than the others. This way, the

search complexity can be further reduced during the memory and communication architecture exploration stage.

3.7 Case Study: Hardware Multi-processor LDPC Decoder Design

To analyze and evaluate the accelerator design methodology discussed above, it is applied to the design of demanding hardware accelerators for LDPC code decoders for some of the newest demanding communication system standards. In this process, generic architecture templates are used similar to the one shown in Figure 3.1. The aim of the design process is to find the best possible application-specific accelerator architecture for a particular LDPC application, through promising instantiations of the generic accelerator architecture templates, behavior mapping and scheduling on those templates, and analysis of the constructed alternative architectures. The required quality of the accelerator is characterized by the LDPC code required to be decoded and its associated parity check matrix (PCM) representing the accelerator's required behavior, as well as, by a set of parametric requirements related to the error correcting performance, throughput, and accelerator area. The design parameters that can be influenced and decided during the DSE include the following:

- the algorithm to be used for decoding;
- the intrinsic and extrinsic messages bit-precision (when accounting for the error-correcting performance needed in the form of bit error rate);
- the maximum number of iterations and the stopping criteria (dependent on the type of algorithm used);
- the accelerator operating frequency;
- the micro-architecture of the elementary processing units (specifically, the parallelism level of the processing units);
- the macro-architecture (specifically, the number of processing units to be used and assignment of nodes to processing units);
- the number, size, structure and organization of memory modules;
- the kind and organization of interconnect resources and switching network (e.g. logarithmic or barrel shifter, Benes network).

Since the research reported in this thesis is devoted to multi-processor architectures and their design methods, the first three parameters related to the decoding algorithms were not so interesting as the other ones, and therefore they were set to some constant values. In the case study and related design space exploration,

the focused was on the last five parameters directly related to the accelerator multi-processor architectures.

3.8 Conclusions

In this chapter, a novel quality-driven model-based multi-processor accelerator design methodology was presented that addresses the issues and satisfies the requirements of multi-processor accelerators design for highly-demanding applications. The methodology extension to the design of re-configurable multi-processor accelerators was also discussed. Afterwards, a novel multi-objective and multi-dimensional design space exploration (DSE) framework for this methodology was described that resolves the complex processor, memory and communication issues, and their mutual tradeoffs effectively and efficiently. The numerous important tradeoffs related to various design dimensions addressed by the proposed DSE framework were described in detail. In particular, it was discussed how the proposed DSE framework resolves the complex problem of the combined processor micro-/macro-architecture design, as well as, the corresponding memory and communication architecture design for massively parallel multi-processor accelerators. The DSE algorithm and its different stages were discussed in detail related to each of the design issue. It was described how the DSE algorithm explores and constructs various multi-processor accelerators of the required quality expressed by the design quality metrics for a particular application. Moreover, it was shown how the proposed DSE algorithm, at on one hand, ensures the optimality of the solutions, while on the other hand, much reduces the complexity of the search. The advantages of the proposed design methodology in relation to the existing design approaches were also discussed. At the end, the application of the multi-processor accelerator design methodology to the design of decoders for the LDPC decoding applications was briefly discussed.

CHAPTER 4

Implementation of the Design Methodology for its Application to LDPC Decoding

Application of the quality-driven template-based hardware multi-processor design methodology proposed and discussed in the previous chapter to the semi-automatic design space exploration (DSE) and synthesis of accelerators for a given application class starts with adaptation of the methodology to this class, which mainly involves development and characterization of architecture templates for this application class. This chapter focuses on the application of our proposed design methodology to the design of decoders for LDPC applications. LDPC decoding algorithms and its derivatives are introduced first. Then, the generic architecture template for LDPC decoders, template features, various template modules, and the related module library are described. The library includes various kinds of generic application-class specific processors, as well as various kinds of generic memories and interconnect resources. The design, modeling and characterization are thoroughly explained for various kinds of processors for required computations, various kinds of memories, various kinds of switches for communication between the processors and memories, and other related components, as well as, of the top architecture module representing the accelerator multi-processor architecture, which together represent the generic architecture template. Also, various possible complex tradeoffs among different architecture elements are explained. The DSE for the LDPC decoding is discussed in detail.

The rest of the chapter is organized as follows. Section 4.1 is about the LDPC decoder design considerations. Section 4.2 introduces LDPC decoding algorithms

and its derivatives. Section 4.3 describes the LDPC decoder architecture template and it main features. Section 4.4 describes the design of the generic component library including generic application-class specific processors, memories and interconnects, as well as, their characterization. Section 4.5 describes the DSE for the LDPC decoding. Section 4.6 describes the architecture template instantiation and rapid prototyping. Section 4.7 discusses how the method ensures the construction of correct architectures. Finally, Section 4.8 concludes this chapter.

4.1 LDPC Decoder Design Considerations

Design of hardware LDPC decoders is complex and difficult for several reasons. First of all, the LDPC decoder design space is a complex multi-dimensional space. Several LDPC codes have been developed for various communication system standards with several code rates (R) and code lengths (L) in different application scenarios. From the algorithmic perspective, several iterative decoding algorithms are available with varying error-correcting performances and complexities. Finally, many decoder architectures and their implementations are possible. Each aspect has a different impact on the accelerator design and generates various tradeoffs. For instance, more effective error-correcting algorithms require more complex computation processes, which results in a lower performance or in an increased accelerator hardware complexity. A need to account for various LDPC standards with several PCMs, code rates or code lengths in one system, demands for the accelerator to be (re-)configurable. High-throughput demands and other stringent requirements decide to a high-degree the accelerator architecture and implementation strategies. Having particular application requirements, all the above mentioned design aspects have to be taken into account to design an effective and efficient hardware accelerator for LDPC decoding.

The architecture spectrum of LDPC decoders spans between the two extremes of a fully-serial and a fully-parallel realizations with in between a large number of partially-parallel choices. The advantage of the fully-parallel implementations is their high throughput, but they require a lot of hardware, consume much energy, and are extremely inflexible regarding adaptation to various code lengths and code rates. The problems of interconnect, memory, and computational nodes complexity reduction for the fully-parallel decoders and high-throughput has been the focus of several researchers [44–51]. The fully-serial approach results in a low throughput, but at a low area and complexity, and with a high flexibility [43]. In result, the partially-parallel approach seems to be the most promising for the purpose of an adaptable accelerator design, and most popular, mainly for the reasons of a relatively low adaptability cost of the decoder to various codes lengths, code rates, moderate area and complexity, and tradeoff possibility between throughput, area and other parameters [52–83]. Although the research reported in this thesis accounts for the full spectrum of accelerator architectures, the focus was on the partially-parallel architectures. Further information on the

above proposed architectures and their architectural features can be found in Section 2.6 of Chapter 2 of this thesis.

4.2 LDPC Decoding Algorithms

LDPC codes can be decoded efficiently using the sum-product (SP) algorithm [39], also known as message-passing (MP) or belief propagation (BP) algorithm. Inputs of the algorithm are the so-called intrinsic log-likelihood ratios (LLRs), I_n, of the received symbols based on the channel observations, defined as [39]:

$$I_n = LLR(y_i) = log\Big(\frac{P(x_i = 1|y_i}{P(x_i = 0|y_i}\Big) \tag{4.1}$$

where, x_i stands for transmitted symbol, y_i for received symbol. Based on the intrinsic LLR values, the algorithm iteratively updates the extrinsic LLR messages among the variable nodes (VNs) and check nodes (CNs) along the edges of the Tanner graph for a certain number of iterations, as shown in Figure 4.1. Let C_{mn} denote a check node LLR message sent from the check node m to the variable node n. Let V_{mn} denote a variable node LLR sent from the variable node n to check node m.

The algorithm iteratively updates the VNs and CNs messages in the following steps.

1. *Initialization*: Initialize each variable node LLR, Q_n, and the relevant V_{mn} to the intrinsic LLR, I_n, of the corresponding received symbol.

$$V_{mn} = I_n \tag{4.2}$$

2. *Updates the messages from check nodes to variable nodes*: In the first half of the iteration, the CN m receives the messages V_{mn} from the neighboring VNs and propagates back the updated messages C_{mn} computed as:

$$C_{mn} = \prod_{j \in \{N_m \backslash n\}} sign(V_{mj}) \times \Phi^{-1}\Bigg\{\sum_{j \in \{N_m \backslash n\}} \Phi(|V_{mj}|)\Bigg\} \tag{4.3}$$

where N_m represents the set of all VNs connected to CN m, and $N_m \backslash n$ denotes the set of VNs excluding n connected to the CN m. $\Phi(x)$ is a non-linear, non-limited function defined as

$$\Phi(x) = \Phi^{-1}(x) = -ln\Big(tanh\Big(\frac{x}{2}\Big)\Big) \tag{4.4}$$

3. *Updates the messages from variable nodes to check nodes*: In the next half of iteration, the VN n receives the messages from the neighboring CNs and

propagates back the updated messages V_{mn} computed as:

$$V_{mn} = I_n + \sum_{i \in \{M_n \backslash m\}} C_{in} \tag{4.5}$$

where I_n is the intrinsic LLR of the current bit n, M_n is the set of all check nodes connected to the variable node n, and $M_n \backslash m$ is the set of CNs excluding m connected to the VN n.

4. *Check stop criterion*: The LLR Q_n of each symbol is calculated as:

$$Q_n = I_n + \sum_{i \in \{M_n\}} C_{in} \tag{4.6}$$

A hard decision is made according to the signs of the messages of each symbol as follows:

$$X_i = \begin{cases} 1, \text{if } Q_n > 0 \\ 0, \text{else} \end{cases} \tag{4.7}$$

The decoded codeword is created as:

$$X_w = (X_1, X_2, X_3, ...X_n) \tag{4.8}$$

If the parity check equation $H.X_w^T = 0$ is satisfied, the decoding stops and X_w is considered as valid codeword. Otherwise, the algorithm repeats from step (2), until the maximum number of iterations is reached.

These steps can be well-described in the form of a data-flow diagram, as shown in Figure 4.2. In order to reduce the complexity of the check node computation (Equation 4.3), in state-of-the-art decoders, a reduced complexity min-sum (MS) [40] decoding version of the SP algorithm is employed that introduces a slight loss in performance (communication efficiency). Equation 4.3 is estimated for MS algorithm as:

$$C_{mn} = \left\{ \prod_{j \in \{N_m \backslash n\}} sign(V_{mj}) \right\} \times \min_{j \in \{N_m \backslash n\}} (|V_{mj}|) \tag{4.9}$$

To compensate for the loss in performance of the min-sum algorithm a modified min-sum (MMS) algorithm [42] is also proposed, which would achieve a nearly comparable performance to SP algorithm. Another algorithm, the so-called layered belief propagation (LBP) [41], has a different node update schedule than the SP algorithm. Unlike the SP algorithm that first updates all CNs followed by the update of all VNs at any iteration, the LBP updates a single or a certain number of CNs (called layer), followed by the update of the whole sub-set of connected

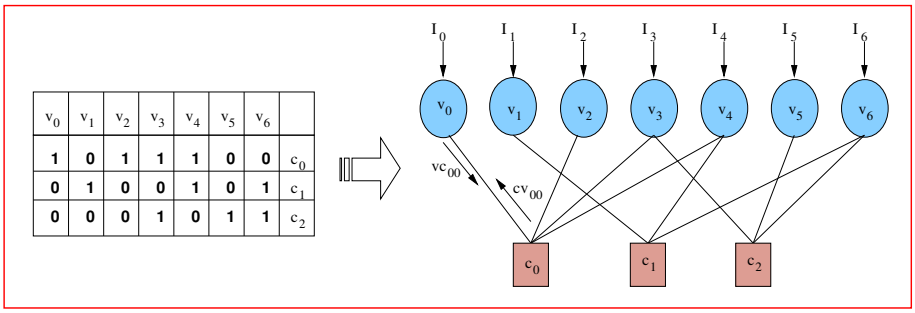

Figure 4.1: PCM for a (7,4) LDPC code and its corresponding Tanner graph, where $\{v_0.....v_6\}$ represents variable nodes, $\{c_0.....c_2\}$ represents check nodes and $\{I_0.....I_6\}$ represents the input intrinsic channel information

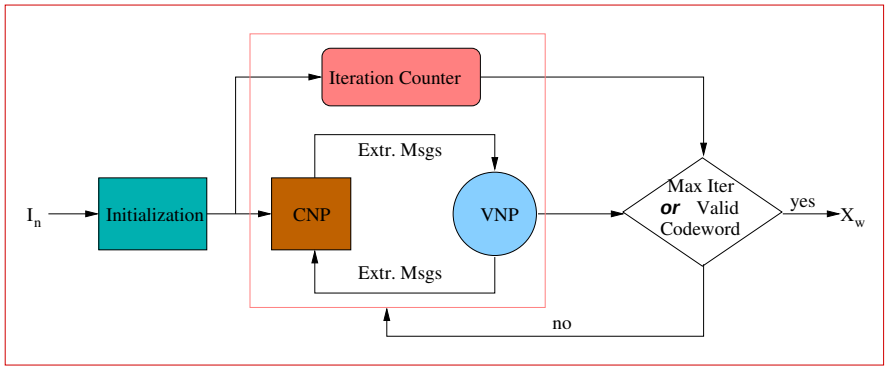

Figure 4.2: Decoding flow diagram representing the main steps of MPA algorithm

VNs, and so on. For example, Figure 4.3 shows this layer-wise decoding for the (7,3) LDPC code. After the check node update $\{C_1\}$, all the connected variable variable nodes $\{V_1, V_3, V_4, V_5\}$, highlighted in red, are updated followed by the next check node $\{C_2\}$ and its connected variable nodes, as shown in green in Figure 4.3. The advantage of LBP is its almost double faster convergence. The MMS-LBP is the most popular algorithm adopted in the state of the art decoders due to their comparable performance figures to SP algorithm, lower complexity and an almost *2-fold* higher throughput. However, the sequential layer-wise processing is the main limitation of the LBP algorithm, what prevents its utilization in applications that require ultra-high throughput.

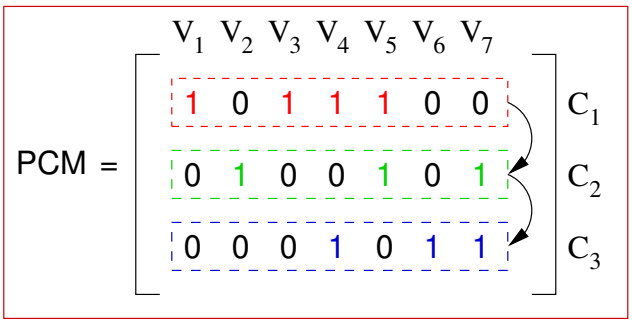

Figure 4.3: PCM for a (7,4) LDPC code and its layer-wise decoding using layered belief propagation (LBP) algorithm

4.3 LDPC Decoder Architecture Template and Template Features

Figure 4.5 shows in detail a graphical representation of the top-level of the generic architecture template for LDPC decoding. It prescribes the presence and general structural organization of the decoder's architectural resources. It involves parameterized elementary VNP and CNP processors, various memories (M_{cv}, M_{vc}, M_{ch}, M_{HD}), a communication network with several communication segments that enables the communication among various processors and memories, several read-only-memories (ROMs) that can be configured to particular PCM, Hard Decision and Parity Check Unit, as well as, the Main Sequencer/Controller and Channel I/O Interface. The M_{cv} and M_{vc} are shared between the CNP and VNP processors, while the M_{ch} and M_{HD} are used by the VNP processors for reading of channel data and writing of decoded messages, respectively, as shown in Figure 4.5. Different instances of the generic architecture templates and their processing, memory and communication modules define different specific accelerators.

Since the processing elements are generic, therefore, for each kind of processing elements and related computations that are mapped on the processing elements, a certain computation time (measured in clock cycles) depending on the micro-parallelism of the processing unit is assumed. For example, a fully-parallel processing element would perform the computation in a single cycle, while the serial would take as many clock cycles as the total number of inputs. Moreover, the generic processors, both the CNP and CNP, are designed in such a way that they support in parallel the processing of two check nodes or two variable nodes, respectively. For instance, the serial and partially-parallel processing units (VNP, CNP) overlap the processing of two nodes. Further, details on the micro-architecture design of CNP and VNP processors can be found in the next section of this chapter.

Moreover, the memory operations (load and store) are performed in parallel with the execution of processing elements in the proposed architecture template, as shown in Figure 4.4. This way the total execution cycles for all the decoding computations would reduce to just the computation time taken by the processing units hiding the memory read (load) and write (store) latencies. In each clock cycle, as many memory operations (load and store) can be performed as the computation operations require to match the computation loads with memory loads. An important feature of the proposed architecture template is that any memory can communicate with any processor through a configurable communication network. Such application-specific communication networks are unavoidable for this kind of demanding applications due to the complex global and irregular information (data) flows in the application.

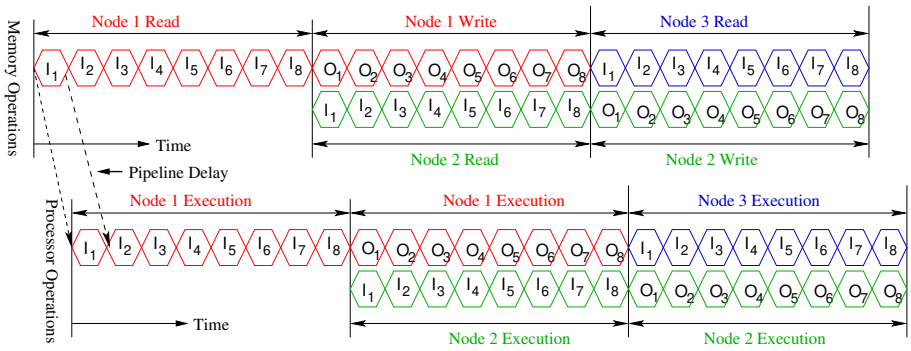

Figure 4.4: Memory operation is performed in parallel with the processor execution in the proposed top-level generic architecture template for LDPC decoding

One of the important design quality metrics for LDPC decoding is the throughput. The throughput in Mbps (T_{Mbps}) for the LDPC decoding can be estimated based on the message passing (MP) algorithm and the parameter values of the architecture elements using the following formula:

$$T_{Mbps} = \frac{R.N.F_{MHz}}{CCPI.I_{tot}} \tag{4.10}$$

where R stands for the code rate, N stands for the code length (size of data frame), I_{tot} stands for the total number of iterations required to decode a frame, F_{MHz} stands for the clock frequency measured in MHz and $CCPI$ represents the number of clock cycles per iteration. Knowing the values on the right hand side of equation 4.10, the throughput can be estimated. For a particular application, the LDPC code and its parameters such as N, R, I_{tot} and Frame Error Rate (FER) are decided in advance. Therefore, the parameters that remain to determine the throughput are the $CCPI$ and F_{MHz}. The $CCPI$ directly depends on the

micro- and macro-architecture parallelism exploited. The F_{MHz} depends on the processor's critical path delays plus the physical delays of the communication and memory structures.

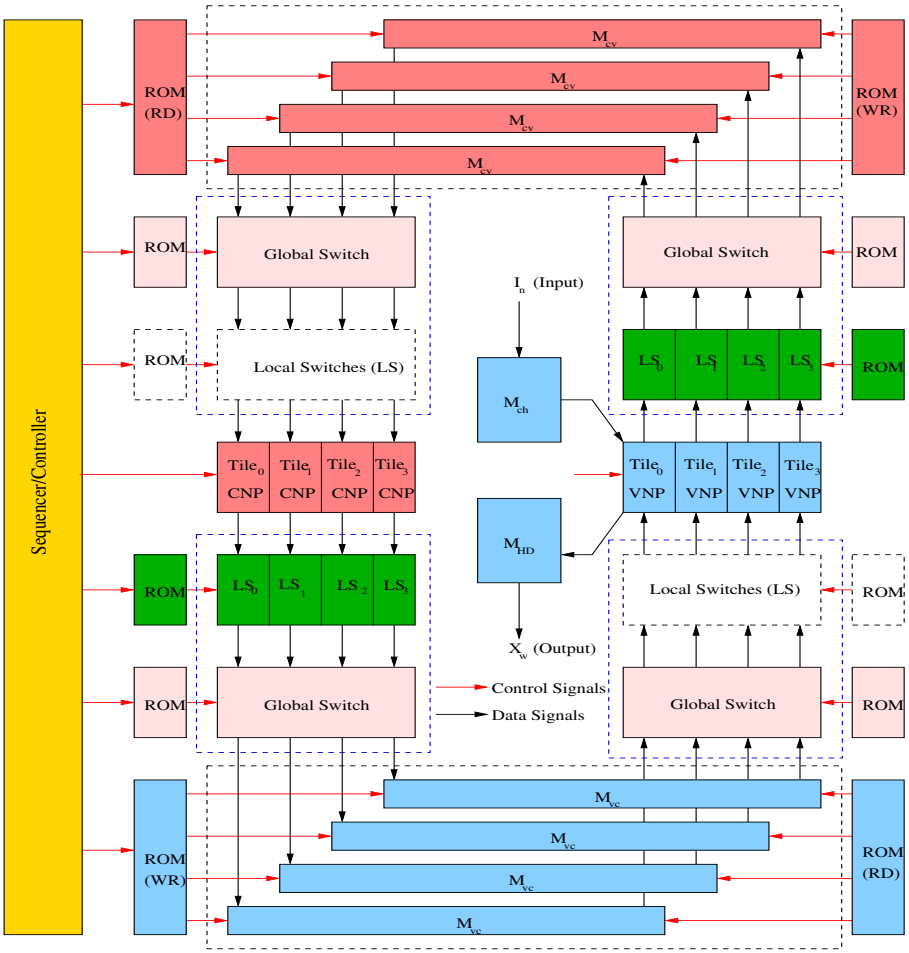

Figure 4.5: A detailed view of the top-level LDPC decoder architecture template

Above the main architectural elements and the operational features of the proposed generic architecture template for LDPC decoding are explained. Here, those parameters are discussed which are of interest for the design's decision-making during the actual DSE for a specific set of design requirements. The generic architecture template is a scalable multi-processor template in the range of hundreds of heterogeneous hardware processors (CNP, VNP) at the macro-

architecture level. This would enable us to consider different number and types of processors during the DSE for a particular set of requirements, and this way to explore the influence of various micro-/macro-parallelism combinations on the system parameters, like delay, latency, performance, area and power consumption, etc. For each type of processor, the generic architecture template supports various micro-architectures with different area, delay, latency and power characteristics, which in turn influence the design parameters in different ways and to different degrees. For instance, different processor micro-architectures for a specific task can result in different number of clock cycles to execute the task and the related clock speed and latency (which would influence the performance), amount of hardware resources and power consumption, etc. Importantly, the micro-/macro-architecture combinations can result in many various architectures with much higher diversity in different design dimensions (e.g. area, delay, latency, throughput, power, etc). Similarly, the generic architecture template includes the support of various generic memory and communication structures that are different regarding their features (e.g. area, delay, latency, throughput, power consumption, etc). For instance, various kinds of interconnection structures (e.g. single-stage and multi-stage switches and cyclic shifters) and various kinds of memory structures (e.g. single or multi-port, vector or multi-bank, SRAM or register-based). This way, with the assistance of the top-level generic architecture template and its generic modules, many architectures with diverse processor micro-/macro-architectures, as well as, the processor's compatible memory and communication architectures can be selected, decided and constructed during the DSE with different physical features based on the actual application behavioral and parametric constraints and objectives. Moreover, the top-level generic architecture template when instantiated enables a fast track for the final evaluation and synthesis of various decoder architectures using the back-end SoC and FPGA synthesis and physical design tools. In the next section, each architecture element is described in detail, as well as, the design, modeling and characterization of the architecture elements with values and ranges of different parameters expressing their characteristic features of interest for the architecture design decisions.

4.4 Design of Generic Component Library

The generic components library includes a top-level generic architecture template, as well as, generic templates of application-specific processors, memories and communication components. The components are designed and modeled in Verilog HDL that can be targeted to various implementation technologies. For performing the experiments reported in this work, it has been targeted at CMOS 90nm technology[1]. The processors are configurable which enable us to explore the various tradeoffs at the micro-architecture level. Analogously, various tradeoffs can be explored for the configurable memory and communication modules.

[1]TSMC 90nm LPHP Standard Cell Library

4.4.1 Processing Elements Design and Characterization

In this section, the architecture design and characterization of the processing elements used for the computation of check and variable node processing are discussed. Also, numerous possible partially-parallel micro-architectures are discussed that are situated between the two extremes of fully-serial and fully-parallel architectures, as well as their characterization.

Check Node Processor Micro-architectures

Equation 4.9 represents the computations performed by the check node processor (CNP). Each output (check to variable node messages (C_{mn})) is determined by finding the minimum value among all d_c input values (variable to check node messages (V_{mn})) excluding one input. The excluded input is the variable to check node messages (V_{mn}) from that variable node n for which the output (check to variable node message (C_{mn})) is computed. Figure 4.6 further clarify this input/output relationship.

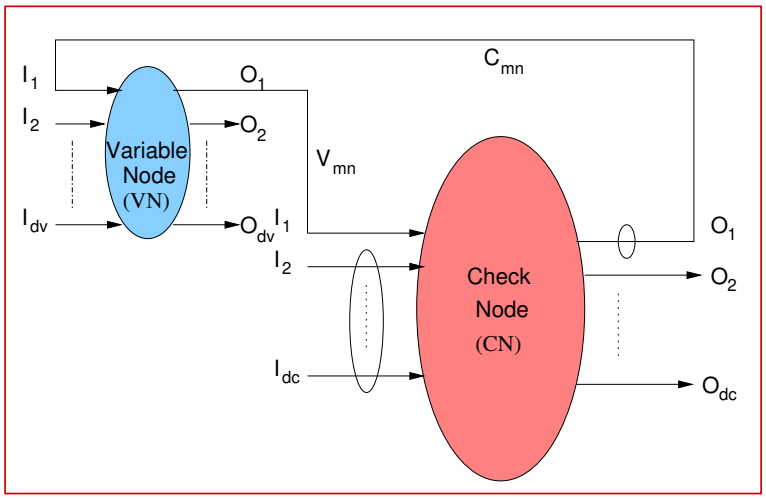

Figure 4.6: Check node computation and input/output relationship

For every check node, a total of d_c minimum values are computed that are sent to the d_c connected variable nodes $\{v_1.....v_{dc}\}$. Hence, using a single binary search tree for the minimum computation, it would require to repeat the same computation d_c times to compute all the minimum values, as shown in Figure 4.7. Otherwise, it would require d_c binary search trees to compute all the minimum values in parallel. On one hand, computing all the d_c minimum in parallel using d_c independent binary search trees would have a huge computational complexity

of d_c^2 in terms of area and $log_2(d_c - 1)$ in terms of delay. On the other hand, implementing even a single binary search tree for a large number of inputs (32 inputs in case of rate-7/8 672-bit IEEE 802.15.3c LDPC code) is costly as it would require $d_c - 1$ comparators but also would result in a huge delay of $log_2(d_c - 1)$ due to large number of comparator stages (in this case 5). Further, this direct parallel implementation of the binary search tree would make the partially-parallel micro-architecture impossible and also limits the configurability of the processing units for different d_c values.

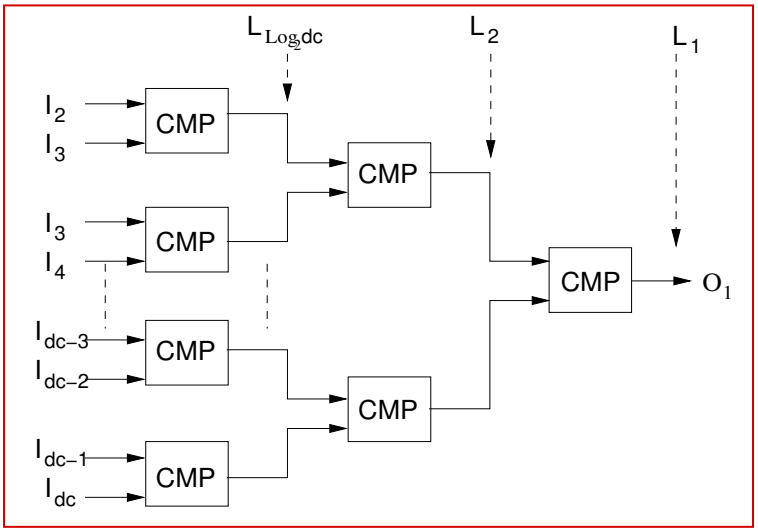

Figure 4.7: Direct implementation with multiple binary comparator trees each computing a single output of the CN, the figure shows the implementation for the output O_1.

In the second possible approach, the comparisons are organized, combined and distributed in such a way that at the end one get the individual minimum excluding the input for which the output has to be computed (a sort of compression and expansion binary trees), as shown in Figure 4.8. This approach for the CN computation has been presented in the past for the LDPC decoders [80]. The micro-architecture of the CNP based on this approach has the computational complexity of $d_c/2$ in terms of comparator stages and $3(d_c - 2)$ in terms of number of two-input comparator units (CMP). This micro-architecture has the drawback that it could only be used for parallel micro-architectures of processing elements. Further, it can only support a certain number of inputs, and cannot be scaled for various numbers of inputs.

To address the above drawbacks, we proposed the following approach for the

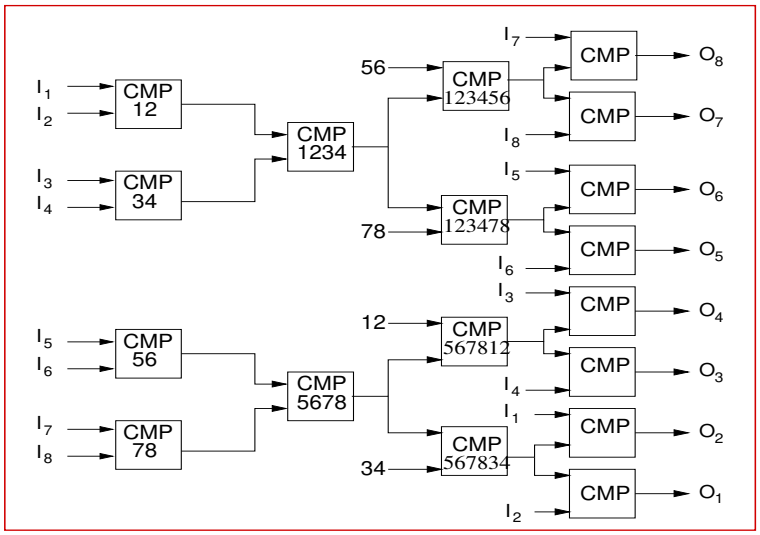

Figure 4.8: Parallel micro-architecture with computation reorganization based on combining intermediate summations and shuffling

CN computation. First, the two minimal values, minimum (min) and second minimum (min_2), among all the input values are computed. Based on the min and min_2 computation, the computations of all the outputs can be performed as follows:

- the min is assigned to those outputs whose value is greater than or equal to the min_2, and
- the min_2 is assigned to those outputs whose value is equal to the min.

It can be represented using the following equation:

$$O_x = \begin{cases} min_2, \ if \ Idx_x == Idx_{min}, else \\ min, \ where \ x = \{1, ..., d_c\} \end{cases} \tag{4.11}$$

where Idx_x represents the index of an arbitrary input x and Idx_{min} represents the index of the min. This way, for every check node all the d_c minimum values (outputs) for all the connected variable nodes $\{v_1.....v_{dc}\}$ are determined using just a single search tree. In this case, the search tree computes only the min and min_2 among all the input values.

This approach delivers the following two main benefits. Firstly, all the d_c minimum computations can be performed using a single binary search tree, without multiple binary trees that search for the minimum of $d_c - 1$ values independently. Secondly, to cope with the complexity of the single binary search tree

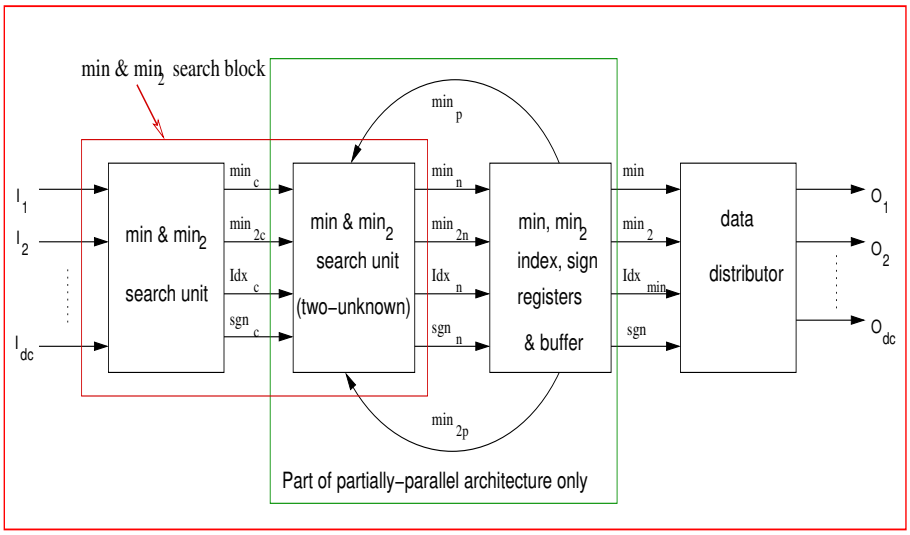

Figure 4.9: Top level scalable generic architecture of check node processor

that computes the min and min_2, this search can be performed in any fashion: serial, partially-parallel or fully-parallel. This enables to construct different micro-architectures with different tradeoffs regarding delay, number of cycles (cycle count), area and power consumption, etc.

The top-level architecture of the CNP based on the proposed approach is shown in Figure 4.9. The architecture composed of two kinds of min & min_2 *search* blocks, a *storage* block and a *data distribution* block. The CN computation is organized in two stages in the CNP.

In the first stage, the min and min_2 are computed for a CN among all the inputs. It is performed as follows. For a certain sub-set of inputs, during the first clock cycle the architecture computes the (current) minimum (min_c) and second (current) minimum (min_{2c}) using the search unit (min & min_2 *search unit*). The size of the sub-set depends on the partial parallelism. For instance, 1 in case of serial, P_{mic} in case of partial parallel with micro-parallelism of P_{mic} and d_c in case of fully-parallel. In the same cycle, these values are then combined with the previous minimum (min_p) and the second minimum (min_{2p}) value (if any[2]) to compute the new minimum (min_n) and the second minimum (min_{2n}) and is stored in temporary registers, as shown in Figure 4.9, which becomes min_p and min_{2p} in the next clock cycle. Then, in the second clock cycle, the min_n and min_{2n} is computed for the next sub-set of inputs and combined with the min_p and min_{2p} (see Figure 4.10-A). This way all the inputs of a CN are processed sequentially in sub-sets to find the global minimum (min) and second

[2]For the first sub-set of inputs, they are compared with the maximum values

global minimum (min_2). Also, in this stage, the sign computation, represented in equation 4.9, is performed by a simple XOR (exclusive-OR) operation of the signs of the inputs. At the same time, the signs of the inputs are pushed into a sign FIFO that are used latter in the second stage during the computation of the signs of outputs. At the completion of processing of all the inputs for a particular CN, the global min and min_2 are moved into a buffer from the temporary registers (see Figure 4.10-B).

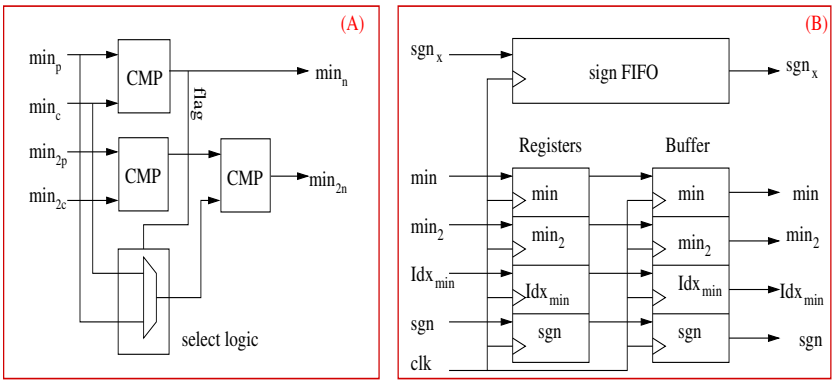

Figure 4.10: (A) min & min_2 search unit (two-unknowns) (B) Storage unit composed of temporary registers, a buffer and a sign FIFO

In the second stage, the computation of the outputs for the current CN is initiated in the distributor using the global min and min_2 and index of the minimum (Idx_{min}) stored in the buffer module. Also, the signs of inputs are popped up from the sign FIFO, XOR with the total sign value (sgn) stored in the buffer to compute the signs of the outputs. In parallel, the search unit initiates the computation of the next CN, i.e. its first stage. This is possible because the search unit is free during the second stage of the current CN processing. This parallel (overlap) processing of two check nodes (CNs) in a CNP is shown in Figure 4.11, which is our proposed throughput enhancement strategy.

The number of clock cycles required to complete the processing of a single CN is d_c and d_c/P_{mic} for the serial and partially parallel (micro-parallelism of P_{mic}) architectures, respectively. However, the fully-parallel CNP performs the CN computation in a single clock cycle. Mathematically,

$$T_{cc} = d_c/P_{mic},\ P_{mic} = \{1, ..., d_c\} \tag{4.12}$$

where T_{cc} represents the total number of clock cycles required for processing a single check node. Let $D_{P_{mic}}$ represent the critical path delay of a CNP of micro-parallelism P_{mic}, then the total computation time can be determined as follows:

$$T_{P_{mic}} = T_{cc} \times D_{P_{mic}},\ P_{mic} = \{1, ..., d_c\} \tag{4.13}$$

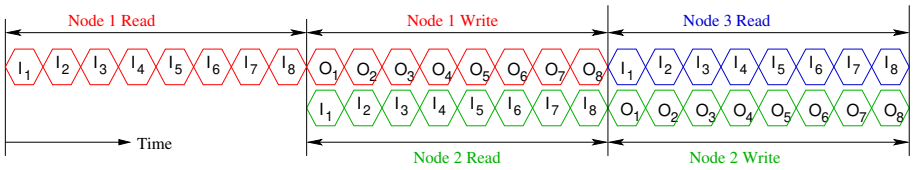

Figure 4.11: Parallel (overlap) processing of two CNs in a CNP having d_c = 8 and process inputs in serial

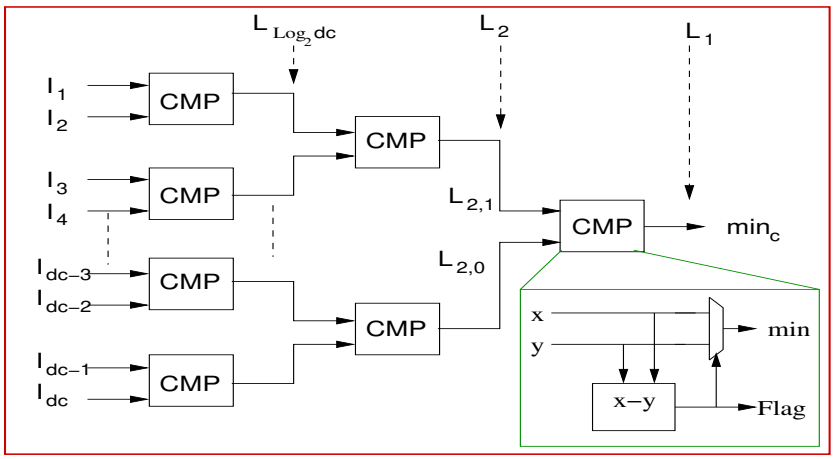

Figure 4.12: Minimum computation part of the min & min_2 search unit

where $T_{P_{mic}}$ represents the total execution time for processing a single node using a processor with micro-parallelism of P_{mic}. The $T_{P_{mic}}$ can be used as a useful measure of timing, as serial processors have relatively low critical path delays, however, require a large number of clock cycles to compute a node, and vice versa for parallel processors. Hence, these tradeoffs have to be taken into account during the micro-architecture exploration.

The internal structure of each block of the CNP is described in more detail as follows. The *min* & min_2 *search unit* can be divided into two parts for understanding purposes. One part computes the *min* and the other min_2. The part that computes the *min* is composed of two-input compare-select-flag units (CMP). The CMPs are organized in a binary search tree to compute the *min*, as shown in Figure 4.12. The CMP unit computes the *min* of two inputs by subtraction and then based on the sign flag selects the *min*. The flag is used further in the search for the min_2 after finding the *min* as follows. The tree is traced back based on the CMP flags of each stage to find min_2 in the following manner. Starting from

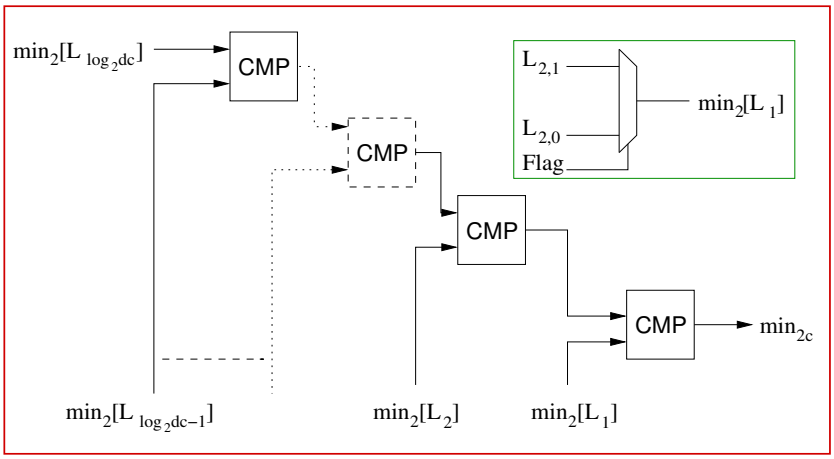

Figure 4.13: Second minimum computation part of the min & min_2 search unit

the leaf node, at each level of the binary search tree the min_2 is traced back and is compared with another min_2 one-level up on one of the selected branch of the tree. That tree branch is selected on which the min is found, for the other tree branch, i.e. of min_2, it is obvious that a value less than min_2 is not possible, as it is minimum search tree and therefore a candidate for min_2. Hence, that min_2 branch is not searched further and supplied as the lower input of the leaf CMP unit, as shown in Figure 4.13. This way another min_2 is searched that will make the lower branch of another CMP on the sequential tree till the root of the tree is reached that makes the upper branch of the tree, as shown in Figure 4.13.

The data distributor block generates 1, P_{mic} or d_c number of outputs for serial, partially-parallel and fully-parallel micro-architecture, respectively, based on the min, min_2 and the Idx_{min}, as shown in Figure 4.14. To generate the min for each output, the minimum index (Idx_{min}) is compared with the index of the output (Idx_x). In case of match that output will get the min_2, otherwise it will be assigned the min value, as given in equation 4.11. This approach needs a comparison operation of each output index (Idx_x) with the index of the minimum (Idx_{min}). For low micro-parallelism or codes with low d_c values, the direct comparison is possible with low complexity, as only a few comparators would be required. However, for high micro-parallelism this would required a large number of comparators equivalent to the micro-parallelism (P_{mic}). For example, the CN of degree 32 of the rate-7/8 672-bit IEEE 802.15.3c LDPC code, when realized in fully-parallel would require 32 such comparators, which is costly in terms of resources (area). Hence, the data distributor is designed in such a way that it avoids the comparison operations, as shown in Figure 4.14. It works as follow.

First, the Idx_{min} is decoded by a decoder of size $log_2(d_c) \times d_c$. Then, based

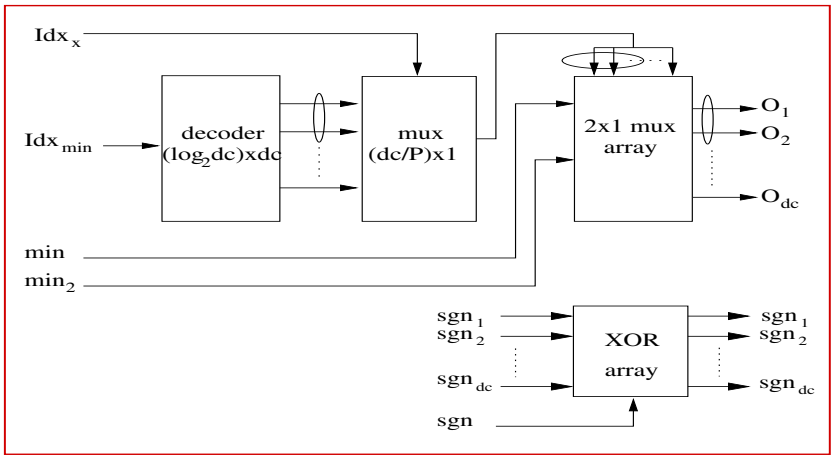

Figure 4.14: Generic and scalable date distributer unit of check node processor

on the sub-set of outputs, i.e. P_{mic}, for which the min has to be computed, that group of the decoder output is selected by a group select multiplexer of size $d_c/P_{mic} \times 1$. The output of this multiplexer acts as a select line of a 2×1 multiplexer array that selects min or min_2 for each output based on whether the decoded value is 0 or 1. For example, for $d_c = 32$ and $P_{mic} = 8$, a decoder of size 5×32, a group select multiplexer of size 4×1 (4 groups each of 8 inputs) selecting 8 decoder outputs, and a 2×1 multiplexer array of size 8 would be required for this data distributer. The proposed data distributor is also generic and scalable to different number of outputs just by changing the parameter values of different sub-blocks.

For micro-architecture with $P_{mic} \leq 2$, the above architecture of CNP does not seem to be adequate, because for serial architecture only a single input has to be compared with the previous min_p and min_{2p} to compute the new min_n and min_{2n}. Therefore, we propose a separate specialized architecture for the serial case, as shown in Figure 4.15-A. For two inputs, it is possible to use the two-level min and min_2 search structure by first computing the min and min_2 and then passing down to the final search stage, min & min_2 *search unit* (*two-unknowns*), to compute the new min_n and min_{2n}. Instead of doing this, the two-stages are combined, as shown in Figure 4.15-B. This results in saving of at least one comparator and reduction in the critical path delay.

The proposed generic architecture of the CNP is utilized in the architecture DSE framework to explore the various tradeoffs regarding different micro-/macro-parallelism combinations for certain design requirements. A particular instance[3] of the proposed generic CNP micro-architecture with 4-inputs and 4-outputs (32-

[3]Schematics of the synthesized CNP processors can be found in the Appendix-B

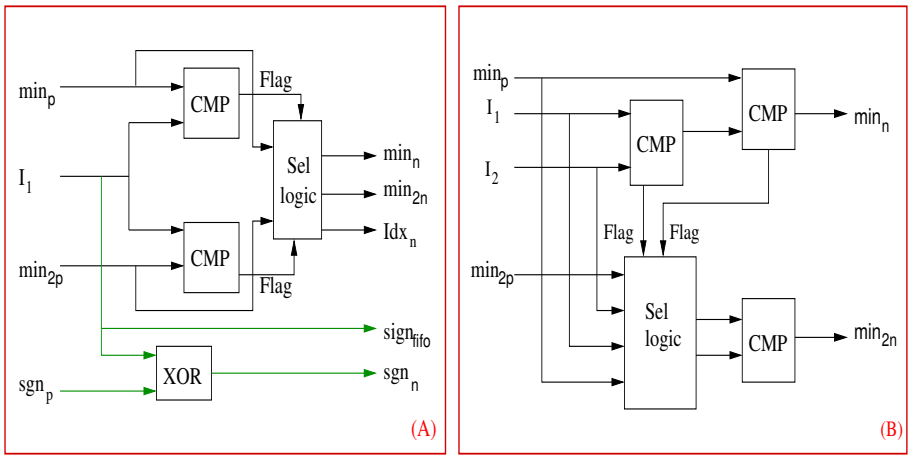

Figure 4.15: (A) Serial micro-architecture with a single input (B) Partially-parallel micro-architecture with two inputs. Note: (A) and (B) show only the min & min_2 search unit part of the CNP and not its storage and distribution unit

inputs in total) is shown in Figure 4.16. It should be noted that such highly optimized application-specific generic processor architecture templates cannot be constructed (designed) automatically with the high-level-synthesis (HLS) tools.

Although the CNP architectures similar to the proposed approach can be found in the literature, but they exploit it only for the fully-serial [52–59, 61, 63] or the fully-parallel micro-architectures [64, 71, 83]. On the other hand, we exploit the approach for the full range of micro-architectures from the partially-parallel (except the serial and the two inputs partially-parallel micro-architecture) to the fully-parallel micro-architectures. This actually requires a large micro-architecture level modifications (e.g. different min and min_2 search units, sign FIFO depths and widths, different data distributors, etc) and is not directly scalable at all.

In order to overcome the large critical path delays of the fully-parallel CNP micro-architectures, the architectures mentioned above [64, 71, 83] used aggressive pipelining (in some cases multiple pipeline stages) within the CNP micro-architecture to increase the clock speed, thereby increasing the throughput. Although pipelining is a well known throughput enhancement strategy. However, for the high-end applications that requires a large number of such processors, it would cause a huge increase in the accelerator cost and power consumption due to the extremely large number of required pipelined registers. It should worth to note that each processor requires as many number of pipeline registers as the micro-architecture parallelism and not a single register. Moreover, for the fully-parallel pipelined micro-architecture, pipelining is also required on the memory and communication side due to the following two reasons:

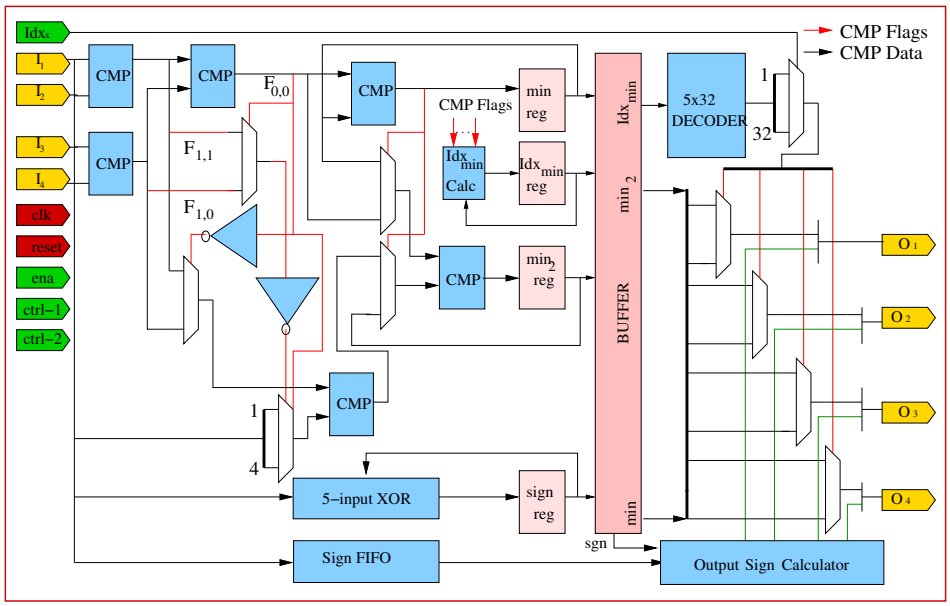

Figure 4.16: A particular instance of the generic CNP with 4-inputs and 4-outputs (32-inputs in total)

- the memory and/or communication structure delays may surpass the processors delays after the processor pipelining, and
- to balance or synchronize the pipeline.

This causes a further increase in the overall accelerator cost and power consumption, etc. Thus, for high-end applications that demand a large number of processors, processor micro-architecture pipelining does not seem anymore to be a valid throughput enhancement strategy. Its adequacy have to be carefully explored. Therefore, the proposed DSE framework also takes into account this issue. Moreover, the proposed various partially-parallel micro-architectures limits the critical path in a much better way just by reducing the number of inputs processed simultaneously, unlike in the state-of-the-art CNP processors by using aggressive structure pipelining of the fully-parallel micro-architecture to enhance the throughput or meet the clock speed constraint.

Characterization of Check Node Processor

The proposed generic check node processors are modeled in Verilog HDL and characterized its various instances in TSMC 90nm LPHP standard cell library. The characterization results are shown in Table 4.1. Although the generic check

Table 4.1: CNP characterization results for different micro-architectures parallelism in TSMC 90nm LPHP Standard Cell Library

Parallelism (P_{mic})	d_c	Optimized for Area*			Optimized for Speed		
		Area (mm^2)	Delay (ns)	Clock Cycles	Area (mm^2)	Delay (ns)	Clock Cycles
1	8	0.001865	2.303	8	0.002759	0.751	8
2		0.002501	5.119	4	0.003933	1.413	4
4		0.003487	7.137	2	0.005998	2.105	2
8		0.003615	9.388	1	0.008709	2.709	1
1	16	0.002115	2.303	16	0.002969	0.746	16
2		0.002936	5.119	8	0.004361	1.413	8
4		0.003894	7.137	4	0.006403	2.105	4
8		0.005808	10.914	2	0.010949	3.175	2
16		0.007459	12.617	1	0.018210	3.556	1
1	32	0.002539	2.303	32	0.003412	0.751	32
2		0.003714	5.119	16	0.005155	1.413	16
4		0.004647	7.137	8	0.007149	2.105	8
8		0.006525	10.914	4	0.009810	3.175	4
16		0.010239	14.061	2	0.020649	3.972	2
32		0.015102	16.710	1	0.034077	4.816	1

*Area of a NAND Gate in CMOS 90nm = 2.1168 um^2

node processor can be configured for any micro-parallelism and check node degree (d_c). However, Table 4.1 shows only the results for the micro-parallelism and check node degree (d_c) configurations employed in the IEEE 802.15.3c LDPC codes, that would be further used for the discussion on the various micro-/macro-architecture tradeoffs in the next chapter.

Further, the proposed CNP micro-architectures are characterized using the following two optimization criteria:

- optimization for area, and
- optimization for performance

In optimization for area, the binary search tree is implemented using the low-area ripple carry adders (RCA). However, for high-performance, the same architecture is realized using the fast carry look ahead adder (CLA) albeit with much larger increase in area compare to its RCA-based counterpart, as shown in Figure 4.17. The critical path delay of the CNP most of the time decides the critical path of the whole accelerator.

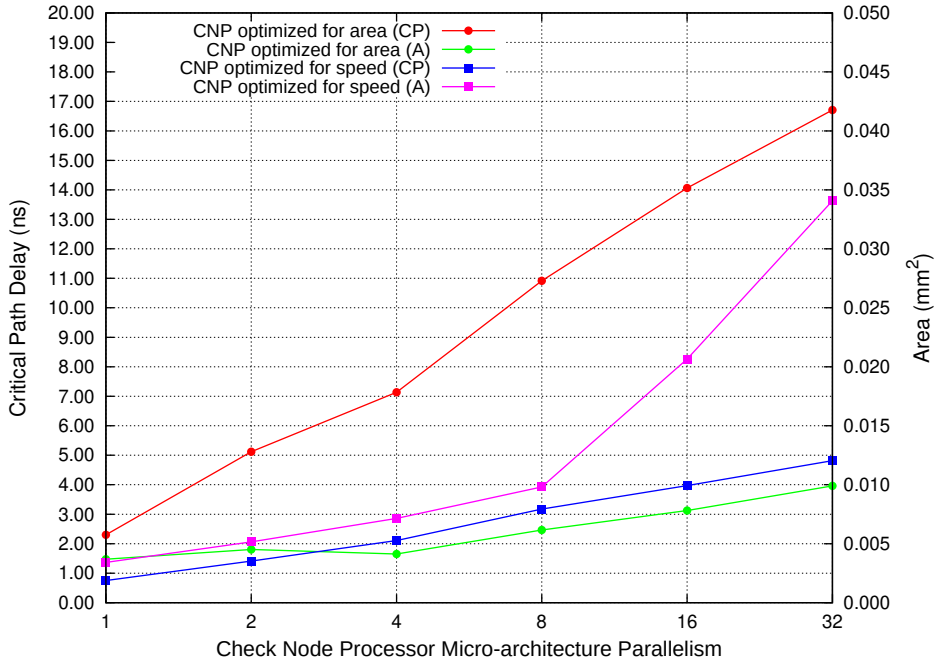

Figure 4.17: Area and critical path (CP) delay tradeoffs for different micro-parallelism of CNP: Case 1: Processor optimized for area (RCA-based adder tree realization), Case 2: Processor optimized for speed (CSA-based adder tree realization)

Variable Node Processor Micro-architectures

Variable node computations are represented by equation 4.5 and 4.6. Equation 4.5 determines the sum of all d_v input values (check to variable node messages (C_{mn})) excluding one input. The excluded input is the check to variable node messages (C_{mn}) from that check node m for which the output (variable to check node message (V_{mn})) is computed. Figure 2 further clarify this input/output relationship.

For every variable node a total of d_v such additions are performed that are sent to the d_v connected check nodes $\{c_1.....c_{dv}\}$. A single adder tree can be used to perform the addition, however, it would require to repeat the same computation d_v times for every variable node. On the other hand, it would require d_v adder trees to perform all the additions in parallel. Moreover, from equation 4.6, it is clear that all the inputs including the I_n have to be added together to compute Q_n. Using this computation organization, it would require as many as v_d adder trees, each adder tree with $log_2(d_v)$ number of stages of 2-input adders. In terms

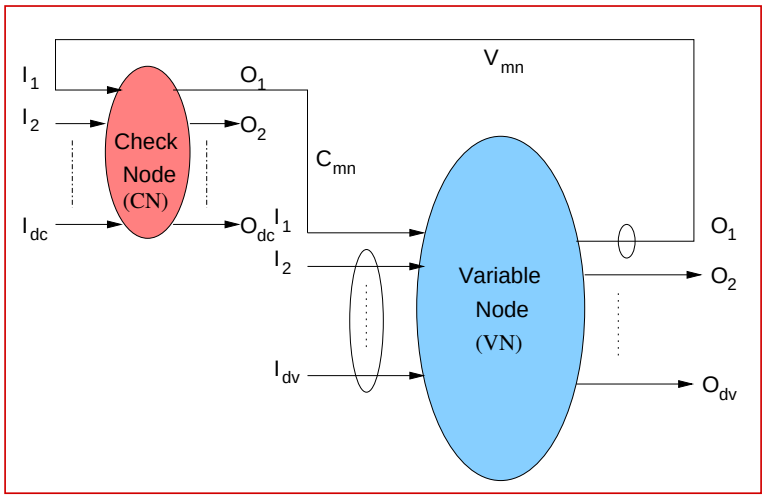

Figure 4.18: Check node computation and input/output relationship

of two-input adders, it would require $d_v(d_v\text{-}2)$ such adders. A single binary adder tree for a single output computation is shown in Figure 4.19.

Another parallel computation approach, however, is possible in which the additions are first combined and then distributed (of a sort of compression and expansion binary trees) in such a way that at the end one get the individual addition results excluding adding the input for which the output has to be computed, as shown in Figure 4.20. This approach for the VN computation has been presented in the literature for the LDPC decoders [59, 80]. The approach has the computational complexity of $d_c/2$ in terms of the number of the comparator stages and $3(d_v\text{-}2)$ in terms of number of the two-input adder units (ADD). The approach has the drawback that it can only be used for parallel VNP micro-architectures. Moreover, it can only support a certain number of inputs that cannot be easily scaled to various number of inputs. Also, for a large number of inputs (12 inputs in case of IEEE 802.11n LDPC code) it would cause a large delay due to a large number of adder stages.

In order to address the above drawbacks, we propose the following approach for the VN computations. To reduce the overall computational complexity both in terms of area (adder trees) and delay (adder stages), as well as, to make possible configuration of the partial parallel processing unit for different number of inputs, the computation can be reorganized as follows:

- first compute the total sum including the channel data, when using a single adder tree, and
- then compute the output variable to check messages V_{mn} for a particular

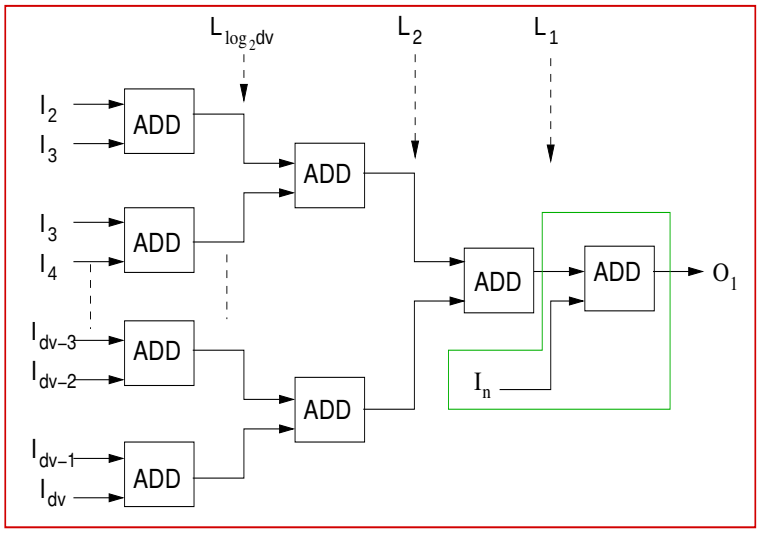

Figure 4.19: Direct implementation with multiple binary addition trees each computing a single output of the VN, the figure shows the implementation for the output O_1

variable node n connected to a check node m, by subtracting that check to variable node message C_{mn} from the total sum.

This way for every variable node all the d_v outputs (V_{mn} messages) for the connected check nodes $\{c_1.....c_{d_v}\}$ are determined using just a single adder tree and a subtractor array composed of d_v subtractors. Furthermore, the adder tree can be realized in serial, partially-parallel or fully-parallel form to overcome the long critical path delays.

The top-level generic architecture for the VNP based on the proposed approach is shown in Figure 4.21. It consists of an adder tree with scalable number of inputs ranging from a single input to all d_v inputs, accumulator, FIFO, buffer and a subtractor array.

The VN computation is organized in two stages in the VNP. In the first stage, all the inputs are added together including the channel input (I_n). It is performed as follows. For a certain sub-set of inputs (1 in case of serial, d_v/P_{mic} in case of partial parallel with micro-parallelism of P_{mic} and d_v in case of fully-parallel), during the first clock cycle the adder tree sum up the current inputs with the channel input (I_n) and is stored in the accumulator register as the current sum (sum_c), as shown in Figure 4.21, which becomes the previous sum (sum_p) in the next clock cycle. Then, in the second clock cycle, the next sub-set of inputs is added together with the sum_p. A multiplexer is used to select between the sum_p and the I_n required, because I_n is required to be added only once. This way

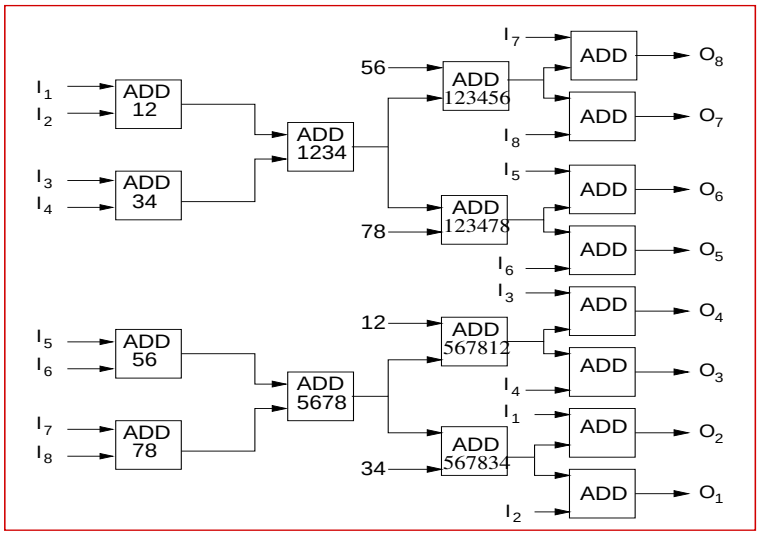

Figure 4.20: Parallel micro-architecture of VNP with computation reorganization based on combining intermediate summations and shuffling

all the inputs are summed up in sub-sets (size of sub-set depends on the partial parallelism) with the sum_p to compute the total sum (sum). At the same time, the inputs are pushed into a FIFO that are used latter in the second stage during the computation of the outputs. At the completion of processing of all the inputs for a particular VN, the total sum is moved into a buffer from the accumulator.

In the second stage, the computation of the outputs for the current VN are initiated by subtracting the particular inputs popped up of the FIFO from the total sum stored in the buffer (see Figure 4.22). In parallel, the adder tree initiates the computation of the next VN, i.e. its first stage. This is possible, because the adder tree is free during the second stage of the current VN processing. This parallel (overlap) processing of two VNs in a VNP is shown in Figure 4.23, which is our proposed throughput enhancement strategy.

The number of clock cycles required to complete the processing of a single VN is d_v and d_v/P_{mic} times for the serial and partial parallel (micro-parallelism of P_{mic}) architectures, respectively. However, the fully-parallel VNP performs the VN computation in a single clock cycle. Mathematically,

$$T_{cc} = d_v/P_{mic},\ P_{mic} = \{1, ..., d_v\} \tag{4.14}$$

where T_{cc} represents the total number of clock cycles required for processing a single variable node.

The FIFO stores the input check-to-variable C_{mn} messages in the partial parallel architectures that are used latter in the computation of the outputs after

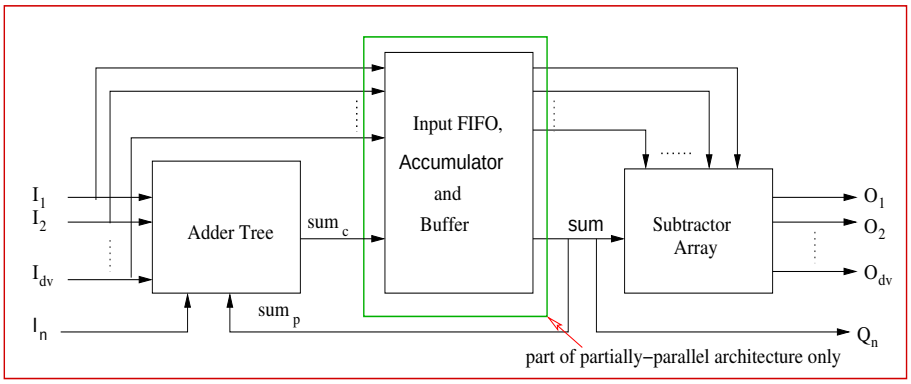

Figure 4.21: Top level scalable generic architecture of variable node processor

the completion of partial additions of all d_v inputs. This data reuse reduces the memory bandwidth requirement, however, at the cost of some local storage in the form of FIFO buffers. For the fully-parallel architecture all the d_v outputs are computed simultaneously, and the accumulator, buffer and FIFO are not required. Moreover, the adder tree can be implemented using 2-input adder or a carry-save adder, as shown in Figure 4.22. This give another level of exploration at the micro-architecture level.

The proposed generic architecture of the VNP is utilized in the architecture DSE framework to explore the various tradeoffs regarding different micro-/macro-parallelism combinations for certain design requirements. A particular instance[4] of the proposed generic VNP micro-architecture with 4-inputs and 4-outputs is shown in Figure 4.24. It should be noted that such highly optimized application-specific generic processor architecture templates cannot be constructed (designed) automatically with the high-level-synthesis (HLS) tools.

Although the VNP architectures similar to the proposed approach can be found in the literature, but they exploit it only for the fully-serial [52–57, 61, 63] or the fully-parallel micro-architectures [66, 79]. On the other hand, we exploit the approach for the full range of micro-architectures from the fully-serial to the fully-parallel with in between various partially-parallel micro-architectures. This actually requires a large micro-architecture level modifications (e.g. different FIFO depths and widths, sizes of subtractor array and adder tree, etc). The micro-architecture pipelining of the fully-parallel VNP is used by the above-mentioned architectures to reduce the critical path delays. This give rise to similar kind of issues as discussed earlier for the CNP processor.

[4]Schematics of the synthesized VNP processors can be found in the Appendix-B

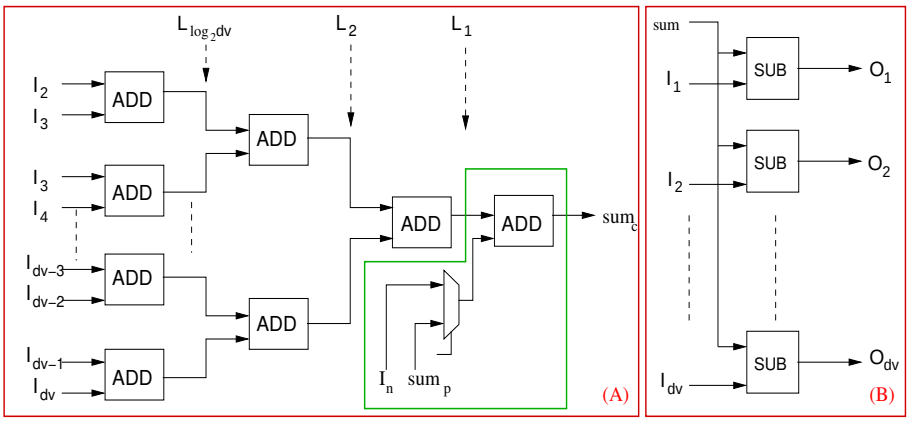

Figure 4.22: (A) Architecture of the adder tree (B) Architecture of the subtractor array

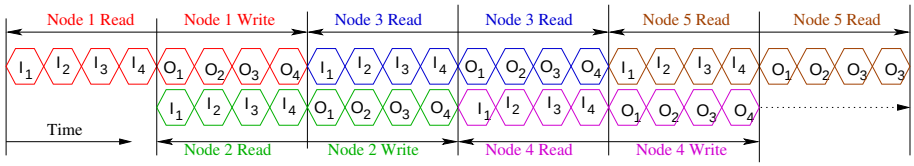

Figure 4.23: Parallel (overlap) processing of two VNs in a VNP having d_v = 4 and process inputs in serial

Variable Node Processor Characterization

The proposed VNP processor can be configured for any number of inputs (micro-parallelism), as well as, the total number of inputs required for a variable node processing. Various instances of the proposed VNP are characterized using the Cadence CAD tool flow for the TSMC 90nm LPHP standard cell library. The characterization results for VNPs of various degrees (d_v) are shown in Table 4.2.

Micro-architecture Level Tradeoffs

Some important tradeoffs at the level of micro-architecture using the characterization data (see Table 4.1 and 4.2) are discussed below. It is quite obvious that increase in the processing parallelism causes an increase in area. However, the increase in area is not linear with the parallelism increase, which are important from the viewpoint of optimization. An interesting specific case is the fully-parallel micro-architecture of CNP with $d_c = 8$, as the total area of the processor is almost the same as its counterpart with a lower micro-architecture parallelism

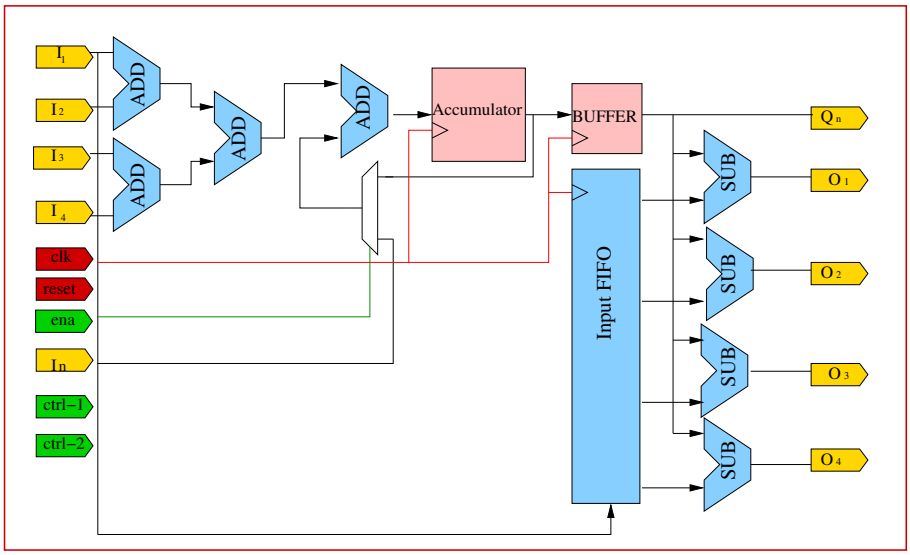

Figure 4.24: A particular instance of the generic VNP with 4-inputs and 4-outputs

of 4 (see Table 4.17). This is due to the fact that the storage block (temporary registers and buffer) are not required in case of the fully-parallel processor. It is also interesting to note that the serial micro-architecture, although, has a low critical path delay and area, would require more clock cycles for a single check or variable node computation. However, the same computation can be performed in a single cycle by the fully-parallel architecture albeit with a longer cycle delay and larger area compared to the serial architecture. The total computation time (T_t) that take into account both the influence of the computation clock cycles (cycle count) and critical path delay can be used as a useful performance measure. It can be computed for both kind of processors using the following equation:

$$T_t = \begin{cases} d_c/P_{mic} \times D_{P_{mic}}, \ P_{mic} = \{1, ..., d_c\} & \text{for} \quad \text{CNP} \\ d_v/P_{mic} \times D_{P_{mic}}, \ P_{mic} = \{1, ..., d_v\} & \text{for} \quad \text{VNP} \end{cases} \tag{4.15}$$

Based on the above equation, $T_t = 8.392\ ns$ (=4×2.098) and $A = 0.0021\ mm^2$ for the serial VNP micro-architecture ($P_{mic} = 1$). However, for the fully-parallel VNP micro-architecture ($P_{mic} = 4$), $T_t = 3.079\ ns$ (=1×3.079) and $A = 0.0033\, mm^2$. Thus, the gain in performance is *2.72-times* with area overhead *1.57-times* for the fully-parallel micro-architecture compared to the serial micro-architecture. The partially-parallel architecture ($P_{mic} = 2$) has a performance penalty of only *0.73-times* and area saving of *1.12-times* compared to the fully-parallel micro-architecture. In comparison to the fully-serial micro-architecture,

Table 4.2: VNP characterization results for different micro-architectures parallelism in TSMC 90nm LPHP Standard Cell Library

Parallelism (P_{mic})	d_c	Optimized for Area*			Optimized for Speed		
		Area (mm^2)	Delay (ns)	Clock Cycles	Area (mm^2)	Delay (ns)	Clock Cycles
1	4	0.002118	2.098	4	0.003290	0.847	4
2		0.002944	2.101	2	0.005089	0.921	2
4		0.003313	3.079	1	0.011673	1.366	1
1	8	0.002902	2.098	8	0.003995	0.847	8
2		0.003728	2.101	4	0.006376	0.958	4
4		0.005380	2.419	2	0.008899	1.437	2
8		0.006603	3.931	1	0.017568	2.724	1
1	12	0.003686	2.098	12	0.004778	0.847	12
2		0.004512	2.101	6	0.007161	0.958	6
3		0.004121	2.253	4	0.007046	1.210	4
4		0.006164	2.419	3	0.009683	1.437	3
6		0.007816	2.768	2	0.012683	1.908	2
12		0.009899	4.708	1	0.024108	3.616	1

*Area of a NAND Gate in CMOS 90nm = 2.1168 um^2

it has an area penalty of *1.38-times*, however, a performance gain as high as *1.99-times*. Thus, the partially-parallel architecture shows good tradeoffs both from the viewpoint of performance, as well as area. Summing up, when the performance is a dominating design objective then the parallel micro-architecture performs the best provided that the area constraint is satisfied. When the area is the main design objective then the serial micro-architecture is the best, under the condition that the performance constraint is satisfied. Also, if the clock speed is a hard constraint, then the fully-parallel may not satisfy the clock speed constraint due to its long critical path delay.

However, these are exclusively the local design decisions, only from the point of view of the processors micro-architecture out of the context of the macro-architecture decisions. Moreover, the memory and communication has to be considered and this may change the micro-architecture decisions that were taken locally. For example, for the fully-parallel micro-architecture, a distributed memory architecture and the corresponding interconnect network have to be provided to adequately support the processing parallelism. Therefore, it is very probable that the distributed memory and interconnect structure will result in a substantial area overhead and delay. For the fast serial micro-architecture, it is very prob-

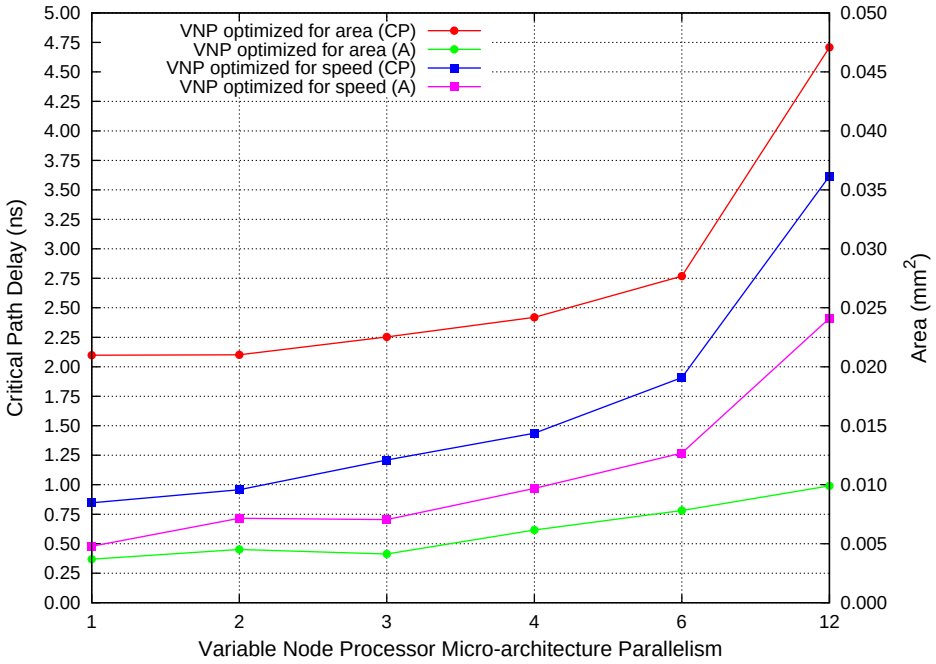

Figure 4.25: Area and critical path (CP) delay tradeoffs for different micro-parallelism of VNP: Case 1: processor optimized for area (RCA-based adder tree realization), Case 2: processor optimized for speed (CSA-based adder tree realization)

able that the delay of the centralized memory becomes much higher than of the processing element. From the above, it should be clear that all the architecture decisions have to be taken in combination. This reconfirms once more that such combined decisions require an adequate DSE framework, which is provided by the proposed architecture design methodology.

4.4.2 Memory Elements Design and Characterization

There are four different kinds of memories involved in the LDPC decoder design. These memories also differ from each other in their required size, number of banks, word lengths and port configurations. The memories are *check node memories* (M_{cv}) to store the C_{mn} messages, *variable node memories* (M_{vc}) to store V_{mn} messages, *channel memories* (M_{ch}) to store the channel messages (I_n) and *hard decision memories* (M_{HD}) to store the hard decision messages (V_{HD}). The size, word length, ports and organization (vector/multi-port/multi-bank) for each type of memory is decided during the DSE for a particular LDPC code and

its parametric constraints and objectives. Also, from the implementation point of view, there are two choices. The memory can be implemented using the fast flip-flop (FF) based registers in logic cells or as an embedded SRAM in memory cells. For the embedded SRAM characterization, the HP CACTI 5.3 tool is used to generate different memory configurations. Table 4.3 shows the characterization results for some of the many memory configurations both for the FF-based and SRAM memories, and to be used in the memory architecture exploration and synthesis.

Table 4.3: Flip-Flop based and SRAM memory characterization results. FF-based memories are characterized in TSMC 90nm LPHP Standard Cell Library while the cell-based SRAM memories using HP CACTI 5.3 tool

Memory Type	Parameters (1R/1W Port)	Memories with different Depths and Width=168			
		16	32	64	128
FF-based	Area (mm^2)	0.057320	0.113030	0.224011	0.445970
	Delay (ns)	1.343	1.620	1.456	1.791
SRAM	Area (mm^2)	0.092939	0.102027	0.120094	0.155192
	Delay (ns)	0.526153	0.606462	0.630445	0.748682

4.4.3 Communication Elements Design and Characterization

In LDPC decoding, there are two kinds of processors involved namely, the CNPs and VNPs and their corresponding memories M_{cv} and M_{vc}. Different kinds of switches are needed to communicate the processors (CNPs, VNPs) and memories (M_{cv}, M_{vc}). To efficiently perform the DSE, several kinds of switches (single-stage and multi-stage) and cyclic shifters (single stage and multi-stage) are designed and modeled as Verilog templates with diverse parameters and parameter ranges. Among others, the two important parameters are the number of input and output ports and the port data width.

For the communication elements characterization, the Cadence CAD tool flow was used, involving the Cadence RTL compiler for synthesis and Cadence Encounter for floor planning, placement and routing. For different switches and their configurations, the characterization was performed automatically using a configuration and characterization tool, which is a part of the architecture exploration framework.

The single-stage and multi-stage switches are differently organized in terms of stages, which in turn results in different values for physical switch parameters, such as area, delay, etc. For example, a cyclic shifter can be realized using a single-stage multi-input multiplexer (MUX) (see Figure 4.26) or by multiple simple

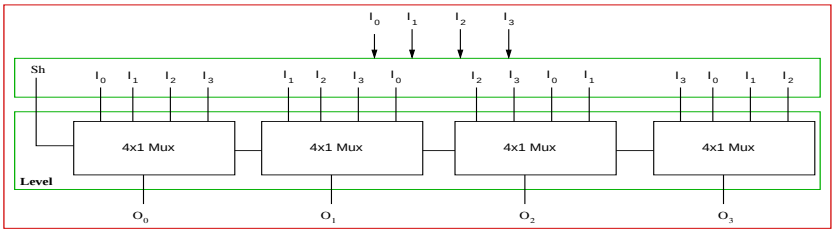

Figure 4.26: An example 4×4 single stage cyclic shifter

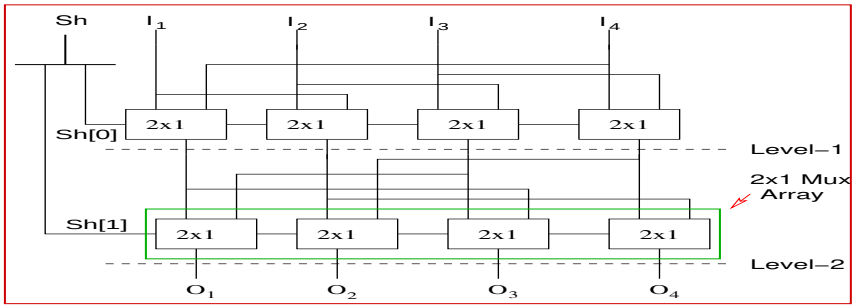

Figure 4.27: An example 4×4 multi-stage cyclic shifter

2×1 multiplexers cascaded in multiple stages (log-shifters) (see Figure 4.27). As concerned resources, the direct implementation of an $N \times N$ shifter would require in total N multiplexers each of size $N \times 1$, while $N log_2 N$ multiplexers each of size *2×1* for a multi-stage shifter. The delay in case of the direct implementation is that of a single $N \times 1$ input multiplexers, while for the multi-stage shifter, the delay would be $log_2 N$, i.e. the number of stages. Similarly, a global switch that provides all the inputs/outputs permutations can be realized using a Benes network of $N/2(2N log_2 - 1)$ *2×2* crossbar switches (see Figure 4.29) or a direct implementation using N $N \times 1$ multiplexers (see Figure 4.28). In case of the direct implementation, the delay is of a single $N \times 1$ multiplexer, while for the multi-stage shifter, the delay would be of $2log_2 N - 1$ *2×2* crossbar switches, i.e. the number of stages. This simple delay evaluation holds for small switch sizes, but shows a huge deviation for large switches after they are physically placed and routed. This once more reconfirms that the actual architectural element characterization and using the data from the actual characterization for the DSE are really necessary for taking the adequate design decisions. The characterization results for the single-stage and multi-stage cyclic shifters with different number of ports and data width configurations are shown in Table 4.4.

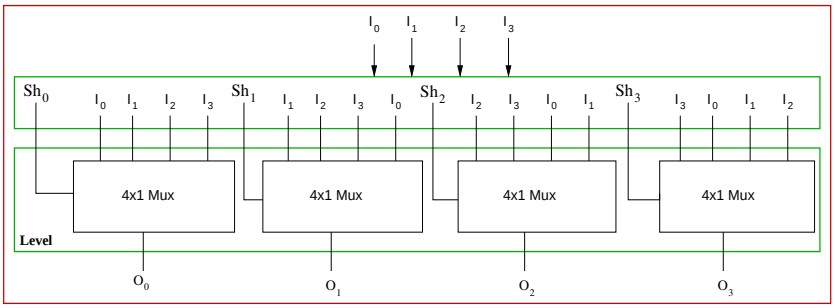

Figure 4.28: An example 4×4 single stage permuter

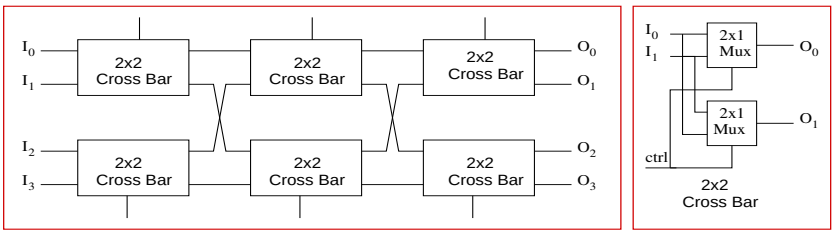

Figure 4.29: An example 4×4 multi-stage permuter (Benes network)

4.4.4 Sequencer/Controller Design for LDPC decoder

Two phase message passing (TPMP) LDPC decoding algorithm consists of four main steps: initialization, check node computation, variable node computation and decoding termination. The steps are well-described in Section 4.2 of this chapter. The order in which these steps are performed is shown in Figure 4.2. The controller for the LDPC decoder consists of a Finite State Machine (FSM) controller and a set of counters that together implement the control mechanism for the above four steps, as shown in Figure 4.30. FSM is the main sequencer/controller. Its state diagram and operations are described below.

The FSM involves four main stages corresponding to the main decoding steps: initialization stage, check node computation stage, variable node computation stage and decoding termination stage, as shown in Figure 4.31. Each main stage consists of a number of internal states that implement the required control related to that stage. Initialization stage consist of two states, the IDLE and INIT state, as shown in golden in the state diagram of Figure 4.31. Upon the assertion of initialization signal (*Start*), the FSM jumps to the initialization state (INIT). In the INIT state the channel memory is initialized with the received frame (codeword) to be decoded. In the same state the initialization of the variable node memories M_{vc} that store the variable node messages is also performed (see details in Section

Table 4.4: Single-stage and Multi-stage cyclic shifters characterization results in TSMC 90nm LPHP Standard Cell Library

Shifter Type	W	Parameters	Shifters with different I/O ports and Width=8					
			2	4	8	16	32	64
Single Stage	8	Area (mm^2)	0.000233	0.001062	0.004371	0.017441	0.071126	0.27836
		Delay (ns)	0.122	0.508	0.559	0.880	0.953	0.945
Multi Stage	8	Area (mm^2)	0.000233	0.000839	0.002427	0.00635	0.015697	0.037384
		Delay (ns)	0.122	0.276	0.429	0.582	0.735	0.888
Single Stage	168	Area (mm^2)	0.00485	0.020049	0.078298	0.313072	1.274514	5.09458
		Delay (ns)	0.122	0.869	0.794	1.239	1.230	1.350
Multi Stage	168	Area (mm^2)	0.00485	0.017667	0.05117	0.133255	0.331488	0.789721
		Delay (ns)	0.122	0.930	1.086	1.242	1.514	1.67

4.2 of this chapter). The track of the number of samples of the received frame is kept in a counter. The channel memory M_{ch} is implemented in the ping-pong fashion using double buffering technique. The double buffering technique works as follow. It employs two channel memories instead of one. One of the channel memories (M_{ch}) is used in decoding of the current frame, while the other is used for storing of the next incoming frame. At the start of next frame decoding, they switch their functionality (ping-pong). This way it reduces the decoding latency related to the input/output (I/O) interface. The same technique is used for the hard decision memory M_{hd} that stores the decoded codewords.

After the initialization, the FSM transits to the check node computation stage upon the assertion of *InitDone* signal. The check node computation stage consist of three states, as shown in pink in Figure 4.31. The CNP can perform the computation in parallel for two consecutive nodes. Internally the check node computation is organized in two phases (see details in Section 4.4.1 for the check node processor micro-architecture). While processing the inputs for the current CN, it also processes the outputs for the previous CN. Therefore, the CN computation is divided into three states, CN_RD, CN_RDWR and CN_WR state. This division is required to generate specific control signals for each of the CN computation phases. In CN_RD state only the first phase of CNP is performed (first CN and no previous CN), in the CN_RDWR both CNP phases are performed for different nodes (current and previous CN), while in the CN_WR state only the second phase of CNP is performed (last CN). More specifically, in the CN_RD all the inputs of the first check node (CN) are processed as a first phase of the check node processor. Then, the FSM jumps to the state CN_RDWR by the assertion of the signal *FCN*. In this state, the FSM enables the processing of inputs for the current CN and outputs for the previous CN in parallel. The necessary control signals are generated by the FSM to control the processors and the corresponding switches and memories that are active during this state. The FSM output signals,

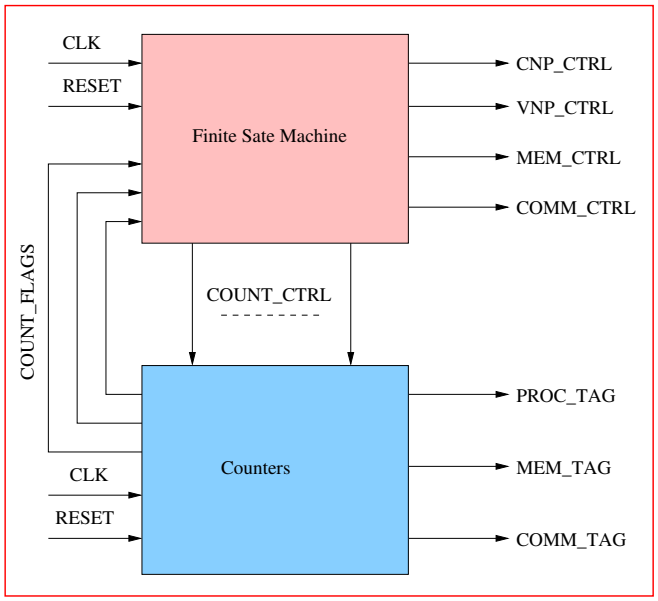

Figure 4.30: Main Controller/Sequencer top-level architecture for LDPC decoding

however, are not shown in the state diagram[5] of Figure 4.31. The FSM remains in this state and controls the CN computations (input/output phases) for all the nodes till the last CN input phase is performed. To process the last CN output phase, the FSM jumps to the CN_WR state by asserting the signal ACN true. In this state, the FSM enables the necessary control signals to perform only the second phase of the CN and terminates the CN computation stage. An example of a CNP control signal is *proc_cnp_ctrl_load*. When this control signal is asserted, it initializes the processor with the maximum values in order to find the min and min_2 among all the CN inputs. This is done at the beginning of each of the CN computations.

After processing the last CN output phase, the FSM transits to the VN_RD computation stage, as shown in blue in the state diagram of Figure 4.31. The VN computation is divided into three states, similarly to the check node computation stage, that enables the processing of two variable nodes in parallel. The VN computations are different than the CN computations and require different control signals. The control signals ensure the proper timing of the VNPs, and the corresponding switches and memories involved in the VN computations. For instance, consider a control signal *proc_vnp_ctrl_ch_in*. When this control signal is asserted, it adds up the received channel value to the running sum at the beginning of every VN computation. The VN computations proceed in a way similar

[5]Details of the FSM output control signals can be found in Appendix-B

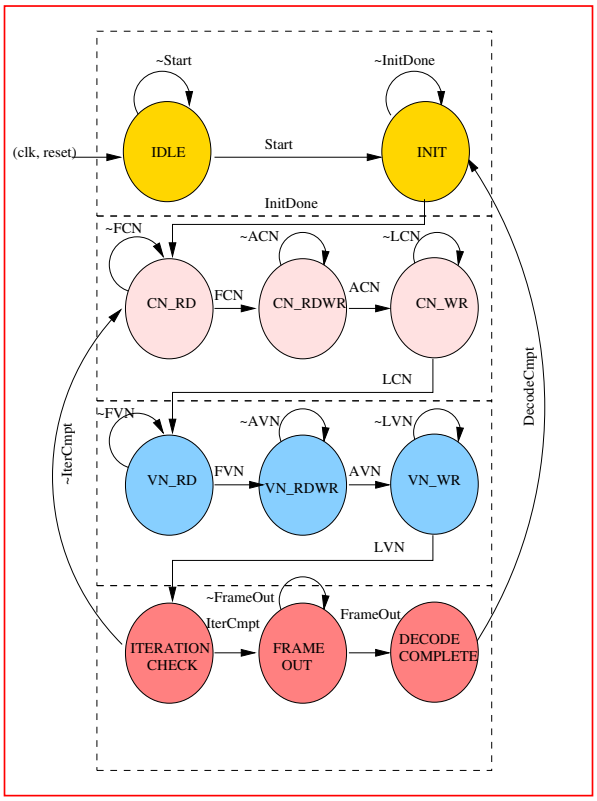

Figure 4.31: Finite state machine (FSM) of controller for LDPC decoder

to the CN stage until they reach the VN_WR state. In this state, when the last VN output phase is finished, the FSM goes to the ITER_CHECK state, which is a part of the termination stage, as shown in red in Figure 4.31. In this state, the FSM checks the number of iterations. If the maximum number of iterations is reached for a frame or a codeword is correctly decoded, the FSM jumps to the frame output state FRAME_OUT. Otherwise, the FSM continues with the next iteration on the current frame by jumping to the first state, i.e. CN_RD, of the check node computation stage. This is done by the assertion of *IterCmpt* signal as false. In the FRAME_OUT state, the FSM output the current frame in a ping-pong fashion, as discussed earlier for M_{ch} memory. If there are more frames to decode the FSM goes to INIT state, otherwise the FSM remain in the decode complete state DECODE_CMPT.

Several counters are required to keep track of various necessary counts including: the number of iterations, the number of messages per check or variable node, the total number of nodes that are processed. Instead of using multiple

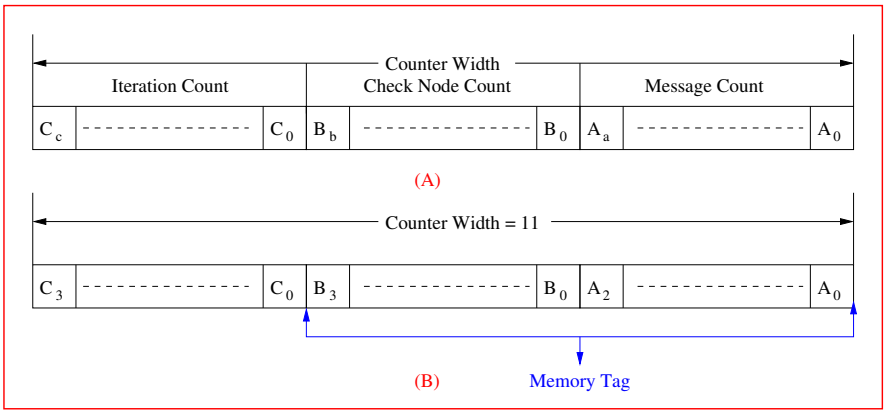

Figure 4.32: (A) Single Combined Counter for different counts purposes (B) Example of counter for rate-1/2 672-bit IEEE 802.15.3c LDPC code with processing parallelism P(1, 21)

counters, a single counter can be used with its various bits assigned to count different values. One counter organization is shown in the Figure 4.32. It splits the bits of a single counter into three parts unlike considering three independent counters connected in cascade. The first part $(A_n.....A_0)$ corresponds to messages per check or variable node, the second part $(B_n.....B_0)$ to the total number of nodes involved in a particular LDPC code and the last part $(C_n.....C_0)$ for the number of decoding iterations. For instance, consider an architecture instance for the rate-1/2 672-bit IEEE 802.15.3c LDPC code with processing parallelism P(1, 21). The CN computation in this case requires a maximum of 3-bits to count the number of messages for a single check node (8-inputs per CN), 4-bits for counting the total number of check nodes (16 macro-rows) and 4-bits to count the number of iterations (assuming the number of iterations $\leq$ 16), as shown in Figure 4.32. The counter flags are then used to to generate different control signals $(FCN, ACN, LCN, FVN, AVN, LVN, IterCmpt, DecodeCmpt)$ for different state transitions in the main FSM. The advantage of using a single counter is two fold: first, it reduces the complexity in terms of resources of the controller, and secondly it saves the time in hand-shaking with the main FSM that would be required in case of several individual counters.

Moreover, the counter values are used as tags for generating physical addresses for various memories $(M_{ch}, M_{cv}, M_{vc}, M_{hd})$. The physical memory addresses corresponding to the tags are stored in various read-only-memories (ROMs). These addresses are quite random, but are computed off-line at design-time during the memory architecture M_{arch} design, and are not computed at run-time. At run-time, the earlier computed and stored addresses are only used as physical addresses for different memories. For instance, consider an architecture instance for

the rate-1/2 672-bit IEEE 802.15.3c code with processing parallelism P(1, 21). For simplicity, consider the case of a check node message vector memory M_{cv} assuming that the architecture P(1, 21) employs a single M_{cv}. The tags for M_{cv} is generated in the following manner. The total number of check node messages to be stored in this memory is 108 and the maximum number of inputs for a check node is 8. Therefore, the first 3-bits of the counter (check node counter) are used to tag the individual messages of a node, while the other 4-bits for counting the number of check nodes. They together form the memory tags for the ROMs that store the physical addresses for the M_{cv}, as shown in Figure 4.32.

The same counters are also used as tags to generate physical routing signals for various switches and shifters. The routing addresses are of two types: local routing and global routing. Both the local and global routing addresses are computed during the communication architecture C_{arch} design. Moreover, the local routing addresses (values) have a one-to-one correspondence with the values of the non-zero entries of the parity check matrix (PCM). The various routing address are computed off-line at design-time during the C_{arch} design, and are stored in different ROMs for each of the communication segments. The off-line computed addresses, which are stored in ROMs, are then used at run-time for controlling the communication (interconnection) among different memories and processors.

Since the processors implement relatively simple operations, their control is also simple and involves just a few control signals (e.g. 5-control signals both in case of CNP and VNP). Therefore, instead of designing separate FSM controllers for individual processors, a centralized FSM controller is designed, which is much efficient in terms of resources. However, for the cases when the individual processors implement complex and divergent operations, such a centralized controller is not only difficult to design but also would be inefficient both from the point of view of resources and timing.

4.5 Application of the Design Space Exploration and Synthesis Approach to the LDPC Decoders

In previous sections, the design of the generic top-level architecture template and its generic modules are explained for the LDPC decoders, which is one of the most important, difficult and time-consuming steps of the proposed architecture design methodology. In this section, it is described that how the architecture design space exploration and synthesis (DSE&S) approach for multi-processor accelerators proposed in Chapter 3 is used for the design of LDPC decoders. The focus will be on the massively parallel multi-processor LDPC accelerators required for the high-end applications. As briefly introduced in Chapter 3, the DSE is performed for a given LDPC code and related set of behavioral and parametric requirements through the instantiation of the generic architecture templates, computation process scheduling and mapping on the architecture templates, and

analysis and selection of the constructed alternative architectures.

For LDPC decoders, the required accelerator quality is defined by a particular LDPC code, its associated parity check matrix (PCM), its decoding algorithm together with decoding parameters, and the parametric constraints, objectives and tradeoff preferences. The hard constraints are usually on the throughput (T_{cstr}) and the clock frequency (F_{cstr}). The optimization objectives are related to the circuit area (A_{obj}) and power consumption (P_{obj}). Also, the required tradeoff between the area and power consumption should be specified. The PCM (Tanner graph) of a given LDPC code is a fully-parallel abstract behavioral specification of the LDPC code. A given PCM contains abstract information about all kinds of information processing elements and corresponding architecture elements that are relevant from the viewpoint of architecture exploration and synthesis. In particular, it defines the different kinds of processing and corresponding processors (although does not give their precise processor definition) and their communication with each other. The non-zero PCM entries represent the memory requirements.

Algorithm 2 : Design space exploration (DSE) and synthesis algorithm for LDPC decoders

1: ***Input*** $\rightarrow$ PCM, $Throughput$ (T_{cstr}), $Frequency$ (F_{cstr})
2: ***Input*** $\rightarrow$ *set Aspiration Point* (AP) *for Area* (A_{obj}) and $Power$ (P_{obj})
3: ***Output*** $\leftarrow$ $S_{Arch} = \{Arch_1, ..., Arch_n\}$ // set of architectures that satisfy the hard constraints of the clock speed and throughput and optimize the tradeoffs among the area and power
4: ***Initialize:*** $S_{Arch} = \emptyset$, // where $Arch_{inst}$ = {processors, memory, communication}
5: $SED\_Mic(P_{mic}, F_{cstr})$ // search, evaluate and decide P_{mic}
6: $SED\_Mac(P_{mac}, T_{cstr})$ // search, evaluate and decide P_{mac}
7: $SED\_MemComm(P_{mac}, P_{mic})$ // search, evaluate and decide compatible M_{arch} and C_{arch}
8: **for all** ($Arch_{inst} \in S_{Arch}$ *such that* $isvalid[Arch_{inst}] == True$) **do**
9: ***select:*** $Arch_{inst} \rightarrow$ opt(A_{obj}, P_{obj}) // optimally satisfy A_{obj} and P_{obj}
10: ***synthesize:*** $Arch_{inst}$ // synthesize the architecture
11: **end for**
12: **for all** ($Arch_{inst} \in S_{Arch}$ *such that* $isvalid[Arch_{inst}] == False$) **do**
13: *modify (improve) the architecture templates or lower the requirements*
14: **end for**

The DSE algorithm organizes the design decision (state) space for the design-decision-making at two levels. The top-level breaks down the decision space into a number of main distinct issues in a tree like structure, as described in Chapter 3 (see Algorithm 2). The lower-level represents the actions (operators) that are performed at different main stages (corresponds to the nodes in the search tree at a particular level) during the search for the solutions (see Algorithms 3, 4, 5).

The DSE algorithm takes as it inputs the PCM (behavioral specification), as well as the required clock speed and throughput as the hard constraints, and the area and power consumption minimization as the optimization objectives (see algorithm 2). The clock speed and throughput constraints, and the optimization objectives of power and area represent the required properties of solutions in the

objective space. The decision space involves the decisions about the CNP and VNP processor's micro- and macro-architectures, as well as, the decisions about the different memories needed (M_{cv}, M_{vc}, M_{ch}, M_{HD}) and the decisions about the communication network among the processors (CNP, VNP) and memories (M_{cv}, M_{vc}, M_{ch}, M_{HD}). The constraints, objectives, and their required tradeoffs and the design decision vector define together the multi-objective optimization (decision) problem to be solved through the DSE. Our DSE tool, called Multi-Processor Accelerator Explorer (MPA-Explorer), makes the decisions about the design parameters included in the decision vector, when observing the hard constraints and optimization objectives. These decisions influence the constraints and objectives through influencing the accelerator throughput, clock frequency, area and power consumption.

The clock speed and the throughput are directly influenced and decided by the processing parallelism. i.e. the type and number of processors. Therefore, the architecture exploration starts with the decision of the processing parallelism, and specifically, the selection of the micro-architecture (P_{mic}) parallelism for each of the computation tasks, as represented by the procedure "SED_Mic" (see line 5 of algorithm 2). SED (*search*, *evaluate* and *decide*) reflects the three steps that are performed at each stage of the design decision making. The other reasons for initializing the search with the micro-architecture decisions are much fewer legal values for P_{mic} compared to P_{mac}. This decision variable ordering is called "minimum-remaining-values (MRV) heuristic". It makes possible pruning of the search tree by eliminating a large number of branches at the second step, i.e. macro-architecture step (usually more possibilities for macro-architectures than micro-architectures). This is due to the fact that when building up a solution, if any constraint is not satisfied, one can immediately reject all possible ways of extending the current partial solution.

Algorithm 3 : Micro-architecture exploration for LDPC decoder

1: ***Procedure:*** $SED\_Mic(P_{mic}, F_{cstr})$
2: ***Input*** $\rightarrow PCM$, *set of Tasks*, F_{cstr}, $S_{Arch} = \emptyset$
3: ***Output*** $\leftarrow S_{Arch} = \{set\ of\ P_{mic}\}$
4: **for all** $i \in P_{mic}$ for CNP **do**
5: **if** $(P_{mic}[i] \rightarrow F_{del} \geq F_{cstr})$ **then**
6: ***select:*** $P_{mic}[i]$ // select the micro-architecture that satisfies the clock constraint
7: ***create and update:*** $Arch_{inst} \rightarrow P_{mic}[i]$ // create the architecture instance $Arch_i$ and update it by adding the micro-architecture P_{mic} information
8: $isvalid[Arch_{inst}] = True$ // Tag the architecture in the architecture database as *valid*
9: $i := i + 1$ // Test another P_{mic} instance
10: **else**
11: go to step 14 // relation holds: $x^{(1)}(i) < F_{cstr} \rightarrow x^{(2)}(i') < F_{cstr}, s.t.\ (i \neq i') \wedge (i' > i)$
12: **end if**
13: **end for**// Loop for the micro-architectures that satisfy the clock constraint
14: ***repeat*** *steps 4-13 for* VNP

The details of the procedure "SED_Mic" are given in Algorithm 3. Various

micro-architectures for the CNP and VNP are searched, evaluated and among them those are selected that satisfy the F_{cstr}. In this step, some of the micro-architectures with the higher parallelism can be excluded from further consideration due to a long critical path delay, and consequently low maximum delivered clock frequency (F_{del}) (see line 4-13 of algorithm 3). For CNP, $P_{mic} = \{1, ..., d_c\}$ and for VNP, $P_{mic} = \{1, ..., d_v\}$, where d_c and d_v represent the maximum CN and VN degrees of a given LDPC code, respectively. For instance, for the rate-7/8 672-bit IEEE 802.15.3c code, $P_{mic} = \{1, ..., 32\}$ for CNP and $P_{mic} = \{1, ..., 4\}$ for VNP. It should be noted that the mapping possibilities in the DSE are limited to the pre-designed generic processor architecture templates.

Algorithm 4 : Macro-architecture exploration for LDPC decoders

1: ***Procedure:*** $SED\_Mac(P_{mac}, T_{cstr})$
2: ***Input*** $\rightarrow PCM$, *set of* P_{mic} *for each* $Task$, T_{cstr}, S_{Arch}
3: ***Output*** $\leftarrow S_{Arch} = \{set\ of\ P_{mac}\}$
4: **for all** $Arch_{inst} \rightarrow P_{mic} \in S_{Arch}$ for CNP **do**
5: **for all** $j \in P_{mac}$ for CNP **do**
6: ***determine:*** P_{mic} & P_{mac} for VNP // Combined micro- and macro-architecture parallelism for VNP must match the CNP
7: ***perform and evaluate:*** *scheduling&mapping*: $Arch_{inst} \rightarrow$ {CNP, VNP} // compute the schedule length in clock cycles
8: ***determine:*** T_{del} for $Arch_{inst} \rightarrow$ {CNP, VNP}
9: **if** $(Arch_{inst} \rightarrow T_{del} \geq T_{cstr})$ **then**
10: ***select:*** $P_{mac}[j]$ // P_{mac} for both the CNP and VNP that satisfy the T_{cstr}
11: ***update:*** $Arch_{inst} \rightarrow$ [CNP $\rightarrow P_{mac}$, VNP $\rightarrow P_{mac}$] // update the architecture $Arch_{inst}$ by adding P_{mac} information for both the CNP and VNP
12: go to step 5 // no need to test further P_{mac} {dominated architecture instances}
13: **end if**
14: **if** $((Arch_{inst} \rightarrow T_{del} < T_{cstr}) \wedge (Arch_{inst} \rightarrow P_{mac} == max(P_{mac}))$ **then**
15: $isvalid[Arch_{inst}] = False$ // Tag the $Arch_{inst}$ in S_{Arch} as invalid
16: go to step 5 // do not satisfy the throughput with maximum macro-parallelism
17: **else**
18: $j := j + 1$ // Test another P_{mac} instance
19: **end if**
20: **end for**
21: **end for**// Loop for the micro-/macro-architectures combinations that satisfy the T_{cstr}

After selecting the set of micro-architectures (P_{mic}) to be further considered, the decisions about the macro-architectures (P_{mac}), i.e. the number of processors, are made for both the CNP and VNP processors, as represented by the procedure "SED_Mac" (see line 6 of algorithm 2). For CNP, P_{mac} = {1,.....,CNs}, and for VNP, P_{mac} = {1,.....,VNs}, where CNs and VNs represent the total number of check and variable nodes in the PCM of a given LDPC code, respectively. For instance, for the rate-1/2 672-bit LDPC code, $P_{mac} = \{1, ..., 336\}$ for CNP and P_{mac} = {1,.....,672} for VNP. While the number of VNs and CNs are 64800 and 32400, respectively, for the rate-1/2 64800-bit DVB-S2 LDPC code.

The search, evaluation and selection of the macro-architectures for CNP and

VNP are based on the throughput constraint (T_{cstr}) and the dominance relation. It is performed as follows. First, P_{mac} is determined for CNP for each of the P_{mic} selected in the previous step. It should be noted that the search for the macro-architectures progresses in the order of increasing values of the decision variable, i.e. P_{mac}. Another search strategy based on the order of the values of the decision variable can also be devised that may impact the efficiency of the search. After making a decision on the value of P_{mac} for the CNP, the P_{mac} for VNP is determined in such a way that the combined processing parallelism (P_{mic}, P_{mac}) for VNP would be the same (balanced) as for the CNP (line 6 of Algorithm 4). The micro-/macro-architecture combination for both the CNP and VNP defines an architecture instance ($Arch_{inst}$). To check whether $Arch_{inst}$ satisfies the T_{cstr} or not, T_{del} is computed for the $Arch_{inst}$, which is then compared with the T_{cstr} (see line 5-13 of Algorithm 4). To compute the delivered throughput (T_{del}) for each of the micro- and macro-architecture (P_{mic}, P_{mac}) combination, the scheduling and mapping of the CN and VN computations are performed on the CNP and VNP processors, respectively (line 7 of Algorithm 4). The throughput delivered (T_{del}) in Mbps that reflects the performance metric for the LDPC decoding can be estimated based on the Message Passing (MP) algorithm using the following formula:

$$T_{Mbps} = \frac{R.N.F_{MHz}}{CCPI.I_{tot}} \tag{4.16}$$

where R stands for the code rate, N stands for the code length (size of data frame), I_{tot} stands for the total number of iterations required to decode a frame, F_{MHz} stands for the clock frequency measured in MHz and CCPI represents the number of clock cycles per iteration. In the above equation, all the parameters are known except the number of clock cycles per iteration (CCPI). The CCPI depends on the processing parallelism (micro- and macro-architecture decisions) and the scheduling of the CN and VN computations on each of the micro- and macro-architecture combinations. The computation of each kind of node can be performed in parallel. The CCPI is then the sum of the clock cycles required for CN and VN computations. This corresponds to the proposed relax scheduling (RS) technique. The computation scheduling and mapping in the LDPC decoding seem to be quite trivial, as there are only two kinds of computation processes, the check and variable node processing, that alternatively operate on each other data. However, since overlapping of the CN and VN computations is possible, it turns out to be a very complex scheduling problem. Nevertheless, we developed the overlapped scheduling algorithms for the overlapping of CN and VN computations[6], which corresponds to the proposed tight scheduling (TS) technique. Even if it is very well resolved, its (near-)optimum solution results in relatively low gains in performance, while very much increasing the communication network and memory complexity. The scheduling freedom can be better utilized for the reduction of memory and communication network complexity than to increasing

[6]See Appendix-1 on the overlapped scheduling algorithms for LDPC decoding

the computation speed. Knowing the schedule length (CCPI.I_{tot}), the throughput delivered (T_{del}) can be computed for each micro-/macro-architecture combination using equation 4.16. If $T_{del} \geq T_{cstr}$, then the $Arch_{inst}$ satisfies the throughput constraint (see lines 9-12 of Algorithm 4).

Since the search for the macro-architectures progresses in increasing order of the values of P_{mac} for a given P_{mic}, any higher value for P_{mac} above the value for the first found solution that satisfies the T_{cstr} would result in a dominated solution. Therefore, the search terminates after finding the first feasible solution for a given P_{mic}, and the search proceeds with another value of P_{mic} (see line 12 of Algorithm 4). This way, among all the possible P_{mac} for CNP and VNP processors only those are selected for further consideration that minimally satisfy T_{cstr}. The minimal satisfiability criteria for the macro-architecture (P_{mac}) is based on the dominance relation. This also much reduces the search complexity, as any higher value of P_{mac} above the one that minimally satisfy T_{cstr} would result in a dominated solution and is not considered in the next stage of exploration. It is also possible that for a given P_{mic} the T_{cstr} cannot be satisfied with any macro-architecture parallelism even exploiting the macro-architecture parallelism to the maximum. In this case, all the micro-/macro-architecture (P_{mic}, P_{mac}) combinations that corresponds to that particular P_{mic} are pruned away from the set of candidate architectures (S_{Arch}) (see lines 14-19 of Algorithm 4).

Algorithm 5 : Memory and communication architectures exploration for LDPC decoders

1: ***Procedure:*** $SED\_MemComm(P_{mic}, P_{mac})$
2: ***Input*** $\rightarrow PCM, \forall\ Tasks\ (P_{mic}, P_{mac})$, F_{cstr}, T_{cstr}, S_{Arch}
3: ***Output*** $\leftarrow S_{Arch} = set\ of\ \{M_{arch}, C_{arch}\}$
4: **for all** $Arch_{inst} \in S_{Arch}$ such that $isvalid[Arch_{inst}] = True$ **do**
5: ***determine:*** *required storage and data transfer bandwidth* $= P_{mic}.P_{mac}\ .\ b$
6: ***SED_Mem:*** *memory bandwidth using various memory architectures* M_{arch}
7: ***SED_Comm:*** *data transfer bandwidth using various communication architectures* C_{arch}
8: **for all** $M_{arch} \wedge C_{arch}$ **do**
9: ***determine:*** *the memory and communication delays for each* M_{arch} *and* C_{arch}
10: ***compute:*** $(Arch_{inst} \rightarrow F_{del}) \wedge (Arch_{inst} \rightarrow T_{del})$ // Take into account the M_{arch} and C_{arch} structures delays
11: **if** $(Arch_{inst} \rightarrow F_{del} > F_{cstr}) \wedge (Arch_{inst} \rightarrow T_{del} > T_{cst})$ **then**
12: ***update:*** $Arch_{inst} \rightarrow [M_{arch}, C_{arch}]$
13: **else**
14: $isvalid[Arch_{inst}] = False$
15: **end if**
16: **end for**
17: **for all** $Arch_{inst} \in S_{Arch}$ such that $isvalid[Arch_{inst}] = True$ **do**
18: ***select:*** M_{arch} and $C_{arch} \rightarrow min${Area *or* Power}
19: **end for**
20: **end for**// Loop for the memory and communication architecture exploration

In the second stage, the memory and communication architectures are decided for each of the constructed and selected candidate partial architectures

$Arch_{inst} \in S_{Arch}$ representing particular micro- and macro-architecture combinations (P_{mic}, P_{mac}), as represented by the procedure "SED_MemComm" (see line 7 of algorithm 2). The processors (CNP, VNP) with micro-parallelism P_{mic} have m data input ports and the same number of data output ports. They demand a compatible memory architecture with m input and m output ports. The required aggregate storage and data transfer bandwidth per clock cycle for a micro-/macro-architecture (P_{mic}, P_{mac}) combination can be expressed as:

$$bandwidth/cc = P_{mic}.P_{mac}.b \tag{4.17}$$

where b represents the bit width of data. The M_{arch} for LDPC decoders involves four different memories (M_{cv}, M_{vc}, M_{ch}, M_{HD}) as defined in the top-level generic architecture template. The M_{cv} and M_{vc} are shared between the CNP and VNP processors, while the M_{ch} and M_{HD} are used by the VNP processors for reading of channel data and writing of decoded messages, respectively. The CNP reads the check node data from the M_{cv} and after processing writes the result data to the M_{vc} memories. Similarly, the VNP reads the variable node data from the M_{vc} and after processing writes the result data to the M_{cv}. Therefore, M_{cv} must have sufficient read ports to satisfy the read request of CNPs and must have to enough data write ports to satisfy the write requests of VNPs. Similarly, M_{vc} must have enough data read ports to satisfy the read requests of VNPs and must have enough write ports to satisfy the write requests of CNPs. The required bandwidth is computed for each of the architecture instance ($Arch_{inst}$) based on the combined processing parallelism (P_{mic}, P_{mac}) (see line 5 of Algorithm 5).

To ensure the data storage and transfer bandwidth required by the CNP and VNP processors on low cost and satisfactory delays, different memory and communication architectures are searched, evaluated and selected during the DSE using the procedures "SED_Mem" and "SED_Comm", respectively (see lines 6-7 of Algorithm 5). The memory architectures include the simple multi-port vectorized memories and the complex multiple single/dual-port vectorized or multi-bank memories. The communication architectures include the simple homogenous fully-flat architectures and the complex heterogeneous hierarchical partitioned communication architectures. The memories can be realized as SRAMs or register-based memories, and the communication architectures using single-stage and multi-stage switches and shifters. The DSE algorithm explores and selects the most promising of the communication and memory architectures for a particular micro-/macro-architecture combination, while taking into account the design constraints and optimization objectives.

To satisfy the memory bandwidth required for the CNPs and VNPs and to avoid the memory access conflicts, data in each type of memory (M_{cv}, M_{vc}, M_{ch}, M_{HD}) is partitioned and stored in distributed vector memories. This task is performed by the "SED_Mem" procedure. The problems of data organization into vectors and the required number of shared vector memory tiles (partitions) are resolved together with the communication architecture design, when the flat

communication network is transformed into the hierarchical network. However, providing as many shared vector memory tiles (partitions) as the processing tiles would only partially solve the problem due to the possible memory access conflicts. Therefore, the data distribution and data mapping in the partitioned memories are performed with the aim to eliminate the memory access conflicts, as well as, to ensure that the communication strategies would be applicable. Based on the analysis of the PCM of a given LDPC code, the processing elements (VNP, CNP) are organized in a corresponding hierarchical way into several tiles (groups). The tiles are then structured into one global cluster or several global communication-free smaller clusters (if possible), and their respective data in memory tiles. This task is performed by the procedure 'SED_Comm". The tiles and clusters replace a flat communication network with several much smaller hierarchically organized autonomous communication networks.

The search, evaluation and selection of communication and memory architecture is made according to the following criteria. For a particular micro- and macro-architecture combination, only these communication and memory architectures are considered that satisfy the throughput (T_{cstr}) and clock speed (F_{cstr}) constraints. Therefore, the delivered throughput (T_{del}) and clock speed (F_{del}) are recomputed once more for each of the candidate architecture in S_{Arch}, while taking into account the communication and memory architecture delays (see lines 8-16 of Algorithm 5). From among these architectures, only such are further considered, which cause the lowest increase in the area or power (see lines 17-19 of Algorithm 5).

It should be noted that the communication or memory architectures do not cause extra clock cycles[7] for the communication or memory operations (load and store), as the architecture template performs the memory operations in parallel with the computations and the communication is realized using point-to-point (P2P) links and not using shared buses or Network-on-Chip (NoC). Moreover, the processors, memories and communication structures are synchronized on a single clock, i.e. they use the same system clock and not independent clocks. The clock-based synchronization ensures correct work of the overall platform (accelerator).

This way, several most promising accelerator architectures (S_{Arch}) are constructed taking into account the mutual tradeoffs among processors, memories and communication networks (see lines 8-10 of Algorithm 2). If none of the architectures satisfies the constraints, then the requirements can be lowered or the architecture template can be modified. Another iteration of DSE can then be initialized (see lines 11-14 of Algorithm 2). A more precise explanation of various parts of the architecture exploration and construction process is presented in the successive chapters. The various possible micro-/macro-architecture tradeoffs regarding processors will be explained in the next chapter using as an example the design of LDPC decoders for the IEEE 802.15.3c LDPC codes and their

[7]This would reduce the overall throughput by increasing the total number of clock cycles required for the LDPC decoding

performance demands. The particular strategies of the communication and memory architecture exploration and synthesis will be described in Chapter 6 of this thesis. Although various issues are discussed separately in different chapters to explain them better and highlight the significance of each issue, the actual DSE is performed in combination for all the design elements (processors, memories and interconnects).

4.6 Architecture Template Instantiation and Rapid Prototyping

One of the main parts of the architecture design methodology is the automatic architecture template instantiation. The DSE specifies a set of promising architectures for a particular LDPC code and parametric requirements of its application. Subsequently, some selected architectures have to be actually instantiated prototyped and/or produced through automatic instantiation of the parameterized templates that model the architectures for the selected parameter values. The following are some of the main parameters that are used in the automatic template instantiation process for a particular LDPC code decoder:

- the micro-architecture of the elementary processing units (specifically, the parallelism level of the processing units);
- the macro-architecture (specifically, the number of processing units to be used and assignment of CN/VN nodes to processing units);
- the number, size, structure and organization of memory modules;
- the kind, organization of and parameters of interconnect and switching resources among various memories and processors (e.g. specific instances of logarithmic or barrel shifters or permuter or Benes networks).

Based on the values of the above parameters, the architecture template instantiation engine creates the actual RTL-Level model of the whole architecture in Verilog HDL, including the corresponding controller. This simulatable and synthesizable Verilog model can be directly used for the architecture correctness verification and rapid prototyping. This way, the architecture can be further more accurately analyzed and characterized for performance, area and power consumption using the back-end SoC synthesis and place and route tools. Further, FPGA emulation of the proposed architecture can be performed. The actual implementation results can be used as a feedback to the DSE engine for further architecture optimizations. Other advantages are the acceleration of the overall architecture DSE and synthesis process, and the avoidance of the manual translation errors among various design representations. An example of the list of parameters for a particular architecture instance generated by our Multi-Processor Accelerator Explorer (MPA-Explorer) tool is shown in Table 4.5.

Table 4.5: MPA-Explorer Tool output file and the set of parameters decided during the design space exploration for an architecture instance of the rate-1/2 672-bit IEEE 802.15.3c LDPC code

Block Types	Parameter Values
Processors	PE_cnp 84 2 8 8
	PE_vnp 84 2 4 8
Memories	mem_cnp sramff 8 16 168 1 1 0
	mem_vnp sramff 8 16 168 1 1 0
	mem_ch sramff 4 8 168 1 1 0
	mem_hd sramff 4 8 21 1 1 0
Networks	sw.fr.memc.to.cnp L1 benes_sw 1 8 168 PP 0 0 0 0
	sw.fr.cnp.to.memv L1 benes_sw 1 8 168 PP 0 0 0 0
	sw.fr.memv.to.vnp L2 benes_sw 1 8 168 barrel_shifter 8 0 21 8
	sw.fr.vnp.to.memc L2 benes_sw 1 8 168 barrel_shifter 8 0 21 8

4.7 Method Correctness

The method and the corresponding Multi-Processor Accelerator Explorer (MPA-Explorer) tool produce the structural register-transfer-level (RTL) specification of the required architecture (platform instance). The platform instance RTL-level specification is mainly composed of RT level models of processors, memories, communication network among the processors and memories, I/O interface and a controller (see Table 4.5). The DSE framework decides on a particular architecture and the parameters of the architecture elements for a given set of requirements. The parameters contain all the information needed to uniquely determine the actual hardware structure of the multi-processor accelerator. The functional correctness of the constructed architecture is guaranteed by the method provided the individual architecture elements work correctly. The functional verification of various kinds of processors, memories and interconnect structures is performed using the RTL-level simulation during the development of the generic architecture elements.

The interface compatibility among the architectural elements is guaranteed by the method. The architecture elements have the same number of interface binary signals from both sides (bandwidth compatible). Two types of signals are distinguished for all the architecture elements, namely, the control and the data path signals. The data path signals are used for the dataflow among the modules, while the control signals are used for various control purposes depending on the architecture elements involved. For example, in case of a processor, the control signals are used to control its internal state behavior (if any). The internal states

are only present when it implements the same computations in serial or partially parallel fashion instead of in fully-parallel. For the communication network (composed of switches and shifters), the control signals are used to control the switch temporal interconnections among the different processors (CNPs and VNPs) and memories (M_{cv} and M_{vc}). The control signals for switches and shifters (represented by the values of the non-zero entries in the block-structured PCM) are stored in read-only-memories (ROMs). In case of memories, the control signals are used to control the read/write mode of memories and also to provide the necessary memory addresses. Since irregular memory accesses are involved due to the complex interrelationships among the data and the computing operations in LDPC decoding, the memory addresses are computed off-line for reading and writing data from and into the memories (M_{cv}, M_{vc}, M_{ch}, M_{HD}). Several ROMs are used to store the off-line computed addresses for reading and writing of data in memories. This way the processors, memories and the interconnect networks are properly controlled that guarantee the correct behavior of the whole platform. The memories can be implemented in two ways: either as logic cells (Flip-Flops) or the actual cell-based SRAM memories. For high processing parallelism, the memories are usually implemented in logic turning the whole architecture into a standard cell-based design.

The whole system implements the controller/data path paradigm of Glushkov, where controller is an FSM and the whole system works with one clock and is fully synchronous (clocked-mode). Correct work of a synchronous (clocked-mode) system is generated by a proper relation between the input signals to the flip-flops and clock signal of the flip-flops (the FF input signals must be stable during the setup, rise and hold time of given FFs). Analogous way, the signaling correctness for SRAM memories is guaranteed. The delays in all modules (and this way the total delays in all modules from FF-output to inputs) are precisely controlled (known) because they are actually measured during the characterization. The clock frequency (period) is decided based on the actual delay, and this way the correctness of this fully synchronous system is guaranteed. Moreover, the MPA-Explorer tool generates a test bench for the top-level architecture in Verilog HDL. The test bench is used to simulate and verify the top-level instantiated multi-processor accelerator architecture. This way the method and its pre-verified generic architecture elements and specifically its architecture DSE exploration and synthesis guarantee the correctness of the constructed architecture.

4.8 Conclusions

In this chapter, the implementation of the proposed architecture design methodology is described for the architecture design of LDPC decoder. The core activities of the architecture design method are realized through the development of the architecture template and its modules for the LDPC decoding applications, as well as, the DSE for various tradeoffs at the micro-/macro-architecture level. Applica-

tion of the DSE and synthesis approach to the LDPC decoder is discussed. Our DSE tool, called Multi-Processor Accelerator Explorer (MPA-Explorer), makes the decisions about the architecture elements to form the complete architecture taking into account the design constraints and objectives. These decisions influence the constraints and objectives through influencing the accelerator throughput, clock frequency, area and/or power consumption. The various possible micro-/macro-architecture tradeoffs regarding processors will be explained in the next chapter using as an example the design of LDPC decoders for the IEEE 802.15.3c LDPC codes and their performance demands. The particular strategies of the communication and memory architecture exploration and synthesis will be described in Chapter 6 of this thesis.

CHAPTER 5

Micro-/Macro-architecture Exploration

In the previous chapter, the implementation of the proposed design methodology is presented for the LDPC decoding applications. In this chapter, a part of the design methodology and design space exploration (DSE) framework is thoroughly discussed that addresses the combined micro-/macro-architecture parallelism exploration, the mutual tradeoffs between the micro- and macro-architecture, as well as, the influence of the processor architecture decisions on the memory and communication structures. This discussion is illustrated with experimental case studies related to the design of LDPC decoders for the newest communication system standards. For the combined micro-/macro-architecture tradeoffs exploration, the IEEE 802.15.3c LDPC decoders are used as a case study, while taking into account their parametric constraints and objective related to specific practical application cases.

The micro-architecture level exploration is related to the realization of various RTL-level multi-input multi-output (MIMO) operations involved in the algorithms of many highly-demanding applications. For example, the IEEE 802.15.3c standard specifies four different LDPC codes with variable nodes from a single input/output to a maximum of 4 inputs/outputs, while the check nodes from a minimum of 5 inputs/outputs to as high as 32 inputs/outputs. The implementation spectrum of these operations spans from the fully-serial to the fully-parallel with in between these two bounds a large number of partially-parallel architectures. The realization of the MIMO operations in fully-parallel or in fully-serial may not be satisfactory for different stringent design constraints and objectives, which necessitates a careful exploration of the micro-architecture level parallelism.

This is due to the fact that the fully-serial micro-architecture requires a large number of computation cycles, however, due to a long critical path delay in case of the fully-parallel micro-architecture, which negatively influence the performance.

At the macro-architecture level multiple such (partially-) parallel processors have to be considered to satisfy the ultra-high throughput requirements of many of the modern demanding applications. This is possible because these applications involve massive data parallelism or task-level functional parallelism. For example, the rate-1/2 672-bit IEEE 802.15.3c LDPC code consist of 672 variable nodes (VNs) and 336 check nodes (CNs) that correspond maximally to the same number of variable and check node processors, respectively, at the macro-architecture level. Consequently, depending on the actual performance requirements complex multi-processor accelerators can be build of the elementary processors or accelerators to satisfy the requirements. Moreover, for various performance levels, there are different tradeoffs among the resources at the micro- and macro-architecture level. For example, for the LDPC decoders, lower number of more parallel processors (CNPs and VNPs) can be required to reach a similar performance level as with more but less parallel processors.

Additionally, the memory and communication structures are strongly related to the parallelism at each of the two architecture levels, due to the complex inter-relationships between the data and computing operations. Although the results of the exploration of the micro-/macro-architecture are discussed in isolation from the corresponding memory and communication exploration just to get an insight into the kind of possible tradeoffs. However, the processor, memory and communication architectures are considered together during the actual design space exploration (DSE). The results of the experiments with various processing parallelism and its influence on the communication and memory architectures shows the dominating influence of the memory and communication on the overall system costs and performance. Moreover, the influence of the operation scheduling shows some important tradeoffs among the different types of schedules that must be taken into account and are discussed in a separate section of this chapter.

The rest of the chapter is organized as follows. Section 5.1 describes in detail the various tradeoffs possibilities both at the micro- and macro-architecture level. Section 5.2 introduces the IEEE 802.15.3c LDPC codes that are used for experimentation. Section 5.3 discusses the results of the experiments performed for various micro- and macro-architecture combinations and their mutual tradeoffs. The influence of increase in the processing parallelism on the memory and communication architectures is discussed in Section 5.4. Section 5.5 discusses the influence of the operation scheduling on the performance and on the complexity of the memory and communication architectures. Finally, Section 5.6 closes this chapter with some important conclusions.

5.1 Micro-/Macro-architecture Parallelism Exploration

As argued in the previous chapters, for an adequate quality accelerator a careful tradeoffs exploration and exploitation at the micro- and macro-architecture levels is unavoidable, specifically in relation to the parallelism at both architecture levels. Choosing the right number of processors for a given application with its requirements, i.e. the macro-architecture decision cannot be separated from the right processor type selection, i.e. the micro-architecture decision and the micro-architecture implementation choices, especially, for the kind of applications involving complex multi-input multi-output MIMO operations.

The memory and communication structure design further complicates the decision problem of the micro-/macro-architecture, especially, for the kind of applications involving complex interrelationships between the data and computing operations, as both the micro- and macro-architecture directly influence the memory and communication architecture design but in different ways and to different degrees. Therefore, complex tradeoffs have to resolved to find an adequate balance between the kind of processors and their macro-level organization, and the related memory and communication architectures, and specifically, between the amount of parallelism at the micro- and macro-architecture level and the memory and communication structure complexity. This task can effectively and efficiently be performed through an adequate DSE for a number of promising micro-/macro-architecture combinations and, finally, the selection of the best of them for an actual realization (see Figure 5.1).

The micro-architecture level decisions influence the system-level decision in several ways. For instance, increase of the micro-parallelism for a specific task can reduce the number of clock cycles to execute the task and this way improve the performance. However, it is accompanied by increase in the resources as well as the critical path delays (reduces the processor clock speed), which negatively influences the overall performance. This kind of tradeoffs exploration is specifically important for the kind of applications involving complex multi-input and multi-output (MIMO) operations. Implementation of these operations using different parallel micro-architectures influences the critical path delay, the number of clock cycles required and the amount of resulting hardware significantly, and this way has a complex influence on the performance, power consumption and area. Through the experimental results, it will be shown that neither the fully-parallel nor the fully-serial micro-architectures are adequate for the most high-end demanding applications. This results from these complex influences and tradeoffs, which necessitates partial parallelism exploration and exploitation at both architecture levels. Additionally, the clock speed might be a hard constraint, in that case many micro-architectures (specifically the more parallel ones) may not satisfy the clock speed constraint. To adequately make this kind of decisions, processors with different parallelism levels have to considered during the DSE. The promising micro-architectures that satisfy the hard constraints and other design

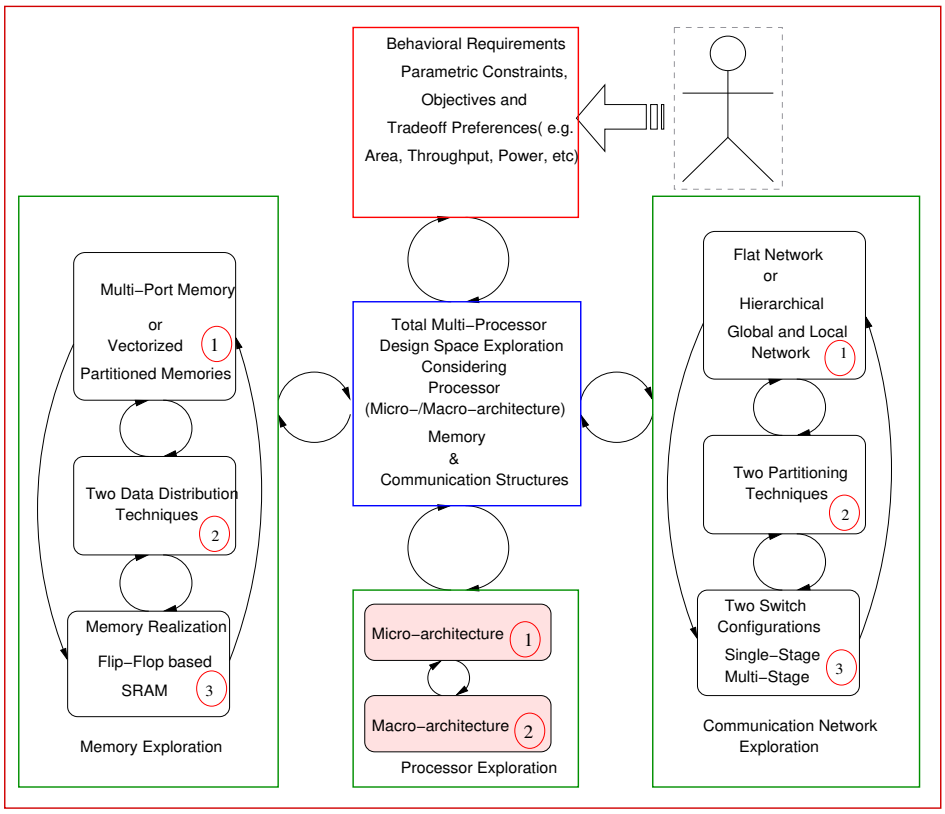

Figure 5.1: Exploration of the processor's micro- and macro-architectures parallelism in the total multi-processor DSE framework (highlighted in pink)

objectives have to be considered for further design exploration and evaluation (see Figure 5.1).

Another important tradeoff is the power against area for a certain performance level. Exploiting the higher degree of parallelism is a well-known method to trade power against area through operating more processors at a lower clock speed. The power/area tradeoff under a certain performance constraint can also be achieved by exploiting processors with different micro-parallelism, unlike the traditional power (optimization) reduction technique just by increasing the number of processors. In particular, processors with different micro-parallelism can be tried while keeping the macro-parallelism the same, but in general, various micro-/macro-parallelism combinations should be explored for this aim. This way substantially higher power reduction can be obtained at a lower area than what is offered by the traditional parallelism/power tradeoff approach. Please ob-

serve that high parallelism is accompanied by an increase in the static power that for the latest nanometer technologies has a growing contribution compared to the dynamic power. Thus, the joint consideration of the micro-/macro-architecture parallelism provides a new dimension for the power optimization and power/area tradeoffs under a given performance constraint. The proposed combined micro-/macro-architecture based power optimization approach is further elaborated by the experiments presented in the latter sections of this chapter.

An important benefit of a combined micro-/macro-architecture exploration is to restrain the over-dimensioning of the system regarding its cost under a performance constraint. Considering the macro-architecture or micro-architecture independently under a performance constraint may unnecessarily increase the system cost, because it is possible that the performance might be marginally not met. Increasing the macro-architecture parallelism in this situation would be a costly decision. As the micro-architecture design also influences both the performance and cost, but it does it with different tradeoffs between the two aspects than at the macro-architecture level. This can be resolved in a much better way through taking into account the micro-architecture enhancement that may restrain the system from over-dimensioning, while meeting the performance constraint with no or relatively lower increase in cost. This way the cost would be adequately tuned to the performance, which is only possible through the joint micro-/macro-architecture parallelism consideration. Thus, the joint consideration of the micro-/macro-architecture parallelism provides a novel way for controlling the system cost against over-dimensioning under a given performance constraint.

All the above mentioned tradeoffs will be elaborated through the design and analysis of various multi-processor accelerator architectures for LDPC decoding for the latest communication systems standards, when using the proposed generic architecture template and generic components library supported by the DSE framework. In the next section, the LDPC codes of the IEEE 802.15.3c standard [3] will be introduced that were used for the experimentation partly reported in this thesis.

5.2 LDPC Codes of the Newest Communication System Standards

For the purposes of analysis and evaluation of the proposed multi-processor accelerator design methodology and its design space exploration (DSE) framework, it is applied to the design of numerous decoder architectures for the LDPC codes specified in the IEEE 802.15.3c standard [3]. These LDPC codes are standardized for the future high-speed communication systems, such as mmWave WPAN. The standard specifies four LDPC codes of rate 1/2, 5/8, 3/4 and 7/8, but of the same code length of 672-bit. The codes are irregular and have different check node degrees (d_c) and variable node degrees (d_v), as shown in Table 5.1.

Table 5.1: Variable and check node degree distribution, number of macro-rows and macro-columns of the PCMs of various IEEE 802.15.3c LDPC codes

Code Rate	Macro-columns (N_b)	Macro-rows (M_b)	Column Weight (d_v)	Row Weight (d_c)	Non-zero Blocks
1/2	32	16	[20,4,4,4]={4,3,2,1}	[4,4,4,4]={7,6,8,6}	108
3/4	32	8	[24,3,2,2]={4,3,2,1}	[2,2,2,2]={14,15,13,16}	116
7/8	32	4	[29,1,1,1]={4,3,2,1}	[1,1,1,1]={29,30,31,32}	122

For various applications that impose different performance requirements three different performance classes {1,2,3} are part of the IEEE 802.15.3c standard. Class 1 is targeted to low-power low-cost mobile market while maintaining relatively high data rates of up to 1.5 Gbps. Class 2 is specified to achieve high data rates up to 3 Gbps. Class 3 is specified to support ultra-high performance applications with data rates above 3 Gbps, like the transmission of uncompressed high definition (HD) video and audio. Each class utilizes one or more codes for diverse requirements ranges. For example, the rate-1/2 LDPC code is specified for the 385 and 770 Mbps in class 1 and for 1540 Mbps data rate in class 2. The rate-7/8 LDPC code is specified for class 2 applications with the throughput requirement of 3080 Mbps. However, the rate-3/4 code is defined in all classes {1, 2, 3} for different throughput requirements. For instance, 1320 Mbps for class 1, 2640 Mbps for class 2 and 5280 Mbps for class 3 applications. These performance requirements and the particular LDPC code will be used to illustrate the proposed design methodology and its DSE framework.

The PCMs of rate 1/2, 5/8, 3/4 and 7/8 LDPC codes are shown in Table 5.2, 5.3, 5.4 and 5.5, respectively. It is clear that different applications demand various performance ranges from moderate to extremely high. Therefore, to satisfy these different performance levels, especially, high or ultra-high, processing parallelism at both architecture levels needs to be adequately explored and exploited. This kind of parallelism exploration is offered by the proposed multi-processor accelerator design methodology and its DSE framework.

5.3 Micro-/Macro-architecture Design and Tradeoff Experiments

A series of design experiments were conducted with multi-processor decoders for the IEEE 802.15.3c LDPC codes with different processing parallelism combinations. The main aim of the experiments was to explore the numerous design tradeoffs between the micro- and macro-architecture design, when considering the micro- and macro-architectures in combination. We were especially interested in investigating the influence of processing parallelism applied at each of the architecture levels on the design quality metrics (performance, power con-

Table 5.2: block-structured PCM of the rate-1/2 IEEE 802.15.3c LDPC code with 32 macro-columns and 16 macro-rows, size of each sub-matrix is 21x21 and code length is 672, '-' represents zero matrices

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
1	-	-	-	5	-	18	-	-	-	-	3	-	10	-	-	-	-	-	-	5	-	-	-	-	-	-	-	5	-	7	-	-
2	0	-	-	-	-	-	16	-	-	-	-	6	-	-	-	0	-	7	-	-	-	-	-	-	-	10	-	-	-	-	-	19
3	-	-	6	-	7	-	-	-	-	2	-	-	-	-	9	-	20	-	-	-	-	-	-	-	-	-	19	-	10	-	-	-
4	-	18	-	-	-	-	-	0	10	-	-	-	-	16	-	-	-	-	9	-	-	-	-	-	4	-	-	-	-	-	17	-
5	5	-	-	-	-	-	18	-	-	-	-	3	-	10	-	-	5	-	-	-	-	-	-	-	-	-	-	-	-	-	7	-
6	-	0	-	-	-	-	-	16	6	-	-	-	0	-	-	-	-	-	7	-	-	-	-	-	-	-	-	-	19	-	-	-
7	-	-	-	6	-	7	-	-	-	-	2	-	-	-	-	9	-	20	-	-	-	-	-	-	-	-	-	-	-	10	-	-
8	-	-	18	-	0	-	-	-	-	10	-	-	-	-	16	-	-	-	-	9	-	-	-	-	-	-	-	-	-	-	-	17
9	-	5	-	-	-	-	-	18	3	-	-	-	-	-	10	-	-	5	-	-	4	-	-	-	-	5	-	-	-	-	-	7
10	-	-	0	-	16	-	-	-	-	6	-	-	-	0	-	-	-	-	-	7	-	4	-	-	-	-	-	10	-	19	-	-
11	6	-	-	-	-	-	7	-	-	-	-	2	9	-	-	-	-	-	20	-	-	-	4	-	19	-	-	-	-	-	10	-
12	-	-	-	18	-	0	-	-	-	-	10	-	-	-	-	16	9	-	-	-	-	-	-	12	-	-	4	-	17	-	-	-
13	-	-	5	-	18	-	-	-	-	3	-	-	-	-	-	10	-	-	5	-	-	-	-	-	-	-	5	-	-	-	-	-
14	-	-	-	0	-	16	-	-	-	-	6	-	-	-	0	-	7	-	-	-	-	-	-	-	10	-	-	-	-	-	-	-
15	-	6	-	-	-	-	-	7	2	-	-	-	-	9	-	-	-	-	-	20	-	-	-	-	-	19	-	-	-	-	-	-
16	18	-	-	-	-	-	0	-	-	-	-	10	16	-	-	-	-	9	-	-	-	-	-	-	-	-	-	4	-	-	-	-

Table 5.3: block-structured PCM of the rate-5/8 IEEE 802.15.3c LDPC code with 32 macro-columns and 12 macro-rows, size of each sub-matrix is 21x21 and code length is 672, '-' represents zero matrices

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
1	0	-	-	5	-	18	16	-	-	-	3	6	10	-	-	0	-	7	-	5	-	-	4	4	-	10	-	5	-	-	-	-
2	-	-	6	-	7	-	-	-	-	2	-	-	-	-	9	-	20	-	-	-	4	-	-	-	-	-	19	-	-	-	-	-
3	-	18	-	-	-	-	-	0	10	-	-	-	-	16	-	-	-	-	9	-	-	12	-	-	4	-	-	-	-	-	17	-
4	5	0	-	-	-	-	18	16	6	-	-	3	0	10	-	-	5	-	7	-	4	-	-	-	-	-	-	-	-	-	-	-
5	-	-	-	6	-	7	-	-	-	-	2	-	-	-	-	9	-	20	-	-	-	4	-	-	-	-	-	-	-	-	-	-
6	-	-	18	-	0	-	-	-	-	10	-	-	-	-	16	-	-	-	-	9	-	-	12	-	-	-	-	-	-	-	-	-
7	-	5	0	-	16	-	-	18	3	6	-	-	-	0	10	-	-	5	-	7	4	4	-	-	-	5	-	-	-	-	-	-
8	6	-	-	-	-	-	7	-	-	-	-	2	9	-	-	-	-	-	20	-	-	-	4	-	19	-	-	-	-	-	-	-
9	-	-	-	18	-	0	-	-	-	-	10	-	-	-	-	16	9	-	-	-	-	-	-	12	-	-	-	-	-	-	-	-
10	-	-	5	0	18	16	-	-	-	3	6	-	-	-	0	10	7	-	5	-	-	4	4	-	10	-	5	-	7	-	-	-
11	-	6	-	-	-	-	-	7	2	-	-	-	-	9	-	-	-	-	-	20	-	-	-	4	-	19	-	-	-	-	-	10
12	18	-	-	-	-	-	0	-	-	-	-	10	16	-	-	-	-	9	-	-	12	-	-	-	-	-	-	4	-	17	-	-

Table 5.4: block-structured PCM of the rate-3/4 IEEE 802.15.3c LDPC code with 32 macro-columns and 8 macro-rows, size of each sub-matrix is 21x21 and code length is 672, '-' represents zero matrices

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
1	0	-	-	5	-	18	16	-	-	-	3	6	10	-	-	0	-	7	-	5	-	-	4	4	-	10	-	5	-	-	-	-
2	-	18	6	-	7	-	-	0	10	2	-	-	-	16	9	-	20	-	9	-	4	12	-	-	4	-	19	-	-	-	-	-
3	5	0	-	-	-	-	18	16	6	-	-	3	0	10	-	-	5	-	7	-	4	-	-	4	5	-	10	-	19	-	-	-
4	-	-	18	6	0	7	-	-	-	10	2	-	-	-	16	9	-	20	-	9	-	4	12	-	-	4	-	19	-	10	-	-
5	-	5	0	-	16	-	-	18	3	6	-	-	-	0	10	-	-	5	-	7	4	4	-	-	-	5	-	-	-	-	-	-
6	6	-	-	18	-	0	7	-	-	-	10	2	9	-	-	16	9	-	20	-	-	-	4	12	19	-	-	-	-	-	-	-
7	-	-	5	0	18	16	-	-	-	3	6	-	-	-	0	10	7	-	5	-	-	4	4	-	10	-	5	-	7	-	19	-
8	18	6	-	-	-	-	0	7	2	-	-	10	16	9	-	-	-	9	-	20	12	-	-	4	-	19	-	4	-	17	-	10

Table 5.5: block-structured PCM of the rate-7/8 IEEE 802.15.3c LDPC code with 32 macro-columns and 4 macro-rows, size of each sub-matrix is 21x21 and code length is 672, '-' represents zero matrices

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
1	0	18	6	5	7	18	16	0	10	2	3	6	10	16	9	0	20	7	9	5	4	12	4	4	4	10	19	5	10	-	-	-
2	5	0	18	6	0	7	18	16	6	10	2	3	0	10	16	9	5	20	7	9	4	4	12	4	5	4	10	19	19	10	-	-
3	6	5	0	18	16	0	7	18	3	6	10	2	9	0	10	16	9	5	20	7	4	4	4	12	19	5	4	10	17	19	10	-
4	18	6	5	0	18	16	0	7	2	3	6	10	16	9	0	10	7	9	5	20	12	4	4	4	10	19	5	4	7	17	19	10

sumption and area). To be able to obtain the experimental results presented in this chapter, several most promising hardware multi-processor architectures had to synthesized and analyzed for different codes and various application requirements of these codes. The synthesis and evaluation of such a huge number of architecture instances in a reasonably short time was only possible through usage of the proposed DSE framework, supported by the generic accelerator architecture template(s) and a parameterizable generic component library for each component type, like processors, memories and communication network elements. The experiments are performed through the design of various decoders for the rate-1/2 672-bit IEEE 802.15.3c LDPC code under the assumption of data-precision of 8-bits for computations and 10 iterations per frame decoding. The main consideration in this set of experiments is to show the processor micro- and macro-architecture influence on performance, power consumption and area and their mutual tradeoffs. The experimental results are analyzed and discussed in the following three profiles:

1. Fixed micro-parallelism and various macro-parallelism
2. Fixed macro-parallelism and various micro-parallelism
3. Similar parallelism strengths

At one hand, the experiments will provide an insight into the various tradeoffs between the micro- and macro-architecture parallelism. On the other hand, based on the analysis of the results from these experiments, a set of general design rules is established. These rules are incorporated into the DSE framework to effectively prune the design search space and quickly reach an adequate accelerator architecture for a certain quality requirement. Also, this specific experiments organization keeping one level of parallelism fixed, while varying the other will show the impact more clearly. Moreover, it will reconfirm the fact that only with such a design approach it would be possible to explore the numerous tradeoffs in the design of massively parallel multi-processor accelerators.

5.3.1 Fixed Micro-parallelism and Various Macro-parallelism

This set of experiments is used to explore the numerous tradeoffs with various micro- and macro-parallelism combinations, while keeping the micro-parallelism

at a fixed level and varying the macro-parallelism possible to exploit for the LDPC codes as mentioned in the previous section.

The exploration experiments are performed for various combinations of micro-architecture parallelism {1, 2, 4, 8} and macro-architecture parallelism {21, 42, 84, 168, 336}, as shown e.g. in Figure 5.2. In all figures presenting experimental results, P(a, b) denotes a combined micro- and macro-architecture parallelism. In tuple P(a, b), a represents the micro-architecture parallelism of a processor (i.e. the number of processor inputs/outputs), and b represents the macro-architecture parallelism (i.e. the number of processors). The tuple P(a, b) represents a certain micro- and macro-architecture combination with the combined micro- and macro-parallelism (a, b) of the CNP processors (shown on the x-axis in the figures presenting the results). Similar notation for the combined processing parallelism is used for the VNP processors (although, not shown on the x-axis of the result figures).

Design quality metrics, their estimation and evaluation

Among others, the three major design quality metrics are the performance, power consumption and area, as discussed in Section 3.4 of Chapter 3. In the following, it is described how these quality metrics are estimated for various micro-/macro-architecture combinations for LDPC decoders. These metrics are then used to evaluate and compare various architectures based on their total delivered quality and this way to make quality-driven architecture design decisions.

The throughput in Mbps that reflects the performance metric for the LDPC decoding can be estimated based on the message passing (MP) algorithm using the following formula:

$$T_{Mbps} = \frac{R.N.F_{MHz}}{CCPI.I_{tot}} \tag{5.1}$$

where R stands for the code rate, N stands for the code length (size of data frame), I_{tot} stands for the total number of iterations required to decode a frame, F_{MHz} stands for the clock frequency measured in MHz and CCPI represents the number of clock cycles per iteration. When CCPI is multiplied with I_{tot}, it gives the total schedule length, as shown e.g. in Figure 5.2. The total schedule length represents the number of clock cycles (CC) to decode a single LDPC frame on a given micro-/macro-architecture combination. Our proposed schedulers compute the schedule length for a particular micro-/macro-architecture combination, which is then used to determine the throughput using equation 5.1. This way various processor micro-/macro-architecture combinations can be compared in terms of performance, which is one of the design quality metrics. Figure 5.2 shows the exploration results regarding the schedule lengths, determined in clock cycles using the relaxed scheduling (RS) approach, for various micro- and macro-architecture parallelism combinations. The exploration results will be discussed latter in this section.

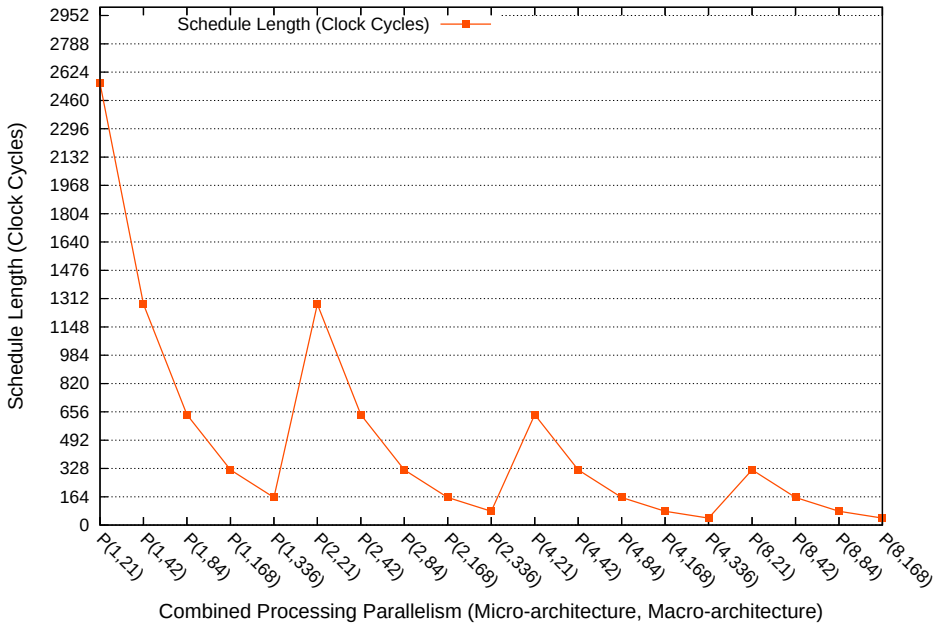

Figure 5.2: Schedule length in clock cycles (CC) for the rate-1/2 672-bit IEEE 802.15.3c LDPC code and assuming 10 iterations/frame decoding.

In general, the relaxed schedule (RS) is a kind of schedule in which some rescheduling possibility of operations to processors is preserved (while still meeting the performance constraints), thereby utilizing this operation scheduling freedom for the memory and communication cost reduction during the memory and communication architecture design. The relaxed schedule is applied in cases where the memory and communication structure influence dominate the processor influence on all or some important design aspects. The other proposed scheduling approach, the so-called tight schedule (TS), mainly aims at the exploitation of any possibility to maximize the processor utilization. This would result in the reduction of the total schedule length and this way enhance the performance, equivalently, a reduction in the number of processors for a fixed performance. The tight scheduling is applied only in cases where the processor influence dominates the memory and communication structure on all or some important design aspects. However, it can influence further the complexity of the communication and memory architectures. When the processor utilization is maximized (equivalently minimize the schedule length for the allocated resources), then the necessary data transfer bandwidth must be guaranteed at any cost of memory and communication. Importantly, in the tight schedules, there is no further possibility of the operation rescheduling to the processors that can be used for the reduction of memory and communi-

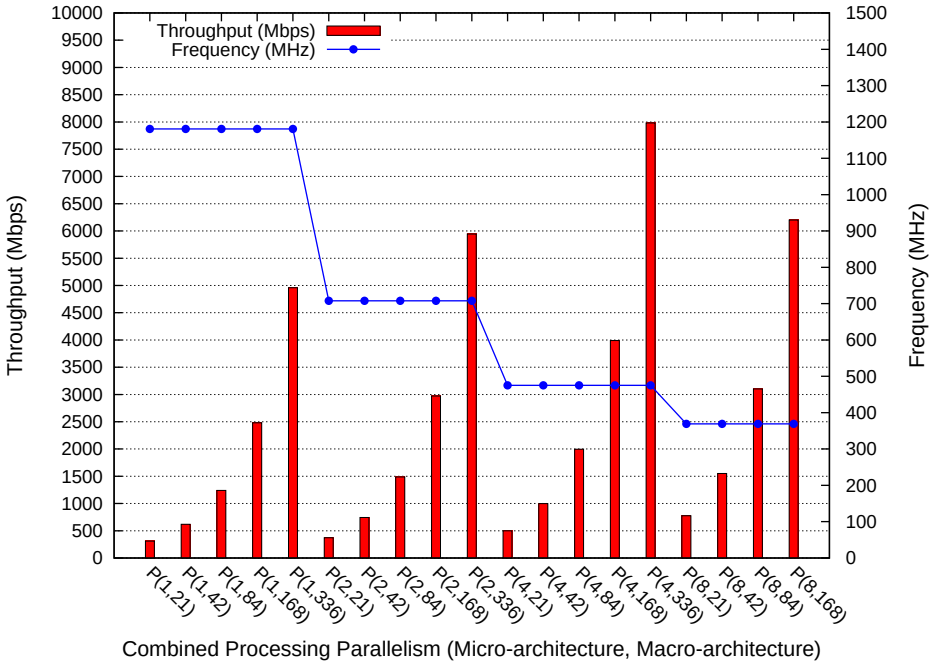

Figure 5.3: Performance/Frequency tradeoffs for fixed micro-parallelism and various macro-parallelism for the rate-1/2 672-bit IEEE 802.15.3c LDPC code

cation complexity. For LDPC decoding, the tight scheduling corresponds to the overlapping of the check node computations with the variable node computations (this maximizes the CN and VN processors utilization), while the relax scheduling corresponds to the processing of CNs computations without overlapping with VNs computations.

As concerned the area, the total processors' area for each architecture instance is calculated using simple addition of the area of individual processors. For instance, for the tuple P(1, 84), i.e. 84 serial processors, the total processors' area is 0.508116 mm^2 (=$84 \times A_{cnp} + 84 \times A_{vnp}$), where A_{cnp} and A_{vnp} represents the area of CNP and VNP processors each with the micro-architecture parallelism of 1 (e.g. see Figure 5.4). Similarly, the area for a particular memory and communication architecture for an architecture instance is computed using the number and types of various memory elements (e.g. single/multi-port memories, etc) and the number and types of various communication elements (e.g. switches and cyclic shifters, etc), respectively.

As concerned the power consumption estimation, for both types of processors, the total power PW_{total} was calculated the same way by summing up their

corresponding static PW_{static} and dynamic $PW_{dynamic}$ power:

$$PW_{total} = PW_{static} + PW_{dynamic} \tag{5.2}$$

The processors modules were characterized for the TSMC 90nm LPHP standard cell library using the back-end Cadence CAD tool flow. The dynamic power was computed using the following formula:

$$PW_{dynamic} = \alpha C V^2 f \tag{5.3}$$

where α is the signal activity factor of the circuit, C the total switching capacitance, V the supply voltage and f the frequency. $\alpha = 0.5$ was assumed.

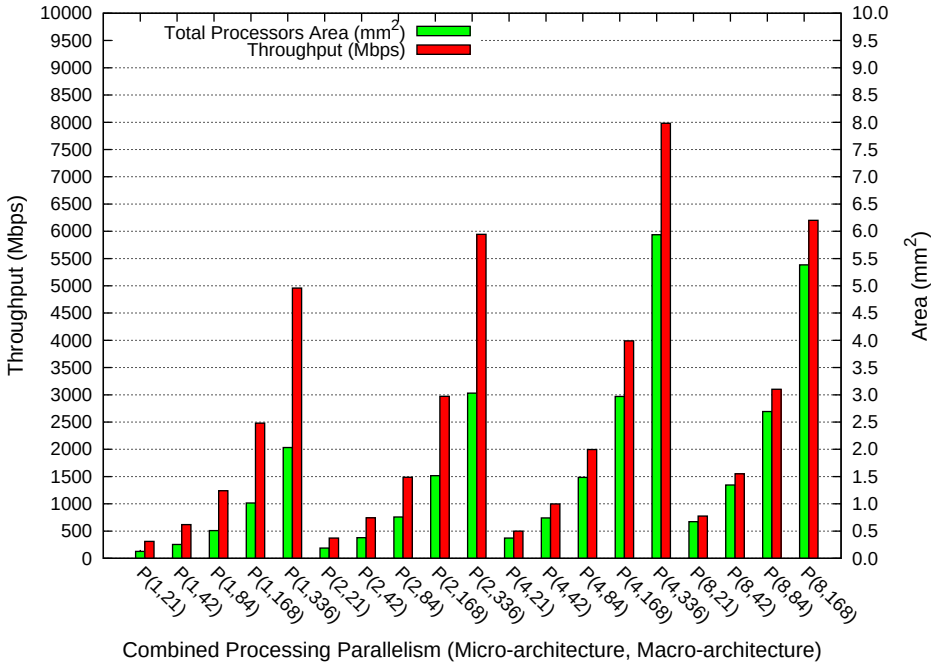

Figure 5.4: Area/performance tradeoffs for fixed micro-parallelism and various macro-parallelism for the rate-1/2 672-bit IEEE 802.15.3c LDPC code

In the following, the experimental results are discussed from the viewpoint of the above-mentioned design quality metrics. For each combination of fixed micro-parallelism in the set {1, 2, 4, 8} and increasing macro-parallelism {21, 42, 84, 168, 336}, the throughput scales linearly, as shown in Figure 5.3. The area and also power consumption increase linearly, as shown in Figure 5.4 and

5.5, respectively[1]. However, the ranges of these parameters are influenced to a large degree by the corresponding micro-architectures. A lot of published work on accelerators for LDPC decoding only consider the two extremes of the micro-parallelism, i.e. fully-serial and fully-parallel architectures.

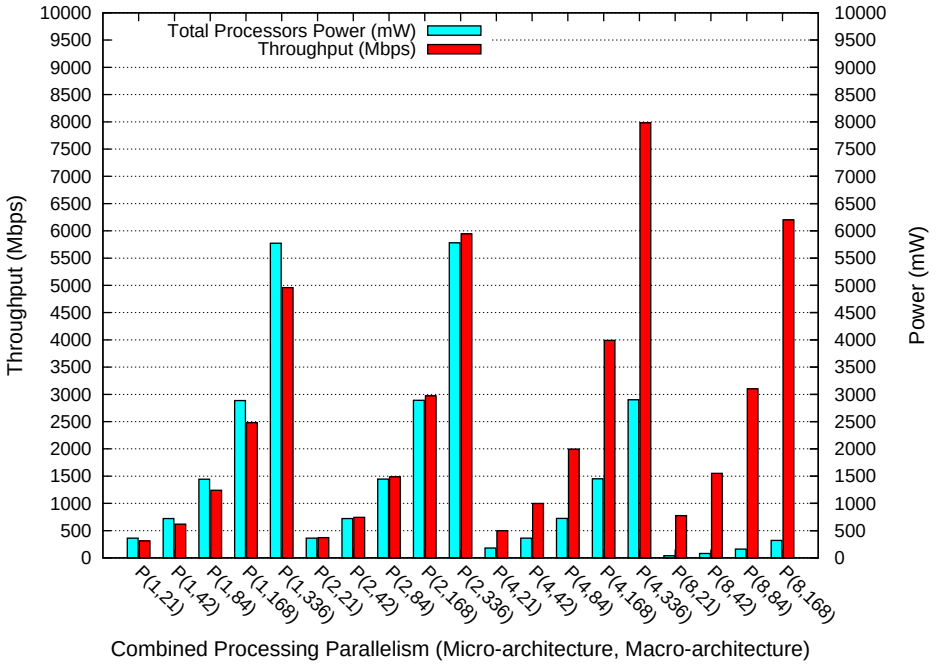

Figure 5.5: Power/Performance tradeoffs for fixed micro-parallelism and various macro-parallelism for the rate-1/2 672-bit IEEE 802.15.3c LDPC code

However, the serial and parallel micro-architectures combined with only increasing macro-parallelism to achieve a certain performance in many cases do not result in optimal solutions regarding area or power consumption. Moreover, for the high-end applications, it may be difficult or even impossible to achieve the required performance by only considering one or another extreme of the micro-parallelism and exclusively increasing the macro-parallelism to the maximum possible value. For the lowest micro-parallelism, it is due to the upper limit on the macro-parallelism that can be exploited in the application and the large number of computation clock cycles required to compute a single check or variable node. However, for the highest possible micro-parallelism levels, it is due to the huge drop in the frequency of the fully-parallel micro-architectures with the parallelism

[1]Without considering the memory and communication influence

increase (specifically for the high rate codes, e.g. rate-7/8 code).

The impact of frequency on the performance and the related tradeoffs for different micro-/macro-architecture combinations for the rate-1/2 672-bit IEEE 802.15.3c LDPC code are shown in Figure 5.3. The results show a drop of almost *5-times* in the frequency of accelerator for the fully-parallel architecture compared to the fully-serial architecture, what negatively influences the performance gain expected from the increased micro-parallelism. Therefore, the exploration of the partially-parallel architectures at both levels has to be performed in order to construct an architecture that adequately satisfies the high-performance demands and other design objectives, as e.g. related to area or energy consumption. The above considered tradeoffs and limitations will be further explained using the following example.

Example 1: Performance/Power/Area (PPA) tradeoffs analysis for fixed micro-parallelism and various macro-parallelism

Let's assume a throughput constraint $T_{cstr} \geq 2.0$ Gbps. Several architectures satisfy this throughput constraint, however, with different resources/power tradeoffs, as shown in Figure 5.3. Among all the architectures that satisfy this throughput constraint, P(1, 168) is the most efficient regarding area (lowest area), however, the worst regarding power (*17.95-times* higher power consumption than P(8, 84)). On the other hand, the architecture P(8, 84) is the most efficient regarding power (lowest power), while worst regarding area (*2.48-times* higher area than P(1, 168)), as shown in Table 5.6. It is worth to note these huge relative differences are among those architecture that are optimal for at least one design objective, while meeting the performance constraint. It is interesting to note that P(1, 168) is the fully-serial micro-architecture, while the P(8, 84) is a fully-parallel micro-architecture, each with different macro-architecture level parallelism. The partially-parallel architecture P(4, 84) architecture provides the best tradeoff regarding all design parameters with *1.40-times* larger area and *4.51-times* higher power consumption in comparison to the lowest-area and lowest-power architectures, respectively (see Table 5.6).

This analysis leads to an interesting conclusion that if the goal is to achieve a moderate level of performance with the area minimization as the main design objective, then the serial micro-architecture with an adequate macro-parallelism have to be explored first. However, for the same moderate level of performance with power as the main design objective, the fully-parallel micro-architectures with an adequate macro-parallelism has to be explored first.

These observations can be used in the form of design rules to efficiently search the design space for a low-area or low-power LDPC decoder architectures while meeting the performance constraint. These observations are incorporated in design rules of the proposed architecture DSE framework to quickly search and evaluate the most promising architectures and this way too much accelerate the overall design process.

Table 5.6: Performance/Power/Area tradeoffs for different micro-/macro-architecture combinations satisfying the performance constraints ($T_{cstr} \geq$ 2.0 Gbps); the area and performance penalty are computed with respect to the optimal architecture regarding a single design objective among the group.

Combined Parallelism	Performance (% of max)	Area (x-times of min)	Power (x-times of max)
P(1, 168)	40.00	min	17.95
P(2, 168)	80.00	1.45	17.97
P(4, 84)	64.34	1.40	4.51
P(8, 84)	max	2.48	min

5.3.2 Fixed Macro-parallelism and Various Micro-parallelism

Another interesting consideration is to explore the impact of increasing the micro-parallelism on the performance and other design parameters by a fixed macro-parallelism. For this analysis, the experimental results regarding performance are reorganized as shown in Figure 5.6. From the results, it can be concluded that for a fixed macro-parallelism and with increasing micro-parallelism the throughput does not scale linearly, unlike was the case of fixed micro-parallelism and increasing macro-parallelism. This is due to the longer critical path delays ($D_{P_{mic}}$) of a processing element with the increase in micro-parallelism (P_{mic}). On one hand, increasing the micro-parallelism reduces the processing time by the reduction of the number of clock cycles (T_{cc}) required for a given computation. On the other hand, the expected performance from increasing the micro-parallelism will be limited by the clock speed ($1/D_{P_{mic}}$) of the processing element. It is very probable that at some point there is no further improvement in performance due to the dominating influence of processing elements critical path delays, i.e.,

$$T'_{cc} \times 1/D_{P'_{mic}} < T_{cc} \times 1/D_{P_{mic}},\ where\ P'_{mic} > P_{mic}, \tag{5.4}$$

Hence, in such cases the partially-parallel micro-architectures have to be considered with adequate macro-parallelism levels to satisfy the demands of high-performance. Also, the area does not grow linearly like in the previous case, because by increase of the micro-parallelism with fixed macro-parallelism the area increases proportionally to the complexity of the micro-architecture, as shown in Figure 5.7.

The non-linearity both in performance and area are interesting from the viewpoint of performance optimization. In the traditional multi-processor architecture design, if the required performance T_{cstr} is marginally higher than what can be delivered T_{del} with a given number of processors, i.e. $T_{cstr} > T_{del}$, then one more processor is added. This may unnecessarily over dimension the system in terms

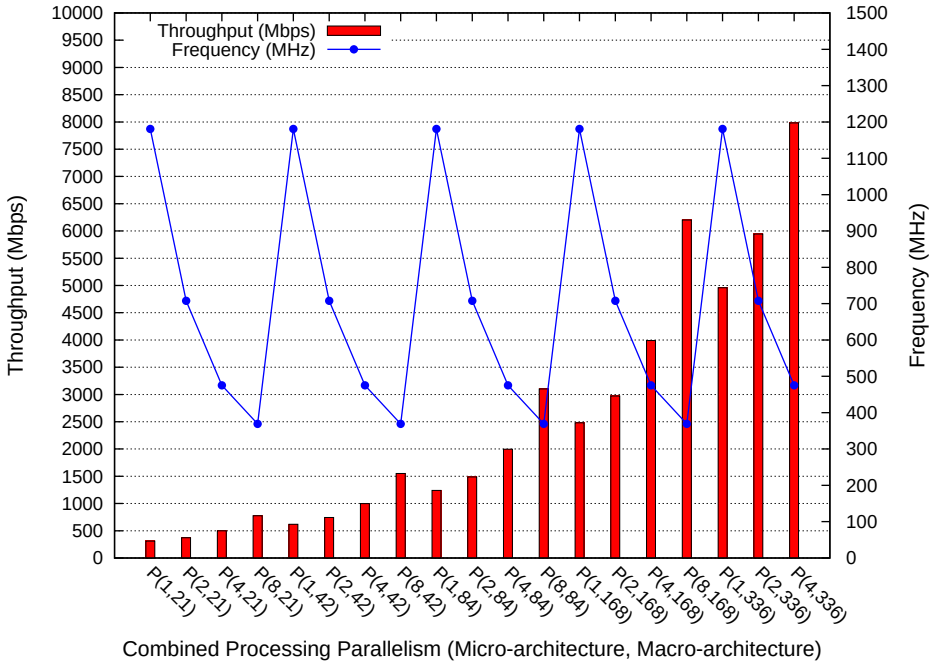

Figure 5.6: Performance/Frequency tradeoffs for fixed macro-parallelism and various micro-parallelism for the rate-1/2 672-bit IEEE 802.15.3c LDPC code

of performance, power consumption and area. Hence, only increasing the micro-parallelism with a fixed macro-parallelism may fill the performance gap in a much better way, and may reduce to a larger degree this over dimensioning. This way the performance constraint would be satisfied in an optimal way with minimum resources overhead.

Moreover, a decreasing trend in power consumption is observed with increase in the micro-parallelism and a fixed macro-parallelism, as shown in Figure 5.8. Consequently, the combined micro-/macro-architecture exploration provides a substantially better power optimization. In this respect, the proposed power optimization and tradeoff approach is quite novel and unique comparing to the more traditional approaches. In the traditional approaches, power reduction (optimization) is usually achieved by doubling the resources to keep up with the performance constraint while reducing the frequency by half. This way the dynamic power is reduced without any loss in performance. As it is well known that the dynamic power is estimated using the following power model:

$$PW_{dynamic} = \alpha C V^2 f \tag{5.5}$$

where $PW_{dynamic}$ represents the dynamic power, α the activity factor of the cir-

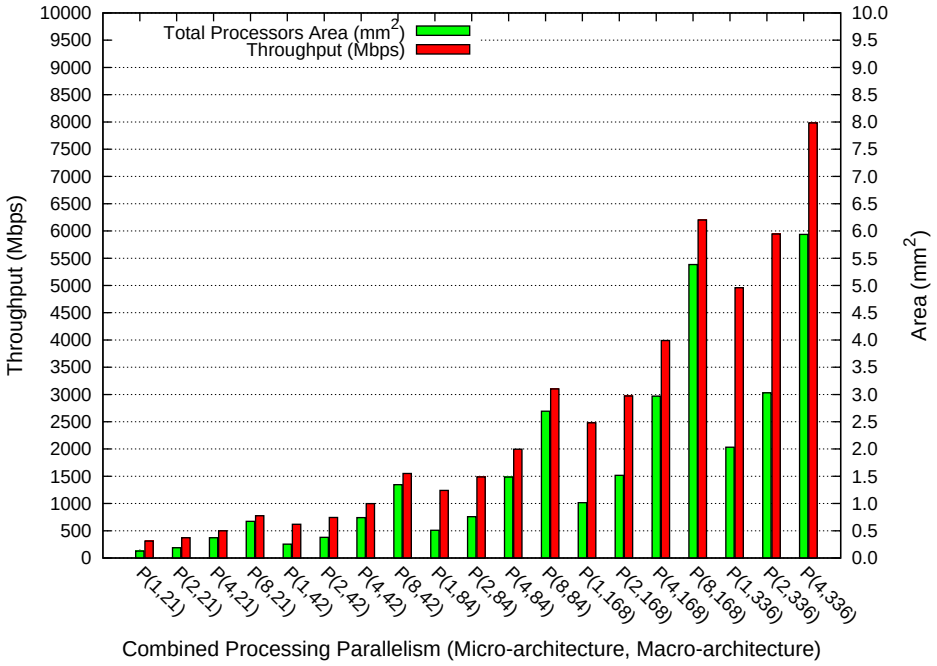

Figure 5.7: Area/performance tradeoffs for fixed macro-parallelism and various micro-parallelism for the rate-1/2 672-bit IEEE 802.15.3c LDPC code

cuit, C the total switching capacitance, V the supply voltage and f the frequency. Moreover, the static power PW_{static} increases proportionally with the increase of resources (area), therefore PW_{static} becomes *2-times* if resources get doubled. However, through the joint consideration of the micro- and macro-architecture, this can be performed in a much better way just by changing only the micro-architecture parallelism, while keeping the macro-architecture the same, as discussed above (assuming that the performance constraint is met). This approach has mainly two benefits compared to the traditional approach.

- lower area overhead
- lower power consumption achieved.

The low area overhead also reduces the total static power PW_{static}. Although the accelerator is clocked at a somewhat higher frequency in our approach, still a lower power consumption is achieved, as the resources are not doubled nor the frequency is halved. This is mainly due to the micro-parallelism exploitation, which is not possible in the traditional approaches. Thus, the proposed combined micro-/macro-architecture exploration provides a novel better way of power opti-

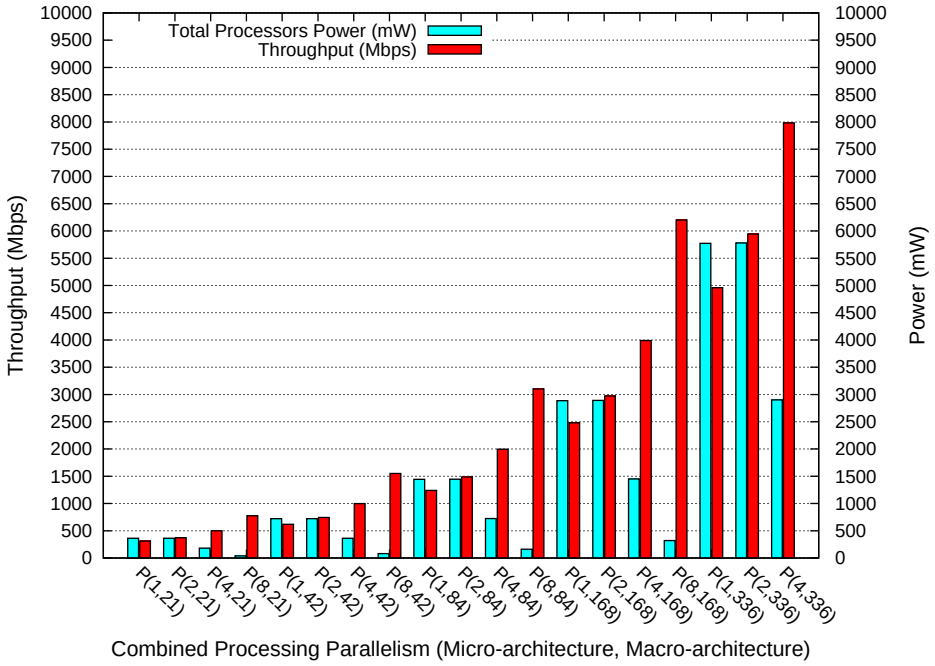

Figure 5.8: Power/Performance tradeoffs for fixed macro-parallelism and various micro-parallelism for the rate-1/2 672-bit IEEE 802.15.3c LDPC code

mization. The following example is provided to further illustrate these tradeoffs. Moreover, the architecture P(1, 168) and P(2, 168) has almost the same power consumption despite of the twice the resources at the micro-architecture level. This is mainly due to the lower operating frequency of P(2, 168) (see Figure 5.6).

Another important observation is that even exploiting the full micro-parallelism with a low macro-parallelism does not guarantee a high performance. Hence, the application-specific instruction-set processor (ASIP) that implement the micro-parallelism in the form of a functional unit inside of ASIPs data-paths are not able to satisfy such stringent requirements. From this, it can be concluded that it is impossible to implement the ultra-high performance applications, while only using an ASIP with the computational kernels implemented in the form of an instruction hardware. Consequently, for the ultra-high performance applications, the hardware architectures that adequately combining the micro- and macro-parallelism should be used. This conclusion also supplements the growing trends in the industry of shifting from homogenous system architectures towards more and more heterogeneous system architectures to overcome the performance and energy crises.

Example 2: Performance/Power/Area (PPA) tradeoffs analysis for fixed macro-parallelism and various micro-parallelism

This example demonstrates the design tradeoffs optimization regarding performance, power and area using various micro-parallelism while keeping the macro-parallelism at a fixed level. The non-linear trends in performance, power consumption and area are quite clear from the results. The reasons are well described earlier in this section. This example will further highlight the performance and power tradeoffs and optimizations. Let's assume a throughput constraint T_{cstr} $\geq$ 1.5 Gbps and a serial micro-architecture (P_{mic}=1). Then, the architectures with the macro-parallelism $P_{mac} \leq 84$ or lower do not satisfy this throughput, while the architectures with $P_{mac} \geq 168$ over dimension the system both from the point of view of cost and performance. Consequently, an increase in the micro-parallelism satisfies the demanded throughput in a better way without increasing the macro-architecture parallelism. This is only possible through the combined consideration of the micro- and macro-architectures. For example, the architecture P(2, 84) provides an area saving of *1.32-times* and power saving *1.98-times* compared to P(1, 168), while satisfying the throughput constraint. From this it is clear that the combined micro- and macro-architecture exploration are extremely important to satisfy the various design constraints and objectives in the best possible way.

To demonstrate the power/area tradeoffs and optimizations, the example given above is continued. From the performance point of view, P(2, 84) seems to be the optimal architecture. Now, suppose the goal is to reduce the power by half, then by the traditional power/frequency tradeoff the resources are increased by *2-times* and the frequency is reduced by half. Applying our power reduction, another design point can be selected just by increasing the micro-parallelism ($P_{mic} = 2$), while keeping the same macro-parallelism ($P_{mac} = 84$). This way a larger power reduction and lower area overhead is achieved. For this example, a relatively high power saving of almost *2-times* is achieved, even ignoring the static power PW_{static}, and the area saving of *1.02-times* compared to the traditional approach. This is only possible through the combined consideration of the micro- and macro-architectures.

5.3.3 Similar Parallelism Strengths

The last direction in which the result will be analyzed is to see the impact of similar parallelism strength on the performance and other design parameters. The results are organized for similar parallelism strengths, as shown in Figure 5.9. The performance results show a trend of relatively higher performance for those architectures having relatively lower micro-parallelism. Although, increasing the micro-parallelism reduces the processing time in terms of clock cycles, it causes an increase of the critical path delays ($D_{P_{mic}}$). The results show that the critical path delays of the processing elements have more influence on the throughput

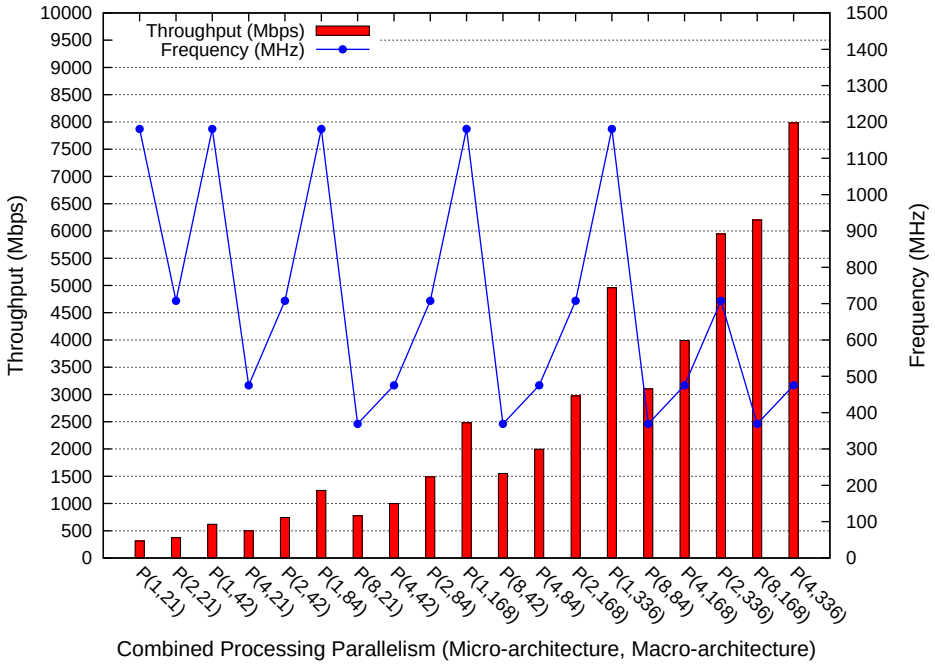

Figure 5.9: Performance/Frequency tradeoffs for similar parallelism strengths for the rate-1/2 672-bit IEEE 802.15.3c LDPC code

than the gain in clock cycles obtain by exploiting more micro-parallelism. This is the main reason for the difference in performances (see Figure 5.9).

Concerning the area, the architectures with relatively lower micro-parallelism have a relatively larger area compared to those exploiting more micro-parallelism, as shown in Figure 5.10. It is the same kind of trend as for the performance, but for power it is completely the other way round, i.e. the architectures having a relatively lower micro-parallelism are the least power efficient (see Figure 5.11). The following example illustrates further these tradeoffs.

Example 3: Performance/Power/Area (PPA) tradeoffs analysis for similar parallelism strengths

Consider the architectures with the equivalent combined parallelism strength 336, i.e. {P(8, 42), P(4, 84), P(2, 168), P(1, 336)}, as pointed out in Figure 5.10. The architecture instance P(1, 336) is the best regarding performance. It provides as high as *2.67-times* gain in performance compared to the lowest performance architecture in the group, as shown in Table 5.7. However, this is accompanied by a substantial area and power overhead. The area overhead is *1.50-times* and

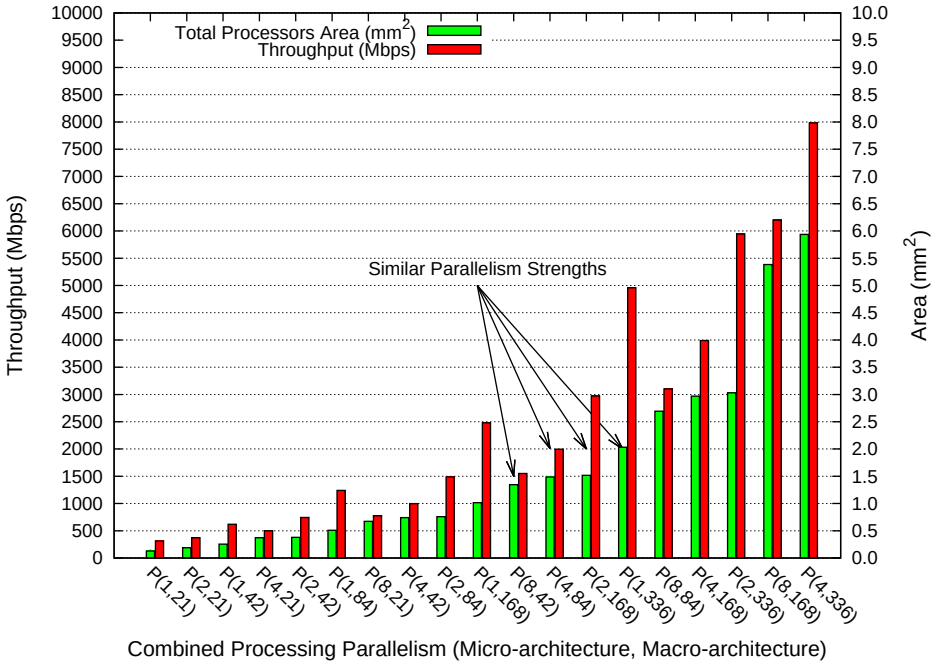

Figure 5.10: Area/Performance tradeoffs for similar parallelism strengths for the rate-1/2 672-bit IEEE 802.15.3c LDPC code

an extremely high power overhead of *71.13-times* in comparison to the least performance, power and area architecture in the group. The architecture P(4, 84) exploiting the partial parallelism at both architecture levels provides the best tradeoffs in relation to all the design quality metrics (performance, power consumption and area, etc).

Summing up, when designing accelerators for highly-demanding applications both the micro- and macro-architecture have to be considered in combination. The micro- or macro-architecture design in isolation would not only make it difficult to realize the high-performance, but would also seriously degrade the quality of the resultant accelerator in other design dimensions such as power and area, as discussed in relation to the experiments for the LDPC decoder applications. From this it is clear, that without an adequate DSE considering the micro- and macro-architecture in combination, it would be impossible to find an accelerator architecture satisfying the diverse quality constraints and objectives. Furthermore, based on the design exploration experiments and their analysis, some techniques and design rules are proposed to quickly search the design space for an adequate architecture, while taking into the diverse design constraints and optimization objectives.

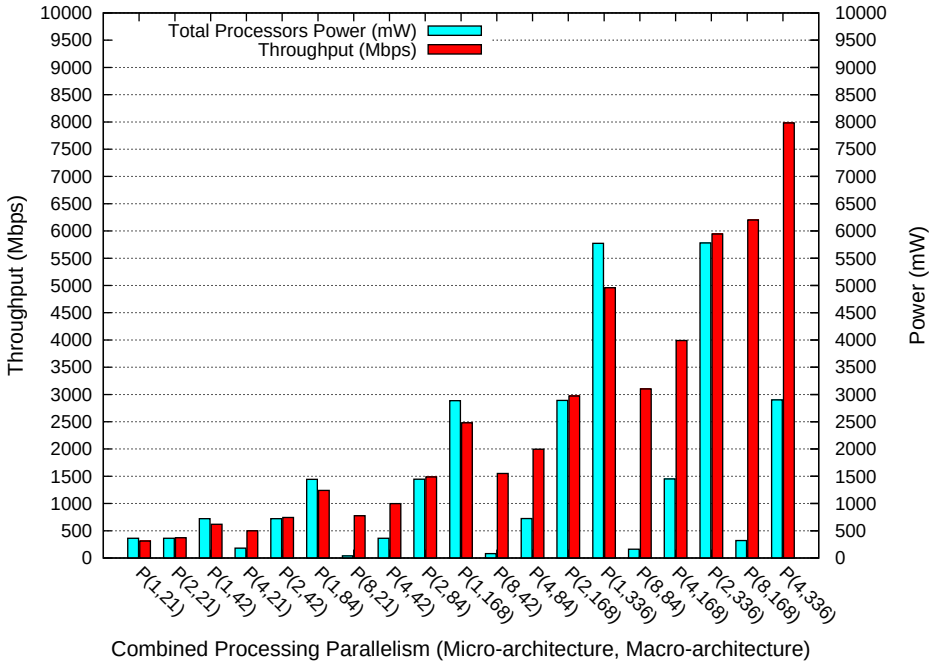

Figure 5.11: Power/Performance tradeoffs for similar parallelism strengths for the rate-1/2 672-bit IEEE 802.15.3c LDPC code

5.4 Processing Parallelism Influence on Communication and Memory

In the previous section, the micro-/macro-architecture exploration and the related tradeoffs are discussed in isolation, i.e. without considering the micro- and macro-architecture decisions on the memory and communication structure. Also, the memory and communication structures influence on the system performance, area, and power consumption are not considered. In this section, the memory and communication influence are taken into account on the performance and area with increase of the processing parallelism at both architecture levels.

First of all, from the experiments it is very clear that the memory and communication structures drastically influence the performance (see Figure 5.12). It is evident, for instance, from a simple comparison of a case in which the memory and communication structures influence on the throughput is taken into account to the case earlier discussed where the memory and communication are not considered (see Figure 5.3). In this set of experiments, the communication between the memories and processors is realized using the traditional flat communica-

Table 5.7: Performance/Area/Power tradeoffs for similar parallelism strength; area penalty and performance loss with respect to the optimal architecture regarding a single design objective among the group.

Combined Parallelism	Performance (% of max)	Area (x-times of min)	Power (x-times of min)
P(8, 42)	37.5	min	min
P(4, 84)	47.5	1.11	9.01
P(2, 168)	72.0	1.13	35.95
P(1, 336)	max	1.50	71.13

tion scheme. The performance saturates quite quickly with increase in either the micro- or macro-architecture parallelism. This is due to the dominating effect of the memory and communication structures delays that limits the performance to only approximately 450 Mbps. Similarly, the communication network dominates the other architecture elements in area with the increase in the micro- and macro-parallelism, as shown in Figure 5.13, except for very low parallelism levels. To enhance the performance with the increase of parallelism, several memory and communication architectures are proposed that are discussed in Chapter 6 of this thesis.

From the above, it is clear that there exist strong interrelationship between the amount of processing resources at the micro-/macro-architecture level and the corresponding memory and communication architecture. One cannot be designed adequately without the consideration of the other.

5.5 Operation Scheduling Influence on Performance and Communication and Memory

In order to explore the tradeoffs between the processors and the memory and communication architectures complexity in relation to the operation scheduling, the tight schedules (TS) and relaxed schedules (RS) are investigated during the DSE. They are introduced earlier in general and in relation to the LDPC decoders. The realization of the two schedules for the LDPC decoding and their influences on the performance and the complexity of the memory and communication architectures will be further explained as follows.

The aim of the tight scheduling is to maximize the processor utilization, thereby decreasing the schedule length and increasing the performance. Equivalently, it causes a decrease in the number of required processing resources for the same performance level. However, it is very probable for this kind of designs that

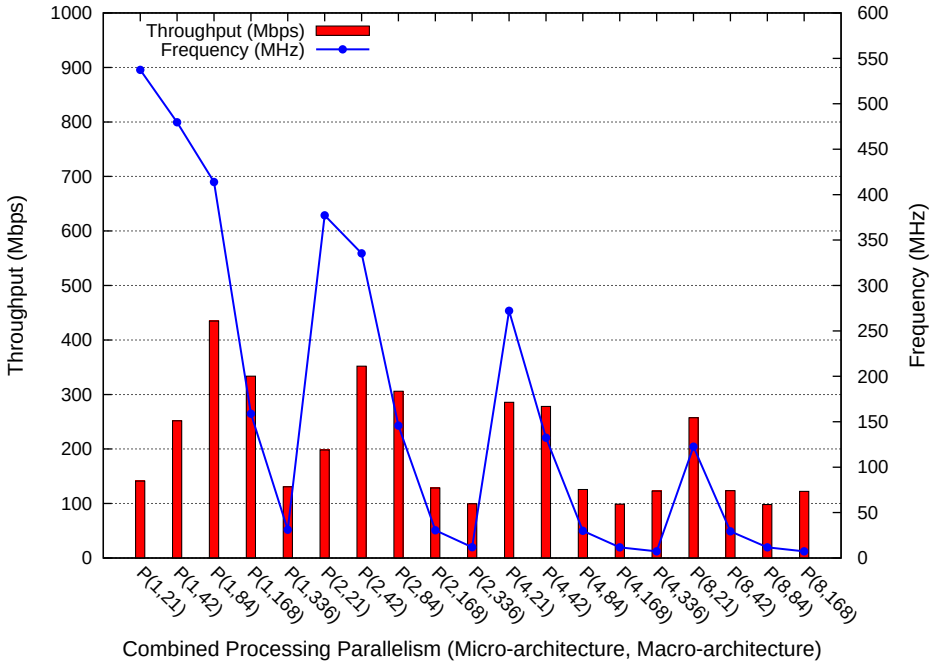

Figure 5.12: Throughput tradeoffs for different combined processing parallelism combinations taking into account the memory and communication structures delays for the rate-1/2 672-bit IEEE 802.15.3c LDPC code

the costs of memory and communication structures may surpass the cost savings due to the decreased processing resources. The tight scheduling strategy is applied only in the cases where the processors dominate in influence the memory and communication architectures in all or some important design aspects. On the other hand, the relaxed scheduling strategy is applied in the cases where the memory and communication architectures dominate the processors influence in all or some important design aspects. In the relaxed scheduling, the processor utilization is kept relatively lower (this way preserving some scheduling freedom), if it helps in reducing the memory and communication costs and delays. This way the operation scheduling freedom can be traded for further reduction of the memory and communication influence on the overall performance, power consumption and area.

The above discussed schedules will be further described for the LDPC decoding applications. In the relaxed scheduling approach, during the processing of check nodes (CNs), all the VNPs waits on the data till all the CNs are processed by the allocated CNPs. Similarly, during the processing of variable nodes (VNs), all the CNPs waits on the data till all the VNs are processed by the allocated

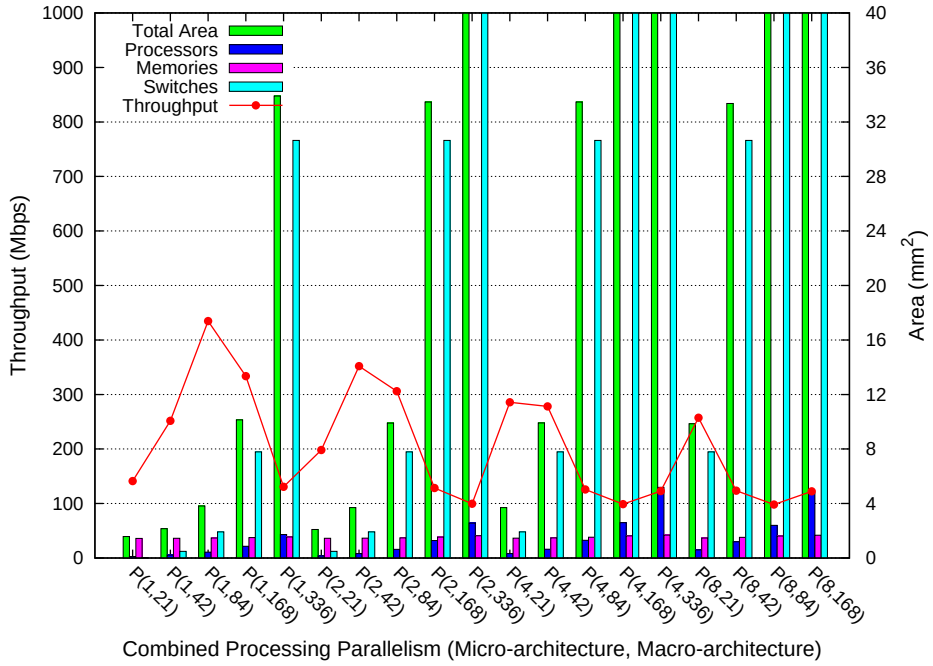

Figure 5.13: Throughput/area tradeoffs for different combined parallelism taking into account the memory and communication delays for the rate-1/2 IEEE 802.15.3c LDPC code

VNPs. However, in the tight scheduling approach, the VNPs do not wait until all the CNs are processed by the allocated CNPs, but start processing earlier (W_{vnp}) if all the data for some VNs are available, as shown in Figure 5.14. Similarly, the CNPs do not wait until all the VNs are processed by the allocated VNPs, but start processing earlier (W_{cnp}) if all the data for some CNs are available, as shown in Figure 5.14. W_{vnp} and W_{cnp} represents the waiting times for VNP and CNP processors. The tight scheduling for the LDPC decoding turns out to be a very complex scheduling problem due to the complex data dependencies among the CNs and VNs. For instance, one CN and the related CNP is responsible for producing partial data not only for a single VN and the related VNP, but for many dependent VNs and the related VNPs. Therefore, for this kind of scheduling, some scheduling algorithms are developed based on the PCM permutation[2] to compute the order of processing and the waiting times W_{vnp} and W_{cnp} for CNs and VNs, respectively. Table 5.8 shows the results of the application of the tight scheduling algorithm to the rate-1/2 672-bit IEEE 802.15.3c codes, and the

[2]The details of these algorithms are given in the Appendix-A

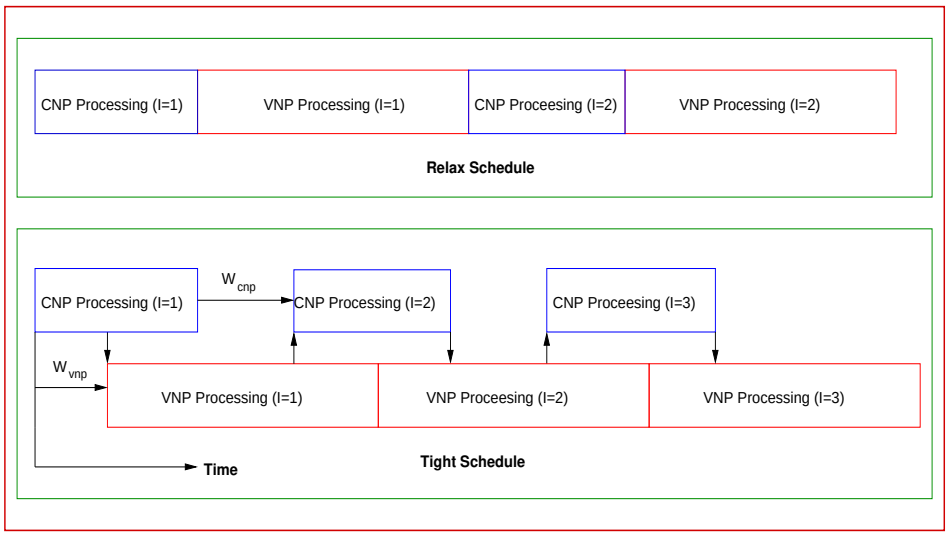

Figure 5.14: Example of relax vs tight operation scheduling for LDPC codes

order of processing of check and variable nodes, as well as, their waiting times. The imposed random order of CNs and VNs further complicates the memory and communications. Moreover, the tight scheduling freezes the rescheduling of CNs and VNs to the corresponding CNPs and VNPs, respectively. This limits any further rescheduling opportunity for the memory and communication complexity reduction. To get a clear picture of the tradeoffs of the two schedules on the performance and the memory and communication architectures, both of them are discussed with respect to the rate-1/2 672-bit IEEE 802.15.3c LDPC code decoders. Assume processing parallelism with $P_{cnp}(8, 21)$ and $P_{vnp}(4, 21)$ for the CNP and VNP processors, respectively. The performance improvement is calculated based on the assumption of a fully-parallel micro-architecture for both CNP ($P_{mic} = 8$) and VNP ($P_{mic} = 4$) and a macro-parallelism ($P_{mac} = 21$) for each kind of processor, as shown in Table 5.9. This is the maximum limit on the best utilization of CNP and VNP processors and the performance gain. With increase in the processing parallelism, the processor utilization decreases due to the relatively longer waiting times of the short schedules. For example for the processing parallelism of $P_{cnp}(8, 21)$ and $P_{vnp}(4, 42)$, the total schedule length is 32 (clock cycles) in the case of relaxed scheduling, while for the tight schedule, the total schedule length is 23, computed using the following equation:

$$TS_{length} = RS_{length} - W_{vnp} - W_{cnp}. \tag{5.6}$$

This results in 28% performance improvement compared to the previous case (33%), as shown in Table 5.9.

Table 5.8: Block-structured PCM of the rate-1/2 672-bit IEEE 802.15.3c LDPC code with the reordered 32 macro-columns and 16 macro-rows for the overlap execution of CNs and VNs. The waiting times are shown by the grey regions of the PCM, size of each sub-matrix is 21x21, '-' represents zero matrices

	30	32	4	6	11	16	29	31	20	3	5	10	27	13	28	26	15	17	19	1	7	12	18	25	21	2	8	9	14	22	23	24
1	7	-	5	18	3	-	-	-	5	-	-	-	-	10	5	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
8	-	17	-	-	-	-	-	-	9	18	0	10	-	-	-	-	16	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
7	10	-	6	7	2	9	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	20	-	-	-	-	-	-	-	-	-
2	-	19	-	-	-	0	-	-	-	-	-	-	-	-	-	10	-	-	-	0	16	6	7	-	-	-	-	-	-	-	-	-
13	-	-	-	-	-	10	-	-	-	5	18	3	5	-	-	-	-	-	5	-	-	-	-	-	-	-	-	-	-	-	-	-
3	-	-	-	-	-	-	10	-	-	6	7	2	19	-	-	-	9	20	-	-	-	-	-	-	-	-	-	-	-	-	-	-
14	-	-	0	16	6	-	-	-	-	-	-	-	-	-	-	-	0	7	-	-	-	-	-	10	-	-	-	-	-	-	-	-
9	-	7	-	-	-	-	-	-	-	-	-	-	-	-	-	5	10	-	-	-	-	-	5	-	4	5	18	3	-	-	-	-
10	19	-	-	-	-	-	-	-	7	0	16	6	-	-	10	-	-	-	-	-	-	-	-	-	-	-	-	-	0	4	-	-
12	-	-	18	0	10	16	17	-	-	-	-	-	4	-	-	-	-	9	-	-	-	-	-	-	-	-	-	-	-	-	-	12
6	-	-	-	-	-	-	19	-	-	-	-	-	-	0	-	-	-	-	7	-	-	-	-	-	-	0	16	6	-	-	-	-
4	-	-	-	-	-	-	-	17	-	-	-	-	-	-	-	-	-	-	9	-	-	-	-	4	-	18	0	10	16	-	-	-
5	-	-	-	-	-	-	-	7	-	-	-	-	-	-	-	-	-	5	-	5	18	3	-	-	-	-	-	-	10	-	-	
16	-	-	-	-	-	-	-	-	-	-	-	-	-	16	4	-	-	-	-	18	0	10	9	-	-	-	-	-	-	-	-	-
15	-	-	-	-	-	-	-	-	20	-	-	-	-	-	-	19	-	-	-	-	-	-	-	-	-	6	7	2	9	-	-	
11	-	-	-	-	-	-	-	10	-	-	-	-	-	9	-	-	-	-	20	6	7	2	-	19	-	-	-	-	-	-	4	-

Table 5.9: Performance gain using the tight scheduling algorithm compared to the relaxed scheduling for LDPC decoders

Code Rate	Code Length	Relax Schedule Clock Cycles	Tight Schedule Clock Cycles ($W_{cnp} + W_{vnp}$)	Performance Gain	Performance Improvement
1/2	(672,336)	48	32 (7+9)	1.5	33.33 %
5/8	(672,420)	44	30 (6+8)	1.46	31.81 %
3/4	(672,504)	40	31 (4+3)	1.21	17.50 %

Even for this low processing parallelism case, the influence of the operation scheduling on the memory and communication architecture complexity and the related tradeoffs cannot be ignored. To clarify the matter, the performance gains that are obtained using the tight schedule are compared to the relaxed schedule for an architecture with $P_{cnp}(8, 21)$ and $P_{vnp}(4, 42)$. Although the tight scheduling approach delivers 28% higher performance than the relaxed scheduling approach. However, even for this low parallelism, the memory and communication structures dominate the processors in cost, as shown in Table 5.10.

With growing parallelism, this cost increase is relatively higher than the cost increase of the processors. Therefore, the performance can be better optimized through utilizing more processors due to their relative low impact on the overall cost of the system than through the tight scheduling. This preserves the rescheduling opportunity during the memory and communication architecture exploration. Therefore, the relaxed scheduling is a better choice in this case to effectively utilize the scheduling freedom in reducing the cost of memory and communication

Table 5.10: Influence of the processor scheduling on the performance and on the memory and communication for the rate-1/2 672-bit IEEE 802.15.3c LDPC code

Schedule Type	Area (mm^2)				Throughput (Mbps)
	Processor	Memory	Switch	Total	
Tight Schedule	0.673	2.134	7.810	10.617	329
Relax Schedule	0.673	1.472	7.810	9.955	257
Relax Schedule*	0.673	1.472	0.417	2.562	632

*After application of memory and communication design strategies

structures using the memory and communication architectures design strategies, which are discussed in the next chapter. The application of the memory and communication design strategies results in the reduction of both the memory and communication structures complexities and delays, thereby backing up again the performance (initially an apparent disadvantage of the relaxed schedule), as shown in Table 5.10.

5.6 Conclusions

In this chapter, it is clearly demonstrated that without considering the micro- and macro-architecture design in combination, it is impossible to arrive at an adequate quality accelerator. These considerations were supported by experiments and case studies with the LDPC decoder design for the IEEE 802.15.3c LDPC codes for the future highly-demanding communication systems. The experiments were focused on exploration of the micro-/macro-architecture tradeoffs regarding the performance, power consumption and area. The experiments clearly demonstrate that there exist various complex tradeoffs at the micro-/macro-architecture level that could only be resolved through an adequate DSE involving a combined micro- and macro-architecture exploration. In particular, it is shown that neither the fully-serial nor the fully-parallel micro-architectures are adequate to satisfy the ultra-high performance requirements. To satisfy the ultra-high performance requirements, the combined micro-/macro-architecture exploration are necessary, which explores and exploits various partially-parallel architecture combinations.

Moreover, two optimization approaches are proposed: one for performance and the other for power. Both of the proposed approaches are based on the consideration of micro-/macro-architecture level parallelism in combination. These approaches are much better than the traditional performance and power optimization approaches. With the proposed performance optimization approach, the performance requirements are satisfied in the best possible way without over-dimensioning of the system, through taking into account the influence of paral-

lelism at both architecture levels on the overall system costs. Similarly, unlike the traditional power/area tradeoffs under a certain throughput constraint, the proposed approach provides much better power optimization, and additionally with a very low area overhead, through the combined micro-/macro-architecture parallelism exploration.

Also, the combined micro-/macro-architecture exploration provides an adequate balance between the various design objectives through the orchestrated partial parallelism exploitation at both architecture levels. If area is the main design objective then the serial micro-architectures with an adequate macro-architecture perform the best, provided the performance constraint is satisfied. However, if power consumption minimization is the design objective, then the fully-parallel micro-architecture performs the best, provided the performance constraint is satisfied. Adequate partial parallelism exploitation at both architecture levels for a given application delivers the best tradeoff among all the design objectives.

Furthermore, it is demonstrated that the memory and communication influence dominate the processor's influence on all the design dimensions for moderate and high parallelism levels. Finally, the two proposed operation scheduling approaches are implemented and compared for LDPC decoding. It is shown that the tight schedules provides performance enhancements compared to relaxed schedule of approximately 28% for a certain combined processing parallelism levels. The performance can be traded off against the memory and communication cost by an increase in the number of processors, as the increase in the processor cost is relatively lower than the memory and communication cost increase, using the relaxed scheduling. Moreover, the rescheduling opportunity in the case of relaxed scheduling can be exploited for the memory and communication cost reduction, as the memory and communication have a dominating influence on the overall system cost with increase in the processing parallelism.

CHAPTER 6

Communication and Memory Architecture Exploration

In the previous chapter, the micro- and macro-architecture parallelism exploration and the related tradeoffs are discussed. Also, the strong interrelationships among the processor micro- and macro-architectures and the corresponding memory and communication architectures are highlighted. This chapter covers the design of communication and memory architectures of massively parallel hardware multi-processors, which are necessary for the implementation of highly-demanding applications. In this chapter, it is demonstrated that for the massively parallel hardware multi-processors the traditionally used flat communication architectures and multi-port memories do not scale well, and the memory and communication network influence on both the throughput and circuit area dominates the processors influence. To resolve the problems, we proposed to design highly optimized application specific hierarchical communication and memory architectures through exploring and exploiting the regularity and hierarchy of the actual data flows of a given application. Furthermore, we proposed some data distribution and related data mapping schemes in the shared (global) partitioned memories with the aim to eliminate the memory access conflicts, as well as, to ensure that the proposed communication design strategies will be applicable. These architecture synthesis strategies are incorporated into the quality-driven model-based multi-processor design method and related automated architecture exploration framework. Using this framework, a large series of experiments is performed that demonstrate many various important features of the synthesized memory and

communication architectures. They also demonstrate that the proposed method and its DSE framework are able to efficiently synthesize well scalable memory and communication architectures even for the high-end multi-processors. Moreover, architectures for the LDPC decoder constructed by our method are compared to the state-of-the-art LDPC architectures from the related research literature.

The rest of this chapter is organized as follow. Section 6.1 discusses the memory and communications issues of massively parallel hardware multi-processors. Section 6.2 discusses the proposed memory and communication architecture exploration and synthesis approach. Section 6.3 covers the application of the proposed memory and communication architecture synthesis techniques to the design of LDPC decoders for the latest communication system standards. A comprehensive discussion on the experimental results closes this section. Section 6.4 discusses the architectures constructed for LDPC decoders by our method to the architectures proposed in the literature. Finally, Section 6.5 concludes this chapter.

6.1 Issues of Communication and Memory Architecture Design for Massively Parallel Hardware Multi-processors

Hardware acceleration of critical computations has been intensively researched during the last decade, mainly for signal, video and image processing applications, for efficiently implementing transforms, filters and similar complex operations. This research was, however focused on the monolithic processing unit synthesis with the so called "high-level synthesis" (HLS) methods [11–19], and not on the massively parallel hardware multi-processor accelerators required for the high-end applications. Specifically, this research did not address the memory and communication architecture design of the multi-processor accelerators.

Although some research results related to the memory and communication architectures can be found in the literature [27–33] in the context of programmable on-chip multi-processor systems, the memory and communication architectures were proposed there for the much larger and much slower programmable processors. They are not adequate for the small and ultra-fast hardware processors of the massively parallel multi-processor accelerators due to a much too low bandwidth and scalability issues. The approaches proposed in the context of the programmable on-chip multi-processors utilize the time-shared communication resources, such as shared buses or Network-on-Chip (NoC). Such communication resources are however not adequate to deliver the data transfer bandwidth required for the massively parallel multi-processor accelerators. In case of the high-end multi-processor accelerators, the application-specific processors and the corresponding memory and communication architectures must be compatible (match each other) with respect to bandwidth (parallelism). Therefore, the communication architectures cannot be realized using the traditional NoC or bus communica-

tion to connect the processing and storage resources, but requires point-to-point (P2P) communication architectures compatible with the parallel processing and memory resources. The traditional NoC-based communication architectures utilize a network of switches, as for instance, each switch connected to one resource (processor, memory, etc) and four interconnected neighboring switches forming a mesh [32]. This way a large number of resources can be connected without using long global wires and thus reducing the wire delays (scalability). However, the time-shared links introduce extra communication cycles, which negatively impact the communication and overall performance. The performance degradation grows with the increase of the number of processing elements and more global or irregular application communication patterns, and grows especially fast for applications that require a large number of processors and massive global or irregular communication. Our approach to communication architecture is somewhat similar to the approaches proposed in [27, 28], but only in relation to the concept of hierarchical organization of the computation and communication resources, while the implementation of this concept is different.

Since LDPC decoding is used as a representative application in the evaluation of the design method, as well as the memory and communication architectures, the processor, memory and communication architectures proposed for the LDPC decoding are briefly discussed. In the past, several partially-parallel architectures have been proposed for the LDPC decoding [56–58, 61, 62, 72, 82]. However, they only deliver a throughput of a few hundreds of Mbps. For such low throughput, a very limited processing parallelism is exploited, and in consequence, simple communication architectures are needed in the form of simple shifters. The proposed partial parallel architectures are not adequate for the high-end applications that require throughputs in the ranges of multi-Gbps. To achieve such ultra-high throughput, massive parallelism has to be exploited. This makes the memory and communication architecture design a very challenging task.

From the above discussion of the related research, it follows that the memory and communication architecture design being of crucial importance for the high-end hardware multi-processors is not adequately addressed by the related research. In this section, the issues of the memory and communication architecture design for massively parallel multi-processor accelerators will be discussed in more details, as sketched in Chapter 2.

Many modern applications (e.g. various communication, multimedia, networking or encryption applications, etc) involve sets of heterogeneous data-parallel tasks with complex inter-task data dependencies and interrelationships between the data and computing operations at the task level. Often the tasks iteratively operate on each other's data. One task consumes and produces data in one particular order, while another consumes and produces data in a different order. Additionally, in the high performance multi-processor accelerators, parallelism has to be exploited on a massive scale. However, due to area, energy consumption and cost minimization requirements, partially parallel architectures are often used, which are more difficult to design than fully parallel ones. Moreover, many of

the modern demanding applications involve algorithms with massive data parallelism or task-level functional parallelism. To adequately serve these applications, hardware accelerators with parallel multi-processor macro-architectures have to be considered. These macro-architectures have to involve several identical or different concurrently working hardware processors, each operating on a (partly) different data sub-set. All this results in complex memory accesses and complex communication between the memories and processing elements. For applications of this kind, the main design problems are related to an adequate resolution of memory and communication bottlenecks and to decreasing the memory and communication hardware complexity.

Moreover, each of the processors of the multi-processor can be more or less parallel, what results in the necessity to explore the various possible tradeoffs between the parallelism at the micro- and macro-architecture level. The two architecture levels are strongly interwoven also through their relationships with the memory and communication structures. Each micro-/macro-architecture combination affects the memory and communication architectures in a different way. For example, exploitation of more data parallelism in a computing unit micro-architecture usually demands getting the data in parallel for processing. This requires simultaneous access to memories in which the data reside (this results in e.g. vector, multi-bank or multi-port memories) and simultaneous transmission of the data (this results e.g. in multiple interconnects), or pre-fetching the data in parallel to other computations. This substantially increases the memory and communication hardware. From the above, it should be clear that for applications of this kind complex interrelationships exist between the computing unit design and corresponding memory and communication structure design. Also, complex tradeoffs have to be resolved between the accelerator effectiveness (e.g. computation speed or throughput) and efficiency (e.g. hardware complexity, power and energy consumption etc.).

The traditionally used simple flat communication scheme, independent of its specific implementation, does not scale well with the increase in the number of processing elements and/or memories. For instance, in switch-based architectures, both the switch complexity and the number of switches grow with the increase of the number of processing elements and/or memories. In the traditional flat interconnection scheme, for n processing elements that have to communicate with m memories, an mxn (input ports x output ports) crossbar switch is required, as shown in Figure 6.1. In result, when used for a massively parallel hardware multi-processors the communication network influence usually dominates the processing elements influence on the throughput, circuit area and energy consumption. Finally, the large flat switch that would be necessary for such a massively parallel multi-processor can be difficult to place and route, even with the most advanced hardware synthesis tools. The place and route may use a long time or in some cases not finish their work at all. This represents an actual practical limitation on the interconnect design.

Regarding the memory issues, the memory bandwidth (number of ports)

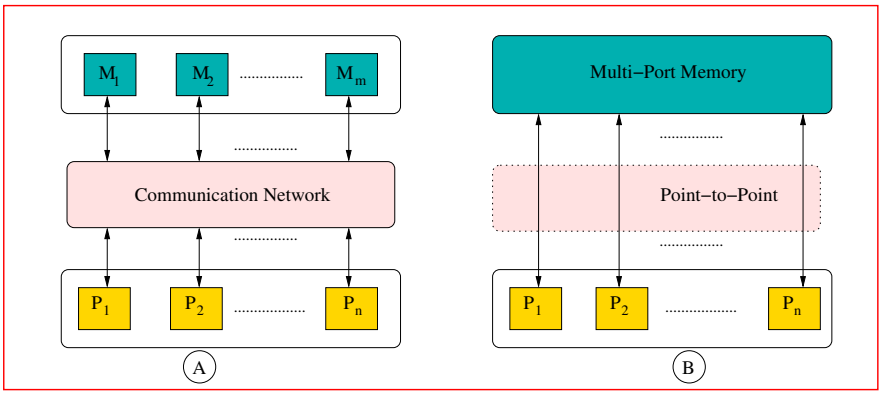

Figure 6.1: (A) Example of communication network among M global memories and N processors (B) Multi-port memory structure to satisfy the bandwidth requirement of multi-processors with low complexity point-to-point interconnects

should be compatible with the processing bandwidth. Thus, a multi-port memory with as many memory ports as required by the processing elements (aggregate bandwidth) seems to be the most natural and straightforward approach (see Figure 6.1). However, with increase in the processing parallelism, the required memory bandwidth (number of ports) increases. The situation quickly deteriorates with parallelism increase resulting in an excessively high complexity due to high memory bandwidth (number of ports) required in the massive parallelism range. For the massively parallel multi-processors, the single multi-port memory would have a prohibitively large area and long delay, when satisfying the required memory bandwidth (see Figure 6.2). Therefore, the data have to be organized in multiple multi-bank or vector memories to satisfy the required memory bandwidth, while keeping the delay and area of the memory architecture substantially lower. Consequently, the most important issues of the memory architecture design are the following:

- the organization of data in vectors (tiles) and the data tiles into multiple memory tiles (partitions) to satisfy the required bandwidth,
- the data distribution and related data mapping into the memory tiles ensuring the conflict-free memory accesses and reducing the memory-processor communication complexity.

It is possible that a data distribution scheme would be conflict free, but data might be distributed very randomly in the memory partitions. This would increase the communication complexity. Therefore, a memory exploration and synthesis method should adequately addresses the issues of memory partitioning and data distribution.

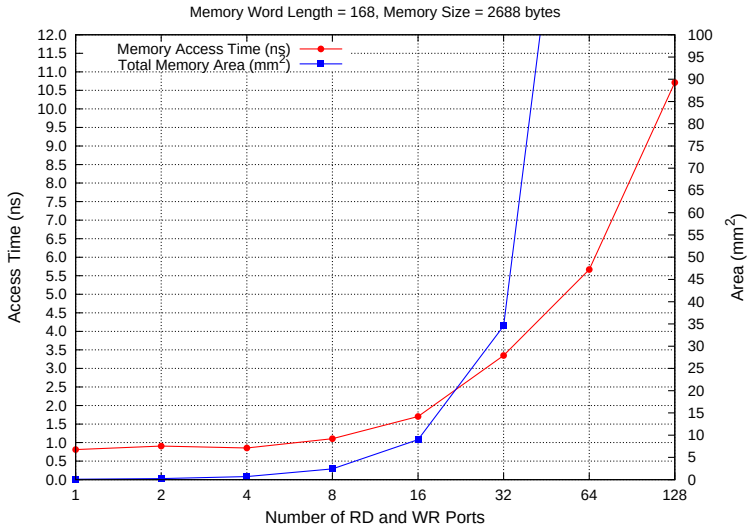

Figure 6.2: Area vs. Access Time of multi-port memory characterized for CMOS 90nm process using HP CACTI 5.3 cache/memory compiler

Also, with increase of the processing parallelism, data have to be partitioned and stored in more and more distributed parallel memories for more parallel access. This causes the memory block sizes to shrink. At some point, it becomes not to be any more efficient to store the data in embedded SRAM memories, but the register-based (Flip-Flop) memories have to be used, which are more efficient for small memory sizes. This issue is taken into account during the memory architecture design. The experiments with different memory configurations demonstrated that for sizes lower than (Height $\times$ Width=32×168), the SRAM memories are less efficient than the FF-based memories. For example, a memory of size (16×168) when implemented as embedded SRAM is almost *1.6-times* larger than when implemented as FF-based[1] memory, and the area proportion grows fast with further decrease in memory sizes. Therefore, for the case of IEEE 802.15.3c LDPC decoders the SRAM based memories are only efficient (and considered in our DSE and experimental designs) for a combined processing parallelism of up to 84 only.

Additionally, the memory and communication issues are not orthogonal in nature, resolving and optimizing one issue in separation heavily influences the other. Thus, the memory and communication architecture synthesis has to be realized as one coherent synthesis process accounting for the mutual influences and tradeoffs. Moreover, in such applications, the task mapping to the related hardware processors, data mapping to memories and communication between

[1]TSMC 90nm LPHP Standard Cell Library

processors and memories represent so strongly interrelated design decisions that they cannot be performed independently. In particular, decreasing the memory system complexity may heavily increase the communication complexity or vice versa.

Summing up, the massive data, operation-level and task-level parallelism to be exploited to achieve the ultra-high throughput required by the highly-demanding modern applications, the complex interrelationships between the data and computing operations, and the combined massive-parallelism exploitation at the two architecture levels, make the design of an effective and efficient communication and memory architecture a very challenging task. To effectively perform this task, the (heterogeneous) massive parallelism available in a given application has to be explored and exploited in an adequate manner to satisfactorily fulfill the design requirements through constructing an architecture that satisfies the required performance, area and power tradeoffs.

To illustrate the requirements and issues of memory and communication architecture design, as well as, to introduce and illustrate the proposed design approach, LDPC decoding is used as a representative application.

Usually, iterative Message Passing algorithms (MAP) are used for decoding of the LDPC codes [39]. The algorithm starts with the so-called intrinsic log-likelihood ratios (LLRs) of the received symbols based on the channel observations. During decoding specific messages (extrinsic) are exchanged among the check nodes and variable nodes along the edges of the corresponding Tanner graph for a number of iterations. The variable and check node processors (VNP, CNP) corresponding to the VN and CN computations, iteratively update each other data, until all the parity checks are satisfied or the maximum number of iterations is reached. The data related to the check and variable node computations are stored in the corresponding shared check and variable nodes memories (M_{cv}, M_{vc}), respectively. The CNPs read data from M_{vc} in their required order and after processing write back in M_{cv} in the order required by VNPs, and vice versa for VNPs. The complicated inter-task data dependencies result in complex memory accesses and difficult-to-resolve memory conflicts in the corresponding partially-parallel architectures.

The Tanner graphs corresponding to practical LDPC codes of the newest communication system standards involve hundreds of variables and check nodes, and even more edges. Thus, the LDPC decoding for these standards represents a massive computation, as well as, complex storage and communication task. Moreover, the modern communication system standards require a very high throughput in the range of Gbps and above, for applications like digital TV broadcasting, mmWave WPAN, etc. For realization of the multi-Gbps throughput required by these standards massively parallel hardware multi-processors are necessary. For such multi-processors, the memory and communication architecture design play a decisive role.

6.2 Communication and Memory Architecture Design for High-end Multi-processors

In the previous section, the issues and requirements of memory and communication architectures design for high-end hardware multi-processors are discussed. An important part of the proposed design methodology addresses the issues of communication and memory architectures design for massively parallel hardware multi-processors, and specifically, the memory and communication architectures scalability for the high-end applications. In this section, some communication and memory design strategies are proposed that enable to construct effective and efficient architectures for the high-end multi-processors. It is also discussed that how these strategies are incorporated into the proposed architecture exploration framework, and how they are used to quickly explore the various tradeoffs among the different architecture options and to select the most promising architecture.

Our approach is based on the exploration of computation and communication hierarchies and flows present in a given application, and on using the knowledge from this exploration for the automatic design of communication and memory architectures. Based on the analysis of the communication hierarchies and data flows, the processing elements are organized in a corresponding hierarchical way into several tiles (groups). The tiles are then structured into one global cluster or several global communication-free smaller clusters (if possible), and their respective data in memory tiles. The tiles and clusters replace a fully-flat communication network (that otherwise would be required and would result in a high interconnect complexity), with several much smaller hierarchically organized autonomous communication networks (what results in a substantially smaller interconnect complexity and improved scalability).

Since the global communication complexity and delays grow drastically with the increase of parallelism, some strategies are developed to decompose the global cluster into multiple much smaller communication-free clusters. For a particular application, this partitioning is performed by taking into account the application parallelism and by adequate mapping of computation tasks and their data to the processors and memories, respectively. This localization of communication involving several small size clusters eliminates the global inter-tile communication, and results in a substantial improvement of the communication architecture scalability for the high-end applications.

Secondly, in the cases where the inter-tile global communication is unavoidable, a decomposition strategy is used in which one global cluster (global-switch) is decomposed into multiple smaller clusters (switches) again by exploiting a careful analysis of data in memories. Finally, we also exploit several different kinds of switches (e.g. single-stage switches or multi-stage switches), each appropriate to be used in a different context. All these strategies combined in a proper way result in resolution of the communication bottlenecks and related physical interconnect issues in the architecture. This way an optimized well-scalable com-

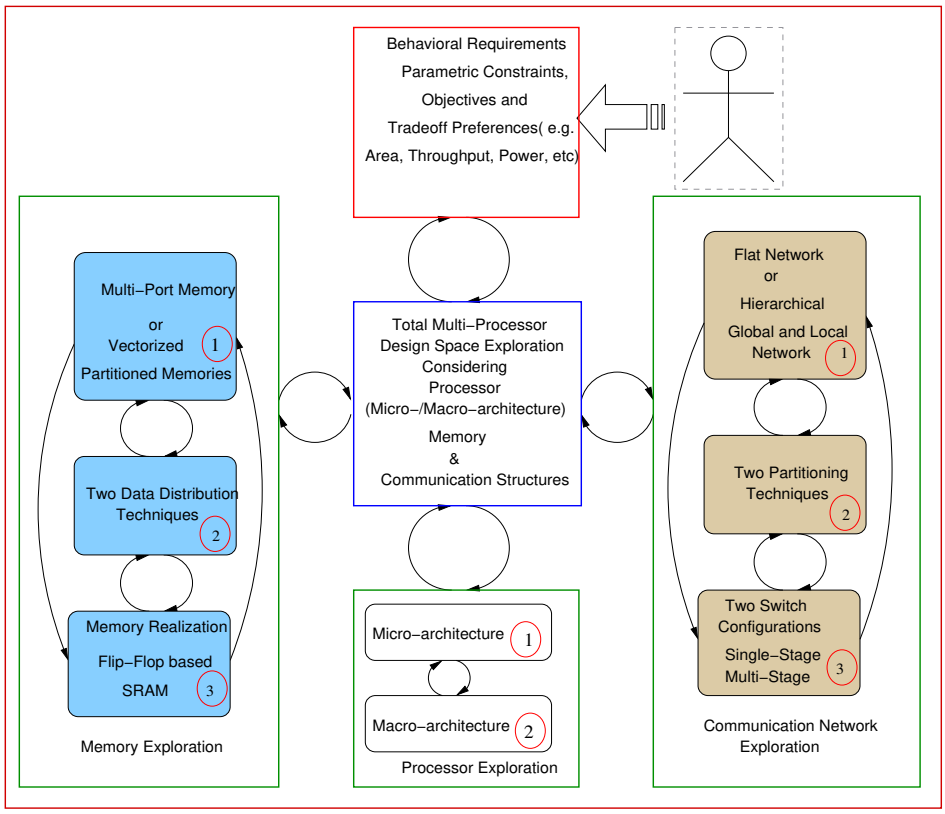

Figure 6.3: Design Space Exploration (DSE) of communication and memory architectures using various strategies

munication architecture is designed, while at the same time realizing an effective and efficient application-specific memory-processor communication, as well as an adequate task and data mapping to particular processing elements and memories, respectively. The above-introduced strategies can be applied in different possible combinations. For example, a two-level hierarchical organization may be followed by partitioning or realized as the two-level network with different single-/multi-stage switch configurations. Different strategy combinations result in different tradeoffs. The above strategies and the order in which they can be applied is represented in the form of a flow diagram in Figure 6.3.

Due to the complex interrelationships between the data and computing operations at the task level and complex inter-task data dependencies, an adequate customization of memory architecture is one of the major design tasks for the massively parallel hardware multi-processors. For a given application, all data

(input and intermediate) specified in the form of single and multi-dimensional arrays have to be stored in multiple shared memories. Different tasks and their corresponding processors impose different access requirements (read/write orders) on the shared memories. Taking into account the single task access requirements on the shared memories would certainly paralyze the other tasks that access the same shared memories for other computations. To ensure the required memory bandwidth and conflict-free data access, data have to be partitioned, distributed and mapped in multiple vector or multi-bank memories, as discussed in the previous section. This way the overall complexity of the memory architecture will be lower and at the same time would satisfy the required memory bandwidth. The problems of data organization into vectors and the required number of shared vector memory tiles (partitions) are resolved together with the communication architecture design, when the flat communication network is transformed into the hierarchical network. However, providing as many shared vector memory tiles (partitions) as the processing tiles would only partially solve the problem due to the possible memory access conflicts. Therefore, the data distribution and data mapping in the partitioned memories are performed with the aim to eliminate the memory access conflicts, as well as, to ensure that the proposed communication strategies would be applicable. It is worth to be noted that the proposed memory partitioning and data distribution approach avoids data duplication. The data distribution and data mapping approach is described below, using an example of two heterogeneous data-parallel tasks sharing multiple memories.

Let us assume a set of m data-parallel tasks T_i=$\{T_1.....T_m\}$, and another set of n data-parallel tasks T_j=$\{T_1.....T_n\}$. Let $|P_i(P_{mic}, P_{mac})|$ and $|P_j(P_{mic}, P_{mac})|$ be the number of processing tiles allocated to the tasks T_i and T_j, respectively, where P_{mic} represents the micro-architecture parallelism and P_{mac} represents the macro-architecture parallelism of each processing tile. Let $|M_{i,j}| = P_i(P_{mic}) \times P_i(P_{mac})$ and $|M_{j,i}| = P_j(P_{mic}) \times P_j(P_{mac})$ be the number of memory tiles shared among the processing tiles $|P_i|$ and $|P_j|$. Further, it is assumed that $|P_i|$ reads data from $|M_{i,j}|$ and writes to $|M_{j,i}|$ and vice versa for $|P_j|$.

For data distribution, an interleaved (cyclic) data distribution scheme is proposed. This approach regularly and uniformly distributes data in memories, which enables us to use the proposed communication strategies. Further, this approach has the additional benefit that it minimizes the complexity of the addressing logic. The data distribution is performed in two stages based on interleaving to resolve the read and write access conflicts, respectively. Depending on the number of shared memory tiles (partitions), the data distribution is performed as given by the equation below:

$$M_{i,j}(x) = S_{i,j} \ \% \ |M_{i,j}|, \ where \ 0 \leq x \leq |M_{i,j}|, \ i=1.....m, \ j=1.....n \qquad (6.1)$$

where $M_{i,j}(x)$ represents the specific shared vector memory tile to which a particular data tile $S_{i,j}$ is mapped, where the subscripts i and j in $S_{i,j}$ represents the data dependence between the task T_i and T_j, and $|M_{i,j}|$ represents the total

number of shared memory tiles from which processors $|P_i|$ read and $|P_j|$ write their data. All the data tiles $S_{i,j}$ are organized as two dimensional arrays that facilitate the automatic data distribution in the shared memory tiles $|M_{i,j}|$ using equation 6.1.

Figure 6.4 shows our data distribution approaches in the shared partition memories with 4 memory partitions. This data distribution (represented by distribution-L1) will resolve all the memory read conflicts for processors $|P_i|$ that will be in the case if no memory partitioning is done and all data is stored in a single memory (single port), as shown in Figure 6.4. On the other hand, when the processor tiles $|P_j|$ write to the share memory tiles $|M_{i,j}|$, it might result in write conflicts because of the order imposed by the $|P_i|$ processor tiles for conflict free read on the data tiles, as given in equation 6.1. Therefore, another level of data interleaving is used so that the processor tiles $|P_j|$ write their data without any conflict, while ensuring that $|P_i|$ read accesses will not be effected. Unlike the interleaving which is at the level of a data tile, this is rather performed at the block level. All the data tiles distributed in the partitioned memories $|M_{i,j}|$ for the task $|T_i|$ are first divided into sets of equal size blocks (each block consist of a set of data tiles), then the data tiles of each block are skewed (interleaved) by a certain value. The data blocks are formed by taking into account the information about the set of tasks T_j and their relevant data tiles $S_{i,j}$ that are scheduled simultaneously on the processor tiles P_j. The block are formed in such a way that each block contain a single data tile $S_{i,j}$ from the scheduled sub-sets of tasks T_j, and to avoid the conflicts, data tiles are then interleaved (skewed) in each block by some value. This way the processor tiles $|P_j|$ can write to the shared memory tiles $|M_{i,j}|$ without any conflict, when ensuring that the corresponding read will not be effected. The block-level data distribution can be determined using the equation below:

$$B_n(x) = n, \textit{ where } 0 \leq n \leq |B_n| \tag{6.2}$$

where *B(x)* represents the block index number, n represents the value of the interleaving (skew factor) and $|B_n|$ represents the total number of blocks. The same conflict-free read/write access order is valid for $|M_{j,i}|$ shared memory tiles except that the read/write access order is just reversed. It is equally possible that during data distribution for resolving the read conflicts, it might also resolve the write conflicts. In such scenario, the second level data distribution would not be needed. Further, the shared partitioned memories can be implemented using Flip-Flop (FF) based registers or embedded SRAM memories. The HP CACTI, a cache and SRAM compiler, is integrated into the DSE framework for memory characterization with different configurations required during DSE. The above strategies and the order in which they can be applied is represented in the form of a flow diagram in Figure 6.3. The above-discussed communication and memory design approach and its strategies will be further explained using as a representative test case the design of LDPC decoders for the future demanding communication system standards.

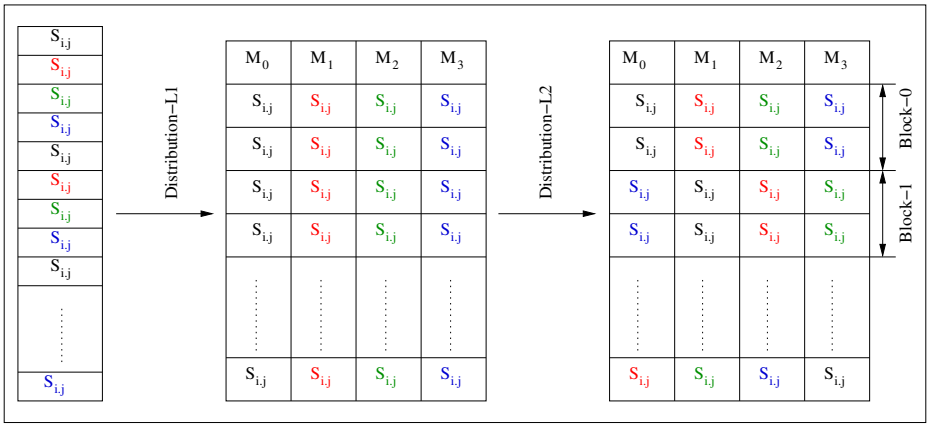

Figure 6.4: Data distribution strategies in multiple shared memories for conflict-free accesses

6.3 Case Study: Communication and Memory Architectures of LDPC Decoders

Practical LDPC codes, such as those adopted in the IEEE 802.15.3c standards for future demanding communication systems, exhibit a very complicated, but not fully random, information flow structure, in which certain regularity and hierarchies are present [3]. According to the communication and memory architecture synthesis method introduced in the previous section, the information flow structure of such an application has to be carefully analyzed. The aim of this analysis is to discover the application regularities and hierarchies in order to exploit them for the design of an effective and efficient communication architecture with (possibly several levels) of hierarchical localized communication clusters. For instance, the practical LDPC codes are defined by block-structured PCMs. A block-structured PCM groups a certain number of rows (CNs) of PCM into a macro-row and the same number of columns (VNs) into a macro-column, creating this way the corresponding macro-entries of the block matrix. For example, 21 rows and columns form a macro-row and macro-column, respectively, for the PCM shown in Table 6.1. The particular macro-entries of this table represent particular sub-matrices corresponding to the particular 21 rows and 21 columns.

The interconnections among particular macro-rows and macro-columns of the block-structured PCM are defined by the non-zero entries (sub-matrices), zero entry '-' means no interconnection. Every macro-row is connected to a different sub-set of macro-columns in a complex pseudo-random way and vice versa. For example, the macro-row {1} is connected to the macro-columns {4, 6, 11, 13, 20, 28, 30} and the macro-column {1} is connected to the macro-rows {2,

Table 6.1: block-structured PCM, H_{base}, of 1/2 rate IEEE 802.15.3c LDPC code with 32 macro-columns and 16 macro-rows, size of each sub-matrix is 21x21 and codelength is 672, '-' represents zero matrices

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
1	-	-	-	5	-	18	-	-	-	-	3	-	10	-	-	-	-	-	-	5	-	-	-	-	-	-	-	5	-	7	-	-
2	0	-	-	-	-	-	16	-	-	-	-	6	-	-	-	0	-	7	-	-	-	-	-	-	-	10	-	-	-	-	-	19
3	-	-	6	-	7	-	-	-	-	2	-	-	-	-	9	-	20	-	-	-	-	-	-	-	-	-	19	-	10	-	-	-
4	-	18	-	-	-	-	-	0	10	-	-	-	-	16	-	-	-	-	9	-	-	-	-	-	4	-	-	-	-	-	17	-
5	5	-	-	-	-	-	18	-	-	-	-	3	-	10	-	-	5	-	-	-	-	-	-	-	-	-	-	-	-	-	7	-
6	-	0	-	-	-	-	-	16	6	-	-	-	0	-	-	-	-	-	7	-	-	-	-	-	-	-	-	-	19	-	-	-
7	-	-	-	6	-	7	-	-	-	-	2	-	-	-	-	9	-	20	-	-	-	-	-	-	-	-	-	-	-	10	-	-
8	-	-	18	-	0	-	-	-	-	10	-	-	-	-	16	-	-	-	-	9	-	-	-	-	-	-	-	-	-	-	17
9	-	5	-	-	-	-	-	18	3	-	-	-	-	-	10	-	-	5	-	-	4	-	-	-	-	5	-	-	-	-	-	7
10	-	-	0	-	16	-	-	-	-	6	-	-	-	0	-	-	-	-	-	7	-	4	-	-	-	-	-	10	-	19	-	-
11	6	-	-	-	-	-	7	-	-	-	-	2	9	-	-	-	-	-	20	-	-	-	4	-	19	-	-	-	-	-	10	-
12	-	-	-	18	-	0	-	-	-	-	10	-	-	-	-	16	9	-	-	-	-	-	-	12	-	-	4	-	17	-	-	-
13	-	-	5	-	18	-	-	-	-	3	-	-	-	-	-	10	-	-	5	-	-	-	-	-	-	-	5	-	-	-	-	-
14	-	-	-	0	-	16	-	-	-	-	6	-	-	-	0	-	7	-	-	-	-	-	-	-	10	-	-	-	-	-	-	-
15	-	6	-	-	-	-	-	7	2	-	-	-	-	9	-	-	-	-	-	20	-	-	-	-	-	19	-	-	-	-	-	-
16	18	-	-	-	-	-	0	-	-	-	-	10	16	-	-	-	-	9	-	-	-	-	-	-	-	-	-	4	-	-	-	-

5, 11, 16}. However, the interconnections within each sub-matrix of the block-structured PCM are defined by regular circularly shifted identity matrices with shift values represented by the non-zero entry in the matrix. Hence, in the corresponding hardware multi-processor, the communication within a single non-zero sub-matrix can be realized locally using a quite regular local communication network, while the communication among the macro-rows and macro-columns is irregular and can be realized using a global communication network, as shown in Figure 6.5. This way, through exploitation of the knowledge of the information flows within the application, the complex irregular fully-flat communication network can be replaced with a much simpler and more regular hierarchical two-level communication network. This substantially decreases the communication network complexity (see Figure 6.7) compared to the case of the fully-flat communication scheme (see Figure 6.6) for different micro-/macro-parallelism combinations. In these and following figures presenting experimental results, P(a, b) denotes a combined micro- and macro-architecture parallelism. In tuple P(a, b), 'a' represents the micro-architecture parallelism of a processor (i.e. the number of processor inputs/outputs), and 'b' represents the macro-architecture parallelism (i.e. the number of processors). The tuple P(a, b) represents a certain micro- and macro-architecture combination with the combined micro- and macro-parallelism (a, b) of the CNP processors, correspondingly (shown on the x-axis in the figures presenting the results). Similar notation for the combined processing parallelism is used for the VNP processors (although, not shown on the x-axis of the resulting figures). As shown in Figure 6.8, the area saving is as high as *25-times* for the architecture instance P(4,336).

The area estimates are very accurate as we perform a prior floorplanning of the top-level design (macro-architecture) and the actual design and physical characterization of various instances of the generic architecture modules (processors, memories and communication resources), when accounting for the interconnect effects during the module characterization. Since the macro-architecture design

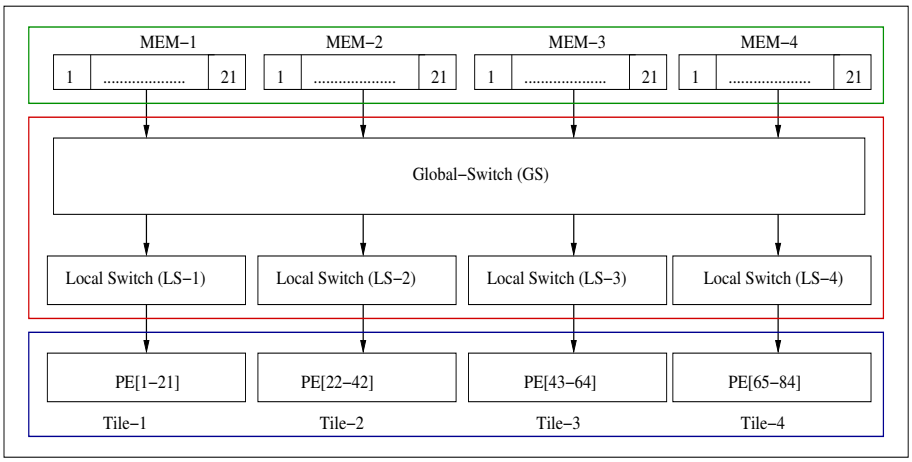

Figure 6.5: Example of the hierarchical communication network of LDPC decoders for IEEE 802.15.3c LDPC code decoder of 1/2 rate (R), code length 672 (L) and (micro, macro) parallelism of (1,84)

(composition of architecture modules to form the accelerator) is very regular and follows the same general structure for all architecture instances, the corresponding floorplan and actual layout are very regular and have almost the same general form for all architecture instances. Therefore, the parameter predictions based on the parameter values for the individual blocks and the floorplan do not much differ from the actual values from the layout both regarding the area and performance estimates. The blocks and the top-level design are modeled in Verilog HDL that can be targeted to various implementation technologies. For performing the experiments reported in this thesis, it has been targeted at CMOS 90nm technology[2]. For blocks characterization (parameters estimations), Cadence Encounter RTL compiler was used for synthesis and Cadence Encounter RTL-to-GDSII system 9.12 for physical place & route.

Moreover, the communication network and memory dominate the processors in area, as shown in Figure 6.7. In particular, for higher processing parallelism, the communication network influence on the area much dominates the processor influence. The processor's contribution to the total area is shown in the dark blue, communication network's contribution in light blue and memory's contribution in magenta in Figure 6.7.

Regarding the performance, except for the low parallelism levels for which the flat scheme performs well, for the moderate and high-level of parallelism the hierarchical two-level interconnect approach provides superior performance. The performance gain is as high as *12-times* for architecture instance P(2,336), as shown

[2]TSMC 90nm LPHP standard Cell Library

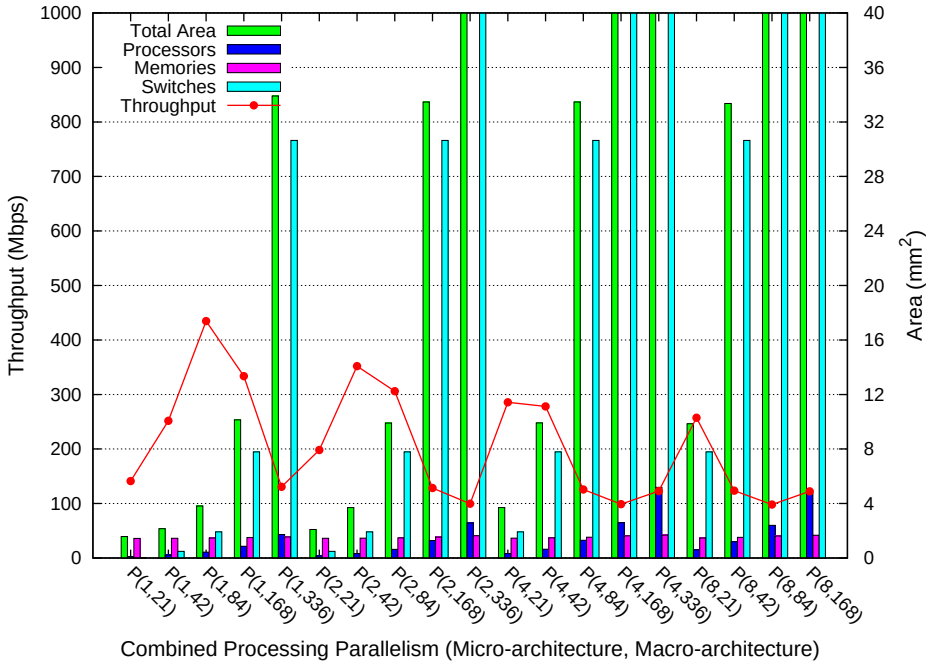

Figure 6.6: Area/Performance tradeoffs for the flat communication network

in Figure 6.9. Moreover, the performance saturates at a certain higher parallelism level for the flat communication scheme, and a drop in performance can be observed by further increase in parallelism, because the switch delays dominate the processor delays. The same trend can be observed for the two-level communication network, but at a different parallelism level (e.g. P(4,336),P(8,84)), as shown in Figure 6.7.

The hierarchical communication network is also very efficient from the viewpoint of power consumption (see Figure 6.10). The power scales well in the massive parallelism regions. For instance, by the *4-times* increase of the micro-architecture parallelism from P(2,168) to P(8, 168), there is only approximately *5-times* increase of power consumption. Similarly, by the *4-times* increase of the macro-architecture parallelism from P(8, 42) to P(8, 168) there is only approximately *5-times* increase of power consumption. As shown in Figure 6.10, the hierarchical communication network delivers savings in power as high as *3-times* compared to the flat communication network.

For both types of communication network, the total power PW_{total} was calculated the same way by summing up their corresponding static PW_{static} and

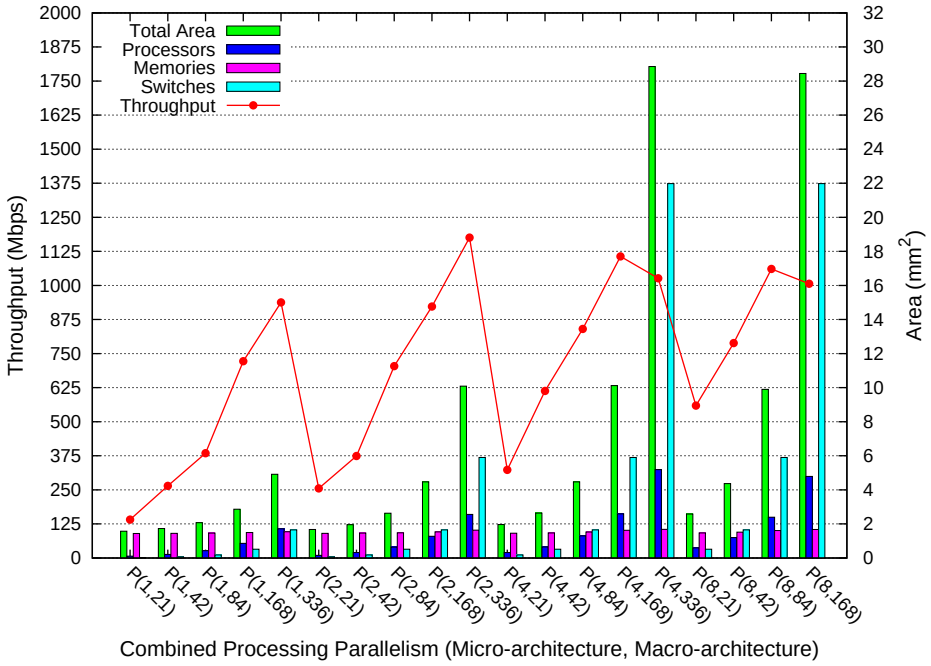

Figure 6.7: Area/Performance tradeoffs for the two-level hierarchical

dynamic $PW_{dynamic}$ power:

$$PW_{total} = PW_{static} + PW_{dynamic} \tag{6.3}$$

The dynamic power was computed using the following formula:

$$PW_{dynamic} = \alpha C V^2 f \tag{6.4}$$

where α is the signal activity factor of the circuit, C the total switching capacitance, V the supply voltage and f the frequency. $\alpha = 0.5$ was assumed.

To be able to obtain the experimental results presented in this section, a large set of most promising hardware multi-processor architectures were synthesized and analyzed. The synthesis and evaluation of such a large set of architecture instances in a reasonably short time was only possible through usage of the automated DSE framework, supported by the accelerator architecture template(s) and a parameterizable generic component library for each component type, such as memories, processors and interconnect networks (switches).

The following four types of memories are involved in the decoding of LDPC codes:

1. M_{cv} to store the CN messages,

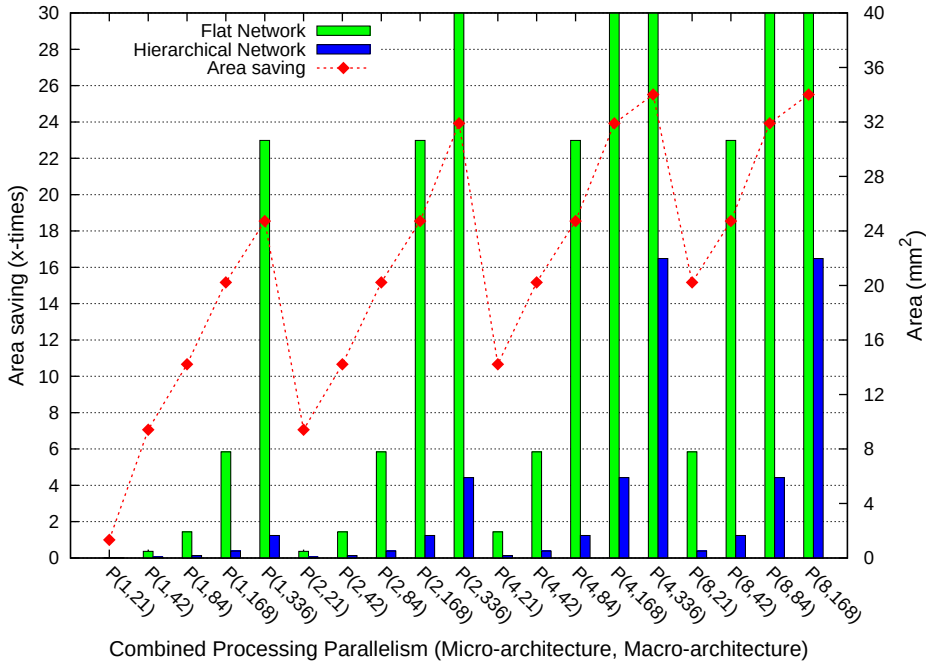

Figure 6.8: Area tradeoffs for the flat vs hierarchical communication network

2. M_{vc} to store the VN messages,
3. M_{ch} to store the channel messages I_{ch}, and
4. M_{HD} to store the hard decision messages V_{HD}.

The check node memories (M_{cv}) and variable node memories (M_{vc}) are shared between the CNP and VNP processors. However, the M_{ch} and M_{HD} are used by the VNP processors for reading of channel data and writing of decoded messages, respectively. The CNP reads the check node data from the M_{cv} and after processing writes the result data to the M_{vc} memories. Similarly, the VNP reads the variable node data from the M_{vc} and after processing writes the result data to the M_{cv}. This represents back-to-back (cyclic) data dependencies between the the CNPs and VNPs. The total amount of data required to be store in M_{cv} and M_{vc} memory is equal to the number of non-zero (NZ_{sm}) elements (sub-matrices) of the block-structured PCM. Since the total number of non-zero elements in the rate-1/2 672-bit IEEE 802.15.3c LDPC code is 108, 108 elements have to be stored in each of the M_{cv} and M_{vc} memories. Please note that a sub-matrix in the block-structured PCM represents the interconnections between the CNs and VNs, and should not be considered as the actual data processed by the CNPs and

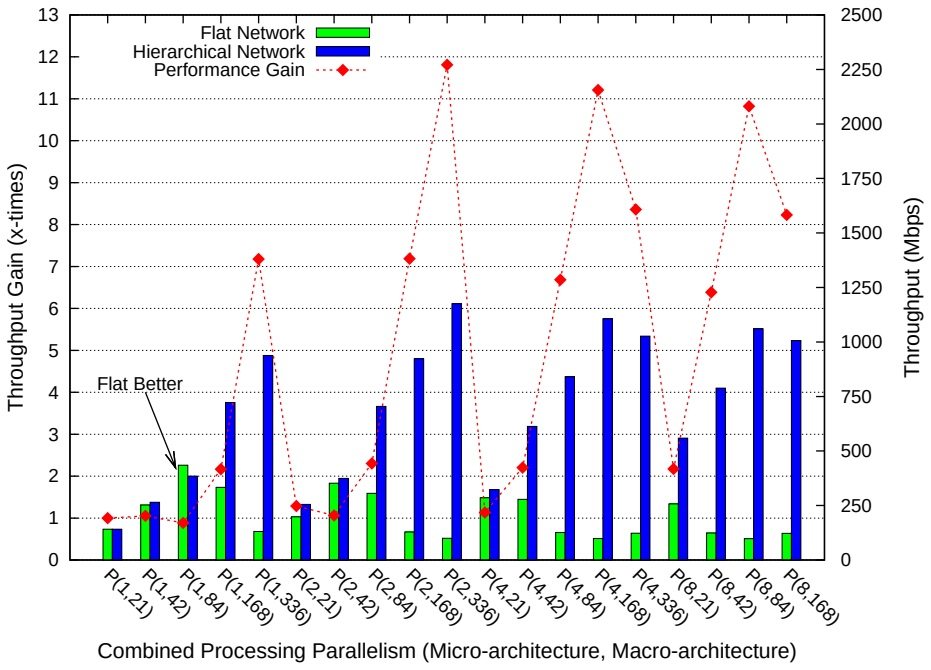

Figure 6.9: Throughput tradeoffs for the flat vs hierarchical communication network

VNPs.

Moreover, for each type of task the data related to each of the non-zero element (sub-matrix) of the block-structured PCM can be stored in a single vector location of a single vector memory, because the sub-matrices actually represent the identity shifted matrices. In the identity matrices, there is only a single non-zero element in each row or column. The width, depth and number (partitions) of each type of memory (M_{vc} or M_{cv}) depend on the processing parallelism exploited for each kind of processors (CNPs and VNPs). For instance, consider the case of the micro-architecture parallelism of four and the macro-architecture parallelism equal to the total sub-matrix parallelism (21 for rate-1/2 672-bit code). In this case, all the data for the CNPs and VNPs can be stored in four vector memories of each kind M_{cv} and M_{vc}. As the macro-architecture parallelism is equal to the total sub-matrix parallelism, and for each sub-matrix the related data can be stored in a single location of a single vector memory, a single vector memory is sufficient to provide the necessary bandwidth for each kind of processors (CNPs and VNPs). Four memory partitions are required as the assumed micro-architecture parallelism is four. This way the aggregate memory bandwidth is satisfied by four partitioned vector memories M_{cv} and M_{vc} for the CNP and VNP, respectively. Concerning the size of the memory partitions, each of the memory partition is of

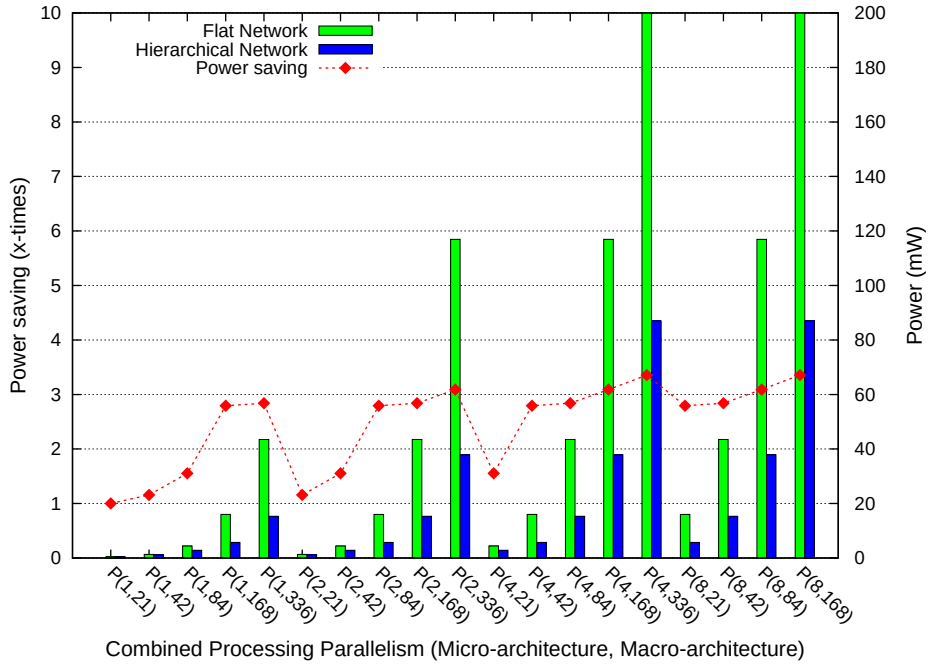

Figure 6.10: Power tradeoffs for the flat vs hierarchical communication networks

depth $\lceil \mathrm{NZ}_{sm}/(4\times1) \rceil$ and width of *sub-matrix* $\times$ b ($=21\times b$), where b represents the data width of CN and VN messages.

The problem yet to be solved is the data distribution and data mapping in the so constructed partitioned shared memories (M_{cv} and M_{vc}) located between the CNPs and VNPs. An adequate resolution of this problem is necessary due to different data access patterns of CNPs and VNPs to the shared memories. The data can be accessed element by element in case of the fully-serial processors or at once in case of the fully-parallel processors. A CNP requires to access the data from the same (macro-)row at the non-zero locations (row-major order), while a VNP requires to access the data from the same (macro-)column (column-major order). For instance, the CNPs of macro-row {1} need to access the data from the non-zero locations {4, 6, 11, 13, 20, 28, 30} of macro-row {1} stored in M_{cv} (see Table 6.1). While the VNPs of macro-column {1} require to access the data from the non-zero locations {2, 5, 11, 16} of macro-column {1} stored in M_{vc}. Both in the case of CNP and VNP, the processed data have to be stored back in M_{vc} and M_{cv} in the same access order as for read, respectively. Thus, the CNPs and VNPs share the M_{cv} and M_{vc} for read and write, but with a different access order. The CNP accesses the shared M_{cv} row-wise for read, while the VNP accesses the same shared memory column-wise for write. On the other hand, the VNP accesses the

shared M_{vc} column-wise for read and the CNP accesses the same shared memory row-wise for write. Thus, in order to ensure that both the read and write accesses for both kinds of processors (CNPs, VNPs) would be conflict-free, data have to be appropriately distributed and mapped in the shared M_{vc} and M_{cv} memories. It is not possible to access the shared memories row-wise and column-wise without any access conflict.

Some approaches have been proposed in the literature to overcome the memory access conflicts for LDPC decoding [66, 70, 71]. These approaches use as many vector memories as the number of non-zero (NZ_{sm}) elements in the block-structured PCM for each task. However, this causes a huge increase in the number of memories, even for the proposed architectures which exploit a low processing parallelism. For the case of IEEE 802.15.3c codes, it would result in as high as 216 memories with a depth of one and width of ($21 \times b$). Also, 32 memories for each of the M_{ch} and M_{HD} messages would be required.

In the proposed method, the memory partitioning is performed depending on the processor parallelism for each architecture instance, as discussed earlier. The data distribution and related data mapping is performed using the approach described in the previous section. The data distribution and related mapping into the shared memories M_{cv} and M_{vc} and among the CNP and VNP tiles are performed taking into account the processing parallelism and the related task mapping. The data partitioning, data distribution and mapping approach provides as many memory partitions as the number of processing tiles. This enormously reduces the number of shared vector memories compared to the proposed approaches [66, 70, 71]. Since the M_{ch} and M_{HD} memories are only accessed by the VNPs, their memory partitioning, data distribution and data mapping are trivial, as these memories are not shared among the different tasks. Due to small sizes of data involved in the decoding of IEEE 802.15.3c LDPC codes, the influence of memories on the overall area and delay remains low in the whole range from centralized and relatively larger sizes to extremely distributed and very small sizes, as shown in Figure 6.7. Nevertheless, adequate memory design, as well as, data distribution and mapping to distributed memories are of primary importance for an effective and efficient communication design. In consequence, even for applications with small data sizes, the memory architecture design remains one of the major design issues.

Recently, several papers on architectures for processing a single sub-matrix of a single macro-row (serial CNP) and a single sub-matrix of a single macro-column (serial VNP) are published. However, processing of a single sub-matrix in isolation only requires a simple local communication network (switch) and a simple memory structure [52, 56, 57, 62]. It does not solve the problem of an effective and efficient processing of the whole PCM matrix and related communication architecture for this aim. Some architectures for processing only a single macro-row and a single macro-column were also proposed that required multiple local communication networks but no global communication network [58, 61, 82]. However, for the demanding accelerator cases, multiple macro-rows and macro-columns have to be

processed in parallel, and this requires the solution of a much more complex system of global and local communication problems. Our solution to these problems represents a generic hierarchical communication network and distributed memory architecture, as shown in Figure 6.5. We propose solutions for the actual total memory and communication problem, and not only for some of its isolated parts, as considered by the published research.

Despite of the local regularity at the level of a single sub-matrix, the global communication network complexity quite quickly increases with the increase of the micro-/macro-parallelism (see Figure 6.7). Therefore, to improve the scalability of the communication architectures for the high-end applications, we proposed partitioning techniques which reduce the complexity of the global inter-tile communication. These techniques are discussed in the following sections.

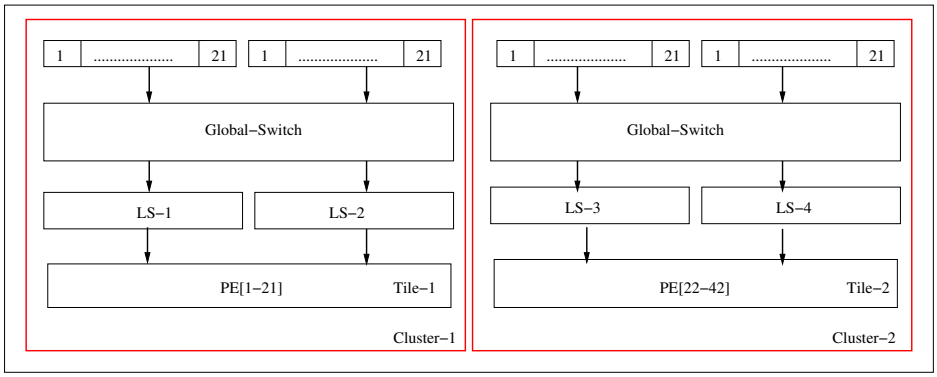

Figure 6.11: Data distribution based communication partitioning

6.3.1 Data Distribution Based Communication Partitioning

The basic idea of this technique is to reduce the data distribution related to a sub-set of node tiles from all memory tiles to a minimum possible number of shared memory tiles. This way, the corresponding sub-set of the processing tiles to which the node tiles are mapped would not be required to communicate with all the shared memory tiles. In consequence, a simpler communication network will be sufficient to communicate a specific sub-set of node tiles to a specific sub-set of shared memories tiles. This way, the single global switch can be partitioned into several smaller switches. A necessary and sufficient condition to be satisfied for those sub-set of node tiles sharing a sub-set of memory tiles is that those sub-set of nodes cannot be scheduled together (although, they can be processed in parallel) due to the memory access conflict.

The above sketched proposed approach will be further explained with respect to a particular accelerator instance with 2 tiles, each with a micro-parallelism of

2, and a particular memory organization with each node tile data is stored in a vector word of a single-port vector memory or a bank of a multi-bank memory, as shown in Figure 6.11. According to the memory partitioning, data distribution and mapping methodology presented in Section 6.2, the data-distribution and assignment to different shared memories is performed in the way to store all the related data in a minimum number of shared memory tiles, while ensuring that the data will be accessed without conflict. The corresponding memory assignment and data mapping for accelerator instance with 2 tiles each with micro-parallelism of 2 is shown in Table 6.2. The subscripts a and b in $s_{a,b}$ represent the data related to a particular check node a and variable node b tiles, respectively. The data related to variable node tiles {1, 2, 3, 4} and {5, 6, 7, 8} are stored in the same memories namely {M_0, M_2} and {M_1, M_3}, respectively, and the micro-parallelism of each tile (2 tiles in total) is 2. Therefore, any two node tiles among the group {1, 2, 3, 4} and {5, 6, 7, 8} cannot be scheduled together, as each tile requires two data elements from the same memory, which is not possible. However, the node pairs {1,5}, {2,6}, {3,7}, {4,8} can be scheduled together (one possible way) as the required data is located in separate memories. Moreover, all data related to variable nodes {1} and {5} are stored respectively in memories {M_0, M_3} and {M_1, M_2}. This way both tiles have to communicate with a subset of shared memories (only 2 in this case) and the corresponding inter-tile global communication network is partitioned into two smaller communication network.

Unfortunately, this approach can not be applied to the case of high parallelism levels. With the parallelism increase, data have to be distributed into several memory banks for parallel access. In such a scenario, the inter-tile communication will become unavoidable. In the next section, we introduce the second partitioning technique to reduce the global communication network complexity and to explore the tradeoffs among the different schemes.

6.3.2 Data Identification Based Communication Partitioning

With increase of the number of tiles, it becomes impossible to avoid the inter-tile communication using the technique presented in the previous section. Data have to be distributed into multiple vector or multi-bank memories for conflict-free parallel access, and in consequence, a global interconnect network is required. However, the global switch complexity increases drastically with the parallelism increase from moderate to high (see Figure 6.7). Moreover, the physical synthesis tools cannot even place and route such a huge single global switch. Therefore, rather than using a single global switch to provide the inter-tile communication, different combinations of several switches and associated switch sizes should be used to reduce the complexity of the inter-tile communication. This decomposition of a huge single global-switch into several much smaller ones can be achieved by taking advantage of the communication and computation hierarchy present in the application, while considering the individual memory and processor tiles. To partition the single global switch, our technique exploit the two-level communica-

Table 6.2: Data distribution and assignment in the partitioned memory (M_{cv}) for an architecture instance with the combined processing parallelism P(2, 42) both for the CNP and VNP processors.

	Check Node Memory Banks (M_{cv})			
Word	M_0	M_1	M_2	M_3
w_0	$s_{1,4}$	$s_{1,6}$	$s_{2,1}$	$s_{2,7}$
w_1	$s_{3,3}$	$s_{3,5}$	$s_{4,2}$	$s_{4,8}$
-	-	-	-	-
w_8	$s_{5,1}$	$s_{5,7}$	$s_{6,2}$	$s_{6,8}$
w_9	$s_{7,4}$	$s_{7,6}$	$s_{8,3}$	$s_{8,5}$
-	-	-	-	-
w_{16}	$s_{9,2}$	$s_{9,8}$	$s_{10,3}$	$s_{10,5}$
w_{17}	$s_{11,1}$	$s_{11,7}$	$s_{12,4}$	$s_{12,6}$
-	-	-	-	-
w_{24}	$s_{13,3}$	$s_{13,5}$	$s_{14,4}$	$s_{14,6}$
w_{25}	$s_{15,2}$	$s_{15,8}$	$s_{16,1}$	$s_{16,7}$
-	-	-	-	-

tion network hierarchy. As in the two-level hierarchical communication network, each one of the memory tiles communicates with one of the processor tiles at a time through a single global switch. Therefore, the communication hierarchies and data flows are further explored at the level of memory banks of a memory tile and at the level of processor of a processor tile, while keeping the two-level hierarchy.

More specifically, by analysis of the specific characteristic of the memory-processor communication patterns, specific sub-sets of banks are identified in different memory tiles that simultaneously communicate with only specific corresponding sub-sets of processors in different processor tiles. In case of the LDPC codes from the case study, the memory banks $\{M_{1,1}, M_{2,1}, M_{3,1}, M_{4,1}\}$ in different memory tiles communicate with a subset of processors $\{P_{1,1}, P_{2,1}, P_{3,1}, P_{4,1}\}$ in different processor tiles, for the architecture instance shown in Figure 6.5. In $M_{a,b}$, a represents a memory number and b bank number, the same way in $P_{a,b}$, a represents a tile number and b processor number. Based on the identified patterns a corresponding application-specific communication among the memories and processing tiles is then realized using several switches of much smaller sizes compared to the huge size of the single global switch. Both the number of switches and the size of each switch are decided by the processing parallelism, i.e. the number

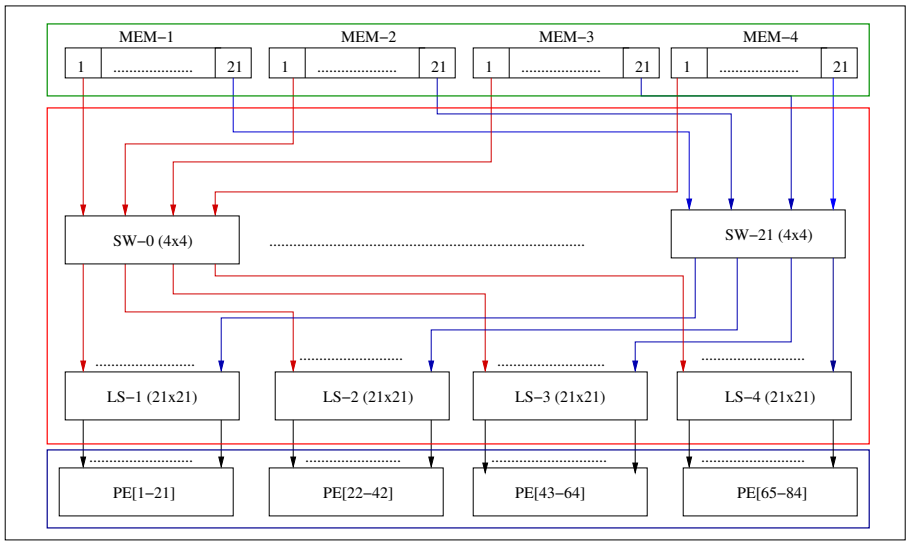

Figure 6.12: Data identification based communication partitioning

of processing elements, parallelism of each processing element (micro-parallelism) and their organization into tiles. For the architecture instance shown in Figure 6.5, the required switch size is 4 (subset of 4 elements), while the required number of switches is 21 (21 subsets of 4 elements), as shown in Figure 6.12. It is worth to note that this approach is only applicable if the processors are organized in tiles and the associated data in the corresponding parallel multi-bank or vector memories. As a general rule for LDPC codes and for a particular micro-/macro-parallelism combination, the number of smaller switches are equal to the tile size (processing elements/tile) and switch size equal to the total number of processing tiles, as shown in Figure 6.12, for the architecture instance of Figure 6.5.

Through this partitioning the complexity of the single inter-tile global switch is much reduced, and the exploitation of the massive-parallelism present in the application is possible at a reasonable cost. The partitioning of the global switches also eliminates the physical synthesis problems. As it can be seen from the results in Figure 6.13, for the high parallelism levels the interconnect complexity remains at a comparable level as for the case of a single global switch (see Figure 6.7). However, much higher gain in performance can be obtained due to the lower interconnect delays of the much smaller switches compared to the non-partitioned global switch. This way a throughput as high as 5 Gbps can be achieved satisfying the peak throughput requirements of example IEEE 802.15.3c codes. After applying the same partitioning technique to the smaller global switch of the data distribution based technique, the required switch size would be half of the original switch size and the number of switches doubles compared to the case when

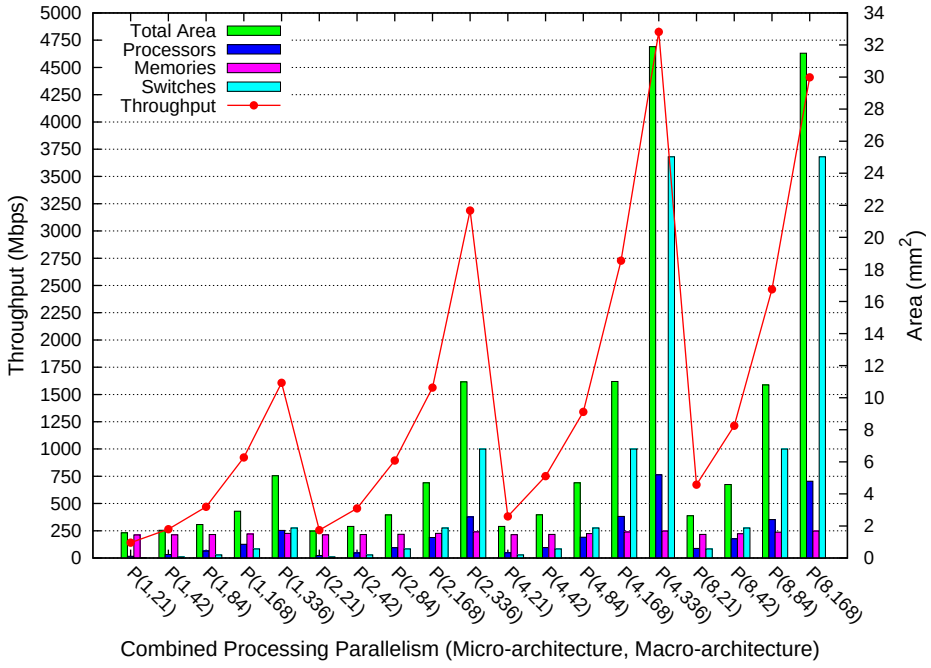

Figure 6.13: Area/Performance tradeoffs for the data identification based communication partitioning

the data identification based partitioning is applied in isolation. This results in different tradeoff regarding area and switch delay. The gains as high as *5-times* in performance can be achieved using the partitioned network compared to the non-partitioned two-level hierarchical communication network, however, with an area penalty of less than *1%*. Nevertheless, the communication network area still dominates the area of processing elements and memories. Therefore, one more technique is introduced below to further reduce the communication network area, while preserving the performance.

6.3.3 Single-stage versus Multi-stage Switches for Communication Network

For each approach discussed so far, ranging from a flat to a hierarchical partitioned two-level communication network, the design tradeoffs can be further explored regarding communication architecture by using different kinds of physical switch networks (e.g. single-stage or multi-stage switches). Moreover, for the two-level hierarchical communication network, the single-stage and multi-stage switches can also be used in combination. For example, single-stage switches can be utilized

for global interconnects and multi-stage switches for local interconnects or vice versa, etc. Furthermore, this approach can be applied to the already partitioned single global switch using one of the earlier introduced partitioning approaches or to a non-partitioned two-level hierarchical single global switch based network. This enable us to explore different design tradeoffs regarding performance, area and power consumption and ultimately ensure the scalability. For this specific kind of exploration, different kinds of switches with diverse parameter ranges (input width and number of input/output ports) are modeled and characterized in TSMC 90nm LPHP Standard Cell Library using the Cadence CAD tool flow. Cadence RTL compiler was used for synthesis and Cadence Encounter for floor planning, placement and routing. All these configuration and characterization is done automatically by a configuration and characterization tool, which is a part of the proposed DSE framework.

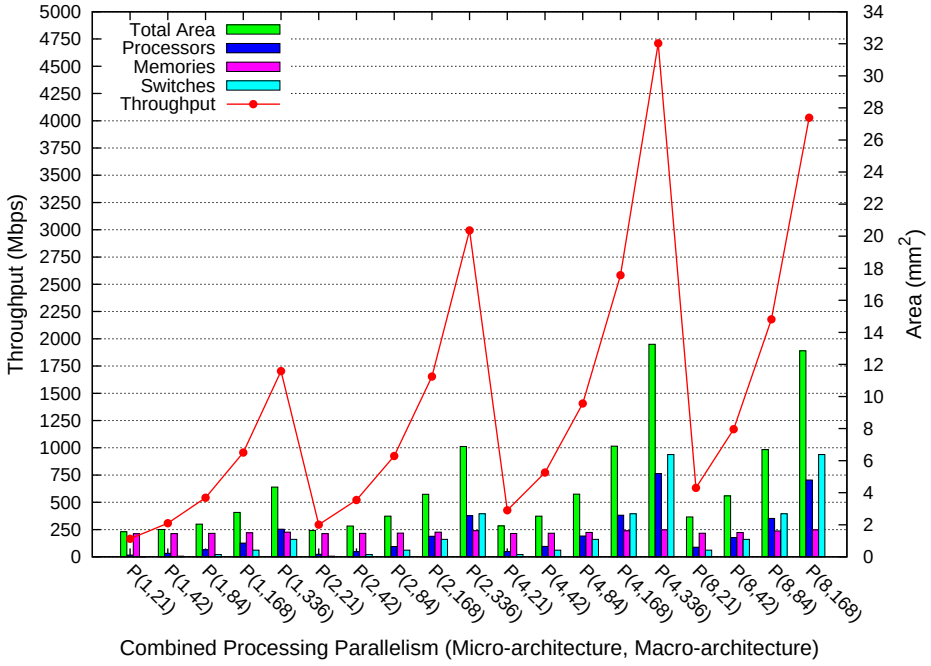

Figure 6.14: Area/Performance tradeoffs for the data identification based communication partitioning when realized using multi-stage switches

The single-stage and multi-stage switches are differently organized in terms of stages, which in turn results in different values for physical switch parameters, such as area and delay, etc. For example, the local intra-tile communication network of LDPC decoders, which is defined by a certain circular shifts matrix,

can be realized using a single-stage multi-input multiplexer (MUX) based shifters or by multiple simple 2×1 multiplexers cascaded in multiple stages (log-shifters).

As concerned resources, the direct implementation of an $N\times N$ shifter would require N multiplexers each of size $N\times1$. However, the same $N\times N$ shifter, when implemented as a multi-stage shifter, would require $Nlog_2N$ multiplexers each of size 2×1. The critical path delay in case of direct implementation is the delay of a single $N\times1$ input multiplexers, while for the multi-stage shifter, the delay would be log_2N, i.e. the number of stages. Similarly, a global switch that provides all permutations can be realized with a Benes network of $N/2(2Nlog_2\text{-}1)$ 2×2 crossbar switches or a direct implementation with N $N\times1$ multiplexers. The critical path delay in case of the direct implementation is the delay of a single $N\times1$ multiplexers, while for the multi-stage shifter the delay would be $2log_2N\text{-}1$ 2×2 crossbar switches, i.e. the number of stages. This simple delay evaluation holds for small switch sizes, but shows a huge deviation for large switches after they are physically placed and routed.

Based on the switch characterization regarding delay, power consumption and area, a set of experiments is conducted for different switch combinations. The results are presented only for the most promising combinations regarding area and performance, i.e. a partitioned two-level communication network. Figure 6.14 shows the results for the area and performance of a partitioned two-level communication network. The local and global switches are realized using multi-stage switches. The partitioned two-level multi-stage (both local and global) communication network outperforms in area the single-stage (both local and global) realization. There is a performance penalty of less than *1%* at most, while a significant area saving as high as *4-times* compared to the case of a partitioned global communication network with single-stage switches. The communication network complexity is much reduced, which in turn ensures a better scalability for massively parallel multi-processor accelerators.

6.3.4 Experimental Results Discussion

A series of experiments is conducted for IEEE 802.15.3c LDPC codes with different micro-/macro-architecture parallelism combinations to explore the numerous complex design tradeoffs, specifically, in relation to the memory and communication architecture. The experiments are performed for the rate-1/2 672-bit IEEE 802.15.3c LDPC code, assuming a high data-precision of 8-bits for communication and 10 iterations per frame. Most of the more specific results of the experiments are discussed in the previous sections to illustrate the memory and communication design approaches. However, the overall view and general conclusions are not less important.

The flat communication scheme can only be used for the low-end applications, because its complexity explodes for the moderate and high parallelism levels, as shown in Figure 6.15. For the moderate parallelism levels, the two-level hierarchical communication network performs well. Compared to the flat network,

it delivers large area and power savings, and performance gains for all the accelerator instances. They can be as high as *25-times* in area, *3-times* in power consumption, and *12-times* in performance. Unfortunately, it shows non-scalable behavior for the massively parallel multi-processor cases, represented in green in Figure 6.15.

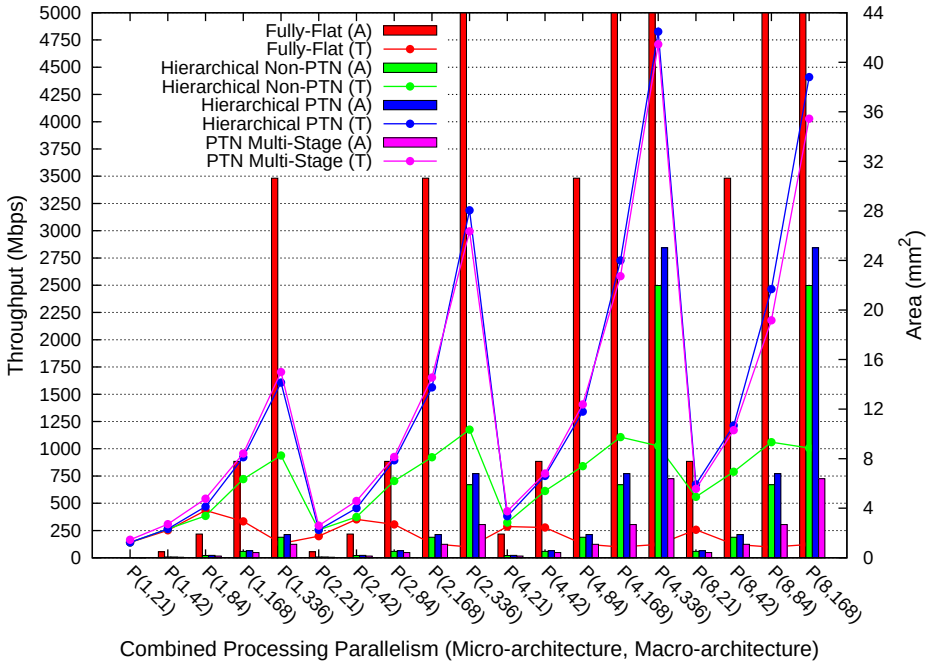

Figure 6.15: Combined results representing interconnect parameters for various processing parallelism

To ensure a much better scalability of the communication architectures for the massively parallel multi-processors, all the partitioning techniques are required to be used in combination. By the combined use of all the partitioning techniques, there is a performance gain for all the accelerator instances as high as *5-times* on the cost of a very small area penalty of less than *1%* compared to the non-partitioned two-level hierarchical communication network (represented in blue in Figure 6.15). Despite the high gains in performance, the communication network area still dominates the area of processing elements for the high parallelism cases. Therefore, multi-stage and single-stage switch combinations should be adequately used in the implementation of the communication architectures to preserve the performance level while reducing the area. The multi-stage/single-stage switch combination approach results in a low performance penalty of less than *1%* at

most, while offering a significant area savings as high as *4-times*.

An interesting observation is that independent of whether the two-level hierarchical communication network is partitioned or not, the application of different combinations of the single-stage and multi-stage realization to local and global switches always results in a superior scalability in one design dimension or in the other design dimension. The experiments on different switch combinations show that the highest performance is achieved for the single-stage global switch and the multi-stage local switches, while for the lowest area the multi-stage realizations are required for both the global and local switches. The same sort of behavior can be observed for the two-level non-partitioned communication network, when the single and multi-stage switch combinations are applied. The results of the experiments show that a partitioned two-level communication network is the most promising one. It improves the scalability of communication architecture in all design dimensions with excellent tradeoffs.

In general, for all the architectures having a low parallelism at both levels and for all communication architectures, both the performance and area scale linearly. This trend is clearly visible (see Figure 6.15) for the combined parallelism of up to 84 for architecture instances such as {P(1, 84), P(2, 42), P(4, 21)}. For the higher parallelism levels beyond these points the flat networks do not scale, but the simple hierarchical communication networks scale to some extent for the moderate parallelism levels, but much worse for slightly higher parallelism levels, and they saturate at the massively parallel micro-/macro-architectures such as P(4, 168).

The partitioning techniques with multi-stage and single-stage switch combinations are necessary to ensure the scalability for high-parallelism levels with many complex design tradeoffs that have to be taken into account depending on the actual requirements of the application. For example, the hierarchical data identification based partitioning with single/multi-stage switch combination provides almost as high as 5 Gbps of throughput with much smaller area compared to the other two-level partitioned single-/multi-stage switch combinations.

Summing up, the above discussed memory and communication architecture design approaches, when applied in combination, ensure a good scalability of the memory and communication architecture for the massively parallel multi-processor hardware accelerators, as well as, enable exploration and exploration of the complex design tradeoffs for the memory and communication architecture in the context of various micro-/macro-architecture combinations. This reconfirms the fact that without having a quality-driven model-based accelerator design approach equipped with such techniques, such complex design tradeoffs for massively parallel multi-processors cannot be explored adequately, and high-quality architectures cannot be designed in a reasonable time.

6.4 Comparison of LDPC Decoder Architectures from Related Research and our Method

In this section, a comparison is performed of the LDPC decoder architectures from related research to architectures constructed by our method MPA-Explorer tool. A meaningful comparison of architectures for the LDPC decoding is a complicated task due to the multi-dimensional space of the characteristic features of LDPC decoders. The major LDPC decoder features include: LDPC code types (e.g. regular or irregular; and random or block codes), rate (R) and code length (L); LDPC decoding algorithm and related error-correcting performance and complexity; number of decoding iterations and the data quantization to achieve a certain Bit-Error-Rate (BER) and Frame-Error-Rate (FER); earlier decoding termination strategies to speed up the decoding process; decoder implementation technology (e.g. various CMOS ASIC and FPGA technologies with different characteristics); processing and data parallelism exploitation scheme; performance required and/or performance, area and power consumption tradeoff targeted.

Each of the above LDPC decoder characteristics can take from several to very many different values, correspondingly. From the above, it should be clear that a given decoder architecture may perform differently and require different area and power when used for codes with different characteristics, different decoding algorithms or implemented with different technologies. Usually, the published state-of-the-art research works [52–82] related to the LDPC decoder architectures compare and evaluate the architecture proposals in relation to the performance, and less often to power consumption and area (PPA). Unfortunately, while evaluating and comparing the architecture proposals based on PPA, the related research works usually do it without paying enough attention to many important aspects of the LDPC decoding. In result, the architecture comparisons are often meaningless. For instance, some of the related research works compare performance of one architecture with another one applied to decoding with a very different number of decoding iterations/frame [52–82]. This does not make much sense, because a lower number of iterations results in a shorter decoding time. Similarly, some of the related research works compare architectures on different types of LDPC codes or for different data quantizations (while the decoder using less number of data bits will have lower area). Also, a direct comparison of architectures implemented with different technologies is questionable. This makes many comparisons presented in the literature to a large degree unfair and of low value. Moreover, the focus of the published research on the LDPC decoders [52–82] is mainly limited to performance, while usually ignoring the other design dimensions (e.g. area and power consumption). This all makes it difficult to evaluate and compare various LDPC architectures.

This section aims at comparing several most advanced LDPC decoding architectures proposed in the published state-of-the-art research to architectures that can be constructed using our multi-processor accelerator architecture explo-

ration and synthesis method and related MPA-Explorer tool. It will compare the architectures in relation to the performance, power and area (PPA) values. To make the comparison more fair and meaningful, it will compare the architectures in relation to some LDPC codes, decoding algorithms, implementation technologies, etc. The architecture exploration and synthesis tool MPA-Explorer is capable to construct different architectures for LDPC decoders with different types of micro- and macro-architecture combinations. In particular, using the MPA-Explorer the architectures proposed by the related state-of-the-art research can be constructed. Therefore, the LDPC decoder architectures proposed in the related LDPC literature are first constructed to find their PPA values. Moreover, to make the comparison more fair and meaningful, instead of comparing architectures for different codes, decoding algorithms and implementation technologies a standard set of parameters is assumed. All architectures are then compared in relation to this standard set. The standard set of parameters used in the comparison is the following: code type (IEEE 802.15.3c), code rate (R=1/2, 7/8), code length (L=672-bit), number of iteration (10), data quantization (8-bit), implementation technology (CMOS 90nm LPHP). After constructing each particular architecture from the related LDPC literature and computing its PPA values, the accelerator design space with the MPA-Explorer tool is initiated with the same throughput required as for this particular architecture from the related literature to explore the different micro- and macro-architecture combinations, and finally to construct the most promising architectures. Afterwards, the results obtained using the DSE are compared to the ones obtained for a particular architecture from the related literature. This enables us to demonstrate the significance of the architecture DSE and show the high-quality of the architectures from our method. The architectures presented in the literature exploit the two extremes of micro-architectures (fully-serial and fully-parallel) with different macro-architectures to satisfy certain throughput requirements. Therefore, the architectures are first sub-divided based on the type of processing parallelism exploited by each of the proposed in the literature architectures and construct the architecture with this particular micro- and macro-architecture as in the architecture from the literature. Then, the DSE tool constructs the most promising architectures for the same level of performance.

Let us now go into details. Several partially parallel architectures have been proposed in the past for the LDPC decoding [53, 55, 56, 58, 61, 62, 72, 81, 82]. However, they deliver a throughput of only a few hundreds of Mbps. The proposed architectures are not adequate for the high-end applications that require throughputs in the ranges of multi-Gbps (4∼6 Gbps). They employ the fully-serial micro-architectures for both the VNPs and CNPs with macro-architecture parallelism limited to the size of the single sub-matrix of the PCM. Moreover, they use a simple communication and memory architectures in the form of simple shifters and vector memories, correspondingly. With the MPA-Explorer tool, the most advanced of these architectures are implemented, namely, the one presented in [55]. However, it delivers a throughput of only 141 Mbps, as shown in Table

6.3. An extension of the architecture approach from [55] is presented in [52]. It is the so-called vectorized LDPC decoder, that exploits the fully-serial micro-architectures for both the CNP and VNP, while the macro-architecture can be as parallel as the maximum parallelism available in the code. In the case of IEEE 802.15.3c code, the macro-architecture parallelism is 672 for VNP and 84, 168 and 336 for CNP of code rates 7/8, 3/4 and 1/2, respectively. However, also the enhanced architecture from [52] is not able to deliver the ultra-high throughput of (4∼6 Gbps) required for the IEEE 802.15.3c code, as shown in Table 6.3. It is mainly due to the large number of computation cycles required to perform the check and variable nodes computations. The second reason is the drop in the accelerator frequency due to the long critical path delay of the large multiplexers used for selecting the particular message memories. The best architecture constructed for this case by our DSE tool provides almost the same performance of 1 Gbps, but with a *1.94-times* lower area and *3.87-times* lower power consumption than the architecture from [52]. Our architecture exploits a micro-architecture parallelism of 2 and macro-architecture parallelism of 84.

Table 6.3: Implementation results and comparison of the existing LDPC decoder architectures with the architectures constructed by our multi-processor accelerator synthesis tool (MPA-Explorer)

Architecture	Code Spec (672, k)	Frequency (MHz)	Throughput (Mbps)	Area (mm^2)	Power (mW)
Brack [55]	k = 336	537	141	1.57	411
Rovini [52]	k = 336	223	938	4.92	5860
Proposed	k = 336	440	924	2.53	1512
Tong [69]	k = 336	369	590	1.86	115
	k = 588	207	332	1.88	95
Zhong [74]	k = 336	305	2926	8.38	298
	k = 588	168	1887	8.42	254
Proposed	k = 336	239	4027	12.84	301
	k = 558	159	2676	12.81	569
Liu [78]	k = 588	119	1001	10.10	1300
Sha [79]	k = 588	475	1995	4.53+168R*	2826
Proposed	k = 558	231	1943	6.86	1326
Proposed	k = 336	280	4709	13.25	3160
Proposed	k = 588	184	3091	13.04	3091

*Pipeline registers for clock speed improvement

Architectures exploiting the fully parallel micro-architectures for both the CNP and VNP have been proposed [66, 69, 73]. The macro-architecture parallelism, i.e. number of CNP and VNP processors, is determined there based

on the number of macro-rows and macro-columns in the block-structured PCM. They consider as many CNPs as the number of macro-rows in the block-structured PCM and as many VNPs as the number of macro-columns. In order to avoid the memory conflicts and to reduce the communication network complexity, they utilize the fully distributed memory structure with as many memories as the number of non-zero entries (NZ_{sm}) in the block-structured PCM. Each of the memories stores the data related to a single sub-matrix. For IEEE 802.15.3c code rate-1/2 LDPC code, it would result 216 single-port memories in total or 108 double-port memories plus 32 memories for channel messages. The architecture proposed in [69] is implemented with our DSE tool. It provides a low throughput of 590 Mbps for rate-1/2 code and 332 Mbps for rate-7/8 code, as can be seen in Table 6.3. An extension of this approach has been proposed in [74–76] that allocate q number of CNPs and VNPs for each of the macro-rows and macro-columns such that $p\%q=0$, where p represents the sub-matrix size and % represents the modulus operator. For IEEE 802.15.3c LDPC code q can be equal to 3 or 7, as the sub-matrix size is 21. The memory architecture is the same as in the previous case, but vector memories are used instead of scalar memories, with vector size equal to q. One of enhanced architectures proposed in [74] is implemented with the MPA-Explorer tool. The architecture proposed in [74] provides a high throughput as high as 3 Gbps compared to the case of architectures exploiting the serial micro-architectures, but has larger area. Nevertheless, this architecture does not satisfy the ultra-high performance of 4 Gbps, while one of our architectures provides a throughput of 4 Gbps at almost the same power consumption, but at a larger area. The main feature of the architecture in [74] is that the global communication network is transformed into point-to-point (P2P) links by the highly fragmented memory structure. However, the distributed memory approach in [74] is only adequate for the short length codes, because the distributed memory architecture of the short length codes have low influence on the total area compared to the global communication network. However, the same architecture from [74] realized for the 4608-bit rate-8/9 code results in the area of 49 mm^2 due to the higher impact of the fragmented memories (320 in number) [74]. Our proposed architecture employ a micro-architecture parallelism of 8 and macro-architecture parallelism of 84, and provides an adequate balance between the complexities of the memory and the communication structures.

Several architecture based on the partially-parallel micro-architecture for CNPs and fully-parallel for VNPs have been proposed in [77–79] for the high rate codes. The architectures in this category are somewhat similar to architectures proposed by us, just due to utilization of partially-parallel micro-architectures. The architecture proposed in [78] employs a partially-parallel micro-architecture for CNP with the micro-parallelism of 8 (in total 32 inputs), and a fully-parallel micro-architecture for VNP, with the macro-architecture parallelism of 84 for each kind of processors (CNP and VNP). However, the throughput achieved is limited to only 1 Gbps, due to the interconnect scheme based on the flat multiplexers and de-multiplexers. The same architecture implemented using our tool exploiting a

hierarchical partitioned network achieves *1.94-times* higher throughput, by *1.43-times* lower area and almost the same power consumption. The architecture proposed in [79] exploits the micro-architecture parallelism of 4 for both CNP and VNP, and the macro-architecture parallelism of 84. The higher throughput of this architecture being 2 Gbps is achieved due to the higher clock speed. However, the architecture requires 672×2 ($= 84 \times 4 \times 2$) 8-bit registers for pipelining[3], what in turn much adds to the total area and further increases power consumption. The high clock speed also results in a higher power consumption. None of the architecture supports the ultra-high (4∼6 Gbps) throughputs. Compared to the last architecture, our architecture provides almost the same performance by *2.13-times* lower power consumption, due to exploitation of a higher micro-architecture parallelism of 8 for CNP compared to the micro-architecture parallelism of 4 in [79], and with some area overhead. However, the small area overhead of our architecture is fully compensated, as the substantial pipeline register contribution is not added to the overall area for the architecture proposed in [79].

From the above comparison, it is clear that our DSE tool MPA-Explorer is able to construct better LDPC decoder architectures than the considered advanced LDPC decoder architectures proposed by the related state-of-the-art research. Moreover, the proposed DSE approach selects the most promising architecture taking into account various application specific design constraints and objectives and realizing required tradeoffs among the objectives. Also, unlike the architectures that utilize the highly distributed memory structures to avoid the memory conflicts, our approach takes into account the processing parallelism, and only then decides the adequate memory and communication structures that provide the necessary bandwidth, as well as, avoid the conflicts with minimum overhead. Furthermore, one of our proposed architectures provides the required ultra-high throughput of 5 Gbps for the rate-1/2 LDPC decoder, and 3 Gbps for the rate-7/8 LDPC code, as shown in Table 6.3. Additionally, our approach is well scalable and enables the construction of various partially-parallel architectures for diverse LDPC codes employed in different standards, with diverse performance, power consumption and area requirements.

6.5 Conclusions

In the former sections, the communication and memory architecture design issues of the massively parallel multi-processor accelerators were discussed that are necessary to realize the required ultra-high throughput of the highly-demanding modern applications. The discussion was focused on the communication and memory bandwidth and scalability issues. It was demonstrated that in the massively parallel hardware multi-processors the memory and communication influence on both the throughput and circuit area dominates the processors influence and the

[3]Two pipeline stages

communication and memory design are strictly interrelated. Therefore, communication and memory architecture design is one of the major aspects of our new accelerator design methodology.

It was demonstrated that the traditionally used simple flat communication architectures and multi-port memories do not scale well with increase of the accelerator parallelism. In consequence, they are not adequate for the massively parallel accelerators. We proposed to design the application-specific hierarchical partitioned organizations of the communication architectures and vectorized memories exploiting the regularity and hierarchy of the actual information flows of a given application. This drastically improves the scalability.

In particular, it was demonstrated that for the moderate parallelism levels the two-level architectures with several local communication-free clusters or a single global cluster perform well, with performance gains as high as *12-times* and area savings as high as *25-times* compared to the flat communication scheme. However, for the high parallelism levels only the partitioned hierarchical approach ensures the high scalability regarding performance with gains as high as *5-times* and a small area penalty of less than *1%* compared to the non-partitioned two-level hierarchical communication network. To further increase the performance or eliminate the area penalty, the multi-stage and single-stage switch combinations can be employed for local and global switches of the two-level partitioned communication network, resulting in area saving as high as *4-times*.

Regarding the memory design, the partitioned vectorized shared (single port) memories seem to be the most promising for the massively parallel hardware multi-processors.

To guarantee the required memory and communication bandwidth, and achieve the communication and memory scalability in the whole considered performance range, all the strategies of communication architecture synthesis, as well as, of memory partitioning, data distribution and related data mapping were incorporated into the multi-processor accelerator design method and related automated architecture exploration framework.

Using this framework a large set of experiments were performed including all the experiments referred to in this thesis. The experiments demonstrated that the proposed quality-driven model-based accelerator design method, and specifically, its memory and communication synthesis techniques, are adequate for the hardware multi-processor design for the modern highly-demanding applications.

Finally, in the last section of this chapter, the LDPC decoder architectures from related research were compared to architectures constructed by our method and MPA-Explorer tool. This comparison shows that the proposed DSE tool MPA-Explorer is able to construct better LDPC decoder architectures than the considered advanced LDPC decoder architectures proposed by the related state-of-the-art research. This reconfirms once more that the proposed quality-driven model-based accelerator design method is adequate for the multi-processor accelerators design for the modern highly-demanding applications.

CHAPTER 7

Conclusions and Future Work

In this chapter, the major conclusions of the research reported in this thesis are presented. Moreover, several promising directions are proposed for future research work.

7.1 Conclusions

The main aim of the research reported in this thesis was to analyze the issues and requirements of hardware accelerator design for the modern highly-demanding applications, as well as, to propose, implement, analyze and evaluate an adequate semi-automatic design method addressing the issues and satisfying the requirements. Below, the major novel contributions to realize the main aim of this research project are presented.

First of all, we thoroughly analyzed several modern highly-demanding applications, and discovered and analyzed the main issues and challenges of architecture design for these applications, and specifically the issues that can not be resolved using the traditional architecture design methodologies for hardware accelerators. The analysis showed that many of the modern highly-demanding applications involve algorithms: with complex interrelationships among the data and computing operations at the task level and complex inter-task data dependencies; complex multi-input multi-output (MIMO) operations; massive data parallelism or task-level functional parallelism; and impose ultra-high performance demands, as well as, different requirements in relation to energy, area and other parameters.

Moreover, they often require an adaptable accelerator design accounting for the design-time or field-use adaptation.

Based on the results of this analysis, the requirements were formulated that have to be satisfied by an adequate design methodology for such applications. In brief, the accelerator design for such kind of applications has to involve both the micro- and macro-architecture design for the processors, and the corresponding adequate memory and communication architectures design. Moreover, the processors micro-/macro-architecture and the memory and communication architectures are strongly interrelated and their design cannot be performed in separation. Complex mutual tradeoffs have to be resolved among the processor parallelism at the two levels, i.e. between the micro- and macro-architecture, and the corresponding memory and communication architectures, as well as, between the performance, power consumption and area. However, the earlier proposed and used for accelerator design HLS methods and related EDA tools only support the micro-architecture synthesis of a single processing unit, while not taking into account the macro-architecture, memory and communication synthesis and not accounting for the relationships and tradeoffs among these design aspects, which is necessary in the design of hardware accelerators for highly-demanding applications. Without considering in combination the micro- and macro-architectures for processors, and the corresponding memories and communication architectures, as well as, the mutual tradeoffs among these design dimensions, and among the performance, power consumption and area, it would be impossible to guarantee a high-quality accelerator architecture. No adequate design methodology was in place for such kind of complex multi-processor accelerators required for the highly-demanding applications. The lack of an adequate design methodology resulted in a large number of ad-hoc proposed solutions in the form of various particular ad-hoc point architectures for various problem instances with different throughput requirements.

The major contribution of the research reported in this thesis is a novel quality-driven model-based multi-processor accelerator design methodology supported by a novel multi-objective and multi-dimensional design space exploration (DSE) framework, that addresses the issues and satisfies the requirements of the hardware multi-processor accelerators for highly-demanding applications. To our knowledge, despite a more than a decade of research on the hardware accelerators no similar holistic quality-driven design approach has been proposed. The methodology is quality-driven and model-based. It exploits the concept of a generic architecture platform, modeled using generic architecture templates, and is supported by a novel multi-objective and multi-dimensional DSE framework.

The proposed multi-objective and multi-dimensional design space exploration framework performs an effective and efficient exploration and exploitation of various tradeoffs between the processing parallelism at the micro- and macro-architecture level, and the corresponding memory and communication architectures, as well as, among the performance, power consumption and area, to arrive at high-quality accelerator architectures. The DSE framework is multi-dimensional in the sense

that it considers jointly the processor, memory and communication sub-systems, as well as, the mutual complex tradeoffs among them. It is multi-objective in the sense that it can target various performance, power consumption and area constraints/objectives and tradeoffs among them through controlling the basic parameters that influence them. *To enable this, the proposed DSE framework is equipped with several novel scheduling, processing parallelism exploration, and memory and communication architecture exploration strategies. They make it possible to explore effectively and efficiently the numerous complex mutual tradeoffs between the micro- and macro-architecture, and the corresponding memory and communication architectures, as well as, among the performance, power consumption and area.* The DSE algorithm performs the exploration and decision-making for each of the architecture design aspect in a parallel constructive way, while taking into account the design constraints and optimization objectives. This way, it ensures not only the high-quality of the constructed architectures, but also reduces the complexity of the design decision search.

Using the generic processor architecture templates, which constitute a part of the proposed design methodology, numerous architectures involving different combinations of processing parallelism at the micro- and macro-architecture level can be instantiated. *This enables to explore the numerous mutual complex tradeoffs between the micro- and macro-architecture levels and their influence on design parameters like, performance, power consumption, area, etc, to make adequate decisions on the number and type of processors.* Moreover, it enables to tradeoff one design objective against the others, while meeting the design constraints.

Two novel operation scheduling techniques, tight scheduling (TS) and relaxed scheduling (RS), are proposed to tradeoff the processors cost against the memory and communication structures costs, dependent of whether the processors cost dominates or the memory and communication structures costs, respectively. This is different than the traditional scheduling techniques that mainly aim at the processor sub-system optimization as the overall design objective. *Moreover, several novel memory and communication architectures are proposed that ensure the scalability of the memory and communication architectures for the massively parallel multiprocessor accelerators required for the highly-demanding applications.*

The design and implementation of the generic template and its modules is one of the most difficult and time-consuming process of the proposed design methodology, as it involves the design of optimized parameterizable processing units, communication and memory elements and their implementation in Verilog HDL, which corresponds to the generic structure models for an application class. The generic models can then be synthesized for different set of parameters delivering various possible solutions with different qualities, which enables an adequate DSE. Physical characterization of the generic models enables quick and quite accurate evaluation and selection of the constructed architectures. To analyze and evaluate the proposed design methodology and its related DSE framework, a series of extensive case studies are performed through implementing and applying the methodology to an industrial-strength application of LDPC decoding. The

performed analysis and evaluation clearly confirmed that the methodology adequately supports the design of complex multi-processor hardware accelerators. *To enable experimental research, the top-level generic architecture and the novel generic check node processor (CNP) and variable node processor (VNP) micro-architectures are proposed that span the full range of micro-architectures from the fully-serial to the fully-parallel, with in between a large number of partially-parallel architectures.* This is different than in the state-of-the-art published research that only considers the trivial cases of the fully-serial or the fully-parallel micro-architectures.

Through the problem and experimental result analysis, it is clearly demonstrated that without considering the micro- and macro-architecture design in combination, it would be very difficult to arrive at an adequate high-quality accelerator. These considerations were supported by experiments and case studies with the LDPC decoder design for IEEE 802.15.3c LDPC codes for the future highly-demanding communication systems. The experiments were focused on exploration of the micro-/macro-architecture tradeoffs regarding the performance, power consumption and area. They clearly demonstrated that there exist various complex tradeoffs at the micro-/macro-architecture level that could only be resolved through an adequate DSE involving a combined micro- and macro-architecture exploration. *In particular, it is demonstrated that neither the fully-serial nor the fully-parallel micro-architectures are adequate to satisfy the ultra-high performance requirements. To satisfy the ultra-high performance requirements, the combined micro-/macro-architecture exploration is necessary which explores and exploits various partially-parallel architecture combinations.* Usually, pipelining is employed to improve the clock speed and this way enhance the performance. Surprisingly, it is not adequate in many cases. Specifically, for the architectures exploiting massive parallelism at both architecture levels. It is not adequate due to the excessively high cost and power penalty. Hence, pipelining should be exploited in only some cases when it delivers substantial improvement on acceptable costs. This kind of tradeoffs are also discussed and highlighted in the micro-/macro-architecture parallelism exploration experiments.

Moreover, two novel performance and power optimization approaches are proposed based on the combined consideration of the micro-/macro-architecture level parallelism. The proposed approaches are much better than the traditional macro-architecture level power and performance optimization approach. With the proposed approach the performance requirements are satisfied without overdimensioning of the system, through taking into account the influence of parallelism at both architecture levels on the overall system costs. Considering the macro-architecture or micro-architecture independently under a performance constraint may unnecessarily increase the system cost, because it is possible that the performance might be marginally not met. Increasing the macro-architecture parallelism in this situation would be a costly decision. As the micro-architecture design also influences both the performance and cost, but it does it with different tradeoffs between the two aspects than at the macro-architecture level, this can be

resolved in a much better way when performing an adequate micro-architecture enhancement. This may restrain the system from over-dimensioning, while meeting the performance constraint with no or relatively lower increase in costs. This way the costs would be adequately tuned to the performance, which is only possible through the joint micro-/macro-architecture parallelism consideration.

Through the combined micro-/macro-architecture parallelism exploration, the proposed approach provides much better power optimization and power/area tradeoffs under a certain throughput constraint than the traditional approach, and additionally with a very low area overhead. Unlike the traditional power (optimization) reduction technique just by increasing the number of processors, the power/area tradeoff optimization under a certain performance constraint can also be achieved by exploiting processors with different micro-parallelism. In particular, one can consider processors with different micro-parallelism, while keeping the macro-parallelism the same, but in general, various micro-/macro-parallelism combinations should be explored for this aim. This way, substantially higher power reduction can be obtained at a lower area cost than what offered by the traditional parallelism/power tradeoff approach. Please observe that high macro-parallelism is accompanied by a high increase in the static power, that for the latest nano-meter technologies has a growing contribution compared to the dynamic power. *This conclusion also supplements the current trends in the industry of migrating from the homogeneous system architectures towards more and more heterogeneous architectures to overcome the performance and energy crises.*

Moreover, the combined micro-/macro-architecture exploration provides an adequate balance between the various design objectives through the orchestrated partial parallelism exploitation at both architecture levels. In particular, if area is the main design objective then the serial micro-architectures combined with adequate macro-architectures perform the best, provided the performance constraint is satisfied. However, if power consumption minimization is the design objective, then the fully-parallel micro-architecture performs the best, provided the performance constraint is meet. An adequate partial parallelism exploitation at both architecture levels for a given application requirements delivers the best tradeoff among all the design objectives.

One of the major challenges addressed in this thesis is the bandwidth and scalability issues of the communication and memory architectures of the massively parallel hardware multi-processors that are necessary for the implementation of highly-demanding applications. For the massively parallel hardware multi-processors, the traditionally used flat communication architectures and multi-port memories do not scale well, and the memory and communication network influence on both the throughput and circuit area dominates the processors influence. To resolve the problems and ensure scalability, we proposed to design highly optimized application-specific hierarchical and/or partitioned communication and memory architectures through exploring and exploiting the regularity and hierarchy of the actual data flows of a given application. The experimental results demonstrate that the proposed method and its DSE framework are able to effi-

ciently synthesize well scalable memory and communication architectures even for the high-end multi-processors. The gains as high as *12-times* in performance and *25-times* in area can be obtained when using the hierarchical communication networks instead of the flat networks. However, for the high parallelism levels only the partitioned approach ensures the scalability in performance. By the combined use of all the partitioning techniques there is a performance gain for all the accelerator instances as high as *5-times* on the cost of a very small area penalty of less than *1%* compared to the non-partitioned two-level hierarchical communication network. The multi-stage/single-stage switch combination approach results in a low performance penalty of less than *1%* at most, while offering a significant area savings as high as *4-times*.

Altogether, the proposed quality-driven model-based design methodology and its multi-objective and multi-dimensional design space exploration pave a novel and adequate way to the design of hardware multi-processor accelerators for highly-demanding applications. According to our knowledge, the so formulated accelerator design problem and its proposed above solution are not yet considered in any of the previous works related to hardware accelerator design. The research work reported in this thesis will provide adequate means for the design of high-quality hardware accelerators for the next generations of highly-demanding complex embedded applications. *Many of the proposed concepts and techniques of the combined micro-/macro-architecture exploration, scheduling, optimization and tradeoff exploration, as well as, the memory and communication architectures design strategies, can also be used directly or after some modifications for the design, exploration and synthesis of application-specific programmable multi-processor systems, as for instance MPSoCs based on ASIPs.*

7.2 Future Work

In this thesis, a novel design methodology is presented for the hardware multi-processor accelerators for highly-demanding applications. The experimental results show the adequacy of the design methodology, what opens some new research directions and opportunities for the design of application-specific systems. Below, some of the new promising research direction are briefly discussed, as well as, some of the new issues and challenges that have to be addressed to forward with the research on massively parallel multi-processor accelerators based on the proposed novel architecture design methodology.

Applying the proposed Design Methodology to other Application Domains

The proposed design methodology is implemented, analyzed and evaluated for the multi-processor accelerators design for the LDPC decoder applications. In a similar way, it can be applied to many other highly-demanding applications fields, e.g. different kinds of encoding/decoding in image processing and multimedia,

medical image processing, 3D graphics, UHDTV, encryption applications, etc, that require massively parallel hardware accelerators to satisfy their performance demands. Application of the methodology to another domain only requires development and characterization of domain-specific generic templates for this domain.

Evaluation of Run-time (Re-)configurable Hardware Accelerators

A part of the proposed design methodology is related to the run-time reconfigurable multi-processor accelerators. A design method was proposed for the run-time reconfigurable hardware accelerators, but the actual implementation and evaluation of this part of the method is not yet performed.

High-Level-Synthesis Tools Integration

Since the micro-architecture synthesis for hardware processors is one of the main part of the proposed multi-processor accelerator design methodology, HLS tools could be integrated into the architecture exploration and synthesis framework for the micro-architecture synthesis of individual processing elements. However, the proposed architecture design flow is based on the concept of scalable generic micro-architectures. Therefore, the actual integration requires to extend the existing HLS methods and tools to support the design of scalable generic micro-architectures.

Three-dimensional Technology for Massively Parallel Multi-processors

Three-dimensional (3D) stacking of chips not only offers a smaller physical package but also shortens wires in the stacked dies using through-silicon vias (TSVs), which can allow higher performance. Therefore, the future 3D chip technology may be beneficial to overcome the long interconnect delays of the massively parallel hardware multi-processors for highly-demanding applications involving hundreds of processors and the related memory and communication structures. The proposed methodology can be easily adopted for the 3D systems. To take the full-benefit of the upcoming 3D technology an adequate architecture floorplanning is mandatory to decide the processor, memory and communication stacks, as well as, the processor, memory and communication elements placements on each stack. The 3D technology seems to be one of the most promising technologies to address and realize complex communication structures among the processors and processors and/or processors and memories at the physical level of the design for the applications involving complex information flows. For example, for LDPC decoders the placement of the two kinds of processors, each type on one stack, sandwiched by a pair of memory and communication stacks, can be used as one floorplan proposal to overcome the long interconnects and associated delays among the various processors and memories when realized in a 3D chip technology.

In the above, some directions are suggested for further investigation or exploration of the proposed design methodology, together with some of the upfront known issues that have to be addressed in the architecture exploration and synthesis of the massively parallel hardware multi-processors for the future highly-demanding applications.

Appendix-A

The algorithms given below describe the tight (overlap) scheduling of the check nodes (CN) and variable node (VN) computations for the LDPC decoding application. The proposed algorithms are based on the idea of the permutation of rows and columns of the parity check matrix (PCM) of a given LDPC code to find an overlapped schedule of the check and variable node computations. These algorithms permute the rows and columns of the given PCM in such a way as to create the empty spaces (zero-entries) in the bottom-left and top-right corner of the PCM. The zero-entries at the top-right corner of the permuted PCM for the check nodes and at the bottom-left for the variable nodes means that all the required data for a single or a set of check/variable nodes is available, and their processing can be stared earlier than waiting for the remaining nodes of other type to finish their execution. Table 1 shows the result of the row/column permutation for the rate-1/2 672-bit IEEE 802.15.3c LDPC code, which create zero-entries at the top-right and bottom-left of the PCM. The rows and columns can be permuted in different orders, i.e. first permuting the rows and then the columns or vice versa. These different orders can result in different ratios of overlapping. The algorithm 1 is the main algorithm that applies in different orders the algorithms 2 and 3 for rows and columns permutation of the parity check matrix. Algorithm 2 performs the rows permutation, while the algorithm 3 performs the columns permutation. Algorithms 2 and 3 are illustrated in more detail in algorithms 4 and 5, respectively. Since practical LDPC codes consist of huge sparse parity check matrices, therefore these algorithms are implemented in MATLAB (Release R2010b) due to the efficient processing of matrix data structures in MATLAB.

Algorithm 1: Tight scheduling Algorithm for LDPC Decoding

Input $\rightarrow H(r, v)$
Output $\leftarrow cnode_{seq}, vnode_{seq}, Overlap, H_p$
Initialize $cnode_{seq} = \emptyset, vnode_{seq} = \emptyset$
Select *processing rows first (RF) or columns first (CF)*
if (RF) **then**
    compute RPM from H // see algorithm 2 for details
    compute CPM from RPM // see algorithm 3 for details
    assign CPM to H_{RF}
end if
if (CF) **then**
    compute CPM from H // see algorithm 3 for details
    compute RPM from CPM // see algorithm 2 for details
    assign RPM to H_{CF}
end if
$compute_{diag}(H_{RF}$, Olap(H), Olap(V)) // zero entries for H_{RF} diagonally from top right (Olap(H)) and bottom left (Olap(V)) corner
$Overlap(RF) = Olap(H) + Olap(V)$
$compute_{diag}$ (H_{CF}, Olap(H), Olap(V)) // zero entries for H_{CF} diagonally from top right (Olap(H)) and bottom left (Olap(V)) corner
$Overlap(CF) = Olap(H) + Olap(V)$
if $(Overlap(RF) > Overlap(CF))$ **then**
    Overlap = Overlap(RF) // maximum value
    $H_p = H_{CF}$ // H_p represents the permuted H(r,v) matrix
else
    Overlap = Overlap(CF) // maximum value
    $H_p = H_{VF}$ // H_p represents the permuted H(r,v) matrix
end if

Table 1: Block-structured PCM of the rate-1/2 672-bit IEEE 802.15.3c LDPC code with the reordered 32 macro-columns and 16 macro-rows for the overlap execution of CNs and VNs. The waiting times are shown by the grey regions of the PCM, size of each sub-matrix is 21x21, '-' represents zero matrices

	30	32	4	6	11	16	29	31	20	3	5	10	27	13	28	26	15	17	19	1	7	12	18	25	21	2	8	9	14	22	23	24
1	7	-	5	18	3	-	-	-	5	-	-	-	-	10	5	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
8	-	17	-	-	-	-	-	-	9	18	0	10	-	-	-	-	16	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
7	10	-	6	7	2	9	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	20	-	-	-	-	-	-	-	-	-
2	-	19	-	-	-	0	-	-	-	-	-	-	-	-	-	10	-	-	-	0	16	6	7	-	-	-	-	-	-	-	-	-
13	-	-	-	-	-	10	-	-	-	5	18	3	5	-	-	-	-	-	5	-	-	-	-	-	-	-	-	-	-	-	-	-
3	-	-	-	-	-	-	10	-	-	6	7	2	19	-	-	-	9	20	-	-	-	-	-	-	-	-	-	-	-	-	-	-
14	-	-	0	16	6	-	-	-	-	-	-	-	-	-	-	-	0	7	-	-	-	-	-	10	-	-	-	-	-	-	-	-
9	-	7	-	-	-	-	-	-	-	-	-	-	-	-	-	5	10	-	-	-	-	-	5	-	4	5	18	3	-	-	-	-
10	19	-	-	-	-	-	-	-	7	0	16	6	-	-	10	-	-	-	-	-	-	-	-	-	-	-	-	-	0	4	-	-
12	-	-	18	0	10	16	17	-	-	-	-	-	4	-	-	-	-	9	-	-	-	-	-	-	-	-	-	-	-	-	-	12
6	-	-	-	-	-	-	19	-	-	-	-	-	-	0	-	-	-	-	7	-	-	-	-	-	-	0	16	6	-	-	-	-
4	-	-	-	-	-	-	-	17	-	-	-	-	-	-	-	-	-	-	9	-	-	-	-	4	-	18	0	10	16	-	-	-
5	-	-	-	-	-	-	-	7	-	-	-	-	-	-	-	-	-	5	-	5	18	3	-	-	-	-	-	-	10	-	-	-
16	-	-	-	-	-	-	-	-	-	-	-	-	-	16	4	-	-	-	-	18	0	10	9	-	-	-	-	-	-	-	-	-
15	-	-	-	-	-	-	-	-	20	-	-	-	-	-	-	19	-	-	-	-	-	-	-	-	-	6	7	2	9	-	-	-
11	-	-	-	-	-	-	-	10	-	-	-	-	-	9	-	-	-	-	20	6	7	2	-	19	-	-	-	-	-	-	4	-

Algorithm 2: Tight scheduling Algorithm for LDPC Decoding (PCM Row-based Permutation)

Input $\rightarrow matrix(r, v)$
Output $\leftarrow cnode_{seq}, RPM$
Initialize $cnode_{seq} = \emptyset$
for all i such that $1 \leq i \leq r$ **do**
  compute LZ_{row} // represents consecutive zeros to the most left of a row i of H/CPM
  compute RZ_{row} // represents consecutive zeros to the most right of a row i of H/CPM
  $D_{row}(i) := LZ_{row}(i) - RZ_{row}(i)$
  $m := i - 1$
  for all m such that $m \geq 1$ **do**
    if $(D_{row}(i) < D_{row}(m))$ **then**
      Insert *i into* $cnode_{seq}$ *before m*
    else if $(D_{row}(i) = D_{row}(m))$ **then**
      $insert_{row}$(***Insert*** i $cnode_{seq}$ before/after m) // $insert_{row}$ represent a function that computes whether i should be inserted before or after m
    else
      Insert *i into* $cnode_{seq}$ *after m*
      go to step 4
    end if
  end for
end for
RPM = permute(matrix(r, v), $cnode_{seq}$*)* // permute the matrix(r, v) rows according to the $cnode_{seq}$ and passed to the main algorithm (1)

Algorithm 3: Tight scheduling Algorithm for LDPC Decoding (PCM Column-based Permutation)

Input $\rightarrow matrix(r, v)$
Output $\leftarrow vnode_{seq}, CPM$
Initialize $vnode_{seq} = \emptyset$
for all j such that $1 \leq j \leq v$ **do**
  compute LZ_{col} // represents consecutive zeros from the top of a column j of RPM/H
  compute RZ_{col} // represents consecutive zeros from the bottom of a column j of RPM/H
  $D_{col}(j) := LZ_{col}(j) - CZ_{col}(j)$
  $k := j - 1$
  for all k such that $k \geq 1$ **do**
    if $(D_{col}(j) < D_{col}(k))$ **then**
      Insert *j into* $vnode_{seq}$ *before k*
    else if $(D_{col}(j) = D_{col}(k))$ **then**
      $insert_{col}$(***Insert*** j $vnode_{seq}$ before/after k)
    else
      Insert *j into* $vnode_{seq}$ *after k*
      go to step 4
    end if
  end for
end for
CPM = permute(matrix(r, v), $vnode_{seq}$*)* // permute the matrix(r, v) columns according to the $vnode_{seq}$ and passed to the main algorithm (1)

Algorithm 4: Tight scheduling Algorithm for LDPC Decoding (Details of Algorithm 2)

Input $\rightarrow matrix(r, v)$
Output $\leftarrow cnode_{seq}, RPM$
Initialize $cnode_{seq} = \emptyset$
for all i such that $1 \leq i \leq r$ **do**
  compute LZ_{row} // represents consecutive zeros to the most left of a row i
  compute RZ_{row} // represents consecutive zeros to the most right of a row i
  $D_{row}(i) := LZ_{row}(i) - RZ_{row}(i)$
  $j := i - 1$
  for all j such that $j \geq 1$ **do**
    if $(D_{row}(i) < D_{row}(j))$ **then**
      Insert *i into* $cnode_{seq}$ *before j*
    else if $(D_{row}(i) = D_{row}(j))$ **then**
      if $(D_{row}(i) < 0)$ **then**
        if $(LZ_{row}(i) < LZ_{row}(j))$ **then**
          Insert *i into* $cnode_{seq}$ *before/**after** j*
        else if $(LZ_{row}(i) = LZ_{row}(j))$ **then**
          Insert *i into* $cnode_{seq}$ *before/**after** j*
        else
          Insert *i into* $cnode_{seq}$ ***before**/after j*
        end if
      else if $(D_{row}(i) = 0)$ **then**
        if $(LZ_{row}(i) < LZ_{row}(j))$ **then**
          Insert *i into* $cnode_{seq}$ *before/after j*
        else if $(LZ_{row}(i) = LZ_{row}(j))$ **then**
          Insert *i into* $cnode_{seq}$ *befor/after j*
        else
          Insert *i into* $cnode_{seq}$ *before/after j*
        end if
      else
        if $(LZ_{row}(i) < LZ_{row}(j))$ **then**
          Insert *i into* $cnode_{seq}$ ***before**/after j*
        else if $(LZ_{row}(i) = LZ_{row}(j))$ **then**
          Insert *i into* $cnode_{seq}$ *before/after j*
        else
          Insert *i into* $cnode_{seq}$ *before/**after** j*
        end if
      end if
    else
      Insert *i into* $cnode_{seq}$ *after j*
      go to step 4
    end if
  end for
end for
RPM = permute(matrix(r,v), $cnode_{seq}$*)* // permute the matrix(r,v) rows according to the $cnode_{seq}$

Algorithm 5: Tight scheduling Algorithm for LDPC Decoding (Details of Algorithm 3))

Input $\rightarrow matrix(r, v)$
Output $\leftarrow cnode_{seq}, vnode_{seq}$
for all i such that $1 \leq i \leq n$ **do**
  compute LZ_{col} // represents consecutive zeros from the top of a column j
  compute RZ_{col} // represents consecutive zeros from the bottom of a column j
  $D_{col}(j) := LZ_{col}(j) - CZ_{col}(j)$
  $k := k - 1$
  for all k such that $k \geq 1$ **do**
    if $(D_{col}(j) < D_{col}(k))$ **then**
      Insert j *into* $vnode_{seq}$ *before* k
    else if $(D_{col}(j) = D_{col}(k))$ **then**
      if $(D_{col}(j) < 0)$ **then**
        if $(LZ_{col}(j) < LZ_{col}(j))$ **then**
          Insert j *into* $vnode_{seq}$ *before/**after*** k
        else if $(LZ_{col}(j) = LZ_{col}(k))$ **then**
          Insert j *into* $vnode_{seq}$ *before/**after*** k
        else
          Insert j *into* $vnode_{seq}$ ***before**/after* k
        end if
      else if $(D_{col}(j) = 0)$ **then**
        if $(LZ_{col}(j) < LZ_{col}(k))$ **then**
          Insert j *into* $vnode_{seq}$ *before/after* k
        else if $(LZ_{col}(j) = LZ_{col}(k))$ **then**
          Insert j *into* $vnode_{seq}$ *befor/after* k
        else
          Insert j *into* $vnode_{seq}$ *before/after* k
        end if
      else
        if $(LZ_{col}(j) < LZ_{col}(k))$ **then**
          Insert j *into* $vnode_{seq}$ ***before**/after* k
        else if $(LZ_{col}(j) = LZ_{col}(k))$ **then**
          Insert j *into* $vnode_{seq}$ *before/after* k
        else
          Insert j *into* $vnode_{seq}$ *before/**after*** k
        end if
      end if
    else
      Insert j *into* $vnode_{seq}$ *after* k
      go to step 4
    end if
  end for
end for
CPM = permute(matrix(r,v), $vnode_{seq}$*)* // permute the matrix(r,v) rows according to the $vnode_{seq}$

Appendix-B

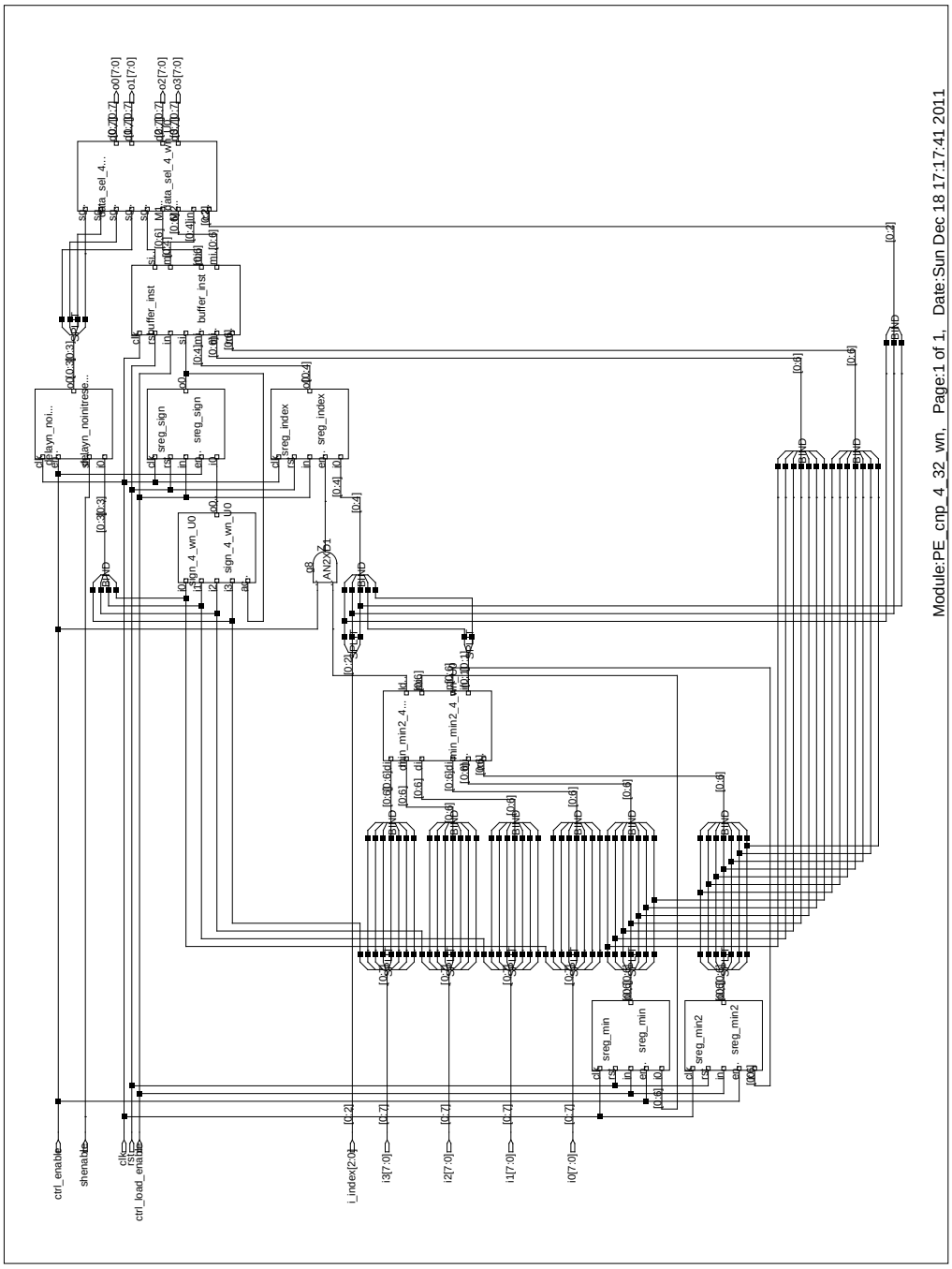

Figure 1: Check Node Processor Schematic after Synthesis in Cadence RC Compiler

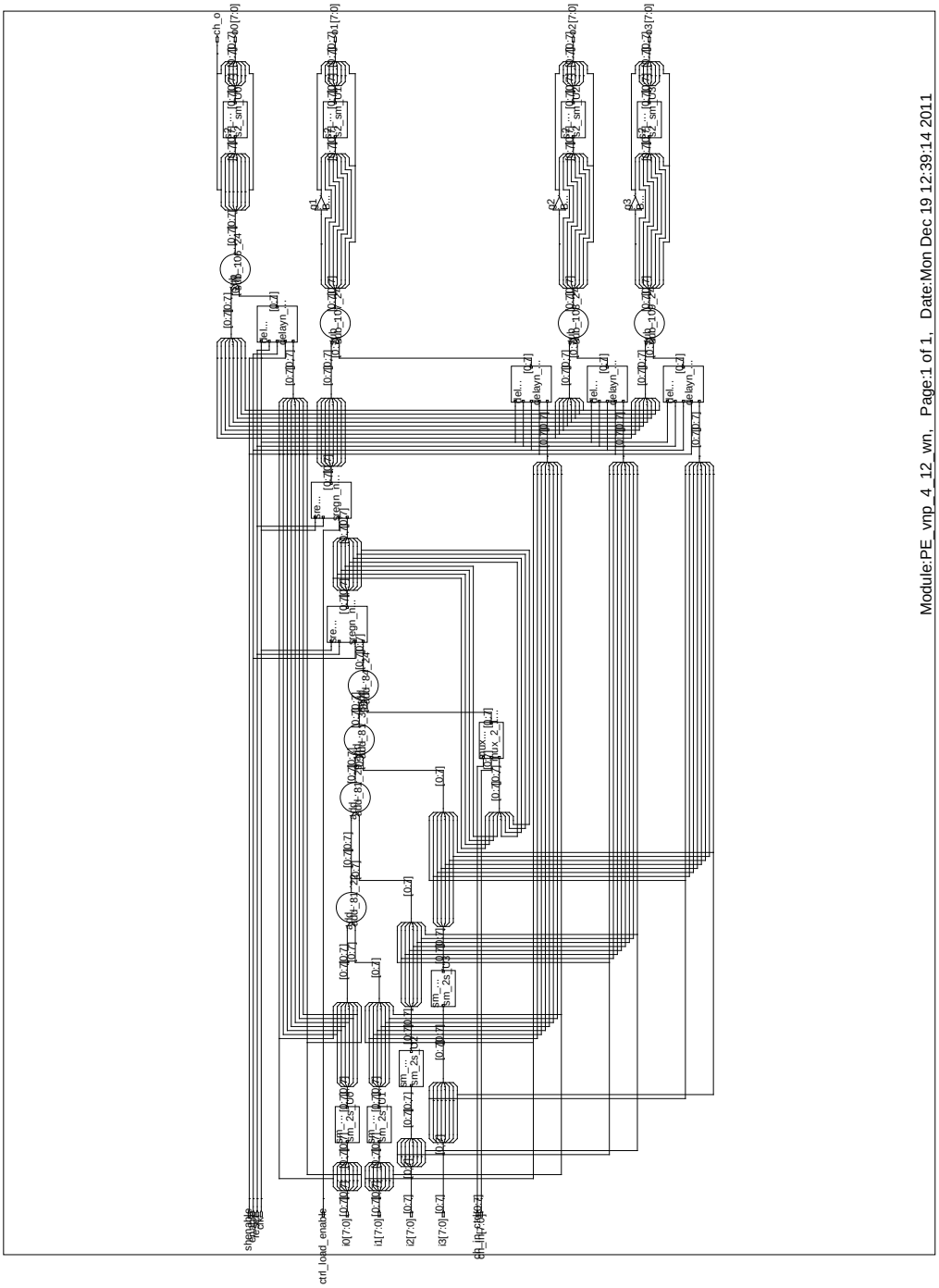

Figure 2: Variable Node Processor Schematic after Synthesis in Cadence RC Compiler

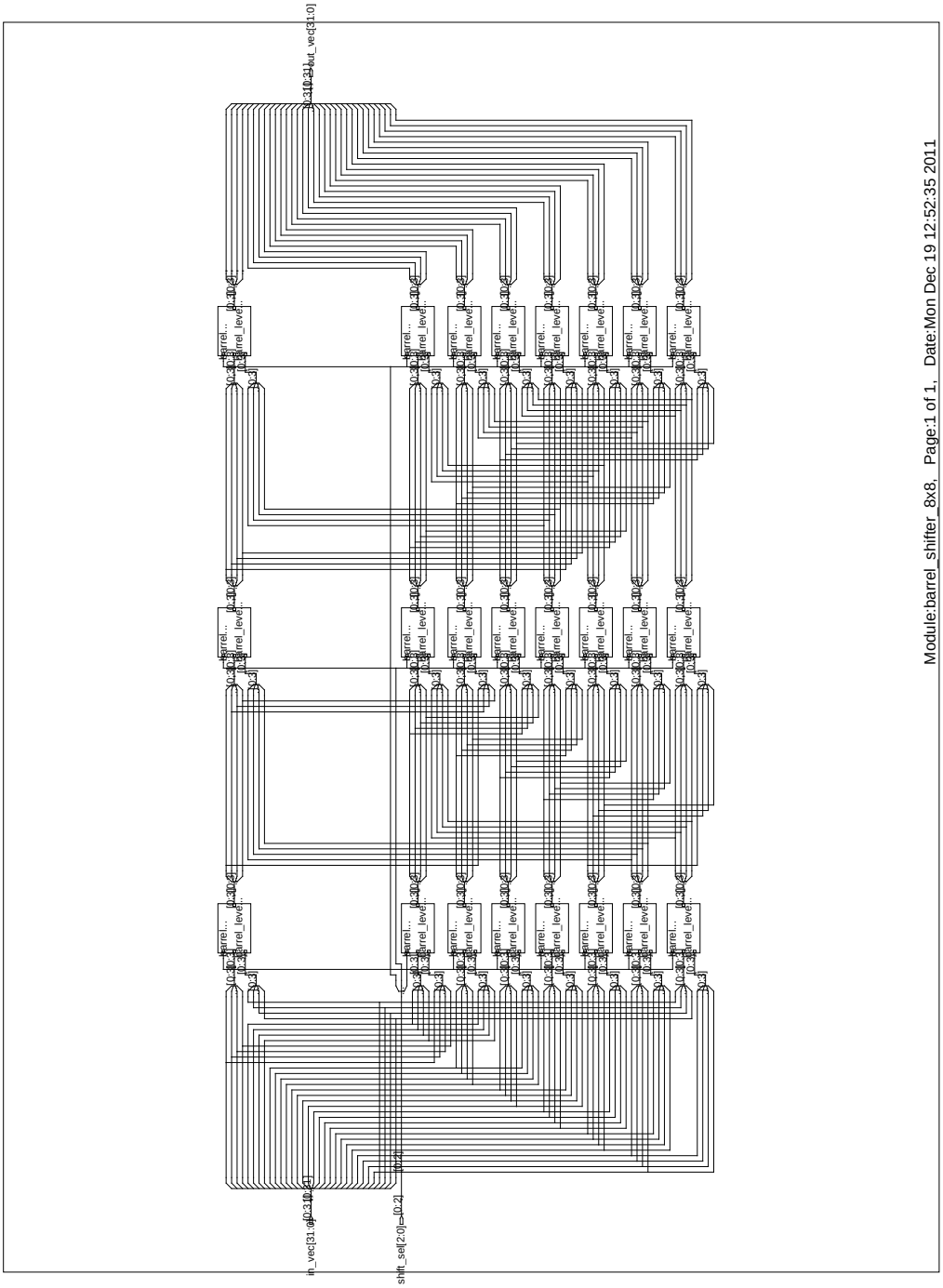

Figure 3: Multi-stage Cyclic Shifter Schematic after Synthesis in Cadence RC Compiler

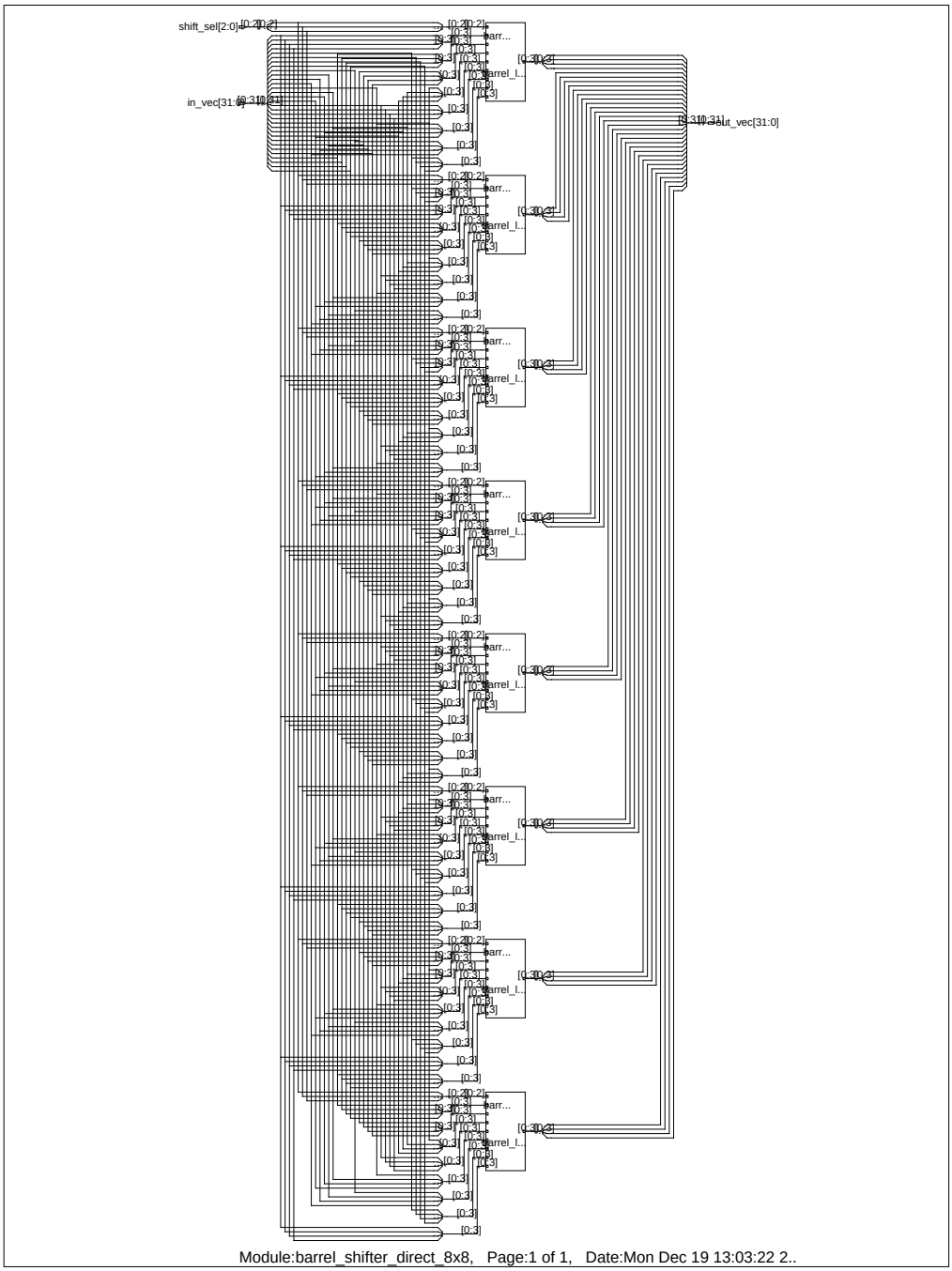

Figure 4: Single-stage Cyclic Shifter Schematic after Synthesis in Cadence RC Compiler

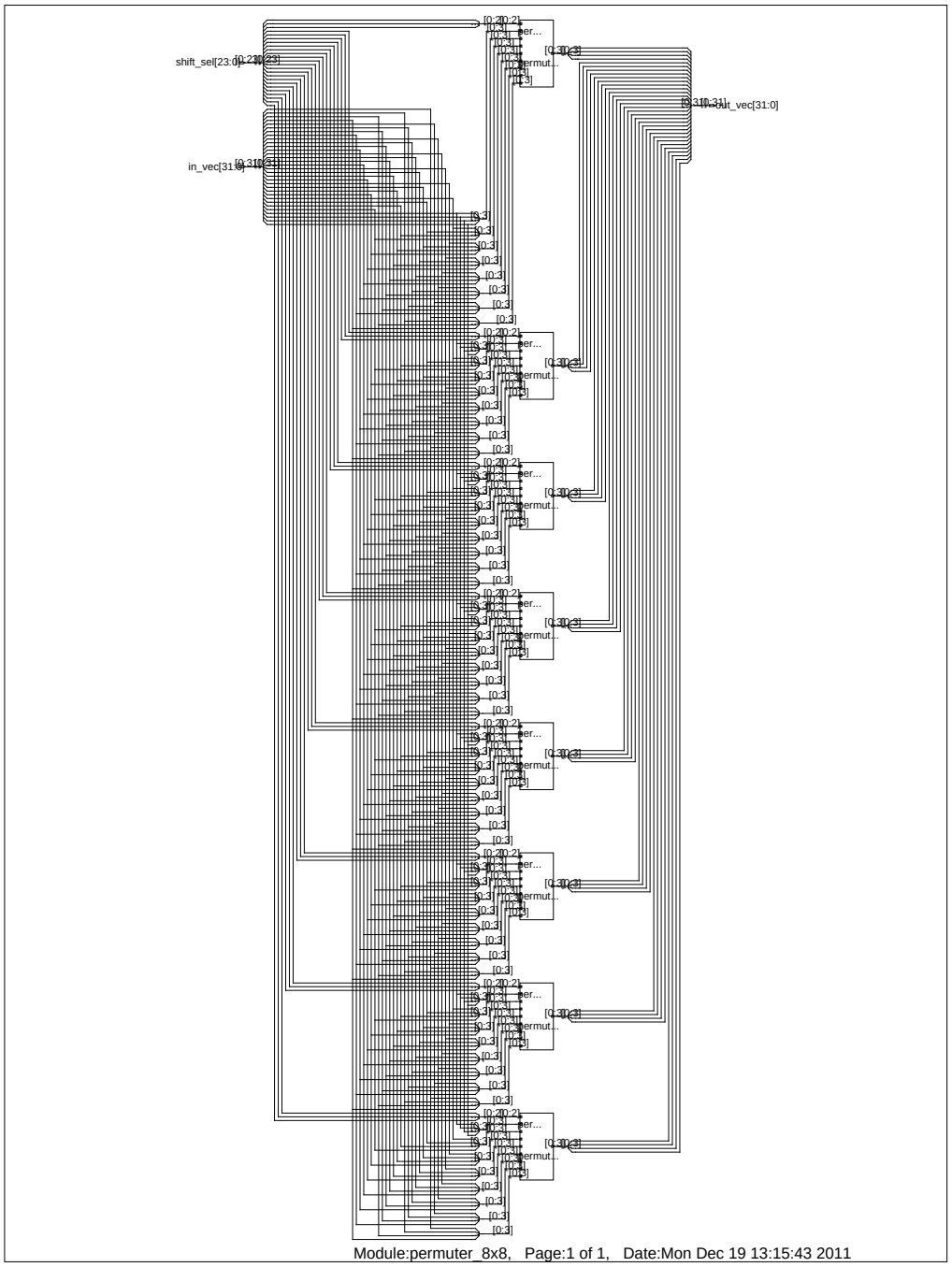

Figure 5: Single-stage Permuter Schematic after Synthesis in Cadence RC Compiler

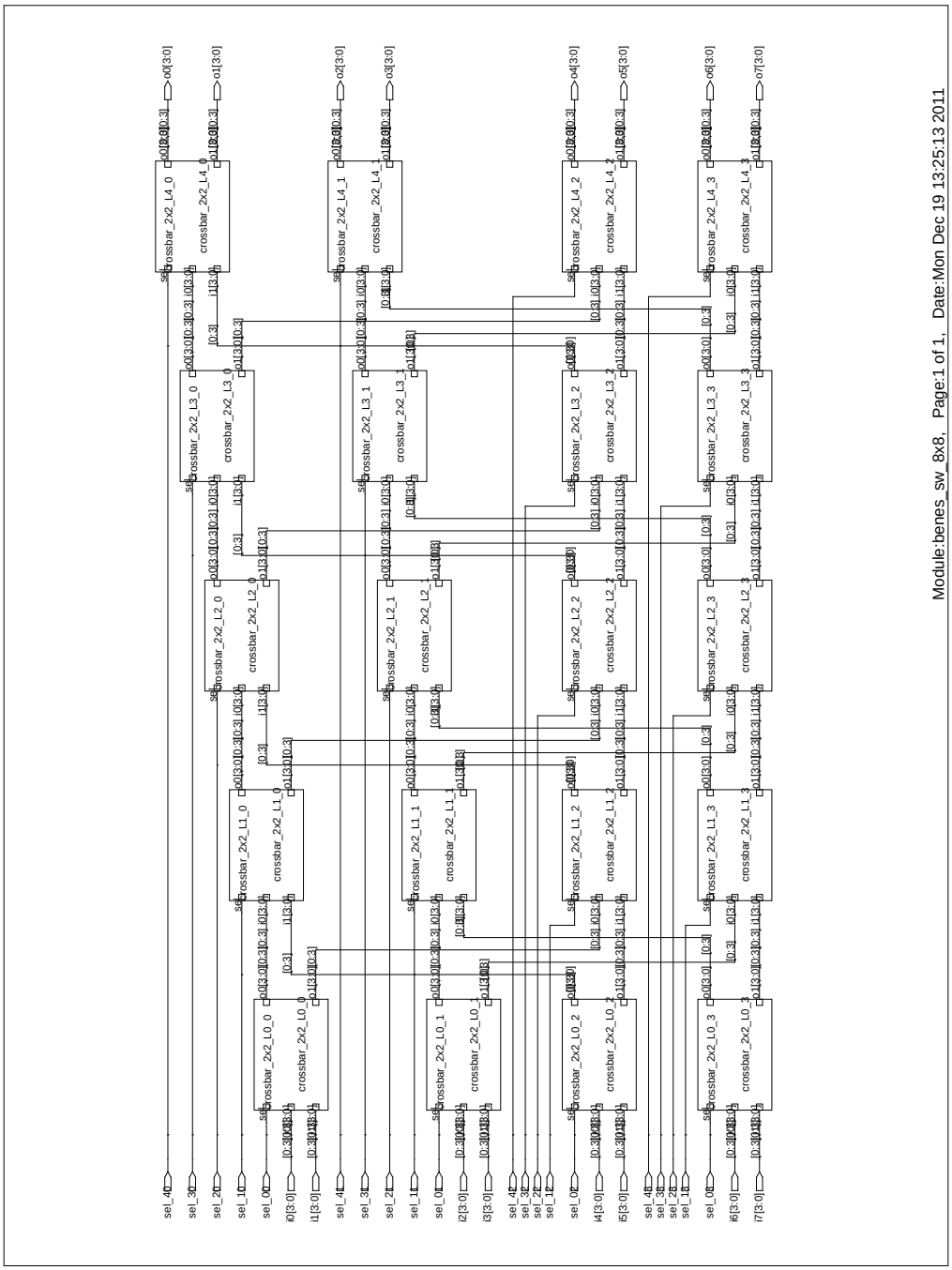

Figure 6: Multi-stage Permuter Schematic after Synthesis in Cadence RC Compiler

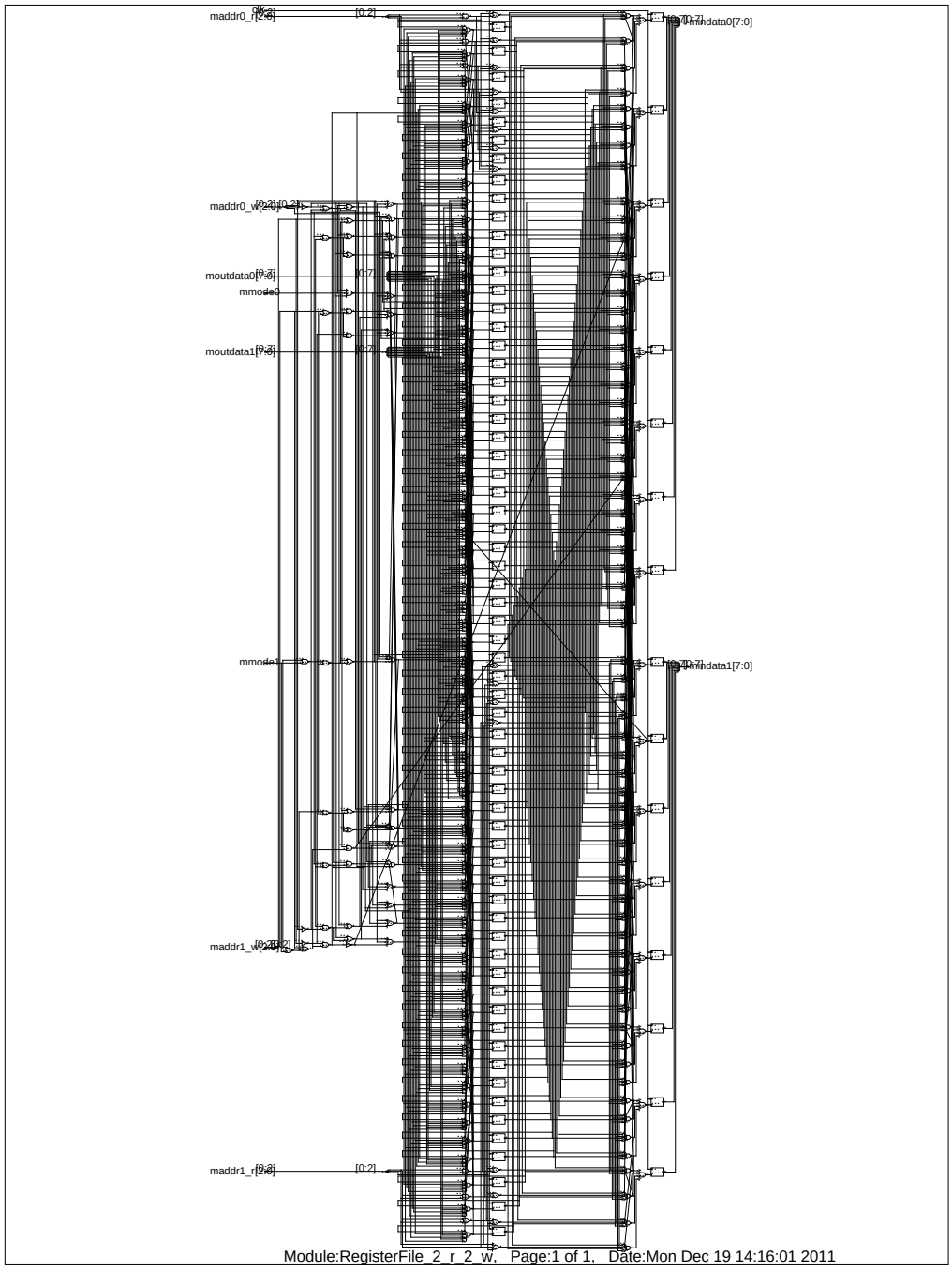

Figure 7: 2-Read and 2-Write Port Register File Schematic after Synthesis in Cadence RC Compiler

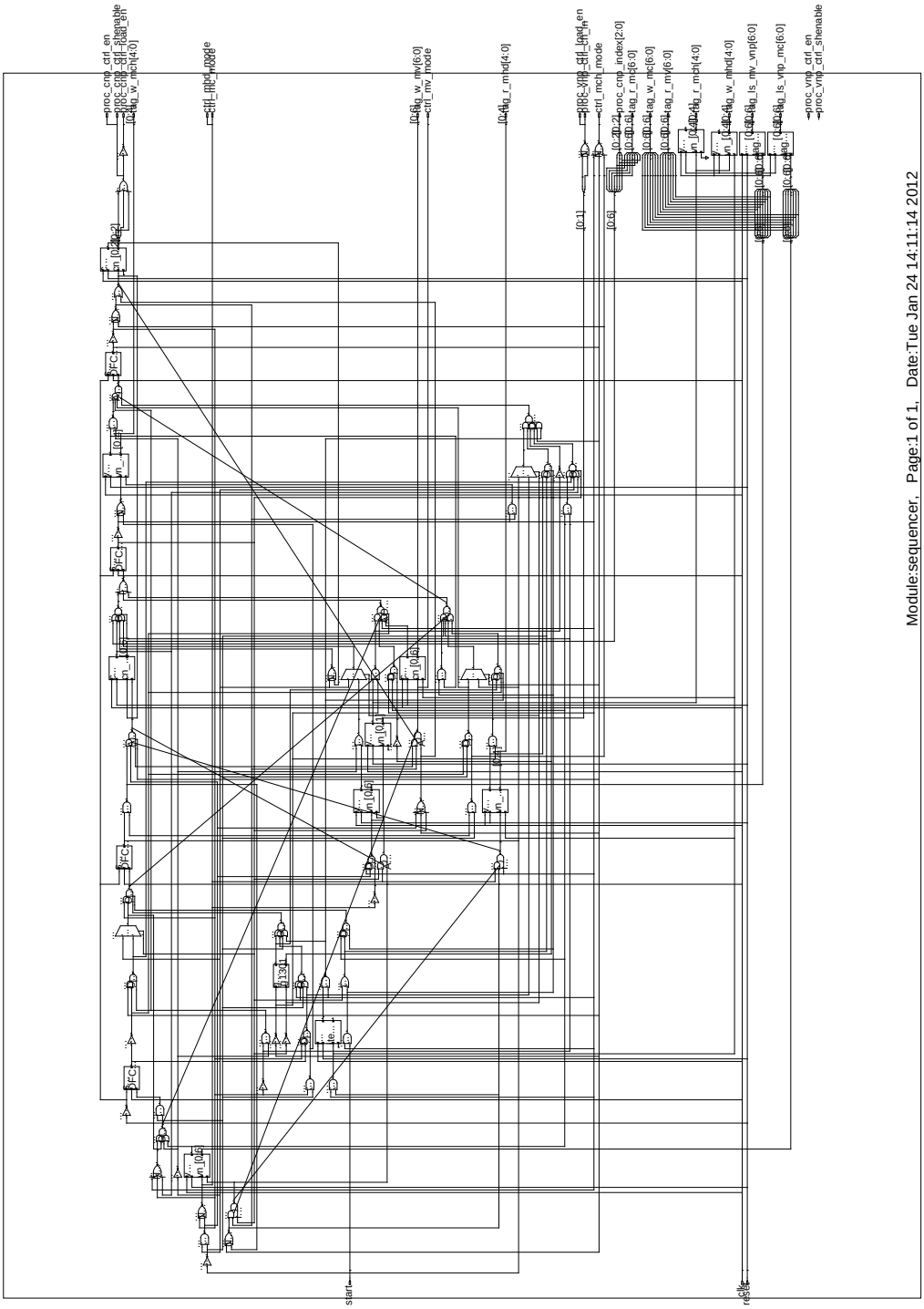

Figure 8: Sequencer/Controller Schematic after Synthesis in Cadence RC Compiler

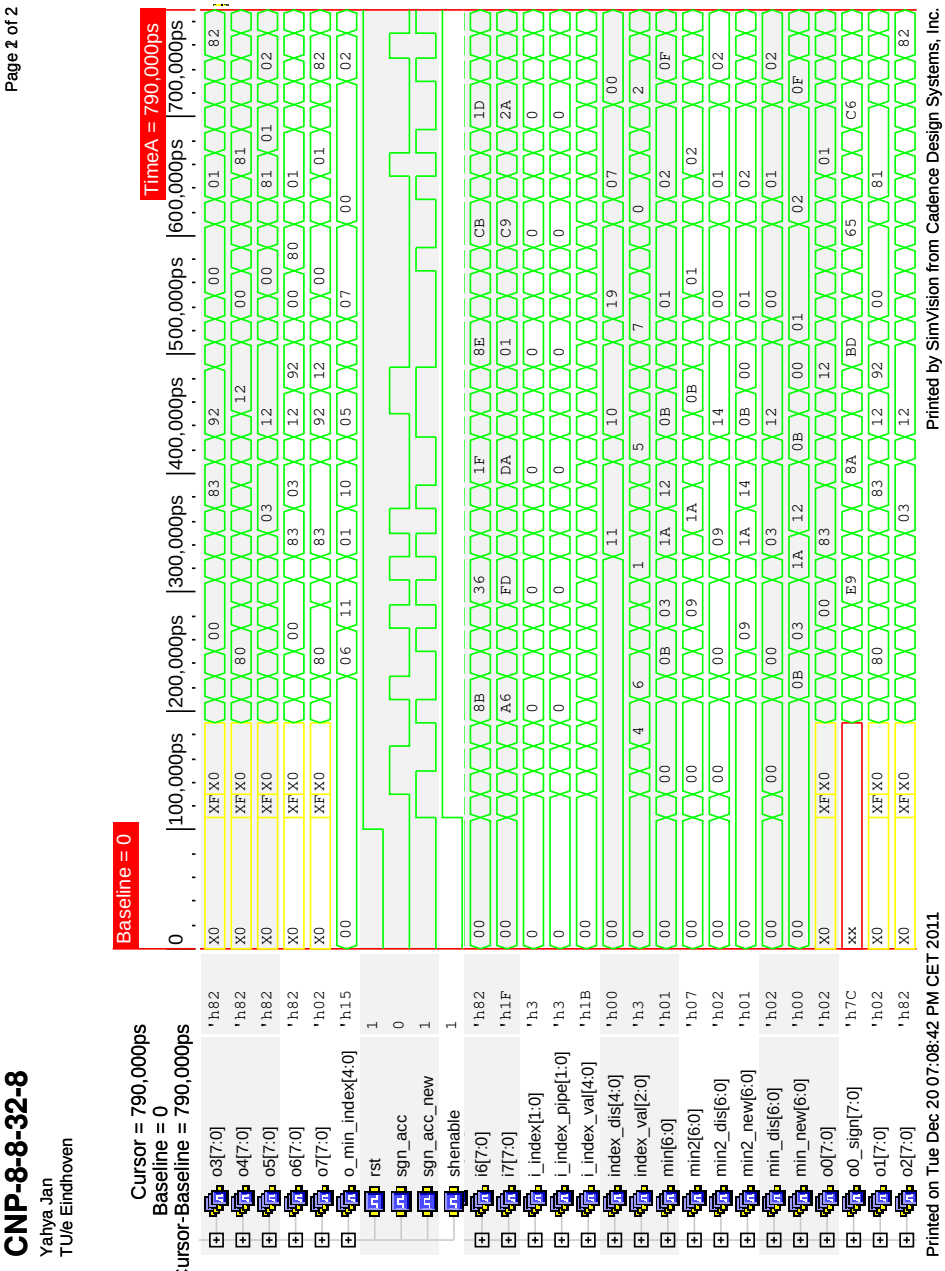

Figure 9: Functional Simulation of 8-input 8-output CNP in Cadence SimVision

Table 2: Signals and their description related to the simulation waveform of the CNP-8-8-32-8 block as given in Figure 9.

Signals	Description
clk	The clock signal controls the Check Node Processor (CNP) timing behavior
reset	The reset signal initializes the VNP
ctrl_load_enable	Controls the loading of processed data from the temporary registers to the output buffer
shenable	Controls the shifting of the input sign data in the FIFO inside the CNP
enable	CNP processor global control
o_min_index[4:0]	Index of the minimum value among all the inputs
sgn_acc	Signal represents the xor of all the signs of inputs
i0[7:0], i1[7:0], ..., i7[7:0]	CNP processor data input ports each of 8-bit data width
o0[7:0], o1[7:0], ..., o7[7:0]	CNP processor data output ports each of 8-bit data width

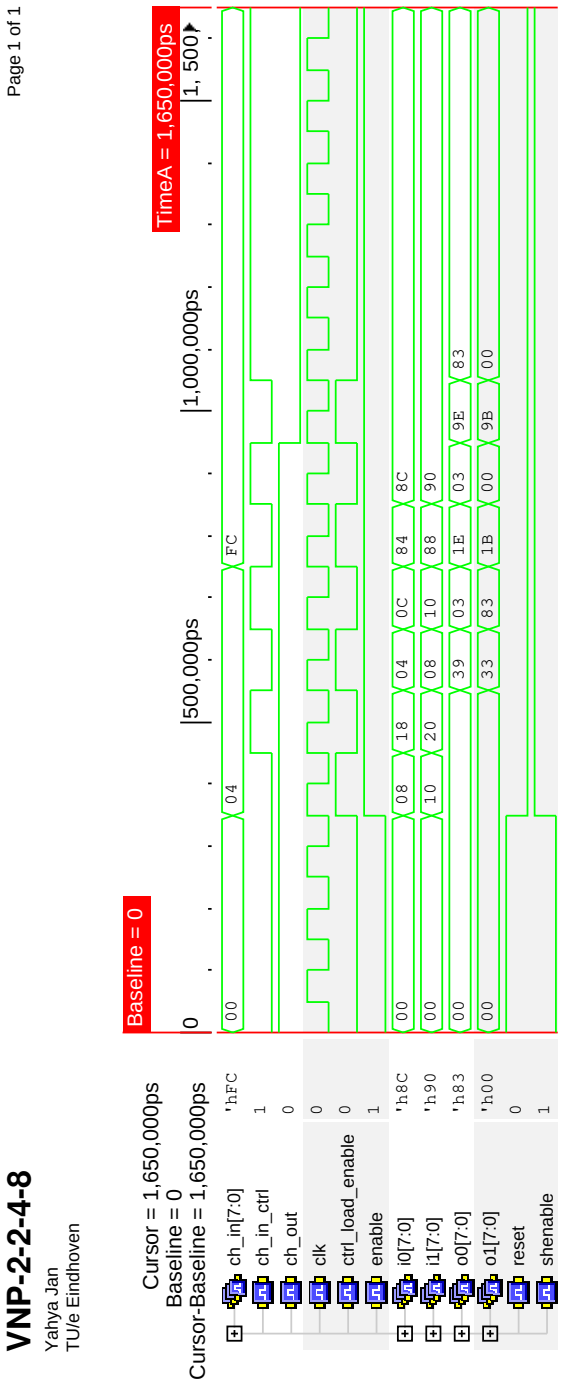

Figure 10: Functional Simulation of 2-input 2-output VNP in Cadence SimVision

Table 3: Signals and their description related to the simulation waveform of the VNP-2-2-4-8 block as given in Figure 10.

Signals	Description
clk	The clock signal controls the Variable Node Processor (VNP) timing behavior
reset	The reset signal initializes the VNP
ch_in_ctrl	Controls the channel input data to be processed by the VNP
ctrl_load_enable	Controls the loading of processed data from the accumulator to the output buffer inside the VNP
shenable	Controls the shifting of the input data in the FIFO inside the VNP
enable	VNP processor global control
i0[7:0], i1[7:0]	VNP processor data input ports each of 8-bit data width
ch_in[7:0]	VNP processor channel data input port of 8-bit data width
o0[7:0], o1[7:0]	VNP processor data output ports each of 8-bit data width
ch_out	VNP processor output port that represents the value of the decoded bit

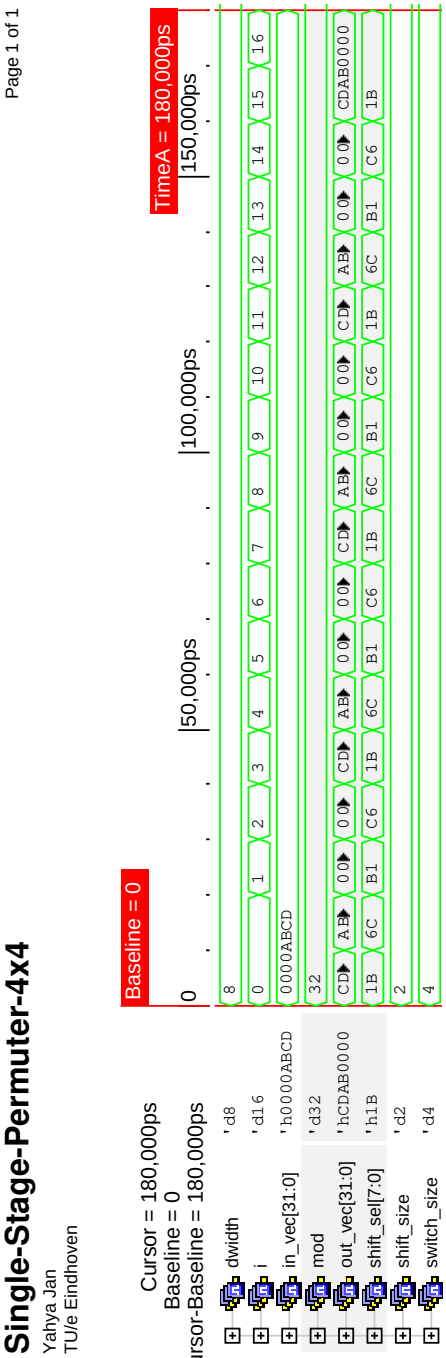

Figure 11: Functional Simulation of 4x4 Multi-Stage Permuter in Cadence SimVision

Table 4: Signals and their description related to the simulation waveform of the Single-Stage-Permuter-4x4 block as given in Figure 11.

Signals	Description
dwidth	The parameter represents the size of the data
shift_sel[7:0]	Switch control bus to control the multiplexers with 2-bits for each one of the 4-input multiplexer
in_vec[31:0]	Input Data bus comprised of 4 individual inputs each of 8-bit data width to form a 32-bit data bus
out_vec[31:0]	Output Data bus comprised of 4 individual outputs each of 8-bit data width to form a 32-bit data bus
switch_size	The parameter represents the size of the switch

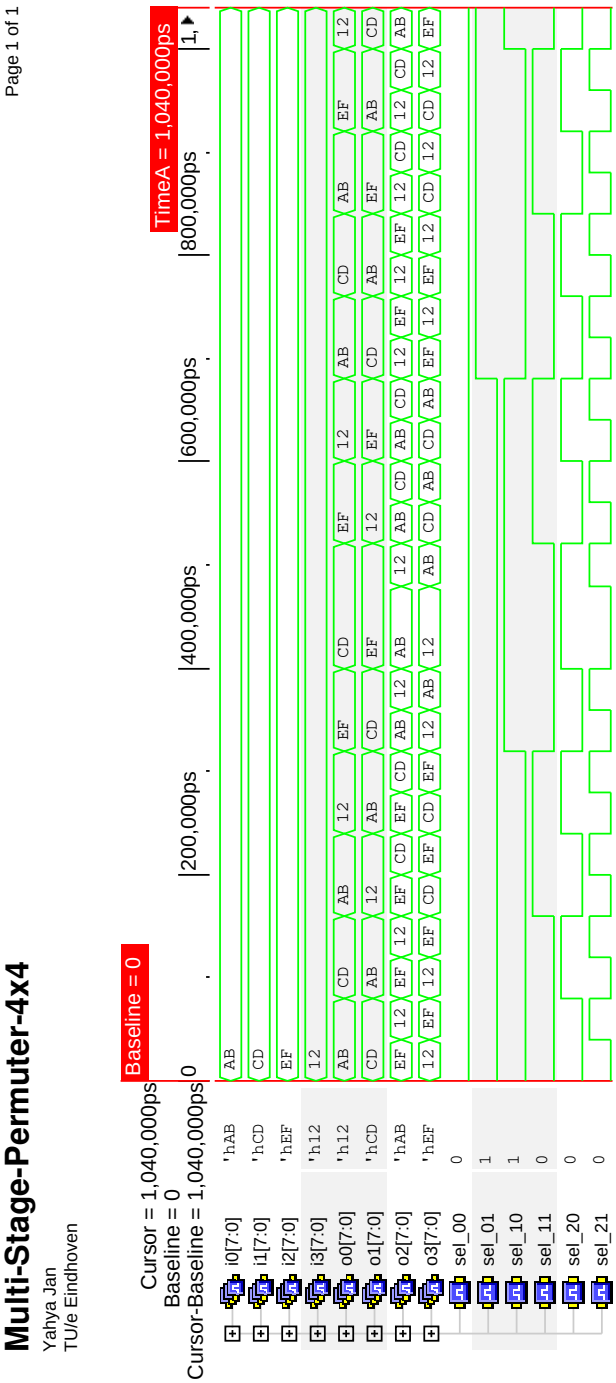

Figure 12: Functional Simulation of 4x4 Multi-Stage Permuter in Cadence SimVision

Table 5: Signals and their description related to the simulation waveform of Multi-Stage-Permuter block as given in Figure 12.

Signals	Description
shift_sel[7:0]	Switch control bus to control the multiplexers with 2-bits for each one of the 4-input multiplexer
i0[7:0],...,i3[7:0]	Individual data input ports of multi-stage switch each of 8-bit data width
o0[7:0],...,o3[7:0]	Individual data output ports of multi-stage switch each of 8-bit data width
sel_00	Controls the first crossbar switch of the first level switches of the multi-stage switch
sel_01	Controls the second crossbar switch of the first level switches of the multi-stage switch
sel_10	Controls the first crossbar switch of the middle level switches of the multi-stage switch
sel_11	Controls the second crossbar switch of the middle level switches of the multi-stage switch
sel_20	Controls the first crossbar switch of the last level switches of the multi-stage switch
sel_21	Controls the second crossbar switch of the last level switches of the multi-stage switch

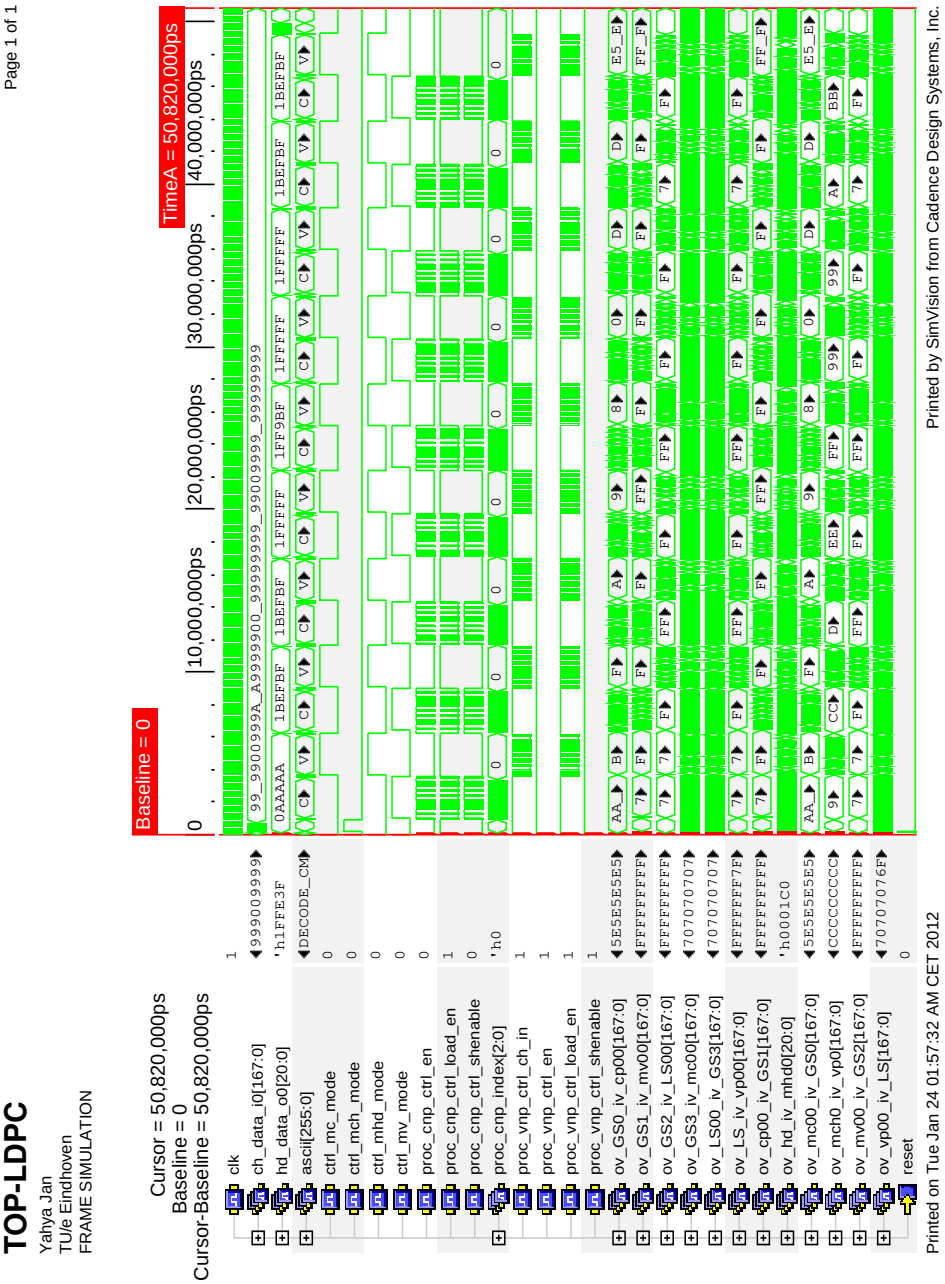

Figure 13: Functional Simulation of Top-level LDPC Decoder of a single frame in Cadence SimVision

Table 6: Signals and their description related to the simulation waveform of TOP_LDPC_CMPT block as given in Figure 13.

Signals	Description
clk	The clock signal controls the decoder timing behavior
reset	The reset signal initializes the decoder
ch_data_i0[167:0]	Decoder input data bus that represents the received channel data (code word)
ch_data_o0[20:0]	Decoder output data bus that represents the decoded code word
ascii[255:0]	The parameter is the ASCII representation of the state register of the FSM of the decoder (4-bit state register)
ctrl_xxx_mode	The memory mode (read/write) control signals for different memories
proc_cnp_xxx_xxx	All the signals represent the CNP processor control signals as described for CNP waveform
proc_vnp_xxx_xxx	All the signals represent the VNP processor control signals as described for VNP waveform
ov_mc00_iv_GS0[167:0]	The data bus connects the outputs of the M_{cv} memories to the inputs of global switch 0
ov_GS0_iv_cp00[167:0]	The data bus connects the outputs of the global switch 0 to the inputs of CNP processors cluster
ov_cp00_iv_GS1[167:0]	The data bus connects the outputs of the CNP cluster to the inputs of global switch 1
ov_GS1_iv_mv00[167:0]	The data bus connects the outputs of the global switch 1 to the inputs of M_{vc} memories
ov_mv00_iv_GS2[167:0]	The data bus connects the outputs of the M_{cv} memories to the inputs of global switch 2
ov_GS3_iv_mc00[167:0]	The data bus connects the outputs of the global switch 3 to the inputs of M_{cv} memories
ov_LS_iv_xxx[167:0]	The data bus connects the outputs of local switches to the inputs of a processor cluster or global switch
ov_xxx_iv_LS[167:0]	The data bus connects the outputs of a processor cluster or global switch to the inputs of local switches
ov_mch0_iv_vp0[167:0]	The data bus connects the outputs of M_{ch} memories to the inputs VNP processor clusters
ov_hd_iv_mhd0[20:0]	The data bus connects the decoded outputs of VNP processor clusters to the inputs of M_{HD} memories

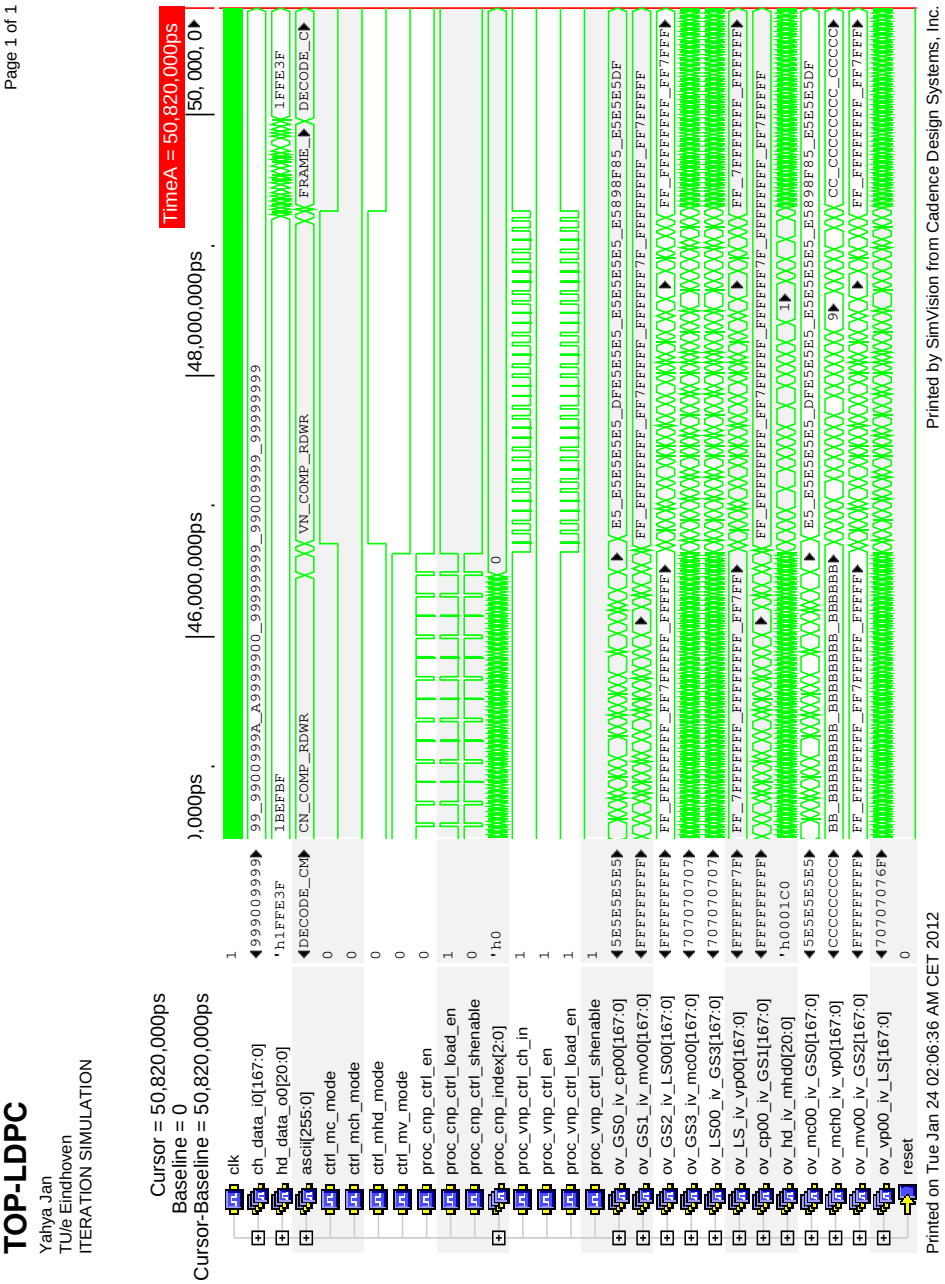

Figure 14: Functional Simulation of Top-level LDPC Decoder of a single iteration of a single frame in Cadence SimVision

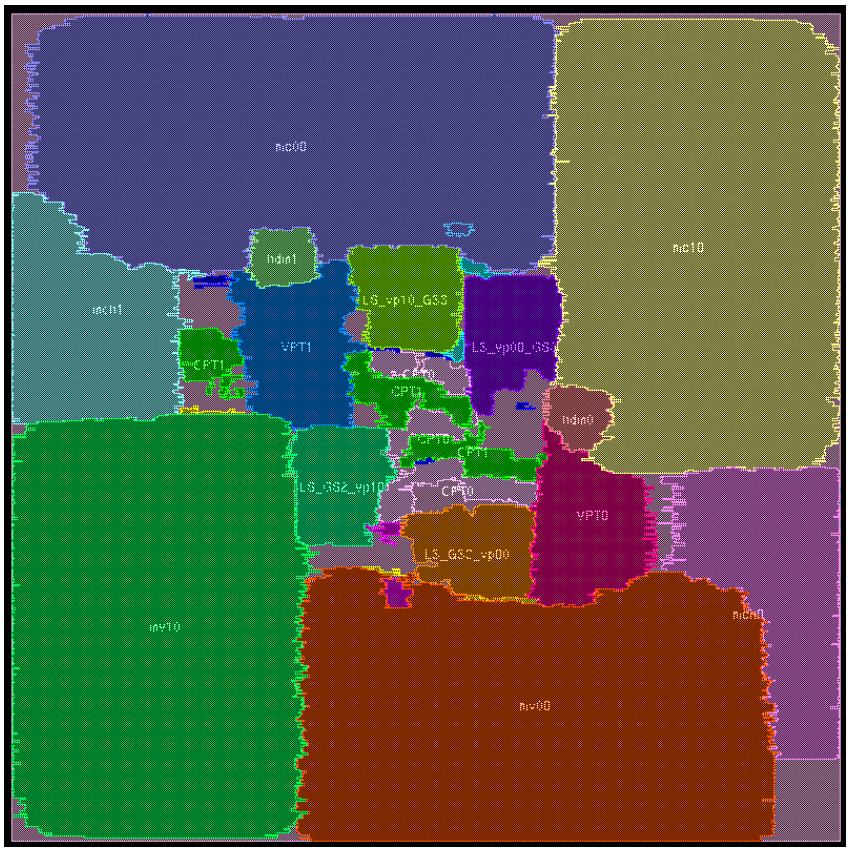

Figure 15: Snapshots of floorplan of an instance of LDPC decoder taken from Cadence Encounter(R) RTL-to-GDSII System 9.12

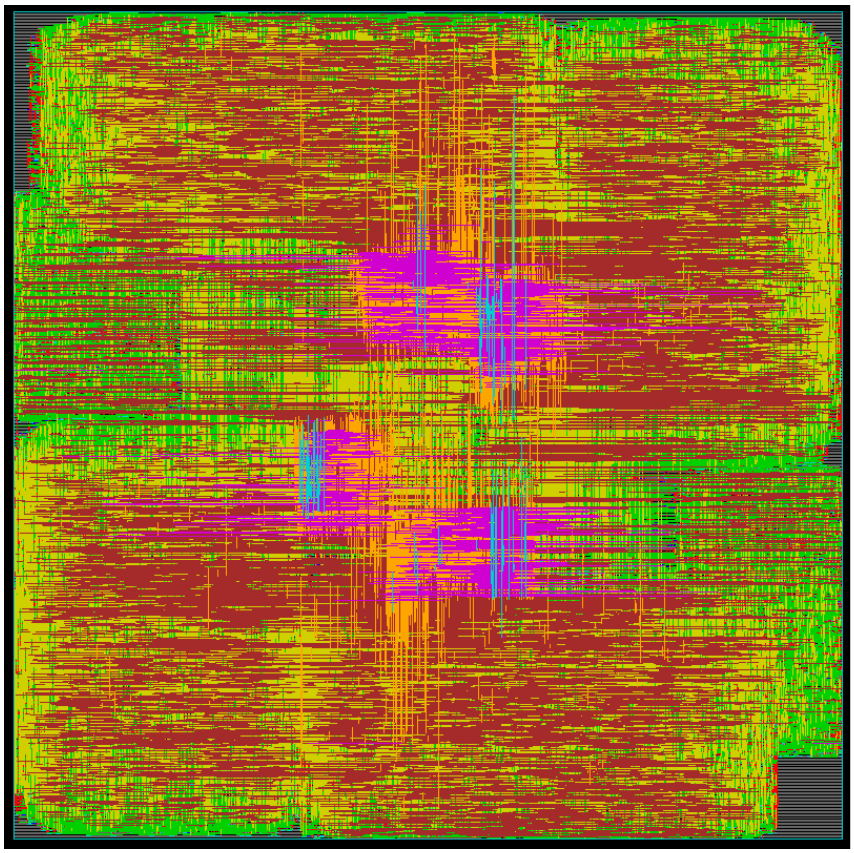

Figure 16: Snapshots of placed and routed instance of an LDPC decoder taken from Cadence Encounter(R) RTL-to-GDSII System 9.12

Bibliography

[1] Lech Józwiak, Nadia Nedjah, and Miguel Figueroa. Modern development methods and tools for embedded reconfigurable systems: A survey. *Integr. VLSI J.*, 43:1–33, January 2010. ISSN 0167-9260. doi: 10.1016/j.vlsi.2009.06.002.

[2] R. Gallager. Low-density parity-check codes. *Information Theory, IRE Transactions on*, 8(1):21–28, January 1962. ISSN 0096-1000. doi: 10.1109/TIT.1962.1057683.

[3] Ieee draft: Local and metropolitan area networks–specific requirements–part 15.3: Wireless medium access control (mac) and physical layer (phy) specifications for high rate wireless personal area networks (wpans): Amendment 2: Millimeter-wave based alternative physical layer extension. *IEEE Unapproved Draft Std P802.15.3c/D08, Mar 2009*.

[4] G. Lechner, J. Sayir, and M. Rupp. Efficient dsp implementation of an ldpc decoder. In *Proceedings of IEEE International Conference on Acoustics, Speech, and Signal Processing, 2004 (ICASSP '04).*, volume 4, pages 665–668, may 2004. doi: 10.1109/ICASSP.2004.1326914.

[5] G. Falco Fernandes, L. Sousa, and V. Silva. Massively ldpc decoding on multicore architectures. *IEEE Trans. on Parallel and Distributed Systems*, 22(2):309– 322, February 2011. ISSN 1045-9219.

[6] Yahya Jan and Lech Jozwiak. Cabac accelerator architectures for video compression in future multimedia: A survey. In *Proceedings of the 9th International Workshop on Embedded Computer Systems: Architectures, Modeling, and Simulation*, SAMOS '09, pages

24–35, Berlin, Heidelberg, 2009. Springer-Verlag. ISBN 978-3-642-03137-3. doi: http://dx.doi.org/10.1007/978-3-642-03138-0_4. URL http://dx.doi.org/10.1007/978-3-642-03138-0_4.

[7] Wei Yu and *et al.* A high performance cabac decoding architecture. *IEEE Transactions on Consumer Electronics*, pages 1352–1359, Nov. 2005. ISSN 0098-3063. doi: 10.1109/TCE.2005.1561867.

[8] Vivienne Sze and *et al.* Parallel cabac for low power video coding. *In proceedings of 15th IEEE International Conference on Image Processing, 2008. ICIP 2008.*, pages 2096–2099, Oct. 2008. ISSN 1522-4880. doi: 10.1109/ICIP.2008.4712200.

[9] H. Shojania and *et al.* A high performance cabac encoder. *In proceedings of the 3rd International IEEE-NEWCAS Conference, 2005.*, pages 315–318, June 2005. doi: 10.1109/NEWCAS.2005.1496683.

[10] Erik Franken and Remco Duits. Crossing-preserving coherence-enhancing diffusion on invertible orientation scores. *Int. J. Comput. Vision*, 85:253–278, December 2009. ISSN 0920-5691. doi: 10.1007/s11263-009-0213-5. URL http://dl.acm.org/citation.cfm?id=1644192.1644214.

[11] R. Schreiber, S. Aditya, B. Ramakrishna Rau, V. Kathail, S. Mahlke, S. Abraham, and G. Snider. High-level synthesis of nonprogrammable hardware accelerators. In *Proceedings of IEEE International Conference on Application-Specific Systems, Architectures, and Processors, 2000.*, pages 113 –124, 2000. doi: 10.1109/ASAP.2000.862383.

[12] Krzysztof Kuchcinski and Christophe Wolinski. Global approach to assignment and scheduling of complex behaviors based on hcdg and constraint programming. *J. Syst. Archit.*, 49:489–503, December 2003. ISSN 1383-7621. doi: 10.1016/S1383-7621(03)00075-4. URL http://dl.acm.org/citation.cfm?id=967622.967624.

[13] S. Gupta, N. Dutt, R. Gupta, and A. Nicolau. Spark: a high-level synthesis framework for applying parallelizing compiler transformations. In *Proceedings of 16th International Conference on VLSI Design, 2003.*, pages 461 – 466, jan. 2003. doi: 10.1109/ICVD.2003.1183177.

[14] Z. Guo, B. Buyukkurt, W. Najjar, and K. Vissers. Optimized generation of data-path from c codes for fpgas. In *Proceedings of Design, Automation and Test in Europe, 2005.*, pages 112 – 117 Vol. 1, march 2005. doi: 10.1109/DATE.2005.234.

[15] M. Puschel, J.M.F. Moura, J.R. Johnson, D. Padua, M.M. Veloso, B.W. Singer, J. Xiong, F. Franchetti, A. Gacic, Y. Voronenko, K. Chen, R.W.

Johnson, and N. Rizzolo. Spiral: Code generation for dsp transforms. *Proceedings of the IEEE*, 93(2):232 –275, feb. 2005. ISSN 0018-9219. doi: 10.1109/JPROC.2004.840306.

[16] W. Sun, M. J. Wirthlin, and S. Neuendorffer. Fpga pipeline synthesis design exploration using module selection and resource sharing. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 26(2):254 – 265, feb. 2007. ISSN 0278-0070. doi: 10.1109/TCAD.2006.887923.

[17] Saraju P. Mohanty, Nagarajan Ranganathan, Elias Kougianos, and Priyardarsan Patra. *Low-Power High-Level Synthesis for Nanoscale CMOS Circuits*. Springer Publishing Company, Incorporated, 1 edition, 2008. ISBN 0387764739, 9780387764733.

[18] J. Cong, Bin Liu, S. Neuendorffer, J. Noguera, K. Vissers, and Zhiru Zhang. High-level synthesis for fpgas: From prototyping to deployment. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 30(4):473 –491, april 2011. ISSN 0278-0070. doi: 10.1109/TCAD.2011.2110592.

[19] Andrew Canis, Jongsok Choi, Mark Aldham, Victor Zhang, Ahmed Kammoona, Jason H. Anderson, Stephen Brown, and Tomasz Czajkowski. Legup: high-level synthesis for fpga-based processor/accelerator systems. In *Proceedings of the 19th ACM/SIGDA international symposium on Field programmable gate arrays*, FPGA '11, pages 33–36, New York, NY, USA, 2011. ACM. ISBN 978-1-4503-0554-9. doi: http://doi.acm.org/10.1145/1950413.1950423. URL `http://doi.acm.org/10.1145/1950413.1950423`.

[20] CatapultC synthesis. Mentor's Graphics Corporation. [Online] Available at: `http://www.mentor.com/`.

[21] Cynthesizer. Cynthesizer Closes the ESL-to-Silicon Gap. [Online] Available at: `http://www.forteds.com/products/cynthesizer.asp/`.

[22] Impulse CoDeveloper. Impulse Accelerated Technologies. [Online] Available at: `http://www.impulsec.com/products.htm/`.

[23] CyberWorkBench. CyberWorkBench. [Online] Available at: `http://www.nec.com/global/prod/cwb/index.html/`.

[24] Cadence. C-to-Silicon Compiler. [Online] Available at: `http://www.cadence.com/`.

[25] Xilinx. AutoESL High-Level Synthesis Tool. [Online] Available at: `http://www.xilinx.com/tools/autoesl.htm/`.

[26] Synphony. High-Level Synthesis with Synphony C Compiler. [Online] Available at: `http://www.synopsys.com/`.

[27] M. Tudruj and L. Masko. Communication on the fly for hierarchical systems of chip multi-processors. In *Proceedings of 6th International Symposium on Parallel Computing in Electrical Engineering (PARELEC), 2011*, pages 19 –24, april 2011. doi: 10.1109/PARELEC.2011.32.

[28] Ling Wang, Chunda Ding, Shenghai Zhong, and Jianwen Zhang. Gnls: a hybrid on chip communication architecture for soc designs. *Int. J. High Perform. Syst. Archit.*, 3:157–166, May 2011. ISSN 1751-6528. doi: http://dx.doi.org/10.1504/IJHPSA.2011.040468. URL `http://dx.doi.org/10.1504/IJHPSA.2011.040468`.

[29] Matteo Monchiero, Gianluca Palermo, Cristina Silvano, and Oreste Villa. Exploration of distributed shared memory architectures for noc-based multiprocessors. In *Proceedings of International Conference on Embedded Computer Systems: Architectures, Modeling and Simulation, 2006. IC-SAMOS 2006.*, pages 144 –151, july 2006. doi: 10.1109/ICSAMOS.2006.300821.

[30] M. Kistler, M. Perrone, and F. Petrini. Cell multiprocessor communication network: Built for speed. *Micro, IEEE*, 26(3):10 –23, may-june 2006. ISSN 0272-1732. doi: 10.1109/MM.2006.49.

[31] Avinash Karanth Kodi and Ahmed Louri. A scalable architecture for distributed shared memory multiprocessors using optical interconnects. In *Proceedings of 18th International Parallel and Distributed Processing Symposium, 2004.*, page 11, april 2004. doi: 10.1109/IPDPS.2004.1302914.

[32] Hyung Gyu Lee, Naehyuck Chang, Umit Y. Ogras, and Radu Marculescu. On-chip communication architecture exploration: A quantitative evaluation of point-to-point, bus, and network-on-chip approaches. *ACM Trans. Des. Autom. Electron. Syst.*, 12:23:1–23:20, May 2008. ISSN 1084-4309. doi: http://doi.acm.org/10.1145/1255456.1255460. URL `http://doi.acm.org/10.1145/1255456.1255460`.

[33] Sergio Saponara, Luca Fanucci, and Esa Petri. A multi-processor noc-based architecture for real-time image/video enhancement. *Journal of Real-Time Image Processing*, pages 1–15. ISSN 1861-8200. URL `http://dx.doi.org/10.1007/s11554-011-0215-8`. 10.1007/s11554-011-0215-8.

[34] Lech Jóźwiak. Quality-driven design in the system-on-a-chip era: Why and how? *Journal of Systems Architecture*, 47(3-4):201 – 224, 2001. ISSN 1383-7621. doi: DOI:10.1016/S1383-7621(00)00046-1.

[35] L. Jozwiak and S.A. Ong. Quality-driven decision making methodology for system-level design. In *Proceedings of the 22nd EUROMICRO Conference EUROMICRO 96. 'Beyond 2000: Hardware and Software Design Strategies'.*, pages 8 –18, sep 1996. doi: 10.1109/EURMIC.1996.546360.

[36] L. Jozwiak and S.A. Ong. Quality-driven template-based architecture synthesis for real-time embedded socs. In *9th EUROMICRO Conference on Digital System Design: Architectures, Methods and Tools, 2006. DSD 2006*, pages 397 –406, 0-0 2006. doi: 10.1109/DSD.2006.79.

[37] Lech Józwiak. Modern concepts of quality and their relationship to design reuse and model libraries. *In Current Issues in Electronic Modeling, Chapter 8, Issue 5, 1995, Kluwer Academic Publishers, Dordrecht*, 1995.

[38] L. Jozwiak and A. Douglas. Hardware synthesis for reconfigurable heterogeneous pipelined accelerators. In *Proceedings of Fifth International Conference on Information Technology: New Generations, 2008. ITNG 2008.*, pages 1123 –1130, april 2008. doi: 10.1109/ITNG.2008.65.

[39] D.J.C. MacKay. Good error-correcting codes based on very sparse matrices. *IEEE Transactions on Information Theory*, 45(2):399 –431, mar 1999. ISSN 0018-9448. doi: 10.1109/18.748992.

[40] M.P.C. Fossorier, M. Mihaljevic, and H. Imai. Reduced complexity iterative decoding of low-density parity check codes based on belief propagation. *IEEE Transactions on Communications*, 47(5):673–680, May 1999. ISSN 0090-6778. doi: 10.1109/26.768759.

[41] D.E. Hocevar. A reduced complexity decoder architecture via layered decoding of ldpc codes. In *IEEE Workshop on Signal Processing Systems, 2004. SIPS 2004.*, pages 107 – 112, oct. 2004. doi: 10.1109/SIPS.2004.1363033.

[42] Jun Heo and K.M. Chugg. Optimization of scaling soft information in iterative decoding via density evolution methods. *IEEE Transactions on Communications*, 53(6):957–961, June 2005. ISSN 0090-6778. doi: 10.1109/TCOMM.2005.849782.

[43] Borivoje Nikolic Engling Yeo, Payam Pakzad and Venkat Anantharam. Vlsi architectures for iterative decoders in magnetic recording channels. *IEEE Trans. Magnetics*, 37:748–755, 2001.

[44] A.J. Blanksby and C.J. Howland. A 690-mw 1-gb/s 1024-b, rate-1/2 low-density parity-check code decoder. *IEEE Journal of Solid-State Circuits*, 37 (3):404 –412, mar 2002. ISSN 0018-9200. doi: 10.1109/4.987093.

[45] C. Howland and A. Blanksby. Parallel decoding architectures for low density parity check codes. In *IEEE International Symposium on Circuits and Systems, 2001. ISCAS 2001.*, volume 4, pages 742–745, may 2001. doi: 10.1109/ISCAS.2001.922344.

[46] A. Darabiha, A.C. Carusone, and F.R. Kschischang. Multi-gbit/sec low density parity check decoders with reduced interconnect complexity. In *IEEE*

International Symposium on Circuits and Systems, 2005. ISCAS 2005., pages 5194–5197 Vol. 5, May 2005. doi: 10.1109/ISCAS.2005.1465805.

[47] Euncheol Kim and G.S. Choi. Ldpc code for reduced routing decoder. In *Proceedings of Asia-Pacific Conference on Communications, 2005*, pages 991 –994, oct. 2005. doi: 10.1109/APCC.2005.1554212.

[48] N. Onizawa, T. Hanyu, and V.C. Gaudet. Design of high-throughput fully parallel ldpc decoders based on wire partitioning. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 18(3):482 –489, march 2010. ISSN 1063-8210. doi: 10.1109/TVLSI.2008.2011360.

[49] Federico Quaglio, Fabrizio Vacca, Cristiano Castellano, Alberto Tarable, and Guido Masera. Interconnection framework for high-throughput, flexible ldpc decoders. In *Proceedings of the conference on Design, automation and test in Europe: Designers' forum*, DATE '06, pages 124–129, 3001 Leuven, Belgium, Belgium, 2006. European Design and Automation Association. ISBN 3-9810801-0-6. URL `http://dl.acm.org/citation.cfm?id=1131355.1131381`.

[50] Zhengya Zhang, V. Anantharam, M.J. Wainwright, and B. Nikolic. An efficient 10gbase-t ethernet ldpc decoder design with low error floors. *IEEE Journal of Solid-State Circuits*, 45(4):843 –855, april 2010. ISSN 0018-9200. doi: 10.1109/JSSC.2010.2042255.

[51] S. Sharifi Tehrani, S. Mannor, and W.J. Gross. Fully parallel stochastic ldpc decoders. *IEEE Transactions on Signal Processing*, 56(11):5692 –5703, nov. 2008. ISSN 1053-587X. doi: 10.1109/TSP.2008.929671.

[52] Massimo Rovini, Giuseppe Gentile, Francesco Rossi, and Luca Fanucci. A minimum-latency block-serial architecture of a decoder for ieee 802.11n ldpc codes. In *Proceedings of International Conference on Very Large Scale Integration, 2007. VLSI - SoC 2007. IFIP*, pages 236–241, Oct. 2007. doi: 10.1109/VLSISOC.2007.4402504.

[53] M. Rovini, N.E. L'Insalata, F. Rossi, and L. Fanucci. Vlsi design of a high-throughput multi-rate decoder for structured ldpc codes. In *Proceedings of 8th Euromicro Conference on Digital System Design, 2005.*, pages 202–209, Aug.-3 Sept. 2005. doi: 10.1109/DSD.2005.77.

[54] Zhiqiang Cui, Zhongfeng Wang, and Youjian Liu. High-throughput layered ldpc decoding architecture. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 17(4):582–587, April 2009. ISSN 1063-8210. doi: 10.1109/TVLSI.2008.2005308.

[55] T. Brack, M. Alles, F. Kienle, and N. Wehn. A synthesizable ip core for wimax 802.16e ldpc code decoding. In *IEEE 17th International Symposium*

on Personal, Indoor and Mobile Radio Communications, 2006, pages 1–5, Sept. 2006. doi: 10.1109/PIMRC.2006.254126.

[56] T. Bhatt, V. Sundaramurthy, V. Stolpman, and D. McCain. Pipelined block-serial decoder architecture for structured ldpc codes. In *Proceedings of IEEE International Conference on Acoustics, Speech and Signal Processing, 2006. ICASSP 2006*, volume 4, pages IV–IV, May 2006. doi: 10.1109/ICASSP.2006.1660946.

[57] K. Gunnam, Gwan Choi, Weihuang Wang, and M. Yeary. Multi-rate layered decoder architecture for block ldpc codes of the ieee 802.11n wireless standard. In *IEEE International Symposium on Circuits and Systems, 2007. ISCAS 2007.*, pages 1645–1648, May 2007. doi: 10.1109/ISCAS.2007.378835.

[58] P. Radosavljevic, A. de Baynast, M. Karkooti, and J.R. Cavallaro. Multi-rate high-throughput ldpc decoder: Tradeoff analysis between decoding throughput and area. In *IEEE 17th International Symposium on Personal, Indoor and Mobile Radio Communications, 2006*, pages 1–5, Sept. 2006. doi: 10.1109/PIMRC.2006.254392.

[59] Luoming Zhang, Lin Gui, Youyun Xu, and Wenjun Zhang. Configurable multi-rate decoder architecture for qc-ldpc codes based broadband broadcasting system. *IEEE Transactions on Broadcasting*, 54(2):226–235, June 2008. ISSN 0018-9316. doi: 10.1109/TBC.2007.913400.

[60] Marjan Karkooti, Predrag Radosavljevic, and Joseph R. Cavallaro. Configurable ldpc decoder architectures for regular and irregular codes. *J. Signal Process. Syst.*, 53(1-2):73–88, 2008. ISSN 1939-8018. doi: http://dx.doi.org/10.1007/s11265-008-0221-7.

[61] Sangmin Kim, G.E. Sobelman, and Hanho Lee. Flexible ldpc decoder architecture for high-throughput applications. In *IEEE Asia Pacific Conference on Circuits and Systems, 2008. APCCAS 2008.*, pages 45–48, 30 2008-Dec. 3 2008. doi: 10.1109/APCCAS.2008.4745956.

[62] P. Urard, E. Yeo, L. Paumier, P. Georgelin, T. Michel, V. Lebars, E. Lantreibecq, and B. Gupta. A 135mb/s dvb-s2 compliant codec based on 64800b ldpc and bch codes. In *Proceedings of IEEE International Solid-State Circuits Conference, ISSCC. 2005*, pages 446 –609 Vol. 1, feb. 2005. doi: 10.1109/ISSCC.2005.1494061.

[63] A. Nagashima, Y. Imai, N. Togawa, M. Yanagisawa, and T. Ohtsuki. Dynamically reconfigurable architecture for multi-rate compatible regular ldpc decoding. In *Proceedings of IEEE Asia Pacific Conference on Circuits and Systems, 2008. APCCAS 2008.*, pages 705–708, 30 2008-Dec. 3 2008. doi: 10.1109/APCCAS.2008.4746121.

[64] K.K. Gunnam, G.S. Choi, and M.B. Yeary. A parallel vlsi architecture for layered decoding for array ldpc codes. In *Proceedings of 20th International Conference on VLSI Design, 2007.*, pages 738 –743, jan. 2007. doi: 10.1109/VLSID.2007.19.

[65] Takashi Yokokawa, Osamu Shinya, Yuji Shinohara, and Toshiyuki Miyauchi. A high performance and programmable decoder vlsi for structured ldpc codes. In *Proceedings of International Symposium on Communications and Information Technologies, 2006. ISCIT '06.*, pages 370–375, 18 2006-Sept. 20 2006. doi: 10.1109/ISCIT.2006.340067.

[66] Xin-Yu Shih, Cheng-Zhou Zhan, Cheng-Hung Lin, and An-Yeu Wu. An 8.29 mm2 52 mw multi-mode ldpc decoder design for mobile wimax system in 0.13 m cmos process. *IEEE Journal of Solid-State Circuits*, 43(3):672–683, March 2008. ISSN 0018-9200. doi: 10.1109/JSSC.2008.916606.

[67] Se-Hyeon Kang and In-Cheol Park. Loosely coupled memory-based decoding architecture for low density parity check codes. In *Proceedings of the IEEE Custom Integrated Circuits Conference, 2005.*, pages 703 – 706, sept. 2005. doi: 10.1109/CICC.2005.1568765.

[68] Kyung-Il Baek, Hanho Lee, Chang-Seok Choi, Sangmin Kim, and Gerald E. Sobelman. A high-throughput ldpc decoder architecture for high-rate wpan systems. In *IEEE International Symposium on Circuits and Systems (ISCAS), 2011*, pages 1311 –1314, may 2011. doi: 10.1109/ISCAS.2011.5937812.

[69] Hao Zhong and Tong Zhang. Block-ldpc: a practical ldpc coding system design approach. *IEEE Transactions on Circuits and Systems*, 52(4):766 – 775, april 2005. ISSN 1549-8328. doi: 10.1109/TCSI.2005.844113.

[70] Hao Zhong, Tong Zhang, and E.F. Haratsch. Vlsi design of high-rate quasi-cyclic ldpc codes for magnetic recording channel. In *Custom Integrated Circuits Conference, 2006. CICC '06. IEEE*, pages 325 –328, sept. 2006. doi: 10.1109/CICC.2006.320902.

[71] Kai Zhang, Xinming Huang, and Zhongfeng Wang. High-throughput layered decoder implementation for quasi-cyclic ldpc codes. *IEEE Journal on Selected Areas in Communications*, 27(6):985 –994, august 2009. ISSN 0733-8716. doi: 10.1109/JSAC.2009.090816.

[72] C. Studer, N. Preyss, C. Roth, and A. Burg. Configurable high-throughput decoder architecture for quasi-cyclic ldpc codes. In *Proceedings of 42nd Asilomar Conference on Signals, Systems and Computers, 2008*, pages 1137 –1142, oct. 2008. doi: 10.1109/ACSSC.2008.5074592.

[73] Yanni Chen and K.K. Parhi. Overlapped message passing for quasi-cyclic low-density parity check codes. *IEEE Transactions on Circuits and Systems*, 51(6):1106 – 1113, june 2004. ISSN 1549-8328. doi: 10.1109/TCSI.2004.826194.

[74] Hao Zhong, Tong Zhang, and E.F. Haratsch. Vlsi design of high-rate quasi-cyclic ldpc codes for magnetic recording channel. In *IEEE Custom Integrated Circuits Conference, 2006. CICC '06.*, pages 325 –328, sept. 2006. doi: 10.1109/CICC.2006.320902.

[75] Daesun Oh and K.K. Parhi. Efficient highly-parallel decoder architecture for quasi-cyclic low-density parity-check codes. In *IEEE International Symposium on Circuits and Systems, 2007. ISCAS 2007.*, pages 1855 –1858, may 2007. doi: 10.1109/ISCAS.2007.378276.

[76] Xiaoheng Chen, Jingyu Kang, Shu Lin, and V. Akella. Accelerating fpga-based emulation of quasi-cyclic ldpc codes with vector processing. In *Design, Automation Test in Europe Conference Exhibition, 2009. DATE '09.*, pages 1530 –1535, april 2009.

[77] Luoming Zhang, Lin Gui, Youyun Xu, and Wenjun Zhang. Configurable multi-rate decoder architecture for qc-ldpc codes based broadband broadcasting system. *IEEE Transactions on Broadcasting*, 54(2):226 –235, june 2008. ISSN 0018-9316. doi: 10.1109/TBC.2007.913400.

[78] Lingzhi Liu and C.-J.R. Shi. Sliced message passing: High throughput overlapped decoding of high-rate low-density parity-check codes. *IEEE Transactions on Circuits and Systems*, 55(11):3697 –3710, dec. 2008. ISSN 1549-8328. doi: 10.1109/TCSI.2008.926995.

[79] Jin Sha, Zhongfeng Wang, Minglun Gao, and Li Li. Multi-gb/s ldpc code design and implementation. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 17(2):262 –268, feb. 2009. ISSN 1063-8210. doi: 10.1109/TVLSI.2008.2002487.

[80] M. Karkooti and J.R. Cavallaro. Semi-parallel reconfigurable architectures for real-time ldpc decoding. In *Proceedings of International Conference on Information Technology: Coding and Computing, 2004.*, volume 1, pages 579 – 585 Vol.1, april 2004. doi: 10.1109/ITCC.2004.1286526.

[81] Yang Sun, Marjan Karkooti, and Joseph R. Cavallaro. High throughput, parallel, scalable ldpc encoder/decoder architecture for ofdm systems. In *IEEE Dallas/CAS Workshop on Design, Applications, Integration and Software, 2006*, pages 39 –42, oct. 2006. doi: 10.1109/DCAS.2006.321028.

[82] Yang Sun, Guohui Wang, and Joseph R. Cavallaro. Multi-layer parallel decoding algorithm and vlsi architecture for quasi-cyclic ldpc codes. In

IEEE International Symposium on Circuits and Systems (ISCAS), 2011, pages 1776 –1779, may 2011. doi: 10.1109/ISCAS.2011.5937928.

[83] V.K.K. Srinivasan, C.K. Singh, and P.T. Balsara. A generic scalable architecture for min-sum/offset-min-sum unit for irregular/regular ldpc decoder. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 18(9): 1372 –1376, sept. 2010. ISSN 1063-8210. doi: 10.1109/TVLSI.2009.2023659.

[84] Zebo Peng and Krzysztof Kuchcinski. An algorithm for partitioning of application specific systems. In *in Proceedings of the European Conference on Design Automation (EDAC*, pages 316–321, 1993.

[85] K. Kuchcinski. Embedded system synthesis by timing constraints solving. In *System Synthesis, 1997. Proceedings., Tenth International Symposium on*, pages 50 –57, sep 1997. doi: 10.1109/ISSS.1997.621675.

[86] Hyunok Oh and Soonhoi Ha. A hardware-software cosynthesis technique based on heterogeneous multiprocessor scheduling. In *Hardware/Software Codesign, 1999. (CODES '99) Proceedings of the Seventh International Workshop on*, pages 183 –187, 1999. doi: 10.1109/HSC.1999.777429.

[87] Karam S. Chatha and Ranga Vemuri. An iterative algorithm for hardware-software partitioning, hardware design space exploration and scheduling. *Design Automation for Embedded Systems*, 5:281–293, 2000. ISSN 0929-5585. URL `http://dx.doi.org/10.1023/A:1008954218909`. 10.1023/A:1008954218909.

[88] R.P. Dick and N.K. Jha. Mogac: a multiobjective genetic algorithm for hardware-software cosynthesis of distributed embedded systems. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 17 (10):920 –935, oct 1998. ISSN 0278-0070. doi: 10.1109/43.728914.

[89] J. Teich, T. Blickle, and L. Thiele. An evolutionary approach to system-level synthesis. In *Hardware/Software Codesign, 1997. (CODES/CASHE '97), Proceedings of the Fifth International Workshop on*, pages 167 –171, mar 1997. doi: 10.1109/HSC.1997.584597.

[90] A.D. Pimentel, C. Erbas, and S. Polstra. A systematic approach to exploring embedded system architectures at multiple abstraction levels. *IEEE Transactions on Computers*, 55(2):99 – 112, feb. 2006. ISSN 0018-9340. doi: 10.1109/TC.2006.16.

[91] H. Nikolov, T. Stefanov, and E. Deprettere. Systematic and automated multiprocessor system design, programming, and implementation. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 27(3):542 –555, march 2008. ISSN 0278-0070. doi: 10.1109/TCAD.2007.911337.

[92] H. Nikolov, M. Thompson, T. Stefanov, A. Pimentel, S. Polstra, R. Bose, C. Zissulescu, and E. Deprettere. Daedalus: toward composable multimedia mp-soc design. In *Proceedings of the 45th annual Design Automation Conference*, DAC '08, pages 574–579, New York, NY, USA, 2008. ACM. ISBN 978-1-60558-115-6. doi: 10.1145/1391469.1391615. URL http://doi.acm.org/10.1145/1391469.1391615.

[93] A. Gerstlauer, C. Haubelt, A.D. Pimentel, T.P. Stefanov, D.D. Gajski, and J. Teich. Electronic system-level synthesis methodologies. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 28(10): 1517 –1530, oct. 2009. ISSN 0278-0070. doi: 10.1109/TCAD.2009.2026356.

[94] Christophe Wolinski, Maya Gokhale, and Kevin McCabe. Polymorphous fabric-based systems: model, tools, applications. *J. Syst. Archit.*, 49:143–154, September 2003. ISSN 1383-7621. doi: 10.1016/S1383-7621(03)00074-2. URL http://dl.acm.org/citation.cfm?id=959096.959099.

[95] Christophe Wolinski and Krzysztof Kuchcinski. A constraints programming approach for fabric cell synthesis. In *Proceedings of the 8th Euromicro Conference on Digital System Design*, pages 356–363, Washington, DC, USA, 2005. IEEE Computer Society. ISBN 0-7695-2433-8. doi: 10.1109/DSD.2005.5. URL http://dl.acm.org/citation.cfm?id=1099541.1100220.

[96] Susan L. Graham, Peter B. Kessler, and Marshall K. McKusick. gprof: a call graph execution profiler. *SIGPLAN Not.*, 39:49–57, April 2004. ISSN 0362-1340. doi: http://doi.acm.org/10.1145/989393.989401. URL http://doi.acm.org/10.1145/989393.989401.

[97] GNU Gcov. Test coverage program. [Online] Available at: http://gcc.gnu.org/onlinedocs/gcc/Gcov.html/.

[98] Vfanalyst. Vector Fabrics. [Online] Available at: http://www.vectorfabrics.com/.

[99] Lance. A retargetable c compiler. [Online] Available at: http://www.lancecompiler.com/.

[100] Kingshuk Karuri, Mohammad Abdullah Al Faruque, Stefan Kraemer, Rainer Leupers, Gerd Ascheid, and Heinrich Meyr. Fine-grained application source code profiling for asip design. In *Proceedings of the 42nd annual Design Automation Conference*, DAC '05, pages 329–334, New York, NY, USA, 2005. ACM. ISBN 1-59593-058-2. doi: http://doi.acm.org/10.1145/1065579.1065666. URL http://doi.acm.org/10.1145/1065579.1065666.

[101] Lei Gao, Jia Huang, Jianjiang Ceng, Rainer Leupers, Gerd Ascheid, and Heinrich Meyr. Totalprof: a fast and accurate retargetable source code profiler. In *Proceedings of the 7th IEEE/ACM international conference on Hardware/software codesign and system synthesis*, CODES+ISSS '09, pages 305–314, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-628-1. doi: http://doi.acm.org/10.1145/1629435.1629477. URL http://doi.acm.org/10.1145/1629435.1629477.

[102] Amitabh Srivastava and Alan Eustace. Atom: a system for building customized program analysis tools. In *Proceedings of the ACM SIGPLAN 1994 conference on Programming language design and implementation*, PLDI '94, pages 196–205, New York, NY, USA, 1994. ACM. ISBN 0-89791-662-X. doi: 10.1145/178243.178260. URL http://doi.acm.org/10.1145/178243.178260.

[103] Cliff Young. The harvard atom-like tool (halt) manual, 1998.

[104] Robert F. Cmelik. Spixtools: Introduction and user's manual. Technical report, Mountain View, CA, USA, 1993.

[105] Tensilica. Xtensa Xplorer. [Online] Available at: http://www.tensilica.com/.

[106] Stretch. Stretch profiler for S5000 and S6000 families of software configurable processor. [Online] Available at: http://www.stretchinc.com/.

[107] The Stanford Suif Compiler Group, Stanford University. Available at: http://suif.stanford.edu/.

[108] M.W. Hall, J.M. Anderson, S.P. Amarasinghe, B.R. Murphy, Shih-Wei Liao, and E. Bu. Maximizing multiprocessor performance with the suif compiler. *Computer*, 29(12):84 –89, dec 1996. ISSN 0018-9162. doi: 10.1109/2.546613.

[109] M.B. Gokhale and J.M. Stone. Napa c: compiling for a hybrid risc/fpga architecture. In *Proceedings of IEEE Symposium on FPGAs for Custom Computing Machines, 1998.*, pages 126 –135, apr 1998. doi: 10.1109/FPGA.1998.707890.

[110] Yanbing Li, Tim Callahan, Ervan Darnell, Randolph Harr, Uday Kurkure, and Jon Stockwood. Hardware-software co-design of embedded reconfigurable architectures. In *Proceedings of the 37th Annual Design Automation Conference*, DAC '00, pages 507–512, New York, NY, USA, 2000. ACM. ISBN 1-58113-187-9. doi: http://doi.acm.org/10.1145/337292.337559. URL http://doi.acm.org/10.1145/337292.337559.

[111] M. Gokhale, J. Stone, J. Arnold, and M. Kalinowski. Stream-oriented fpga computing in the streams-c high level language. In *IEEE Symposium on Field-Programmable Custom Computing Machines, 2000*, pages 49 –56, 2000. doi: 10.1109/FPGA.2000.903392.

[112] Mihai Budiu and Seth Copen Goldstein. Compiling application-specific hardware. In *Proceedings of the Reconfigurable Computing Is Going Mainstream, 12th International Conference on Field-Programmable Logic and Applications*, FPL '02, pages 853–863, London, UK, UK, 2002. Springer-Verlag. ISBN 3-540-44108-5. URL http://dl.acm.org/citation.cfm?id=647929.740544.

[113] RAW architecture. Massachusetts Institute of Technology, Labaratoryfor Computer Science. [Online] Available at: http://www.cag.csail.mit.edu/raw/.

[114] Kiran Bondalapati, Pedro C. Diniz, Phillip Duncan, John Granacki, Mary W. Hall, Rajeev Jain, and Heidi Ziegler. Defacto: A design environment for adaptive computing technology. In *Proceedings of the 11 IPPS/SPDP'99 Workshops Held in Conjunction with the 13th International Parallel Processing Symposium and 10th Symposium on Parallel and Distributed Processing*, pages 570–578, London, UK, 1999. Springer-Verlag. ISBN 3-540-65831-9. URL http://dl.acm.org/citation.cfm?id=645611.662348.

[115] W.A. Najjar. Compiling code accelerators for fpgas. In *Hardware/Software Codesign and System Synthesis (CODES+ISSS), 2007 5th IEEE/ACM/IFIP International Conference on*, page 2, 30 2007-oct. 3 2007.

[116] P. Banerjee, N. Shenoy, A. Choudhary, S. Hauck, C. Bachmann, M. Haldar, P. Joisha, A. Jones, A. Kanhare, A. Nayak, S. Periyacheri, M. Walkden, and D. Zaretsky. A matlab compiler for distributed, heterogeneous, reconfigurable computing systems. In *IEEE Symposium on Field-Programmable Custom Computing Machines, 2000*, pages 39 –48, 2000. doi: 10.1109/FPGA.2000.903391.

[117] O. Mencer. Asc: a stream compiler for computing with fpgas. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 25 (9):1603 –1617, sept. 2006. ISSN 0278-0070. doi: 10.1109/TCAD.2005.857377.

[118] Trimaran. An Infrastructure for Research in Backend Compilation and Architecture Exploration. [Online] Available at: http://www.trimaran.org/.

[119] Z. Nawaz, O.S. Dragomir, T. Marconi, E.M. Panainte, K. Bertels, and S. Vassiliadis. Recursive variable expansion: A loop transformation for reconfigurable systems. In *Proceedings of International Conference on Field-Programmable Technology, 2007. ICFPT 2007.*, pages 301 –304, dec. 2007. doi: 10.1109/FPT.2007.4439271.

[120] Z. Nawaz, T. Marconi, K. Bertels, and T. Stefanov. Flexible pipelining design for recursive variable expansion. In *Proceedings of IEEE International Symposium on Parallel Distributed Processing, 2009. IPDPS 2009.*, pages 1 –8, may 2009. doi: 10.1109/IPDPS.2009.5161215.

[121] O.S. Dragomir, T. Stefanov, and K. Bertels. Loop unrolling and shifting for reconfigurable architectures. In *Proceedings of International Conference on Field Programmable Logic and Applications, 2008. FPL 2008.*, pages 167 –172, sept. 2008. doi: 10.1109/FPL.2008.4629926.

[122] Utpal K. Banerjee. *Loop Transformations for Restructuring Compilers: The Foundations*. Kluwer Academic Publishers, Norwell, MA, USA, 1993. ISBN 079239318X.

[123] A. Aiken and A. Nicolau. Optimal loop parallelization. In *Proceedings of the ACM SIGPLAN 1988 conference on Programming Language design and Implementation*, PLDI '88, pages 308–317, New York, NY, USA, 1988. ACM. ISBN 0-89791-269-1. doi: http://doi.acm.org/10.1145/53990.54021. URL `http://doi.acm.org/10.1145/53990.54021`.

[124] A. Aiken and A. Nicolau. Optimal loop parallelization. *SIGPLAN Not.*, 23:308–317, June 1988. ISSN 0362-1340. doi: http://doi.acm.org/10.1145/960116.54021. URL `http://doi.acm.org/10.1145/960116.54021`.

[125] Alain Darte, Yves Robert, and Frédéric Vivien. Compiler optimizations for scalable parallel systems. chapter Loop parallelization algorithms, pages 141–171. Springer-Verlag New York, Inc., New York, NY, USA, 2001. ISBN 3-540-41945-4. URL `http://dl.acm.org/citation.cfm?id=380466.380471`.

[126] K. W. Rudd and M. J. Flynn. Instruction-level parallel processors-dynamic and static scheduling tradeoffs. In *Proceedings of the 2nd AIZU International Symposium on Parallel Algorithms / Architecture Synthesis*, PAS '97, pages 74–, Washington, DC, USA, 1997. IEEE Computer Society. ISBN 0-8186-7870-4. URL `http://dl.acm.org/citation.cfm?id=523978.826105`.

[127] J.A. Fisher. Trace scheduling: A technique for global microcode compaction. *IEEE Transactions on Computers*, C-30(7):478 –490, july 1981. ISSN 0018-9340. doi: 10.1109/TC.1981.1675827.

[128] Wen Mei W. Hwu, Scott A. Mahlke, William Y. Chen, Pohua P. Chang, Nancy J. Warter, Roger A. Bringmann, Roland G. Ouellette, Richard E. Hank, Tokuzo Kiyohara, Grant E. Haab, John G. Holm, and Daniel M. Lavery. The superblock: An effective technique for vliw and superscalar compilation. *The Journal of Supercomputing*, 7:229–248, 1993. ISSN 0920-8542. URL http://dx.doi.org/10.1007/BF01205185. 10.1007/BF01205185.

[129] Richard E. Hank, Scott A. Mahlke, Roger A. Bringmann, John C. Gyllenhaal, and Wen-mei W. Hwu. Superblock formation using static program analysis. In *Proceedings of the 26th annual international symposium on Microarchitecture*, MICRO 26, pages 247–255, Los Alamitos, CA, USA, 1993. IEEE Computer Society Press. ISBN 0-8186-5280-2. URL http://dl.acm.org/citation.cfm?id=255235.255298.

[130] Scott A. Mahlke, David C. Lin, William Y. Chen, Richard E. Hank, and Roger A. Bringmann. Effective compiler support for predicated execution using the hyperblock. *SIGMICRO Newsl.*, 23:45–54, December 1992. ISSN 1050-916X. doi: http://doi.acm.org/10.1145/144965.144998. URL http://doi.acm.org/10.1145/144965.144998.

[131] Timothy J. Callahan, John R. Hauser, and John Wawrzynek. The garp architecture and c compiler. *Computer*, 33:62–69, April 2000. ISSN 0018-9162. doi: 10.1109/2.839323. URL http://dl.acm.org/citation.cfm?id=619050.621455.

[132] Michael R. Garey and David S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1990. ISBN 0716710455.

[133] J. D. Ullman. Np-complete scheduling problems. *J. Comput. Syst. Sci.*, 10:384–393, June 1975. ISSN 0022-0000. doi: http://dx.doi.org/10.1016/S0022-0000(75)80008-0. URL http://dx.doi.org/10.1016/S0022-0000(75)80008-0.

[134] Jacek Bazewicz and Et Al. *Scheduling Computer and Manufacturing Processes*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2nd edition, 2001. ISBN 3540419314.

[135] J. A. Fisher. Trace scheduling: A technique for global microcode compaction. *IEEE Trans. Comput.*, 30:478–490, July 1981. ISSN 0018-9340. doi: http://dx.doi.org/10.1109/TC.1981.1675827. URL http://dx.doi.org/10.1109/TC.1981.1675827.

[136] P.G. Paulin and J.P. Knight. Force-directed scheduling for the behavioral synthesis of asics. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 8(6):661 –679, jun 1989. ISSN 0278-0070. doi: 10.1109/43.31522.

[137] Roni Potasman, Joseph Lis, Alexandru Nicolau, and Daniel Gajski. Percolation based synthesis. In *Proceedings of the 27th ACM/IEEE Design Automation Conference*, DAC '90, pages 444–449, New York, NY, USA, 1990. ACM. ISBN 0-89791-363-9. doi: http://doi.acm.org/10.1145/123186.123333. URL http://doi.acm.org/10.1145/123186.123333.

[138] Youn-Long Lin. Recent developments in high-level synthesis. *ACM Trans. Des. Autom. Electron. Syst.*, 2:2–21, January 1997. ISSN 1084-4309. doi: http://doi.acm.org/10.1145/250243.250245. URL http://doi.acm.org/10.1145/250243.250245.

[139] Carnegie-Mellon University. SRC-CMU Research Center for Computer-Aided Design, R.A. Walker, and Semiconductor Research Corporation. *A survey of high-level synthesis systems*. 1989. URL http://books.google.com/books?id=1jrbHwAACAAJ.

[140] Giovanni De Micheli. *Synthesis and Optimization of Digital Circuits*. McGraw-Hill Higher Education, 1st edition, 1994. ISBN 0070163332.

[141] Reinaldo A. Bergamaschi, Salil Raje, and Louise Trevillyan. Control-flow versus data-flow-based scheduling: combining both approaches in an adaptive scheduling system. *IEEE Trans. Very Large Scale Integr. Syst.*, 5: 82–100, March 1997. ISSN 1063-8210. doi: 10.1109/92.555989. URL http://dl.acm.org/citation.cfm?id=249537.249562.

[142] Petru Eles, Krzysztof Kuchcinski, and Zebo Peng. *System Synthesis with VHDL: A Transformational Approach*. Kluwer Academic Publishers, Norwell, MA, USA, 1998. ISBN 0792380827.

[143] G. Constantinides, P. Cheung, and W. Luk. Heuristic datapath allocation for multiple wordlength systems. In *Proceedings of the conference on Design, automation and test in Europe*, DATE '01, pages 791–797, Piscataway, NJ, USA, 2001. IEEE Press. ISBN 0-7695-0993-2. URL http://dl.acm.org/citation.cfm?id=367072.367979.

[144] Deming Chen, Jason Cong, and Yiping Fan. Low-power high-level synthesis for fpga architectures. In *Proceedings of the 2003 international symposium on Low power electronics and design*, ISLPED '03, pages 134–139, New York, NY, USA, 2003. ACM. ISBN 1-58113-682-X. doi: http://doi.acm.org/10.1145/871506.871541. URL http://doi.acm.org/10.1145/871506.871541.

[145] Hassan Al Atat and Iyad Ouaiss. Register binding for fpgas with embedded memory. In *Proceedings of the 12th Annual IEEE Symposium on Field-Programmable Custom Computing Machines*, pages 167–175, Washington, DC, USA, 2004. IEEE Computer Society. ISBN 0-7695-2230-0. URL http://dl.acm.org/citation.cfm?id=1025123.1025826.

[146] Annie Avakian and Iyad Ouaiss. Optimizing register binding in fpgas using simulated annealing. In *Proceedings of the 2005 International Conference on Reconfigurable Computing and FPGAs (ReConFig'05) on Reconfigurable Computing and FPGAs*, pages 16–, Washington, DC, USA, 2005. IEEE Computer Society. ISBN 0-7695-2456-7. doi: 10.1109/RECONFIG.2005.27. URL http://dl.acm.org/citation.cfm?id=1114693.1115261.

[147] Jason Cong, Yiping Fan, Guoling Han, Yizhou Lin, Junjuan Xu, Zhiru Zhang, and Xu Cheng. Bitwidth-aware scheduling and binding in high-level synthesis. In *Proceedings of the 2005 Asia and South Pacific Design Automation Conference*, ASP-DAC '05, pages 856–861, New York, NY, USA, 2005. ACM. ISBN 0-7803-8737-6. doi: http://doi.acm.org/10.1145/1120725.1121055. URL http://doi.acm.org/10.1145/1120725.1121055.

[148] Jason Cong, Yiping Fan, and Wei Jiang. Platform-based resource binding using a distributed register-file microarchitecture. In *Proceedings of the 2006 IEEE/ACM international conference on Computer-aided design*, ICCAD '06, pages 709–715, New York, NY, USA, 2006. ACM. ISBN 1-59593-389-1. doi: http://doi.acm.org/10.1145/1233501.1233648. URL http://doi.acm.org/10.1145/1233501.1233648.

[149] Taemin Kim and Xun Liu. Compatibility path based binding algorithm for interconnect reduction in high level synthesis. In *Proceedings of the 2007 IEEE/ACM international conference on Computer-aided design*, ICCAD '07, pages 435–441, Piscataway, NJ, USA, 2007. IEEE Press. ISBN 1-4244-1382-6. URL http://dl.acm.org/citation.cfm?id=1326073.1326164.

[150] Lech Jozwiak, Dominik Gawlowski, and Aleksander Slusarczyk. Multi-objective optimal controller synthesis for heterogeneous embedded systems. In *Proceedings of International Conference on Embedded Computer Systems: Architectures, Modeling and Simulation, 2006. IC-SAMOS 2006.*, pages 177 –184, july 2006. doi: 10.1109/ICSAMOS.2006.300825.

[151] Saraju P. Mohanty, Nagarajan Ranganathan, Elias Kougianos, and Priyardarsan Patra. *Low-Power High-Level Synthesis for Nanoscale CMOS Circuits*. Springer Publishing Company, Incorporated, 1 edition, 2008. ISBN 0387764739, 9780387764733.

[152] R. Composano. Path-based scheduling for synthesis. In *Proceedings of the Twenty-Third Annual Hawaii International Conference on System Sciences, 1990.*, volume i, pages 348 –355 vol.1, jan 1990. doi: 10.1109/HICSS.1990.205135.

[153] I. Radivojevic and F. Brewer. Incorporating speculative execution in exact control-dependent scheduling. In *Design Automation, 1994. 31st Conference on*, pages 479 – 484, june 1994. doi: 10.1109/DAC.1994.204150.

[154] Milan Vasilko and Djamel Ait-Boudaoud. Architectural synthesis techniques for dynamically reconfigurable logic. In *Proceedings of the 6th International Workshop on Field-Programmable Logic, Smart Applications, New Paradigms and Compilers*, pages 290–296, London, UK, 1996. Springer-Verlag. ISBN 3-540-61730-2. URL `http://dl.acm.org/citation.cfm?id=647923.741204`.

[155] Karthikeya M. Gajjala Purna and Dinesh Bhatia. Temporal partitioning and scheduling data flow graphs for reconfigurable computers. *IEEE Trans. Comput.*, 48:579–590, June 1999. ISSN 0018-9340. doi: 10.1109/12.773795. URL `http://dl.acm.org/citation.cfm?id=306886.306889`.

[156] M. Kaul and R. Vemuri. Optimal temporal partitioning and synthesis for reconfigurable architectures. In *Proceedings of the conference on Design, automation and test in Europe*, DATE '98, pages 389–397, Washington, DC, USA, 1998. IEEE Computer Society. ISBN 0-8186-8359-7. URL `http://dl.acm.org/citation.cfm?id=368058.368247`.

[157] Iyad Ouaiss, Sriram Govindarajan, Vinoo Srinivasan, Meenakshi Kaul, and Ranga Vemuri. An integrated partitioning and synthesis system for dynamically reconfigurable multi-fpga architectures, 1998.

[158] Iyad Ouaiss, Sriram Govindarajan, Vinoo Srinivasan, Meenakshi Kaul, and Ranga Vemuri. A unified specification model of concurrency and coordination for synthesis from vhdl. In *Proceedings of the 4th International Conference on Information Systems Analysis and Synthesis*, pages 771–778, 1998.

[159] Kamlesh Rath and Jian Li. Synthesizing reconfigurable sequential machines using tabular models. In *Proceedings of the 5th Reconfigurable Architectures Workshop (RAW-98*, 1998.

[160] Hritam Dutta, Dmitrij Kissler, Frank Hannig, Alexey Kupriyanov, Jürgen Teich, and Bernard Pottier. A holistic approach for tightly coupled reconfigurable parallel processors. *Microprocess. Microsyst.*, 33:53–62, February 2009. ISSN 0141-9331. doi: 10.1016/j.micpro.2008.08.007. URL `http://dl.acm.org/citation.cfm?id=1502820.1503070`.

[161] Bingfeng Mei, Serge Vernalde, Diederik Verkest, Hugo De Man, and Rudy Lauwereins. Exploiting loop-level parallelism on coarse-grained reconfigurable architectures using modulo scheduling. In *Proceedings of the conference on Design, Automation and Test in Europe - Volume 1*, DATE '03, pages 10296–, Washington, DC,

USA, 2003. IEEE Computer Society. ISBN 0-7695-1870-2. URL http://dl.acm.org/citation.cfm?id=789083.1022741.

[162] Girish Venkataramani, Walid Najjar, Fadi Kurdahi, Nader Bagherzadeh, Wim Bohm, and Jeff Hammes. Automatic compilation to a coarse-grained reconfigurable system-opn-chip. *ACM Trans. Embed. Comput. Syst.*, 2:560–589, November 2003. ISSN 1539-9087. doi: http://doi.acm.org/10.1145/950162.950167. URL http://doi.acm.org/10.1145/950162.950167.

[163] S. Gupta, N. Dutt, R. Gupta, and A. Nicolau. Loop shifting and compaction for the high-level synthesis of designs with complex control flow. In *Proceedings of Design, Automation and Test in Europe Conference and Exhibition, 2004.*, volume 1, pages 114 – 119 Vol.1, feb. 2004. doi: 10.1109/DATE.2004.1268836.

[164] F. Hannig, H. Dutta, and J. Teich. Mapping of regular nested loop programs to coarse-grained reconfigurable arrays - constraints and methodology. In *Proceedings of 18th International Parallel and Distributed Processing Symposium, 2004.*, page 148, april 2004. doi: 10.1109/IPDPS.2004.1303132.

[165] Sudarshan Banerjee, Elaheh Bozorgzadeh, and Nikil Dutt. Parlgran: parallelism granularity selection for scheduling task chains on dynamically reconfigurable architectures. In *Proceedings of the 2006 Asia and South Pacific Design Automation Conference*, ASP-DAC '06, pages 491–496, Piscataway, NJ, USA, 2006. IEEE Press. ISBN 0-7803-9451-8. doi: http://dx.doi.org/10.1145/1118299.1118419. URL http://dx.doi.org/10.1145/1118299.1118419.

[166] Yazhuo Dong, Jie Zhou, Yong Dou, Lin Deng, and Jinjing Zhao. Impact of loop unrolling on area, throughput and clock frequency for window operations based on a data schedule method. In *Proceedings of the 2008 Congress on Image and Signal Processing, Vol. 1 - Volume 01*, CISP '08, pages 641–645, Washington, DC, USA, 2008. IEEE Computer Society. ISBN 978-0-7695-3119-9. doi: http://dx.doi.org/10.1109/CISP.2008.211. URL http://dx.doi.org/10.1109/CISP.2008.211.

[167] Steven Novack and Alexandru Nicolau. Resource-directed loop pipelining. In *Proceedings of the 9th International Workshop on Languages and Compilers for Parallel Computing*, LCPC '96, pages 192–206, London, UK, 1997. Springer-Verlag. ISBN 3-540-63091-0. URL http://dl.acm.org/citation.cfm?id=645674.663454.

[168] Yanbing Li, Tim Callahan, Ervan Darnell, Randolph Harr, Uday Kurkure, and Jon Stockwood. Hardware-software co-design of embedded reconfigurable architectures. In *Proceedings of the 37th Annual Design Automation*

Conference, DAC '00, pages 507–512, New York, NY, USA, 2000. ACM. ISBN 1-58113-187-9. doi: http://doi.acm.org/10.1145/337292.337559. URL http://doi.acm.org/10.1145/337292.337559.

[169] Timothy J. Callahan and John Wawrzynek. Instruction-level parallelism for reconfigurable computing. In *Proceedings of the 8th International Workshop on Field-Programmable Logic and Applications, From FPGAs to Computing Paradigm*, pages 248–257, London, UK, 1998. Springer-Verlag. ISBN 3-540-64948-4. URL http://dl.acm.org/citation.cfm?id=647925.739053.

[170] Timothy J. Callahan and John Wawrzynek. Adapting software pipelining for reconfigurable computing. In *Proceedings of the 2000 international conference on Compilers, architecture, and synthesis for embedded systems*, CASES '00, pages 57–64, New York, NY, USA, 2000. ACM. ISBN 1-58113-338-3. doi: http://doi.acm.org/10.1145/354880.354889. URL http://doi.acm.org/10.1145/354880.354889.

[171] M. Weinhardt and W. Luk. Pipeline vectorization. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 20(2):234 –248, feb 2001. ISSN 0278-0070. doi: 10.1109/43.908452.

[172] Steven Derrien and Sanjay V. Rajopadhye. Loop tiling for reconfigurable accelerators. In *Proceedings of the 11th International Conference on Field-Programmable Logic and Applications*, FPL '01, pages 398–408, London, UK, 2001. Springer-Verlag. ISBN 3-540-42499-7. URL http://dl.acm.org/citation.cfm?id=647928.739897.

[173] M. Wazlowski, L. Agarwal, T. Lee, A. Smith, E. Lam, P. Athanas, H. Silverman, and S. Ghosh. Prism-ii compiler and architecture. In *Proceedings of IEEE Workshop on FPGAs for Custom Computing Machines, 1993.*, pages 9 –16, apr 1993. doi: 10.1109/FPGA.1993.279484.

[174] J. Babb, M. Rinard, C.A. Moritz, W. Lee, M. Frank, R. Barua, and S. Amarasinghe. Parallelizing applications into silicon. In *Proceedings of Seventh Annual IEEE Symposium on Field-Programmable Custom Computing Machines, 1999. FCCM '99*, pages 70 –80, 1999. doi: 10.1109/FPGA.1999.803669.

[175] Cameron project. Colorado State University. [Online] Available at: http://www.cs.colostate.edu/.

[176] Forte Design Systems. Cynthesizer Closes the ESL-to-Silicon Gap. Available at: http://www.forteds.com/products/cynthesizer.asp/.

[177] Simulink HDL Coder. MathWorks Inc. [Online] Available at: http://www.mathworks.com/products/slhdlcoder/.

[178] Philippe Coussy, Cyrille Chavet, Pierre Bomel, Dominique Heller, Eric Senn, and Eric Martin. GAUT: A High-Level Synthesis Tool for DSP Applications. In Philippe Coussy and Adam Morawiec, editors, *High-Level Synthesis*, chapter 9, pages 147–169. Springer Netherlands, 2008. ISBN 978-1-4020-8587-1. doi: 10.1007/978-1-4020-8588-8_9. URL http://dx.doi.org/10.1007/978-1-4020-8588-8_9.

[179] LLVM. The LLVM Compiler Infrastructure. [Online] Available at: http://llvm.org/.

[180] Andrzej P. Wierzbicki. Reference point approaches and objective ranking. In Jürgen Branke, Kalyanmoy Deb, Kaisa Miettinen, and Roman Slowinski, editors, *Practical Approaches to Multi-Objective Optimization*, number 06501 in Dagstuhl Seminar Proceedings, Dagstuhl, Germany, 2007. Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI), Schloss Dagstuhl, Germany. URL http://drops.dagstuhl.de/opus/volltexte/2007/1121.

[181] Lech Jozwiak. Advanced ai search techniques in modern digital circuit synthesis. *Artificial Intelligence Review*, 20:269–318, 2003. ISSN 0269-2821. URL http://dx.doi.org/10.1023/B:AIRE.0000006607.94352.28. 10.1023/B:AIRE.0000006607.94352.28.

[182] Kalyanmoy Deb and J. Sundar. Reference point based multi-objective optimization using evolutionary algorithms. In *Proceedings of the 8th annual conference on Genetic and evolutionary computation*, GECCO '06, pages 635–642, New York, NY, USA, 2006. ACM. ISBN 1-59593-186-4. doi: 10.1145/1143997.1144112. URL http://doi.acm.org/10.1145/1143997.1144112.

[183] K. Deb and A. Kumar. Light beam search based multi-objective optimization using evolutionary algorithms. In *IEEE Congress on Evolutionary Computation, 2007. CEC 2007.*, pages 2125 –2132, sept. 2007. doi: 10.1109/CEC.2007.4424735.

Glossary

Acronyms and abbreviations

APP	Application analysis and parallelization
ASIC	Application-specific integrated circuit
ASIP	Application-specific instruction-set processor
BER	Bit-error-rate
BFS	Breadth-first-search
CN	Check node
CNP	Check node processor
DFS	Depth-first-search
DSE	Design space exploration
DSP	Digital signal processing
EDA	Electronic design automation
FER	Frame-error-rate
FPGA	Field-programmable gate array
Gbps	Gigabits per second
GPGPU	General purpose computation on graphics processing units
GPP	General purpose processor
HLS	High-level-synthesis

LDPC	Low density parity check codes
Mbps	Megabits per second
MIMO	Multi-input multi-output operations
MODM	Multi-objective decision making
MPA	Multi-processor accelerator
MPSoC	Multi-processor system-on-chip
NOC	Network-on-chip
PCM	Parity check matrix
QoR	Quality-of-results
RISC	Reduced instruction set computing
RS	Relaxed scheduling
RTL	Register-transfer-level
SoC	System-on-chip
SRAM	Static random access memory
TS	Tight scheduling
UHDTV	Ultra-high-definition digital television
VN	Variable node
VNP	Variable node processor

Curriculum Vitae

Yahya Jan was born in Charsadda, Pakistan on March 3, 1980. He completed his bachelor (B.Sc) in Electrical and Electronic Engineering in 2003 (3rd Position at the University level) from N.W.F.P University of Engineering and Technology Peshawar, Pakistan. In 2005, he received his master (M.Sc) degree in Information Technology (IT) from Pakistan Institute of Engineering and Applied Sciences (PIEAS), Islamabad, Pakistan. From 2006 till 2008, he was working as a system and hardware researcher and developer in the research and development (R&D) institute Informatics Complex in Pakistan. He was responsible for the research and development of embedded FPGA-based controllers for various industrial monitoring and control applications. In 2008, he started his PhD research project in the Faculty of Electrical Engineering (Electronic Systems group) at the Eindhoven University of Technology (TU/e), under the direct supervision of Dr. Lech Jozwiak. His research project was funded by the Higher Education Commission (HEC) Pakistan in collaboration with NUFFIC Netherlands. During his work at TU/e, he assisted in the design-oriented education and co-supervised several student projects. He reviewed several papers for international scientific journals and conferences in the area of his research. Results of his PhD research project include: a method for multi-processor hardware accelerator design and related prototype design automation tool, as well as, his PhD thesis and several journal and conference papers discussing the problems researched and method developed.

List of Publications

Journal Articles

1. Yahya Jan, Lech Jozwiak, Scalable communication architectures for massively parallel hardware multi-processors. In *Journal of Parallel and Distributed Computing, Available online 7 February 2012, ISSN 0743-7315, 10.1016/j.jpdc.2012.01.017.*
(http://www.sciencedirect.com/science/article/pii/S0743731512000299)

2. Yahya Jan, Lech Jozwiak. Communication and Memory Architecture Design of Application-specific High-end Multi-processors. In *journal of VLSI Design Volume 2012, Article ID 794753, 20 pages, doi:10.1155/2012/794753. Hindawi Publishing Corporation.*

Conference Papers

1. Yahya Jan, Lech Jozwiak. Survey of advanced CABAC accelarator architectures for future multimedia. In *Proceedings 5th international workshop on Reconfigurable computing architectures, tools and applications ARC 2009.* Karlsruhe, Germany, March 16-18, 2009 (Lecture Notes in Computer Science, Vol. 5453, pp. 342-348). Springer.

2. Yahya Jan, Lech Jozwiak. CABAC accelerator architectures for video compression in future multimedida : a survey. In *Proceedings 9th international workshop on Embedded computer systems : architectures, modeling, and simulation, SAMOS 2009.* July 20-23, 2009, Samos, Greece. (Lecture Notes in Computer Science, Vol. 5657, pp. 24-35). Springer.

3. Lech Jozwiak, Yahya Jan. Design of Hardware Accelerators for Demanding Applications. In *Proceedings of the IEEE Latin American Symposium on Circuits and Systems*. 24-26 Feb 2010 Brazil. (pp. 252-255). IEEE.

4. Lech Jozwiak, Yahya Jan. Quality-driven methodology for demanding accelarator design. In *Proceedings of the IEEE International Conference on Quality Electronic Design 2010, ISQUED 2010*. March 22-24, 2010, San Jose, USA. (pp. 380-389). Los Alamitos: IEEE Computer Society Press.

5. Lech Jozwiak, Yahya Jan. Architecture design of reconfigurable accelerators for demanding apllications. In *Proceedings of the International Conference on Information Technology: New Generations ITNG 2010*. Las Vegas, USA, 12-14 April 2010. (pp. 1201-1206). Los Alamitos: IEEE Computer Society Press.

6. Yahya Jan, Lech Jozwiak. Architecture Exploration of Re-configurable Hardware Accelerators for Highly-demanding Applications. In *Proceedings of the 1st STW.ICT Conference*. 18-11-2010 Veldhoven, Netherlands.

Note: The contents of this thesis is based partially on the above-mentioned journal and conference publications. In particular, the contents of chapter 2 and 3 are based on the conference publications {1, 2, 3, 4, 5, 6} and {3, 4, 5, 6}, respectively, while the contents of chapter 5 and 6 are based on the journal publications.

Reader's Notes