Quality-driven methodology for demanding accelarator design

Quality-Driven Methodology for Demanding Accelerator Design

Lech Jóźwiak, Yahya Jan
Eindhoven University of Technology
L.Jozwiak@tue.nl

Abstract

This paper focuses on mastering the architecture development of hardware accelerators for demanding applications. It presents the results of our analysis of the main problems that have to be solved when designing accelerators for modern demanding applications, and illustrates the problems with an example of accelerator design for LDPC code decoders for the newest communication system standards. Based on the results of our analysis, we formulate the main requirements that have to be satisfied by an adequate methodology for demanding accelerator design, and propose an architecture design methodology which satisfies the requirements.

Keywords

hardware accelerators, architecture design, design-space exploration, combined macro- and micro-architecture design

1. Introduction

The recent spectacular progress in modern nano-electronic technology enabled implementation of complex information processing systems on single chips, and facilitated rapid progress in mobile and autonomous computing, as well as, wire-less and wired communication. On the other hand however, it introduced unusual technology and system complexity. Increasingly complex and sophisticated systems are required to reliably perform real-time computations to extremely tight schedules with energy, power and area efficiency never demanded before. In consequence, opportunities created by modern technology can effectively be exploited only through adequate usage of efficient application-specific system architectures and circuit implementations exploiting more adequate concepts of computation, storage and communication. This requires effective and 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.

Numerous embedded system projects have demonstrated that heterogeneous systems, exploiting a mixture of different programmable and hardwired processors customized to different parts of complex applications provide drastically higher performance and lower power consumption than traditional homogeneous systems. As critical parts of these heterogeneous systems hardware accelerators have to be developed for the most demanding parts of the applications.

This paper focuses on mastering the architecture development of hardware accelerators for demanding applications. It presents the results of our analysis of the main problems that have to be solved in design of accelerators for modern demanding applications, and illustrates the problems with design of hardware accelerators for LDPC code decoders for the newest demanding communication system standards. The analysis presented demonstrates that the today's high-level synthesis is not sufficient to adequately support the complex hardware accelerator design process for the modern demanding applications. A more complex and sophisticated design methodology is needed. Based on the results of our analysis, we formulated and present in this paper the main requirements that have to be satisfied by an adequate methodology for demanding accelerator design, and propose a quality-driven accelerator design methodology which satisfies the requirements. This methodology is being used to design hardware accelerators for different LDPC code decoders for some of the newest communication system standards.

2. Issues and requirements of demanding accelerator design

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++ or SystemC 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 [4]-[11]. 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 [1] [2]:

- 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 high-level synthesis [4]-[11] and emerging commercial high-level synthesis tools [12]. 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 [2]), what increases difficulty of an adequate pipeline construction.

However, many modern applications (e.g. various decoders in (wireless) communication and multimedia, network access nodes, encryption applications, transformations in medical image processing etc.) are of different kind. They require hardware acceleration of critical information processing algorithms that involve data parallelism and complex interrelationships between the data and computing operations that have to be performed on the data. This results in complex (global) memory accesses and complex communication between the memories and computing units in the related hardware accelerators. For this kind of applications, the problems of hardware accelerator design are not limited to an adequate micro-architecture design for computing units. The main design problems are related to an adequate resolution of memory and communication bottlenecks, and to decreasing the memory and communication hardware complexity, what has to be achieved through an adequate memory and communication structure design. 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 (what results in e.g. vector, multi-bank or multi-port memories) and simultaneous transmission of the data (what 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.).

Finally, many of the modern demanding applications involve algorithms with massive data parallelism at the macro-level or task-level functional parallelism (e.g. LDPC code decoders of the newest communication system standards like IEEE 802.11n, 802.16e, 802.15.3c, 802.3an, 802.15.3c, 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 trade-off 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 trade-off 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 macro-/micro-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.

From the above it should be clear, that the existing high-level synthesis, specifically developed and limited to RTL-level micro-architecture synthesis of processing units, is only able to partly support the internal architecture design for particular computation units, and is not sufficient to adequately support the total complex 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 accelerators than the existing high-level synthesis. This new methodology should of course account for the computation unit micro-architecture synthesis (being the subject of high-level synthesis), but it has also to adequately address many more issues, including:

- memory and communication structure synthesis,
- macro-architecture synthesis of the multi-accelerator structures,
- strong interrelationships between the computation unit, memory and communication organization, and between the micro-and macro-architecture, and
- tradeoff exploitation between the micro and macro-architecture, and between the computation unit, memory and communication structures.

Only an appropriate accelerator architecture design space exploration and trade-off 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 design-space exploration by "what if" analysis, than on the fully automatic synthesis of individual computing units as offered by the today's high-level synthesis tools. At least partially, a different kind of design support is crucial here than that offered by the traditional high-level synthesis. High-level synthesis tools and other automatic synthesis 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.

To satisfy the needs of an adequate design of the modern demanding accelerators, we propose in this paper a quality-driven design methodology which satisfies the above discussed requirements. In the sequel to this paper, we use the accelerator design process for LDPC decoders to illustrate and further explain the above discussed issues and requirements of demanding accelerator design, and to introduce and illustrate our proposed accelerator design methodology.

3. Main issues of accelerator design for LDPC decoders

A systematic LDPC encoder encodes a message of k information bits into a codeword of length n with the k message bits followed by m parity checks, as shown in Fig.1. The parity checks are computed using a parity generator matrix (PGM) G of size kxn. Each parity check is computed based on a sub-set of message bits. The codeword is transmitted through a communication channel to the decoder. The decoder checks the validity of the received codeword by re-computing the parity checks, using a sparse binary matrix H of size mxn, called parity check matrix (PCM). For a codeword to be valid, it must satisfy the set of all m parity checks. In Figure 2 an example PCM for a (7,4) LDPC code is given. ``1" in a position $H_{i,j}$ of this matrix means that a particular bit participates in a parity check equation. For example, in the first row the bits at positions b_0, b_2, b_3, b_4 participate in the computation of the parity check c_0, that is, $c_0 = b_0 \oplus b_2 \oplus b_3 \oplus b_4$, where $\oplus$ represents the exclusive-OR operation.

Each parity check matrix can be represented by its corresponding bipartite graph (Tanner graph) [13]. The Tanner graph corresponding to an *(n, k)* LDPC code consists of n variable (bit) nodes (VN) and $m = n - k$ check nodes (CN), connected with each other through edges, as shown in Figure 2. Each row in the parity check matrix represents a parity check equation c_i, $0 \le i \le m-1$, and each column represents a coded bit b_j, $0 \le j \le n-1$. An edge exists between a CN i and VN j, if the corresponding value $H_{i,j}$ is non-zero in the PCM.

Usually, iterative Message Passing Algorithms (MPA) [14] 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 [14]. 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 MPA algorithm are graphically represented in Fig. 3. Since the Tanner graphs corresponding to practical LDPC codes of the newest

Fig. 1: LDPC encoding and decoding process

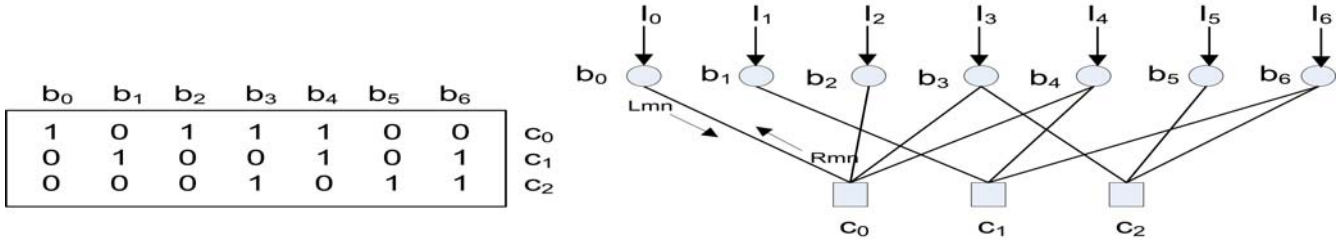

Fig. 2: PCM for a (7,4) LDPC code and its corresponding Tanner graph, where $\{b_0 \ldots b_6\}$ represents variable (bit) nodes, $\{c_0 \ldots . c_3\}$ represents check nodes and $\{I_0 \ldots . . I_6\}$ represents the input intrinsic channel information.

communication system standards involve hundreds variable 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 so high throughput complex highly parallel hardware accelerators are necessary.

In many practical MPA algorithms, the variable node computations are implemented as additions of the variable node inputs and the check node computations as log or tanh function computation for each check node input and addition of the results of the log/tanh computations. 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 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.

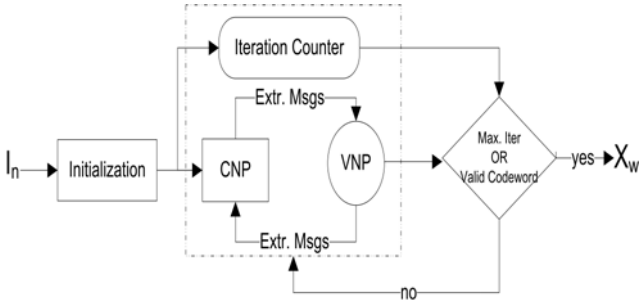

Fig. 3: Decoding flow diagram representing the main steps of MPA algorithm

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 [17] to a fully parallel [18], with large variety of partially parallel architectures [19]-[22] between them. 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 register transfer level (RTL) can be parallel), and macro-architecture level (where complex multi-accelerators can be build 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 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 macro-architecture/micro-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 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. As explained in Section 2, to perform the design space exploration a new adequate design methodology is necessary.

4. Quality-driven accelerator design methodology for demanding applications

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 Sections 2 and 3. This methodology accounts for the micro-architecture synthesis of basic accelerators, as well as, for the macro-architecture synthesis of the multi-accelerator structures. The new methodology considers the macro-architecture and micro-architecture synthesis, as well as, the computing, memory and communication structures' synthesis as one coherent complex task of the accelerator architecture synthesis, 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, as well as, for an effective tradeoff exploitation regarding the effectiveness and efficiency of the micro- and macro-architecture, and of their modules.

The methodology is quality-driven and model-based. It builds on the idea formulated by the first author of this paper [3] that ***system design*** is actually about a ***definition of the required quality***, in the sense of a satisfactory answer to the questions: what quality is required and how can it be achieved? Consequently, quality-driven design methods and tools are necessary to ensure that our systems 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 [3]. 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 [3]: ***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. Design space exploration 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 [3].

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 [3][4]. 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.

For the reasons discussed above, our accelerator design methodology adopts the ***quality-driven model-based design exploration and architecture synthesis approach*** proposed by the first author of this paper [3], and exploits the concept of generic architecture template. 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. The behavioral, structural and parametric requirements impose limitations on the structure of a required accelerator solution, but they do it in different ways. Structural requirements define the acceptable or preferred accelerator 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. Behavioral requirements also define the structures indirectly, by requiring the structures to expose a certain externally observable behavior that realizes the required behavior.

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 trade-off 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 selected abstract generic architecture templates are adequately instantiated and used to design space exploration that aims at analysis of various architectural choices and macro-/micro-architecture tradeoffs, and finally, at the construction of one or several most promising accelerator architectures supporting the required behavior and satisfying the demanded constraints and objectives. During the design space exploration, the system of computations represented by the application behavior model has to be appropriately distributed over the structure of modules of each proposed instance of the generic architecture template and scheduled, to define an actual system architecture that is required to satisfy the constraints and optimize the objectives of the quality model, in the context of specific trade-off preferences between the objectives. Since the accelerator architecture synthesis is a complex process involving joint micro- and macro-architecture synthesis of combined processing, memory and communication structures, an adequate design space exploration will usually require re-iterations and refinements. In result of the iterative design space exploration, one or more satisfactory application-specific architectures are constructed, and after their further analysis, and possible further refinement and optimization, one of them is selected to become the actual accelerator architecture. This way the quality-driven template-based design space exploration helps to arrive at adequate combinations of architectural choices for the design of a high quality accelerator for a particular set of application requirements. According to our knowledge, the so formulated accelerator design problem and its proposed above solution concept are not yet explored in any of the previous works

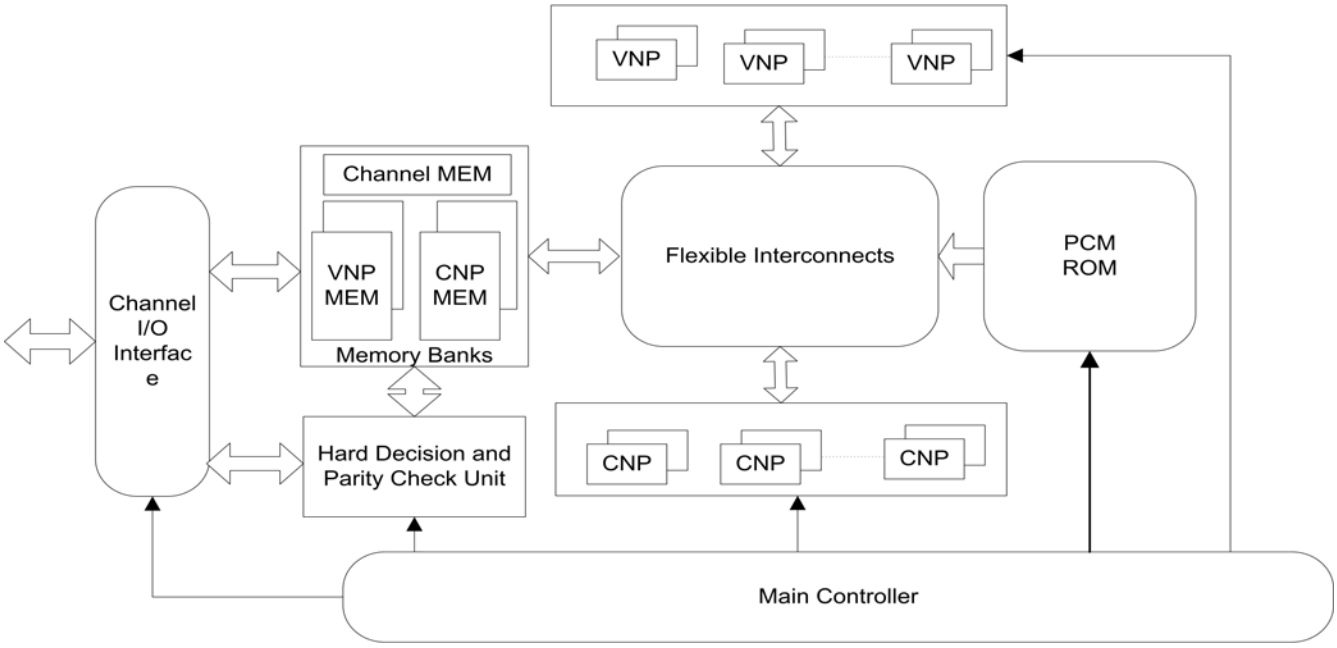

Fig. 4: Example of a generic architecture template for LDPC decoding accelerators

related to hardware accelerator design.

In more precise terms, our ***quality-driven model-based accelerator 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 applications' 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 modules' 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 design space exploration 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. The

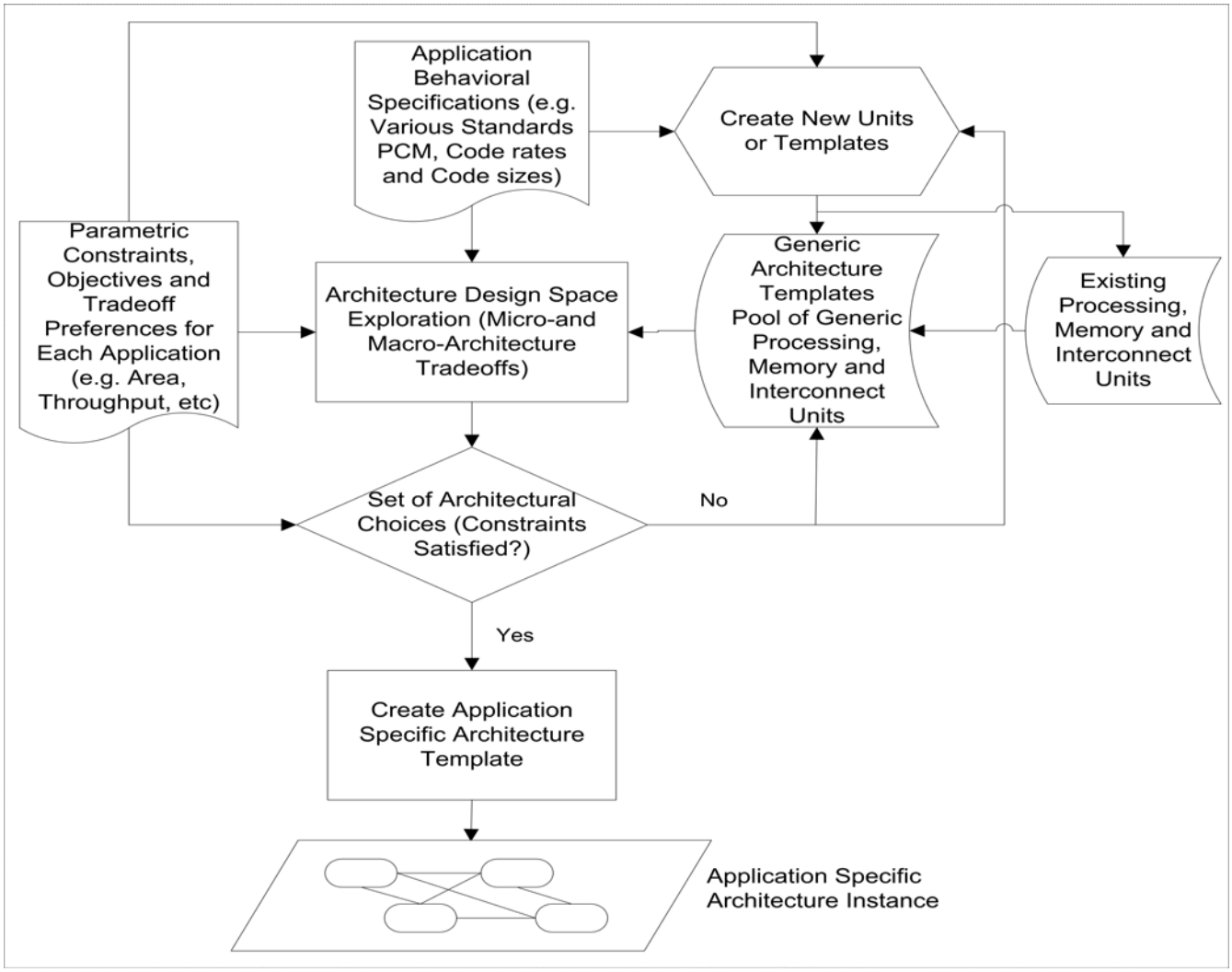

Fig. 5: Architecture exploration framework of the proposed accelerator design methodology

significant analysis and design effort required to design the architecture templates is however compensated due to enabling an effective and efficient design space exploration when using the templates. Fig. 4 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 elementary VNP and CNP processors and memories, configurable interconnects between the 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. 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 Fig. 5).

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 Fig. 5). 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 satisfactory 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 trade-off 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 in this way constructed architecture is subsequently examined and analyzed to check to what degree the constraints, objectives and preferences are satisfied, and 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 trade-off 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 trade-off preferences. 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 this way 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 Fig. 5). 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 Fig. 5).

During the design space exploration two major aspects of the accelerator design are considered and decided concurrently: its macro-architecture and micro-architecture. At the same time, the tradeoffs between these two aspects in relation to the design quality metrics (such as throughput, area, energy consumed, cost etc.) are analyzed and decided. 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 other important parameters, but they do it in different way 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 design space exploration several different promising combinations of the micro- and macro-architectures are constructed and analysed.

To decide the most suitable architecture, the promising architectures constructed during the design space exploration are analyzed and characterized in relation to various metrics of interest (such as throughput, area, energy consumption, cost) 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 architectures. 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 design space exploration 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.

5. Application to LDPC accelerator design

Currently, we are applying the accelerator design methodology discussed above to the design of demanding hardware accelerators for LDPC code decoders for some of the newest demanding communication system standards. In this process, we use generic architecture templates similar to this shown in Fig. 5. The aim of this 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 this way 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 representing the accelerator's required behavior, as well as, by a set of parametric requirements related a. o. to the error correcting performance, throughput, and accelerator area. The design parameters that can be influenced and decided during the design space exploration 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 [15] or Omega [16] network).

Our first impression regarding the usefulness of the design methodology proposed above to the accelerator design for the LDPC decoding is positive. The actual implementation of the methodology is not as difficult as it can appear when considering the high problem complexity, and it results in an affective and efficient design space exploration process which enables us to construct and analyze several promising LDPC decoding accelerator architectures in a short time.

5. Conclusion

This paper presented the results of our analysis of the main problems that have to be solved in design of accelerators for modern demanding applications, demonstrated that the today's high-level synthesis is not sufficient to adequately support the design process of complex hardware accelerators, formulated the main requirements that have to be satisfied by an adequate methodology of accelerator design for demanding applications, and proposed a quality-driven model-based accelerator design methodology which satisfies the requirements. Currently, we are applying the methodology to the design of hardware accelerators for LDPC code decoders for some of the newest demanding communication system standards. Our future research will involve application of this methodology to design accelerators for several modern applications.

9. References

[1] L. Jóźwiak, A. Douglas: Hardware Synthesis for Reconfigurable Pipelined Accelerators, Proc. Of ITNG'2008 – IEEE International Conference on Information Technology: Mew Generations, Las Vegas, NV, USA, April 7-9, 2008, IEEE Computer Society Press, Los Alamitos, CA, USA, pp. 1123-1130.

[2] Y. Jan and L. Jóźwiak: CABAC Accelerator Architectures for Video Compression in Future Multimedia: A Survey, Proc. of SAMOS'2009 – 9th International Workshop on Embedded Computer Systems: Architectures, Modeling, and Simulation, Samos, Greece, July 20-23, 2009, LNCS 5657, Springer-Verlag, Berlin Heidelberg, Germany, pp. 24-35.

[3] L. Jóźwiak: Quality-driven Design in the System-on-a-Chip Era: Why and How? Journal of Systems Architecture, Elsevier Science, Amsterdam, The Netherlands, 2001, Vol. 47/3-4, pp. 201-224.

[4] L. Jóźwiak, N. Nedjah and M. Figueroa: Modern Development Methods and Tools for Embedded Reconfigurable Systems – a Survey, Integration – The VLSI Journal, Elsevier Science, Volume 43, No 1, January 2010, pp. 1-33.

[5] R. Schreiber at al: High-level synthesis of nonprogrammable hardware accelerators, Proc. of ASAP'2000, pp. 113–124.

[6] K. Kuchcinski, C. Wolinski: Global approach to assignment and scheduling of complex behaviours based on HCDG and constraint programming, Journal of Systems Architecture, Vol. 49, 2003, pp. 489–503.

[7] S. Gupta, N. Dutt, R. Gupta, A. Nicolau, SPARK: A high-level synthesis framework for applying parallelizing compiler transformations, Proc. Int. Conf. on VLSI Design, 2003, pp. 461–466.

[8] Z. Guo, B. Buyukkurt, W. Najjar, and K. Vissers: Optimized generation of data-path from C codes for FPGAs, Proc. DATE'05, 2005, pp. 112–117.

[9] M. Puschel at al: SPIRAL: Code generation for DSP transforms, Proceedings of the IEEE, Vol. 93, No. 2, 2005, pp. 232–275.

[10] S. Sun, W. Wirthlin, M. J. Neuendorffer: FPGA Pipeline Synthesis Design Exploration Using Module Selection and Resource Sharing, IEEE Trans. on CAD, Vol. 26, No.2, 2007, pp. 254–265.
[11] S,P Mohanty, N. Ranganathan, E. Kougianos, P. Patra, P.: Low-Power High-Level Synthesis for Nanoscale CMOS Circuits, Springer, 2008, pp. 1–298.
[12] Synfora PICO platform for accelerator synthesis from C, http://www.synfora.com/.
[13] R. Tanner: A recursive approach to low complexity codes, IEEE Trans. on Inf. Theory, 27(5), 1981, pp. 533-547.
[14] D.J.C. MacKay: Good error-correcting codes based on very sparse matrices, IEEE Trans. on Inf. Theory, 45(2), 1999, pp. 399-431.
[15] G. Malema and M. Liebelt: Interconnection Network for Structured Low-Density Parity-Check Decoders, Proc. 2005 Asia-Pacific Conf. on Communications, 2005, pp. 537-540.
[16] M.M. Mansour and N.R. Shanbhag: High-throughput LDPC decoders. IEEE Trans. on VLSI System, 11(6), 2003, pp. 976-996.
[17] E. Yeo, P. Pakzad, B. Nikolic and V. Anantharam: VLSI Architectures for Iterative Decoders in Magnetic Recording Channels, IEEE Trans. on Magnetics, 37, 2001, pp. 748-755.
[18] A. Darabiha, A.C. Carusone and F.R. Kschischang: Multi-Gbit/sec low density parity check decoders with reduced interconnect complexity, Proc. ISCAS'2005, 2005, pp. 5194-5197.
[19] K. Gunnam, G. Choi, W. Wang and M. Yeary: Multi-Rate Layered Decoder Architecture for Block LDPC Codes of the IEEE 802.11n Wireless Standard, Proc. ISCAS'2007, 2007, pp. 1645-1648.
[20] K. Sangmin, G.E. Sobelman and H. Lee: Flexible LDPC decoder architecture for high-throughput applications, Proc. APCCAS'2008, 2008, pp. 45-48.
[21] L. Zhang, L. Gui, Y. Xu and W. Zhang: Configurable Multi-Rate Decoder Architecture for QC-LDPC Codes Based Broadband Broadcasting System, IEEE Trans. on Broadcasting, 54(2), 2008, pp. 226-235.
[22] Z. Cui, Z. Wang and Y. Liu: High-Throughput Layered LDPC Decoding Architecture, IEEE Trans. on VLSI Systems, 17(4), 2009, pp. 582-587.