1

APPLICATION-DRIVEN SYNTHESIS METHODOLOGIES FOR REAL-TIME PROCESSOR ARCHITECTURES

Francky Catthoor[1], Lars Svensson[1]
Klaus Wölcken[2]

[1] *IMEC, Leuven*
[2] *Commission of the E.C., Brussels*

ABSTRACT

In this chapter, we present an overview of the objectives and key achievements of the work described further in the book. Our topic is the synthesis of application-specific architectures and their realizations, targeted to microelectronic products involving real-time processing. Engineers who design real-time systems experience two classes of difficulties. One is methodical: there is no good design strategy to bring a concept to an effective realization. The other is practical: it is hard to produce a cost-effective, customized architecture for a given throughput or latency. In the course of the last four years, the contributing authors of this book have cooperatively addressed these bottlenecks. We propose a set of solutions that form part of a complete methodology for the target domain of dedicated processors for real-time systems. All the work has been driven by real-life case studies. To demonstrate our methods, a number of these realistic examples are treated in the subsequent chapters. The work has been performed in the context of two highly successful ESPRIT basic research actions: ASCIS ("Architecture Synthesis for Complex Integrated Systems") and NANA ("Novel parallel Algorithms for New real-time Architectures").

1 PROBLEM DESCRIPTION

State-of-the-art real-time signal and data processing applications typically involve a rapidly increasing arithmetic complexity. In many cases, this is combined with a need for flexible and powerful decision-making. Furthermore, they not only involve word-oriented processing but also employ advanced vector and matrix operations [1]. Loops are abundant and sometimes non-parallelizable, which leads to computational bottlenecks. This large application domain ranges from audio and speech processing, user-end (portable) telecommunication, and automotive applications, all of which exhibit low to medium throughputs, up to HDTV, image, video, and radar processing, which require much higher sample rates. In addition to such "hard" real-time applications, other data-intensive tasks that require fast execution also increase in importance. Some examples are real-time graphics and medical and other types of data acquisition.

All of these applications have in common that they are difficult to realize on general-purpose hardware [7], when an efficient solution—in terms of power or in terms of area—is required. Such realizations result in considerable overhead in terms of memory requirement, I/O bandwidth, I/O interfaces, and controller cost. Furthermore, there is typically a waste of cycles on the general-purpose rigid data-paths. Similar arguments apply largely for MIMD assemblies of general-purpose processors, even for the current generations of processors oriented towards parallel processing, such as Transputers or the TMS320C40 series. In some cases, the design time for these implementation alternatives can be small. For complex algorithms, though, the mapping stage may require many iterations and time-consuming manual tuning, just like when custom hardware is used. The major advantage is obviously the flexibility to make last-minute changes. This is especially interesting during algorithm development and prototyping, when the algorithm is subject to continuous refinement. However, when the system has reached the production stage and especially in the more mature product generations, we believe special-purpose or application-specific system realizations can be heavily motivated. In particular, this is the case if the application requires that the implementation exhibits low power consumption or a small physical size, or if the production volume is reasonably high, as is the case for the important markets of portable communication equipment and consumer electronics products like HDTV and videophone.

It has to be stressed that the term *application-specific* in this context implies few assumptions about the target implementation technology. The main requirement is that the technology permits the adaptation of the global architecture, in terms of primitive building blocks like adders, register files, and boolean operators, to the specific needs of the application. Hence, the complete range from full-custom design over macro-cell and/or standard-cell based approaches, sea-of-gates technologies, and field-programmable devices all the way to off-the-shelf building blocks can be the target.

2 STATE-OF-THE-ART AND BEYOND

The development cost of a complex application-specific solution can be very high, especially due to the typically long design cycle. Production cost will depend largely on the target technology chosen: small volumes will motivate the use of FPGAs, whereas large volumes favor the use of predefined macro-cells, and intermediate situations may require intermediate solutions. Therefore, much more effort should be spent on reducing the design-time-related cost. Moreover, as a consequence of this long design time, the time-to-market can be unacceptable. This is of major importance in the rapidly changing markets of today. In almost all cases, the long design cycle is heavily dominated by the time-consuming iterations due either to changes in the specifications which affect many design stages, or to late detection of incompatibilities between subcomponent designs.

State-of-the-art industrial CAD tools typically support only the lower end of the design path, i.e., structural synthesis from a structural specification down to the actual implementation [13]. In addition, the register-transfer and functional (combinatorial) levels in the design trajectory have become increasingly mature in the past five years, both in academia and independent research labs (e.g., IBM, U.C. Berkeley, University of Colorado at Boulder, Stanford, and INP Grenoble) and recently also in industry (e.g., Synopsys, Cadence, Mentor, Racal-Redac, Siemens, and Philips). Support is mainly restricted to the register-transfer-level scheduling and allocation of scalar signals to individual registers, from simple operations to basic RT-level operators and from individual transfers to multiplexers and connections.

The dominant part of the design trajectory above register-transfer level is still largely unsupported. One of the reasons for this is the huge amount of dedicated architectural options available to the designer [1, 32, 40, 23, 6]. Moreover, for automatic synthesis on the higher abstraction levels to be a viable alternative to hand design, especially in the consumer or domestic markets, a sufficiently large design efficiency has to be achieved. The results should approach those of manual designs, in terms of throughput, physical size, power consumption, and packaging. Therefore, currently emerging solutions are largely restricted to specification and co-simulation environments based on standards like VHDL or other widely distributed hardware description languages like the Verilog language. Usually, the description level applied is still very close to the RT-level structure. Moreover, *all* design decisions must be taken manually, even those that are less critical for meeting the design constraints, very error-prone, or very time-consuming.

In order to really solve the design-time bottleneck for real-time system design, this is not sufficient. There is a clear need for *higher level design support,* including *synthesis tools at the architectural or behavioral (system) level* which include *power- and area-efficient automated techniques.* The tools should still be fully controllable by the designer, so *user interaction* is another key issue. We believe that in order to achieve this ambitious goal, we need [9]:

- A range of efficient *target architecture styles* architecture style underlying the synthesis strategies. This is the only way to guarantee a sufficient coverage of the design space without losing efficiency. Dedicated regular arrays and application-specific multiplexed processors are two key target styles that are addressed in this book.
- Domain-specific *synthesis tools* which fully exploit the characteristics of the target architecture style and the corresponding application domain. The tools will frequently differ for the alternative target styles. For instance, a scheduler that is efficient for one style may be unsuitable or inefficient for another.

This has not been widely realized until about 1991, even if some of the issues had been addressed earlier. A detailed comparison with these approaches is provided below in sections 5 and 6 and in the subsequent chapters.

3 CONTRIBUTION OF THIS BOOK

The goal of the work on architecture synthesis within the ASCIS and NANA projects has been to contribute design methodologies and synthesis techniques which address the design trajectory from *real behavior* down to the RT-level *structural specification* of the system. Our view of this synthesis process is illustrated in figure 1.

The work described in this book specifically addresses the following topics, which we believe will have a substantial impact on future application-specific design methodologies:

- Refined system specification models that have embedded in them the practical requirements of architecture synthesis and verification methods. This is opposed to the simulation-oriented semantics of most of the widely used specification formalisms, such as VHDL; still, support for a VHDL subset has been investigated.

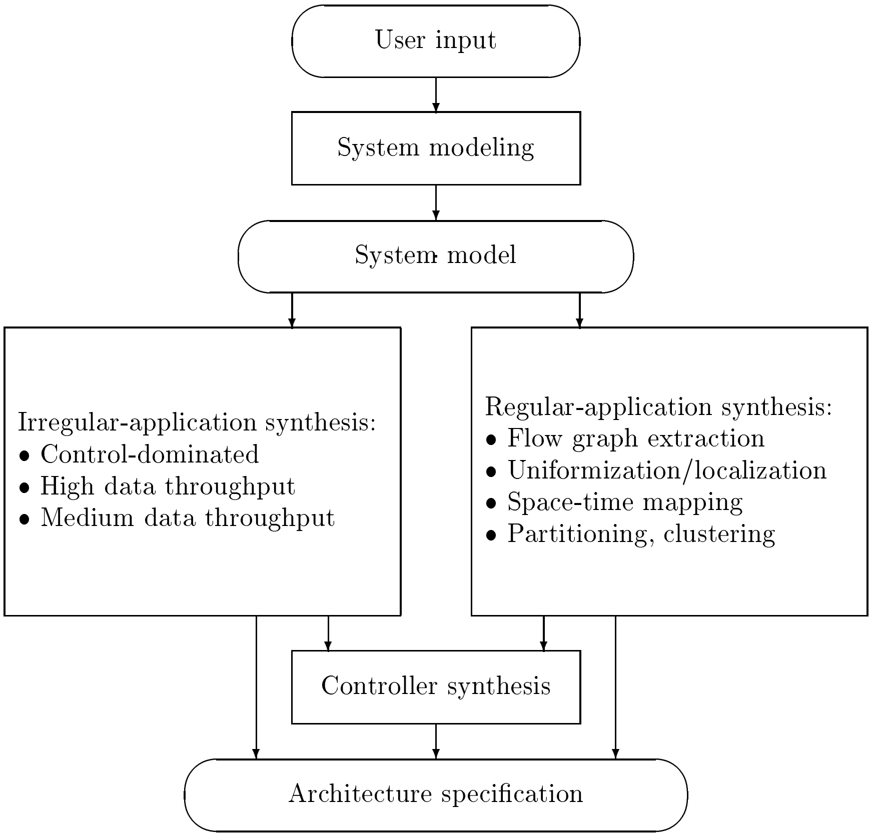

Figure 1 Architecture synthesis as viewed within the ASCIS and NANA projects. The input from the user is converted to a formal system model which can also be written out in a readable specification format. Different design trajectories must be followed, depending on the characteristics of the application. The figure only shows the two main branches we have addressed. The final high-level performance-driven controller synthesis is similar for both branches. In the next design stages, which have not been addressed by the projects, the detailed synthesis of the data-paths and controllers on the RT and logical levels still has to take place. This is then followed by the physical design stage.

- Novel design methodologies and synthesis techniques for regular array processor architectures intended for regular, high-throughput data-flow applications.
- Novel design methodologies and synthesis techniques for multiplexed processor architectures intended for irregular high- and medium-throughput applications. Both control-flow- and data-flow-dominated target domains are treated.
- Performance-driven controller synthesis which makes high throughput possible by overcoming part of the control bottleneck.

In order to provide complete support for the high-level synthesis trajectory, many design problems must be addressed. We do not claim that the material presented in this book covers the complete path, and certainly not all target application domains or architecture styles. However, we do believe we have contributed to the solution of a number of the most crucial problems in the domains of architecture synthesis and the related issue of behavioral specification.

Moreover, the research effort of the ASCIS and NANA partners has been planned with real applications in mind. Real-life drivers have been selected to steer the development of the required methodologies and techniques. Some of the main drivers will also be used as real-life demonstrator applications in the following chapters.

The extent of each contribution by the partners and how they fit in a more global design context will be summarized in somewhat more detail in the subsequent sections. Most of this work is also reflected in the subsequent chapters. The structure of the book and the links between the other chapters will be described in section 7.

4 SYSTEM SPECIFICATION MODEL

Any synthesis method has to start from an initial specification, described in terms of a model which has to meet a number of requirements. In particular, there is clear need for:

- A specification model that explicitly contains the information necessary to extract the desired functional behavior *and* the timing requirements in our real-time application domain.
- The means to establish relations between the different elements or kernels of the model while synthesis progresses. For example, flow-graph nodes (operations) eventually have to be associated with structure nodes (operators).

A global reduction of design time is only possible if the aspect of verification is addressed. Leaving errors in a design until production is, of course, a disaster: it will cause large extra costs to locate the error, correct it, and restart the production process. Avoiding these design iterations by simulation alone has the well-known problems of an incomplete check, a prohibitively large computational load, and a difficult interpretation of the huge amount of simulation responses. For these reasons, *formal verification* has been addressed by several research institutes. Here, the goal is to formally prove that the system will behave well in all circumstances.

To make formal verification possible, there are again several prerequisites on the specification model:

- The model has to consist of formally defined elements. This allows formal reasoning about its behavior. The models commonly used in simulation programs are not strong enough for this.
- A partial or full description of the desired behavior of the system must be available, again expressed using mathematical formalisms. This mathematical specification should be extracted from a more user-friendly input, so that the designer does not have to learn complex formalisms.

Most of the current design models are not suitable for the real-time signal processing domain and do not deal with the above requirements. At the start of the ASCIS project, there was a clear need to fill this gap. This has led to the ASCIS data flow-graph (DFG) model described in chapter 2. It differs from most other data-flow approaches by a uniform embedding of conditional and loop constructs, allowing, for instance, the specification of conditional or repeated I/O and giving maximal opportunity for system optimizations.

A number of other design models have been proposed recently: the Sil model in the context of the Sprite (Esprit 2260) project, involving Philips and its partners [21]; the DSFG/LIB models in the Cathedral project at IMEC [25]; and the RLEXT design model from the University of Illinois [20] come to mind. However, the RLEXT model addresses mostly RT-level issues and not the true behavioral synthesis models. The other models mentioned have been established in close cooperation with the Ascis work.

The resulting refined DFG format has been adopted as a common specification format by all the Ascis and Nana partners. Moreover, initiatives have been taken to come to a standardization in this domain together with several other European partners.

In addition, effort has been spent on formal verification itself, especially regarding timing issues for control-flow-dominated systems. A study of propositional temporal logic has shown its feasibility to verify correctness of sequential logic circuits, CMOS transistor circuits, and finite state machine diagrams against each other [16]. This work has been received with great interest in the worldwide research community, since verifications were done that previously seemed untractable. It has been applied in the context of performance-driven controller synthesis, as described in chapter 10.

Verification has also been performed directly on the DFG [17]. This is a highly interesting perspective, since data flow graphs are widely used as the starting point for synthesis. The verification here is a more global data-independent scope, allowing the verification of much larger systems at the cost of a reduced resolution: detailed, data-dependent functional behavior is not modeled.

It is also important to provide a conversion from existing specification languages to the DFG model. For this purpose, a Vhdl modeling scheme has been developed [27]. This allows to represent the DFG as a subset of Vhdl and vice versa. Also, conversions from other languages that cover important application domains have been investigated, namely HardwareC [22] and Silage [15]. More information is provided in chapter 2.

5 SYNTHESIS OF ARRAY PROCESSORS

As the complexity of what can be integrated on a single die or in a multichip module is increasing tremendously, it becomes more and more attractive to implement special-purpose architectures using highly modular parallel algorithms [36]. In that case, the modularity inherent in the algorithms should be reflected in a regular architecture with one- or two-dimensional (1-D or 2-D) repetitivity in the arrangement of processing elements, local communication, and distributed local storage. Examples of such regular organizations are systolic or wavefront arrays, which are typically defined as regular networks of locally interconnected elementary processors [23]. In order to avoid a long critical path spanning several processors, intensive pipelining is used between the processors, leading to the term "systolic." Reasons to consider this regular array style include:

- *High performance* due to the systematic use of parallelism and pipelining.
- *Decreased design time* obtained by implementing only a few elementary cells which are then replicated many times.
- *Design modularity,* i.e., the possibility of using the same set of cells for the various values of some size parameter of the algorithms.
- *Simplification of test and fault-tolerance issues*, as the complexity of the analysis is broken by the regularity of the design.
- *Potentially lower power consumption.* Regular parallel architectures permit trading off clock speed and parallel computation in a much more flexible way than irregular ones. This issue is important when power consumption is to be minimized.

An important driver which has influenced several of the subtasks in the array synthesis methodologies in this book is the solution of the algebraic path problem (APP) which is occurring frequently in mathematical equation solvers and in data analysis. This driver will be discussed in detail in chapter 3. Several crucial subtasks will be identified, and the results for several architectures optimized for the APP problem will be described. For many of these subtasks, a link will be made with the synthesis techniques and tools necessary to address these issues.

Research on formal methods for the design of array architectures has been going on for about 10 years [35, 29]. Briefly, these methods consist of transforming an iterative specification of the desired algorithm into the specification of an architecture, by mapping the initial iteration space onto a new "space-time" space, composed of the processor number space and the execution time. The majority of the approaches has concentrated on pure systolic arrays [2, 8, 43, 11, 23, 29, 36, 37], as is also evident from the references of chapters 6 and 4. Moreover, most of the techniques reported until now are based on either heuristic transformations applied in a user-defined sequence, or on linear or affine transformations of the index space and dependence vectors of an iterative algorithm that is already uniformized. The heuristic transformations of the first class are executed in an unformalized, ad-hoc way. This leads to an architecture mapping which is very sensitive to user input. Most of the affine transformations are only dealing with part of the mapping trajectory, namely starting from a level usually referred to as uniform recurrence equations (UREs) [18].

In addition, most array architecture design methods deal either with the design of non-real-time systems or with programmable arrays composed of rigid PEs, mostly composed with off-the-shelf components. However, the area of fully customized designs for real-time signal and data processing subsystems is important for industry. We believe this is not addressed in a sufficiently effective way by the existing techniques, as indicated in chapters 6 and 4.

Hence, we have felt the need for several extensions to the design trajectory:

- The initial specification should be at a higher level than the conventional level of uniform recurrence equations (UREs). A "front end" is needed to build the dependence graph from a more user-friendly specification of the algorithm. Several specification languages have been proposed here. Procedural models with FORTRAN-style DO loops are used in chapters 4 and 5 for data processing applications, involving complex data updates in the algorithm that are more difficult to express without the concept of "variables." In contrast, functional languages like ALPHA [3] and SILAGE [15] are introduced in chapter 6 for more regular signal processing applications that may be elegantly described in single-assignment form. For all these languages, a conversion is needed to some type of dependence graph. For the functional languages this is relatively simple, but for the procedural languages, novel techniques have been developed to achieve this goal.

- The gap between high-level behavioral descriptions and the URE level has to be bridged with formalized high-level transformations on the signal flow graphs. These involve mainly reindexing to arrive at a set of affine recurrence equations in a single domain [44] and localization for dealing with broadcast and global operations [43]. At this stage, the possibility of interactive guidance is also of crucial importance [3]. This topic will be addressed in more detail in chapter 6.

- The properties of real-time signal and data processing applications have to be exploited to arrive at fully efficient application-specific architectures. This requires extensions to the basic linear space-time mapping methods [44] as proposed in chapters 3 and 6. In addition, an alternative affine space-time transformation method based on the existence of independent subsets in the index space is introduced in chapter 5. This method has led to efficient arrays with high hardware utilization [24].

- Both word-level and bit-level parallelism should be exploited to reach the extreme throughputs which are needed in particular applications like radar processing [33]. Extensions in this direction are introduced in chapter 5. Employing the features of various arithmetic systems, such as 2's complement and residue number systems (RNS) [42], bit-level systolic arrays can now be derived automatically.

- After space-time mapping, the conditional execution of several operations on the same processing element (PE) may be needed. Regularization techniques to address this problem are presented in chapter 4.

- In order to match the required throughput or the required array size, it is necessary to partition the initial architecture with full index ranges into a set of serially executed subsets and then to cluster these onto a smaller sized array. This important task has been addressed in depth [5] and is described in chapter 4. Extensions which can result in a better preservation of the regularity, especially for latency-limited applications, are proposed in chapter 3.

The comparison of our approaches with some other techniques, which do address particular issues that we feel to be vital for this domain, will be presented in more detail in the subsequent chapters.

Some of the applications studied during the development of the array processor design methods will be described along with the methods themselves. The applications range from so-called "kernel algorithms" (matrix computation, filters in one and two dimensions, dynamic programming, singular value decomposition, etc.) to real-life applications such as image analysis, video coding, and computer graphics algorithms.

6 SYNTHESIS OF MULTIPLEXED PROCESSORS

A second important architecture style for real-time signal processing systems is the multiplexed processor style, characterized by a set of application-specific, time-multiplexed data-paths steered by a hierarchically organized controller. This style is tuned to irregular applications, requiring a medium to high sample rate (10 kHz – 10 MHz). Many applications at these rates require a combination of computation-intensive arithmetic and complex decision-making operations. For these applications, the array style (as described in section 5) is unsuitable.

1 Control-flow-dominated processors

The multiplexed processor style spans a large complexity range, both for the data-paths and for the controller. An algorithm with a small amount of data flow with mainly operations on scalar data, but with a complex mix of nested loops, data-dependent iterations (including global exceptions and unconditional loop exits), and nested conditions, is characterized as control-flow-dominated. These algorithms abound in embedded control applications and in some telecommunication systems. They require a complex controller with a relatively simple but programmable data-path. Much effort has been spent on synthesis techniques for this important target application domain [49, 30]. However, several issues have not yet been fully addressed. In particular, support for a flexible building block (BB) library that not only contains standard BBs—such as ALUs—but also complex devices with as yet unknown clock cycle delays—such as cache memories or I/O processors—is necessary for a system to be useful for real-life designs. Efficient algorithms for allocation and scheduling tuned to these extensions are needed, too [31]. Also, interactive optimization capabilities should be stressed. Research to find a solution for these problems has led to the AMICAL environment [12] which is described in chapter 9.

2 Data-flow-dominated processors

In domains like automotive applications and back-end speech or audio processing, the data-flow issues are more dominant than the control flow. Then, there is a need for another target style with more emphasis on the data-paths and the storage. However, due to the medium sample rates, the data-paths will still need to be highly multiplexed and thus partly programmable in order to share hardware as efficiently as possible. Synthesis techniques for related target domains have been studied in the past [41, 38, 25]. However, in many cases there is a need to map such applications, which are not time-critical, onto pre-defined architectures for prototyping purposes. In this context, solutions based on field-programmable gate arrays (FPGAs) have become especially attractive lately. This style has been addressed using applications from the automotive field [47, 48]. The memory organization, the mapping onto the fixed data-path, and the connection network and scheduling have been the focuses here. The results on this highly multiplexed processor synthesis are described in chapter 8.

When mostly data-flow computations have to be performed combined with a limited control flow in a limited sample period, this calls for highly customized, pipelined, and more complex data-paths to solve the timing-related bottlenecks. This lowly multiplexed processor style is especially suitable for irregular image and video processing, user-end telecom, and front-end audio or speech processing subsystems, where the operations must be performed partly in parallel [6]. Dealing with these subsystems involves synthesis subtasks [25] that have not been addressed so far in existing synthesis approaches [26], such as how to map multidimensional signals in an effective way on distributed memory organizations. Most of the effort has gone into reorganizing the control flow for a given data flow [45]. In addition, the definition of the dedicated pipelined data-path organization has been addressed [14]. This has led to a complete lowly multiplexed processor synthesis approach, which is further discussed and compared to the state of the art in chapter 7.

In application domains like computer graphics, the iterative constructs for the data-flow-dominated algorithms are data-dependent. As a result, analysis and architecture synthesis techniques need to be developed for iterative constructs with potentially unbounded iterator ranges. In contrast to the situation in fixed-period signal processing where the *worst-case* response time has to meet the sample period criterion, now the *average* throughput is the important performance parameter. This problem has been addressed in part within ASCIS [4], but the results are not yet incorporated in this book.

The basic techniques to perform scheduling and assignment tasks are also important items. Although good heuristics exist for many specific problems, there is also a general demand for generic global optimization techniques to efficiently support the synthesis process. Simulated annealing [19] is a good example of such a technique, which has proven to be very useful. However, the complexity of the cost function that is evaluated for every potential step in the solution space and the resulting large computation time restrict the use of this technique in efficient interactive synthesis environments.

In recent years, new optimization methods have been published, promising general applicability with less computational complexity than simulated annealing. Two such new techniques have been investigated in the context of our projects:

- Genetic algorithms are search algorithms based on the mechanics of natural selection and natural genetics. Promising results in the area of scheduling for a given allocation have been reached [46].
- Optimization based on neural network solvers has been investigated, too. This technique has been applied with good results to the assignment problem as it occurs during lowly multiplexed processor synthesis [34].

Both of these techniques are characterized by their potential for massive parallelism. This aspect is very promising for their future use in the increasingly parallel computing environments.

3 Performance-driven controller synthesis

Designing high performance processors requires not only high performance data-paths but also fast and efficient controllers. An appropriate controller architecture is needed, as well as an efficient physical implementation.

Controller synthesis has been the subject of a very large amount of effort both in academia and industry (see section 2). However, until recently, the aspect of performance-driven manipulations has been largely neglected. Therefore, within the ASCIS context, effort has been devoted to filling this gap, even though this is not strictly within the high-level architectural synthesis domain as we define it. The results are described in chapter 10.

First of all, work has been performed on the optimization of the control unit architecture. One particular architectural feature that has been investigated is the subroutine mechanism. Subroutines enable the reuse of hardware resources; this could significantly reduce the area of the controller while also providing a shorter critical delay path. An algorithm has been developed that automatically transforms a controller specification to exploit subroutines in an efficient way [39]. Comparison to hand-made designs has shown that the algorithm in many cases produces results comparable to manual design.

In addition, effort has been spent on the optimizations related to technology mapping. In contrast to the majority of the approaches in literature until recently, the focus has been on performance-driven mapping onto compiled functional cells [28].

7 CHAPTER OVERVIEW

In order to provide a clear overview of the work described in this book and the way in which the other chapters are linked, they will be listed below with a short description of their content. The intention in the chapters of this book has *not* been to describe the novel synthesis techniques in detail. Many results have already been published in conferences and journals which deal with these techniques. The most important publications are listed in the bibliographies of the individual chapters. Instead, most of the emphasis will be placed on the high-level methodologies and on how these are inspired by the targeted application domains. In addition, most results are illustrated with a realistic demonstrator application. This will permit easy assessment of the power and the impact of the obtained results. It will also clarify the current status of the realization of the synthesis tools and environments.

Behavioral system modeling

- "Behavioral specification for synthesis"

  This chapter summarizes the basic model concepts that are used for the initial specification, but that are also highly tuned towards the requirements of the subsequent architecture synthesis steps. In addition, the link to behavioral specification languages used within the synthesis environments is made, with emphasis on a VHDL subset.

Synthesis methodologies for regular applications

- "Formal methods for solving the algebraic path problem"

  This chapter concentrates on one particular application and the methods needed to effectively tackle the architecture design. An historical overview of the evolution in the design methodologies and their effectiveness permits assessment of the significant progress made in this area. From this overview, the need for support of a number of synthesis tasks is emphasized. Many of the necessary techniques and tools are described in the other chapters.

- "HiFi: from parallel algorithm to fixed-size VLSI processor array"

  This chapter concentrates on methods suitable for extracting regular data-flow-dominated applications from a procedural specification, on methods for dealing with the conditional control flow resulting from mapping different operations on the same processor element, and on mapping techniques onto regular arrays with a fixed array size. Moreover, the necessary design models at different levels in the trajectory are discussed, and a uniform solution is described. A Floyd-Steinberg algorithm for use in document manipulation for a digital copier application is used as a demonstrator.

- "On the design of two-level pipelined processor arrays"

  This chapter concentrates on methods suitable for extracting the uniform data-flow for regular applications expressed with procedural DO loops. Synthesis techniques supporting the important extension to bit-level arrays are also discussed. This makes it possible to design arrays with a very high throughput, where all the algorithmic parallelism is exploited. A 1-D convolution algorithm which is a subsystem in many real-time processing applications is used as a demonstrator in this chapter.

- "Regular array synthesis for image and video applications"

  This chapter concentrates on space-time methods suitable for dealing with many broadcast localization alternatives and for mapping regular, data-flow-dominated applications onto regular arrays with complex processing elements. This is linked with a complete transformation environment that supports these array mapping techniques. Also, the need for extending the level of the initial specification to real behavior is motivated, including the requirements on additional synthesis techniques to fill the resulting gap in specification. A complete motion estimation subsystem from a local area network video coding application is used as a demonstrator.

Synthesis methodologies for irregular applications

- "Memory and data-path mapping for image and video applications"

  This chapter concentrates on methods suitable for mapping high-throughput, data-flow-dominated applications onto fully application-specific, lowly multiplexed processor architectures. The emphasis lies on the techniques for the memory management needed to deal with the many multidimensional signals, and techniques for the organization of the complex customized data-paths. An updating singular-value decomposition algorithm for data acquisition and a video format conversion application are used as demonstrators.

- "Automatic synthesis in mechatronic applications"

  This chapter concentrates on methods suitable for mapping medium-rate, data-flow-dominated applications onto programmable (FPGA-based) architectures for rapid prototyping purposes. The emphasis lies on the highly multiplexed target architecture and on important steps which need to be dealt with, such as memory organization, data and control flow transformations, and scheduling/binding. A differential heat release computation subsystem in an automotive application is used as a demonstrator.

- "Synthesis for control-flow-dominated machines"

  This chapter concentrates on methods suitable for control-flow-dominated applications. A flexible library of complex building blocks, including predefined I/O interfaces and memories, is targeted. Emphasis is placed on scheduling and allocation techniques tuned to this domain and on the interactive environment in which the tools are embedded. The demonstrator of this chapter is a complete telephone answering machine controller.

Synthesis methodologies for real-time controllers

- "Controller synthesis and verification"

  This chapter addresses the performance-driven issues in high-level controller design. In particular, effective ways to decompose the controller architecture are presented. The effect of compiled cells on performance-driven technology mapping is also discussed. Finally, it provides results on formal verification aspects in a real-time context.

8 CONCLUSION

This book provides an overview of state-of-the-art work in the domain of behavioral and architectural synthesis for real-time processing applications. The contributions are based on work carried out within the ASCIS and NANA research projects from mid-1989 to mid-1992. In this introductory synopsis, we have motivated that specific requirements to solve some crucial design bottlenecks are not met with existing design automation support. In particular, specific characteristics of real-time processing were not at all or not sufficiently exploited until now. In the subsequent chapters, several major contributions will be proposed to this important domain for future systems industrial needs, especially in the important markets of portable user-end telecommunication equipment, multimedia support, automotive processing, and consumer electronics such as HDTV, videophone, and digital audio broadcasting. We believe a transfer of these new technologies to development-oriented projects, including large systems companies, will help to shape the future systems and design automation industry in Europe and abroad.

To a large extent, the success of our work in this domain has been due to the excellent cooperation between the partners, both in the context of formal exchanges between the different tasks and by informal contacts at the frequent meetings and workshops. We believe that this cross-fertilization between the partners is the main added technical value of the ESPRIT type of basic research actions. The three years of cooperation has definitely led to an increase of our research productivity.

REFERENCES

[1] J. Allen. Computer architecture for digital signal processing. *Proc. of the IEEE*, 73, number 5, pages 854–873, May 1985.

[2] J. Annevelink and P. Dewilde. HiFi: A functional design system for VLSI processing arrays. *Proc. IEEE International Conf. on Systolic Arrays*, San Diego, pages 413–452, May 1988.

[3] A. Benaini, P. Quinton, Y. Robert, Y. Saouter, and B. Tourancheau. Synthesis of a new systolic architecture for the algebraic path problem. *Science of Computer Programming*, 1989.

[4] J. P. Brage. Hardware description languages for synthesis: problems and possibilities. In *Proc. of the tenth NORCHIP Seminar '92*, Helsinki, Finland, pages 22–29. Nov 1992.

[5] J. Bu and E. Deprettere. Processor clustering for the design of optimal fixed-size systolic arrays. *Algorithms and Parallel VLSI Architectures*, Vol. A, pages 341–362. North Holland, Elsevier, Amsterdam, 1991.

[6] F. Catthoor and H. De Man. Application-specific architectural methodologies for high-throughput digital signal and image processing. *IEEE Trans. on Acoustics, Speech and Signal Processing*, 37, number 2, pages 176–192, Feb 1990.

[7] B. Cole et al. The embedded processor breaks out of its niche. *Electronics*, pages 61–87. McGraw-Hill, Mar 1988.

[8] J.-M. Delosme and I. Ipsen. Efficient systolic arrays for the solution of Toeplitz systems. In W. Moore, A. McCabe, and R. Urquhart, editors, *Systolic arrays,* pages 37–46. Adam Hilger, Bristol, 1987.

[9] H. De Man, F. Catthoor, G. Goossens, J. van Meerbergen, J. Rabaey, and J. Huisken. Architecture-driven synthesis techniques for mapping digital signal processing algorithms into silicon. *Proc. of the IEEE,* 78, number 2, pages 58–78, Feb 1990.

[10] P. Dewilde and E. Deprettere. Architectural synthesis of large, nearly regular algorithms: design trajectory and environment. *Annales des télécommunications*, 46, number 1-2, pages 48–59, Jan–Feb 1991.

[11] J. Fortes and D. Moldovan. Parallelism detection and transformation techniques useful for VLSI algorithms. *Journal of Parallel and Distributed Computing*, 2, pages 277–301, 1985.

[12] A. Jerraya, I. Park, and K. O'Brien. AMICAL: an interactive high-level synthesis environment. *Proceedings of the European Design Automation Conf.*, Paris, France, Feb 1993.

[13] D. Gajski, editor. *Silicon Compilation.* Addison-Wesley, 1988.

[14] W. Geurts, F. Catthoor, and H. De Man. Time constrained allocation and assignment techniques for high throughput signal processing. *Proc. 29th ACM/IEEE Design Automation Conf.*, Anaheim, pages 124–127, Jun 1992.

[15] P. Hilfinger, J. Rabaey, D. Genin, C. Scheers, and H. De Man. DSP specification using the Silage language. *Proc. Int. Conf. on Acoustics, Speech and Signal Processing*, Albuquerque, NM, pages 1057–1060, Apr 1990.

[16] G. Janssen. Hardware verification using temporal logic: a practical view. In L. Claesen, editor, *Proc. IMEC-IFIP WG 10.2/10.5 International Workshop on Applied Formal Methods for Correct VLSI Design,* Houthalen, Belgium, pages 291–300. Elsevier, Amsterdam, Nov 1989.

[17] G. de Jong. Verification of data flow graphs using temporal logic. In L. Claesen, editor, *Proc. IMEC-IFIP WG 10.2/10.5 International Workshop on Applied Formal Methods for Correct VLSI Design,* Houthalen, Belgium, pages 301–310. Elsevier, Amsterdam, Nov 1989.

[18] R. Karp, R. Miller, and S. Winograd. The organization of computations for uniform recurrence equations. *Journal of the Association for Computing Machinery*, 14, number 3, pages 563–590, July 1967.

[19] S. Kirkpatrick, C. Gelatt Jr., and M. Vecchi. Optimization by Simulated Annealing. *Science*, No. 220, pages 671–680, 1983.

[20] D. Knapp and M. Winslett. A prescriptive formal model for data-path hardware. *IEEE Trans. on Computer-Aided Design*, CAD-11, number 2, pages 158–184, Feb 1992.

[21] T. Krol, J. van Meerbergen, C. Niessen, W. Smits, and J. Huisken. The SPRITE input language: an intermediate format for high-level synthesis. *Proc. European Conf. on Design Automation*, Brussels, Belgium, pages 193–199, March 1992.

[22] D. Ku and G. De Micheli. Synthesis of ASICs with Hercules and Hebe. In R. Camposano and W. Wolf, editors, *Trends in high-level synthesis.* Kluwer, Boston, 1991.

[23] S. Y. Kung. *VLSI Array Processors.* Prentice Hall, 1988.

[24] E. Kyriakis-Bitzaros and C. Goutis. An efficient decomposition technique for mapping nested loops with constant dependencies onto regular processor array processors. *Journal of Parallel and Distributed Computing*, 16, pages 258–264, 1992.

[25] D. Lanneer, S. Note, F. Depuydt, M. Pauwels, F. Catthoor, G. Goossens, and H. De Man. Architectural synthesis for medium and high throughput signal processing with the new CATHEDRAL environment. In R. Camposano and W. Wolf, editors, *Trends in high-level synthesis.* Kluwer, Boston, 1991.

[26] M. McFarland, A. Parker, and R. Camposano. The high-level synthesis of digital systems. *Proc. of the IEEE*, 78, number 2, pages 301–318, Feb 1990.

[27] J. Madsen and J. Brage. Flow graph modeling using VHDL bus resolution functions. *Proc. of the First European Conference on VHDL Methods,* IMT, Marseille, Sep 1990.

[28] J. Madsen. *Layout synthesis using compiled cells.* PhD Thesis, Department of Computer Science, Technical University of Denmark, 1992.

[29] D. Moldovan. Advis: a software package for the design of systolic arrays. *Proc. IEEE Int. Conf. on Computer Design*, Port Chester NY, pages 158–164, Oct 1984.

[30] P. Michel, U. Lauther, and P. Duzy, editors. *The synthesis approach to digital system design.* Kluwer, Boston, 1992.

[31] K. O'Brien, M. Rahmouni, and A. Jerraya. DLS: a scheduling algorithm for high-level synthesis in VHDL. *Proc. Europ. Design Automation Conf.*, Paris, France, Feb 1993.

[32] R. J. Offen, editor. *VLSI Image Processing.* McGraw-Hill, 1985.

[33] V. Paliouras, D. Soudris, and T. Stouraitis. Systematic derivation of the processing element of a systolic array based on residue number system. *Proc. IEEE Int. Symp. on Circuits and Systems*, San Diego, 1992.

[34] W. Philipsen and G. de Jong. Refinement of Petri nets: the neural net approach. *Proc. of the Int. Neural Network Conf.*, Paris, France, Vol. 1, pages 266–269. Kluwer Academic Publishers, Boston, Jul 1990.

[35] P. Quinton. Automatic synthesis of systolic arrays from recurrent uniform equations. *11th Int. Symp. Computer Architecture*, Ann Arbor, pages 208–214, Jun 1984.

[36] P. Quinton and Y. Robert, editors. *Algorithms and parallel VLSI architectures II.* Elsevier, Amsterdam, 1992.

[37] S. Rao and T. Kailath. Architecture design for regular iterative algorithms. In E. E. Swartzlander, editor, *Systolic Signal Processing Systems*, pages 209–297. Dekker Inc, New York, 1987.

[38] J. Rabaey, H. De Man, J. Vanhoof, G. Goossens, and F. Catthoor. CATHEDRAL II: a synthesis system for multi-processor DSP systems. In D. Gajski, editor, *Silicon Compilation*, pages 311–360. Addison-Wesley, 1988.

[39] K. Ranerup and L. Philipson. Control architecture selection from state graph characteristics. *Proc. ASCIS Open Workshop on Controller Synthesis*, Technical University of Denmark, Lyngby, Denmark, Sep 1991.

[40] P. Ruetz, R. Jain and R. Brodersen. Comparison of parallel architectures for image processing systems. *Proc. IEEE Int. Symp. on Circuits and Systems*, San Jose, pages 732–737, April 1986.

[41] C. B. Shung, R. Jain, K. Rimey, E. Wang, M. Srivastava, B. Richards, E. Lettang, K. Azim, L. Thon, P. Hilfinger, J. Rabaey, and R. Brodersen. An integrated CAD system for algorithm-specific IC design. *IEEE Trans. on Computer-Aided Design*, CAD-10, number 4, pages 447–463, Apr 1991.

[42] F. Taylor. Residue arithmetic: a tutorial with examples. *IEEE Computer Magazine*, pages 50–62, May 1984.

[43] V. van Dongen and P. Quinton. Uniformization of linear recurrence equations: a step towards the automatic synthesis of systolic arrays. *IEEE International Conf. on Systolic Arrays*, San Diego, pages 473–482, May 1988.

[44] M. van Swaaij, J. Rosseel, F. Catthoor, and H. De Man. Synthesis of ASIC regular arrays for real-time image processing systems. *Journal of VLSI signal processing*, 3, pages 183–192, Kluwer, Boston, Sep 1991.

[45] M. van Swaaij, F. Franssen, F. Catthoor, and H. De Man. High-level modeling of data and control flow for signal processing systems. In M. Bayoumi, editor, *Design methodologies for VLSI DSP architectures and applications.* Kluwer, Boston, 1992.

[46] N. Wehn, M. Held, and M. Glesner. A novel scheduling/allocation approach for data-path synthesis based on genetic paradigms. *Proc. IFIP Working Conference on Logic and Architecture Synthesis*, Paris, France, May 1990.

[47] N. Wehn, H. Herpel, T. Hollstein, P. Pöchmüller, and M. Glesner. High-level synthesis in a rapid-prototype environment for mechatronic systems. *Proc. of EURO-DAC'92*, Hamburg, pages 188–193, Sep 1992.

[48] P. Windirsch, H. Herpel, A. Laudenbach, and M. Glesner. Application-specific microelectronics for mechatronic systems. *Proc. of EURO-DAC'92*, Hamburg, pages 194–199, Sep 1992.

[49] W. Wolf, A. Takach, and T-C. Lee. Architectural optimization methods for control-dominated machines. In R. Camposano and W. Wolf, editors, *Trends in high-level synthesis.* Kluwer, Boston, 1991.