CHAPTER 1

Introduction

The application domain for digital signal processing encompasses many areas such as speech, image and radar processing and a number of telecommunication applications. Recent advances have had an enormous impact on everyday life. Consumer electronics products, from cellular phones to anti-lock brake controllers in automobiles all rely on built-in signal processing arrangements to perform these functions. In the earlier times the products were so simple that their design principles were well understood. Recently, digital signal processors and application-specific integrated circuits have brought very large and complex embedded systems into existence. It has been noticed that the traditional methods in system design fail to efficiently support the high performance hardware technology available. Moreover, the number of different design options is reasonably large due to numerous elementary components. According to the often-cited Moore’s Law, the amount of information storable on a given amount of silicon doubles every 12 to 18 months. As a consequence, the issues of complexity and flexibility are having an enormous effect on system developers and end users. Practices and tools in embedded system design are changing fast and it is difficult to master their progress.

The design of various products is subject to stringent non-functional requirements, including
performance, power dissipation, reliability and reusability. At the same time, strong economical requirements exist. Time-to-market challenges for designers require that increasingly complex systems of hardware and software are designed within shorter time spans. For many consumer electronics applications the lifetime of a product can be as short as nine months, especially in the area of telecommunication. As a consequence, the length of the product design cycle must be reduced to months. Because of the nature of the applications of consumer electronics, it is necessary to cope with specifications that can change continuously under stringent time-to-market constraints. High production volumes go along with demand for safety and quality and lesser maintenance needs. Physical constraints, such as size and weight are also extremely critical. These aspects necessitate a flexible design paradigm that supports specification changes at late stages of the design cycle.

The traditional methods for embedded system development consist of separate design threads for hardware and software. A specification is developed and sent to isolated hardware and software design teams. Hardware-software partitions are decided in advance and adhered to as much as possible. Last minute changes in specifications cannot be accommodated because changes necessitate expensive redesign. This approach is rather slow and error-prone and may fail to meet the requirements today. Clearly, fully parallel hardware and software design threads facilitate faster design and fewer errors, resulting in significantly shorter product design cycles. Communication and interaction of team members from
diverse design cultures is extremely important, because it initiates invaluable interaction between the design threads.

Simultaneous design of hardware and software is not a new idea: systems with significant portions of hardware and software have been produced for decades. Within a time span of two decades, the level of abstraction in hardware design has evolved from transistor level to gate level and more recently, from register-transfer level to behavioral level [1]. To facilitate system design even further, the appropriate level of abstraction is the system level [2]. This rise in the level of abstraction calls for new computer aided design tools and methodologies. Comprehensive tool support should provide a uniform framework supporting all stages of product development, based on interactive user guidance.

Algorithmic developments as well as architecture and process technology advances make it possible to have products ranging from cellular phones supporting WAP, to sophisticated electronic games and video on demand in the market. The rapid entry of these products is shaping the course of Electronic Design Automation (EDA) technology, as the most advanced fabrication techniques and EDA methodologies converge on consumer electronics design. To ensure that these products meet performance goals, minimize costs and avoid failures, the designers increasingly turn to high-speed emulation [3] and prototyping technologies [4], [5].
The ever-increasing arithmetic complexity and data rates however drive the realization of these systems to application specific integrated circuits (ASICs). At the same time product life cycles are shortening and hence time-to-market turns into a key issue, triggering a strong demand for efficient and design cycle spanning automating tools.

1.1 Problems faced by co-design

There are many traditional barriers to effective co-design and co-verification: organizational structures, which can impede communication across disciplines, outdated paradigms and differing technical jargon. Hardware suppliers, software suppliers and EDA vendors often lack an integrated view of the design flow, links between flows and adequate modeling support. In general, there are major gaps in the links to implementation, causing less than reliable quality results and limiting the flexibility of design teams. Especially needed are tools that can better estimate the constraints between the boundaries, before iterating through a difficult flow.

In hardware design, top-down constraints come from product design requirements and bottom-up constraints come from "physical data" in libraries and estimation tools. Simulation and analysis tools allow us to find the conflicts between these two kinds of constraints, such that either some aspects of design implementation are changed or the product requirements are modified, if implementation constraints cannot be satisfied. Software exists at a higher level of
abstraction than hardware; however, its only physical limitation is a function of the hardware on which it executes. Thus, bottom-up constraints for software can only be realized in a hardware context. Co-verification permits the top-down and bottom-up constraints to interact, revealing design errors not readily observable from one perspective alone.

Co-verifying hardware that is still being designed implies some degree of modeling strategy. The modeling of physical reality is a messy business and is imperfect. Because of the inherent complexity, modeling strategies in hardware-software co-design tend to be domain-specific, implementation specific and application-specific. To make matters more complicated, heterogeneous verification requires modeling interactions at the correct abstraction level of interest. Substantially different abstraction levels may be needed depending on the design requirements.

Numerous techniques are employed to verify combined hardware-software systems, but each has its own set of limitations. The result is often a highly fragmented set of design methodologies with no "universal" solution. Attempting to run application code on top of a software processor model that is itself running on top of software (the simulator) is rarely practical. Yet, running the code on the native hardware processor may exclude access to internal registers and control over the processor state, which can be critical to the software development process. It is also possible to run code on models of hardware emulated through dedicated programmable hardware,
offering near real-time speed for code execution. Unfortunately, some designs require real-time interaction with other hardware and external environments and thus cannot use any co-verification strategy that does not support full speed code execution.

The questions soon outnumber the answers e.g.

- What co-design and co-verification strategy will be required?
- Which modeling techniques must be utilized?
- When should a particular technique be used?
- Can they be mixed?
- What modeling tradeoffs should be accepted?
- Which EDA tools and libraries can support these needs? How will this affect our overall design methodology and impact time-to-market requirements?

1.2 Opportunities in co-design

Hardware-software co-design accelerates the design process [6], [7]. It is hardly a new idea. For decades, design teams have had to face the realities of combining digital computing with software algorithms in designing the complex systems. The problem has been that until recently, verifying the interaction between the two has required the building up of actual prototype hardware. While this has always been a painful exercise, at best, it has generally sufficed—until now.

The system design trends of the '90s have intensified the problem further and also opened up new
potential solutions. Semiconductor processes now permit entire systems to fit on a single chip. Most of the processing of "real world" data, audio, video, sensors; has migrated to the digital realm. The "system on a chip" (SoC) phenomenon is now adding to the momentum, as co-design turns from merely a good idea into an economic necessity. And the prediction for the future point to greater embedded software content in hardware systems than ever before.

Two major opportunities to capitalize on in the hardware-software co-design area are:

1. The top-down system-level co-design and co-synthesis work that has been under way at the universities and research centers for the last few years.
2. The major advances being made by the EDA companies in the area of high speed emulation systems to form the basis for co-verification strategies.

1.3 Major trends in hardware-software co-design

Co-design focuses on the areas of system specification [8], architectural design [9], hardware-software partitioning [10] and the iteration between the hardware and software [7] as the design progresses. Co-design is complimented by the co-verification area that focuses on the hardware-software integration and test.

In the early days, the initial partitioning of the system hardware and software was done by the system engineers. Code development and hardware
development were performed sequentially, with little cross coupling until actual prototype hardware was built. Today the selected portions of the design and development flow are coupled, requiring multiple, fragmented modeling efforts to support specific co-design and co-verification goals throughout the design process. In the near future, several factors would evolve that may converge the designer’s environment for both co-design and co-verification.

First, the workstation processing speeds continue to increase at an impressive rate. While much can be attributed to IC manufacturing process capability that enables shrinking design rules and higher edge rates. Other factors such as advanced CPU architectures, operating systems and compilers continue to enable software to run faster on simulated hardware. The use of hardware emulation has made it possible to increase the software execution capability from one to two instructions per second (IPS) on a hardware simulator to greater than 100K IPS on a hardware emulator. However, the corollary is that designing those very same systems to deliver this capability also requires more co-design and co-verification than before to achieve design goals.

Design re-use has been a popular buzzword for many years, but until recently remained an elusive concept. Now that SoC densities and processes are becoming a reality, previous and current generation ICs are finding their way into new designs as embedded cores in a mix-and-match fashion. This will force greater convergence of methodologies for co-design and co-verification, since both data-flow and control-flow
dominant hardware must reside on a single chip. This higher priority should lead to a new breed of tools that help integrate co-design and co-verification choices.

In mid nineties, the U.S. government initiated a program to develop new co-design methodologies and automation tools in support of the embedded design of signal processors. The rapid prototyping of application-specific signal processors (RASSP) program combines the concepts of hardware-software co-design and design reuse with other novel approaches that promise to raise the state of the art across a number of design applications.

The European Commission has funded the Open Microprocessor Initiative (OMI). OMI has a set of long term objectives to reduce the design time for an embedded micro-controller system. This commission has just recently taken the decision to open participation in OMI to non-EC companies.

In future, both hardware and software designers will increasingly need tools for estimating the impact of design changes earlier in the design cycle, before committing to specific hardware-software partitioning.

1.4 Design specifications and methodologies

Applying the right modeling strategy at the right time is the key to catching elusive design errors quickly. Yet it is often necessary to consider multiple approaches as the design project progresses, rather than just one.
Now the question arises as to how can multiple modeling approaches fit into an already tight design cycle? The answer is highly dependent on the specific goal and the constraints of one's design project, as well as computational domain. In general, the design flows are fairly complex. The process of co-design and co-verification are also iterative, with multiple passes required to validate increasingly refined levels of detail. The design flows may look very different depending on whether one is designing an ATM switching system, a cellular phone, computer system, aircraft auto-pilot, automotive cruise control, video focusing unit, graphics compression chip, or a custom RISC processor and so on.

The main area of concentration for hardware-software co-design is in the area of embedded systems. An embedded system's functionality is usually fixed and is primarily determined by the system's interaction with its environment. Embedded systems usually have numerous modes of operation, they must respond rapidly to exceptions and possess a great deal of concurrency. The component subsystems commonly found in embedded systems include some or all of the following:

- software,
- firmware,
- Application Specific Integrated Circuits (ASICs),
- general-purpose and/or domain-specific processors,
- memory,
- core-based ASICs,
- application-specific multiprocessors,
- Field Programmable Gate Arrays (FPGAs) and
- analog circuits.
Thus, designing a complex embedded system poses a difficult problem to designers.

Today's embedded-system designer does not have comprehensive assistance in performing system design tasks. No universally accepted methodology or tool is available to help the designer to create a functional specification and map it to a system level architecture. Most system designers (with the possible exception of the DSP area) work in an ad-hoc manner, relying heavily on informal and manual techniques and exploring only a handful of possibilities. In the last few years a number of university programs have been focusing on developing a hierarchical modeling methodology to improve the situation. An example of this methodology [8], [9] that has been extensively researched at the University of California at Irvine is briefly outlined here. In such a methodology, the system's functionality is initially precisely specified, numerous system-level implementations are explored with the aid of tools. A refined description is automatically generated, which represents any implementation decisions.

More specifically, the following tasks are necessary to create a system level design in the above methodology.

1. Specification capture: To specify the desired system functionality, the functionality is decomposed into pieces by creating a conceptual model of the system. A description of the model is translated in a language. The description is validated by simulation or verification techniques.
The result of specification capture is a functional specification, which lacks any implementation detail.

Specification capture is a difficult problem because the designer may not know a system's functionality precisely at the outset and specifications are often imprecise. To make the problem worse, the specification capture stage does not receive as much attention as subsequent implementation stages and thus many functional errors are not detected until a low-level implementation is available. Unfortunately, functional errors are far more difficult to correct at later stages of product development than during the specification stage.

2. Exploration: Numerous design alternatives are explored to find one that best satisfies the constraints. To do this, the initial description is transformed into one or more suitable implementations. A set of system components are allocated and their physical as well as performance constraints are specified. The functional specification is partitioned amongst allocated components. The quality of each design alternative is estimated in these exploration sub-problems.

3. Specification: The initial specification is refined into a new description that reflects the decisions that were made during the exploration. A system description is generated detailing the system's processors, memories and buses. Co-simulation is used to verify that this refined description is equivalent to the initial
specification. The result of specification refinement is a system-level description that possesses some implementation details of the system-level architecture that has been developed, but is largely functional.

4. Software and Hardware: An implementation is created for each of the components, using software and hardware design techniques. A standard processor component requires software synthesis, which determines a software execution order satisfying resource and performance constraints. An ASIC design can be obtained through high-level (behavioral) synthesis, which converts the behavioral description into a structure of components from a register-transfer (RT) library containing micro-architectural components such as ALUs, registers, counters, register files and memories. The control logic and some RT components are synthesized with finite-state-machines and logic synthesis techniques. The result of software and hardware design is an RT-level description, which contains optimized code for software and RT-level net-lists for custom components.

5. Physical design: This step generates manufacturing data for each component. For software, this is as simple as compiling code into an instruction set sequence. For hardware an RT-level net-list is converted into layout data for gate arrays, FPGAs, or custom ASICs, using tools for physical placement, routing and timing.

These five tasks roughly define embedded-system design methodology from product conceptualization to
manufacturing. After each task, a more refined system description is generated, reflecting the decisions made in that task. The hierarchical modeling methodology enables high productivity by preserving consistency through all levels and thus avoiding unnecessary iteration. Each model verifies different system properties. The functional specification verifies the completeness and correctness of system functions. The system-level description verifies system performance and communication protocols. The RT-level description verifies the developed software code and the custom design's operation during each clock cycle. The physical description verifies the system's detailed timing and electrical characteristics. Hierarchical modeling distinguishes modern, system-level methodologies from past methodologies, which captured only the physical model late in the design cycle, making specification or architecture changes nearly impossible.

1.5 Model Creation

To specify a system's functionality, it must first be decomposed and the relationships between various parts described. In general, a model is a formalization of allowable parts and their relationships.

There are many models for describing a system's functionality. One is the data-flow graph [11], [12], [13], [14], [15], [16], which decomposes functionality into activities that transform data (such as a piece of a program) and the data-flow between the activities. Another model is the finite-state machine
(FSM) [17], [18], which represents the system as a set of states and a set of arcs that indicate transition of the system from one state to another when certain events occur. Extensions of this model include hierarchy and concurrency. A third model, communicating sequential processes (CSP) [19], decomposes the system into a set of concurrently executing processes, each of which executes a sequence of program instructions including variable assignments, loops, branches and procedure calls. A fourth model, the program-state machine (PSM) [20], [21], combines the previous two models by permitting each state of a hierarchical/concurrent FSM to contain actions described by means of program instructions. Other models include Petri nets [22], flowcharts, entity-relationship diagrams, Jackson diagrams, control-data-flow graphs, queuing models and object-oriented models [23], [24]. Object-oriented models are very good for maintaining a consistent set of models, due to their inheritance and polymorphism features. This capability makes it possible to introduce models in a progressive manner with simple models becoming available very early and more elaborate models developed, as they are needed.

No model is ideal for all classes of systems. For example, the data-flow model may be most natural for a system that repeats the same data transformation on streams of data, such as a digital signal-processing system. The FSM model may be most appropriate for a system that does not perform complex computations but must respond to complex sequence of external events, such as a control-dominated system. The CSP model is most appropriate for systems that perform complex data
transformations, possibly in parallel e.g. software applications. The PSM model in many ways comprises of the FSM and CSP models, so it is appropriate not only for control-dominated systems but also for data-dominated systems such as software applications.

However, the best model is the one that most closely matches the characteristics of the system it models. For embedded hardware-software systems the characteristics include hierarchy, concurrency, state transitions, exceptions and program instructions.

1.6 Description Generation

The choice of a model is the most important influence on the ability to understand and define system functionality during system specification. Once the appropriate model has been chosen, the system functionality must be captured in a functional specification, using one of many different languages [2]. A functional specification is easy to generate if there is a one-to-one correspondence between model characteristics and language constructs. If a language construct does not exist for a particular characteristic then a set of constructs needs to be improvised that describes that characteristic.

There are several languages that designers commonly use to specify functionality. VHDL and Verilog are popular standards that support easy description of a CSP model through their process and sequential-statement constructs. They are also commonly used to describe FSMs, although neither language possesses explicit constructs directly
supporting state transitions. Esterel [25], [26] is similar to those languages, adding constructs to support exceptions. Statecharts supports description of hierarchical and concurrent FSMs, including exceptions. SpecCharts supports capture of the CSP model, hierarchical/concurrent FSMs and the PSM model. SDL, a CCITT (International Consultative Committee for Telegraph and Telephone) standard, supports descriptions of hierarchical data-flows diagrams with an FSM at the leaf level. Finally, Silage supports easy description of data-flow models through its data stream and recurrence constructs.

1.7 Software Synthesis

A system-level description usually possesses complex features not found in traditional programming languages such as C. A typical compiler usually cannot compile these features. Software synthesis is the task of converting a complex description into a traditional software program compilable by traditional compilers.

1.8 Hardware-Software Co-design Present in Universities

There are a number of very good research programs underway at various universities addressing different aspects of the area of hardware-software co-design. As pointed out earlier, there is not one universal solution to the hardware-software problem. Initially each university group generally focused on a particular aspect of the problem or application area. The earlier projects focused more in the digital
signal processing area utilizing the data-path approach, while the more recent projects are expanding into the reactive control area and also combinations of both data-path and reactive control along with the appropriate communication protocols. Some examples of the various University projects are given below.

The Ptolemy Project [27], [28] at the University of California at Berkeley (UCB) choose the digital signal processing application area and focus on the data-path approach with a heterogeneous design methodology in mind. The Cathedral Project at IMEC [29], also, chooses digital signal processing with a data-path approach, but focus on a vertical design methodology with an emphasis on synthesis. This project, which was one of the earliest DSP project, has since been commercialized by Mentor in their DSP Station Product. The Pope Project at IMEC, renamed as CoWare [30] has the aim to address both reactive control and real-time data-path systems in the telecommunication area. The GRAPE [31 .. 38] project at the Katholic University at Leuven is, also, in the DSP area and is a system level development environment for the specification, compilation, debugging and emulation of digital signal processing applications. Another project at UCB has focused on the automotive application area with the emphasis on reactive real-time control utilizing FSM-based modeling with the aim of developing a formal methodology for specification, modeling, automatic synthesis and verification of these types of systems.

The University of California at Irvine has built upon their work in high level synthesis and specification capture around SpecCharts to build a
specify-explore-refine methodology [8], [9] based on a hierarchy of models at different levels of abstraction. Stanford has concentrated on the co-synthesis of both hardware and software and the methods to describe constraints. Carnegie Mellon University (CMU) has developed a methodology that focuses on co-simulation and co-synthesis. Princeton and the University of Washington both have projects in the area of software synthesis for interface protocols.

It is noted that with the exception of the CMU work, the majority of the university projects do not address the automatic partitioning area and are at the very early stages of addressing synthesis at the system level, while most projects do utilize synthesis at the behavioral level.

1.9 Hardware-Software Co-design - Vendors

The area of the Electronic Design Automation (EDA) Industry that has begun to focus on the system design problem has become known as the Electronic System Design Automation (ESDA) segment of the EDA Industry. The ESDA segment first appeared in 1994 and really started to deliver products in the 1995-96 time frame.

The tools provided by vendors in the Electronic System Design Automation (ESDA) can possibly be classified into four major areas: domain-specific, application-specific, enabling technologies and bridging technologies.
Domain-specific offerings build on top of a suite of enabling technologies to target a specific design discipline. The main domain covered by products today is in the area of digital signal processing. The DSP Station from Mentor is a commercialization of the Cathedral Project from IMEC and is certainly the most vertically integrated from the behavioral level and downwards. The SP Worksystem from the Alta group of Cadence, also, focused on digital signal processing with many specialized verification and analysis features.

Application-specific platforms include not only point and domain-specific tools, but also customized product-design environments for a finite set of applications such as for wire-less communications, networking and multimedia applications. Unlike the domain-specific offerings, which are a necessary foundation, application-specific offerings focus on the entire system. The application-specific area is still in its infancy and would be a good area to examine in a Hardware-Software Co-design study.

Vendors of enabling technology supply specialized point tools that are generally useful in many applications. An ESDA design solution requires multiple point tools and a customized product-design environment in order to be effective. Examples of such enabling technologies include graphical entry tools such as Statemate from i-Logix and Visual HDL from Summit Design, system-level simulators such as Workbench from SES and TD Technologies and behavioral synthesis tools from Synopsys and Synthesia.
Bridging technologies between ESDA and traditional EDA includes both design and verification links to downstream activities. On the design side examples include single and multiprocessor code generators for target processors; specialized, highly efficient, application-specific module generators; HDL generation for logic-level synthesis.

On the verification side, some vendors provide specialized capabilities for mixed-level verification. This includes co-simulation of system-level modules with HDL-level modules. The Eaglei product from Eagle Design Automation is aimed at co-verification. The emulation products from both Quickturn and Zycad offer at least three to four orders of magnitude speed improvement over hardware simulation programs for verifying software.

In general the vendors offer a number of point tools to address different aspects of the problem. The main domain-specific area is digital signal processing with the emphasis on data-path models. The application specific area is still to be addressed. The main area of hierarchy is at the architectural level and below. The area of co-verification is starting to be addressed in a point manner with new products.

1.10 Development environment

The high-tech consumer products pose the most demanding challenge for the designers due to intense competition in the market and the enormous volumes of production coupled with short product development times. The economies of scale offered by system-on-chip technologies have thus been irresistible, but the
designs are difficult to verify. Despite requirements for careful design and full verification of each function, development times are as tight as ever. In short one must maximize performance, optimize feature sets, minimize cost, exhaustively verify functionality and still minimize time to market. Hence, EDA tools are necessary to meet the goals for high performance, feature set, time to market and exhaustive verification. High-speed emulation and prototyping systems are the answer to these requirements.

1.10.1 Real-time emulation

The typical design cycle for an ASIC starts with a requirements phase, consisting of both functional and non-functional requirements (power consumption, physical properties, cost, environmental characteristics). The functional requirements are translated into an algorithmic design. This design has to be verified for compliance to the specifications. Simulation on a host computer is common practice at this stage. This technique has however several drawbacks, that worsen with increasing system complexity. First, algorithmic complexity may lead to very long simulation times (from a few days to some weeks) to process only a very small time frame. The examples of systems taking enormously long times are described in [39]. Such long simulation times prohibit full testing of the algorithm. Playing back the results of the simulation in real-time may result in only a few seconds of real-life signals (audio or video sequences), which hampers testing subjective qualities such as sound and image clarity. Secondly, if the algorithm has user-controlled parameters, these
have to be tested by changing them at run-time and observing the effects. Again, in the small time frame this is not possible. Another problem arises from the environmental characteristics. Typically, in simulation a model is created of the signal characteristics and behavior of the environment. For new technologies, such models may however not be available or the behavior may be too complex to model it accurately [33].

These problems can be solved if a hardware prototype is available in the early design stages. If the hardware prototype is flexible and capable of performing the algorithm in real-time, then it can be placed immediately into the final environment, its results can be examined at the final speed and effects of parameter changing can be observed. Early customer feedback can be obtained and incorporated. Furthermore such a prototype can serve to convince management and non-technical people of the benefits and feasibility of a project. When the prototype is used for functional verification and parameter tuning, it is said to provide a functional emulation. In later design stages, a bit-true emulation may be performed, in which finite word length effects are introduced. Effects of truncation, rounding etc. can be studied, to determine the minimal word length satisfying the requirements and guaranteeing stability, as this reduces ASIC costs. When the ASIC architecture is developed, an architecture-true emulation and a timing emulation can further unveil problems specific to these design phases.
1.11 Organisation of this work

It is clear that in order to meet real-time challenges co-design methodologies have to be adopted. Moreover, an emulation platform should be able to support rapid-prototyping of a system populated with heterogeneous processors.

The goal of this work is to find the suitability of a design flow for mapping the stream oriented DSP applications, with data rates known at compile-time, onto targets consisting of multiple general purpose DSPs and FPGAs/ASICs based heterogeneous systems. It is assumed at the beginning that this work would result in a completely different flow than for DSP targets due to the different properties of DSPs and FPGAs/ASICs. Also, the heterogeneous DSP-FPGA systems need to restrict the data memory requirements to an absolute minimum due to highly constrained memory resources on FPGAs. Hence, memory minimisation algorithms are developed to determine the minimal amount of data memory for which a deadlock free schedule exists.

Chapter 2 deals with various co-design methodologies, frameworks and tools. Also, the need for software acceleration and the ways to achieve this are presented. Here the role of special programming languages as well as HDL is discussed. It is derived that a formal design approach should be capable of describing system behaviour at varying levels of abstraction.

In chapter 3 various data flow graphs used for representing flow of data or processing, in a rapid prototyping environment are discussed. Programming
models for DSP and FPGA targets are discussed and the suitability of a programming model, for DSP targets, for populating FPGAs is ascertained. It has been found that a target device independent environment can accommodate FPGAs without any detrimental effect on performance of the system. Thus the same flow could be used for DSPs as well as for FPGAs and heterogeneous DSP-FPGA systems could be supported easily. For FPGAs, it rapidly becomes clear that scheduling an application without restricting the data memory requirements to the absolute minimum would lead to designs that could not be implemented, due to their highly constrained memory resources.

In chapter 4, data memory minimisation algorithms for elementary chains to determine the minimal amount of data memory for which a deadlock free schedule exists, are developed. These algorithms form the basis for memory minimisation in graphs represented by chains and cycles.

In chapter 5, data memory minimisation algorithms for cycles with one source node and one sink node and without initial tokens are developed. These algorithms eliminate the drawbacks of the minimisation procedures used for chains.

In chapter 6, data memory minimisation algorithms for cycles with one source node and one sink node and with initial tokens are developed. As a special case a two-node cycle is dealt with.

Thus the co-design platforms, normally restricted to contain only DSPs, become suitable for ASIC-DSP based heterogeneous systems. These systems provide an edge in rapid-prototyping scenarios and can meet short time-to-market challenges in a better way.

25