FACTA UNIVERSITATIS (NIŠ)
SER.: ELEC. ENERG. vol. 16, April 2003, xx-xx

A New Approach for System-level Architecture Exploration

Vladimir Živković, Pieter van der Wolf, Ed Deprettere and Erwin de Kock

Abstract: In this paper, we present a new design-space exploration approach which we call the symbolic program approach. The symbolic program approach is based on both the trace driven approach and the control data flow graph approach. As expected, the trajectory of the symbolic program approach appears somewhere in-between the two extremes mentioned above. Thus, it leads to the shorter simulation time while it can still give fairly accurate performance numbers. Moreover, it produces results that can be readily taken as input for further design.

Keywords: System-level, design-space exploration, traces, CDFG, symbolic programs.

1 Introduction

Designers of single-chip signal-processing systems experience two conflicting trends: The transistor count of SoCs is exponentially growing - allowing ever increasing complexity of applications - and the design time and cost are forced to decrease because of highly competitive consumer application markets. A promising trend in SoC design is to move away from detailed cycle accurate abstraction levels, at least during the initial design stages, and to focus on system-level platform-based exploration and design methodologies

Manuscript received October 2002. An earlier version of this paper was presented at the 37rd International Conference on Information, Communication and Energy Systems and Technologies, ICEST 2002, October 1-4, 2002, Niš, Serbia.

V. Živković and E. Deprettere are with Leiden Institute of Advanced Computer Science, Leiden University, Leiden, the Netherlands (e-mail: lale@liacs.nl). P. van der Wolf and E. de Kock are with the IST/ESAS group. Philips Research Laboratories, Eindhoven, the Neterlands.

that rely on libraries of IP components. Thus, System Level Methodologies and Tools (SLMT) must open the way to early and effective exploration of large design spaces, as opposed to Cycle Accurate Level approaches where a walk through the design space is tedious and costly. Such explorations enable quick evaluation of alternative designs and help to gain confidence for particular design choices early on with relatively small investments.

1.1 Related Work

In SPADE [11] applications and architectures are modelled in such a way that mapping of applications onto architectures, prior to architecture/application co-simulation, can be done easily. The co-simulation is trace-driven (TD): The execution of the application generates traces of symbolic instructions that are interpreted by the architecture that reveals timing behaviour. Since the architecture does not process the actual data, the TD co-simulation is fast. Several case studies [14] have shown that this approach is quite successful.

A similar approach was presented by Lahiri [7]. Their approach gives fast performance analysis of bus-based SoC communication architectures. Starting from application graphs of symbolic instructions, and architecture specifications, mapping of applications onto architectures is done by deriving a non-executable graph-like representation without control information. This system model representation is compact and the scheduling of graph-nodes (augmented symbolic instructions) during the system simulation is relatively simple.

In the COSY (COdesign Simulation and SYnthesis) approach [5], designers can, starting from a functional specification, do communication refinement and design-space exploration using performance models, and efficiently get to an optimal implementation. Furthermore, they can effectively exchange IPs because of a clear separation of functionality and architecture, as well as the separation of IP behaviour and communication.

In the Task Concurrency Management (TCM) approach [2] the objective is to provide a systematic exploration of design spaces at the higher levels of abstraction and to provide better schedules of the system tasks. TCM is targeting more towards system-level synthesis than system-level exploration. However, the TCM reasoning about a compact representation that contains "the most relevant" system information can be useful for design-space exploration. Particularly, in the TCM approach the system-model is specified in

a combined Multi Thread Graph - Control Data Flow Graph (MTG-CDFG) model [10]. This model gives sufficient information for concurrency extraction and code transformations, which result in better system performances.

2 Problem Statement

SLMTs are not yet mature enough and they can not achieve sufficiently high levels of confidence to be applied in design- specific cases. Present inaccuracies originate from abstractions of crucial system behaviours (e.g., using abstract *read* and *write* constructs instead of the more refined communication and synchronisation primitives [12]); from insufficient support of a designer's mapping strategies (e.g., poor modelling of intra-task concurrency [2]); and from sometimes severe restrictions in the number of Models of Computation (MoC) that are supported (e.g, non-deterministic behaviour that is lacking [3], [10]). One of the challenges in system-level Design Space Exploration (DSE) is to achieve high levels of confidence in the performance numbers that are collected during quantitative performance analysis.

Improvement of the accuracy is the first but not the only requirement that system-level methodologies must satisfy. It is also necessary that system-level methodologies connect well with methodologies used for detailed design (the "state-of-the-art" shows that this connectivity is not always satisfactory in SLMTs). Moreover, SLMTs should allow incremental refinement of the high-level architecture and mapping models. The incremental and "uninterrupted" flow from SLMTs to detailed design is an additional challenge in system-level design and DSE.

2.1 Defining the Scope

We are interested in platform-based system-level design for streaming applications. Due to the previous, architectures we want to explore are heterogeneous, and, therefore, hard to program and/or evaluate [13]. We find that, in order to deal with these issues efficiently, we need to enable the *separation of concerns*, i.e., separation of application modelling, architecture modelling, and mapping [1]. We also find that we need a MoC that matches target media architectures very well. Such model is the Kahn Process Network (KPN) MoC [4]. The KPN MoC is based on concurrent processes (being sequential inside) that communicate in a First-In-First-Out (FIFO) manner. The matching is the more so when the KPN model is enhanced with non-determinism modelling capabilities as available in the YAPI tool [3]

which provides a *select* operation that explicitly models the influence of the scheduling on the application execution.

There are two major directions that can be followed starting from this point: (1) Exploration of architectures and mappings driven by application models, where the aim is to obtain performance numbers starting from the specification of the application models, or (2) system synthesis driven by application models, where the aim is to generate the source code of an implementation from the application specification. In this paper we follow the first direction, i.e., system-level DSE. However, we aim to do it in such a way that the exploration trajectory connects well to the trajectory for system synthesis.

3 Mapping Approaches

A crucial step in DSE is the mapping of application models onto architecture models in order to evaluate the performance of different application-architecture combinations. However, the mapping as a single-step procedure is hardly possible but rather should consist of a sequence of mapping steps, because application and architecture models never match ideally. The order in which these steps are taken is important, since the impact on performance improvements or cost reductions may be depending on it. As a result, application-to-architecture mapping is a concept that can be translated into more than one mapping approach.

A key issue while distinguishing among the exploration based mapping approaches is which representations of the system are used during the mapping steps. That is, which aspects of the system are captured in a description that is transformed in the mapping process.

We have identified and compared two extreme mapping approaches that can be taken in an exploration context. We call these approaches the Trace Driven (TD) approach ([11], [7]) and the Control Data Flow Graph (CDFG) approach (based on [16], [2]). We have also identified third *hybrid* approach, which we call the Symbolic Program (SP) approach. The SP approach is positioned between these two extremes. The main differences are in terms of functionality (executability) of representations and existence of control inside of representations. Table 1 shows more compactly which characteristics are (not) supported in which mapping representations (and, implicitly, approaches). Note that there is the fourth representation we have shown in Table 1, too. This is the Data-Flow Graph (DFG) representation [8].

However, DFGs are out of the scope of this paper.

In the next three subsections, we present the three approaches, as well as a comparison among them.

Table 1. Approaches mapped on the executability-and-control matrix.

Executable→ / *Control*↓	*YES*	*NO*
YES	CDFG code	Symbolic Program
NO	DFG code	Symb.Instr.Trace

3.1 TD Mapping Approach

In the TD approach, an application model is executed on a single data set. Behaviours in the processes of the application model can be abstracted by generating for every channel-read, every channel-write, and every function in the process a symbolic instruction *read*, *write*, and *execute*, respectively. These symbolic instructions are sent to an *application trace* in the order in which they are generated. These traces drive the architecture components in a non-functional architecture model in which the symbolic instructions are interpreted in terms of *performance numbers*, such as latency, throughput, resource utilisation, bus contention, etc. The TD approach is visualised in Figure 1 (left part). A trace generator executes the YAPI model using a particular data set and generates execution traces. For each process a trace is produced. A mapping layer takes the application traces as its input and produces transformed traces, the *architecture traces*, that are driving the components of the non-functional architecture model. If more than one trace drives a single component (many-to-one mapping), then run-time scheduling may be applied in this component. Assignments are otherwise static.

The TD approach is suited for simulation of deterministic applications. However, if a non-deterministic application is to be mapped onto an architecture, and an application-architecture co-simulation is to be performed, then the trace driven simulation is more complex. The reason is that the application traces may now depend on the actual schedules used during architecture simulations. Additionally, a serious problem with the TD approach is that the refinement of the application trace in the mapping layer is severely restricted because the application trace does not contain symbolic instructions for control expressions, such as branching, selection, and

repetition. Moreover, information about dependency or in-dependency of application primitives is not preserved in the application trace. Although more information could be added to the basic read, write, and execute instructions in the TD traces, it may not always be natural to do so, because of the TD approach feature that it operates on only a single set of data.

3.2 CDFG Mapping Approach

While the TD approach was there as a DSE approach, the CDFG approach was not. We had to pull the CDFG approach out of the low-level design [16] and to place it at system-level in order to make the comparison. In the CDFG approach, the application model is captured in a representation that preserves the functionality of all the application tasks, including control constructs. These representations take the form of Control Data Flow Graphs (CDFGs) [16]. As shown in Figure 1 (right part), an application CDFG is derived by parsing the source code. Then, the mapping layer takes as input the application CDFG, and produces a transformed CDFG that is used in the architecture simulation. During the execution of the architecture model, run-time scheduling may also be performed. Since functionality has been preserved in the mapping process, actual data sets can now be processed during architecture simulations. This may enhance the accuracy of these simulations.

The final objective of this approach is to obtain a source code that can be easily transfered to detailed design. In our opinion the trajectory of the TCM approach [2] comes close to the CDFG approach, be it that the nature of the *control constructs* is still static (cyclo-static) [10]. Similarly to the CDFG transformations (transformation steps), the TCM approach includes chained activities. These activities are performed on the system model specified at a *grey-box* level of abstraction. What the grey-box level models are doing is hiding details that are not, at least not explicitly, related with the concurrency and system trade-offs. Therefore, these models are sufficiently accurate since they still have all information needed for the system synthesis.

In contrast to traces, CDFGs have to be first compiled by a cross-platform compiler in order to be loaded into the architecture simulator. As a consequence of operating on CDFGs, manipulating CDFGs is more complex than manipulating traces. The reason is simply due to the fact that CDFGs can (and usually do) contain *hierarchy*, *recursion*, and *cycles*. In addition, CDFGs are more fine grained than traces used in the TD approach. Ultimately, the CDFG simulation becomes as detailed as instruction set simulation, and

hence simulation times grow rapidly.

3.3 A Hybrid Mapping Approach - Symbolic Program

The two approaches we mentioned so far can also be qualified as follows: in the trace driven approach the mapping steps transform traces, whereas in the CDFG approach the mapping steps transform CDFGs. In both approaches the architecture models can have varying degrees of abstraction. The CDFG approach somewhat resembles a flow for system design, as the transformed CDFGs may be used to generate refined application code that can be used upon implementation. On the other hand, the TD approach is more exploration specific and does not produce refined application code that can be used in detailed design. Hence, the TD approach helps the architect to gain insight and evaluate design alternatives, but does not yield mapped code for implementing the selected alternative.

Given the pros and cons of both approaches, it is natural to look for an approach that is somehow in between these two extremes, i.e., that is suitable for both exploration and design purposes while offering sufficient accuracy and high simulation speeds for effective explorations. We have already noticed earlier that this would imply that additional (control) information would have to be inserted in the trace. This, however, would boil down to move from symbolic instructions to Symbolic Programs (SP). On the other hand, in an exploration context, we would still like to simulate architecture organisations in a trace driven way, due to the high simulation speed of the TD approach. Remember that a major property of the TD approach is that data is not processed in the architecture model. Therefore, symbolic instructions cannot simply be replaced with SP, since control in the SP needs to be resolved upon architecture simulation. However, the combination of control construct outcomes for a particular data-set (*trace of control outcomes*) can be provided by an execution of the application model so that with this information traces can again be generated from the SP. This being the case, we propose a novel mapping approach for DSE which is depicted in Figure 1, and is as follows.

The SP approach is based on two mapping model components: one that provides the control outcome information in a TD manner, and the other that provides SP code, which is a compact CDFG-like representation of a YAPI process. The control outcome trace is needed since SPs are not executable but do contain control constructs (see Table 1). The transformation steps are denoted as *trace transformations* and *SP* (or code) *transformations* in

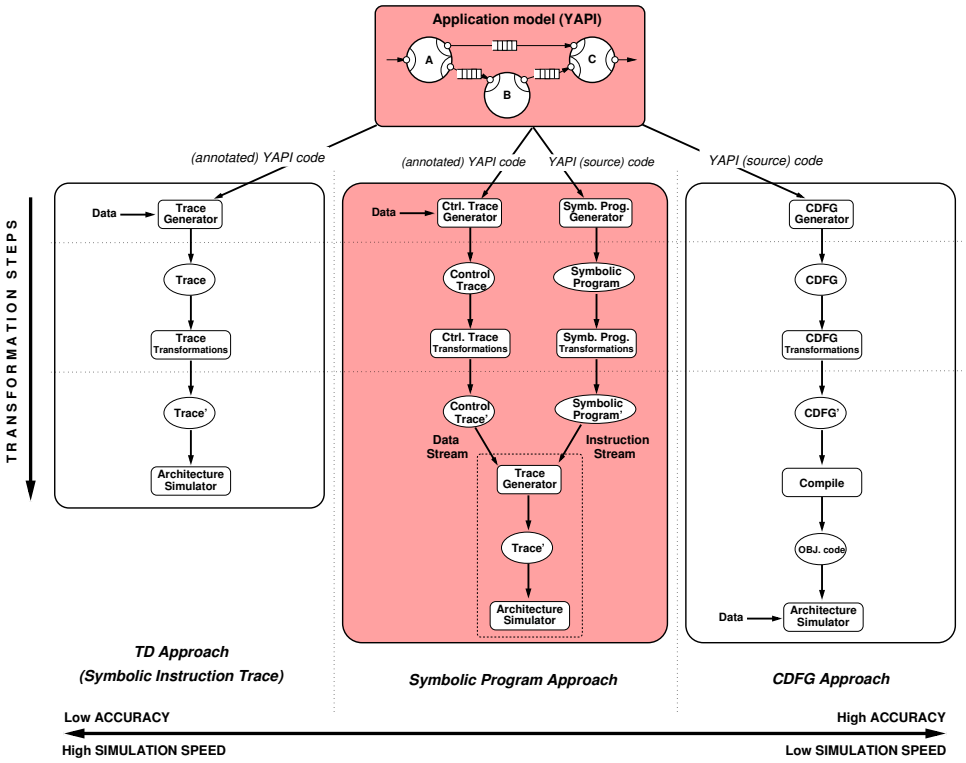

Fig. 1. The SP approach - Hybrid mapping approach

Figure 1. The transformations apply the techniques needed to support refinement of both the control outcome trace and the symbolic program. The more refined the SPs and the control traces that drive the architecture model are, the more accurate the results will be.

Figure 1 also shows the positioning of the three approaches with respect to accuracy and simulation speed. As a hybrid solution the SP approach contains the control information while the TD approach does not. This makes possible that the cost of the control can be included in the architecture simulation. Also, the non-determinism can be handled more easily than in the TD approach. On the other hand, the control does not need to be fully implemented, which is beneficial compared to the CDFG approach. Also, transforming the SPs is easier than transforming CDFGs, since they are more compact. Therefore, the SP approach represents a DSE approach where designers can (1) perform design-steps as in the case of detailed design (indicated with dashed lines in Figure 1), (2) run fast simulations of

architectures being explored, and (3) have more accurate numbers than in the case of the TD simulations [11], [7].

4 An Illustrative Example

The SP-based model we introduced in [15] can be used for modelling and simulation of design cases where the TD-based model [11] can not. We support the previous claim with an example of mapping an *affine nested loop program* (ANLP) process with no control-data dependencies on top of the unit instantiated from the SP-based model. We choose such program deliberately because we want to show the impact of this unit on the execution time. We want to underline that this does not limit the usage of the SP-based model only on such applications. We move the ANLP process through some of the code (SP) transformations mentioned earlier.

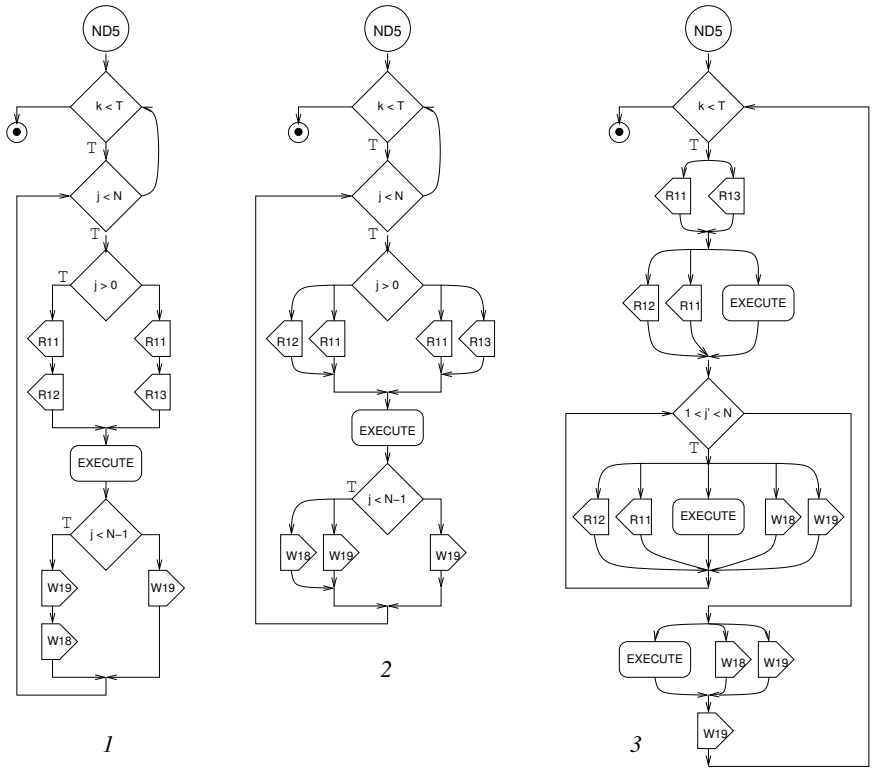

Fig. 2. The CDFGs of ND5 before and after code transformations being applied

Figure 2 part *1* shows the CDFG of the ND5 process. This CDFG is a visualisation of the SP produced by the SP Generator (see Figure 1). The SP can be transformed further into the other SP with the VLIW-like execution of independent *read*s, *write*s, and *execute*s allowed (see Figure 2 part *2*), or even transformed into the another SP that supports both the VLIW-like execution and software pipelining [9] (see Figure 2 part *3*).

Let us now consider that the processing unit operates in the "backwards compatible mode," or, in other words, that we use the Trace Driven execution model that we feed with a sequential trace. If we run the previously described SPs on-top of such a model the performance numbers will show no difference. The performance numbers we have mentioned here represent the amounts of execution times in each case. However, if the unit with more flexibility than the previous one is employed, the relative improvement is as illustrated in Figure 3. Note that one should not interpret these numbers as the performance improvements, but rather as an *improvement of the modelling flexibility of the system-level processor model.* The improvements of flexibility for the SPs in Figure 2 parts *2* and *3* are given in Figure 3 as $i1(x)$ and $i2(x)$, respectively. The improvements depend on a set of parameters, from which we choose to expose the "communication-budgets versus computation-budgets" ratio[1]. As one can see, the improvement can vary between 0 and 0.76 with respect to the type of the processing element being used in system-level simulation. This means that absence of modelling features can seriously jeopardise the simulation accuracy.

5 Conclusions

Starting from previous work based on the TD approach [11], we identified the "opposite" DSE approach which we call the CDFG approach. After searching for pros and cons of these two extreme approaches, we proposed a new hybrid mapping approach for DSE - the SP approach - which is the main contribution of this paper. The main advantages of the proposed SP solution are that (1) the important characteristics of both extreme approaches are preserved (namely, the fast TD-based simulations, and the CDFG control constructs), and consequently (2) that weak aspects of both extreme approaches are improved (namely, the accuracy of the TD approach, and the inefficiency of the CDFG approach). This yields a DSE approach that is both accurate and efficient, while it connects well to more detailed design

[1] We assume delays associated with R and W primitives are identical

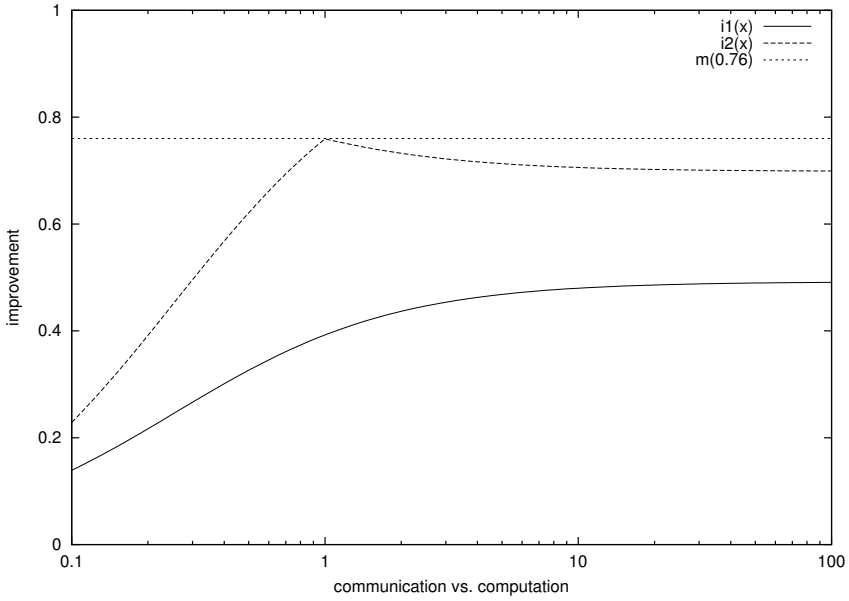

Fig. 3. The relative improvement between 1. and 2., and 1. and 3.

trajectories.

Acknowledgments

This work was performed in part in the Archer project, funded by Philips Semiconductors. We want to tank Todor Stefanov (Leiden University) and Ondrej Popp (Philips Research Laboratories) for their contributions to this paper.

References

[1] B. Kienhuis, et al., *An Approach for Quantitative Analysis of Application-specific Dataflow Architectures*, in Proceedings of ASAP'97, July 14-16, 1997.

[2] C. Wong, P. Marchal, and P. Yang, *Task concurrency management methodology to schedule the MPEG4 IM1 player on a highly parallel processor platform*, in Proc. International Symposium on Hardware/software CODES'01, Denmark, Apr. 2001.

[3] De Kock et al., *YAPI: Application modelling for signal processing systems*, in Proc. of DAC'2000, LA, USA, June 2000.

[4] G. Kahn, *The semantics of a simple language for parallel programming*, Information processing 74 - North-Holland Publishing Company, 1974.
[5] Jean-Yves Brunel et al., *COSY Communication IP's*, in Proceedings of DAC'2000, Los Angeles, USA, June 2000.
[6] J. Hennessy, D. Patterson, *Computer Architecture - A Quantitive Approach*, Morgan Kaufmann Publishers, 1996.
[7] K. Lahiri, A. Raghunathan, and S. Dey, Fast Performance Analysis of Bus-Based Systems-On-Chip Communication Architectures, in Proc. IEEE/ACM ICCAD'99, San Jose, CA, Nov. 7-11 1999, pp. 566-572.
[8] Keshab Parhi, *VLSI Digital Signal Processing Systems - Design and Implementation*, John Wiley & Sons, 1999.
[9] Monica Lam, *Software Pipelinening: An Effective Scheduling Technique for VLIW Machines*, in Proc. of SIGPLAN'88 Conference on Programming Language Design and Implementation, Atlanta, Georgia, June 22-24, 1988.
[10] N. Cossement, R. Lauwereins, and F. Catthoor, *DF*: An extension of synchronous dataflow with data dependency and non-determinism*, in Forum on Design Languages, Tuebingen, Germany, Sep. 2000.
[11] P. Lieverse et al., *A methodology for architecture exploration of heterogeneous signal processing systems*, in Proc. 1999 Workshop on Signal Processing Systems, Taipei, Taiwan, Oct. 1999.
[12] P. Lieverse, P. van der Wolf, and E. Deprettere, *A Trace Transformation Technique for Communication Refinement*, in Proc. International Symposium on Hardware/software CODES'01, Denmark, Apr. 2001.
[13] S. Sriram, S. Bhattacharyya *Embedded Multiprocessors - Scheduling and Synchronization*, Marcel Dekker, 2000.
[14] T. Stefanov et al., *System Level Design with Spade: an M-JPEG Case Study*, in Proc. IEEE/ACM ICCAD'01, San Jose, CA, Nov. 4-8 2001.
[15] Vladimir D. Živković et al., *Design Space Exploration of Streaming Multiprocessor Architectures*, in Proc. 2002 Workshop on Signal Processing Systems, San Diego, California, Oct. 2002.
[16] Wayne Wolf, *Computers as Components - Principles of Embedded Computing System Design*, Morgan Kaufmann Publishers, 2001.