A METHODOLOGY FOR ARCHITECTURE EXPLORATION OF HETEROGENEOUS SIGNAL PROCESSING SYSTEMS

Paul Lieverse[1], Pieter van der Wolf[2], Ed Deprettere[1], Kees Vissers[2]
[1]Delft University of Technology, Dept. of Information Technology and Systems, Delft, The Netherlands
[2]Philips Research Laboratories, Eindhoven, The Netherlands
email: p.lieverse@its.tudelft.nl

Abstract — We present a methodology for the exploration of signal processing architectures at the system level. The methodology, named SPADE, provides a means to quickly build models of architectures at an abstract level, to easily map applications, modeled as Kahn Process Networks, onto these architecture models, and to analyze the performance of the resulting system by simulation. The methodology distinguishes between applications and architectures, and uses a trace-driven simulation technique for co-simulation of application models and architecture models. As a consequence, architecture models need not be functionally complete to be used for performance analysis while data dependent behavior is still handled correctly. We have used the methodology for the exploration of architectures and mappings of an MPEG-2 decoder application.

1 INTRODUCTION AND PROBLEM DEFINITION

Modern signal processing systems are increasingly becoming multi-functional systems that also have to support multiple standards. For example, digital televisions, set-top boxes, and mobile devices have to offer a variety of functions and must support different standards for transmission and coding of digital contents. This calls for programmability. However, performance requirements and constraints on cost and power consumption still require significant parts of these systems to be implemented in dedicated hardware blocks. As a consequence, such systems have a *heterogeneous architecture*, i.e., they consist of both programmable and dedicated components. What is needed is design technology that helps designers to define such heterogeneous architectures starting from a *set* of target applications. A classical approach that departs from a single application and iteratively pushes it through some transformational process towards a dedicated implementation architecture, is not suited for the design of programmable systems.

In this paper we present a methodology, named SPADE (System level Performance Analysis and Design space Exploration), for *architecture exploration* of heterogeneous signal processing systems. This exploration starts from executable specifications of a *set* of representative target *applications*. The result is the definition of an *architecture* capable of executing these applications within predefined constraints with respect to cost, real-time response, etc.

In order to support efficient architecture exploration, the design has to start with abstract, yet executable models. This is illustrated in Figure 1. The cost of model construction and model evaluation is higher at the more detailed levels of abstraction, while the flexibility to explore alternative architectures is significantly reduced at these levels. The SPADE methodology permits architecture exploration to start at a level where abstract executable models can be used to efficiently evaluate architectures for the selected set of applications. A key feature of the SPADE methodology

is that it allows architectures to be evaluated for different applications and, moreover, that this can easily be repeated for different architectures. Further, it provides a linkup with more detailed design trajectories.
In Section 2 we discuss related work. Section 3 presents a number of basic principles underlying the SPADE methodology. The methodology is discussed in detail in section 4. Section 5 presents a case study that we have performed with SPADE. Conclusions are presented in section 6.

2 RELATED WORK

In the field of application modeling a lot of research has been done on *models of computation*. Known models such as Synchronous Dataflow (SDF) [1], Dataflow Process Networks [2], and Kahn Process Networks [3] have been studied thoroughly. In the Ptolemy project [4] these research efforts have been brought together and united to constitute a basis for the Ptolemy framework. Ptolemy mainly focuses on application modeling and simulation, but does not yet support explicit mapping of application models onto models of architectures.
Various groups are currently working on architecture modeling and performance analysis at the system level, each with a different approach. In the Polis [5] environment, applications are described as a network of Codesign Finite State Machines (CFSMs). This model is very well suited for reactive systems, but less suited for signal processing applications. Each element in the network of CFSMs can be mapped onto either the hardware part or the software part of an architecture consisting of a microcontroller and dedicated hardware. The performance of such a mapping can be evaluated by simulation, in which the software execution times for a particular microcontroller are estimated by accumulating the latencies of abstract RISC instructions. Both hardware and software, including a real-time kernel, can be synthesized from the CFSM description.
Chinook [6][7] is targeted towards the design of embedded systems composed of commercial off-the-shelf components or IPs, typically microprocessors or programmable logic blocks. There is an explicit mapping from a behavioral description onto a target architecture. Communication between components can be synthesized, including device drivers, for standard communication protocols, such as I^2C, CAN, and Ethernet. The simulator tool, Pia [8], allows the communication of a system to be simulated at different levels of abstraction.
In the RASSP project [9] DSP systems are modeled completely in VHDL, at differ-

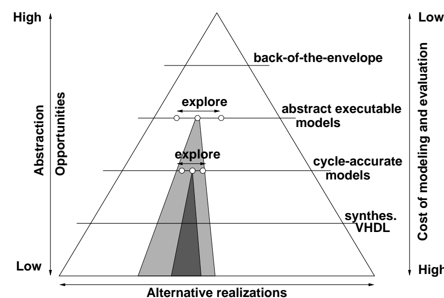

Figure 1: Abstraction pyramid: successive exploration at increasing levels of detail.

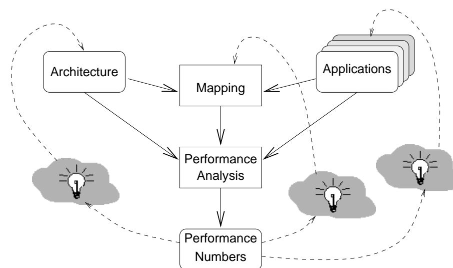

Figure 2: The Y-chart: a general scheme for the design of programmable architectures.

ent levels of abstraction. Performance models model only the timing behavior, not the functional behavior. Abstract behavioral models capture both functionality and timing; the functionality is completely incorporated in the architecture model. As part of the RASSP project, the ADEPT environment has been developed [10]. In this environment, a system is constructed by interconnecting a collection of predefined architecture modules. There is no separation between application and architecture; architecture modules contain both functional and timing behavior.

In [11] an approach for quantitative analysis of architectures is presented that uses abstract architecture models. As a result, architecture models can be constructed easily and the evaluation of the performance can be done very fast, which allows for the exploration of a large number of design alternatives. The environment described in [11] is however limited to a specific class of dataflow architectures.

Often, designers build a fully functional dedicated C/C++ model for a specific system architecture, e.g. [12]. A framework such as Scenic [13] may alleviate this task. However, the reusability of such models is moderate.

In contrast to most of these approaches, SPADE distinguishes between application models and architecture models, and supports an explicit mapping step. Architecture models do not need to model the functional behavior, while SPADE still handles data dependent behavior correctly by employing a specific simulation technique. As a consequence, a high reusability of both application models and generic architecture building blocks is obtained. This enables efficient exploration of alternative architectures. The modeling techniques employed by SPADE make it particularly suited for signal processing systems.

3 BASIC PRINCIPLES

3.1 The Y-Chart

We claim that for the design of programmable systems a clear distinction must be made between *applications* and *architectures*, and that an explicit *mapping* step must be supported. This will permit multiple target applications to be mapped one after the other onto candidate architectures for evaluation of their performance. In line with Kienhuis et al. [14] we conclude that the development of programmable architectures should follow a general scheme, which can be visualized by the Y-shaped figure in Figure 2, the *Y-chart*.[1]

In the upper right part of Figure 2 is the set of applications that drives the design of the *architecture*. Typically, a designer studies this set of applications, makes some initial calculations and proposes an architecture. The effectiveness of this architecture is then to be evaluated and a comparison with alternative architectures is to be made. Architectures are evaluated quantitatively by means of *performance analysis*. For this performance analysis, each application is *mapped* onto the architecture and the performance of each application–architecture–mapping combination is evaluated. The resulting performance numbers may inspire the architecture designer to improve the architecture. He may also decide to restructure the application(s) or to modify the mapping of the application(s). These designer actions are denoted by the light bulbs in Figure 2.

[1] Note that Gajski has proposed a different Y-chart to illustrate three dimensions in IC design [15].

3.2 Workload and Resources

The distinction between applications and architectures is based on the principle visualized in Figure 3. An application imposes a *workload* on the *resources* provided by an architecture. The workload consists of both *computation workload* and *communication workload*. The resources can be *processing resources*, such as programmable cores or dedicated hardware units, *communication resources*, such as bus structures, and *memory resources*, such as RAMs or FIFO buffers. The architecture design process is concerned with the specification of the resources that can best handle the workloads imposed by the target applications.
In SPADE, applications and architectures are modeled separately. Application models should be *functional models* that are relatively free of architectural aspects. Vice versa, architecture models should define the architecture resources in such a way that they can be used for all applications from the benchmark set. Such a *decoupling* enables *reuse* of both application models and architecture models and facilitates an explorative design process in which application models are subsequently mapped onto architecture models.

3.3 Trace-Driven Simulation

In order to evaluate the performance of an application–architecture–mapping combination, we must provide the *interfacing* of application models to architecture models, including specification of mappings. For this purpose we extend a technique called *trace-driven simulation*. This is a simulation technique that has been applied extensively for memory system simulation in the field of general-purpose processor design [16]. We use the technique for performance analysis of heterogeneous systems. The application model is structured as a network of *concurrent communicating processes*. Each process produces upon execution a so-called *trace*, which represents the workload imposed on an architecture by that process. Thus, a trace contains information on the communication and computation operations performed by an application process. These operations may be *coarse-grain*. Therefore, our approach differs from classical trace-driven simulation, in which traces contain information on fine-grain RISC operations. Data dependent behavior at the application level is captured in the traces, i.e., the trace data depends on the input data of the application. The traces get interfaced to an architecture model, which accepts them as the workload to be executed; see Figure 4. The traces drive computation and communication activities in the architecture. These activities are executed as specified by the architecture model. During this execution, *time* gets assigned to all events that occur in the architecture model and the performance of the execution of the application on the architecture can be measured.

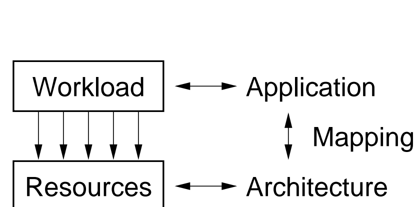

Figure 3: An application imposes a workload on the resources of an architecture.

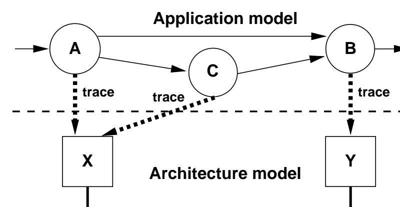

Figure 4: Trace-driven simulation: the execution of the architecture model is driven by traces from the execution of the application model.

4 METHODOLOGY

4.1 Application Modeling

One of the objectives of application modeling is to expose parallelism while making communication explicit. We have chosen to use the Kahn Process Networks [3] model of computation for application modeling. In the Kahn model, parallel *processes* communicate via unbounded FIFO *channels*. Each process executes sequentially. Reading from channels is blocking; writing to channels is non-blocking. The Kahn model is timeless; there is only an ordering on the data in each channel. We have chosen the Kahn model for the following reasons.

- The execution of a Kahn Process Network is deterministic, meaning that for a given input always the same output and the same traces are produced.
- The model fits nicely with signal processing applications, as it can model *stream processing* and it guarantees that no data is lost.
- The model allows an application programmer to easily combine communication primitives, including blocking reads, with control constructs, such as while loops and if-then-else statements. Compared to other models of computation that require applications to be built of atomically firing actors which return control after firing, such as pure dataflow models [2], the Kahn model makes it much easier to partition an application into a set of parallel communicating processes, also when it was initially described as a sequential C/C++ program. As applications often are available as sequential (C-)programs, this is an important advantage.

The top half of Figure 5 shows an example of an application modeled as a Kahn Process Network. SPADE offers an Application Programmers Interface (API) for application modeling that contains the following three functions.

- A *read* function. This function is used to read data from a channel via a process port. Furthermore, the function generates a *trace entry* in the trace of the process by which it is invoked, reporting on the execution of a read operation at the application level.
- A *write* function. This function is used to write data to a channel via a process port. It also generates a trace entry, reporting on the execution of a write operation.
- An *execute* function. This function performs no data processing, but only gener-

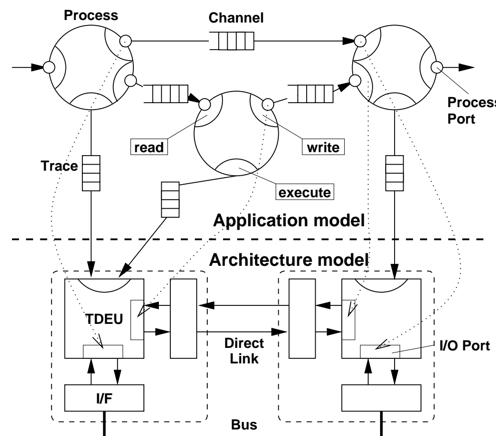

Figure 5: Example of an application model, architecture model, and mapping. The top half shows an application modeled as a Kahn Process Network. Processes are depicted as circles, small circles are process ports, and the circle segments in the processes represent API functions. The bottom half shows an architecture model consisting of two processing resources (the dashed boxes), that each are composed of a Trace Driven Execution Unit and two interfaces, and a bus. The mapping of processes onto processing resources is indicated by the traces; the dotted arrows indicate the port mapping.

ates a trace entry, reporting on processing activities at the application level. The execute function takes a *symbolic instruction* as an argument in order to distinguish between different processing activities. For example, such an instruction may correspond to an IDCT operation on an eight by eight matrix.

The use of the API functions is illustrated by the code fragment in Figure 6.

```
while(1) {
  read(Input, token);
  Process_Token(&token);
  execute(PROCESS_TOKEN);
  write(Output, token);
}
```

Figure 6: Code fragment of an application process, illustrating the use of the API functions read, write and execute. PROCESS_TOKEN is a symbolic instruction.

The trace entries generated by the read and write functions represent the *communication workload* of a process. The trace entries generated by the execute function represent the *computation workload* of a process. The trace entries can be used either to drive the architecture simulation, or, when executing the application *stand-alone*, to analyze the computation and communication workload of an application. The results of such a *workload analysis* are presented in the form of two tables, as exemplified in Table 1. This example relates to the MPEG-2 decoder case described in Section 5. The computation workload table shows the number of times each symbolic instruction is called by the processes. The communication workload table shows the number of tokens and the total number of bytes that is communicated through each channel.

Table 1: Result of a workload analysis

Computation workload.

Process	**Instruction**	**Frequency**
Tidct	IDCT_MB	12,514
Tadd	Skipped_MB	158
Tadd	Intra_MB	2,037
...	...	...

Communication workload.

Channel	**#Tokens**	**#Bytes**
predict_data	88,218	5,645,952
predict_mv	12,514	400,448
...	...	...

4.2 Architecture Modeling

In order to efficiently explore different architectures, it is required that architecture models can be easily constructed. In SPADE, functional behavior is described at the application level. If this behavior is data dependent, the traces, which drive the operation of the architecture, also depend on the input data. Therefore, we can use architecture models that do not need to model the functional behavior, while maintaining functional correctness. Such architecture models can be constructed from *generic building blocks*. As the building blocks are generic, we can provide a library of such blocks. The generic building blocks need to model the different types of resources in an architecture, such as processing resources, communication resources, and memory resources. Defining an architecture then becomes as easy as instantiating building blocks from a library and interconnecting them. Compared to the laborious work of writing fully functional architecture models this can save a designer a lot of time, and therefore enables the exploration of alternative architectures.

The processing resources in the architecture model take the traces generated by the application as an input. We have taken a modular approach to allow the construction of

a great variety of processing resources from a small number of basic building blocks. A processing resource is built from the following two types of blocks.

- A *trace driven execution unit (TDEU)* which interprets trace entries. The entries are interpreted in the order in which they are put in the trace, thereby retaining the order of execution of the application process. A TDEU has a configurable number of I/O ports. Communication via these I/O ports is based on a generic protocol.
- A number of *interfaces*, which connect the I/O ports of a TDEU to a specific communication resource. An interface translates the generic protocol into a communication resource specific protocol, and may also include buffers to model input/output buffering of processing resources. Currently, we have interfaces for communication via a direct link, via a shared bus, and via shared memory, both buffered and unbuffered.

Apart from the TDEU and interface blocks, the current library contains a generic bus block, including a first-come-first-serve arbiter, and a generic memory block. All blocks are parameterized. For each instantiated TDEU a list of *symbolic instructions* and their *latencies* has to be given. This list specifies which instructions from the traces can be executed by the processing resource and how many cycles each instruction takes when executed on this processing resource. These latencies can be obtained either from a lower level model of a processing resource, from estimation tools, or they can be estimated by an experienced designer. For instances of the interface blocks, buffer sizes can be given. For a bus instance, the bus width, setup delay, and transfer delay can be specified. The bottom half of Figure 5 shows an example architecture consisting of two processing resources, each composed of a TDEU and two different interfaces, and a bus. Recall that the generic building blocks are abstract performance models; they only model the timing and synchronization of the architecture. The application model captures the functional behavior.

In addition to the use of the generic building blocks, Spade permits the use of user defined blocks in the definition of architecture models.

4.3 Mapping

Once both an application model and an architecture model have been defined, mapping can be performed. This means that the workload of the application has to be assigned to resources in the architecture as follows.

- Each process is mapped onto a TDEU. This mapping can be many-to-one, in which case the trace entries of the processes need to be scheduled by the TDEU.
- Each process port is mapped one-to-one onto an I/O port. This mapping also implicitly maps the channels onto a combination of communication resources and memory resources, possibly including user defined blocks, such as a specific bus model or memory interface. Typically, these resources do not have an equivalent element in the application model.

An example mapping is shown in Figure 5.

If it appears that the functionality of a single process needs to be distributed over more than one processing resource, then the designer first has to rewrite the application such that this process is partitioned into two or more processes. Then these processes can be mapped onto separate TDEUs.

4.4 Simulation

The simulation is performed by concurrently simulating the application model and the architecture model in a single memory space. Traces are generated *on-the-fly* by the application to drive the architecture simulation. Storage of large amounts of trace information is thus avoided.
The simulation of the application model is based on the Pamela [17] multi-threading environment, where each Kahn process is executed in a separate thread. The simulation of the architecture model is currently based on TSS (Tool for System Simulation), which is a Philips in-house architecture modeling and simulation framework [18]. The library of generic blocks is implemented as a library of TSS modules. SPADE permits the use of user defined blocks, implemented as TSS modules.

4.5 Performance Metrics

The generic building blocks already contain collectors for different performance metrics. The metrics for which data is collected during simulation are resource specific. For the TDEUs, these metrics are the utilization and the number of read and write stall cycles on the I/O ports. For the generic bus model the amount of data that is sent over the bus, the utilization, and the wait times for the bus are collected.

5 CASE STUDY: AN MPEG-2 DECODER

We have used the SPADE methodology for the exploration of the architecture and mapping of an MPEG-2 video decoder application [19]. The exercise started from the C-code of the MPEG-2 video decoder that is publicly available from UC Berkeley. The first step was to partition this sequential program into a parallel Kahn Process Network using the API functions read, write, and execute. The result is shown in Figure 7.

Figure 7: An MPEG-2 video decoder application structured as a Kahn Process Network.

By running the application in stand-alone mode we collected statistics of the workload for different MPEG sequences, both in terms of numbers of operations performed (e.g. the number of IDCT operations) and amounts of data communicated (e.g. the number of blocks transferred from Tisiq to Tidct), as shown in Table 1.
Architecture exploration started from a realistic architecture for MPEG-2 video decoding, being the TM-2700 MPEG decoder architecture from the Philips TriMedia Group [20]. The TM-2700 is a heterogeneous system consisting of a dedicated MPEG decoder attached to a bus structure together with a VLIW CPU and several other dedicated co-processors. We constructed a model for this architecture using only the generic blocks from the library; see Figure 8. No additional blocks had to be built in order to model this architecture.

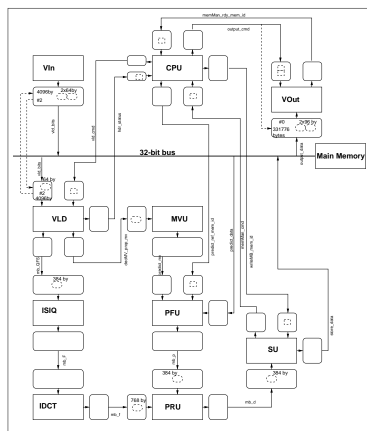

Figure 8: MPEG-2 video decoder architecture model. The rectangular boxes are TDEUs; the rounded boxes are interfaces.

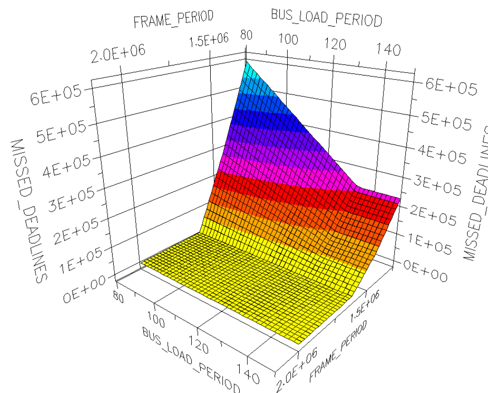

Figure 9: Measure for missed deadlines as a function of frame period and period between bus requests of an additional processor.

Subsequently, a mapping was defined. Next, simulations could be performed, yielding a wealth of performance data on the operation of the architecture. By interpreting the performance data, the understanding of the operation of the architecture was enhanced and bottlenecks were identified.

SPADE has further been integrated with an environment that supports design space exploration by automatically performing multiple simulations at different points in the design space [21]. This facility is useful for performing *sensitivity analysis* in a multi-dimensional parameter space, and to identify valid ranges for latencies of the architecture components. For example, we studied the ability of the architecture to meet the deadlines for frame decoding in relation to the frame rate and some additional bus load. The additional bus load was generated by an extra processor that periodically claims the bus. Figure 9 shows a measure for the missed deadlines as a function of the frame period (in cycles at 200 MHz) and the period of the bus requests of the additional processor. The flat part of the figure is the part where all deadlines were met; for a high bus load (small period) and for a high frame rate deadlines are missed. The simulation speed for the MPEG-2 decoder application mapped onto the architecture model with 10 processing resources, 33 interface blocks, a bus, and a memory, was about 20,000 cycles per second, which is about two minutes per frame.

6 CONCLUSION

SPADE supports efficient exploration of heterogeneous signal processing architectures that must satisfy the workload demands of multiple target applications. Applications can be structured starting from available C-code using the Kahn API functions. The output of this task is a reusable application model which provides statistics on the workload that the architecture must handle. Following the Y-chart, SPADE distinguishes between application models and architecture models. It uses a trace-driven simulation technique for co-simulation. As a consequence, we can use architecture models that do not need to model the functional behavior for performance analysis,

while still handling data dependent behavior correctly. A broad class of architectures can be modeled effectively and efficiently with the generic architecture blocks from the library. The simulation speed, currently about 20,000 cycles per second for a relatively complex design, enables the exploration of a number of architectures and mappings. The TSS architecture simulator further allows the inclusion of more dedicated architecture blocks and thereby also allows incremental detailing starting from the abstract SPADE architecture models.

ACKNOWLEDGMENTS

We want to thank Mudit Goel (UC Berkeley), David La Hei, and Ronald Marcelis for their contributions to the SPADE work.

REFERENCES

[1] Edward A. Lee and David G. Messerschmitt, "Synchronous data flow," *Proc. of the IEEE*, vol. 75, no. 9, pp. 1235–1245, Sept. 1987.

[2] Edward A. Lee and Thomas M. Parks, "Dataflow process networks," *Proc. of the IEEE*, vol. 83, no. 5, pp. 773–799, May 1995.

[3] Gilles Kahn, "The semantics of a simple language for parallel programming," in *Proc. of the IFIP Congress 74*. 1974, North-Holland Publishing Co.

[4] Joseph Buck, Soonhoi Ha, Edward A. Lee, and David G. Messerschmitt, "Ptolemy: A framework for simulating and prototyping heterogeneous systems," *International Journal of Computer Simulation*, Aug. 31 1992, Special issue on Simulation Software Development.

[5] F. Balarin, E. Sentovich, M Chiodo, P. Giusto, H. Hsieh, B Tabbara, A. Jurecska, L. Lavagno, C. Passerone, K. Suzuki, and A. Sangiovanni-Vincentelli, *Hardware-Software Co-design of Embedded Systems – The POLIS approach*, Kluwer Academic Publishers, 1997.

[6] Pai H. Chou, Ross B. Ortega, and Borriello Gaetano, "The Chinook hardware/software co-synthesis system," in *Proc. ISSS'95*, 1995.

[7] Pai Chou, Ross Ortega, Ken Hines, Kurt Partidge, and Gaetano Borriello, "IPCHINOOK: An integrated IP-based design framework for distributed embedded systems," in *Proc. DAC'99*, New Orleans, LA, June 21-25 1999.

[8] Ken Hines and Gaetano Borriello, "Dynamic communication models in embedded system co-simulation," in *Proc. DAC'97*, Anaheim, California, June 9-13 1997.

[9] C. Hein, J. Pridgen, and W. Kline, "RASSP virtual prototyping of DSP systems," in *Proc. DAC'97*, Anaheim, California, June 9-13 1997.

[10] Robert H. Klenke, Moshe Meyassed, James H. Aylor, Barry W. Johnson, Ramesh Rao, and Anup Ghosh, "An integrated design environment for performance and dependability analysis," in *Proc. DAC'97*, Anaheim, California, June 9-13 1997.

[11] A.C.J. Kienhuis, *Design Space Exploration of Stream-based Dataflow Architectures; Methods and Tools*, Ph.D. thesis, Delft University of Technology, 1999.

[12] Dale E. Hocevar, Ching-Yu Hung, Dan Pickens, and Sundararajan Sriram, "Top-down design using cycle based simulation: an MPEG A/V decoder example," in *Proc. Great Lakes Symposium on VLSI'98*, Lafayette, Louisiana, Feb. 19-24 1998.

[13] Rajesh K. Gupta and Stan Y. Liao, "Using a programming language for digital system design," *IEEE Design and Test of Computers*, vol. 14, no. 2, pp. 72–80, Apr.-June 1997.

[14] B. Kienhuis, E. Deprettere, K. Vissers, and P. van der Wolf, "An approach for quantitative analysis of application-specific dataflow architectures," in *Proc. ASAP'97*, July 14-16 1997.

[15] D. Gajski, *Silicon Compilers*, Addison-Wesley, 1987.

[16] Richard A. Uhlig and Trevor N. Mudge, "Trace-driven memory simulation: A survey," *ACM Computing Surveys*, vol. 29, no. 2, pp. 128–170, June 1997.

[17] A.J.C. van Gemund, "Performance prediction of parallel processing systems: The PAMELA methodology," in *Proc. 7th ACM Int. Conference on Supercomputing*, Tokyo, July 1993, pp. 318–327.

[18] Wido Kruijtzer, "TSS: Tool for System Simulation," *IST Newsletter*, vol. 17, pp. 5–7, Mar. 1997, Philips Internal Publication.

[19] Pieter van der Wolf, Paul Lieverse, Mudit Goel, David La Hei, and Kees Vissers, "An MPEG-2 decoder case study as a driver for a system level design methodology," in *Proc. 7th International Workshop on Hardware/Software Codesign CODES'99*, Rome, Italy, May 3-5 1999.

[20] Selliah Rathnam, "A single chip DTV media processor," in *Proc. of Hot Chips 10: A Symposium on High Performance Chips*, Aug. 16-18 1998.

[21] Peter Bingley and Wim van der Linden, "Application of framework technology in system simulation environments," in *Proceedings of the Seminar 'Database Systems and Applications for the Nineties'*. Delft University of Technology, Oct. 11-12 1994.