An MPEG-2 Decoder Case Study as a Driver for a System Level Design Methodology

Pieter van der Wolf[1], Paul Lieverse[2], Mudit Goel[3], David La Hei[1], Kees Vissers[1]

[1]Philips Research Laboratories, Prof. Holstlaan 4, Eindhoven, The Netherlands
[2]Delft University of Technology, Dept. of ITS, Delft, The Netherlands
[3]University of California, Berkeley, Dept. of EECS, Berkeley, U.S.A.

vdwolf@natlab.research.philips.com

Abstract

We present a case study on the design of a heterogeneous architecture for MPEG-2 video decoding. The primary objective of the case study is the validation of the SPADE methodology for architecture exploration. The case study demonstrates that this methodology provides a structured approach to the efficient evaluation of the performance of candidate architectures for selected benchmark applications. We learned that the MPEG-2 decoder can conveniently be modeled as a Kahn Process Network using a simple API. Abstract models of architectures can be constructed efficiently using a library of generic building blocks. A trace driven simulation technique enables the use of these abstract models for performance analysis with correct handling of data dependent behavior. We performed a design space exploration to derive how the performance of the decoder depends on the busload and the frame rate.

1 Introduction and Objectives

In this paper we focus on the design of heterogeneous systems architectures for complex programmable systems. A methodology is needed that supports the definition of a *heterogeneous architecture*, i.e. an architecture consisting of both programmable and dedicated components, starting from a *set of benchmark applications*. For a broad class of applications this methodology must support the efficient exploration of candidate architectures based on *quantitative evaluation* of the *performance* of these architectures for selected benchmark applications.

We are developing such a methodology under the name *Spade* (System level Performance Analysis and Design space Exploration). SPADE aims to support efficient architecture exploration by permitting architectures to be modeled at an *abstract level*. We felt that early feedback on this new methodology was essential for the further development of concepts and tools. For example, does the designer get relevant data on the performance of proposed architectures, can simulations be performed efficiently, etc. Therefore, we decided to perform an industrially relevant case study with the newly developed prototype tools. We selected an ATSC compliant MPEG-2 video decoder as benchmark application. MPEG-2 video decoding is a high performance signal processing application that exhibits dynamic, scene dependent behavior. For this application we had to propose an architecture and to evaluate the performance of the application on the proposed architecture, including an analysis of the sensitivity to various design parameters. The objectives of the case study were to validate the basic principles of the SPADE methodology and to verify the prototype tools and the library of architecture building blocks that comes with the tools.

In the next section we briefly summarize the SPADE methodology. The case study is presented in sections 3 to 7. We summarize the lessons that we have learned in section 8. Related work is presented in section 9 and conclusions are presented in section 10.

2 The Spade Methodology

The SPADE methodology makes a clear distinction between *applications* and *architectures*. An application defines a *workload* that is to be executed by the *resources* of an architecture. SPADE offers a simple Application Programmers Interface (API) to model applications as Kahn Process Networks [1], possibly starting from available C-programs. The main purpose of the application modeling step is to expose the parallelism and communication in the application. The Kahn Process Networks can subsequently be executed stand-alone in order to determine the (data dependent) computation and communication workloads. Architectures can be modeled at an abstract level using generic building blocks from a library. The building blocks may represent 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 application models can subsequently be *mapped* onto the architecture models in order to evaluate the *performance* of the different application–architecture combinations. The SPADE flow is depicted in Figure 1.

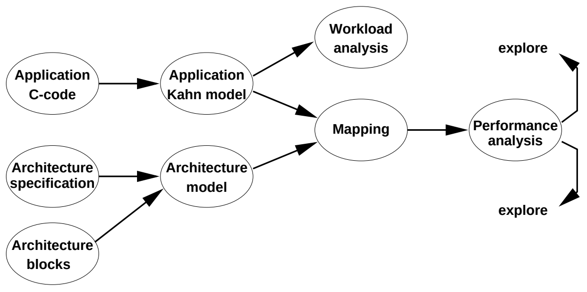

Figure 1: The SPADE flow for executing the case study.

3 The MPEG-2 Video Decoder Application

The SPADE methodology aims to facilitate the use of existing application code. In particular, it permits *application modeling* to start from a functionally correct sequential C-program of the application. The modeling of the MPEG-2 video decoder application started from a C-program that had originally been derived from the MPEG decoder software from UC Berkeley. This C-code was to be turned into a set of parallel communicating processes according to the Kahn Process Networks model [1]. In Kahn Process Networks, 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.

SPADE offers a simple API that can be used to turn a sequential C-program into a Kahn Process Network. The API contains the functions *read*, *write*, and *execute*. With the read and write functions, data can be read from or written to channels via process ports. The read and write calls correspond to the *communication workload* of a process. The execute function can be used to instrument the application code with *symbolic instructions* that identify the *computation workload*. This function itself performs no data processing. Figure 2 shows an example of an application modeled as a Kahn Process Network.

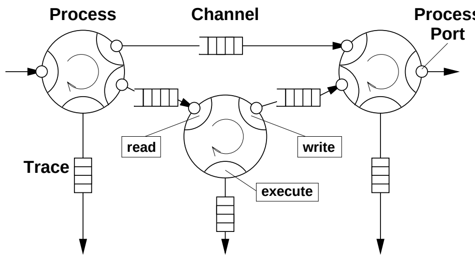

Figure 2: Application modeled as Kahn Process Network. Processes are depicted as circles; small circles represent process ports; the circle segments in the processes represent the use of API functions.

The following code fragment illustrates the use of the API.

```
while(1) {
  Input->read(token);
  ProcessToken(&token);
  Process->execute(PROCESSTOKEN);
  Output->write(token, size);
}
```

The code fragment shows an infinite process that repeatedly reads a token from its input port, processes the token, and sends the result to its output port. Each time a token is processed, this is signaled via the execute call.

The API has been implemented on top of a multi-threading package. Upon execution of the application model, each process runs as a separate thread. Processes synchronize via the read and write operations on the FIFO channels. These operations have been implemented with the help of semaphores, which synchronize the underlying threads.

The modeling of the MPEG-2 video decoder application started with the specification of the processes that may run in parallel as well as the specification of the types of the tokens that are communicated by these processes. Thus, during this functional partitioning phase we decided on the grain sizes of the processes as well as on the grain sizes of the tokens that get communicated by the processes. For example, we decided to have a process Tvld that parses an MPEG bit-stream under control of a process Thdr. The Thdr process is aware of the high level bitstream organization and distributes the retrieved sequence and picture properties to other processes. The Tvld process parses picture data autonomously. It sends macroblock headers into a functional pipeline that retrieves the prediction data for the reconstruction of macroblocks. The coefficient data for the error blocks is sent into a second functional pipeline for inverse scan, inverse quantization, and IDCT. The grain size for this coefficient data is a macroblock. A memory manager process TmemMan was introduced to control the access to the frame memories. It takes care that a frame is used for prediction or display only after it has been reconstructed completely. Thus, we see that during the parallelization of the application, control processes may appear that explicitly synchronize the operation of other processes.

During the actual coding, the sequential C-code of the decoder was split up into processes and the communication among the processes was made explicit by instrumenting the C-code with read and write calls. The parallelization of the C-code required several global data structures to be removed. Next, execute calls were added to be able to monitor the computation workload. The Kahn Process Network is shown in Figure 3.

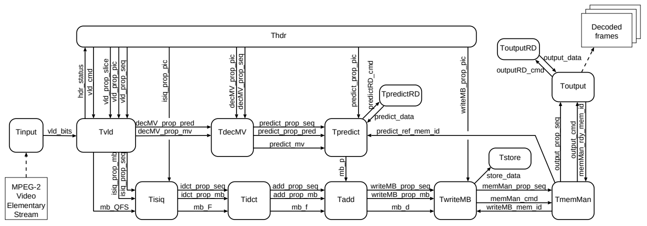

Figure 3: MPEG-2 video decoder modeled as Kahn Process Network.

The application model could now readily be used to analyze the workload of MPEG-2 video decoding for different MPEG streams. Upon execution of the model with a particular MPEG stream, the Kahn API reports:

- For each process: which symbolic instruction is invoked how often by the process.
- For each channel: how many tokens of which size(s) are transferred over the channel.

The results of such a *workload analysis* are presented in the form of two tables, as exemplified by the tables below:

Process	Instruction	Frequency
Tidct	IDCT_MB	12514
Tadd	Skipped_MB	158
Tadd	Intra_MB	2037
...	...	...

Channel	#Tokens	#Bytes
predict_data	88218	5645952
predict_mv	12514	400448
...	...	...

4 MPEG Decoder Architecture

In addition to the application model, we had to define an *architecture model* onto which the application model could be mapped. See the flow in Figure 1. The SPADE methodology is intended for top-down design of heterogeneous architectures and must permit efficient evaluation of a range of candidate architectures. For this case study, we decided to start from a single, but parameterized, architecture specification in order to validate that this architecture could be evaluated correctly, conveniently, and efficiently with SPADE. The parameterization would then allow us to do sensitivity analysis and some design space exploration for this architecture.

For the case study to be useful, we wanted to exercise a realistic architecture for MPEG-2 video decoding. For this we selected the TM-2000 MPEG decoder architecture from the Philips TriMedia Group, for which an internal databook level specification is available. The TM-2000 consists of a dedicated MPEG decoder attached to a bus structure together with a VLIW CPU and several other dedicated co-processors. The parts of the architecture specification that are relevant to MPEG decoding are depicted in Figure 4.

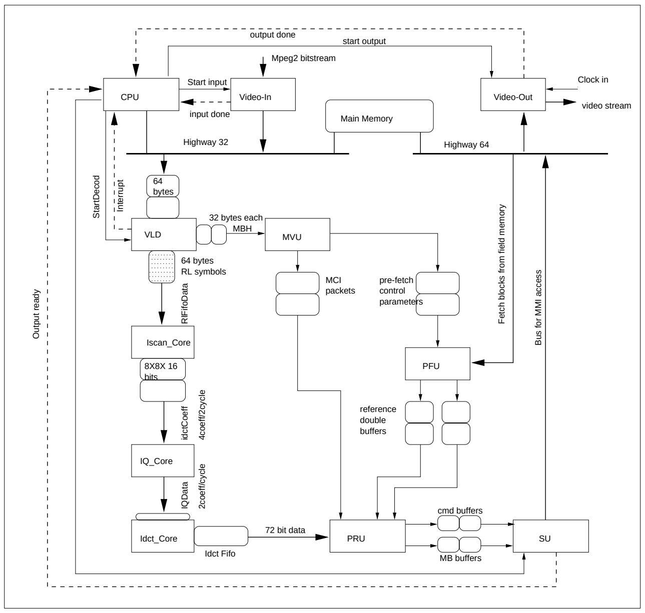

Figure 4: MPEG-2 video decoder architecture.

In this architecture the VLD unit parses an MPEG stream under control of the CPU. The MPEG stream may have been stored in Main Memory by the Video-In unit. The VLD alternately produces a Macro Block Header (MBH) for the MVU and Run Length symbols with coefficient data for the data driven Iscan / IQ / IDCT pipeline. The MVU drives the PFU to fetch prediction data from memory. The PRU combines the prediction data with the IDCT output and passes it to the SU in order to be stored in Main Memory. The Video-Out unit may subsequently retrieve the data from Main Memory. The CPU synchronizes the operation of the Video-In, VLD, SU and Video-Out units. A relevant question for this architecture is whether the latency of the Iscan / IQ / IDCT pipeline is properly balanced with the latency of the prefetching of the prediction data.

A methodology for architecture exploration must allow architecture models to be defined quickly and conveniently. SPADE aims to facilitate the construction of architecture models by offering a library of *generic architecture building blocks* from which architecture models can be composed. These building blocks are generic in that they are *non-functional* blocks that can be used to model a broad class of programmable or dedicated components. Upon execution, the blocks account for the consumption of *time* without processing any real data. This is further explained in section 6.

The generic building block approach is enabled by the *trace driven simulation technique* employed by SPADE. With this technique the application model is executed on top of the architecture model, to actually drive the architecture model with *data-dependent traces*. As a consequence, abstract non-functional architecture models, built from the generic building blocks, can be used for performance analysis of architectures running data dependent applications. The traces transfer information on communication and computation operations performed by the application processes. Specifically, a trace contains an entry for each read, write or execute call performed by an application process. The principle of trace driven simulation in SPADE is illustrated in Figure 5.

An architecture model is composed from the building blocks. This modeling involves the definition of a netlist and requires no additional programming. In SPADE, a processor, be it programmable or dedicated, is modeled by a single *trace driven execution unit*

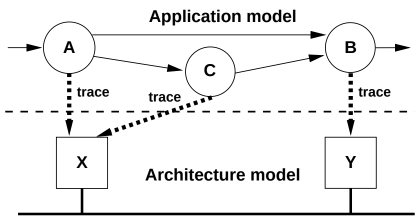

Figure 5: Trace driven simulation: the execution of the architecture model is driven by traces from the execution of the application model.

surrounded by *interface blocks* that are specific for the selected form(s) of communication (e.g. communication over a dedicated link or via a shared bus). Upon instantiation of the building blocks, parameter values can be assigned. For example, for the interface blocks the number of buffers and their sizes can be specified. A small example architecture model is depicted in Figure 6.

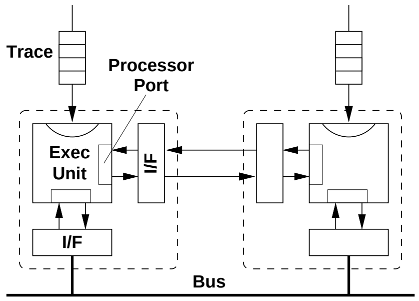

Figure 6: Small example architecture model. The model consists of two processors (the dashed boxes), that are each composed of a trace driven execution unit and two interface blocks, and a bus.

For each instantiated processor a repertoire of *symbolic instructions* and associated *latency values* must be specified. An application process can be mapped onto a processor only if the processor can execute all symbolic instructions that may be issued by the application process. A specified latency value represents the time that the processor needs to execute the fragment of the application that has been annotated with the corresponding symbolic instruction. These latency values may be retrieved from databooks, may be estimated by expert designers, or may be obtained via off-line compilation, simulation or synthesis activities. For example, one may run a code fragment through a compiler to determine the latency associated with the execution of that code fragment on a particular embedded processor core.

For the MPEG case study, we obtained the latency values for most of the symbolic instructions from the databook. For the other symbolic instructions we defined ranges of latency values that we wanted to explore. Remember that the objective of this work was to validate the SPADE methodology and not to validate the TM-2000 MPEG decoder design.

5 Mapping

The next step was to define a mapping from the application model (Figure 3) onto the architecture model. Each process from the application model is mapped onto a processor in the architecture model. Process ports from the application model are mapped onto the processor ports to the interface blocks. The mapping of the process ports determines the mapping of the channels onto the communication structures in the architecture.

For the MPEG case the mapping is rather straightforward. The process Tinput is mapped to Video-In and the process Toutput is mapped to Video-Out. Both Thdr and TmemMan are mapped to the CPU. The remaining processes are mapped one-to-one onto processors in the dedicated MPEG decoder.

SPADE does not require all channels of an application model to be mapped. In general, if a channel imposes only a small communication workload and does not relate to relevant synchronization delays at the architectural level, then the mapping of this channel may be omitted without a serious effect on the performance measurements. For the application model of Figure 3, we concluded for most of the channels that carry sequence or picture properties that the above condition was satisfied. Hence, these channels were not mapped. The remaining channels impose a significant communication workload or contribute to relevant synchronization among processors, and hence were mapped.

6 Performance Analysis

Performance analysis in SPADE is based on the combined *simulation* of an application model and an architecture model, using a trace driven simulation technique. The traces generated by the application model are interpreted by the trace driven execution units in the architecture model. When such an execution unit gets a *read* or a *write* entry from the trace, it sends the corresponding request to an interface and waits for the acknowledge, after which it reads a next trace entry. Between the request and the acknowledge a number of cycles may pass, depending on the interface and synchronization. For example, a dynamic communication delay may occur depending on the availability of a bus and the available room in buffers. When an execution unit gets a *symbolic instruction* from the trace, it looks up the corresponding latency and waits this number of cycles before reading a next trace entry. So, both communication and computation consume time in the simulation of an architecture model.

The generic building blocks have been equipped with measurement facilities to provide information on their performance behavior. They report on such *metrics* as processor utilization, stall times on processor ports, bus utilization, bus access delays, etc.

7 Design Space Exploration

SPADE has further been integrated with a DSE environment that supports design space exploration (DSE) by automatically performing multiple simulations at different points in the design space. From the acquired simulation data this environment builds piecewise linear models that show how selected performance metrics depend on parameter values, such as latencies of architecture components, when these parameters are varied in a range. We have used this DSE environment to perform a *sensitivity analysis* in a multi-dimensional parameter space for the unspecified latencies of architecture components. The outcome of this DSE were the *cycle budgets* for the respective processors based upon an evaluation of the overall system behavior.

A prerequisite for such design space explorations is simulation speed. The combined simulation of application model and architecture model for the MPEG case runs at a speed of a video frame per minute. This permits individual simulation runs to be done as needed, while larger explorations can be run overnight.

We also 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. Figures 7 and 8 show 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 are met; for a high bus load (small period) and for a high frame rate deadlines are missed.

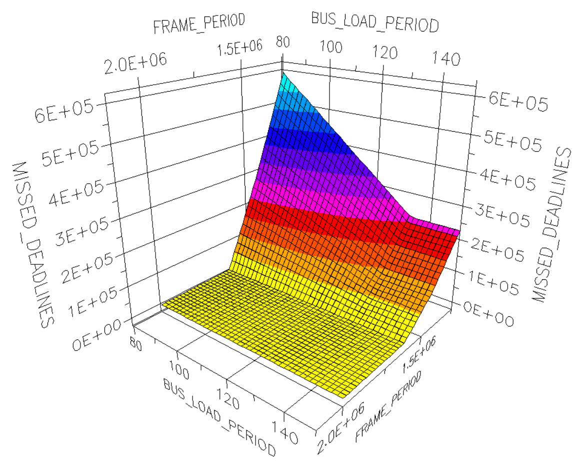

Figure 7: Measure for missed deadlines as function of frame period and period between bus requests of an additional processor.

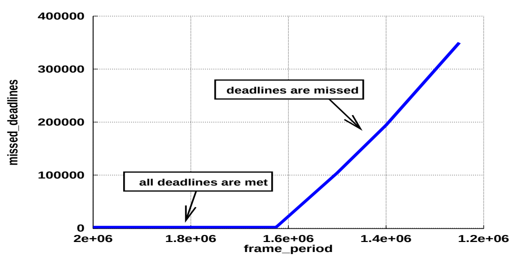

Figure 8: Measure for missed deadlines as function of frame period, for bus load period 115.

8 Lessons Learned

In this section we draw some conclusions with respect to the SPADE methodology, based on the case study that we have performed.

- The Kahn Process Networks model has proven to be an effective model for structuring the MPEG-2 video decoder application. We were able to start from existing C-code. The simple API is easy to use and supports workload analysis of the structured application. The process-based Kahn model supported by a multi-threading implementation offers a significant advantage over pure "fire-and-exit" dataflow models. A read call can simply be embedded in the C-code. Upon execution, the read call makes the thread block till data arrives on the process port and then returns the received data. The control flow of the C-code remains intact and no explicit action has to be taken to save the state of the process. This turned out to be particularly advantageous when coding the Tvld process of the MPEG decoder, which exhibits a lot of conditional behavior and would require a lot of state information to be administered if it were to be split into atomically firing functions.

- SPADE is an example of a Y-chart methodology; the design of heterogeneous architectures with SPADE follows a general scheme, named the Y-chart by Kienhuis et al. [2]. This scheme is also presented in [3]. See Figure 9.

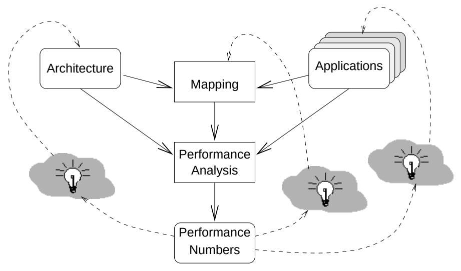

Figure 9: The Y-chart: a general scheme for the design of heterogeneous architectures.

The MPEG case study has shown that the decoupling of application modeling and architecture modeling, as instructed by the Y-chart scheme, is feasible, given that the application model is sufficiently fine-grain. This decoupling provides the flexibility for efficient evaluation of different application–architecture combinations. Further, we have demonstrated that a trace driven simulation technique can be used to execute an application model on top of an abstract architecture model in order to evaluate the performance of an application–architecture combination.

- The decoupling of application modeling and architecture modeling permits design projects to be organized more effectively. Application modeling can focus on deriving a Kahn model, which provides a clear deliverable for downstream activities. This (reusable) model provides statistics on the workload that the architecture must handle. In parallel, architecture modeling can start. Coordination among these parallel activities is required mainly to ensure that application models are sufficiently fine-grain with respect to architecture models.
- The trace driven simulation technique permits the use of abstract non-functional architecture models for performance analysis of architectures running data dependent applications. Such architecture models can be constructed efficiently from generic building blocks from a library. These blocks can be equipped with performance monitoring facilities to provide valuable feedback to the design engineer. As a result, architecture exploration becomes less laborious.
- Application and architecture models need to be *validated* in order to obtain reliable and accurate performance numbers. The application model was functionally validated by running MPEG compliancy tests using an available regression test suite of MPEG streams and decoder output. The architecture building blocks were validated by building small example architectures for which the execution could be monitored on a cycle by cycle basis. By relying on a library of validated building blocks, SPADE significantly simplifies the often time-consuming task of architecture model validation.

9 Related Work

Several system level architecture simulation frameworks have been developed. The Scenic framework [4] allows a designer to use C++ to model and simulate mixed hardware-software systems. TSS (Tool for System Simulation) [5] is the Philips in-house architecture modeling and simulation framework. In TSS an architecture is a network of interconnected *modules*, which are modeled using C. In [6] a performance simulation approach for MPEG audio/video decoder architectures is presented. This approach is based on building 'process-oriented' simulation models in C++. In contrast to SPADE, where architecture models can be composed from generic library blocks, these approaches require dedicated C/C++ models to be constructed for a specific architecture. In case there is data dependent behavior involved, these C/C++ models must be functional models in order to permit accurate performance evaluation. Building such models is laborious, and the reusability is low.

In [2] a Y-chart methodology for quantitative analysis of architectures is presented that uses abstract architecture models. However, architectures that can be analyzed with this environment are restricted to a specific class of dataflow architectures.

In [7] an architecture workbench for multicomputer systems is presented, named Mermaid. Mermaid also makes a clear distinction between the applications and architectures, in order to be able to do performance evaluation of a wide range of architectural design options. Mermaid also employs a trace driven simulation technique. In contrast to SPADE, architecture modeling in Mermaid is not based on a library concept.

10 Conclusion

The MPEG-2 decoder case study demonstrates that the SPADE methodology has unique characteristics that enable the efficient exploration of heterogeneous architectures. These characteristics are summarized in section 8. Further work will focus on a design trajectory that supports gradual refinement of architectures starting from abstract models. This will include support for multi-level simulation, to permit identified bottlenecks to be simulated at a more detailed level while other parts of the architecture are still simulated efficiently at an abstract level.

References

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

[2] 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.

[3] 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.

[4] 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.

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

[6] Dale Hocevar, Sundararajan Sriram, and Ching-Yu Hung, "A performance simulation approach for MPEG audio/video decoder architectures," in *Proc. IEEE ISCAS'98*, June 1998.

[7] A.D. Pimentel and L.O. Hertzberger, "An architecture workbench for multicomputers," in *Proceedings of the 11th International Parallel Processing Symposium*, Geneva, Apr. 1997.