In *1st Workshop on Rapid Simulation and Performance Evaluation: Methods and Tools (RAPIDO)*,
in conjunction with the 4th HiPEAC Conference, Paphos, Cyprus, January 25, 2009

# System Level Performance Simulation for Heterogeneous Multi-Processor Architectures

Martin Streubühr, Christian Haubelt, and Jürgen Teich

Hardware/Software Co-Design, Department of Computer Science
University of Erlangen-Nuremberg, Germany
{streubuehr, haubelt, teich}@cs.fau.de

**Abstract.** Performance modeling for real-time multi-processor architectures is a challenging task in the design phase of embedded hardware/software systems. As SystemC is well suited for designing a functional model of hardware/software systems, it is desirable to use SystemC with its simulation capabilities to evaluate the performance of an architecture for the designed system as well. Some approaches on the basis of SystemC are known to perform this task. However, many of these approaches are not applicable at the system level where complex applications are mapped to heterogeneous multi-processor architectures. In this paper, we propose a modeling framework that permits the task-accurate performance evaluation yet with a moderate simulation overhead. We use a Motion-JPEG decoder application to illustrate and assess our new approach.

## 1 Introduction

Performance evaluation for embedded hardware/software systems has been studied heavily in the past. In this context, the most challenging task is the modeling and analysis of the real-time scheduling strategies on a multi-processor architecture. Some formal approaches to analyze such systems exist. As formal approaches are often limited to corner case analysis and might face problems in the presence of huge systems, simulation seems to be an option for evaluating the performance for a given set of stimuli. Here, the problem at the system level is the modeling of effects of real-time operating systems on multiple, distributed processors.

As SystemC [1], a module-oriented C++-based design language, is well suited for designing hardware/software systems, it is desirable to use SystemC with its simulation capabilities to evaluate the performance of a designed system as well. Some approaches on the basis of SystemC are known to perform this task. Unfortunately, many of these approaches are not applicable at the system level where complex applications are mapped to real-time multi-processor architectures.

In this paper, we propose the modeling framework *Virtual Processing Components* (*VPC*) [2]. The VPC framework takes an actor-oriented functional model of the system written in SystemC as input and permits for a given architecture mapping the task-accurate performance evaluation with only reasonable simulation overhead in SystemC. The architecture mapping of a functional system model onto a multi-processor HW/SW architecture is specified in XML. It allows the modeling and simulation of shared resources, like memories, busses, and multiple processing units running different scheduling strategies. A processing unit may be a micro processor, an embedded controller, a DSP, non-programmable or programmable application-specific logic.

The rest of the paper is organized as follows: In Section 2, we present two related approaches and their advantages and limitations. After introducing some key concepts of functional modeling of a system in Section 3, we introduce the VPC framework in Section 4. We use a Motion-JPEG decoder application as the running example throughout this paper. In Section 5, we present results obtained from applying our performance simulation to a Motion-JPEG decoder.

## 2 Related Work

A simulation approach which permits the modeling of real-time scheduling strategies on the basis of SystemC is described in [3, 4]. For this purpose, an extension of SystemC to model executable software tasks is proposed. The end of each atomic operation, e.g., each SystemC code line, is augmented by an `await()` function call. The possibility to perform preemption is given between atomic operations. A set of software tasks is assigned to each processor. This technique provides the ability to simulate a set of processors running an operating system in parallel to hardware modules. Unfortunately, using this approach requires a substantial modification of the source code.

Another approach described in [5] proposes a so-called *Virtual Processing Unit* (VPU) running several tasks using a priority-based scheduler. For this purpose, software processes are modeled as *timed Communication Extended Finite State Machines* (tCEFSM), where each transition represents an atomic operation. Moreover, each transition consumes the same fixed amount of processor cycles. In this approach, several tasks modeled as tCEFSM can be executed sharing the same VPU. The task execution times depend on the simulated VPU clock speed. The main limitation of this approach lies in the modeling of time, where each transition of a tCEFSM requires the same number of processor cycles, and the support of only a single scheduling strategy.

In this paper, we will overcome the limitations imposed on the two discussed approaches by: (i) task-accurate performance modeling with less modifications of the source code, (ii) support of several preemptive and non-preemptive scheduling strategies, and (iii) component dependent and task-accurate execution times. Hence, our approach supports the same level of abstraction as proposed in [5] while providing efficient mechanisms for performance simulation presented as in [3, 4] at higher levels.

## 3 Functional Modeling Requirements

We assume that the application to be designed is given as a SystemC actor-oriented functional model. In an actor-oriented design [6], the behavior is organized in so-called *actors* which are restricted to communicate with other actors only by explicitly modeled channels. Associated with each actor, there is a so-called activation pattern which is a Boolean formula on predicates over the availability of data in the input channels of the actor [6]. Our methodology presented in this paper can be used together with any actor-oriented SystemC description. Although we do not use the SystemC Transaction Level Modeling (TLM) standard in our examples, it is very easy to combine SystemC TLM with our methodology presented in this paper. Due to space limitations, this aspect is omitted here.

An actor-oriented model of a Motion-JPEG decoder is given in Figure 1. We will use this decoder as our running example throughout this paper. In order to explore de-

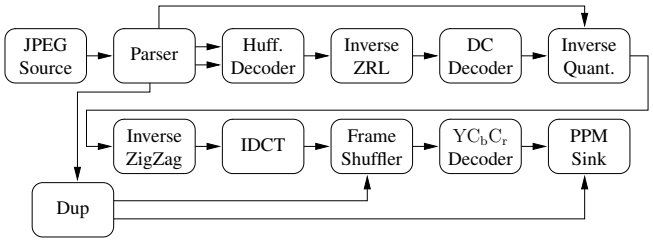


**Fig. 1.** Actor-oriented model of a Motion-JPEG decoder: Each block corresponds to an actor performing a particular operation. Communication is illustrated by directed edges.

sign options and analyze the performance of different designs on heterogeneous multi-processor architectures, we specified the JPEG algorithm in SystemC. Each node in Figure 1 represents an actor implemented as a SystemC module. Every module includes exactly one SystemC thread. Our decoder implementation consists of approximately 8000 lines of code, and has been restricted to color-interleaved base line profile without sub sampling. Using this functional model of the Motion-JPEG decoder, we are able to model execution latencies, resource sharing, and preemptive execution using our SystemC-based simulation framework. We introduce the framework in the following section.

## 4 Virtual Processing Components

In this section, we present our SystemC-based simulation framework called *Virtual Processing Components* (*VPC*). The VPC framework permits the task-accurate performance simulation of applications mapped onto a real-time multi-processor architecture [2].

The key idea of the VPC framework is the modeling of shared hardware resources like processors, busses, and memories as *Virtual Processing Components*. Each *Component* is configured with a certain scheduling policy such as *First Come First Served*, *Round Robin*, *Priority Based*, etc. The actual coupling of the architecture is done via the modeled (actor-oriented) application only. Hence, the VPC framework relies on three concepts: (i) An actor-oriented application modeled in SystemC, as discussed in Section 3. (ii) The modeling of hardware resources as *Components*. (iii) The ability to model preemption. Preemption is done by choosing a scheduling strategy associated with each *Component*. These scheduling strategies simulate preemptive or non-preemptive strategies to solve resource conflicts at a task-accurate granularity.

For each SystemC functional model to be simulated using the VPC framework, the following steps are mandatory: (i) The source code of the SystemC functional model must contain a so called `compute` function that provides the proper interaction and interface to the scheduler so to accurately incorporate the effects of the application on the multi-processor architecture, see e.g. in Figure 2. (ii) The VPC library has to be linked to the modified SystemC functional model. (iii) The mapping of actors to components of the architecture must be specified in a XML configuration file. (iv) The user has to provide stimuli for simulation issues. Note, that a modification in the architecture mapping does not require a recompilation of the SystemC model as this is done in the XML configuration file only. Hence, by varying the architecture mapping parameters, the VPC user is able to accomplish different performance simulation runs.

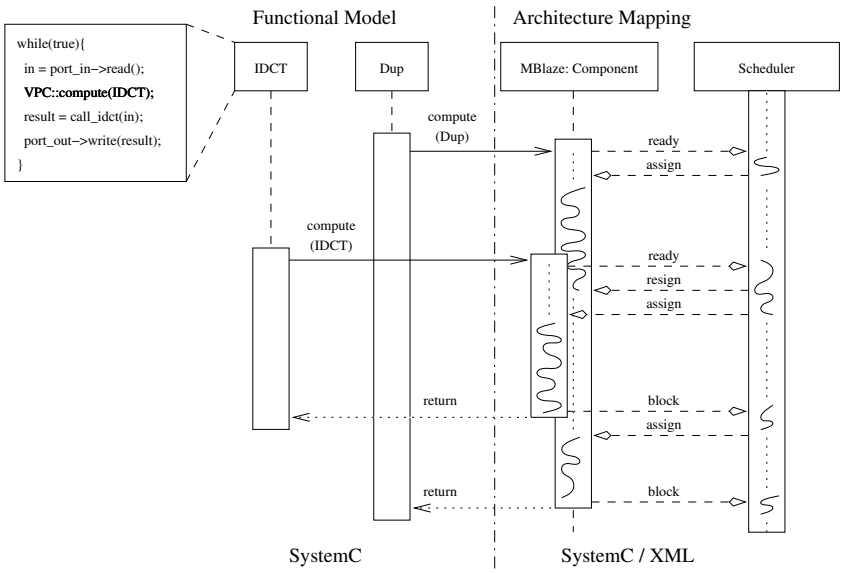


**Fig. 2.** Preemption mechanism of VPC: Two actors *IDCT* and *Dup* compete for the same resource *MBlaze*. The Virtual Processing Component assigns the resource according to a selected scheduling policy.

A *Component* is used as an abstraction from a real hardware resource, i.e., each resource including processors, hardware accelerators as well as communication resources are each represented by a SystemC module. Because of the system level view, our *Components* do not simulate a real hardware like an instruction set simulator or similar, but the core execution time of actors is simulated by our *Components*. The core execution time of an actor on a given hardware resource is assumed to be given, e.g. by profiling or from library data.

Running several actors on the same *Component* leads to interwoven actor activations. Such multi-tasking may thus introduce additional delays. To simulate the execution time using a *Component*, the `compute` member function is introduced. An actor in the application model is mapped onto a *Component* by invoking the `compute` method of this particular *Component*. For this purpose, the `compute` call must be inserted in the source code of the actor right after the activation function in front of the actual computation. As the simulation of the actor's functionality is simulated in zero time (simulated time), the user defined execution time delay[1] is elapsed completely during each `compute` call. Thus, the granularity is task-accurate. To allow the Component identification of the calling actor, a unique identifier is passed as parameter. An example is shown in the source code for the *IDCT* actor in Figure 2. The actor tries to read data from the input port. If the data are available, the actor is activated and can be executed. The `compute` method simulates the execution time delay including any waiting and preemption times. After the `compute` method returns, the functional code is simulated in zero time.

[1] Apart from profiling, the execution time delays might be estimated also using synthesis tools and/or WCET analysis tools.

**Table 1.** Latency and throughput values obtained from simulation compared to measurements from reference implementation. Decoding of four QCIF images has been used as test case.

| Architecture | Performance Simulation | | Reference Measurement | |
|---|---|---|---|---|
| | Latency | Throughput | Latency | Throughput |
| hardware only | 12.61 ms | 81.1 fps | 15.63 ms | 65.0 fps |
| HW/SW implementation 1 | 25.06 ms | 40.3 fps | 23.49 ms | 43.0 fps |
| HW/SW implementation 2 | 4 465 ms | 0.22 fps | 6 275 ms | 0.16 fps |
| software only | 8 076 ms | 0.13 fps | 10 030 ms | 0.10 fps |

The execution time is simulated using the SystemC function `wait`. In case of a resource conflict (several actors are calling `compute` on the same *Component*), a scheduling decision has to be done to resolve the conflict. In order to have more comfort using preemption, a *Scheduler* works together with the *Component*. Both implement an event-based communication mechanism to simulate preemption. The example in Figure 2 shows this mechanism: Once the actor *Dup* is activated, it calls `compute` on resource *MBlaze*. The *Scheduler* assigns the actor *Dup* to the *Component*. To simulate the execution of *Dup*, the *MBlaze* suspends for the execution time of *Dup*. When this execution time expires, the actor execution is completed. Nevertheless, in our example, the actor *IDCT* becomes active and calls `compute` as well. The *Scheduler* resigns the actor *Dup* and assigns the actor *IDCT*. When the entire execution time of the actor *IDCT* expires, the actor is removed from the *Scheduler*. Again, the actor *Dup* is assigned until execution time expires. This simple example already illustrates the ability to design complex preemptive scheduling policies at a task-accurate level within the VPC framework. Note that, as only the timing effects of preemption are simulated, no real task switch occurs and thus no storing and restoring of task states is necessary.

## 5 Results

This section presents results obtained by applying our system level performance simulation to a Motion-JPEG decoder benchmark. The simulation results are compared to four reference implementations of the decoder on a Xilinx Virtex II FPGA (XC2V6000). A hardware, a software, and two hardware/software implementations of the decoder have been used in this case study. For the software parts of the decoder, a MicroBlaze soft-core processor (*MBlaze*) was used.

For setting up the performance model of the Motion-JPEG decoder, the execution times for the actors implemented in hardware or software, and the communication delays for hardware-only, software-only and hardware/software communication are needed. These atomic execution times are obtained from profiling a software-only and a hardware-only implementation. We derived two mixed hardware/software implementations and compared the simulation results with measurements from the according implementations. The simulated and measured performance values are given in Table 1. Here, a Motion-JPEG stream consisting of four QCIF images ($176 \times 144$ pixels) has been used to obtain latency and throughput values. Using this stream, the simulation time for each architecture is about 30 seconds (wall clock time).

For evaluation of the simulation performance, a set of $7,600$ different architectures has been simulated. Each architecture consisted of a proper selection of resources from

a given superset template architecture. We recorded simulation times between 29.19 s and 32.71 s for each simulation on a Linux workstation with 3 GB of RAM and a 2.4 GHz Intel®Core™2 CPU. Average simulation time and standard deviation correspond to 30.44 s and 0.45 s, respectively. Due to configuration by an XML file, different architectures are evaluated without the burden of time consuming recompilation.

The minor discrepancy between the performance simulation and real measured values for the hardware-only solution could be tracked back to the time spent in checking complex activation patterns, occurring for instance in the Huffman Decoder. The underlying reason is that the actor-oriented simulation deploys an event-based simulation, where the evaluation of such checks is performed in zero-time by the simulation kernel, which is not true for the final hardware implementation. The discrepancy between the performance simulation and measured values for hardware-software solutions is due to the schedule overhead in the micro-processor resource. The latter one occurs in our implementation, because all actors bound to a single MicroBlaze are checked in a round-robin fashion whether one of their transitions can be taken or not. (As the time for evaluation of the corresponding activation patterns is not negligible.) A schedule overhead results whenever an actor is polled, i.e., checked for execution, while no execution is possible. This leads to a time consumption which cannot be taken into account by the performance simulation, as the latter one uses an event-based scheduling strategy.

## 6 Summary

In this paper, we proposed a modeling framework called *Virtual Processing Components* that allows for the task-accurate performance evaluation with a reasonable simulation overhead in SystemC. The VPC framework permits the modeling and simulation of multiple distributed processors running arbitrary scheduling strategies. The granularity is given by the task-accuracy. This guarantees a small simulation overhead.

## References


1. IEEE: IEEE Standard SystemC Language Reference Manual (IEEE Std 1666-2005). (2006)
2. Streubühr, M., Falk, J., Haubelt, C., Teich, J., Dorsch, R., Schlipf, T.: Task-Accurate Performance Modeling in SystemC for Real-Time Multi-Processor Architectures. In: Proceedings of Design, Automation and Test in Europe, Munich, Germany, IEEE Computer Society (March 2006) 480–481
3. Hastono, P., Klaus, S., Huss, S.A.: An Integrated SystemC Framework for Real-Time Scheduling Assessments on System Level. In: Proceedings of The 25th IEEE International Real-Time Systems Symposium (RTSS), Lissabon Portugal (December 2004)
4. Hastono, P., Klaus, S., Huss, S.A.: Real-Time Operating System Services for Realistic SystemC Simulation Models of Embedded Systems. In: Proceedings of The International Forum on Specification & Design Languages (FDL'04), Lille, Frankreich (September 2004) 380–391
5. Kempf, T., Dörper, M., Leupers, R., Ascheid, G., Meyr, H., Kogel, T., Vanthournout, B.: A Modular Simulation Framework for Spatial and Temporal Task Mapping onto Multi-Processor SoC Platforms. In: Design Automation & Test in Europe, Munich, Germany (March 2005) 876–881
6. Haubelt, C., Falk, J., Keinert, J., Schlichter, T., Streubühr, M., Deyhle, A., Hadert, A., Teich, J.: A SystemC-based Design Methodology for Digital Signal Processing Systems. EURASIP Journal on Embedded Systems, Special Issue on Embedded Digital Signal Processing Systems **2007** (2007) Article ID 47580, 22 pages doi:10.1155/2007/47580.