

# Task-Accurate Performance Modeling in SystemC for Real-Time Multi-Processor Architectures

M. Streubühr, J. Falk, Ch. Haubelt, J. Teich  
University of Erlangen-Nuremberg, Germany

R. Dorsch, Th. Schlipf  
IBM Deutschland Entwicklung GmbH

## Abstract

We propose a novel framework, called Virtual Processing Components (VPC), that permits the modeling and simulation of multiple processors running arbitrary scheduling strategies in SystemC. The granularity is given by task accuracy that guarantees a small simulation overhead.

## 1. Introduction

Performance modeling for real-time multi-processor architectures is a challenging task when designing hardware/software systems. As SystemC<sup>1</sup> [1] is well suited for designing such systems, it is desirable to also use SystemC's simulation capabilities for performance evaluation.

A simulation approach that permits the ability to simulate a set of processors running an operating system in parallel to hardware modules is described in [2]. A set of software tasks is assigned to each processor and the end of each atomic operation is augmented by an `await` function call allowing for preemption. This results in a substantial modification of the source code. The approach described in [3] proposes a so-called *Virtual Processing Unit* (VPU) running several tasks using a priority-based scheduler. Software processes are modeled as *timed Communication Extended Finite State Machines* (tCEFSM). The main limitation lies in the modeling of time where each transition of a tCEFSM requires the same number of processor cycles.

In this paper, we propose a novel framework, called *Virtual Processing Components* (VPC), that supports the same abstraction level as in [3] while providing efficient mechanisms as presented in [2] at higher levels. The VPC framework permits the simulation of a functional SystemC model mapped onto multiple processors running arbitrary scheduling strategies with a reasonable simulation overhead.

## 2. Virtual Processing Components

The *Virtual Processing Components* (VPC) framework permits the task-accurate performance simulation of appli-

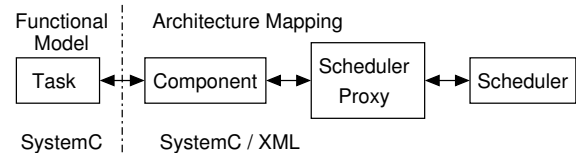


Figure 1. Interaction between a task and VPC.

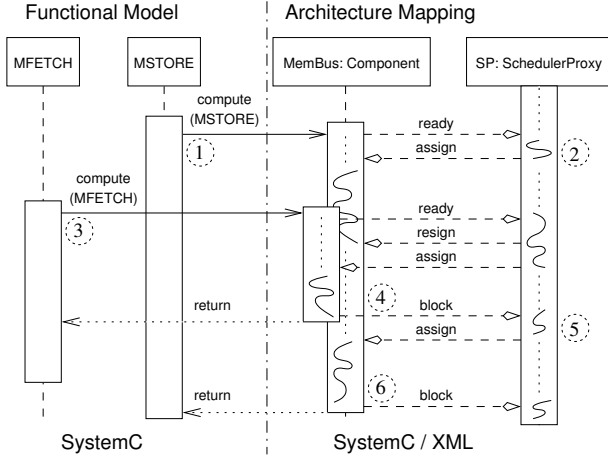
cations mapped onto a real-time multi-processor architecture in SystemC. Given a SystemC functional model, we can derive the underlying directed task graph  $g = (V, E)$  where vertices  $v \in V$  represent tasks and edges  $e \in E$  model the communication channels. In the following, it is assumed that a task  $v$  corresponds to a module in the SystemC design.

The key idea of the VPC framework is the modeling of shared hardware resources like processors, busses, and memories as Virtual Processing Components. Each Virtual Processing Component is configured with a certain scheduling strategy. The allocation of hardware resources to form the multi-processor architecture is defined by an XML configuration file. This configuration file also determines the scheduling strategies associated with the allocated resources as well as the binding of tasks to the resources. New scheduling strategies can be easily integrated in the VPC framework, by implementing the *Scheduler* interface.

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 be extended by `compute` function calls in order to couple the application with the architecture. (ii) The VPC library has to be linked to the functional model. (iii) The architecture mapping must be specified in an XML configuration file. (iv) The user has to provide stimuli for simulation issues. By varying the architecture mapping parameters, the designer is able to evaluate different designs. Note that a modification in the architecture mapping does not require a recompilation of the SystemC model.

Figure 1 shows the interaction of a task and a *Component*. The task calls the `compute` function which issues a schedule request to the *SchedulerProxy*. Finally, the *SchedulerProxy* requests the selected *Scheduler* for a schedule de-

<sup>1</sup>SystemC is a trademark of the Open SystemC Initiative.



**Figure 2. Preemption mechanism for two tasks competing for the same resource.**

cision which is returned to the *Component*.

A *Component* is an abstraction from a hardware resource. Running several tasks on the same *Component* leads to interwoven task activations. Such multi-tasking may lead to additional task delays. To simulate the execution time using a *Component*, the `compute` call must be inserted in the source code of the task  $v \in V$  right after the activation function in front of the actual computation. As the simulation of the task's functionality is done in zero time, the user defined execution time delay is elapsed completely during each `compute` call. Thus, the granularity is task-accurate. The following example illustrates the VPC approach:

```
while(true){
  int in = port_in->read();    // blocking read
  r = Director::getInstance()
    .getResource("Actor A"); // binding
  r.compute("Actor A");       // simulate time
  // functional code
  ...
  port_out->write(result);     // write
}
```

The task tries to read data from the input port. If the data are available the task can be executed. Before simulating the functional code, the task is bound via a *Director* (`getResource` function call) onto a *Component*  $r$  and calling `compute` on  $r$ . The `compute` method simulates the execution time delay including any waiting and preemption times by using the SystemC `wait` function. We simulate preemption by interrupting SystemC `wait` function calls. This can be done using SystemC *events*. Calling `wait` with a time and an *event*  $e$  will block until  $e$  is notified or the time has elapsed. A preempting task has to notify this *event*  $e$ . After the `compute` method returns, the functional code is simulated in zero time (simulated time).

To provide different schedulers, a *SchedulerProxy* is assigned to each *Component*. The *SchedulerProxy* has its own execution context through extending `sc_module`

**Table 1. Results for the InfiniBand HCA**

| # stimuli | config. | latency | throughput   | simulation time |
|-----------|---------|---------|--------------|-----------------|
| 1         | A       | 390ns   | 7.8/ $\mu$ s | 1.32s           |
| 1         | B       | 488ns   | 6.2/ $\mu$ s | 1.20s           |
| 10        | A       | 1100ns  | 3.5/ $\mu$ s | 1.55s           |
| 10        | B       | 2000ns  | 2.4/ $\mu$ s | 1.57s           |
| 100       | A       | 2000ns  | 2.3/ $\mu$ s | 5.80s           |
| 100       | B       | 8900ns  | 2.0/ $\mu$ s | 5.83s           |

and a *Scheduler* object is associated with it. The *SchedulerProxy* provides a unique scheduling interface including `addTask`, `removeTask`, `schedulingDecision`. An *Scheduler* implementation has to override these functions.

The communication between *Component* and *SchedulerProxy* is presented by the example shown in Figure 2 for two task *MFETCH* and *MSTORE*. This simple example already illustrates the ability to design complex preemptive scheduling policies at a task-accurate level.

### 3. Results

We present results from applying our VPC framework to a functional model of an InfiniBand Host Channel Adapter (HCA) [4]. The model is composed of 31 tasks (SystemC modules). The SystemC simulation model of the InfiniBand HCA comprises of approx. 12,500 lines of code. Table 1 presents performance results (average latency, average throughput rate, depending on the number of stimuli, and the simulation time (2.80 GHz, 512 MB RAM)) for two examples of multi-processor architecture mappings. The first architecture A uses four *Components* whereas the second architecture B only uses three *Components*. Note that it is very simple to change the architecture mapping by just editing the corresponding XML configuration file. In particular, it is not necessary to recompile the SystemC simulation model which is one of the key advantages of our approach. For testing issues, both HCAs are triggered with the same set of send and receive traces recorded from real HCA traffic. We see that a complex system like the InfiniBand HCA can be simulated in an amount of time which is acceptable for many applications yet providing resource- and contention-accurate multiprocessor modeling.

### References

- [1] T. Grötker, S. Liao, G. Martin, and S. Swan. *System Design with SystemC*. Kluwer Academic Publishers, 2002.
- [2] P. Hastrono, S. Klaus, and S. A. Huss. An Integrated SystemC Framework for Real-Time Scheduling Assessments on System Level. In *Proceedings of IEEE Int. Real-Time Systems Symposium*, 2004.
- [3] T. Kempf, M. Dörper, R. Leupers, G. Ascheid, H. Meyr, T. Kogel, and B. Vanthournout. A Modular Simulation Framework for Spatial and Temporal Task Mapping onto Multi-Processor SoC Platforms. In *Proceedings of Design Automation & Test in Europe (DATE)*, 2005.
- [4] T. Shanley. *InfiniBand Network Architecture*. PC System Architecture Series. Addison-Wesley, 2003.