# Shared resources high-level modeling in embedded systems using virtual nodes

Chafic Jaber, Andreas Kanstein, Ludovic Apvrille, Amer Baghdadi, Renaud Pacalet

# Shared Resources High-Level Modeling in Embedded Systems Using Virtual Nodes

Chafic Jaber[1], Andreas Kanstein[1], Ludovic Apvrille[2],
Amer Baghdadi[3], Renaud Pacalet [2]

[1]Freescale Semiconductor, 134 Av. du Général Eisenhower, BP 72329, 31023 Toulouse Cedex 1, France

[2] Institut Telecom; Telecom ParisTech; B.P. 193, 2229 rte des Crêtes,
06904 Sophia-Antipolis Cedex, France

[3] Institut Telecom; Telecom Bretagne; Technopôle Brest Iroise, CS83818, 29238 Brest, France

## Abstract

*The increasing complexity of system-on-chip design and shorter time to market constraints has stimulated systems designers to investigate performance characteristics of the final system implementation in the early design stages, by means of modeling the design at a high level of abstraction. This paper presents the virtual node concept for modeling the shared resources of a system-on-chip, therefore specifically dedicated to the study of the impact of shared resources contention on the overall system's performance, which is often defined by concurrent use cases. The overall approach is based on using a specific UML modeling profile and a SystemC-based simulator to execute models and analyze their performance.*

## 1. Introduction

System-on-chip design, that is, integrating multiple functionalities on a single die, has reached a very high level of complexity due to the available integration density. Taking an example from the domain of mobile devices, one can integrate multiple telephony modems and an operating system (OS) like Linux or Symbian, and have them share a single external memory [14]. In a true multitasking OS, the device must be able to handle simultaneous voice and data calls, while also handling complex imaging tasks like image or video capture, in part sharing processing engines, and all serviced by a single external memory. Both HW and SW architectures must be designed carefully with worst-case concurrent use cases in mind, to avoid any negative user experience.

Modeling the use case behavior and the candidate system architectures at an early stage is therefore essential for a successful system design. Fast architecture evaluation requires system modeling at a very high abstraction level. Furthermore, due to typical development cycles, the HW design needs to be closed earlier, sometimes much earlier than the SW design, providing another reason for high-level modeling.
As illustrated in the example, shared resources, like the integrated applications processor and the external memory, have a great influence on the performance of the system-on-chip. Our approach extends the DIPLODOCUS modeling methodology [8] with modeling and simulation techniques for shared resources [7]. This paper defines the concept of the virtual node to model shared resources and estimate their impact on system performance. A virtual node ensures – given a parameterized policy – the scheduling of accesses to shared resources (e.g. CPUs, busses, memories). The virtual node concept helps to build a well-structured simulation model, and also facilitates the creation of simple and reusable architecture component models.

The rest of the paper is organized as follows: Section 2 presents the virtual node concept and Section 3 presents our system modeling methodology and environment where we integrated the virtual node concept. Section 4 discusses related work on system modeling with a special focus on shared resources modeling, and finally Section 5 concludes this paper and gives future guidelines.

## 2. Modeling Accesses to Shared Resources

### 2.1. A Demonstrative Example

Let us consider a simple application example composed of three tasks: T1, T2 and T3; where T1 and T3 are two dependent tasks and exchange data between each other, while T2 is independent. This application will be executed by a hardware architecture composed

of two CPUs (CPU1 and CPU2), one bus (BUS1) and one memory (MEM1). The two CPUs share the bus BUS1 to access the shared memory MEM1. We consider, as well, that T1 and T2 will execute on CPU1, while T3 will execute on CPU2. In this scenario, when T1 and T3 exchange data, the data will be transmitted using the BUS1 and the MEM1. This communication involves two shared *architecture nodes*, the bus and the memory, whose access is determined with access policies: the arbitration policy. The communication latency is the sum of time needed to access shared resources and the contention on them when CPU1 and CPU2 want to access the bus at the same time for example).

Another level of contention is on the CPU scheduling level; a scheduling policy should indeed select a task to execute among the non-blocked ones. For instance, if the scheduling policy of CPU1 is priority based, and T1 is of higher priority than T2, then T1 will be chosen to execute. In this case, T2 execution is delayed until T1 finishes its execution, including the communication cost with T3 that may vary depending on the contention on the bus and the memory.

Finally, we can identify several types of shared resources. In fact each *architecture node* is shared between multiple elements: the CPU is shared between different tasks, the bus is shared between multiple CPUs and the memory is also shared between multiple CPUs using the bus.

This simple example shows clearly how the overall performance of the system depends significantly on shared resources access control and contention. It justifies the three types of resource requests that we have identified:

1- *Computation requests* generated by application tasks to *computation nodes* (e.g. CPUs).
2- *Communication requests* generated by computation nodes to the *communication nodes* (e.g. Bus) in order to transfer data generated/requested by tasks.
3- *Storage requests* generated by *computation* and *communication nodes* to *storage nodes* (e.g. memories)

As each resource can be shared between different requesters, resources should have an access policy that select a request among pending ones. We generalized this by introducing the virtual node concept, explained in the following section.

## 2.2. The Virtual Node

We define the "Virtual Node" (VN) as a generic modeling component that controls the access to a resource by implementing an access policy. It allocates the controlled resource to a requester, for example the VN of a CPU allocates the CPU to a task that is ready to execute, or the VN of a bus allocates the bus bandwidth to a CPU that is trying to reach the memory or other architecture nodes that are connected to the bus.

A request is generated by a requester to access to a resource. It specifies the resource amount that the requester needs. For example, a storage request shall specify the size of data to transfer. In addition, a request has a priority in the case when the VN's access policy is priority based.

Figure 1 shows the shared resources view of the example introduced in section 2.1. Each *architecture node* is controlled by a *virtual node*; for example "VN4CPU1" is the *virtual node* that controls the access to the CPU1. Requests are stored in queues (one **queue** for each virtual node). We have three types of queues: computation, communication and storage queues. The virtual node, using its access policy, chooses among the requests in the queue which one will acquire the access to the resource.

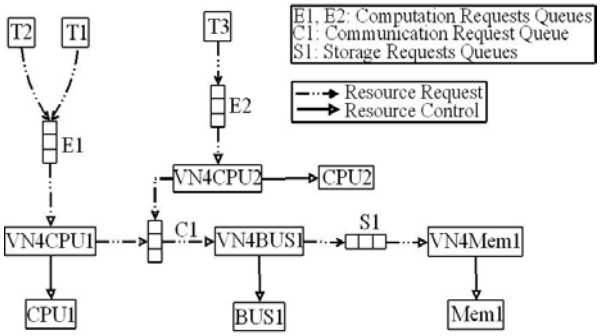

Figure 1: Shared resources modeling with virtual nodes

### 2.2.1. Virtual nodes hierarchy

Embedded systems can concurrently execute different real-time heterogeneous applications; for instance in a modern mobile device multimedia application such as video or audio could execute concurrently with control applications (telecommunication protocols). These applications may have specific scheduling requirements (soft real time, intensive data transfer or execution loop, etc); furthermore applying one access policy to all applications is not the optimal solution [11]. Hierarchical stacking of virtual nodes is our solution to optimize resources sharing when heterogeneous groups of requesters request the resource. A main virtual node controls a hardware resource and a secondary virtual

node controls each group of requesters. This approach allows us to optimize the access policies to satisfy requirements of all groups. We use, as well, the hierarchical composition of virtual nodes for computation, communication and storage resource sharing.

Figure 2 shows an example of two classes of applications; "App1" is controlled using a round robin policy while "App2" is controlled by a priority based policy. The CPU is shared between the two applications by a time sharing policy implemented by the main VN. The main VN (VN4CPU) allocates a time slot of the CPU to an application; the secondary VN (VN4App1 or VN4App2) controlling this application allocates the available execution time to one or more tasks depending on its access policy.

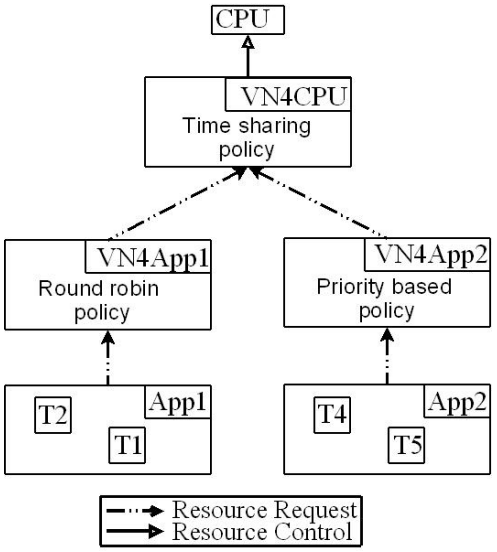

Figure 2: Virtual nodes hierarchy example

## 3. Modeling Methodology and Tool

We integrated the shared resources modeling, presented in the previous section, in the DIPLODOCUS UML modeling methodology [6, 7]. We furthermore defined a translation of DIPLODOCUS models into a simulator based on SystemC in order to execute the system model and estimate its performance. The following subsections give a brief overview of the DIPLODOCUS methodology and the simulation and analysis of its models.

### 3.1. DIPLODOCUS Methodology

DIPLODOCUS is a UML profile [8] targeting design space exploration at a high level of abstraction. It adopts the Y modeling paradigm [3, 4] which consists of modeling separately the application and the architecture, and then integrating both in a mapping model. Application and architecture models are totally independent from each other, and so a designer can easily evaluate candidate architectures using the same application model. It also permits to explore the mapping of two different applications on a given architecture during first stages of projects. DIPLODOCUS is supported by the TTool [1] toolkit that can automatically generate LOTOS or UPPAAL code for formal verification.

We extended the DIPLODOCUS modeling with the notion of virtual nodes to enable the analysis of performance issues related to shared resources, such as contention [7].

### 3.2. Mapping Model Simulation and Analysis

We developed a SystemC-based simulation environment where simulation code can be directly generated from UML DIPLODOCUS models.

Hardware architecture resources are instantiated from a library of pre-defined abstract models for architecture nodes that can be customized by setting the appropriate performance parameters (e.g. pipeline on a CPU, etc.), thus reducing the modeling effort. The designer can also use pre-defined access policies (with or without preemption): round-robin, fixed priority based, time slice scheduling, first-come-first-served. In addition, new access policies can be easily defined.

Simulations produce VCD waveforms containing temporal characteristics of the analyzed system, i.e. of the application, the architecture and of the VNs. In order to get a global view of the system, our simulator provides, for each resource, the utilization factor and the average contention delay on each resource thanks to add-on observers. Buffer overflow situations on storage nodes are also indicated. At last, application temporal behaviors are summarized in terms of end-to-end latency of the application, tasks' execution time, and the ratio of a task being ready or waiting to be scheduled by the VN of a computation node.

The designer can make design decisions based on simulation results: for example, he/she can evaluate the access policies of shared resources, tasks' memory mapping and the optimal capacity of resources (CPU frequency, memory size and hierarchy, bus speed …).

## 4. Related Work

The back annotation techniques like MESH [13] and the one proposed in Schnerr & al. [12] focus on the modeling of task scheduling and extract contention attributes related to communication and memories

from low level simulations. In our approach, we extract this information from the high-level simulation of our models. They utilize analytical and simulation techniques to estimate shared resources contention. Final code is used to estimate the performance. On the contrary, our methodology is applied early in the design flow, and so before the code is released.

On the other hand, early architecture exploration methodologies like Sesame [5] offer a clear distinction between application and architecture concerns, and facilitate flexible system-level performance evaluation. So far Sesame only provides schedulers to allocate computation resources to the application processes: it does not model communication architecture arbitration nor memory mapping.

Kempf *et al.* [2] present a simulation framework for MP-SoC platforms. They use a virtual processing unit (VPU) to schedule the execution of tasks mapped to a processor. The important difference to our approach is that we generalize the notion of a virtual node to model accesses policies to any type of architecture resources, and that we are able to extract performance result of any shared resource.

Hierarchical scheduling methodologies for processors [9] or bus [10] try to optimize resources sharing between multiple groups with different scheduling requirements. Our approach applies the hierarchical control to all shared resources, and more importantly at a high level of abstraction.

## 5. Conclusion and Future Work

This paper describes an important part of our methodology for the rapid investigation of the performance impact of contention on shared resources. We have extended DIPLODOCUS with a modular and extendible generic virtual node to model shared resources. Our virtual node model is hierarchical and can model the sharing of computation, communication and storage resources.

In addition to the aspects presented in this paper, we have also extended DIPLODOCUS with test bench modeling and simulation observers to support model analysis. To evaluate our methodology, industrial case studies are conducted over 4G telecommunication systems [7].

## 6. References


[1] LabSoC. TTool, The TURTLE Toolkit. See http://labsoc.comelec.enst.fr/turtle/ttool.html.

[2] T. Kempf et al. “A modular simulation framework for spatial and temporal task mapping onto multi-processor SoC plateforms”. In Design, Automation and Test in Europe (DATE2005), April 16-20, 2005.

[3] K. Keutzer et al., “System-Level Design: Orthogonalization of Concerns and Platform-Based Design”. In Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions Dec 2000

[4] B. Kienhuis, “An Approach for Quantitative Analysis of Application-Specific Dataflow Architectures”. In Application-Specific Systems, Architectures and Processors, july 1997.

[5] Andy D.Pimentel, Cagkan Erbas, and Simon Polstra. “A systematic approach to exploring embedded system architectures at multiple abstraction levels”. In IEEE Trans. Computers, vol.55, No.2, Feb. 2006.

[6] W. Muhammad et al. “Abstract application modeling for system design exploration”. In Euromicro Conference on Digital System Design (DSD’06), Aug. 2006.

[7] Chafic Jaber et al. “A High-Level System Modeling for Rapid HW/SW Architecture Exploration”. In the International Symposium on Rapid System Prototyping (RSP09), Paris, France, June 2009.

[8] L. Apvrille et al. “A UML-based environment for system design space exploration”. In 13th IEEE International Conference on Electronics, Circuits and Systems (ICECS’06), Nice, France, Dec. 2006.

[9] Giuseppe Lipari and Enrico Bini, “A methodology for designing hierarchical scheduling systems”. In Journal of Embedded Computing, 2005

[10] Trevor Meyerowitz et al. “A Tool for Describing and Evaluating Hierarchical RealTime Bus Scheduling Policies”. In Proc. of 40th Design Automation Conf. (DAC’2003), pp.312–317 (2003).

[11] Insik Shin et al. “Hierarchical Scheduling Framework for Virtual Clustering of Multiprocessors”. In Proceedings of the IEEE Real-Time Systems Symposium, pages 57–67. IEEE Computer Society, 2004.

[12] J. Schnerr et al., “High performance timing simulation of embedded software”. In Design Automation Conference DAC, June 2008

[13] Alex Bobrek et al. “Modeling shared resource contention using a hybrid simulation/analytical approach”. In Proc. Design, Automation and Test in Europe (DATE2004), pp 16-20, Feb 2004.

[14] Freescale Semiconductor. *Integrating Operating Systems with Freescale’s Cellular Software Platform.* http://www.freescale.com/files/wireless_comm/doc/white_paper/INTGFSLCELPLATWP.pdf