HAL open science

High-level system modeling for rapid HW/SW architecture exploration

Chafic Jaber, Andreas Kanstein, Ludovic Apvrille, Amer Baghdadi, Patricia Le Moënner, Renaud Pacalet

▶ **To cite this version:**

Chafic Jaber, Andreas Kanstein, Ludovic Apvrille, Amer Baghdadi, Patricia Le Moënner, et al.. High-level system modeling for rapid HW/SW architecture exploration. IEEE/IFIP International Symposium on Rapid System Prototyping (RSP '09), Jun 2009, Paris, France. pp.88-94, 10.1109/RSP.2009.27 . hal-02124754

HAL Id: hal-02124754

https://hal.science/hal-02124754v1

Submitted on 16 Jun 2022

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Distributed under a Creative Commons Attribution - NonCommercial 4.0 International License

High-Level System Modeling for Rapid HW/SW Architecture Exploration

Chafic Jaber[1], Andreas Kanstein[1], Ludovic Apvrille[2], Amer Baghdadi[3], Patricia Le Moënner[1], 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 – especially the software part of those systems – has stimulated much research work on design space exploration at the early stages of system development. In this paper we propose a new methodology for system modeling based on a specific UML profile. It defines a high design abstraction level for modeling and analyzing hardware resource sharing between system elements. Additionally, a SystemC-based simulator is developed in order to simulate modeled systems and evaluate their performance. Due to the high level of abstraction, the developed simulator enables fast exploration of design solutions.

First promising results are presented and discussed over a mobile platform for the 3GPP LTE protocol stack.

1. Introduction

Most recent Systems-on-Chip (SoC) are meant to host highly complex and interdependent applications at low cost (area, power) to the end user, i.e. the number of resources must be minimized while increasing their utilization by sharing them between multiple applications.

Such shared resources have a strong impact on SoC performance because of the contention they typically induce. Indeed, end-to-end performance of a given application is the sum of the time needed to execute that application with the total contention delay of all involved shared resources.

Therefore, designers have to investigate performance issues due to shared resources contentions. Unfortunately, these issues are sometimes discovered late in the design flow, inducing prohibitive re-engineering costs. A solution to this relies on deeper and earlier system analysis. Early performance analysis based on abstract models of the embedded system has already been demonstrated to increase design efficiency [7, 9, 10]. System-level design frameworks are commonly based on models meant to describe functions to be implemented by a set of candidate hardware architectures [2, 3]. In these approaches, analysis is commonly performed using simulation techniques where hardware is captured at transaction-level and software is represented by functional code or far more abstracted. Unfortunately, these environments frequently suffer four main drawbacks:

1- Relatively slow simulations because of too low-level (ISS or cycle-accurate) models.
2- Application and architecture concerns are mixed, leading to complex architecture exploration when performance requirements are not met.
3- No accurate capture is allowed regarding the impact of the contention on any type of shared resources (CPUs, busses, memories, etc.) when software and hardware are modeled at different abstraction levels.
4- Lastly, simulation traces do not always offer enough information to solve performance issues when requirements are not met.

Paper contribution: To cope with the previous drawbacks, we propose a high level modeling methodology to investigate the influence of shared resources on a system's performance metrics such as latency, throughput and resources utilization. The modeling is done very early in the design flow, when software and hardware architectures have not yet been released. Our methodology also enables the modeling of interactions of the system with its environment. At last, we developed a SystemC-based simulation environment to monitor and analyze designed models.

Paper Plan: Section 2 presents the DIPLODOCUS framework on which our contribution relies. Section 3 presents our system modeling methodology and simulation approach. Section 4 exemplifies our contribution with the LTE protocol stack case study. Section 5 presents related work on system modeling with a special focus on shared resources modeling, and finally section 6 concludes this paper and gives future guidelines.

2. Existing Modeling Framework

We based our research/work on the DIPLODOCUS framework [10, 11] because:

1- Application data is abstracted – by a concept of non-valued samples – leading to very fast simulations and allowing the use of formal techniques.
2- Modeling is based on UML, and so models are graphical and readable by most engineers. This also eases the handover between software, hardware and system engineers.
3- From UML DIPLODOCUS models, formal verification may be performed using automatic code generation to LOTOS or UPPAAL (not investigated in this paper).
4- DIPLODOCUS is supported by an open source toolkit named "TTool" [1] that integrates all the above points. TTool additionally generates documentation from developed models.

DIPLODOCUS adopts the Y modeling paradigm [7, 8] which consists of modeling separately the application and the architecture, and then integrating both during a mapping phase. Application and architecture models are reusable, because they are totally independent from each other, and so a designer can easily evaluate candidate architectures using the same application model. It also permits exploration of mapping of two different applications on a given architecture during first stages of projects.

While the existing DIPLODOCUS methodology clearly defines application and architecture modeling (see Sections 2.1 and 2.2), it lacks techniques to deeply investigate the use of shared architecture elements. Indeed, as stated in the introduction, contention due to shared resources might be really critical when designing SoC. DIPLODOCUS lacks as well the modeling of the interaction with the system environment. Section 3 introduces extension to the current DIPLODOCUS methodology in order to investigate issues related to shared resources.

2.1. Application Modeling

A DIPLODOCUS application [10, 11] is modeled as a network of communicating tasks that can be connected using three communication semantics: (1) *channels* to exchange abstract data samples (2) *events* to exchange signals, and (3) *requests* to ask for the execution of another task. *Channels* and *events* can be blocking or non-blocking, for the sender and for the receiver, while *requests* are always non-blocking for the sender and blocking for the receiver.

An application task behavior is modeled as a UML activity diagram. The latter is a sequence of commands containing control-flow commands (if, for loop, etc.) and communication commands with other tasks (e.g. send a sample, wait for an event, etc.). A data processing sequence is abstracted through metrics estimating the computation complexity of the instructions (e.g. equivalent to executing 1000 'integer add' instructions) which allows to estimate cycle counts after mapping on a computation node, defined next.

2.2. Architecture Modeling

Architecture is modeled as a network of physical resources abstracted by one of these *architecture nodes*: (1) *computation nodes* (CPUs, DSPs, hardware accelerators …), (2) *communication nodes* (busses, routers, switches …), (3) *storage nodes* (memories …). *Architecture nodes* are characterized by a set of performance parameters. For example a CPU could be characterized by its capacity represented in MCPS (Million of Cycles Per Second) while the capacity of a memory is represented by its size in bytes.

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, thus reducing the modeling effort.

The architecture model should define how nodes can communicate with each others. For example a computation node may access a storage node through a communication node. Hence we define an *architecture communication path*. It is the sequence of all the communication nodes that permits to a computation node to access a storage node. This path could be unique or run time selected depending on architecture defined parameters or on a routing policy if modeling is in a context of network on chip.

3. System Modeling Methodology and Simulation

We define the system model as the result of the mapping of the application model onto the architecture model. The mapping phase's goal is to evaluate the execution of an application on a given hardware architecture. The resulting system can be simulated (see section 3.5).

The system model we propose has the following execution semantic: according to its behavior, each task issues requests, one after the other, to shared resources. A shared resource can be either an architecture resource (CPU, bus, etc.) or a fraction of it (the fraction of a memory, of a crossbar, of a CPU etc.). Additionally, we define two types of resource requests:

- **Computation requests**: they are generated by tasks that wish to execute commands on the computation resource on which they are mapped.
- **Communication requests**: Requests to communication or storage resources, generated as a consequence of the execution of requests of the first type.

For example, when a task wants to perform a communication command, it first needs an access to a computation resource to execute the command (first type of requests). The execution of this command may also need an access to a bus (communication resource) and to a memory (storage resource) therefore generating requests of the second type. Both types of requests can be generated concurrently resulting in a contention on requested resources when two or more requests ask for the same resource.

Finally, to address this issue of concurrent access to shared resources, we define "Virtual Nodes".

3.1. Shared Resources Access Control Modeling

As application tasks run concurrently, they generate concurrent requests to the shared resources; as a consequence, an access policy (scheduling algorithm, bus arbitration policy etc.) for each resource shall allocate resource requests. To do so, we introduce a new modeling component called *virtual node* (VN). A VN is an intermediate concept between application structure elements (i.e. a task, a channel, etc.) and architecture elements (CPU, bus, etc.). A VN has a three-step execution semantic:

1. It waits for incoming requests.
2. It selects a request among the possible ones, and according to its allocation policy. This allocation may take into account the needed time to execute each pending request. That time may depend on performance parameters of underlying architecture resources.
3. It waits either for the selected request to finish its execution or for a new incoming request. In that latter case, the allocation is re-evaluated (step 1).

Our model furthermore supports the hierarchical composition of VNs, i.e. a VN may control the access of others VNs. This enables the modeling of more complex architecture behavior by use of generic VN model nodes.

3.2. Mapping Views

We define the mapping phase as the allocation of application tasks to computation nodes and to storage nodes, and in mapping channels/events/requests to storage and communication nodes. Computation nodes access storage nodes using architecture communication paths. The mapping has three views to cope with the fact that each type of resources should be allocated to the corresponding application component:

1. In the *computation mapping view*, application tasks are mapped to computation resources using Computation Virtual Nodes.
2. In the *storage mapping view*, tasks are mapped to memories – using Storage Virtual Nodes – in order to precise where the task code and data reside.
3. In the *communication mapping view*, channels, events and requests are mapped to communication resources using Communication Virtual Nodes. In addition, communications between computation and storage resources (or two computation resources) are resolved thanks to "Communication Managers". A *communication manager* is attached to a computation resource and it can identify a path for all communications performed by this computation resource.

3.3. System's Environment Modeling (Use Cases)

Our extended DIPLODOCUS methodology supports modeling of use cases. A use case is meant to specify an interaction between the modeled system and its environment. Thus, use case modeling corresponds to an abstract description of the environment. For example, the modeling of a telephone call - which is a use case of the telephone system - involves a

necessary description of the consumers and producers to identify the traffic they imply.

Inputs provided by the environment to the system abstract real data, and specify the time at which that data are produced (arrival rate). Outputs shall also be handled by use cases. In a telecommunication context, use cases could model traffic traces (when a packet is received - arrival rate - and what is its size for example). We use stochastic models to define these traffic traces.

Use cases are modeled like applications, i.e. with tasks and communication between those tasks; but use case tasks and communications are not mapped onto hardware architectures: they are just meant to run concurrently with the modeled system and stimulate it.

3.4. Modeling Example

Figure 1 depicts the mapping input on the left side. Together with assigning tasks and communications from the application diagram to architectural resources, the user has to provide mapping parameters such as task priorities and the type of communication used to implement Channel1 between task T2 and task T1. As an example, we want that T2 mapped on HW1 communicates to T1 via shared memory and an interrupt.

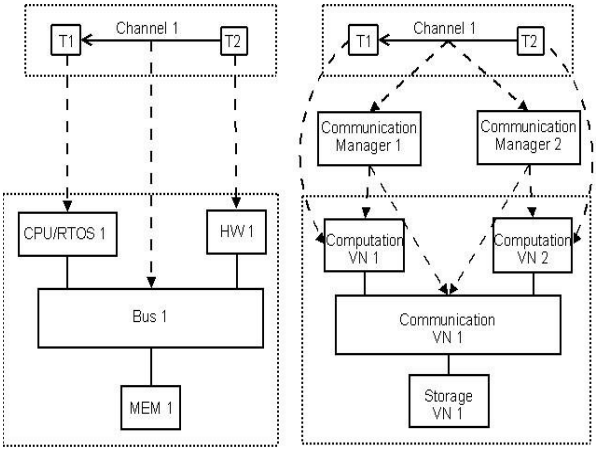

Figure 1: Mapping model example showing the mapping of application to architecture components (left) and the executable model using virtual nodes (right)

The architecture part of the mapping specifies important architectural parameters such as the type of RTOS and CPU, the bus type and its data bus width, etc. The right hand side of Figure 1 shows the mapping model implementation for the simulator. The three mapping views have been represented together. Virtual nodes are picked based on the information provided in the architecture diagram. Computation VN models replace the RTOS/CPU and the HW resources to implement their scheduling. A Communication VN and a Storage VN implement the bus and memory arbitration. Finally, two communication managers are inserted to implement the details of Channel 1 on each computation VN, i.e. how the Computation VN can reach the communication and storage virtual nodes. Instead of address-based communication on bus transaction level, we use virtual addresses and virtual data.

For a block of data sent on Channel 1, Communication Manager 2 places the data into the virtual storage element, via the bus and then sends a notification event to Communication Manager 1. This component then needs to model the interrupt service routine on the Computation VN 1, notify T1, and finally T1 reads the data once woken up and running.

3.5. Models Simulation

We have implemented a SystemC-based simulation environment to simulate modeled systems and to evaluate their performance. The environment takes DIPLODOCUS UML models (Application, Architecture and mapping) as inputs. It generates automatically from them corresponding SystemC code. The architecture of this environment is not presented in this paper due to limited space. Simulations can output VCD waveforms containing temporal characteristics of the analyzed system, i.e. of application, architecture, mapping and use case components. These results are also post-processed to show more global information of the system; for example the WCET (Worst Case Execution Time), the BCET (Best Case Execution Time) or the ACET (Average Case Execution Time) of each task, as well as the utilization factor of each architecture resource.

We intend to co-simulate our models with other network simulators [17, 18] meant to generate traffic used as input by our models. This solution will offer more precise stimuli than stochastic values, at the expense of longer – though still acceptable - simulation time.

4. Case Study – LTE Protocol Stack

In order to illustrate the efficiency of the proposed methodology, we describe in this section the SW/HW exploration of an implementation of the LTE (Long Term Evolution) protocol stack (above the physical layer). LTE is the latest standard in the mobile network technology developed by the 3GPP organization [14]. LTE is meant to substantially improve throughputs, and shall support IP-based traffic with end-to-end quality of service, but it shall

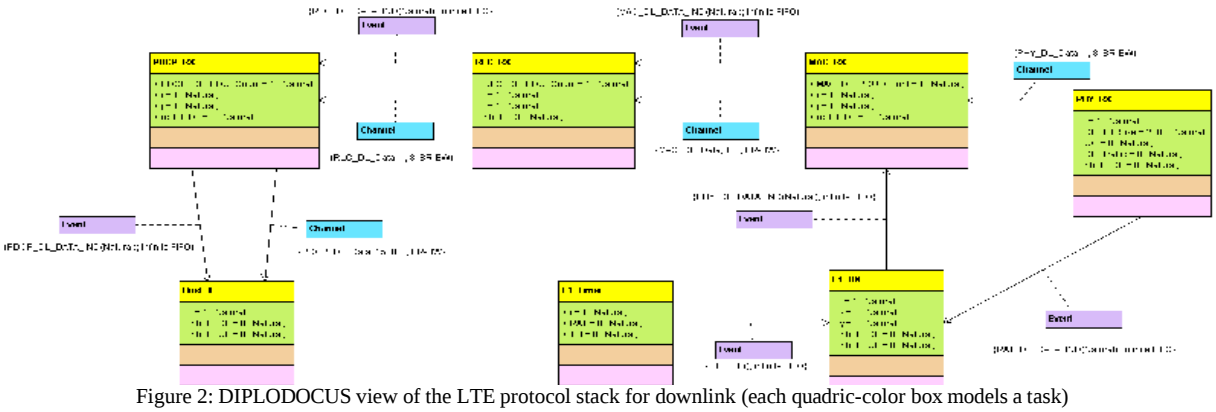

Figure 2: DIPLODOCUS view of the LTE protocol stack for downlink (each quadric-color box models a task)

also greatly increase modem complexity because of massive streams of data packets. This section illustrates how we could apply our methodology to model and analyze the LTE modem.

4.1. Example DIPLODOCUS Model

The example DIPLODOCUS application model of the LTE protocol stack is composed of 14 communicating tasks (Downlink path is shown in Figure 2).. As stated in the DIPLODOCUS methodology, this model is only application-oriented and therefore contains no architecture details.

Figure 3 depicts a possible candidate architecture to run the LTE protocol stack. It contains two DSPs (DSP1 and DSP2) as computation resources, one crossbar (Crossbar1) as communication resource, and four memories as storage resources: dedicated and shared internal and external memories. The modem architecture has been simplified for this investigation; see for example [12] for a discussion of an industrial mobile modem architecture.

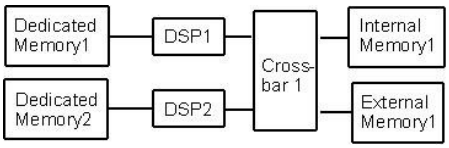

Figure 3: A possible candidate hardware architecture for the LTE protocol stack

As defined in Section 3.2 we created the computation, storage and communication views to implement the mapping. Each of the computation resources has a virtual node attached that defines the scheduling policy. Also, communication and storage virtual nodes are now inserted to implement the respective architecture communication path and node access scheduling. At last, the traffic generated by a video stream transfer using the TCP protocol is modeled in DIPLODOCUS and serves as the use case. Appropriate traffic models can be found in [13]; they specify the number and the size of exchanged packets.

The DIPLODOCUS methodology is based on a high level of abstraction, and the simulation environment provides system architects with extensible and simple to use architecture resources and virtual nodes. For those reasons the modeling effort is low and we modeled and analyzed the system presented in this section within one week, which is a very short time in comparison to the overall design time.

4.2. Simulation Results

In a first step, the LTE mapping explained above is simulated with our SystemC simulation environment, with the objective of identifying the effect of the scheduling policies of the two DSPs on the overall system performances. We will study two policies: priority based with preemption and without preemption.

The LTE protocol stack tasks are mapped on DSP1 and a video decoder is mapped on DSP2. The Crossbar1 virtual node and the storage virtual nodes for the internal and external shared memories use a first come first served arbitration policy.

The simulation speed of this model is 5 times faster than real-time on an example streaming a 3MB VDC file.

The analysis of the modeled system performance is firstly based on waveforms generated by the simulation environment. Figure 4 depicts a waveform describing the execution of some of the LTE protocol stack tasks showed in Figure 2.

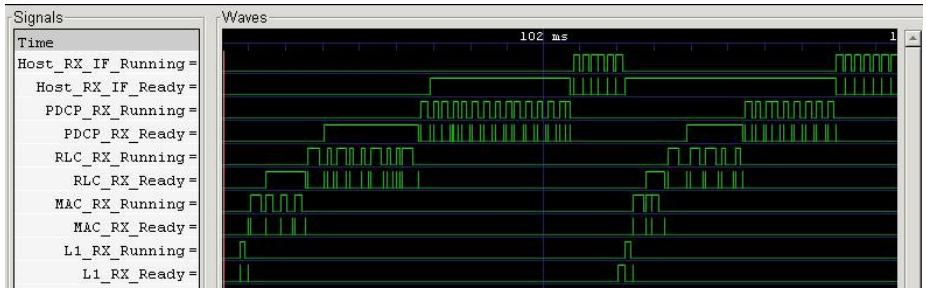

Figure 4: Excerpt of the result of execution of the LTE protocol stack

Automatic post processing capabilities provide more global performance parameters. Table 1 shows for each DSP (DSP1 and DSP2), and for each scheduling policy, the utilization (ratio of execution time without memory access over all the simulation time), the percentage of overhead (from utilization time) due to the virtual node execution (scheduling of tasks, context switches …), the percentage of utilization time needed to access memories (without contention) and the percentage of utilization time where the DSP is suspended due to memory access contention.

Table 1: Simulation results example

		Utilization (%)	Overhead (%)	Memory Access (%)	Contention (%)
DSP1	Preemptive	52	10	22	19
	Non preemptive	49	8	15	17
DSP2	Preemptive	61	27	17	11
	Non preemptive	60	25	14	10

Our simulation environment is capable of tracking a specific execution parameter to study it. Table 2 shows the WCET, the BCET and the ACET the stack needs for receiving a block from the network until it transfers it to the requesting application. This parameter is dependent not only on one task or one architecture resource, but it is also dependent on the overall system (the LTE application, the architecture and the mapping). Table 2 shows as well the impact of the selected scheduling policy.

Table 2: Time execution for one transport block

	WCET (ms)	BCET (ms)	ACET (ms)
Preemptive	1.2	0.4	0.8
Non preemptive	0.9	0.4	0.7

The LTE standard specifies that the maximum value for this parameter should be 1 ms [15], while our worst case execution time in the case of preemptive scheduling was 1.2 ms mainly due to high scheduling overhead and memory accesses in this example architecture.

5. Related Work

Many design methodologies and supporting tools – including DIPLODOCUS - propose a mapping phase once application and architecture models have been performed [15]. Those methodologies extract shared resources impact on system's performances. Like our approach, some methodologies are more particularly focused on early analysis and documentation of complex architectures, while functional modeling and synthesis of implementations is the intent of others (e.g., [16]). The following discussion compares our approach with other high-level methodologies that attempt to estimate impact of shared resources on system performances.

The back annotation techniques like MESH [5] and the one proposed in Schnerr & al. [4] extract performance latencies from a low-level simulator to annotate the higher level model. 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. Also, above-mentioned techniques 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.

Early architecture exploration methodologies like Sesame [9] offer a clear distinction between application and architecture concerns, and facilitate flexible system-level performance evaluation. Application is modeled as a set of Khan processes while architecture is defined at a high level of abstraction in a similar way to DIPLODOCUS. So far Sesame mapping models only provide schedulers to allocate computation resources to the application Khan processes: it does not model communication architecture arbitration nor memory mapping.

Kempf *et al.* [6] 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.

ArchAn [3] and Panama [2] enable modeling at a high level of abstraction and they capture task scheduling as well as the communication architecture and memory mapping modeling. Unfortunately they do not define a clear separation between the application and the architecture. Indeed, the language used to model tasks includes details of the underlying architecture thus reducing the reusability of models.

6. Conclusion and Future Work

Beginning with the idea to focus on high-level system models on a very abstract level, we have extended DIPLODOCUS with (1) a modular and extendible generic virtual node to model shared resources, and (2) a simulation framework in SystemC. Our virtual node model is hierarchical and can model the sharing of computation, communication and storage resources. Modeling shared resources is very important for highly integrated converged mobile devices. We presented first results from our mapping model for architecting the implementation of next generation devices, supporting the computationally very demanding LTE protocol stack. To support the methodology for more complex models, we are working on integrating the mapping model into TTool [1].

7. Acknowledgments

The authors of this paper would like to thank Christopher Yasko for his help on developing the case study, Mukesh Taneja for his work on the traffic models, and Francois Menneteau and Daniel Knorreck for their valuable help in developing the simulation environment.

8. References

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

[2] M. Silbermintz, et al. “SOC modeling methodology for architectural exploration and software development”. In Proc. ICECS’06, Dec 2004.

[3] A.Chatelain and Y. Mathys. “Verification strategy for integration 3G baseband SoC”. In Proc. 40th ACM IEEE Design Automation Conference, June 2003.

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

[5] 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.

[6] 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.

[7] 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

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

[9] 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.

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

[11] 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.

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

[13] IEEE traffic models. See http://www.wirelessman.org/tg3/contrib/

[14] 3GPP. Long Term Evolution of the 3GPP radio technology. See http://www.3gpp.org/Highlights/LTE/lte.htm

[15] F. Balarin et al., “Modeling and Designing Heterogeneous Systems,” Concurrency and Hardware Design, J. Cortadella, A. Yakovlev, and G. Rozenberg eds, Springer, 2002, pp. 228-273.

[16] K. Popovici, X. Guerin, F. Rousseau, P.S. Paolucci, A. Jerraya, “Efficient software development platforms for multimedia applications at different abstraction levels,” in Proc. 18th IEEE/IFIP International Workshop on Rapid System Prototyping, May 2007.

[17] NS2, The network simulator. See http://isi.edu/nsnam/ns/

[18] OMNeT. See : http://www.omnetpp.org/